ARQUITECTURA ASIMÉTRICA MULTI CORE CON PROCESADOR DE PETRI

Tesista: Orlando Micolini

Director: Ing. Armando De Giusti

Codirector: Dr. Marcelo Naiouf

Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas

Facultad de Informática
Universidad Nacional de La Plata
La Plata, República Argentina, Febrero de 2015

Agradecimientos

A mi amor	*Carmen*
A mi fortuna	*Carolina y Pablo*
Por sus valiosos aportes y la confianza depositada en mí y en este trabajo	Armando De Giusti Marcelo Naiouf
Por permitirme disentir	Carlos Barto
De quienes aprendo	Mis alumnos
Con quienes aprendo	Mis compañeros de trabajo
Por su contribución al progreso	La UNLP
Al equipo del LAC que se comprometió con las RdP en la enseñanza, realización de los experimentos, las discusiones de los resultados y por su espíritu de grupo	Carmen Centineo Alessi, Pailler Julio César, Tejeda Miguel Angel, Baldoni Federico E., Romano Matías H. Gallia Nicolás Alejandro, Pereyra Martin Miguel, Morales Pablo Gastón, Ingaramo Pablo Andreas, Julián Nonino, Carlos Renzo Pisetta, Arlettaz Emiliano, Birocco Baudino Sergio, Martina Agustín, Maschio Alfonso, Maria Florencia Caro, Ignacio Furey, Sufán Eduardo Zaqui, Daniele Emiliano Nahuel, Perret Cantoni Matías, Marcelo Cebollada, Luis O. Ventre, Gustavo Wolfmann y Maximiliano Eschoyez.

Contenido

Capítulo 1

Motivación del Paralelismo

El desarrollo del software y hardware paralelo trae como resultado mejoras del tiempo de ejecución, la disminución de consumo y corrección del sistema. Sin embargo esto requiere de un mayor esfuerzo, que lo podemos atribuir en gran medida a:

- la complejidad inherente de la especificación
- la coordinación de tareas concurrentes
- la falta de algoritmos portables
- los entornos estandarizados
- el software y herramientas de desarrollo

Cuando observamos el desarrollo de microprocesadores, nos sentimos tentados a cuestionar la explotación del paralelismo para acelerar las aplicaciones, por el esfuerzo que esto demanda. Puesto que, si tardamos dos años para desarrollar una aplicación paralela, en ese tiempo el hardware subyacente y/o la plataforma de software se ha convertido en obsoleto, el esfuerzo de desarrollo está claramente perdido. Sin embargo, hay algunas tendencias inequívocas en el diseño de hardware, que indican que un solo procesador no puede ser capaz de mantener el ritmo de incrementos de rendimiento. Este es el resultado de la falta de paralelismo implícito del software (Instruction-level parallelism ILP) y el cuello de botella que impone el hardware (bus, memoria, sincronización, retardo, etc.). Al mismo tiempo, las interfaces estandarizadas de hardware tienden a disminuir los tiempos de respuesta de desarrollo de un microprocesador y multi-Core, para máquina paralela basado en el microprocesador. Por otra parte, se ha avanzado considerablemente en la estandarización de los entornos de programación para asegurar un mayor ciclo de vida de las aplicaciones paralelas. Estos argumentos motivan al desarrollo de nuevas plataformas en favor de la computación en paralelo.

Motivación del Trabajo

Redes de Petri (RdP)

Los sistemas informáticos son complejos tanto en su estructura como en su comportamiento, más aun cuando tienen un gran número de estados y numerosas combinaciones de datos y eventos de entrada.

En particular en los sistemas multi-Core y distribuidos aparecen problemas tales como la exclusión mutua, inter bloqueos, propiedades de vivacidad del sistema, sincronización y la comunicación entre los procesos.

La investigación sobre métodos formales para abordar estas cuestiones es un campo fructífero y dinámico. Su desarrollo está motivado por la necesidad de diseñar productos informáticos fiables.

La complejidad natural que implica la concurrencia torna ineludible el empleo de herramientas formales para su estudio en la implementación de sistemas.

Una importante herramienta para el estudio y modelado de sistemas son las Redes de Petri (RdP); estas pueden modelar una amplia variedad de escenarios [1-3]. Permitiendo tanto el modelado del

sistema como su representación matemática, de donde se puede extraer importante información acerca de la estructura y comportamiento dinámico del sistema.

Esencialmente se caracterizan por su simplicidad, ya que las entidades matemáticas que intervienen son pocas y simples; el modelado de secuencias, decisiones, concurrencia y sincronización es simple. Las RdP son adecuadas para expresar todas las semánticas básicas de la concurrencia como: entrelazado, semántica de pasos, semántica de orden parcial, y localidad de estados, y acciones, posibilitando el modelado progresivo por refinamientos sucesivos o por composición modular.

Las RdP se centran en la semántica de orden parcial de los sistemas [4], en la relación causal entre los eventos y en la caracterización estructural de su comportamiento. Son una herramienta para afrontar las limitaciones de las máquinas de estados finitos a la hora de especificar paralelismo. Existen diversos escenarios:

- Máquinas dotadas de miles de procesadores capaces de realizar en pocos segundos cálculos complejos, como inversión de grandes matrices, procesamiento de imágenes, etc.
- Sistemas de tiempo real, diseñados para responder a ciertos de eventos externos en un plazo de tiempo muy limitado.

En estos escenarios nos encontramos con varios procesos concurrentes que cooperan entre sí para alcanzar sus objetivos, que es lograr la coherencia correcta del sistema, tanto desde el punto de vista funcional como temporal.

En la actualidad cuando es necesario proteger el acceso a recursos compartidos en accesos concurrentes, la aparición de los sistemas en tiempo real y de los sistemas multi-Core han motivado que la concurrencia sea un campo de intensa investigación.

La complicación de los sistemas concurrentes ha hecho necesario el desarrollo de modelos formales que permitan representar la corrección de los sistemas diseñados. Los modelos introducidos se basan en el concepto de abstracción[5], en el sentido de que todo modelo debe capturar únicamente aquellas características del sistema que se consideren importantes.

En el caso del modelado de sistemas concurrentes abstraemos gran parte de las acciones desarrolladas por los procesos, para centrarnos, especialmente en la cooperación entre estos procesos, que es el punto de máxima dificultad en el diseño de sistemas concurrentes.

Una vez construido el modelo, es necesario disponer de herramientas que nos permitan analizar determinadas propiedades de “buena conducta” del mismo.

En principio podemos clasificar los métodos de análisis con RdP en estructurales y dinámicos.

Los métodos estructurales, toman ventajas de la estructura de red y disminuyen la complejidad del análisis, y también toman ventajas por ser aplicables a redes independientemente de su marca inicial, las principales propiedades estructurales son:

- Controlable, si cualquier marcado es alcanzable desde cualquier otro marcado.
- Estructuralmente limitada, si está acotada para cada marcado inicia, también se la llama RdP n-limitada (si todas sus plazas son n-limitadas). Por lo que decimos que hay un límite en cuanto al marcado de cualquier lugar "asegura que el sistema alcanzará un

comportamiento estacionario con el tiempo". Hay que considerar que múltiples comportamientos estacionarios son posibles.

- Estructuralmente viva, una RdP es estructuralmente viva si existe alguna marca inicial para la que está viva.
- Conservativa, si se cumple que para todo marcado m , perteneciente al conjunto de marcados de RdP, la cantidad de marcas de m es igual a la cantidad de marcas de m_0. Se conserva la cantidad de tokens.
- Repetitiva, si existe siempre una secuencia de disparos que lleve desde una marcado m hasta el mismo marcado m
- Consistente, una RdP N marcada (R, m_0), se dice es consistente si el árbol de cobertura tiene un circuito dirigido (no necesariamente primaria) que contiene todas las transiciones al menos una vez [6]. La RdP es parcialmente consistente si un circuito dirigido contiene sólo algunas de las transiciones. Se puede mostrar la consistencia con el árbol de accesibilidad.

Mientras que los métodos dinámicos, que dependen del marcado inicial, son los más usados, entre los que se encuentran:

- Alcanzabilidad, el conjunto de alcanzabilidad, para una RdP $\boldsymbol{N}$ marcada (R, m_0), si A es el conjunto de las marcas alcanzables, que se denota por A (R, m_0), estas son alcanzable por una secuencia de disparos.
- Activa o vivacidad (liveliness), se dice que una RdP está viva si en todo momento sus transiciones pueden ser disparadas o si existe una secuencia de disparos validos que lleven a que la transición pueda dispararse. Esto implica que es posible volver siempre a un estado determinado, por ejemplo para la re-inicialización del sistema. El marcado inicial coincide con la re-inicialización, esto significa que el sistema nunca pierde sus capacidades. Puesto que la vivacidad es una restricción muy fuerte se clasifica a la vivacidad de las transiciones según su grado, donde el grado cero corresponde a una transición muerta y el cuatro corresponde a una transición completamente viva.
- Acotada, una RdP $\boldsymbol{N}$ marcada (R, m_0), es k-acotada o acotada si el número de token en cada lugar no es superior a un número finito k para cualquier marcado alcanzable desde m_0, es decir, m$(p) \leq k$ para todo lugar p y todo el conjunto de marcas A (R, m_0).
- Seguras, si todas sus plazas son 1-limitadas (1-safe). Por lo que una transición no puede ser disparada si alguna de sus plazas de llegada está ocupada. Esta red verifica que para toda marca alcanzable $m(p) \leq 1, \forall\, p \in P$
- Interbloqueo (Deadlock), en una RdP ocurre interbloqueo cuando se alcanza un marcado desde el que no se puede disparar ninguna transición, la red está muerta.
- Reversible, una RdP $N = (R, m_0)$ marcada, se dice es reversible si el marcado inicial es accesible desde todas las marcas alcanzables. Esto es: $\forall\, m \in (R, m_0), m_0$ es alcanzable desde m. Lo que es equivalente a decir que: $\forall\, m \in (R, m_0),\ se\ cumple\ m_0 \in (R, m)$.
- Estado Home, una RdP $\boldsymbol{N}$ marcada (R, m_0), tiene un estado home si para todo marcado $m \in R(m0)$, m_0 es alcanzable desde m.
- Cobertura, dada una RdP $N = (R, m_0)$ marcada, se dice que una marca m es cubierta si $\ni\ m' \geq m(p_i)\ \forall\, i \in [1, \dots, n]$. El concepto de cobertura está relacionado con el concepto de disparabilidad potencial o vivacidad del tipo L_1 [7]. Por ejemplo, supongamos que m es el mínimo marcado necesario para que la transición t pueda

dispararse. Entonces t estará muerta, si no es cubierta. La definición de cobertura muestra que se puede dar una respuesta concluyente sobre cobertura siempre que la RdP sea limitada es decir, se trata de un árbol de accesibilidad. Siguiendo la misma lógica de la accesibilidad, la prueba de cobertura del árbol cobertura no es concluyente.

- Persistencia, dada una RdP $N = (R, m_0)$ marcada, se dice es persistente si, para cualquiera de dos transiciones habilitadas, el disparo de una transición no deshabilita la otra. Por lo que podeos decir que: una RdP es persistente si, $\forall\, m \in A(R, m_0)$, una transición habilitada puede desactivarse sólo por su propio conjunto $\bullet t$. Esto significa que, una transición persistente, una vez sensibilizada se mantendrá activada hasta que se dispara. Esto implica, un RdP que tiene conflictos no puede ser persistente, puesto que un mismo lugar es una entrada a más de una transición, lo que es una situación de conflicto. Por lo que, las redes persistentes son siempre libres de conflicto. La noción de persistencia es importante en el contexto de los circuitos asíncronos, donde las velocidades son independientes, y en el diseños de programas paralelos [7].
- Conservativas, dada una RdP $N = (R, m_0)$ marcada. es conservativa si se cumple que para todo marcado m del conjunto de marcas A (R, m_0), perteneciente al conjunto de marcados de RdP, la cantidad de marcas de m es igual a la cantidad de marcas de m_0. Se conserva la cantidad de tokens.

El desarrollo teórico de estos conceptos básicos, que se han desarrollado con las RdP, ha permitido formular las bases para la teoría de comunicación entre componentes asíncronos de sistemas computacionales, la relación entre eventos[8], y la descripción del comportamiento de sistemas concurrentes en términos de relaciones causa efecto[9].

Las RdP representan una alternativa gráfica y matemática para el modelado de sistemas paralelos, concurrentes, asíncronos, no-determinísticos, distribuidos y/o estocásticos [7, 10]. Una RdP es un modelo formal abstracto de flujo de información que posibilita el análisis de sistemas y procesos, ya que permite modelar el comportamiento y la estructura de un sistema, llevando el modelo a condiciones límites que en un sistema real son difíciles de lograr.

Sistemas Multi-Core

Porqué los Multi-Core

La evolución de los procesadores es consecuencia de la mayor integración y la composición de distintos tipos de funcionalidades integradas en un único procesador. Hoy la disponibilidad de transistores ha hecho factible construir en una sola pastilla varios núcleos de procesador que ha resultado en el desarrollo de los multi-Core. El convertirlas a arquitecturas multi-Core ha sido estimulado por los fabricantes. Es la tendencia emergente de los últimos años, y en particular ha sido justificada por:

- El estancamiento del aumento de rendimiento del paralelismo a nivel de instrucción (ILP), obtenido de las mejoras en la micro arquitectura para extraer paralelismo del flujo de instrucciones, lo que se ha convirtiendo en un costo relativamente elevado, si comparamos la mejora de desempeño con lo invertido (complejidad, lógica, potencia, etc.).
- El incremento en la frecuencia ha llegado al límite, debido principalmente, a las limitaciones de potencia. En [11, 12] se indica que un 1% de aumento de velocidad de reloj resulta en un aumento de potencia del 3%. Lo que da como resultado fuentes de

alimentación mayores, más disipación y ruido de la fuente, más densidad de potencia en la pastilla.

Estas son las principales limitaciones por las cuales emergen como respuesta, para obtener un mayor desempeño, los sistemas multi-Core.

Sistemas Multi-Core y Multiprocesador

Los procesadores multi-Core están compuestos por varias unidades CPU, que residen en una misma pastilla y se interconectan en una placa base. La motivación básica para su uso, es la demanda de rendimiento, ya que con estos es posible obtener tiempos de ejecución menores y mejor rendimiento energético, es decir al menos en teoría menor consumo de energía.

Hoy la venta de procesadores multi-Core es mayor que la de un solo núcleo, tanto para sistemas de alta prestaciones como embebidos. Por ejemplo, a finales de 2006 Intel tropezó con la barrera del giga Hertz, por lo que comenzó a vender más procesadores multi-Core, para los segmentos de procesador de escritorio y servidores. Hoy los procesadores de un solo núcleo tienen interés en el mercado cuando se prioriza el costo.

La Figura 1 muestra tres diseños distintos procesadores[11]:

- Mono-núcleo (un procesador)
- Multiprocesador (multi procesadores en distintas pastillas)
- Multiprocesador / Multi-núcleo (multi procesadores en las mismas y distintas pastillas).

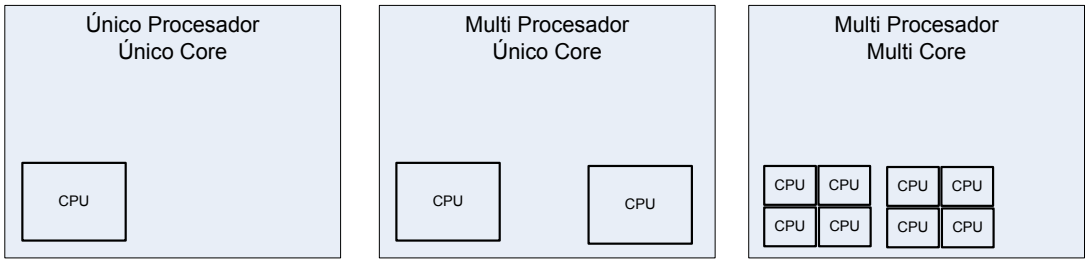


Figura 1: Tres configuraciones de procesadores

Los multiprocesadores, son sistemas que llevan mucho tiempo en el mercado, por ejemplo los primeros x86 de Intel multiprocesadores fue el sistema IPSC a finales de 1980, compuestos por un conjunto de procesadores Intel i386, en conexión de cubo.

El Procesador Multi-Core

Los procesadores Multi-Core tienen dos o más núcleos de procesamiento en la misma pastilla. Se diferencian de lo multi-procesadores, esencialmente en:

- Latencia de Comunicación típicamente menor.
- Ancho de banda mayor, dada las menores distancias de interconexión por estar en la misma pastilla.
- Número de procesadores.

En la siguiente Tabla 1 se muestra los rangos de valores típicos para procesadores actuales.

Tabla 1: Latencia y ancho de banda entre los dos núcleos en multi-Core y multi-procesador

	Mecanismo de Comunicación	Latencia	Ancho de Banda
Multi core	L1[1] MM[2]	10-20 ciclos 100 ciclos	256 bits 64 GBs
Multi Procesador	FSB[3]	En el orden del acceso a memoria	667 MHz 5 GBs

La latencia y ancho de banda, entre dos núcleos en el mismo procesador, está en el orden de 10 veces mejor que entre los núcleos en diferentes procesadores.

A diferencia de los multi-procesadores, los multi-Core son interconectados en una misma pastilla, mientras que los multi-procesadores están interconectados por un bus del sistema externo a la pastilla.

Hay distintas maneras de interconectar los multi-Cores, en sistemas SMP. Se muestran en la Figura 2 dos arquitecturas posibles de multi-Core.

Hay ventajas y desventajas de los diseños mostrados en la Figura 2, podemos mencionar que la tendencia actual es compartir el nivel dos de caché, por tener ventajas en los mecanismos de coherencia y utilización de espacio de caché.

También se clasifican los procesadores en:

- **Multi-Core Homogéneo**, que se compone de cores de procesador que admiten el mismo conjunto de instrucciones (ISA) en todos los núcleos.
- **Multi-Core Heterogéneo**, que se compone de cores de procesador que admiten distintos conjunto de instrucciones (ISA) en distintos núcleos.

Los multi-Core heterogéneos, tienen como ventaja emplear cores especialmente diseñados para tareas específicas. Es decir, optimizado según la necesidad. Por ejemplo procesamiento de paquetes de red, procesadores de criptografía, IP para FFT, etc.

[1] Caché de primer nivel.
[2] Memoria principal.
[3] From size bus

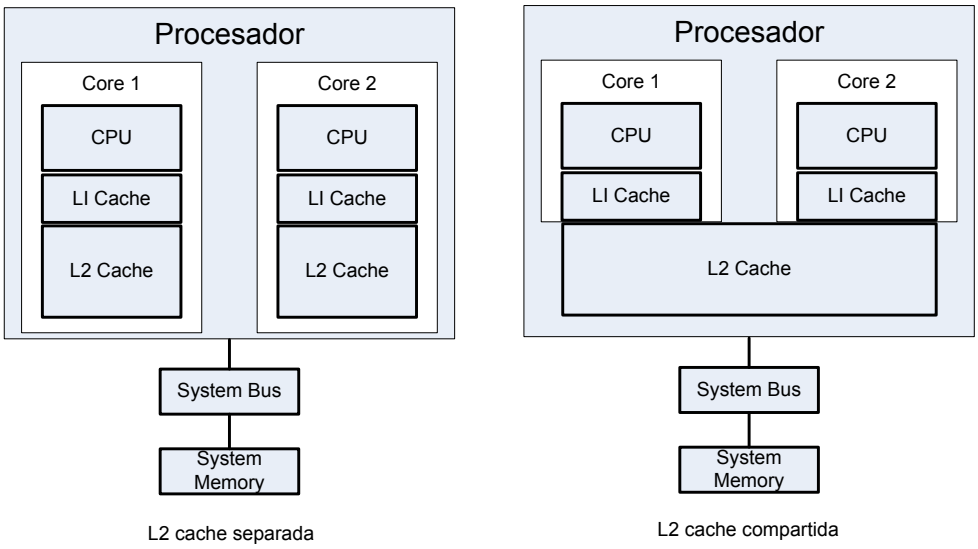


Figura 2: Distintas interconexiones de multi-Core

Su desventaja reside en que el ISE es distinto, lo que dificulta las tareas de desarrollo.

Existe un tipo distinto que son los many-Core[13, 14], se componen de un gran número de núcleos de procesadores interconectados por una red, no necesariamente idénticos en sus funciones individuales.

Otra clasificación de procesadores multi-Core se refiere a cómo los núcleos del procesador se relacionan con el sistema formando un todo. Podemos mencionar dos, que son los multi-Core simétricos y multi-Core asimétricos, que difieren en términos de:

- Acceso a la memoria.
- Sistemas Operativos (SO).
- La comunicación entre los núcleos del procesador.
- Equilibrio de cargas y afinidad al procesador.

Los multi-Core simétricos (SMP), todos los procesadores acceden a todas las regiones de memoria de la misma manera y emplean el mismo tiempo. Generalmente son gestionados y controlados por un único SO. Se caracterizan por compartir la memoria mientras que el SO da soporte a la sincronización y el bloqueo de memoria.

Los multi-Core asimétricos (AMP) se caracterizan porque todos los procesadores no acceden a todas las regiones de memoria de la misma manera y/o con el mismo tiempo de acceso. Si bien pueden ser controlados y gestionados por un solo SO; es frecuente encontrar configuraciones con múltiples SO, donde cada uno es optimizado para el procesador en cuestión. Las áreas de memoria son regiones disjuntas y la comunicación se realiza por paso de mensajes, que también permite la comunicación inter SO.

Los procesadores multi-Core ofrecen la capacidad de usar los recursos de hardware disponibles donde el software específicamente lo requiere. Esto se refleja en mejoras en tiempo de respuesta, mejor rendimiento o códigos. Por ejemplo, en un punto-de-venta (POS), mientras un procesador procesa la actualización de pantalla del terminal, otro procesador procesa las entradas del usuario o la salida de impresión. El servidor que procesa las transacciones, conformado por multi-Core,

donde llegan las solicitudes de las terminales (POS), es capaz de realizar un mayor número de operaciones y el tiempo de respuesta global disminuye.

Ahora bien, la escalabilidad es el grado en que se obtiene mejor desempeño por agregar núcleos al procesador. La Figura 3 muestra, para distintos sistemas, las distintas posibilidades en cuanto al desempeño según agregamos núcleos [15, 16].

El área etiquetada "Superlineal" indica una mejora del rendimiento que escala por encima del número de núcleos del procesador. Esto no es habitual, pero es posible en los casos en que la aplicación se beneficia del mayor tamaño de caché y/o el diseño [17].

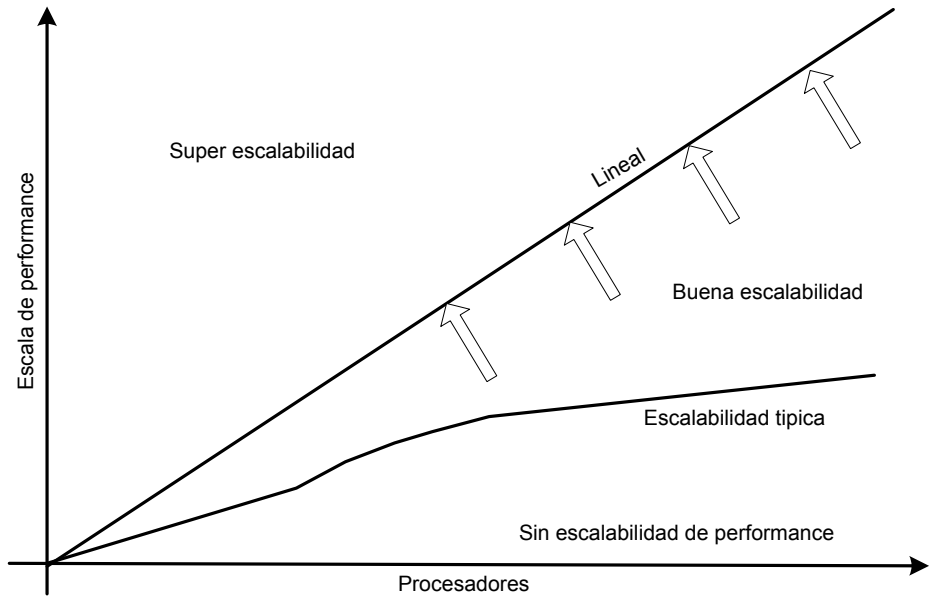


Figura 3: Mejora del desempeño al agregar núcleos

La línea recta corresponde al caso teórico en que el sistema escala, en cuanto al desempeño, proporcionalmente con el número de núcleos. La curva, que está por debajo de esta, muestra el crecimiento típico que una aplicación puede obtener en cuanto al rendimiento.

Como vemos en la Figura 3, el gradiente de desempeño disminuye con el número de núcleos. Esto está motivado por un paralelismo insuficiente o por la sobrecarga de comunicación. El aumento de las comunicaciones, en aplicaciones que colaboran, es inevitable, lo que explica que en muchos casos el crecimiento no sea lineal. Una meta deseable es aproximarse al crecimiento lineal.

También, es necesario considerar que el número de core no puede aumentar ilimitadamente, siendo la comunicación y el consumo por core las variables que impone un límite muy restrictivo [18].

El Software Multi-Core

En la actualidad el reto son los sistemas multi-Core en entorno de memoria compartida.

Las más importantes motivaciones para su desarrollo, que coincidentemente con las de los sistemas mono-Core, son:

- Eficiencia de utilización (índice de utilización), por core y/o módulo.
- Fiabilidad y disponibilidad.
- Rendimiento energético.
- Throughput (volumen de información procesado por unidad de tiempo).
- Capacidad de respuesta del sistema.

Con el fin de continuar aumentando el desempeño, los sistemas son multi-hilos (Hyper-Threading) y multi-Core. Hoy un solo procesador Intel Multi-Core 2 Quad tiene un desempeño del orden de 30.000 MFLOPS. Estos sistemas implementan sus algoritmos haciendo uso de PVM o MPI [19], permitiendo mejor desempeño en cuanto a tiempos y volumen de información.

Por lo que el desarrollo de software debe aprovechar estas características, para lo cual podemos considerar las siguiente tres alternativas:

- No hacer nada.
- Multi-tarea o partición.
- Multi-hilo.

La primera opción, "No hacer nada", tiene como beneficio "mantener el mismo software sin cambios". Pero el desempeño no aumenta, ya que el código no aprovecha los múltiples núcleos o multi hilos y sólo tomara ventaja de los incrementos de rendimiento que resultan de la arquitectura (mejor aprovechamiento del ILP), la frecuencia o las herramientas de software de optimización automática.

La segunda opción es "multi-tarea o partición", es decir ejecutar múltiples procesos al mismo tiempo. Lo que se logra particionado las actividades y asignándolas a los distintos núcleos. Esta es una gestión que realiza el sistema operativo (SO), y debe estar prevista en las aplicaciones. Esta estrategia obtiene mejor rendimiento en procesadores multi-Core.

Para aplicaciones embebidas, la partición es una técnica clave que puede conducir a mejoras sustanciales en el rendimiento y/o la reducción de costos.

Finalmente la "multi-hilo", es una estrategia que generalmente permite obtener importantes beneficios de rendimiento. Pero requiere de más trabajo en el diseño de las aplicaciones, ya que emergen con fuerza la problemática de los sistemas concurrentes.

La Figura 4, muestra un escenario de dos clases de desarrollos de software, con relación a procesadores multi-Core y su desempeño según el tiempo (eje x) y la capacidad del software en explotar los recursos [11]. El eje x representa el tiempo (la línea cortada corresponde al año 2005), mientras transcurre el tiempo aparecen nuevos procesadores. El eje y representa el rendimiento de la aplicación.

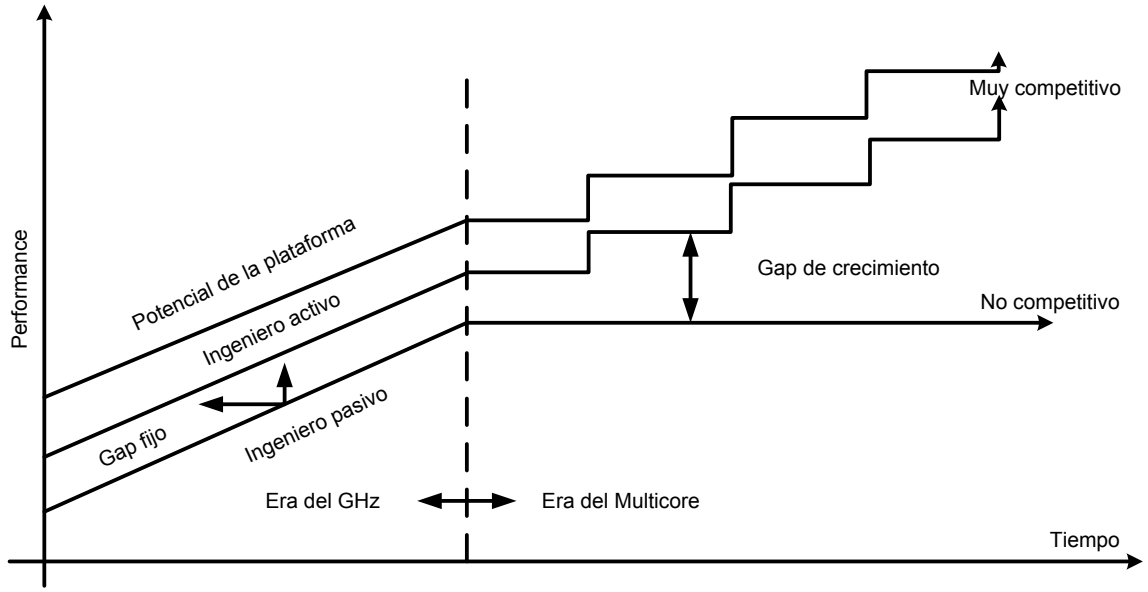


Figura 4: Como tomar ventaja de los multi-Core

La línea superior se denominada "Potencial de la Plataforma" y representa el máximo desempeño teórico de una determinada plataforma. En general, no es posible alcanzarla.

La línea central representa el rendimiento alcanzado por los desarrolladores que invierten esfuerzo en la optimización.

La línea inferior representa el rendimiento obtenido por los desarrolladores que no aprovechan los beneficios del hardware (multi-Core).

En el período de tiempo marcado como la era de "Gigahertz" los desarrolladores podían confiar en el incremento automático del rendimiento del software que resulta de la mejora en el desempeño del hardware, como consecuencia del incremento de la frecuencia y la disminución del CPI (ciclos por instrucción). Por lo que, la brecha relativa entre aquellos que hicieron el esfuerzo y los que no lo hicieron era recuperable en un corto plazo.

La era Gigahertz comenzó en el año 2000 con los procesadores de frecuencia mayor a 1 GHz y terminó en 2005 con la introducción de procesadores multi-Core.

Después del 2005, los procesadores entran en la era de los "multi-Core", donde la nueva tendencia del hardware es obtener mejor rendimiento por el incremento de la cantidad de cores. Pero esto requiere esfuerzo en el desarrollo de software, realizar multi-hilos colaborativos, lo que implica concurrencia.

Ahora, los desarrollos de software deben contemplar la programación multi-hilo para obtener mayor desempeño. Puesto que la obtención de las mejoras emergentes por el hardware, en la era del Gigahertz, ya no son posibles.

Por lo antes dicho, en la era del "multi-Core" la programación paralela es determinante para la mejora del desempeño del software en todos los segmentos de desarrollo (servidores, aplicaciones de escritorio, embebidos, etc.).

Sería fantástico si el software a ejecutar en las nuevas plataformas, aprovechara automáticamente los multi-Core, pero esto hoy es complicado por que generalmente demanda esfuerzo para realizarlo.

En este trabajo se aprovecha esta oportunidad o al menos se mitiga el esfuerzo en ciertos escenarios para lograr el aprovechamiento de los multi-Core.

Porqué Procesadores Multi-Core

Si bien no hemos encontrado ningún estudio concluyente que apoya la afirmación que el rendimientos del ILP está agotado, se puede inferir a través de la tendencia de los procesadores actuales que al menos es complejo obtener mejoras reales, puesto que se fabrican mayoritariamente procesadores multi-Core con los transistores disponibles y no mono-Core.

La Figura 5, proporciona una idea por qué incrementar los núcleos es conveniente en la actualidad.

Las barras de la Figura 5, comparan la potencia y el rendimiento de tres configuraciones, que ejecutan un conjunto de aplicaciones.

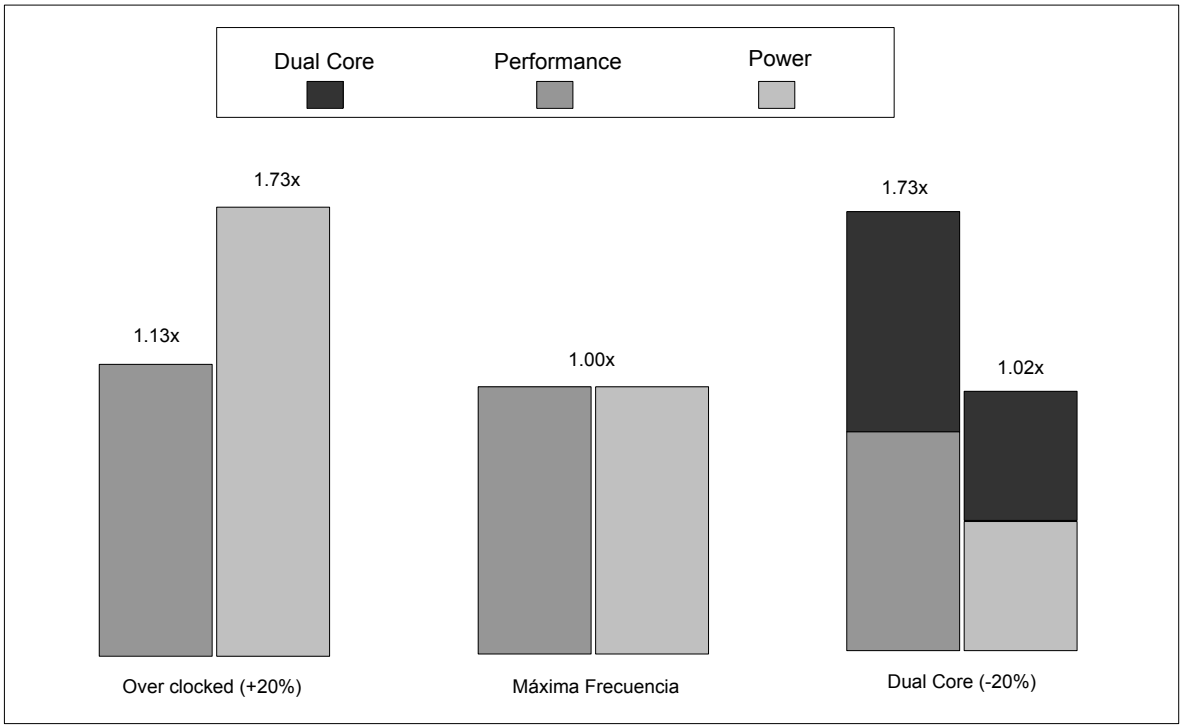


Figura 5: Relación de desempeño y potencia en un escenario de multi-Core

De izquierda a derecha podemos observar lo siguiente:

- Procesador estándar mono-Core, en el que se ha forzado la velocidad en un 20%
- Procesador estándar mono-Core
- Procesador dual-Core, cada núcleo sub-registró un 20%

Comparando la configuración 1 con la configuración 2 se observa que el rendimiento, para la aplicación usada en esta medición, se incrementa 1,13 x mientras que el aumento de potencia es de 1.73x.

Ahora, comparando con la configuración 3 se tiene que el rendimiento aumenta en un factor de 1.73x y la potencia con un factor de 1.02x.

Para este caso, la implementación de multi-Core, tiene mayor rendimiento con menos incremento de energía en comparación con el primer caso donde se aumentó la frecuencia.

Debemos hacer dos advertencias:

1. Estas observaciones no son válidas para todas las aplicaciones, por lo que no se podrá obtener esta mejora, con los multi-Core, con cualquier tipo de trabajo.
2. El aumento de la frecuencia del 20% y la consecuencia del aumento de la potencia en un 70% es una consecuencia de la tecnología

El punto 2 se explica, porque la frecuencia, tiene una relación cúbica con la potencia, la ecuación aproximada para semiconductor (CMOS) es: $\text{Potencia} = \text{k c f}^3$, donde c es la capacitancia dinámica y f es la frecuencia de reloj.

Esto explica porque un aumento del 20% en la frecuencia se traduce en un aumento del 70% en la potencia.

Ámbito de la Tesis

La ejecución e implementación de sistemas reactivos de tiempo real modelados con RdP en procesadores multi-Core heterogéneos, particularmente en sistemas embebidos.

Palabras claves: Sistemas Concurrente, Procesadores multi-Core Heterogéneo, Sistemas de Tiempo Real.

Los sistemas embebidos y concurrentes [20] [21] generalmente se caracterizan por ser reactivos, es decir, mantienen una permanente interacción con el ambiente y/u otros procesos, y responden según su estado y los estímulos externos [22]. Tienen características diferentes a las de los sistemas puramente transformacionales. En muchos casos, no computan resultados, y suele no requerirse que terminen. Ejemplos de sistemas reactivos son: sistemas operativos, software de control, hardware, etc.

La Figura 7 muestra como un sistema reactivo toma los valores del entorno los procesa según sea su estado y los retorna al entorno. Con el fin de compararlo con un sistema funcional se muestra la Figura 6.

Existen numerosas aplicaciones reactivas que son interactivas e impulsadas por eventos, los que se originan desde las aplicaciones y/o su entorno. Estas aplicaciones, orientadas a eventos, procesan los eventos y realizan tareas tales como la actualización del estado de la aplicación y la visualización de datos [23]. Por ejemplo, las interfaces gráficas de usuario son muy interactivas puesto que normalmente tiene que reaccionar y coordinar múltiples eventos.

Estas aplicaciones son difíciles de programar usando enfoques de programación secuenciales, debido a que es dificultoso predecir o controlar el orden de llegada de los eventos externos, puesto que, el sistema cambia de estado según arriban los eventos. Esto implica que el flujo de control de

un programa es conducido por eventos y no por un orden especificado por el programador. Además, cuando hay un cambio de estado por un cálculo o dato, se requiere que el programador actualice manualmente todos los datos dependientes de éste. Este manejo provocado por los cambios de estado y/o dependencia de datos es complejo y propenso a errores; por ejemplo, la realización de los cambios de estado en un momento equivocado o en un orden equivocado.

El paradigma de la programación reactiva ha sido propuesto como una solución adecuada para el desarrollo de aplicaciones orientadas a eventos [23]. Donde se proponen, la abstracción en el lenguaje para la programación de aplicaciones orientadas a eventos, para expresar como el programa reacciona a eventos externos, realiza la gestión automática del flujo de tiempo y los datos, y las dependencias de cálculo. Esto facilita a los programadores la programación del orden de los acontecimientos y las dependencias de cálculo. Para esto, los lenguajes de programación reactivos abstraen la gestión del tiempo, al igual que los recolectores de basura abstraen la gestión de memoria. Esta propiedad de la gestión automática de dependencias de datos se puede observar en el algoritmo de Tomasulo [24]. Donde la ejecución de la instrucción se realiza automáticamente según el orden en que se obtienen los argumentos.

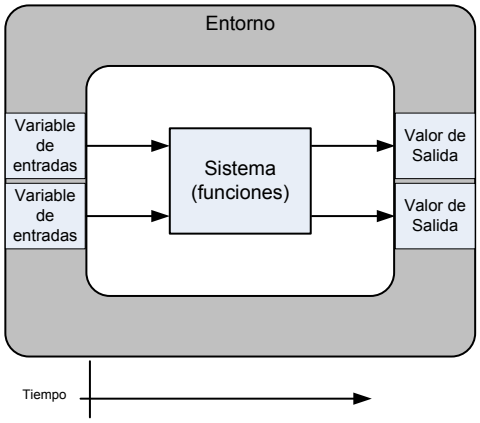


Figura 6: Sistema funcional (transformacional)

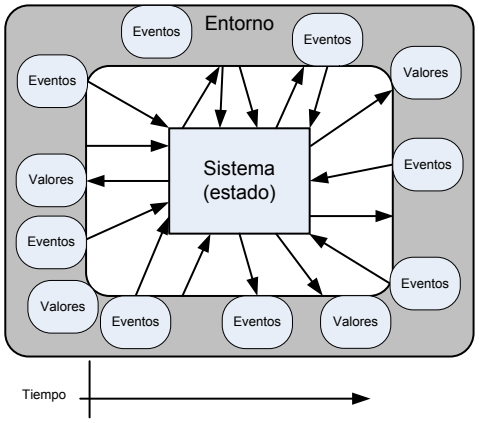


Figura 7: Sistema reactivo

Campos de aplicación de la programación reactiva son los modelos y simulación [25], la robótica [26], visión por computador [27] y la iluminación del escenario [28].

Los sistemas reactivos y concurrentes están compuestos por procesos (o threads o componentes) que necesitan interactuar. Existen varios mecanismos de interacción entre procesos. Entre éstos se encuentran la memoria compartida y el pasaje de mensajes.

Además, los programas concurrentes deben, en general, colaborar para llegar a un objetivo común, por lo cual la sincronización entre procesos es importante.

Típicamente la semántica de los programas concurrentes está basada en sistemas de transición de estados. Un sistema de transición de estados es un grafo dirigido en el cual:

- Los nodos son los estados del sistema (la cantidad de estados pueden ser finitos o infinitos)
- Las líneas son las transiciones atómicas entre estados, dadas por una sentencia atómica (eventos) del sistema.

Hay un nodo singular, que reconoceremos como el estado inicial, en la Figura 8 está representado por la flecha que entra al nodo y no tiene origen.

Una semántica típica para programas concurrentes se basa en sistemas de transición de estados, cuya máquina de estado se representa en la Figura 8. Un sistema de transición de estados es un grafo dirigido que podemos definir como sigue:

$$(S, s_0, \longrightarrow)$$

$$S \text{ es el conjunto de estados}$$

$$s_0 \text{ es el estado inicial}$$

$$Relacion\ de\ Transición$$

$$S = \{0,1\} x \{0,1\}$$

$$s_0 = (0, 0)$$

$$(x, y) \longrightarrow (x, 0)$$

$$(x, y) \longrightarrow (x, 1)$$

$$(x, y) \longrightarrow (x, y)$$

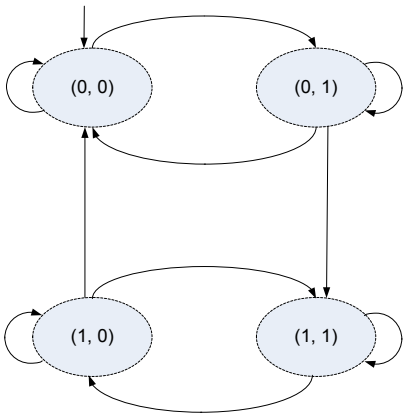


Figura 8: Máquina de estado, para la relación de transiciones

La ejecución del programa, que implementa el sistema, es una secuencia de estados $s_0 s_1 s_2 \ldots s_n$, tal que: s_0 es el estado inicial, para llegar desde s_i a s_{i+1} se lo hace mediante una sentencia atómica (para cada sentencia atómica tendremos una transición) del sistema.

El conjunto de todas las ejecuciones determina el comportamiento del programa concurrente.

En general, es muy difícil razonar sobre programas concurrentes y más aún determinar si el programa es correcto.

Una de las principales razones son las diferentes intercalaciones (interleaving) de acciones atómicas, lo que pueden llevar a diferentes resultados o comportamientos de los sistemas.

El número de los distintos interleaving posibles, en general, es muy grande, lo que hace dificulta obtener resultados confiables a partir de pruebas (testing); es decir que para programas concurrentes es dificultoso determinar su buen funcionamiento solo con testing.

Los sistemas críticos de tiempo real, como lo de controles de alta energía, sistemas de vuelo, sistemas de robots, etc., son sistemas que plantean una problemática compleja y riesgosa, y son sistemas reactivos.

La corrección de su buen funcionamiento depende no solo de los resultados lógicos del cómputo, sino también del instante en el cual se producen los resultados. En un sistema crítico, de tiempo real, el cumplimiento de los requisitos temporales es fundamental para su correcto funcionamiento.

En los sistemas de tiempo real, una componente fundamental es la concurrencia, puesto que estos sistemas interactúan con el entorno el cual evoluciona en paralelo con el sistema. Las RdP especifican concurrencia efectiva, por esto son usadas como referencia para cotejar diferentes modelos de concurrencia.[8]

Las RdP también son aplicadas al modelado y verificación formal de las propiedades de los sistemas de tiempo real [29].

Los sistemas de tiempo real requieren un análisis de planeación para comprobar que se cumplen las limitaciones de tiempo, ésta debe ser anterior a la de la aplicación real que se ejecuta.

El análisis de planeación se aborda en el supuesto de que el sistema en tiempo real es descrito por los procesos concurrentes. Sin embargo, las RdP de alto nivel pueden ser utilizadas para proporcionar una representación conceptual más adecuada que los procesos concurrentes.

Las RdP temporales nos permiten, anticiparnos para determinar la performance de un sistema, mientras se mantiene la especificación formal de los requerimientos [30].

Uno de los objetivos es proporcionar un entorno de especificación fácil de utilizar y con base formal para aplicaciones en tiempo real.

Por lo que nos basamos en un formalismo que ofrece un sólido sustento matemático.

El lenguaje es un medio para la formulación de especificaciones sin ambigüedades, que pueden ser formalmente verificados en cualquier etapa del proyecto.

La capacidad de verificar las especificaciones en las primeras etapas del proyecto es muy importante para revelar errores en su corrección; se puede llegar a un costo mucho menor en comparación con el costo de la eliminación de los mismos errores en fases posteriores.

Podemos considerar a las RdP como un “monitor” de alto nivel, que permiten que las especificaciones sean ejecutadas, simuladas y/o emuladas; verificando y validando formalmente el sistema.

La validez del código del sistema, la verificación de requisitos tanto funcionales como temporales en etapas tempranas del desarrollo, antes de su codificación, se posibilita extendiendo las RdP con tiempo y eventos de entrada salida. Las RdP con extensión temporal son una herramienta muy difundida para el análisis y diseño de sistemas concurrentes[31] [32] [33], como también lo son las RdP no autónomas [34, 35].

La descripción de la conducta de sistemas concurrentes en términos de relaciones causa efecto son muy deseadas [9], también que dichas especificaciones sean ejecutables [36], como también es importante la flexibilidad en las herramientas de desarrollo de un sistema.

Según nuestro buen entender y conocimiento, no son conocidos trabajos que relacionen modelos de RdP de bajo nivel o RdP temporales directamente con la ejecución del sistema haciendo uso de un procesador de Petri, con programación directa y programación en tiempo de ejecución. En cambio si encontramos que existen sistemas que ejecutan RdP por software.

Los aspectos de la generación directa del código a partir de la RdP y la ejecución por hardware de la RdP subyacente, serán los principales objetivos cubiertos por la tesis.

Objetivos y Aportes de la Tesis

La presente tesis se sitúa en el entorno de un proyecto más general, cuyo principal objetivo es la introducción de dos nuevos procesadores que llamaremos "Procesador de Petri" (PP) y "Procesador de Petri Jerárquico" (HPP), como soft-Core o hard-Core. Donde su programación se realizará en forma directa, mejorará el desempeño en la ejecución de software para sistemas reactivos, concurrentes y de tiempo real, en aplicaciones de sistemas embebidos. Esta implementación se sustenta en la capacidad formal que poseen las RdP y las RdP Temporales.

Dichos procesadores tendrán las siguientes características: A partir de un modelo del sistema realizado con una RdP, obtener en forma directa y exacta un código ejecutable, por el PP o HPP de dicho modelo.

El PP o HPP que ejecute dicho código lo haga para colaborar con los procesos en ejecución con distintos procesadores.

- Que el código generado sea independiente de los procesadores que requieran de la colaboración del PP o HPP.
- Que los eventos emergentes del PP o HPP sean independientes de los procesadores que requieran de su colaboración

Dentro de este amplio marco, esta tesis se ha enfocado sobre tres aspectos: el modelado, la implementación de los PP y HPP, y su programación.

Por todo ello, los objetivos que se pretenden alcanzar en este trabajo son los siguientes:

1. Crear, diseñar e implementar un procesador programable según el formalismo de las RdP, para redes de bajo nivel, temporales y jerárquicas, que es el objetivo fundamental de la tesis, y que a priori constan de los siguientes ítems:
    - Soportar aplicaciones de sistemas reactivos y concurrentes en tiempo real.
    - Valorar el impacto sobre las prestaciones del sistema.
    - Implementación de los PP y HPP.
    - Generación automática y directa del código que programa al PP y HPP.

2. Convertir en forma automática el modelo en los aspectos de control, concurrencia y temporales de sistemas de tiempo real, utilizando RdP y RdP con tiempo.
3. Ejecutar el modelo basado en el empleo de RdP y RdP extendidas con tiempo [32] de sistemas concurrentes y de tiempo real.

No son abordados en esta tesis el estudio de la planificación de sistemas de tiempo real a través de las RdP, el análisis de sus propiedades y aspectos de tolerancia a fallos.

Organización de la Tesis

La tesis resta dividida en tres secciones:

- Primera sección
  - Capítulo 1, donde se expone la motivación, la planeación general y el objetivo de la tesis.
- Segunda sección
  - Capítulo 2, donde se exponen los antecedentes de trabajos relacionados y el marco teórico de esta tesis.
- Tercera sección
  - Capítulo 3, expone el análisis de un mecanismo para mejorar la sincronización y exclusión mutua entre procesos y/o hilos, haciendo uso de las RdP sincronizadas o no autónomas.
  - Capítulo 4, se desarrolla un IP-Core cuya responsabilidad es ejecutar el algoritmo de Petri para mejorar la sincronización entre procesos.
  - Capítulo 5, se extiende el PP a PP con tiempo y temporizados, hacemos uso de los formalismos de las RdP con tiempo.
  - Capítulo 6, se dividen las RdP lo que permite determinar un algoritmo alternativo para la ejecución de las RdP y posibilita su implementación en hardware, obteniendo una disminución sustancial de recursos de hardware.
  - Capítulo 7, se presenta el desarrollo de un software para dar soporte a un PP que interactúa con un sistema real.
- Cuarta Sección
  - Capítulo 8, Análisis de resultados, conclusiones y líneas abiertas
- Anexos
  - IP-Core, Monitores, RdP no-autònomas, Estudio recursos para la implementación del PPcT, Arquitectura del PP para brazos con peso uno, Acrónimos, índice de tablas, índice de figuras e índice de definiciones

Producción Científica Derivada de Resultados Parciales de la Tesis

Se ha determinado, en una arquitectura multi-Core SMP, el lugar donde incorporar el PP o el HPP sin alterar el ISA del resto de los core.

Se ha obtenido una familia de procesadores que ejecutan los algoritmos de Petri para dar solución a sistemas reactivos y concurrentes, con una sólida verificación formal que permite la programación directa de los procesadores. Para esto, se ha construido el hardware de un PP y un HPP, con un IP-Core en una FPGA, integrado a un sistema multi-Core SMP, que ejecuta distintos tipo de RdP.

Esta familia de procesadores es configurable en distintos aspectos:

- Tamaño del procesador (cantidad de plazas y transiciones).
- Procesadores con tiempo y procesadores temporales.
- Arquitectura heterogénea, que permite distribuir los recursos empleados para instanciar el procesador según se requiera, y obtener un ahorro sustancial.
- La posibilidad de configurar el procesador en pos de obtener los requerimientos y minimizar los recursos. Muy valorado en la construcción de sistemas embebidos.

En los sistemas con alta necesidad de concurrencia y sincronización, donde se ha evaluado este procesador, las prestaciones han mostrado una importante mejora en el desempeño.

El procesador tiene la capacidad de resolver simultáneamente, por conjuntos múltiples disparos, lo que disminuye los tiempos de consulta y decisión, además los programas ejecutados cumplen con los formalismos de las RdP extendidas y sincronizadas, y los resultados de su ejecución son determinísticos. Los tiempos de respuesta para determinar una sincronización son de dos ciclos por consulta (entre la solicitud de un disparo y la respuesta).

Se ha conectado al procesador con los procesos, esto se ha logrado programando: los tipos de eventos que soporta los procesadores (perennes, no-perenne y simultáneos), la relación entre los procesos y las transiciones y un driver que integra al PP con el sistema operativo.

Es posible reducir el tiempo de desarrollo y la dificultad de diseño, si se hace uso de módulos probados (redes) y se hacen diseños jerárquicos por componentes. Asimismo aumenta la facilidad para su mantenimiento.

Capítulo 2

Estudio de Antecedentes de Controladores con RdP

Introducción

Con el fin de determinar el estado de desarrollo de los controladores de hardware y el software que interactúa con los controladores/procesadores/software/hardware que ejecutan RdP y su interrelación con otros procesadores, se investiga las publicaciones relevantes relacionadas con los objetivos de esta tesis.

Para exponer los resultados, el capítulo se ha divido en tres secciones las que son:

- Distintos tipos de controladores implementados por hardware, interpretados o cableados (compilados).
- Distintos tipos de controladores implementados por software, interpretados o compilados.
- Distintas formas en que han sido divididas las RdP, para su comprensión, diseño y generación de espacio de estados.
- Distintas interfaces de comunicación entre el procesador/controlador y los procesos

Esto se desarrolla en los siguientes apartados.

Objetivos

Estudiar y determinar el estado de desarrollo del hardware de un procesador que implemente el algoritmo de las RdP y los mecanismos de comunicación con el software de otros procesadores.

Situación Temporal del Trabajo

El primer investigador en publicar hardware para ejecutar de RdP ha sido "Hideki Murakoshi" [37] [38] [39] [40] [39] [40] [41] [42] [43]; quien implementó un algoritmo de disparo de la una RdP en una ASIC. Este trabajo fue continuado, para dividir una gran RdP con el fin de mapearla en una ASIC con un número limitado de pines.

Actualmente se han realizado y publicado una importante cantidad de trabajos por la "Universidade Nova de Lisboa - Faculdade de Ciencias e Tecnologia, Portugal UNINOVA - CTS, Portugal" [34] [44][73][45-53][167][54-56].

Estos trabajos y otros han sido analizados para la realización de esta tesis, aquí presentamos las partes relevantes de los trabajos en relación con los de objetivos y fundamentos de esta tesis.

La Importancia de esta Investigación

El presente trabajo se motiva en los largos tiempos de ejecución demandados por los simuladores de TPN para modelos complejos. El desarrollo de software que requieren de estrictas especificaciones de rendimiento temporales demanda la generación, simulación y análisis de modelos computacionales adecuados. Este tipo de software es necesario para las redes de información, sistemas distribuidos y sistemas de control, aeroespacial / defensa, etc. Y dá la oportunidad de aprovechar el paralelismo disponible en los sistemas multi-Core.

Actualmente, la lógica programable ha avanzado y abierto grandes oportunidades para usarla con RdP. La principal aplicación, de la implementación en procesador de RdP en FPGA, sería la mejora del desempeño del software específico de los sistemas modelado por una RdP que se desea ejecutar.

Es la primera etapa, de este proyecto, se determina las posibilidades de ejecución del modelado de RdP con tiempo en un sistema de tiempo real y se determina el desempeño.

Esta simulación es importante puesto que se obtiene anticipadamente la validación de requerimientos y la estimación de desempeño, previo a la codificación del software.

El Tipo de Problema que se Estudia y la Metodología

Este trabajo es el primer paso para construir un sistema con capacidad de simulación de sistemas de tiempo real.

Se trata de "Una implementación con FPGA de un algoritmo de disparo para RdP". Se realizan aportes, sobre el diseño con FPGA, del algoritmo de disparo y el sistema de comunicaciones entre los procesos y las transiciones. Para integrarlo como un procesador de eventos, que se usará en simulación y ejecución de software basados en el modelado de RdP para tiempo real.

El Alcance y Resultados de la Investigación

El objetivo de esta investigación es determinar la viabilidad de la utilización de hardware para acelerar la ejecución de programas con alta demanda de concurrencia.

Hay otras cuestiones fundamentales, como la representación del modelo de Petri, determinar y validar en la FPGA el algoritmo para resolver las indeterminaciones de los disparos múltiples, reconfigurar dinámicamente la FPGA cada vez que el diseñador hace un cambio de la estructura de interconexión en la RdP con tiempo (si es posible), o si en la FPGA sólo es posible en ejecución alguna parte de la estructura de RdP, que permanece estática, y si es posible cargar nuevos parámetros, para la ejecución de la RdP, en tiempo de ejecución. También nos preguntamos si es posible convertir una RdP de envergadura en representaciones más pequeñas que mantengan las características del modelo y sea posible de ejecutar.

Es importante la programación de distintos tipos de eventos y señale que el PP acepta, con el fin de que éste se adapte a distintos tipos de escenarios. Esto también es válido para los eventos y señales de salida.

Los Puntos que no son Considerados

La validez del modelo que implementa la RdP a ejecutar. El foco de esta investigación es ejecutar el modelo que la RdP representa por lo cual es necesario verificar que este represente correctamente al sistema.

La tolerancia a fallos, si bien, se han implementado algunos mecanismos en el procesador para soportar mecanismos de tolerancia a fallos, es posible ampliarlos y se ha decidido implementar estas características después de implementar el PP. En este trabajo solo se detecta el interbloqueo de la red y se programa al disparo de algunas transiciones con una interrupción.

Revisión de la Literatura

Antecedentes de Hardware/Software para Controladores o Procesadores de Petri (PP)

Con el fin de determinar el "estado de desarrollo" del hardware/software de controladores/procesadores que ejecutan RdP, cómo se relacionan los eventos y los "fundamentos para su desarrollo" se realiza la investigación bibliográfica referida al tópico.

¿Que se estudia y expone en el capítulo dos?

- Se estudian, exponen y fundamentan los distintos tipos de controladores implementados por hardware, interpretados o cableados (o compilados).
- Se estudian, exponen y fundamentan los distintos tipos de controladores implementados por software, interpretados o compilados.
- Se estudian y proponen distintas interfaces de comunicación entre el procesador/controlador y los procesos
- Se estudian las distintas formas en que han sido divididas las RdP, para su comprensión, diseño y generación de espacio de estados.
- También, se exponen distintos casos implementados con RdP, con el fin de determinar el ámbito de aplicación de esta investigación.

- Esto se desarrolla en los siguientes apartados de este capítulo.

Estado de Desarrollo de los Distintos Controladores o Procesadores de RdP

Según los tipos de redes y prioridades, eventos y programación

Para conocer el estado de desarrollo y poder realizar una comparación, con el PP implementado en esta tesis, se establecen las distintas características posibles de implementar en un controlador/procesador de RdP. Estas características son agrupadas en las Tabla 2 y Tabla 3 donde se han enumerado, descriptos y colocados un acrónimo para referirnos a la característica soportada por la implementación.

Tabla 2: Características, con respecto a los tipos de redes, del PP posibles de implementar

Con respecto a los tipos de redes		
Numero	Tipo de red y arcos	Acrónimo
1	Redes ordinarias	RO
2	Redes con plazas limitadas	RPL
3	Arcos regular (con peso uno)	APU
4	Arcos con peso entero	APE
5	Arcos inhibidores	AH
6	Arcos de lectura	AR
7	Arcos reset	AR
8	Arcos stopwatch	AS
9	Arcos stopwatch inhibidor	ASH
10	Arcos con función booleana	AFB
11	Transiciones con tiempo	TT
12	Transiciones demoradas	TD
13	RdP jerárquicas	RdPJ

En la Tabla 2, las características 1 a 7, son necesarias para expresar distintos requerimientos y demandan flexibilidad en la implementación de la transición. Esto es importante, puesto que se obtiene un ahorro significativo de componentes. Si se agrupan transiciones con similares características y se implementa un sistema multiprocesador de RdP, donde cada procesador efectúa el requerimiento del grupo, y puesto que es complicado realizar los grupos según los tipos de brazos, ya que se consideran todos los tipos de brazos, y se demanda una división de la red, esta propiedad tiene que ser soportada por el mecanismo de RdP jerárquica.

Las características 8 a 12 son necesarias en los sistemas con tiempo y así responder a los requerimientos de sistemas de tiempo real.

La característica 13 es necesaria para obtener múltiples disparos en un mismo ciclo de ejecución y disminuir recursos en la implementación del PP.

Tabla 3: Características con respecto a las prioridades y eventos del PP posibles

Con respectos a las prioridades, eventos y programación		
Numero	Tipo de disparo y programación	Acrónimo
14	Prioridad en los disparos	PD
15	Eventos de entrada	EE
16	Tipos de eventos de entrada	TEE
17	Eventos de salida	ES
18	Tipos de eventos de salida	TES
19	Programable con matrices y vectores	PMV
20	Programar en tiempo de ejecución	PTE
21	Detección de Interbloqueo	DIB

En la Tabla 3, la característica 14 es requerida para que el sistema sea determinístico.

Las características 15 a 18 son requeridas para comunicar el PP con los programas que corren en los procesadores, que son los que hacen las acciones y la interface con el medio ambiente. Para esto se requieren distintos tipos de eventos o señales, tanto de entrada como de salida.

La característica 18 es requerida para transferir el programa al PP, esto se realiza obteniendo las matrices y vectores del simulador. La programación es tanto de la RdP considerando todos los tipos de brazos, marcado inicial, prioridades, tipo de eventos de entrada y salida, y tiempos para las RdP con tiempo o temporizadas.

La característica 18 es necesaria para reprogramar el PP y aprovechar la localidad, mientras el programa se está ejecutando.

La detección de interbloqueo, característica 21, es necesaria para realizar el testing y debugger del programa, más aún es importante para disminuir la cantidad de recursos.

Las Tabla 4 y Tabla 5 son usadas para sintetizar las características desarrolladas en cada trabajo, donde ¨referencia¨ son las publicaciones de los trabajos realizados por distintos investigadores, y los acrónimos se corresponden a las características implementadas por cada desarrollo.

Tabla 4: Aportes de la referencia con respecto a la los tipos de redes

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ

Tabla 5: Aportes de la referencia con respecto a las prioridades, eventos y programación

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB

Con respecto a la ejecución, programación de la red y división de la red

Para conocer el estado de desarrollo de cómo se relacionan el software y los eventos con los distintos controladores/procesadores de RdP, se construyen las Tabla 6, Tabla 7 y Tabla 8 . Donde se han enumerado, descripto y colocado cada acrónimo referido a la característica soportada por la implementación.

Tabla 6: Características de las redes consideradas con respecto a la ejecución

Con respecto a la ejecución		
1	La red es ejecutada en hardware con un programa específico	EHS
2	La red es ejecutada por un procesador programable para RdP	EHP
3	La red es interpretada por un software	RIS
4	Se soluciona solo un disparo por ciclo de evaluación	DU
5	Se permitirá solucionar múltiples disparos simultáneamente de la red	DM
6	Detección de interbloqueos de la red	DIR
7	Detección de interbloqueos por subred	DIS

En la Tabla 6, la característica 1 se refiere a RdP ejecutadas; mientras que la 2 son RdP interpretadas por un hardware. Aquí, la característica 3 se refiere a RdP interpretadas con la salvedad que lo hace un software que corre en un procesador regular. Las características 4 y 5 hacen referencia a como el sistema maneja las transiciones en conflictos. El disparo de múltiples transiciones permite mejorar el desempeño en la comunicación de los eventos de salida.

Las características 6 y 7 son importantes para las etapas de desarrollo y mantenimiento. La detección de interbloqueo por sub-red es importante para evaluar estados locales en la etapa de desarrollo.

Tabla 7: Características de las redes consideradas con respecto a la programación de la red

Con respecto a la programación de la red		
8	La programación es directa independientemente del tipo de red y la división	PD
9	Es independiente de un lenguaje específico para programar la red	LE
10	Es necesario un programa particular por cada sub-red	PPSR
11	Es necesario un lenguaje en particular para los procesos	LEP
12	La programación en el hardware es en tiempo real	PHTR
13	La programación del software es en tiempo real	PSTR

En la Tabla 7 la característica 8 se refiere a como transferir el programa que ejecuta el intérprete de RdP. Esencialmente hay dos formas: un lenguaje (PNML) o los vectores y matrices con los

que se construye la ecuación de estado. En el caso que se requiera un compilador o traslación, ya sea por el tipo de red o su división, es dependiente.

La característica 9 implica que cualquiera sea el lenguaje o los lenguajes que interpreten o ejecuten a la RdP, ésta siempre se expresa de la misma manera y no requiere compilación.

Las características 10 y 11 implican que el sistema puede ser ejecutado independiente de la plataforma sin cambio o recopilación.

Las características 12 y 13 se refieren a la capacidad que tiene el sistema para cambiar el RdP que se ejecuta en tiempo de ejecución, es decir a reprogramarlo.

Tabla 8: Características de las redes consideradas con respecto a la división de la red

Con respecto a la división de la red		
14	Dividir la red por arco	DA
15	Dividir la red por transición	DT
16	Dividir la red por plaza	DP
17	Se requiere elementos, plazas o transiciones adicionales, para la división	SEA
18	Hay restricciones en la división de la red	RPD
19	Permite la división de la red y combinar de múltiples tipos de redes	CMTR
20	Acepta que la red sea distribuida	RD
21	Acepta que la red sea centralizada	RC
22	Permite la ejecución de múltiple tipos de redes simultáneamente	EMTR
23	Permite redes jerárquicas	RG

En la Tabla 8 las características 14, 15 y 16 se refieren a como se divide la RdP para obtener una RdP jerárquica. Mientras que la característica 17 hace referencia a que hay que duplicar elementos (plazas y/o transiciones) para hacer la división y en consecuencia trae sobrecarga (de comunicación y/o totalización) de ejecución.

La característica 18 se refiere a las distintas restricciones para obtener la división o al hecho de que la red tenga características particulares para poder dividirla en sub redes; como sólo brazos de entrada a una sub red, etc.

La característica 19 permite mejorar la configuración de un sistema, puesto que es posible agrupar las sub redes por recursos, comunicaciones, tipos de eventos, tipos de redes, etc.

Las características 20 y 21 permiten construir sistemas distribuidos y/o centralizados.

La característica 22 es fundamental para abordar sistemas que requieren tiempo, prioridades y distintas redes. Mientras que la característica 23 permite entender y diseñar mejor el sistema, más aún es importante para disminuir la cantidad de recursos.

Las Tabla 9, Tabla 10 y Tabla 11 son usadas para sintetizar las características desarrolladas en cada trabajo, donde ¨referencia¨ son las publicaciones de los trabajos realizados por distintos investigadores, y los acrónimos se corresponden a las características implementadas por cada desarrollo.

Tabla 9: Aportes de la referencia con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS

Tabla 10: Aportes de la "referencia" con respecto a la programación de la red

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR

Tabla 11: Aportes de la referencia con respecto a la división de la red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG

Distintos Tipos de Controladores

Implementados por Hardware, Interpretados o Cableados (compilados)

Implementación del hardware de un sistema multiprocesador controlada por RdP
En el trabajo de Kawahara, Murakoshi, Anzai [37] se presenta una implementación y evaluación de un sistema multiprocesador controlado por RdP. Se diseñó un hardware interconectado con un sistema multiprocesador utilizando HDL (Hardware Description Language). El sistema consta de un controlador de RdP, algunos módulos de cálculo y buses de interconexión. Se ha diseñado una unidad de disparo que comprueba si las transiciones son sensibilizadas o no en Gate-Array. Para determinar la posibilidad de un disparo se requieren tres ciclos de reloj, que comprueba y dispara una transición. Además, se evaluó el rendimiento del sistema, aplicándolo al control de un brazo de robot. Este sistema puede controlar procesos muy rápidos caracterizados por un tiempo tan pequeño como 10 micro-segundos.

Secciones de interés del trabajo
Este trabajo está motivado en la aplicación a robots sofisticados que son muy demandados por la industria y de bajo costo. Para el control de estos robots se requiere de multiprocesadores, para obtener los rendimientos demandados por la industria.

Las RdP son una herramienta matemática y gráfica para describir y analizar sistemas concurrentes, distribuidos y asíncronos, facilitando la implementación de hardware con RdP, aplicada al control de tiempo real con múltiples sistemas, obteniendo mejoras de rendimiento.

Arquitectura propuesta
Se ha propuesto un sistema multiprocesador controlado por una RdP. Este sistema tiene una arquitectura simple y controla procesos paralelos utilizando RdP.

En este sistema, a cada plaza del gráfico de la RdP, se le asigna un proceso y el proceso se ejecuta cuando en el lugar hay un token.

Esto significa que el sistema puede ejecutar los procesos paralelos de acuerdo con la transferencia de token. Sin embargo, todavía no está evaluado el rendimiento del sistema en su modelo de hardware real.

En este trabajo, se implementó por hardware y se evalúa el rendimiento del sistema multiprocesador controlado por RdP.

El sistema propuesto puede acelerar la velocidad del sistema y puede ser implementado en un volumen razonable de hardware.

Se muestra que el sistema es aplicable a los controles de procesos con grano fino.

Componentes principales del sistema controlador con RdP

Se compone de un controlador de RdP, algunos elementos de procesador tales como: elementos de proceso (PE), un Control Común de Memoria (CCM) para la comunicación entre el controlador de RdP y los PEs, y una área de memoria de datos común para el intercambio de datos entre PEs.

El programa de RdP, se almacena en el controlador de RdP, como ¨Tabla de condición de disparo¨ (FCT), ¨Tabla de Transferencia Token¨ (TTT) y ¨Tabla de Atributo de plaza¨ (PAT). El marcado de la RdP se almacena en la ¨Tabla de estado Token¨ (TST).

Los PEs almacenan los programas de las plazas. El FCT indica los lugares de salida de cada transición. Las TTT indican las plazas de salida para cada transición. Las PAT indican atributo de las plazas. Los TST indican la distribución de tokens en las plazas.

El controlador de RdP compara TST con FCT para encontrar una transición sensibilizada; si encuentra una, el controlador de RdP busca los lugares de salida de la transición en el TTT. Si el atributo del lugar es normal, el controlador escribe el número de un lugar a EXQ (cola de ejecución). Si se trata de un lugar final, el controlador de RdP detiene la ejecución del programa de RdP. Cada PE recoge un número de lugar de EXQ, y ejecuta el programa asignado al lugar.

Cuando ha finalizado el programa, escribe en el lugar el número de EDQ (cola final). El controlador de RdP lee un número en lugar EDQ, y reconoce la terminación del programa correspondiente al lugar. Luego, escribe un símbolo en la posición del lugar en TST, y lo compara TST con FCT de nuevo.

Visión general de la Unidad de Disparo

El controlador de RdP requiere muchas tablas para comprobar si la transición es sensibilizada o no. La velocidad del controlador es dominante por la velocidad del sistema. La unidad de disparo es una memoria funcional rápida que comprueba si las transiciones están sensibilizadas.

La In-Table almacena la relación de las plazas de entrada y transición. La Out-Table mantiene la relación entre las transiciones de salida a las plazas.

Los estados necesarios para realizar las acciones de la unidad de disparo, son enumerados a continuación:

1. El controlador de procesador recoge un número de plaza en EDQ.
2. Se busca la salida de una transición que tienen como entrada esta plaza.
3. Se escribe el número de la plaza y el número de transición a la unidad de disparo.
4. La unidad de disparo escribe 1 como un token en TST, y comprueba si la transición esta sensibilizada o no, refiriéndose a In-Table, Out-Table y TST.
5. Si la transición esta sensibilizada, la unidad de disparo actualiza el marcado de TST.

Implementación del sistema controlador con RdP

A continuación se presenta el diseño del conjunto en HDL. Lo que incluye un procesador de control, la memoria, EXQ, EDQ, bus y la lógica de control. El procesador de control supervisa al controlador de RdP. La RdP controla la ejecución de los programas, mediante el uso de la memoria, la unidad de disparo, EXQ y EDQ. El almacenamiento de memoria mantienen los registros FCT, TIT y PAT. EXQ y EDQ, que son registros generales FIFO de memoria.

Los ciclos de reloj empleados por cada unidad se muestran en la Tabla 12:

Tabla 12: Ciclos de reloj para la ejecución de cada unidad del controlador

Ciclos de reloj	Actividad
20	Ejecución de la primera plaza
7	Ejecución de la segunda plaza y las que siguen
13	Para la transiciones no sensibilizadas
2	Para trasferir al bus los datos de la transición
7	Sobrecarga de PE por el uso del controlador: 2 de lectura, 3 para disparo y 2 para escritura

Rendimiento básico del sistema

Se evaluó el rendimiento del sistema multiprocesador en el simulador de HDL. Controlando 62 programas en paralelo.

El tiempo de ejecución de cada programa, en cada lugar, es de 10 micro segundos (200 ciclos de reloj); que es 10 veces la sobrecarga a 20 MHZ.

La mejora de este sistema está en función proporcional al número de procesadores, que es de hasta 10 PES.

Aplicación computacional para el control del brazo robot

En general, los cálculos involucrados en el control en tiempo real del brazo robot es tan intenso que es nesesario el alto rendimiento de sistema multiprocesador, los que se han utilizado para este propósito. Este sistema determina la fuerza motriz y el par a cada articulación, está representado por 19 ecuaciones con el método Newton-Euler.

Análisis de Resultados

En este trabajo se implementó un sistema multiprocesador y se evaluó el rendimiento del sistema propuesto, aplicándolo al control en tiempo real de un brazo robot. Este sistema puede controlar procesos muy rápidos que se caracterizan por el tiempo de sólo 10 micro segundos.

Se ha implementado un sistema controlador con RdP pequeño que no permite evaluarlo en aplicaciones industriales reales. Tampoco se ha estudiado la expansión de la unidad de disparo para aumentar el número de lugares y transiciones. Los autores planean implementar un sistema de mayor tamaño y evaluar el rendimiento del sistema para aplicaciones reales.

Una mejora inmediata es proporcionar sub-net, que son una estructura similar a la llamada de subrutina en los programas. Es útil aplicar sistema jerárquico reales. Se ha propuesto mejorar el uso de recursos en la Unidad de Historia para distinguir el lugar de retorno. Los autores planean mejorar el sistema para proporcionar estructuras de sub redes.

A diferencia de lo indicado por el autor en este trabajo la ejecución toma varios ciclos de reloj, como lo podemos ver en la Tabla 12. Detectamos que los estados que hacen falta para resolver un disparo son más de 20 ciclos. El hecho de usar múltiples tablas impide una relación directa entre la expresión vectorial/matricial de la ecuación de estado de una RdP y la programación.

Las características que se implementan en [37] son mostradas en las Tabla 13 y Tabla 14.

Tabla 13: Características del tipo de redes implementadas en el PP, en el trabajo de [37]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[37]	No	No	Si	No	No	No	No	No	No	No	No	No	No

Tabla 14: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [37]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[37]	No	No	No	No	No	No	No	No

Reducción de la Memoria de la Unidad de Disparo del Controlador de RdP

En el trabajo de Murakoshi y Sugiyama [38], que es una continuidad del anterior, se propone un nuevo controlador de RdP, fácil de implementar en ASIC. Este se utiliza como controlador de un multiprocesador. Este controlador mejora la velocidad de procesamiento paralelo, puesto que es descrito por una RdP.

La responsabilidad de la unidad de disparo es verificar la condición para el inicio de los procesos, que se asignan a los lugares de la RdP. Esto se lleva a cabo en la memoria ASIC, cuya palabra es relativamente amplia, y se comprueba las transiciones sensibilizadas en dos ciclos de memoria. Para esto se requieren grandes campos de memoria.

La unidad de memoria se ha dividido según el tipo de disparos. Si existe un gran número de lugares en la RdP, esta se divide en subredes y se numera cada una haciendo uso de una tabla SubNetPlace. Los autores remarcan que con dos ciclos de memoria son suficientes para comprobar si la transición es sensibilizada o no, usando el mismo tiempo señalado en el controlador que presentara en el trabajo anterior. La unidad de disparo propuesta puede ser implementada por VLSI más fácilmente que la anterior, y puede realizar rápidamente el control.

Secciones de interés del trabajo

Motivación y fundamentos del trabajo

El controlador basado en RdP, que ha sido expuesto en el documento [57, 58], puede realizar procesos rápidos y paralelos descritos por RdP. La estructura de control del sistema es muy sencilla, ya que el control paralelo se implementa por el movimiento de los token en una RdP. Y el programa paralelo se especifica directamente por la RdP, no es necesario aplicar una técnica compleja para la exclusión mutua o de sincronización.

En este trabajo se propone la reducción de la memoria requerida en la Unidad de disparo. En la Unidad Disparo del primer trabajo, cada lugar la RdP requiere un valor.

En el controlador de multiprocesador con RdP, la unidad de disparo que se adoptó para implementar el control de RdP, está compuesta por una memoria ASIC, cuya palabra es muy amplia. La comprobación de las transiciones sensibilizadas se realiza en dos ciclos de memoria. Se requiere grandes campos de memoria (por ejemplo una red con 1024 lugares y 1024 transiciones requiere 2 x 1024 x 1024 = 2 Mbit memoria). Pero la estructura de datos de la Unidad de Disparo es pequeña y son demasiados los bits en la Unidad de Disparo que se utilizan ineficazmente.

Sin embargo, en la reducción de la memoria de la nueva unidad de disparo, los lugares se dividen en subprogramas llamados Sub-Net y son numerados individualmente para cada Sub-Net. Por el uso de esta numeración y un cuadro adicional, llamado Sub Tablero NetPlace, posibilita reducir la memoria de la Unidad de disparo de (lugares x transiciones) bits en (lugares x transiciones SobNet) bits.

Suponiendo que una RdP tiene 1024 lugares y 1024 transiciones, y una subred tiene menos de 64 plazas, podemos reducir en más de un 90% la memoria de la Unidad de disparo. Dos ciclos de memoria son suficientes para comprobar si la transición es sensibilizada o no, que es lo mismo que en el primer controlador. La unidad de disparo propuesto puede ser implementada por VLSI tan fácilmente como en el anterior.

La división de una gran RdP en una red principal y muchas sub-redes más pequeñas jerárquicamente, es de manera similar a un programa principal y subrutinas, como en lenguajes de programación ordinarios, facilitando la comprensión y el diseño. Para esto, se introduce un lugar llamado "Call- lugar", que permite llamar a un sub-red. También se proponen operaciones de " llamada a una sub -net " y "retornar de una sub -net". En este trabajo se presenta la arquitectura de un controlador de RdP que implementa estas características.

Cuando cualquier señal llega a un lugar final de una sub-red, una marca es devuelta al lugar que ha llamado a la sub-red. (Retorno de la sub -net ") y cualesquiera tokens que queden en la sub-red se deben quitar (“se mata a los lugares en la subred”). De lo contrario, algún token que queda en el antiguo sub-net alcanza el lugar final y puede surgir una marca equivocada en el lugar que había llamado a este. Así que es importante eliminar rápidamente todos los token que quedan en la sub-red.

Para llevar a cabo operaciones de alta velocidad, en las "llamadas de subred" y "retornos de sub - net", se propone una unidad de hardware, llamada Unidad de Historia. La función principal de la Unidad de Historia es implementada por una memoria accesible en dos dimensiones, por las de fila y por las columnas, como una memoria asociativa. La Unidad de Historia puede ejecutar operaciones de sub-red más simple y más rápido.

Generalidades del controlador de multiprocesadores con RdP

Es un programa, para especificar la RdP, que usa PNPL (Petri Net lenguaje de programación paralelo) como se propone en el documento[59]. Existen lugares para el inicio y fin de procesos que representan iniciar y finalizar el programa en PNLP. Los lugares Start y End no tienen procesos asignados. Cuando un lugar que está vinculado con un proceso tiene un token, entonces el proceso se ejecuta.

La estructura del controlador de multiprocesadores con “RdP Controlled”, está construido por los bloques: controlador, “comon data memory”, “job processor” y “comonn controler memory”. El controlador tiene una estructura de RdP en tres tablas y las siguientes marcas de la RdP:

- FCT (Tabla de condición de disparo)
- TTT (Token tranfer Table),
- PAT (Tabla de lugar de atributos)
- TST (Tabla de estado de Token) el marcado de la estructura de RdP

Los procesos tienen afinidad con cada plaza.

El Procesador de RdP se implementa como sigue:

1. Se detecta las transiciones sensibilizadas, mediante la comparación de TST y FCT.
2. El Controlador lee los lugares que obtendrán una token de TTT.
3. El Controlador lee atributos de las plazas de PAT, y si el atributo es normal escribir el número de lugar a EXQ (cola de ejecución) en la memoria de control común.
4. Un procesador de tareas obtiene el número de plaza de EXQ, y ejecuta el proceso de fijado en el lugar.
5. Cuando el Proceso termina el trabajo, escribir el número de un lugar a EDQ (cola final).
6. El controlador obtiene el número de su lugar, y actualiza la TST.

Este es el tipo más simple de controlador con RdP, las subred son como subrutina en lenguaje de programación convencional. También el controlador de RdP soporta multiprocesador.

Unidad de Disparo
La unidad de disparo es una memoria ASIC que realiza una rápida ejecución, compara TST y FCT. Está construida como la primera unidad de disparo, a la que se la ha agregado tablas. Las tablas de información y su función son los siguientes:

Tabla PlacePlaceID"

Si hay una plaza con ramas condicionales binaria, el controlador la trata como una plaza con una rama falsa y la otra plaza cono una rama verdadera, es decir como dos lugares diferentes. PlacePlaceID, esta tabla es para la conversión de la combinación de los lugares y su rama binaria de gráfica Petri Net a PlaceID. Son los números internos de identificación en el controlador. Las combinaciones de un lugar y su rama (0 o 1) son direcciones que serán leídas en la salida de la tabla PlaceIP.

PlaceID- Tabla Transición

Esta tabla contiene la conexión de arcos de PlaceID a las transiciones. PlaceID como dirección de la tabla lee una de las transiciones, es decir los arcos conectados a PlaceID. Puede ser que sean sensibilizados después que un lugar que corresponde a PlaceID haya sido terminado.

Tabla TransitionPlace

Esta tabla contiene las conexiones de arco desde la transición a los lugares. Número de transición como la dirección de la tabla lee uno de los lugares, es decir, los destinos de arcos conectados a la transición. Son lugares ejecutables provocados por la transición.

La unidad de disparo comprueba si una transición específica esta sensibilizada o no. La Unidad de disparo puede enviar señales en dos ciclos de memoria, después de un Tag PlaceID una transición se dispara. Se verifica para el disparo si todos lugares de entrada a la transición tienen tokens. Si la señal de disparo es "1" entonces la transición es sensibilizada.

La Unidad de disparo consiste en la tabla de estado Token (TST), Máscara, Kill Máscara y un registro temporal. TST muestra los lugares que tienen token, que significa que los procesos no han terminado (un token 1) implica hacer, mientras que un token (0) es no hacer. Su palabra es 1024 bits de ancho y cada bit corresponde a una PlaceID. La Máscara almacena todas las entradas PlaceID a la transición y se utiliza para verificar que todos los lugares de entrada tienen tokens.

La entrada 0 (1) de la máscara significa que una correspondiente a PlaceID este bit (no) es un lugar de entrada. Inhibe a las buffer de Máscara lugares de salida de Arc Kill de la transición y se utiliza para eliminar un token en el lugar que se descarta por Arc Kill. La entrada 1 (0) significa que un lugar que corresponde a este bit es borrado por Arc Kill. Ambas tablas son dos direcciones dimensionales, y su fila son de transición y sus direcciones de las columnas de PlaceID, siendo cada elemento de un bit de largo. A la palabra de las tablas se puede acceder en un ciclo.

Tipo Reducido de Memoria, Unidad de Disparos

En la primera unidad de disparo, cada lugar en la RdP tiene un número. Ahora un programa interactúa con las sub-redes de RdP, y también hay interacciones entre las subredes, salvo los lugares globales, y las máscaras de la matriz de inhibición que son organizadas por bloques divididos por cada Sub-Net.

En la unidad de memoria reducida, las plazas son numeradas individualmente para cada Sub-Net. Con el uso de esta numeración, se crean tablas adicionales Cakd y Sub-NetPlace que contienen el número de plazas individuales para cada subred, por lo que podemos reducir la memoria de la Unidad de disparos (Plazas x transiciones) en bits para la sub red (Plazas en una sub-red x transiciones). Sub-NetPlace es una tabla de conversión de un lugar y rama a PkceID sólo para los nuevos disparos de la Unidad. La operación para la Unidad de Reducción de Tipo de disparo de memoria no cambió. Y permanece igual que en el primer controlador.

Supongamos que una RdP de 1024 plazas y 1024 transiciones y una Sub-Net tienen LCSA de 64 lugares, y la Unidad de disparo propuesta requiere sólo el 64 kbit memoria. Esto demuestra que podemos reducir más del 90% de la memoria de la Unidad de disparo. Si un determinado Petri Net tiene k subredes, las que tienen más de 64 plazas, se necesario dividir la RdP en mas SubNetr para alcanzar el número de lugares.

La Regla de partición de la RdP se muestra como sigue.

1. Encuentra una parte de una transición a otra transición, que no tiene arcos de entrada de otras partes y no tiene arcos de salida a otras partes.
2. Vuelve a colocar la parte por una plaza de llamado.
3. Añade un lugar de inicio y uno de final como plaza creando una sub-red.

Si la RdP dada no tiene una parte que satisface la condición 1, selecciona un lugar de “hacer” como un lugar global para satisfacer la condición 1. Entonces la regla 2 y 3 se adoptan después de esta sustitución.

Análisis de Resultados

En este trabajo se propone la reducción de la memoria de la Unidad de disparos que está basada en características de la red.

Dos ciclos de memoria son suficientes para comprobar si la transición es sensibilizada o no, que no es lo mismo que hacer la comprobación y la comunicación con los procesos.

La unidad de disparo propuesta puede ser implementada por VLSI, tan simplemente como la anterior.

Vemos que para mantener los estados se requiere de tipificar las plazas, en algunos casos duplicarlas, muchas operaciones de decisión y tablas de direccionamiento.

Las conclusiones son similares que las del caso expuesto en [37], también vemos que se requieren de numerosos ciclos y tablas, estas últimas no tienen relación con las ecuaciones de estado para las RdP.

Las características que se implementan en [38] son mostradas en las Tabla 15 y Tabla 16.

Tabla 15: Características del tipo de redes implementadas en el PP, en el trabajo de [38]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[38]	No	No	Si	No	Si	No	No	No	No	No	No	No	Si

Tabla 16: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [38]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[38]	No	No	No	No	No	No	No	No

Implementación del Algoritmo de Disparo RdP en FPGA

Este documento [60] informa sobre el diseño de una aplicación para ejecutar algoritmos de servicios pesados por disparos de una RdP con un "Field Programmable Gate Array" (FPGA). Esto representa un componente importante en el desarrollo de un coprocesador integrado para la simulación de software basado en modelos de RdP.

Este trabajo, es la primera etapa de un proyecto para proporcionar un entorno de simulación para modelar el desempeño eficiente de software en tiempo real.

Debido a los tiempos de ejecución largos, que se encuentran típicamente en los simuladores de RdP, que ejecutan modelos complejos, el objetivo de esta investigación ha sido determinar la viabilidad de la utilización de hardware para acelerar la ejecución de la simulación.

Secciones de interés del trabajo

Motivación y fundamentos del trabajo
El desarrollo de software con arreglo a las estrictas especificaciones de rendimiento temporal requiere la generación, simulación y análisis de modelos de cómputo apropiados.

Este tipo de software es necesario para las redes de información, sistemas de control distribuido y sistemas de la industria aeroespacial/defensa. Con un enfoque de modelado de rendimiento del software. En un diseño, en particular, se pueden evaluar las especificaciones antes de comprometerse a la fase de codificación y pruebas. Esto es vital en los sistemas críticos de seguridad donde el rendimiento debe ser garantizado con un alto grado de certeza. En casi todos los casos, salvo el trivial, un amplio programa de pruebas no puede ofrecer este grado de certeza.

La actividad de diseño de software para este tipo de sistemas puede ser ayudado en gran medida por las herramientas de modelado de software que soportan explícitamente una especificación temporal y capacidad de verificación.

RdP y Arquitecturas de flujo de datos
Hay varios modelos computacionales que se pueden utilizar como base para la construcción de un modelo de software en tiempo real. Uno de los principales requisitos, además de explicitar la especificación de restricciones temporales, es que la concurrencia y sincronización también deben ser expresadas. Un paradigma de modelado que soporta todos estos requisitos son las redes Lugar/Transición (o RdP) [61]. La simulación de modelos de tipo RdP es un tema importante de investigación en el desarrollo de sistemas en tiempo real [62], y uno de los problemas que ha sido claramente identificados por muchos investigadores [63] [64] es la obtención de un entorno de simulación eficiente. Un componente central de la ejecución eficiente RdP es el algoritmo de disparo de transición.

En general, la simulación del modelo de un sistema de software complejo requiere recursos computacionales sustanciales y algunos investigadores están buscando maneras de incorporar el procesamiento paralelo en la estructura de estos simuladores. La base de este enfoque es que a medida que el sistema que se está modelando es invariablemente concurrente, y como el modelo conserva esta propiedad, entonces el entorno de simulación puede explotar esta propiedad. La mayoría de los enfoques para la simulación concurrente de modelos de tipo RdP son las redes distribuidas de estaciones de trabajo [64] [65] . Esto ofrece una ventaja inmediata la de ser capaz de utilizar componentes estándar para construir el sistema de simulación. Una desventaja importante es que todos estos componentes subyacentes (estaciones de trabajo) tienen procesadores con arquitectura de Von Neumann. En consecuencia, la concurrencia inherente en el modelo no se asigna de manera eficiente para una arquitectura distribuida.

Las RdP también se han aplicado en una matriz Transputer [66], donde los sistemas de memoria común no son necesarios, pero donde se requiere una interconexión altamente flexible. Procesadores y Transputer se apoyan en comunicaciones de alta velocidad y en hardware eficiente para cambios de contexto. Pero no dejan de ser una arquitectura de flujo de control RISC, y en última instancia, sufren la misma ineficiencia en esta solicitud como las arquitecturas de flujo de control CISC.

Hay una fuerte conexión entre los modelos de tipo RdP y el paradigma de la arquitectura de flujo de datos, ya que ambos funcionan de forma dinámica sin el uso de un contador de programa. También tienen ejecución guiada por la disponibilidad de datos token y no usan la memoria compartida. Efectivamente, el modelo de RdP representa un modelo de flujo de datos gráfico que se puede extender con información temporal.

El autor manifiesta en su hipótesis que si una arquitectura de flujo de datos está disponible para ejecutar el modelo tipo de RdP resultará una asignación eficiente de recursos. Esto permitiría que los modelos de sistemas de tiempo real complejos sean analizados con tiempos menores, que los obtenidos en este momento, o permitiría que los sistemas más complejos sean analizados en tiempos reducidos.

Objetivos del proyecto
El objetivo de este proyecto es explorar la implementación de una arquitectura de tipo de flujo de datos para apoyar la ejecución eficiente de los modelos tipo de RdP con tiempo. Este proyecto

forma parte de un proyecto más amplio para desarrollar un entorno de simulación basado en la RdP para sistemas de tiempo real reactivos complejos que deben cumplir con ciertos requisitos de desempeño [67]. La arquitectura propuesta debe simular el funcionamiento del sistema de software a través de la ejecución de un modelo de tipo RdP que representa el sistema en diferentes niveles de abstracción. También debe ser compatible con la realización de diversos algoritmos de análisis de los que se puede extraer información de desempeño estructural del modelo (utilizando los métodos tradicionales de accesibilidad y el análisis de los lenguajes, y el trabajo más reciente de Mínima representaciones algebraicas que permiten el desarrollo de los conceptos de espacio propio y estabilidad para estos sistemas [68]).

La arquitectura propuesta, junto con un procesador de interfaz de front-end, permitirá al diseñador interactuar con el modelo para explorar el rendimiento y límites del diseño. La representación lógica digital del algoritmo de disparo de RdP es sólo una parte de este proyecto; otras cuestiones que también deben abordarse, por ejemplo, son: ¿cuánto tiempo se tarda en programar un diseño en FPGA?, dado que un objetivo útil es reconfigurar dinámicamente la FPGA cada vez que el diseñador realiza un cambio en el patrón de interconexión de RdP. ¿Es esto posible, o si la FPGA sólo aplica alguna parte de la estructura de la RdP que permanece estático, y los parámetros para la ejecución pueden descargarse antes de la ejecución?

Implementaciones previas de RdP
Existen algunas aproximaciones en investigaciones previas para la ejecución de las RdP, en forma de hardware. Los primeros trabajos fueron hechos por Aumiaux [69], que migró las estructuras de los grafos de las RdP a arquitecturas lógicas digitales implementadas en EPLD y EPROM . El trabajo más importante, hasta ahora, ha sido hecho por Murakoshi [39] en el mapeo del algoritmo de disparo de RdP para un ASIC. Este trabajo fue seguido por una exploración de las cuestiones de cómo dividir una gran RdP para el mapeo de un ASIC con un número limitado de pines de E/S [40].

El trabajo se llevó a cabo, y se han producido avances significativos en la lógica digital programable, en particular, en “Field Programmable Gate Arrays” (FPGAs), donde los números de puerta por chip se están acercando a 100.000 con velocidades de reloj de más de 100 MHz. Esto ha creado la oportunidad para que las arquitecturas de computación reconfigurable y la implementación de hardware de numerosos algoritmos computacionalmente intensivos se puedan beneficiar de la aceleración de hardware flexible. Uno de los importantes beneficios de la tecnología FPGA frente a ASIC y la tecnología EPLD es la reprogramación inmediata de la lógica digital y sus patrones de interconexión.

En el contexto de la asignación de una descripción del modelo de RdP, la ventaja de la tecnología FPGA es que los patrones de interconexión inherentes en la descripción estructural de las RdP pueden ser potencialmente muy flexibles para ser implementados en la estructura de una FPGA. Debido a las densidades de compuerta en constante aumento cada vez hay más margen para instanciar RdP más grandes en un solo chip. Además, la creciente disponibilidad comercial de múltiples sistemas FPGA admite la asignación de RdP más grandes a sistemas FPGA. La principal aplicación de un tipo de RdP en FPGA sería como un coprocesador adjunto para soportar específicamente los sistemas de software de modelado y simulación de rendimiento.

Método de División de RdP
El enfoque adoptado por Murakoshi [40] para dividir una gran RdP antes de su implementación en el ASIC es identificar cuatro métodos distintos de seccionamiento de la red.

Esto depende de si se divide por lugares o transiciones, o arcos divididos a cada lado de lugares o transiciones. Esto lo llevó a la decisión de adoptar el corte a través de arcos. A continuación, demostró que este método es compatible con la transferencia de tokens de una subred a otra, y también que es compatible con el manejo de los token en conflictos entre las subredes para los dos casos de la división de los arcos antes de transiciones o lugares.

El enfoque adoptado en este trabajo está inicialmente influenciado por la necesidad de una asignación directa al FPGA. La RdP se secciona alrededor de cada transición, es decir, se crea una transición "módulo" local para cada transición en la red original con los lugares de entrada y salida. Muchos lugares aparecen en múltiples módulos de transición , por lo que la red no se secciona en áreas discretas cuando se recombinan o restituye la red original, como en el enfoque de Murakoshi [40]. La ventaja de este enfoque es que el funcionamiento de cualquier módulo de transición puede ser totalmente independiente de los demás, y la operación se determina localmente en base al número de token en las plazas y las multiplicidades de arcos. Esto requiere un análisis sencillo del acoplamiento de módulos de transición, que se puede hacer a partir de un examen de la matriz de incidencia de la RdP. El proceso de comprobación de los lugares comunes, y la gráfica de los lugares de salida de un módulo de transición en los lugares de entrada en otro módulo de transición, está actualmente a cargo de los programas que corren en el ordenador.

RdP Módulos de transición

Teniendo en cuenta el método de corte simple descrito en la sección anterior, cualquier RdP puede dividirse en módulo de transición extraída, el modelo RdP total del sistema puede ser representado como en la Figura 9.

Hay n lugares de entrada y salida de los lugares m, con los correspondientes recuentos de token en esos lugares que están marcados μ_{ln}, a través de μ_{in} para los lugares de entrada y a través μ_{on} fuente de los lugares de salida.

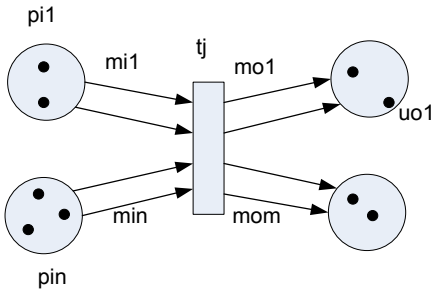


Figura 9: Módulo de RdP

Las multiplicidades de arcos de los lugares de entrada a la transición tj están etiquetados m_{i1} a m_{in} y las multiplicidades de arcos de la transición tj a los lugares de salida son etiquetadas m_{o1} a m_{om} través de fuentes.

Un solo módulo de transición RdP, que sigue las reglas de ejecución de una transición, se puede asignar a la operación de lógica digitales concurrente de dos bloques principales. El primer bloque tiene lugares 2n RAM de entrada, un restador y n posiciones RAM de salida; y el segundo bloque tiene posiciones RAM salida de 2m, un comparador y las ubicaciones de RAM m de salida. Los módulos primarios en la aplicación FPGA se muestran en la Figura 10.

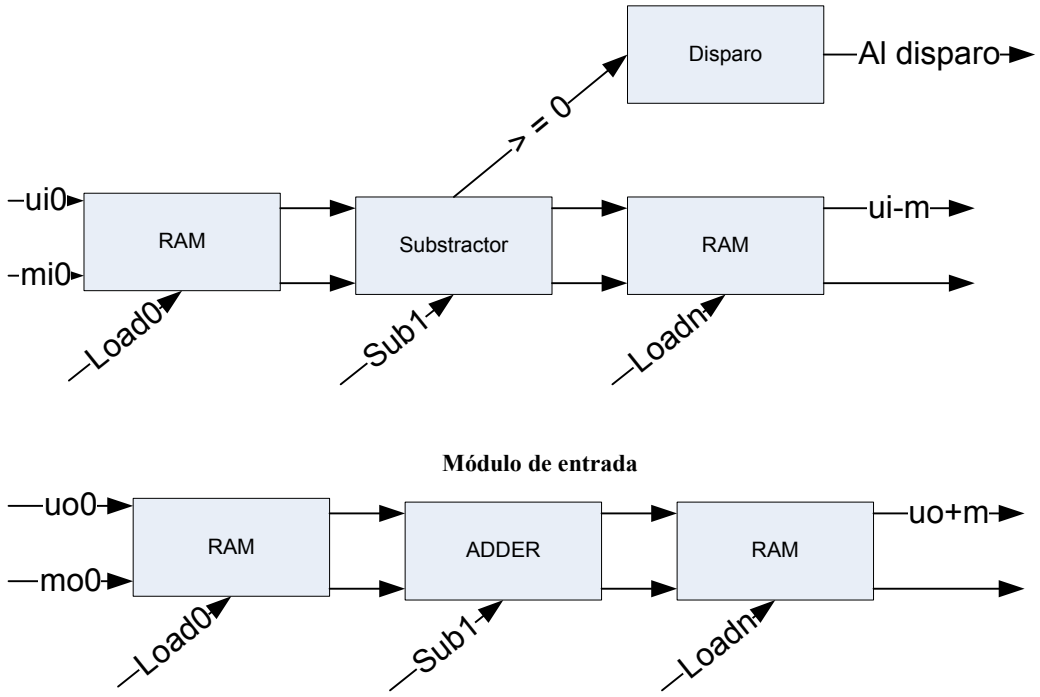


Figura 10: Representación lógica de los módulos en la FPGA correspondientes a las transiciones de la RdP

No se muestra en la Figura 10, la operación de la lógica de control.

Es de destacar que los módulos de entrada y de salida operan al mismo tiempo y la determinación en cuanto a si se debe utilizar su producción se hace en el momento por el software del ordenador principal (en función del estado de todos los disparos).

Conclusiones y trabajo futuro
Este autor ha sido capaz de demostrar la viabilidad de la utilización de un FPGA para ejecutar la simulación de algunas estructuras de RdP simples. Propone distintos temas para futuros trabajos, que aún no se han realizado en este proyecto, por ejemplo:

1. La ejecución del módulo de transición RdP a una velocidad de reloj más rápida.
2. Comparar el rendimiento de la aplicación FPGA con la ejecución de la misma RdP en arquitecturas convencionales.
3. Optimizar la aplicación mediante una mejor utilización de los módulos FPGA disponibles y mejorar la posición en el lugar y en el proceso de ruteo.
4. Optimizar el software del ordenador principal al cambiar algunos componentes adicionales para la implementación de FPGA (por ejemplo, las comprobaciones del módulo de concurrencia de transición).
5. Extender la aplicación actual para incluir la información temporal, para permitir la aceleración de hardware de tiempo de simulaciones de RdP.

Para ampliar más la propuesta, debería incluir marcas de tiempo en los token y arcos con retrasos de tiempo.

Análisis de Resultados
El estudio de la fundamentación y motivación del trabajo es de absoluta vigencia. Hay que destacar la observación que se hace con respecto a la conexión entre los modelos de RdP y el paradigma de la arquitectura de flujo de datos, ya que ambos funcionan de forma dinámica.

La división de RdP por transición repitiendo plazas implica que dos plazas en distintas subredes deben ser actualizadas al mismo tiempo, lo que agrega ciclos y hardware para escrituras simultáneas o escritura y lectura simultánea.

El trabajo no describe como realizar la interface entre los eventos y la ejecución de la RdP, puesto que se propone acelerar la simulación.

Las características que se implementan en [60] son mostradas en las Tabla 17 y Tabla 18.

Tabla 17: Características del tipo de redes implementadas por el PP, en el trabajo de [60]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[60]	Si	No	Si	No	Si	No	No	No	No	No	No	No	Si

Tabla 18: Características del tipo de eventos y programación implementadas por el PP, en el trabajo de [60]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[60]	No	No	No	No	No	No	No	No

Procesador Reconfigurable para la Simulación de RdP

Para la simulación de sistemas, de control de un gran número de objetos tales como los flujos de tráfico, el tráfico de mensajes de red, etc., la CPU puede requerir excesivas y largas corridas de procesos secuenciales convencionales. El artículo [70] describe al procesador y técnicas de reconfiguración para la programación a realizar para la simulaciones de RdP. Achilles es un pila de 3 dimensiones de FPGAs innovadora. El array 3D permite: (a) un gran número de FPGAs puedan caber en un pequeño volumen , (b) se obtiene un alto grado de flexibilidad en forma de dispositivos individuales, (c) la interconexión con uno o más hosts con gran ancho de banda es aceptable. Achilles está ampliando para satisfacer estos requisitos y (d) las pilas para cada cliente puede ser conectados entre una amplia variedad de servidores, de modo que la potencia de cálculo de la pila puede escalar según sea necesario. Los anchos de banda entre la pila y un anfitrión (PC) se han medido en más de 30 Mbytes / segundo en el primer prototipo de la pila. La interconexión es capaz de transferir datos a velocidades de bus PCI con los FPGAs más nuevas, utilizadas en el segundo prototipo en fase de construcción. Esta arquitectura es especialmente adecuada para las simulaciones de RdP en cientos de lugares que pueden estar activas simultáneamente, reduciendo en un orden de magnitud el tiempo necesario para las simulaciones.

Análisis de Resultados
Este trabajo aporta un sistema de comunicación de colas y no un controlador/procesador que ejecute RdP.

Controlador para Evitar Puntos Muertos en RdP por Hardware

En los trabajos previos de Murakoshi, propuso un controlador secuencial de alta velocidad con RdP. Su hardware es capaz de verificar si la transición esta sensibilizada o no, aunque no se toma en consideración los bloqueos. En este trabajo presentado por Murakoshi, Yasunori y otros [41] se ha incluido esta característica.

Si un controlador, descrito con una RdP tiene puntos muertos, lo que se hace es rediseñar nuevamente la RdP para el controlador.

Pero a veces se dificulta cambiar la red o insertar varios lugares para evitar puntos muertos, ya que no existen métodos generales para eliminarlos.

Por lo tanto, proponen aquí un hardware para evitar puntos muertos sin cambiar la RdP.

Para esto se ha incluido una unidad de hardware dentro del controlador RdP que detecta las marcas seguras y las que conducen a puntos muertos. Así se evita que la transición sea disparada. La unidad consta de tres tablas de memoria y algunas unidades lógicas para detectar la coincidencia de una corrida que lleva a una marca de punto muerto. Se requiere solo de un ciclo de memoria para comprobar uno solo estado.

Esta idea se desarrolla en este trabajo [41] y para evitar el marcado no deseado del objeto controlado.

Secciones de interés del trabajo

Motivación y fundamentos del trabajo

La automatización de fábricas requiere que los controladores programables sean muy rápidos, de fácil diseño y que sus posibles errores se puedan analizar y detectar con precisión durante el diseño, más aún, necesitan ser detectados durante las operaciones. Puesto que las RdP tienen los fundamentos teóricos, como el análisis del árbol de accesibilidad, son adecuadas para la descripción y el análisis de control secuencial.

El autor propone un controlador programable en los trabajos [42], basados en una RdP , que describe el estado de un sistema en su conjunto: el controlador y el objeto controlado . La unidad principal está compuesta de la unidad de disparo y la entrada de tramitación unitaria. La unidad de disparo es una memoria ASIC con una palabra amplia. La unidad de manejo de entrada es una estructura de árbol, cuyas hojas son de entrada y la raíz es una salida que escoge la entrada más rápida recibida por la regla de exclusión mutua. Se tarda sólo cuatro ciclos de memoria para procesar una señal de entrada. Por lo que es posible llevar a cabo un control más rápido que los sistemas convencionales, aunque no toma en consideración los puntos muertos.

Si una RdP tiene interbloqueos, es difícil de eliminar, por lo que se deben realizar cambios en la red o adicionar algunos lugares. El autor propone un hardware para evitar los interbloqueos sin cambiar la red, y no un método general.

Generalidades del módulo de detección de interbloqueo
El esquema propone:

1. Determinar los puntos muertos dentro de una RdP por el árbol de accesibilidad.

2. Determinar las marcas generales, la ramificación en el árbol de accesibilidad, unas ramas principales, las marcas posteriores acumuladas y las ramas que conducen a callejones sin salida.
3. Las nuevas tablas, en el controlador de RdP, contienen la información acerca de ramificación de los nodos.
4. El controlador comprueba si la marca actual es igual a una de las marcas que pertenecen a los nodos de ramificación. Si es así, la transición no se dispara.

El hardware adicional para comprobar los puntos muertos se compone de tres tablas de memoria y la lógica para comprobar la igualdad de las marcas. Se requiere de un ciclo de memoria para comprobar un solo estado.

Esta idea se puede utilizar para evitar las marcas no deseadas del objeto controlado

Análisis de Resultados

En este trabajo se ha propuesto dos tipos de hardware para evitar secuencia de puntos muertos en el controlador de RdP. Uno de ellos es para comprobar el marcado actual y su próxima transición. El hardware adicional para comprobar los puntos muertos se compone de tres tablas de memoria y lógica para comprobar la igualdad de las marcas. Si existe el número de interbloqueos, los lugares marcados con uno, es posible evitar puntos muertos sin demora. El otro caso es para comprobar el próximo marcado, asumiendo una transición disparada. El hardware adicional para comprobar los puntos muertos se compone de dos tablas de memoria, un registro y la lógica para comprobar la igualdad de las marcas. Este método solicita más de un ciclo de memoria.

Este trabajo pone en evidencia la necesidad de reprogramar el controlador rápidamente en las tareas de desarrollo y mantenimiento. Y la dificultad para determinar el disparo de la transición de más prioridad cuando las solicitudes de disparos se encuentran en una cola. Esto último implica que se requieren muchos ciclos de reloj para decidir el disparo de más prioridad.

Las características que se implementan en [41] son mostradas en las Tabla 19 y Tabla 20.

Tabla 19: Características del tipo de redes implementadas por el PP, en el trabajo de [41]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[41]	No	No	Si	No	Si	No	No	No	No	No	No	No	Si

Tabla 20: Características del tipo de eventos y programación implementadas por el PP, en el trabajo de [41]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[41]	No	No	No	No	No	No	No	Si

Controlador Programable de Alta Velocidad Basado en RdP

En este trabajo de Murakoshi, Sugiyama y otros [43], se propone un nuevo sistema de automatización basado en el uso de RdP para describir el estado de todo el sistema. El controlador programable propuesto consiste de la unidad de disparo y una unidad de manejo de entrada, como

la descripta en [37]. La unidad de disparo usa una memoria ASIC con una palabra ancha para comprobar si la transición esta sensibilizada o no, lo sé que realiza en dos ciclos de memoria. Desde la entrada, la unidad de manejo puede elegir la entrada de la señal haciendo uso de exclusión mutua, sin necesidad de analizar todas las entradas. Se tarda sólo cuatro ciclos de memoria. Con el sistema propuesto, para procesar señales de entrada, es posible realizar un control mucho más rápido, con respecto a sistemas convencionales con base matemática para el análisis.

Secciones de interés del trabajo

Motivación y fundamentos del trabajo

La automatización de fábricas requiere que los controladores programables sean muy rápidos, simples de diseñar y con facilidades para corregir y analizar los errores durante las operaciones. Las RdP tienen los fundamentos teóricos y son adecuadas para la descripción y el análisis del control secuencial. Se han propuesto varios controladores secuenciales basadas en ellas, como las mostradas en este estudio. Ellos usan la RdP en la descripción como diagrama de función secuencial (SFC), es decir, lugares o transiciones son asignados para controlar los procesos. También proponen este tipo de controlador programable en [42], que es mucho más rápido que cualquier otro controlador convencional.

Éstos pueden especificar sólo tareas de control, no un sistema en su conjunto. Con este método se puede describir un sistema en conjunto, que son: el controlador en sí mismo y de sus objetos controlados. Podremos utilizar los fundamentos matemáticos de RdP, tales como el análisis con árbol de accesibilidad, hacia adelante y hacia atrás, y realizar el análisis del sistema de control.

Aquí se propone un nuevo sistema de automatización basado en el uso de la RdP para describir el estado de un sistema en su conjunto. El controlador programable propuesto consiste en la unidad de disparo y la unidad de entrada de tramitación. La unidad de disparo se modificó a partir de la que se utiliza el controlador presentado en [43] . Y utilizó un micro procesador para procesar operaciones asignadas a los lugares pero el nuevo controlador. No a cualquier procesador, porque los lugares expresan el estado del controlador y no de los procesos. La Figura 11, muestra la unidad de disparo modificada, se trata de una memoria ASIC con palabras muy anchas para comprobar si la transición esta sensibilizada o no, en dos ciclos de memoria. La unidad de manejo de entrada tiene estructura de árbol, con muchas entradas en sus hojas y una salida en su raíz. Se puede elegir la señal entrada utilizando exclusión mutua. Mientras que el uso del controlador convencional demanda mucho tiempo en la exploración de las señales de entrada. Se tarda sólo dos ciclos de memoria para el sistema aquí propuesto, para procesar una señal de entrada de modo que es posible realizar un control mucho más rápido que el sistema convencional y tiene fundamento matemático analizable.

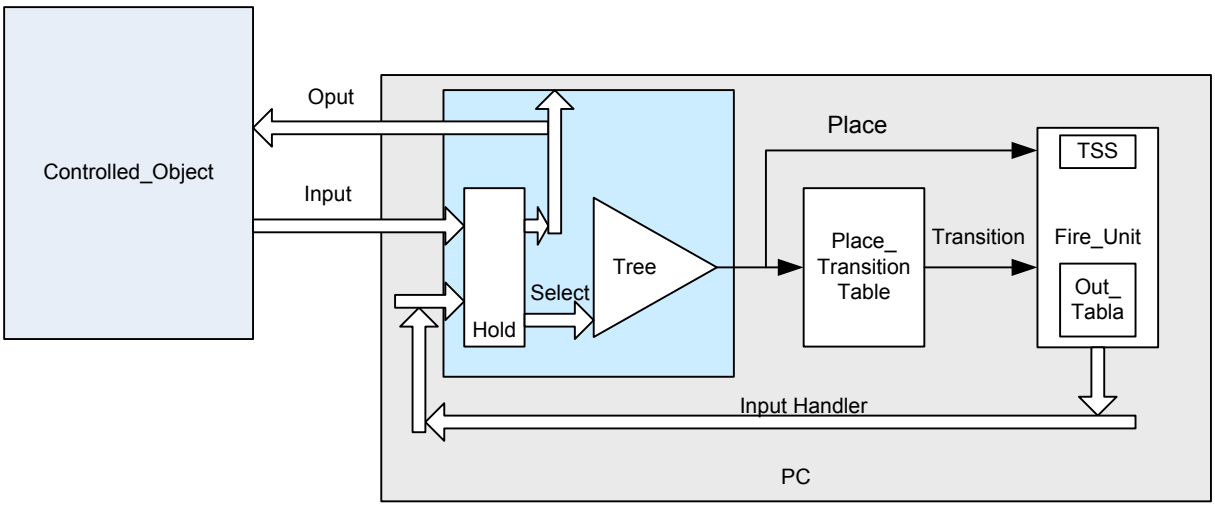


Figura 11: Vista del controlador programable

Análisis de Resultadoss

En este trabajo se propone un nuevo sistema de automatización basado en el uso de la RdP para describir el estado de un sistema en su conjunto. El controlador programable propuesto consiste en la unidad de disparo y la unidad de entrada. Y no se utiliza ningún procesador, porque los lugares expresan el estado del controlador, no procesos. La unidad de disparo es una memoria ASIC con palabra muy amplia, para comprobar si las transiciones están sensibilizadas o no, esto se realiza en dos ciclos de memoria. De las entradas, la unidad de entrada puede elegir la entrada de la señal que llego primero usando la regla de exclusión mutua, no es necesario escanear toda la entrada. Se tarda sólo cuatro ciclos de memoria para el sistema, recientemente propuesto, pueda procesar una señal de entrada.

Por lo tanto, es posible realizar un control mucho más rápido que el sistema convencional y tiene fundamento matemáticamente analizable.

En este trabajo se han disminuido la cantidad de ciclos necesarios para resolver un disparo, pero la manera en que se ha resuelto no permite implementar prioridades en los disparos, puesto que los eventos son resueltos en la secuencia que arriban.

Las características que se implementan en [43] son mostradas en las Tabla 21 y Tabla 22.

Tabla 21: Características del tipo de redes implementadas por el PP, en el trabajo de [43]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[43]	Si	No	Si	No	Si	No	No	No	No	No	No	No	No

Tabla 22: Características del tipo de eventos y programación implementadas por el PP, en el trabajo de [43]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[43]	No	No	No	No	No	No	No	No

Unidad de Control con RdP como Procesador Modular de Señal de Alta Velocidad

En el trabajo realizado por BROFFERIO [71], se propone una representación jerárquica de los algoritmos de procesamiento de señales digitales adecuadas para las implementaciones en tiempo real. Se utilizan modelos de RdP para demostrar todas las posibles operaciones paralelas usando la gráfica de RdP marcada. Por otra parte, se presenta un algoritmo de control de ejecución jerárquico basado en gráficos de Petri retardado.

La arquitectura es estrictamente modular adecuándose al sistema y facilitando la implementación con VLSI. El procesamiento de datos es realizado por sus principales componentes. La representación algorítmica se aplica para el diseño del módulo de control de los componentes del sistema. Más detalles, de los controladores, en el nivel lógico para una amplia gama de procesadores de señales digitales se presentan como una aplicación de la metodología propuesta.

Secciones de interés del trabajo

Los procesadores de señal en tiempo real son implementados eficientemente cuando las limitaciones impuestas por el algoritmo y las acciones que se requieran se corresponden con la tecnológica. Las características de las señales (algoritmos, precisión y secuencialidad) dictan los límites de tiempo de procesamiento. Mientras que las características tecnológicas (espacio, tiempo y energía), junto con el posible grado de concurrencia de los algoritmos sugieren las arquitecturas.

La funcionalidad, la arquitectura y la estructura de procesamiento paralelo son ampliamente presentados en la literatura, una recopilación completa se dá en [72], las arquitecturas de flujo de datos son presentado en “H. T. Kung” [73] y “ S. S. Patil and T. A. Welch” [74], las arquitecturas para el procesamiento de señales se encuentran en “H. T. Kung” [75] , mientras que las cuestiones de aplicación se tratan en “B. I. Dervisoglu” [76],” D. I. Maldovan” [77] y “S. S. Patil and T. A. Welch”[74].

En este trabajo [71] se describen las partes de control de un sistema modular basado en datos, compuesto por módulos de cálculo y de comunicación, y se extiende con la teoría de grafos de las RdP . Una representación del algoritmo adecuado para el control de ejecución en los diferentes niveles operativos, basado en RdP, también es expuesta. En la organización de control jerárquico los elementos básicos de la arquitectura del controlador son basados en gráficos de Petri. Los mismos se introducen, junto con la estructura lógica de las partes principales del controlador, en un sistema modular basado en procesadores de señal digital. Y por último, se expone un ejemplo sencillo de la metodología propuesta.

Análisis de Resultados

Este trabajo propone una representación algorítmica para deducir la estructura del controlador de procesadores de señales digitales de alta velocidad. Se basa en las RdP porque busca una representación explícita de concurrencia del algoritmo. La misma también se puede utilizar como un tutorial para representar el comportamiento del flujo de datos en los equipos.

El controlador se compone de dos partes: una dependiente en el algoritmo y la otra dependiente de la arquitectura del módulo. La arquitectura y la estructura del manejo de la ejecución del algoritmo son la parte concurrente del controlador, que el autor describe detalladamente.Una arquitectura integrada de todo el controlador podría ser de interés para sistemas VLSI.

En la actualidad, un controlador experimental para aplicaciones de voz utilizando un procesador de señal digital TMS 320 se implementa a través de arreglos de compuertas.

La simulación del sistema es necesaria para evaluar las prestaciones de las aplicaciones específicas, especialmente en los casos en los módulos de procesamiento. Es muy usada para la conmutación de datos entre los módulos, que son requeridos por la localidad de comunicación de los sistemas VLSI. Programas de simulación de eventos utilizando la representación jerárquica propuestas (basado en grafos de Petri extendidos) se podrían diseñar.

Es un proyecto insipiente que usa la RdP para dirigir el flujo de datos, proponiendo la construcción de un sistema en VLSI.

Las características que se implementan en [71] son mostradas en las Tabla 23 y Tabla 24

Tabla 23: Características del tipo de redes implementadas por el PP, en el trabajo de [71]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[71]	Si	No	No	No	No	No	No	No	No	No	No	No	No

Tabla 24 Características del tipo de eventos y programación implementadas por el PP, en el trabajo de [71]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[71]	No	No	No	No	No	No	No	No

Del modelo a un controlador de integración de animación gráfica en FPGA, usando generación automática de código

En este trabajo [78], los autores tienen como objetivo generar automáticamente código VHDL, para aplicación en control de sistemas embebidos, susceptibles de ser instanciadas en una plataforma integrada. Estos controladores se llevarán a cabo en plataformas reconfigurables FPGA, incorporando interfaces gráficas dedicadas e interfaces de entrada/salida que permiten su conexión física con el proceso bajo control. El comportamiento del sistema, del controlador, se modela utilizando RdP IOPT.

Una herramienta llamada "Animator4FPGA" fue desarrollada, para alcanzar este objetivo en cooperación con otras herramientas desarrolladas dentro del proyecto FORDESIGN. A saber: editor gráfico de RdP IOPT, herramienta Animador, y una herramienta de generación automática de código a partir de las RdP en el código VHDL.

Secciones de interés del trabajo

La actual evolución del hardware permite incrementar la complejidad de los sistemas electrónicos, lo que dá respuesta a un mayor número de requerimientos. Este aumento de la complejidad de sistemas conduce a las nuevas exigencias en términos de modelado.

Tener un modelo que describe adecuadamente el sistema es un valor añadido para su desarrollo y para su documentación.

Además, al ser posible la generación automática de código a partir del modelo, el tiempo empleado en el desarrollo será menor, por la reducción de errores frecuentes, aunque algunos aspectos de optimización del código producido pueden no estar totalmente considerados.

Proyecto FORDESIGN [79] (que tiene como objetivo principal trabajar con RdP en el área de diseño de sistemas embebidos) responde a estas cuestiones. La principal intención de este trabajo fue el desarrollo de un entorno para la generación automática de hardware descritos en VHDL, que se utilizará en la configuración de plataformas reconfigurables basados FPGA, la incorporación de interfaces gráficas dedicados y las interfaces de entrada/salida.

El objetivo es controlar los procesos, con funciones similares a los de un PLC (Controlador Lógico Programable), utilizando una plataforma reconfigurable.

El proceso de desarrollo se inicia mediante la construcción de modelo de comportamiento del sistema de control con el fin de soportar un enfoque de desarrollo basado en el modelo. Después de eso, las herramientas de automatización de diseño se utilizan para apoyar la generación automática de código y las operaciones de verificación.

También fue un objetivo explorar la asociación / relación entre las características del comportamiento del sistema modelado con las características de la sinóptica asociada, y la animación sinóptica (una vez más, para permitir la automatización de desarrollo de procesos a través de la generación automática de código de ejecución). Esta asociación se expresa a través de un conjunto de reglas, que evolucionan y se implementan según las aplicaciones generadas.

El formalismo utilizado para el comportamiento del modelado del sistema controlador fue RdP [80], que permite la asociación de sus características estáticas (marcado de la RdP) y sus características dinámicas (disparo de las transiciones) a las características gráficas de la sinóptica deseada.

La herramienta informática desarrollada interactúa con otras herramientas desarrolladas en el marco del proyecto FORDESIGN. Esto permite la generación automática de código a partir de modelos de comportamiento. La generación de código VHDL haciendo uso de RdP ha recibido atención durante las últimas décadas [81-83]. Pero los autores no fueron capaces de encontrar una herramienta de generación automática de código VHDL. Otra manera que genera el código VHDL a partir del modelos de RdP plaza/transición se encuentra en de [44], que es el utilizado en este trabajo.

Para validar el prototipo desarrollado se utilizaron los casos específicos de los sistemas embebidos modelados con PN.

Se considera que este trabajo proporciona un valor añadido, dada su novedad y la falta de herramientas para la generación automática de código, que cumple con los objetivos propuestos por el autor.

Sólo se encontraron dos herramientas que tienen objetivos similares, la suite “BRITNEY” [84] y “Synoptic” [85, 86] construido por “Animator”. La característica innovadora de estos trabajos es que tienen tareas de animación para RdP, pero ninguno de ellos para plataformas reconfigurables basados FPGA.

Análisis de Resultados

La principal conclusión de los autores sobre el trabajo presentado, es la implementación con éxito de un entorno innovador para el desarrollo de sistemas de control embebido basado en FPGA, integrados con una interfaz gráfica (que permite la presentación del sinóptico asociado). El marco

de desarrollo integra herramientas de "automatización del diseño", la combinación de desarrollos basados en el modelo con herramientas de generación automática de código.

Las RdP IOPT han demostrado ser adecuadas como modelo de control de sistemas, permitiendo a la aplicación la generación automática de código.

Otra conclusión importante está relacionada con la reducción del tiempo de desarrollo de los controladores. Como cuestión de hecho, con el entorno de desarrollo descrito, la creación del controlador y la visualización de la instalación del estado de un determinado proceso se pueden hacer muy rápidamente, ya que todas las tareas de desarrollo de código fueron reemplazadas por el uso de interfaces específicas para su caracterización.

La metodología de desarrollo del controlador propuesto evita errores resultantes de la generación manual de código, de este modo se reducen los errores en el desarrollo de la etapa de modelado.

La herramienta Animator4FPGA, el ejemplo mostrado, así como otros ejemplos se pueden encontrar en [79].

Sin embargo la falta de tareas de programación en plataformas reconfigurables basados FPGA es su principal debilidad.

Este trabajo [78], es un aplicación del trabajo realizado en [44], y muestra como dirigir, implementar y animar un desarrollo dirigido por un modelo RdP IOPT.

También, se hacen comparaciones con otros desarrollos similares.

Las características que se implementan son las mostradas en las Tabla 25, Tabla 26 y Tabla 27.

Tabla 25: Aportes de[78] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[78]	No	No	Si	Si	Si	No	No

Tabla 26: Aportes de [78] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[78]	No	No	Si	Si	No	No

Tabla 27: Aportes de [78] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[78]	No	Si	Si	Si	Si	No	Si	Si	No	Si

Diseño de Controlador Lógico con Enfoque Jerárquico para Aplicaciones Específicas

En este artículo [87] se presenta un método de síntesis estructurado sobre la base de las RdP jerárquicas. El marco de diseño implementado contiene la síntesis de lógica programable de descripciones basadas en reglas que se obtienen de varios modelos de especificación de controladores concurrentes, especialmente en el control interpretado, con formato de RdP. La especificación en la forma de reglas de decisión condicional simbólicos se transforma en un formato que es aceptado por los simuladores y sintetizadores de FPGA estándar, tales como los de la serie Xilinx. El modelo de la máquina de estado concurrente de controlador lógico se verifica por medio de la teoría de RdP, y luego se traduce a través de procesos automatizados en formato de especificación FPGA seleccionado, por ejemplo formato netlist Xilinx (XNF).

La Técnica propuesta se ilustra mediante la presentación de una solución a un "problema" popular de una estación de perforación. Los métodos son especialmente útiles en el diseño de un controlador industrial específico, aplicando lógica (ASIC) con FPGA.

Secciones de interés del trabajo

El objetivo principal de este trabajo es la presentación de nuevos métodos jerárquicos de asignación directa para Controlador Lógico Programas (LC), en Field Programmable Gate Arrays (FPGAs). En una de las posibles metodologías, se aplica un formato de lógica basado en normas de especificación del comportamiento y mapeo directo de una RdP equivalente a la biblioteca de hardware por medio del sistema de Xilinx. Un equivalente textual de RdP, llamada “RdP Especificación Formato 2” (PNSFZ), se usa como un formato de entrada. Por otra parte, la RdP se puede modelar también alternativamente utilizando lenguajes conocidos de descripción de hardware (HDL), en particular con Verilog o VHDL [88] . Los resultados experimentales muestran que el método propuesto puede producir ahorro, es estructurado y flexible. La Matriz Celular Lógica (LCA) facilita la implementaciones del controladores lógicos dedicado [89] [90].

La traducción de la RdP en LCA es simple. En general las reglas de decisión “If-Then” (declaraciones condicionales no procesales) pueden ser mapeadas en formato Xilinx directamente, o después de algunas sencillas transformaciones adicionales. Finalmente las combinatorias optimistas de múltiples niveles, la colocación y enrutamiento se realizan mediante herramientas de diseño estándar.

RdP como especificaciones del controlador lógico

Un control de RdP coloreadas interpretado, como herramienta gráfica, proporciona una metodología de diseño unificado para la especificación de sistemas de eventos discretos. Pueden ser aplicados en diversas etapas de la implementación del diseño de la descripción del sistema jerárquico para su realización física. Una RdP se utiliza como una herramienta para el modelado y análisis de circuitos digitales, especialmente los controladores concurrentes.

Se han propuesto varias técnicas de especificación y diseño con RdP. Se basan en las herramientas de software que ayudan a los diseñadores a desarrollar y simular (animado) el sistema en un nivel conceptual y abstracto. Por lo general son muy formales y orientadas a la verificación (simulación) de aspectos del diseño. Uno de los más poderosos sistemas de diseño es CPN-Tool, que se dedica a las RdP coloreadas (CPN) [91]. El diseño de CPN se basa en tres métodos de análisis de las CPN: simulación gráfica, ocurrencia e invariantes de lugar y transición.

El primer método, de simulación, es similar a la depuración de la ejecución del programa. Durante la simulación es posible comprobar si un circuito modelado por una RdP se comporta correctamente. Para este fin, un diseñador puede utilizar diferentes mecanismos que han sido previstos en DesigdCPN, tales como los puntos de interrupción. Por lo tanto, es posible reconocer y eliminar los errores en los controladores, incluso en las primeras etapas de diseño.

El segundo método de análisis se basa en un gráfico de ocurrencia (gráfico de modos de alcanzabilidad). Los gráficos se crean automáticamente. La RdP puede ser jerárquica o plana, sin embargo, el gráfico siempre se genera para la red aplanada. Los resultados del análisis se escriben en un archivo. El informe contiene información sobre las propiedades esenciales de los controladores lógicos que describe la red, como vivacidad, marcas muertas y transiciones, inanición, equidad, etc.

El tercer método trata los invariantes de lugar y transición. El usuario construye un conjunto de ecuaciones que son satisfechas por todos los estados alcanzables del sistema. Las ecuaciones se utilizan para comprobar las propiedades de la RdP, por ejemplo, ausencia de puntos muertos.

El controlador de CPN interpretadas (CCIPN) introducida en [88] es similar a la red de alto nivel con token de colores, introducida por Jensen [92]. Aparte de esto, este tipo de red tiene la ventaja de las formas muy usadas en ingeniería, por ejemplo, control de RdP o Grafcet.

Un controlador de CCIPN, es un control que interpreta RdP con token de colores. Es un controlador orientado al control introducido en [93] . Dado que no más de un token, con color especial, puede siempre estar en cualquier lugar, la red debe ser limitada y segura, en relación con cualquier color. La interpretación se aplica a: token de colores, las entradas y salidas externas de color. La descripción de CCIPN es más intuitiva, que otro modelo de CPN similares.

Análisis de Resultados
El autor entiende que el trabajo presentado es eficaz cuando las RdP son sólo modelos pequeños. Mientras que, en circunstancias de grandes redes el enfoque es ineficaz. Pero no evalua otos enfoques como introducir RdP jerarquica u otro tipo de RdP más adecuadas.

Diseño basado en Lógica Programable con Rdp

En esta sección, el autor presenta, la especificación del controlador de la estación de perforación [94], para luego relacionarlo con el control de RdP coloreada interpretada (CCIPN) [88]. En el ejemplo, varias operaciones pueden ocurrir simultáneamente. Una pieza de trabajo se carga en una estación y se perfora, finalmente, en la tercera estación se prueba la profundidad. Algunas acciones pueden ocurrir independientemente de las demás, mientras que otras acciones pueden tener lugar después de que todas las estaciones han completado sus programas individuales.

Enfoque del trabajo
En este trabajo se presenta un enfoque jerárquico para el modelado y diseño de controladores de lógica discreta, basados en las RdP y máquinas de estado concurrente. Combinando los principios de ambas logra el refinamiento de las RdP con posibilidad de mapeo jerárquico de la red en dispositivos de lógica reconfigurable.

El método propuesto tiene un poder sustancial de síntesis, que es suficiente para la introducción de un diseño digital formal sobre un área de desarrollo de aplicaciones dedicadas. Como pueden ser los controladores lógicos específicos, implementados con FPGA actuales. La red basada en la

metodología de diseño de Petri permite el desarrollo jerárquico de la especificación y el diseño, obteniendo una transformación en diferentes niveles de abstracción.

La compatibilidad sintáctica y semántica de las descripciones de RdP y reglas de decisión con netlist estructurada, basada en la red de controladores lógicos, considera la jerarquía explícita incluida en formatos de listas de conexiones para la implementación jerárquica como la más cercana posible. Las transformaciones lógicas formales mapea algoritmos en ecuaciones booleanas finales especificadas, y luego en FPGAs.

Análisis de Resultados
Como se observa en este trabajo las RdP son usadas para el modelado y luego se realiza un traslación (compilación) del modelo a la FPGA.

Aquí lo que se ha realizado, es una biblioteca de componentes con la que se arma un modelo específico. Pero no se muestra como se ejecuta el modelo o si las distintas características del sistema controlado por RdP es ejecutado (nunca menciona RdP no autónomas).

Es un proyecto que usa la CPN, restringiendo fuertemente el tipo, para dirigir la ejecución. Desde el punto de vista de los componentes, para la composición, el aporte es importante. El problema de prioridades complica la implementación, puesto que hay que centralizarlo.

Las características que se implementan en [87] son mostradas en las Tabla 28 y Tabla 29

Tabla 28: Características del tipo de redes implementadas en el PP, en el trabajo de [88]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[87]	Si	No	No	No	No	No	No	No	No	No	No	No	No

Tabla 29: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [87]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[87]	No	No	No	No	No	No	No	No

De un Modelo con RdP a una Implementación de un Controlador Digital con VHDL

El trabajo realizado en [44], se basa en el modelado de sistemas de eventos discretos utilizando modelos de RdP ya estudiados. Tampoco es nuevo el uso de las especificaciones de controladores digitales. Sin embargo, reconoce que existe una falta de herramientas para la generación automática de código. El artículo presenta brevemente el proyecto FORDESIGN que pretende desarrollar un conjunto de herramientas para contribuir a llenar este vacío dentro de los sistemas de automatización y (en red) para el desarrollo de sistemas embebidos. El conjunto de herramientas se basa en una clase de RdP, nombrado como Input-Output Petri Net (IOPT), y su representación mediante la RdP mediante Markup Language (PNML). El artículo presenta algunas reglas utilizadas por un generador de código automático capaz de producir código VHDL a partir de la representación PNML del modelo de RdP.

Secciones de interés del trabajo
Es común que en los sistemas de modelado integrados se use un enfoque de descomposición, basado en un controlador que interactúe con su entorno. Para la implementación del controlador se ha usado una señal de sincronización (un reloj externo global), obligando que esta interacción suceda en puntos periódicos en el tiempo. Esto es generado por el controlador, que es visto como un sistema de eventos discretos, modelado adecuadamente por las RdP.

Las RdP pueden ser vistas como una generalización de las máquinas de estado: las transiciones pueden originar más de un estado activo, de manera de modelar el paralelismo, donde varios estados necesitan estar activos para permitir una transición con el fin de modelar la sincronización.

Además, las RdP son un modelo intrínsecamente concurrente puesto que varias transiciones habilitadas pueden dispararse y cualquier número de estados pueden estar activos al mismo tiempo. Las RdP también tienen una representación gráfica sencilla, como: entes pasivos (por ejemplo, estados o de recursos), lugares nombrados, entidades activas (por ejemplo, acciones, funciones) y transiciones con nombre.

Las siguientes características de las RdP se refieren comúnmente como adecuadas para soportar el modelado de los controladores dentro de los sistemas de automatización y (en red) para el desarrollo de sistemas integrados:

1) Una sintaxis y semántica precisa;
2) El modelado explícito y legible para la concurrencia y sincronización;
3) El modelado explícito de la asignación y el consumo de recursos;
4) La visualización gráfica de los modelos;
5) Soporte para el diseño de abajo hacia arriba (composición) y de arriba hacia abajo (refinamiento), que son cercanas;
6) Posible soporte para el diseño que incluye en el modelado al factor tiempo.

Sin embargo, existe una clara falta de herramientas basadas en las RdP, que soporten el desarrollo de esos controladores. Este es el lema del proyecto FORDESIGN [45] cuyo objetivo principal es el desarrollo de un conjunto práctico y completo de herramientas, que, tomadas en conjunto, son capaces de ofrecer un entorno de desarrollo cómodo y completo basado RdP, a partir del nivel de especificación (usando herramientas de edición dedicadas), y terminando en la aplicación (donde FPGAs son las plataformas de ejecución preferidas), mientras que la integración de herramientas se realizada en la generación automática de código.

El autor presenta en su trabajo una breve visión general del proyecto FORDESIGN, una visión general del conjunto de herramientas en desarrollo y sus relaciones. Seguidamente, introduce la clase de redes IOPT y describe una gramática para permitir la representación de redes IOPT en PNML. Finalmente, presenta las principales estrategias para la generación automática de código en VHDL y su conclusión.

Visión general del proyecto
El proyecto FORDESIGN [45] [46] integra el uso de las RdP en un ciclo de vida de desarrollo para sistemas embebidos. Sin embargo, las RdP no son el lenguaje obligatorio para el desarrollo o modelado. En lugar de ello, las RdP se utilizan como un lenguaje base al que los modelos, expresados en otros lenguajes, pueden ser traducidos. Estos otros lenguajes incluyen las siguientes características:

- Diagramas de estado;
- Gráfica del diagrama de estados;
- Diagramas de secuencia en UML;
- Máquinas de estados finitos jerárquicas y concurrentes;
- Otras clases de RdP.

El proyecto se descompone en cuatro tareas principales [45], a saber:

- Tarea 1 – Seleccionar los modelos de computación para las RdP;
- Tarea 2 - Composición y representaciones jerárquicas con las especificaciones basadas en RdP;
- Tarea 3 – Partición en distintos modelos de computación;
- Tarea 4 - RdP para codificar a través de generadores de código automáticos;

Un aspecto importante es el uso del formato de intercambio PNML como una manera de maximizar la interoperabilidad con herramientas ya disponibles, herramientas de verificación de todos los modelo, y entre las herramientas en desarrollo.

La primera tarea supone un paso previo de análisis, donde se recogen los requerimientos mediante casos de uso. A partir de estos casos de uso, el modelador elige el formalismo más adecuado, a saber, el lenguaje más adecuado según el comportamiento del sistema. Los distintos lenguajes se unifican a través de la traducciones a una clase única de RdP no autónomos que actúan como un lenguaje base y universal en la fase de diseño.

En la fase de implementación, todos los formalismos de comportamiento se convierten a modelos que utilizan esta clase de RdP, que, después de la verificación de propiedad y la eventual partición en componentes, se tradujo a C o a la descripción de hardware en VHDL.

El proyecto hace hincapié en el uso pragmático de las RdPnA y las nuevas formas de integrar y especificar la composición del modelo. Basándose en la definición de un conjunto de operaciones de red, incluyendo suma neta, el soporte de arriba hacia abajo, de abajo hacia arriba y la composición transversal de modelos.

Otro foco principal, es la identificación de métodos para la partición de modelo que permite la ejecución distribuida y generación de componentes para ser identificados en las plataformas de hardware o software, de acuerdo a los requerimientos específicos de rendimiento y costo.

Características de la herramienta
El FORDESIGN enfatiza el uso de herramientas para desarrollo, sean existentes o nuevas. El primer grupo incluye herramientas UML. Estos serán utilizados, principalmente, en la fase de requerimiento y análisis. En particular, se prevé el uso de diagramas de casos de uso. En el segundo grupo, el proyecto incluye el desarrollo de varias herramientas interrelacionadas. Incluyen los siguientes cuatro grupos:

1. Modelado: El editor gráfico que incluye soporte de abajo hacia arriba y de arriba hacia abajo en la construcción del modelo; soporte para la resolución de conflictos para la generación y la reconfiguración;
2. Simulación: El simulador acepta señales de entrada externa y permite la comprobación de las señales de salida resultantes;

3. Verificación: Genera el espacio de estado para PNML2S, y un analizador que genera código C;
4. Síntesis: Traductores de PNML a C, y desde PNML a VHDL. Presentación de Producto para adaptar el código generado a la plataforma.

Esta división de funcionalidades se realiza a través de varias herramientas y tiene dos objetivos complementarios: (1) que permitan el desarrollo y mantenimiento independiente; (2) maximizar las futuras sinergias entre estas y otras herramientas.

En el trabajo se muestra las interdependencias y la comunicación entre los diferentes componentes previstos.

En particular, con el editor gráfico, el modelador sería capaz de ejecutar las siguientes tareas:

1) Especificaciones lectura y escritura de PNML basados [95] [96];
2) Creación de modelos PNML y OPNML [97];
3) Creación estructurada de los diagramas de RdP;
4) La resolución de conflictos;
5) La generación de código, para la ejecución, simulación y verificación de una base de código común;
6) Exploración espacio de estado.

Los generadores de código serían capaces de optimizar el código generado, después de una ejecución inicial, a saber, por el consumo de memoria optimizado basado en límites superiores para el lugar marcas, obtenidos a través de la simulación. Este código podría ser ejecutado en varias plataformas donde el lenguaje C esté disponible. Sin embargo, el foco principal serían las plataformas reconfigurable con FPGAs. Además, las soluciones serán completamente compatibles con el diseño de SoC. La herramienta de configuración permite la especificación de varios detalles para la generación de código optimizado según la plataforma dada.

El objetivo es la obtención de varios sub-modelos paralelos que pueden ser implementados por componentes distintos y separados en software o hardware.

Por último, el objetivo último es la generación automática de código ejecutable para plataformas distintas, a saber soluciones FPGAs y System-on-Chip, donde varios componentes puedan cooperar, integrar software y soluciones de hardware como diferentes componentes.

Este artículo destaca varias herramientas que han sido previstas, que serán utilizadas en las fases de diseño e implementación.

Representación de las RdP IOPT
Como se indicó en el apartado anterior, el controlador se caracteriza por dos componentes principales:

- Descripción de la interacción física con el sistema controlado (la interfaz del controlador);
- Descripción del modelo de comportamiento, que se expresa a través de un modelo IOPT.

La estructura de bloques, de la implementación del controlador, posee interfaces de entrada y salida que manejan la interconexión con el mundo exterior, y el " bloque de ejecución" donde es responsable de la ejecución del modelo IOPT. Es importante destacar que, dependiendo del tipo de plataforma de implementación que está destinado a ser utilizado, estos bloques pueden ser

codificados en diferentes lenguajes que son soportados por una herramienta, la generación automática del código (actualmente C y VHDL están disponibles, aunque sólo VHDL se analiza en el resto de este trabajo) y ajustado a una plataforma específica.

En los párrafos siguientes se describe brevemente, “Relax NG”, una gramática [98] basada en la definición de RdP, plaza transición [99] . Información adicional se puede encontrar en [80].

Un archivo PNML será generado, de acuerdo con la gramática definida, con el fin de coordinar toda la información asociada con las características del controlador.

Además de la definición de distintos atributos, de los tres conceptos principales de PNML, que son los lugares, transiciones y arcos. La gramática define cuatro grupos adicionales de anotaciones de atributos, que son: “NetIOInput”, “NetIOOutput”, “Conf ictSet” y “SynchronySet”.

La sección de NetIOInput incluye las señales de entrada y eventos, y la NetlOOutput tiene una estructura idéntica para las señales y eventos de salida.

Actualmente, se definen dos tipos de señales: booleana y rango. El tipo booleano se explica por sí, mientras que el tipo rango es compatible con el tipo enumerado y la definición de valores a través de un intervalo, teniendo en cuenta los límites mínimo y máximo.

Eventos de ambos tipos (entrada y salida) tienen un identificador y una señal asociada. Además, para los eventos de entrada es obligatorio especificar el flanco de la señal, como ascendente o descendente.

Opcionalmente, es posible especificar un nivel de umbral para la generación de eventos. Esto es significativo para los eventos que dependen de señales de tipo rango, que varían en un rango de valores enteros.

Los otros dos grupos adicionales de anotaciones están relacionadas con conjuntos de transición. El ConflictSet permite la identificación de un conjunto de transiciones que están implicados en un conflicto estructural, mientras que el SynchronySet permite la especificación de un grupo de transiciones utilizando el concepto de canal de comunicación síncrona, tal como se propone en [100]. En el último, una "semántica de fusión" es aplicable a todas las transiciones en el conjunto para comportarse como uno solo con los arcos de entrada y salida de todas las transiciones de ajuste. Por lo tanto, estos conjuntos pueden ser vistos como conjuntos de fusión de transición.

Los siguientes atributos adicionales se definieron para los tres principales tipos de elementos en PNML, a saber, las redes, los lugares y las transiciones:

- Nuevos atributos para lugares: Bound (el máximo alcanzable de marcado de un lugar, teniendo en cuenta el análisis para un marcado inicial específico), SignalOutputActions (dependencias del lugar para la señales de salida) y subred (por representaciones jerárquicas, fuera de alcance de este documento);
- Nuevos atributos para las transiciones: NetIOTransGuard (la guarda de la transición), NetIOTranslnput (señales de entrada de la transición y dependencias del eventos), NetIOTransOutput (dependencias de eventos de salida de la transición), prioridad (prioridad de transición dentro de su ConflictSet) y subred (por representaciones jerárquicas y fuera del alcance de este trabajo);

- Nuevos atributos para arcos: PTArclnscription (permitiendo la especificación de peso), ArcType (permitiendo arcos normales o de prueba tipo), y fuente secundaria y subTarget (para las representaciones jerárquicas, fuera del ámbito de este trabajo).

En el trabajo, se muestra una red IOPT modelando un controlador de estacionamiento, que ha sido editada con Snoopy gráfica [101], lo que genera un archivo PNML. El estacionamiento tiene una entrada y una salida. Es un ejemplo común y su modelo de RdP asociado se puede encontrar en otro lugar [34] [47]. La entrada y la salida se modelan respectivamente mediante las partes izquierda y derecha del modelo. Las plazas libres y ocupadas, modelan el estado del estacionamiento, se consideran un total de cinco plazas de estacionamiento.

Generación de código VHDL con redes IOPT

La generación de código VHDL a partir de la especificación PNML se lleva a cabo mediante la definición de un conjunto de reglas de conversión, basado en los dos componentes principales: interfaz del controlador y el modelo de comportamiento.

Las siguientes subsecciones presentan algunos detalles la implementación de estos componentes, teniendo en cuenta que ha sido usado un circuito síncrono como plataforma de implementación.

Estrategia de implementación

La estrategia propuesta se basa en la aplicación directa de la red IOPT, lo que significa que todos los nodos de la gráfica RdP tendrán un módulo corresponsal directo en el nivel de aplicación.

Cada lugar está representado por un módulo de memoria, que puede ser un registro o un flip-flop dependiendo de las características del lugar (registro para contener implementaciones de lugares acotados y un flip-flop para la implementación lugar seguro);

Cada transición está representado por una función combinacional, que puede generar las funciones de entrada para los módulos de almacenamiento que representan los lugares, lo que permite SET / RESET para flip-flop, y de incremento / reducción para los registros. Esta estrategia es muy conocida por la literatura (por ejemplo[102]). Sin embargo, debido a las características específicas de la gramática definida (y redes IOPT características asociadas) se presta especial atención a las siguientes características:

1) Arcos tabulados;
2) Las prioridades asociadas con las transiciones, de acuerdo con los conjuntos de conflicto definidos;
3) Un atributo para cada lugar;
4) arcos de test.

Desde el punto de vista del modelador, la adición de los arcos y las prioridades de prueba asociados con transiciones puede tener un grave impacto al abordar el tema de la resolución de conflictos. Como cuestión de hecho, el atributo de prioridad permite una solución simple para la resolución de conflictos (incluso desleal) y arcos de prueba permite la uso de árbitros justos [48].

El objetivo asociado con el atributo para lugares acotados es alimentar herramienta de generación automática de código con la información pertinente para la aplicación, especificando los recursos necesarios para realizar la implementación del lugar. En este sentido, la selección del módulo para implementar la aplicación de un lugar específico depende del número de bits necesarios para codificar el valor de cota de plaza.

Por último, desde el punto de vista de la generación de código para la ejecución del modelo, un aspecto muy crítico ha sido teniendo en cuenta, que es la prioridad de una transición específica dentro de su conjunto de conflicto. Esto se puede lograr utilizando un codificador de prioridad como solución para la implementación de las funciones asociadas.

Interfaces de entrada y salida

Los módulos de interfaz de entrada son de tres tipos:

1. Para sincronizar las entradas con el fin de evitar problemas de meta-estabilidad;
2. Para asegurar la generación de eventos basado en el análisis de la historia de las señales de entrada;
3. Para permitir la configuración de una plataforma de implementación específica.

El trabajo presenta el esquema de dos módulos para ser utilizados con las señales de tipo booleano y tipo rango, respectivamente. Para las señales booleana, los eventos se detectan en los flancos de la señal, mientras que para las señales de varios valores (enteros en un rango) se detecta el evento por el cruce borde de la señal con un nivel específico.

Conclusiones y trabajo futuro

La mayoría de las herramientas de RdP desarrolladas se limitan a la modelización, la simulación y, en algunos casos, la verificación.

El énfasis de los trabajos presentados se pone al llegar el código de implementación a través de las herramientas automáticas de generador de código, es decir, lo que permite la generación de código VHDL para la implementación de hardware (que se puede complementar con la generación de código C para la implementación de software).

Al tomar ventaja de todo el conjunto de herramientas en desarrollo dentro del proyecto FORDESIGN [79], que deberíamos lograr el objetivo principal del proyecto: poner las RdP para trabajar en el área de hardware-software de co-diseño de sistemas embebidos.

Análisis de Resultados

Es un proyecto que detalla todo el ciclo de vida del diseño de un sistema reactivo. Está basado en el uso de las IOPT para el diseño de un sistema embebido. Desde el punto de vista de las aplicaciones en sistemas complejos, el aporte es importante.

Las características que se obtiene en [44] son mostradas en las Tabla 30 y Tabla 31. Hay que considerar que muchas de estas características son proyectos futuros.

Tabla 30: Características del tipo de redes implementadas en el PP, en el trabajo de [44]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[44]	Si	Si	Si	Si	No	No	No	No	No	No	No	No	No

Tabla 31: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [44]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[44]	Si	Si	No	Si	No	No	No	No

Distintos tipos de Controladores Implementados por Software, Interpretados o Compilados.

Implementación de una Estación de Trabajo con un Lenguaje de Programación Paralela

En el trabajo realizado por [59] se ha propuesto un lenguaje de programación paralela, denominado PNPL (parallel programming language) que es basado en la RdP.

Este lenguaje tiene componentes gráficos y de texto, los componentes de texto son similares a los programas de como "Pascal" o "C" y los gráficos son RdP. Estas últimas, describen el flujo de control del procesamiento paralelo. Estos programas pueden ser ejecutados en máquinas con múltiples procesadores, controlados por un RdP.

Este trabajo reporta la implementación de un entorno de programación PNPL, el que consiste de un editor, un compilador y un simulador con depurador. Las RdP se dibujan con funciones gráficas del editor que también permite editar texto. Los componentes se pueden editar en múltiples ventanas.

El compilador traduce el programa PNPL a un código intermedio en "C". Luego, implementa procesos paralelos en UNIX. El sistema controla los multiprocesadores con una RdP.

El depurador puede rastrear la ejecución de los componentes gráficos y texto, y es posible trazar el movimiento de token en la RdP.

Este simulador permite desarrollar programas PNPL y evaluarlo como si se ejecutara en el hardware real.

Secciones de interés del trabajo
El modelo de las computadoras ordinarias de Von Neumann tiene una limitación para acelerar los procesos paralelos. Muchas computadoras paralelas y varios lenguajes de programación paralelos se han propuesto, pero estos lenguajes de programación paralela se diseñan utilizando modelos de memoria compartida o modelos de paso de mensajes, proporcionando primitivas sólo para la sincronización y para la exclusión mutua. No se puede describir el control total flujo y se hace difícil reconocer visualmente las estructuras paralelas presentes en los programas.

Por otra parte, los gráficos de RdP son buenos para describir el control paralelo visualmente.

Propone un lenguaje de programación paralela (PNPL) que utiliza RdP para describir el control flujo. En la programación con PNPL, podemos obtener un programa paralelo con el flujo de control, por lo que es más fácil escribir programas paralelos en comparación con los lenguajes ordinarios basados en texto paralelo. Los programas PNPL pueden ser ejecutados rápidamente en sistemas multiprocesador, controlados directamente por el movimiento de los token de la RdP.

Este trabajo reporta sobre la aplicación de un medio ambiente de programación PNPL la para una estación de trabajo.

Esta implementación nos enseña los puntos clave en programación paralela y los problemas encontrados en multiprocesadores.

Programa PNPL

Esquema de un programa PNPL

PNPL es un lenguaje de programación paralelo para multiprocesador controlado por una RdP, que consiste de un gráfico de RdP y componentes de texto.

El autor muestra un ejemplo del producto interno de dos vectores (2, 8) y (9, 6). Se representa con una gráfica de RdP utilizada para el control de flujo. Mientras que con un texto se muestra los componentes, que son los programas relacionados. Los programas y la red se relacionan de la siguiente manera: plaza-proceso es un proceso asignado a una plaza en la red, mientras que en un slot-proceso se asigna a una transición de la red a un programa. Una marca en una plaza indica que se está ejecutando una instancia del programa plaza-proceso, el que está relacionado con la plaza.

En este ejemplo, el programa de red que expresa "producto 1" y "producto2" se puede ejecutar en paralelo.

Parte del PNPL: programa y gráficas

El programa representado con la RdP se realiza con primitivas, las que son enumeradas a continuación. El programa consiste de subredes que se relacionan con una sola red principal.

1. Un token en una plaza Begin, declara inicio de subred o red principal. Indica el inicio y esta plaza no tiene procesos ejecutables.
2. Un token en una plaza End, declara la finalización de una red principal o subred. Se devuelve un token a una red superior.
3. Un token en una plaza Dummy, se utiliza para controlar eventos como la exclusión mutua.
4. Un token en la plaza Global, se usa para conectar una subred con otra subred. La plaza global no tiene ejecución por sí misma. Una plaza global en una subred tiene una plaza idéntica en otra sub red. Un token desaparece de un lugar común y aparece en el correspondiente lugar común en la otra subred.
5. Un token en un lugar Call, pasa una señal a una subred, recibiéndolo de nuevo cuando se retorna de subred.
6. Una plaza Normal, tiene un componente ejecutable que se corresponde con la plaza.
7. Una plaza Branch, tiene dos arcos de salida a estas transiciones. Se selecciona uno de los dos arcos de salida dependiendo del resultado de su proceso. Es similar a la declaración "switch".
8. Un arco Kill, extrae un token de una plaza. Si el lugar tiene un token cuando la transición se dispara, el arco elimina el token de la plaza abortando su proceso.

Los componentes de texto son plazas-programas y ranura (slot). Las plazas-programas y slot son asignados a las plazas y a las transiciones según el grafico de la RdP.

Una plaza-programa describe:

1. Transiciones de entrada al lugar.
2. Transiciones de la salida del lugar.
3. Las variables locales.
4. Cuerpo del programa.

El lugar-programa para una start-lugar es una plaza-programa especial. Aquí se declara constantes y variables globales, que pueden ser utilizados por cualquier plaza-programa en la misma subred.

Las plazas Begin, End, Global y Dummy no tienen cuerpos de programa.

La plaza-programa Normal y las plaza-Branch tiene un cuerpo de programa.

Un slot describe:

1. Lugares de entrada y los valores de la transición.
2. Lugares de salida y valores de la transición.

En un Slots, se declaran los datos de los lugares de entrada de a la transición y los datos de lugares salida de la transición.

En el trabajo se muestra el mecanismo conceptual para pasar valores a través de la slot.

Análisis de Resultados
El trabajo presentado en [59] muestra un lenguaje que incorpora las RdP en el control de flujo de ejecución de un sistema concurrente. Este lenguaje le da semántica a los componentes de la RdP.

Este trabajo muestra las ventajas y la facilidad con que se escribe un programa paralelo si se hace uso del flujo de datos. El hecho de impregnar de distinta semántica a las plazas y transiciones, oculta el modelo e introduce un nuevo grado de dificultad.

Hay una fuerte interacción entre el modelo y lo requerimiento para la ejecución de un PNPL. Las características que se implementan son las mostradas en las Tabla 32, Tabla 33 y Tabla 34.

Tabla 32: Aportes de [59] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[59]	No	No	Si	Si	No	No	No

Tabla 33: Aportes de [59] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[59]	No	No	Si	Si	No	No

Tabla 34: Aportes de [59] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[59]	No	No	Si	Si	Si	No	Si	Si	No	Si

De una RdP IOPT a C

Herramienta de generación de código automática
En este trabajo [49] se presenta una herramienta para la generación automática de código ANSI C a partir de modelos expresados como RdP IOPT (RdP con Input-Output Place-transición). Las

entradas de la herramienta son archivos PNML (Petri Net Markup Language) que contienen los modelos RdP IOPT.

La herramienta tiene dos interfaces diferentes, las que son: una simple interfaz gráfica y otra interfaz de línea de comandos. Esta última es susceptible de ser utilizada por otros sistemas computacionales de soporte, como herramientas de interacción. La herramienta también permite al usuario refinar algunas partes del código generado automáticamente para mejorar el rendimiento del mismo según algunas características de la plataforma de implementación.

El código generado se puede implementar directamente en los controladores específicos: Microcontroladores PIC de bajo costo, así como ordenadores de propósito general, que han sido utilizados para la validación.

Secciones de interés del trabajo
Las RdP son un lenguaje de especificación abstracto de particular interés cuando se utiliza para el modelado de sistemas concurrentes. Las RdP tienen una representación formal, así como también una representación gráfica perceptible. Debido a estas características las RdP son aceptadas cuando se trata de la implementación de herramientas [103]. Para la introducción de las RdP y su uso como modelo referirse a [104] y [105].

A pesar de su amplia utilización en el desarrollo de herramientas, estos están principalmente centradas en el modelado, la validación y de la verificación de los modelos de sistemas. Por otro lado, no hay muchas herramientas para la generación automática de código basado en RdP.

Se entiende comúnmente que la generación manual de código para los controladores permite la producción de código específico de una manera más eficiente para cada problema. Por otro lado, a pesar de que, la generación automática de código reduce el tiempo de generar del código, así como reduce los errores, estas herramientas son poco usadas.

Por otra parte en algunos casos, en los que hay herramientas para la generación de una variedad de lenguajes de destino, es posible generar código para varias plataformas basadas en el mismo modelo, que también puede ser utilizado en la simulación/validación del sistema así como la verificación de propiedades.

Implementación
La herramienta que se presenta en este trabajo es una extensión del trabajo que se presenta en [50], haciendo hincapié en la propia herramienta y no ya en las estrategias para la generación de código y el uso de la herramienta en distintas situaciones.

La Figura 12 resume el marco de ejecución de referencia. Varias reglas se han definido en [50], para el soporte de traducción de las diferentes estructuras de las RdP IOPT a código ANSI C, así también como la interfaz de la plataforma de implementación y modelo de referencia de la ejecución.

Este trabajo ofrece un entorno de generación de código ANSI C basado en RdP IOPT descritos en PNML, listos para ser desplegados en un controlador específico.

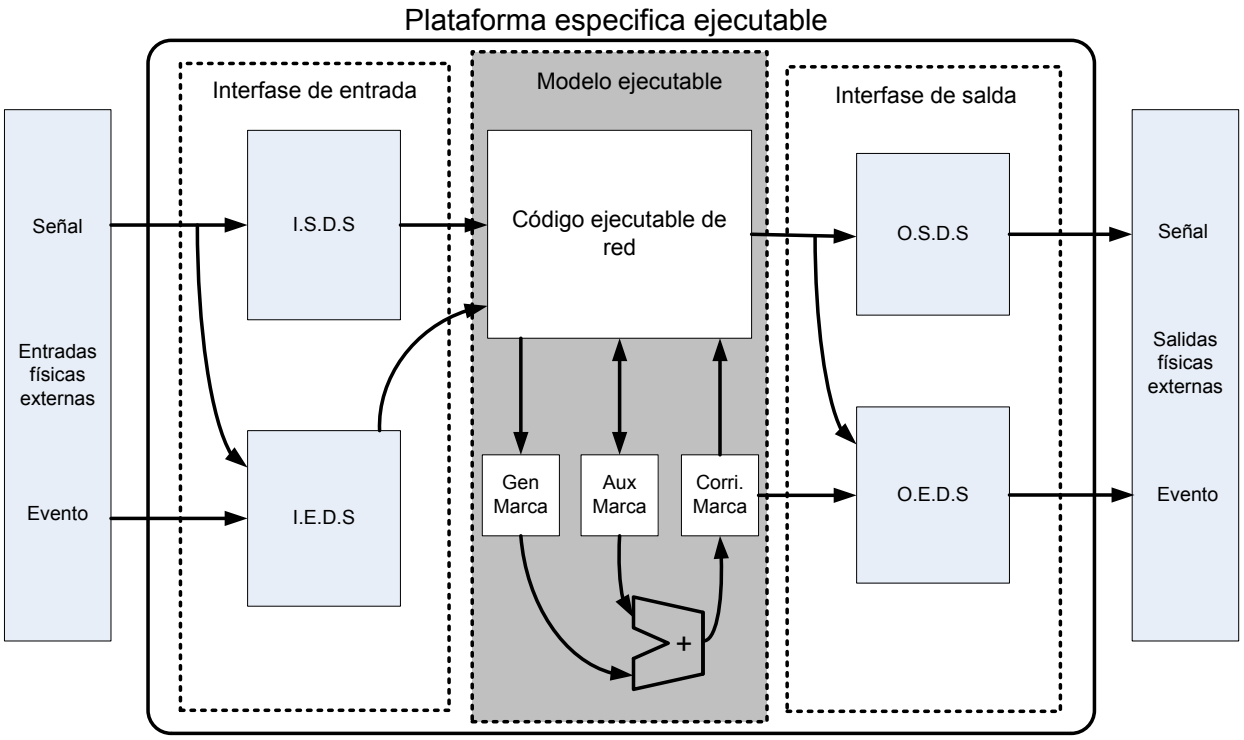


Figura 12: Framework de ejecución presentado en [49]

Los pasos de ejecución que se llevan a cabo por el controlador comienzan por la recepción de todas las señales de entrada externas, los eventos de entrada y la actualización de las variables asociadas definidas para ese propósito. La generación de eventos de entrada que dependen de cambios en las señales de entrada se produce después.

La ejecución del modelo de RdP IOPT se basa en la definición de tres buffers para el marcado, los que son: el marcado actual, marcado auxiliar, y generación del marcado.

Al comienzo de la etapa de ejecución, los valores de la marca actual se guardan en marcado auxiliar y la marca generada se inicializa a cero. Después de eso, se analizan las condiciones para el disparo de las transiciones, y las transiciones habilitadas son disparadas. El orden de los disparos se hace por su orden de prioridad. De esta manera se analiza la prioridad de cada transición y los conflictos son resueltos automáticamente.

Marco de ejecución presentado en [50], donde:

- I.S.D.S – Input Signal Data Structure;
- I.E.D.S. - Input Event Data Structure;
- O.S.D.S – Output Signal Data Structure;
- O.E.D.S. - Output Event Data Structure;
- Gen. Mark. – Generated Marking;
- Aux. Mark. – Auxiliary Marking;
- Curr. Mark. – Current Marking.

El disparo de una transición crea tokens que se agregan a la variables que generan nuevos marcados y quitan token del marcado auxiliar.

Análisis de resultados
La herramienta desarrollada permite la generación automática de código ANSI C asociada a modelos de RdP IOPT, especificados en formato PNML.

La herramienta se ha probado con varios ejemplos que vienen del área de automatización, los que son usados y validado en otros trabajos de los mismos autores. Un área de especial interés ha sido el área de controladores distribuidos basados en modelos RdP IOPT, donde un modelo se descompone en varios componentes paralelos, que se desplegarán en plataformas de ejecución autónomas. El soporte de comunicación es una interconexión directa a la red, e incluye soluciones de red en un chip.

El código generado, después de integrarse con el código dependiente de la plataforma, ha demostrado ser eficaz y robusto.

Este trabajo muestra las ventajas y la facilidad con que se escribe un programa paralelo si se hace uso de RdP IOPT. La sobrecarga introducida por el software de ejecución de la red es importante, puesto que requiere múltiples ciclos para determinar la sensibilidad de las transiciones y el disparo, y no permite integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 35, Tabla 36 y Tabla 37.

Tabla 35: Aportes de[49] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[49]	No	No	Si	Si	Si	No	No

Tabla 36: Aportes de [49] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[49]	No	No	Si	Si	No	No

Tabla 37: Aportes de [49] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[49]	No	No	Si	Si	Si	No	Si	Si	No	Si

Generación Automática de Programas Concurrentes con RdP

Los autores de [106], asumen que la generación automática de código a partir de RdP es un tema importante. En este trabajo se presenta un nuevo enfoque para traducir automáticamente las RdP en un programa concurrente. Las plazas de RdP se consideran como variables y las transiciones, como la declaración de operaciones que cambian el marcado según la semántica de activación y disparo. Para traducir convenientemente el programa con RdP a código C + +, se define un módulo de peso y transición independiente y un grafo de RdP virtual.

Se desarrollan normas de traducción de la estructura: secuencial, estructura concurrente, selección de la estructura y estructura de bucle de las RdP. De acuerdo con estas reglas de traducción, se propone un algoritmo de generación de código para la programación concurrente automática a

partir de RdP. Por último, a través del estudio de casos, se ilustra la eficacia del método desarrollado.

Secciones de interés del trabajo
Las RdP son formalismo conocidos para la descripción y modelización de sistemas, pero su uso se limita en gran parte a las actividades de desarrollo anteriores al sistema de software. Para extender el uso de RdP a todo el proceso de desarrollo de software, varios investigadores han explorado las técnicas de generación de código a partir de las RdP, los distintos lenguajes seleccionados incluyen XL/1 [107] , Ada83 [108], OCCAM [109] y C + + [110] . Weili Yao y Xudong [111].

El presente trabajo presenta un nuevo enfoque para traducir las RdP a un esqueleto de programa en C + + (lenguaje de programación orientado a objetos concurrentes de propósito general) [112]. Sin embargo, la referencia [112] sólo se centra en la sincronización a través de los lugares, y el código de programación no puede generar el código de forma automática. Referencia [111] presenta un enfoque para traducir RdP transición predicado jerárquicos en un esqueletos de programa C + +. El enfoque consiste en una arquitectura global de traducción y un conjunto de reglas de conversión basado en la sintaxis y la semántica de las redes de transición de predicados jerárquicos. Pero la generación automática de código a partir HPrTNs [113] no se puede realizar con facilidad debido a la naturaleza general y la complejidad de las limitaciones asociadas con las transiciones.

E. A. Golenkov presenta los primeros pasos en la creación traductor RdP a C + +, pero no pudo explicar cómo se traducen en detalle [114, 115]. Stephan Filipos ofrece una visión general de las diferentes estrategias para generar código de alto nivel a partir de RdP. Una de estas estrategias se fundamenta en las características de las RdP aquí investigadas con más detalle. Varias mejoras de la idea básica se presentaron y el impacto de estas mejoras en términos de ejecución eficiente de código generado se ilustra en la referencia [116].

Recientemente, Dianxiang Xu presenta una herramienta a la que llamó ISTA (Integración y prueba de sistemas de automatización) [117] para la generación de pruebas automatizadas y ejecución utilizando RdP de alto nivel como modelos de prueba de estado finito. ISTA tiene varias características únicas. Por ejemplo, permite que el código ejecutable de prueba se genere de forma automática a partir de un MID (Model-Implementation Description), incluyendo un alto nivel de RdP como el modelo de prueba y un mapeo de los elementos de RdP para la construcción de aplicación.

En este artículo se presenta un nuevo enfoque para convertir automáticamente las RdP en C++. En estr enfoque, en lugar de las RdP, se considera como variable a las plazas y las transiciones, como la declaración de operaciones que cambian de lugar marcado según la habilitación y el disparo. Con el fin de convertir convenientemente RdP a código de programación C++, módulo de peso y transición independiente se define y también un gráfico llamado de RdP virtual.

Se desarrollan las normas de conversión de la estructura de la secuencia, estructura concurrente, se selecciona la estructura y la estructura de bucle de las RdP. De acuerdo con estas reglas de conversión un algoritmo de código de programación concurrente se genera automáticamente para RdP propuesta. Por último, para demostrar este enfoque de generación automática, realiza un ejemplo.

Análisis de resultados

En este trabajo, se ha investigado la automatización para la generación de código de programa concurrente a partir de RdP. El concepto de módulo secuencial y transición independiente se definen y se construye con una gráfica llamada RdP virtual. Las reglas de traducción de la estructura de la secuencia, estructura concurrente, selección de la estructura y la estructura de bucle de las RdP también son desarrolladas.

La generación de código, de RdP, se puede realizar fácilmente según los métodos propuestos. Otra ventaja importante de este enfoque para la traducción es su generalidad.

Como se ha demostrado por estudios de casos, el enfoque de la traducción ha sido adaptada con éxito para generar programas C++ de RdP. Se señala como futuros trabajos, la aplicación de la idea a RdP con control e hibridas [118] [119].

Este trabajo [106] muestra como traducir RdP en programas C++ haciendo uso de módulos. Aquí también la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones y el disparo, y no permite integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 38, Tabla 39 y Tabla 40.

Tabla 38: Aportes de [106] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[106]	No	No	Si	Si	No	No	No

Tabla 39: Aportes de [106] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[106]	No	No	Si	Si	No	No

Tabla 40: Aportes de [106] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[106]	No	No	Si	Si	Si	No	No	Si	No	Si

Herramienta de Generación de Código para la Simulación y Control basada en CPN

El objeto de este trabajo [120] es implementar una herramienta de RdP para la simulación y control. Esta herramienta se utiliza en conjunción con un editor de RdP llamado NETMAN [121] que conforman un conjunto de herramientas integradas para el diseño, la simulación y el control.

Este artículo discute un tipo de RdP coloreadas que se utilizan para la descripción del modelado y la simulación.

Se discuten los detalles de la generación de código, junto con algunos de los usos posibles del código generado, tanto para la simulación y el control. Se presenta un ejemplo, de una RdP estocástica, y los resultados de la simulación se comparan con la solución numérica.

Secciones de interés del trabajo
Este artículo presenta una herramienta para la simulación y el control que se basa en las RdP coloreadas. Este simulador fue construido, por la necesidad de herramientas para desarrollo, puesto que las RdP son muy poderosa para modelar sistemas reales, pero no son de uso generalizado por los ingenieros, en parte debido a la falta de herramientas disponibles. Entre las que sí están disponibles la mayoría de ellas no son fáciles de usar y no están integradas con herramientas de desarrollo. Una de las razones es que utilizan diferentes políticas de ejecución convirtiéndose en una tarea difícil al usar varios paquetes para un solo modelo.

Otra dificultad es que todavía no hay un formato de descripción estándar para RdP, y siendo que hay muchos formatos diferentes, por lo que requiere la traducción de archivos en cada etapa de desarrollo del modelo. Debido a estas dificultades, se reconoce como beneficio llegar a tener un conjunto de herramientas integradas, lo que proporcionaría capacidades para el diseño, evaluación del desempeño, planificación, análisis y control. Todas estas herramientas deben seguir los mismos métodos de ejecución, estableciendo un conjunto común de extensiones y ser capaces de intercambiar archivos entre ellas.

LA primera construida por el autor, en el conjunto de herramientas, fue llamada NETMAN. Fue establecida principalmente para fines de diseño, aunque también puede crear archivos de RdP para la entrada a las herramientas de análisis y simulación existentes. Las herramientas de análisis y simulación, que utilizamos con NETMAN, ajustaban a condiciones especiales de ejecución, donde era difícil comparar la información obtenida de cada paquete, ya que las necesidades del modelo tenían que ser convertidas con frecuencia para mantener las mismas propiedades semánticas. Por esta razón, fue necesario añadir una herramienta de simulación y control para este conjunto de herramientas. Esta herramienta de simulación es el tema principal de este artículo.

En lugar de simplemente añadir un juego de símbolos a la herramienta de diseño ya existente creó un simulador externo independiente de cualquier sistema operativo o procesador.

Debido al requerimiento, el simulador fue implementado en ANSI C.

Detalles de la implementación
Con la creación de la herramienta el usuario puede utilizar una suite completa, por lo que le resultó importante elegir un modelo compatible con todas las herramientas existentes. Por esta razón, y para permitir que el CPN se expanda a un modelo incoloro, las transiciones no se pueden multiplicar habitualmente. Esto permite la ejecución de la red similar a una RdP temporizada, pero a un costo de hacer una RdP más grande, para representar todos los procesos independientemente.

El poder de modelado del simulador incluye un tipo de RdP coloreada jerárquica, mediante la asociación de un vector de número entero (de color) con cada símbolo, y con cada arco.

Condiciones de disparo simples y funciones pueden ser colocadas en los arcos. Esto se hace mediante la asociación de un vector de color con cada arco de entrada (donde cada color puede ser un número entero constante, o una variable). Cuando un miembro del vector de color es una variable, las correspondientes (nombres de) variables de diferentes arcos se corresponden de

manera que la transición se activa únicamente si todas los token en los arcos de entrada cumplen las condiciones de entrada resultantes.

Después de elegir la transición a disparar, se elige un conjunto de símbolos que han permitido el disparo de la transición, y después a las variables en los colores, de entrada se asignan valores coherentes con esta elección. Si el usuario ha creado una función de disparo, entonces se ejecuta. La principal forma en que se utilizan estas funciones, definidas por el usuario, es cambiar o asignar valores a las variables de los arcos de salida en base a las variables de los tokens de entrada. Esta restricción se hizo para los disparos y activa las funciones para permitir funcionalidad compleja en distintas simulaciones, sin permitir al usuario manipular directamente los token en los lugares. Si al usuario se le permite manipular el estado directamente, entonces la ejecución normal de la RdP puede ser violada.

Todas las funciones son definidas por el usuario y las funciones de disparo están escritas por el usuario en ANSI C. Si surge la necesidad de modelar algo que es demasiado complejo para representar prácticamente con una RdP, acciones y situaciones más complejas pueden ser escritas en ANSI C. Esta es toda la información necesaria para que esta funcionalidad sea añadida a NETMAN, donde el modelo se puede editar de forma rápida y sencilla.

Después de que el modelo de RdP se ha creado en NETMAN, el código C para una simulación se puede crear simplemente eligiendo la opción correspondiente en el menú. El código C puede ser compilado con cualquier compilador ANSI C, que a su vez genera un archivo ejecutable.

El código sirve como simulador basado en eventos, o un controlador basado en tiempo, al cambiar una sola línea en los archivos de origen.

Generación de código
La decisión de crear la simulación mediante la generación de código de NETMAN se hizo por varias razones:

1. ANSI C permite una simulación que podrá ejecutarse casi en cualquier ordenador, aprovechando los mejores recursos disponibles para un proceso de simulación.
2. ANSI C permite que el mismo código de simulación se utilice para el código de control, si está compilado para un microprocesador, cuando ser ejecuta en el modo de prueba (debugger en tiempo real).
3. Se han hecho esfuerzos para que gran parte de la estructura de códigos, de la red, pueda ser colocado en la ROM en tiempo de compilación, porque la mayoría de los controladores microprocesadores tienen muy poca espacio de RAM.

Esto se hace mediante el uso de la palabra clave “const en C”, que es utilizada por muchos compiladores para determinar si las variables se pueden poner en el espacio ROM. Si se requiere alterar la red, la mayoría de los compiladores permiten al usuario, reemplazar este comportamiento para las variables const.

El simulador es básicamente un reproductor de contador, con la misma estructura de código básico para todas las RdP. El código necesario para crear la estructura de datos de la red, y para configurar el marcado de la red es generado por NETMAN, y combinado con las funciones definidas por el usuario y las funciones de ejecución que comprenden un programa ANSI C completo. Desde la lógica principal del simulador no se genera el controlador. Con NETMAN el código que se genera para cada simulación tiene menos propensión para el error. También

permitirá la creación y modificación de la red en tiempo real, si es necesario. Genera el código de manera que el marcado de la red inicial se puede restablecer, con una sola llamada, a la función después que la simulación ha comenzado.

Detalles de implementación
Hay varias cuestiones de aplicación específica que son importantes para hacer funcionar el simulador con las características existentes de NETMAN. Estas características son importantes cuando se utiliza el simulador, y al momento de crear un modelo a simular.

Cuando NETMAN genera el código ANSI C, necesario para crear la estructura de datos para el RdP, hay dos opciones. Una opción es crear el programa mediante el uso de memoria asignada en tiempo de compilación. Esta versión es útil para los microprocesadores, y también se encargará de las grandes redes bajo las máquinas con grandes espacios de direcciones (por ejemplo, en UNIX, Windows NT). La otra opción es generar código que asigna dinámicamente la memoria para el RdP, en forma dinámica cuando se ejecuta el programa.

Este código es útil para máquinas que todavía tienen limitaciones de segmentos, cómo grandes bloques de datos global (por ejemplo, - MacOS, DOS, NT-Windows). Otra característica importante de la generación de código es que un modelo jerárquico se expande en un único gran RdP para fines de simulación, debido a que cada instancia de una subred debe tener su propio estado durante la ejecución de la red. Genera una tabla donde se escribe el código de referencia cruzada de los lugares y las transiciones entre la única red grande y la versión jerárquica. Sin embargo, el simulador no expande la red de color en una única RdP coloreada. Con el fin de conservar el espacio de la estructura de datos, la red de color se ejecuta directamente.

El simulador tiene un algoritmo bien optimizado para la ejecución del RdP. Durante cada ciclo de la red se elige una transición para su disparo. Después se dispara una transición específica. Las únicas transiciones habilitadas posibles son por lo menos en una de dos categorías:

1. la transición en cuestión fue activada antes de la transición seleccionada para disparar o
2. la transición fue habilitado por tokens de entrada o salida del lugar cuyo marcado ha cambiado.

Debido al algoritmo, las únicas transiciones que deben comprobarse cuando una transición ha sido disparada son las transiciones cuyos lugares de entrada corresponden a los lugares de entrada y salida de la transición, que era la última elegida para disparar.

Trabajo futuro
El autor sigue trabajando en el simulador y la herramienta NETMAN. Hay varios principios, actualmente en curso, para aumentar la utilidad de estas herramientas y mejorar su interacción. La herramienta de simulación se está modificando actualmente para permitir la animación de la RdP utilizando NETMAN como una pantalla gráfica de las token que fluyen a través del sistema. Los cambios también se están haciendo en la definición de RdP color. Se encuentra en este trabajo de modo que múltiples transiciones sean posibles de disparar, preservando al mismo tiempo el comportamiento de la RdP estocástico subyacente.

Análisis de resultados
Hay varias características importantes del simulador que se ha discutido, y razones que lo hacen diferente de otros simuladores. Las principales ventajas es que está escrito en ANSI C y por lo tanto es portátil para casi todas las plataformas de procesadores, incluyendo controladores de

microprocesador. El simulador se ejecuta tanto para evento como para los modos basados en el tiempo, lo que lo hace útil tanto para la simulación como para el control. El usuario define permisos y funciones de disparo, que pueden ser escritas en ANSI C. Lo que permite al usuario combinar fácilmente las ventajas de una RdP con la flexibilidad para interactuar ágilmente con una gran cantidad de opciones. El simulador también se basa en un tipo de RdP coloreada, que mantiene al modelo en formato pequeño y manejable. Además los modelos jerárquicos que se pueden construir por NETMAN automáticamente se contraen a una red de un solo nivel con fines de simulación.

El simulador ha tenido un significativo uso, desde la primera versión. Se ha utilizado para una aplicación del control de tráfico de automotores, que implicaba la generación de código de control para un sistema de control distribuido utilizando una red de microprocesadores, así como para simular el comportamiento del sistema de control.

Este trabajo [120]muestra cómo realizar un simulador con RdP en programas C haciendo uso de un algoritmo de disparo de las transiciones.

Aquí también la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones y el disparo, no permitiendo integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 41, Tabla 42 y Tabla 43.

Tabla 41: Aportes de [120] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[120]	No	No	Si	Si	No	No	No

Tabla 42: Aportes de [120] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[120]	No	No	Si	Si	No	No

Tabla 43: Aportes de [120] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[120]	No	No	Si	Si	Si	No	Si	Si	No	Si

Herramienta Tailored para la Generación de Códigos de Modelos con RdP

El uso de las RdP para el modelado de sistemas de eventos discretos está bien estudiado. Sin embargo, las herramientas que permitan la aplicación de estos modelos y el soporte a la generación de código, son todavía muy pocas, casi inexistentes. Este trabajo [34] se basa en la presentación de una clase de RdP sobre la base de redes lugar / transición y conceptos conocidos de las RdP sincronizadas e interpretadas. Esta clase de RdP permite la asociación de las señales de entrada externas a las transiciones, la asociación de señales de salida externas de las transiciones y evolucionar las marcas. Además, la clase de RdP dá soporte a la especificación de los eventos de

entrada y salida. El documento presenta un generador de código capaz de generar una salida optimizada de código ejecutable a partir de estas redes. El código generado se puede optimizar mediante varias estrategias diferentes, que facilitan la creación de código adaptado a plataformas específicas, así como para la red específica.

Secciones de interés del trabajo
Una forma común, para modelar sistemas embebidos, es utilizar una descomposición basada en un controlador que interactúa con su entorno. Mediante el uso de una señal de sincronización (un reloj global externo), esta interacción puede obligar a pasar por puntos periódicos en el tiempo. Estos sistemas de eventos discretos pueden ser convenientemente modelados por las RdP. Este hecho es conocido desde hace mucho tiempo (véase, por ejemplo, el artículo de Holloway [122]). A pesar de esta adecuación, el número de generadores de código a partir de modelos de RdP es extremadamente escasa, extraída de la base de datos de herramientas de RdP [123].

Este artículo presenta un generador de código a partir de modelos de RdP, y es capaz de generar código altamente configurable, susceptible de ser ejecutado incluso en plataformas con mínimos recursos. Para ello se funda en una base de datos de posibles bloques de código. Cada uno de ellas tiene opciones de diseño ligeramente diferente. Estos bloques se eligen y configuran, no sólo con las propiedades específicas de la clase de RdP en el uso, sino también a partir de las características particulares de cada modelo. Por último, el usuario tiene el control sobre otro conjunto de opciones de generación de código. Cuando se toman en conjunto, todas estas opciones permiten la creación de código altamente adaptada a la plataforma en la que se va a ejecutar.

En la actualidad, el generador de código es capaz de manejar una extensión sencilla e intuitiva para colocar redes plaza /transición [80, 124]. Podría decirse que es la clase de RdP que permite su uso en la especificación de sistemas de eventos discretos más utilizados y conocidos en control. Esta clase se denomina RdP entrada /salida plaza/transición (RdP IOPT). Se añade un conjunto mínimo de anotaciones para colocar redes plaza/transición que permiten su conexión con el medio ambiente. Más específicamente, las RdP IOPT permiten la especificación de las señales de entrada externas y eventos de entrada externos asociados a los disparos de las transiciones. También especifican los eventos de salida y señales de salida externas (acciones). El primero se asocia a las transiciones, mientras que el segundo se relaciona con la colocación de marcas.

Un entorno de desarrollo
La generación de código a partir de modelos RdP IOPT es parte de un entorno de desarrollo, que además de la aplicación como generador de código aquí descrito, nombrado como PnGenerator, incluirá un editor gráfico, un simulador y una herramienta de verificación. Prevé el desarrollo del editor gráfico, pero el mismo aún no está iniciado. Las dos últimas herramientas están estrechamente relacionadas con PnGenerator y se están desarrollando actualmente.

El sistema completo se presenta en la Figura 13. También se puede encontrar en dos documentos preliminares donde se presentó la visión completa para el entorno de desarrollo [51] [47] . El editor (el PnEditor en la esquina superior izquierda de la Figura 13) es capaz de generar modelos de RdP en lenguaje PNML , con un formato de intercambio basado en XML para todas las clases de RdP [125] [96]. En la actualidad, la PnGenerator lee este formato y se obtiene código ANSI C.

El autor en desarrollos futuros promete incluir la generación de código mediante diferentes lenguajes, como códigos en VHDL para la implementación de hardware, SystemC para los diseños de nivel de sistema y lenguajes de programación para “Programmable Logic Controllers”.

Finalmente, el código creado por la herramienta de PnGenerator se utiliza actualmente para la ejecución final en una plataforma de hardware específica, pero en un futuro el propósito es llegar a la simulación y análisis a través de la adición de código de interfaz apropiado utilizando el código ejecutable generado para simular y verificar el modelo.

Un modelo como controlador de sistema
Desde el punto de vista del controlador, se ha tenido en cuenta que todos los sistemas que van a ser controlados, pueden poner a disposición un conjunto de señales de entrada de lectura y un conjunto de permisos de escritura para señales de salida. Define los eventos de entrada que modifican el estado consecutivo del modelo de entrada y los eventos de salida que se activan por el disparo de las transiciones. La interfaz del sistema es controlada con una RdP IOPT.

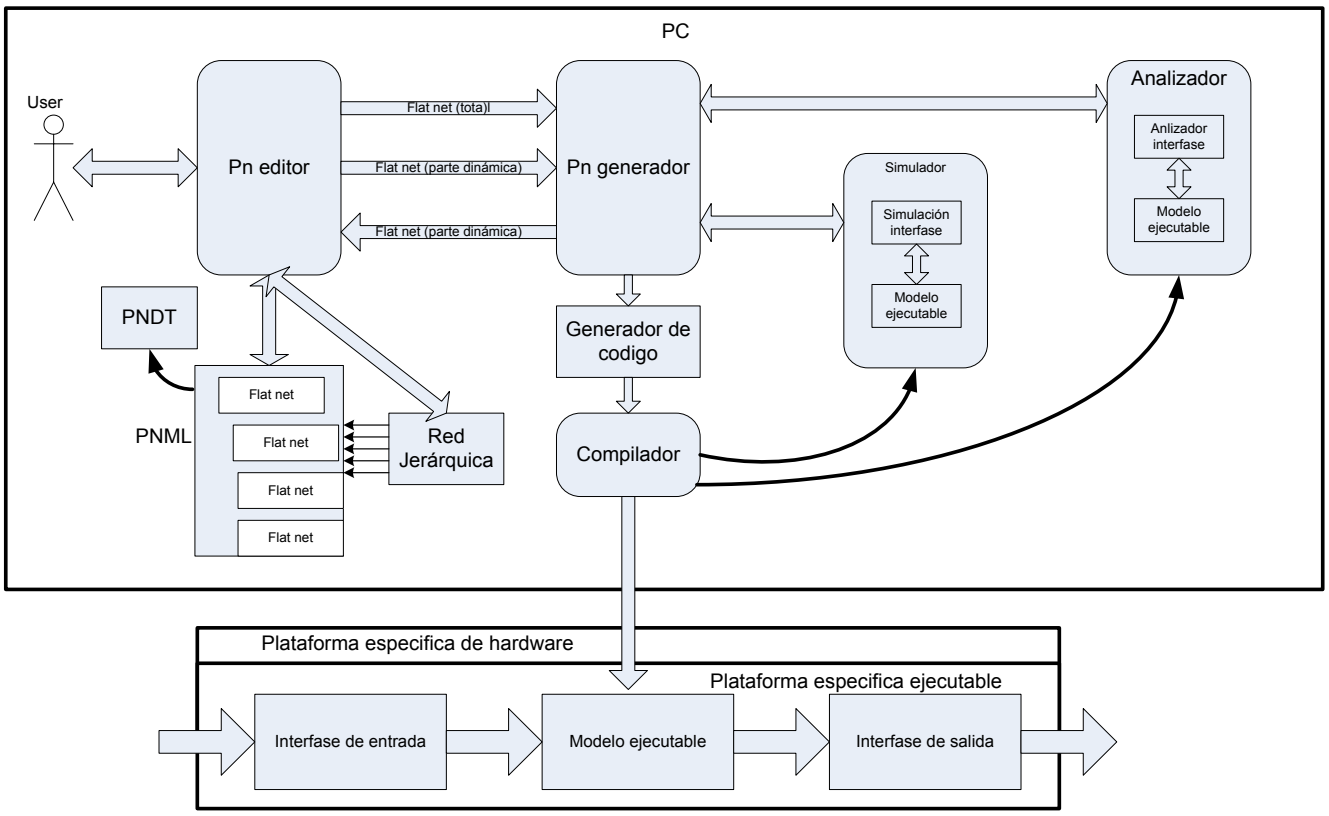


Figura 13: Arquitectura del entorno de desarrollo

Definición 1: Interfaz de sistema

Es una tupla ICS = (IS, IE, OS, OE) que cumple los siguientes requisitos:

1. IS es un conjunto finito de señales de entrada.
2. IE es un conjunto finito de eventos de entrada.
3. OS está configurado finito de señales de salida.
4. OE se establece finito de eventos de salida.

5. $IS \cap IE \cap OS \cap OE = \emptyset$.

Los eventos de entrada se calculan sobre la base de los cambios de las señales de entrada asociadas. Sin embargo, este cálculo se hace transparente para el modelo de red, que vé los eventos de entrada como cualquier otro tipo de señal de entrada. Esto se hace posible por el código externo que establece una interfaz entre las estructuras de datos utilizadas por la red y las señales de entrada externas. El mismo enfoque se sigue con respecto a las señales de salida. Esta interfaz se ilustra en la parte inferior de la Figura 13 (Interfaz de entrada e interfaz de salida), y se explica en detalle en un trabajo previo [47].

Definición 2: Estado de entrada del sistema

Dada una interfaz $ICS = (SE, IE, OS, OE)$ de un sistema controlado (Definición 1), un estado de entrada del sistema se define por un par $SIS = (ISB, IEB)$ que cumplen los siguientes requisitos :

1. ISB es un conjunto finito de señales de entrada relacionadas: $ISB \subseteq IS x\ \mathbb{N}o$.
2. IEB es un conjunto finito de eventos de entrada relacionados: $IEV \subseteq IE\ x\ \mathbb{B}$.

Redes de entrada / salida lugar / transición
La definición de las RdP IOPT supone la disponibilidad de un lenguaje de etiquetas que permite la especificación de expresiones algebraicas, variables y funciones. El conjunto de expresiones booleanas se llama BE y la función Var (E) devuelve el conjunto de variables en una expresión dada E. Como la aplicación es el objetivo final de un modelo RdP IOPT el lenguaje con etiquetas debe ser preferentemente lo mismo que el lenguaje de código generado (por ejemplo, ANSI C), ya que esto simplifica el proceso de generación de código y evita la introducción de un lenguaje adicional en el proceso de desarrollo.

Definición 3: IOPT red

Dado un sistema, a ser controlado con una interfaz $ICS = (SE, IE, OS, OE)$, dada una red de IOPT que es una tupla $N = (P, T, A, M, peso, ISG, es\ decir, oe, osc)$ que cumple los siguientes requisitos:

1. P es un conjunto finito de plazas.
2. T es un conjunto finito de transiciones (disjunto a P).
3. A es un conjunto de arcos, de tal manera que de $A \subseteq ((P\ x\ T) \cup (T\ x\ P))$.
4. M es la función de marcado: $M: P \longrightarrow \mathbb{N}o$.
5. Peso: $A \longrightarrow \mathbb{N}o$.
6. isg es una entrada de señal guarda, función aplicada, a transiciones a las expresiones booleanas (donde todas las variables son señales de entrada): isg: $T \rightarrow BE$, donde $\forall eb \in isg(T), Var(eb) \subseteq IS$.
7. ie es decir, es una función parcial de eventos de entrada y se aplica a las transiciones de eventos de entrada: $ie: T \rightarrow IE$.
8. oe es una función parcial de eventos de salida que se aplicar a las transiciones de eventos de salida: ee: $T \rightarrow OE$.
9. osc es una función de señal de plazas de salida condicional, un conjuntos de reglas osc: $P -> P\ (REGLAS)$, donde las $reglas \subseteq (BES\ x\ OS\ x\ \mathbb{N}), BES \subseteq BE$. y $\forall e \in$

$BES, Var\ (e) \subseteq ML$ con ML el conjunto de identificadores para cada lugar marcado después de una etapa de ejecución definida.

Las señales de entrada externas se asocian a las transiciones de dos maneras distintas:

1. directamente como variables de una expresión booleana que actuará como un guarda para el disparo de la transición;
2. indirectamente, como los eventos que se calculan sobre la base de la modificación de la señal.

Las señales de salida externas se pueden cambiar de dos maneras diferentes:

1. basada en el lugar marcados en el extremo de un paso (Moore-like fashion);
2. basado en el disparo de transición (Mealy-like fashion).

En cuanto a las reglas de transición de disparo fue adoptado el paradigma de RdP sincronizas [104].Cada vez que una transición está activada y la condición externa asociada es verdadera (el evento de entrada y la guarda de la señal de entrada son verdaderas) : la transición se dispara . Desde el punto de vista del modelo de red significa elegir un paso de máxima ejecución (con un máximo número de transiciones). El paradigma sincronizado también implica que la evolución sólo es posible en los instantes de tiempo específico denominado "tics". Estos se definen por un reloj global externo. Una etapa de ejecución es el período entre dos tics.

En las siguientes definiciones que utilizamos M (p) para denotar las marcas de lugar p en una red con la marca M y OT para indicar los lugares de entrada de una transición t determinado o un determinado conjunto de transiciones S: $\bullet t = \{p|\ (p,t)\ \in A\}$; $\bullet S\ \{p|\ (p,t)\ \in A\ \wedge\ t \in\ S\}$.

Definición 4: Condición de habilitar

Dada una $red\ N\ =\ (P,T,A,M,peso,ISG,isg,oe,osc)$ y una interfaz de sistema de $ICS\ =\ (IS,IE,OS,OE)$ entre N y un estado de entrada del sistema $SIS\ =\ (ISB,IEB)$, una transición t está habilitada para dispararse si y sólo si las siguientes condiciones se cumplen:

1. $\forall p\ \in \bullet t\ , M\ (p) \geq\ weight(\ (p,t)$.
2. La guarda de la transición t , señal de entrada, se evalúa como verdadera para la señal de entrada según es asociada por: i$sg\ (t)\ <ISB>$.
3. $(ie\ (t), true)\ \in\ IEB$

Definición 5: Paso de red IOPT

Sea $N\ =(P,T,A,M,peso,ISG,isg,oe,osc)$ una red y $ICS\ =(ES,IE,OS,OE)$ de una interfaz de sistema de entre N y un sistema con estado de entrada $SIS\ =\ (ISB,IEB)$. Vamos también $ET\ \subseteq T$ el conjunto de todas las transiciones habilitadas como se define en la Definición 4. Entonces, Y es un paso en N si y sólo si la siguiente condición se cumple:

$$Y \subseteq ET \wedge \forall t_1 \in (ET \setminus Y), \exists\ SY \subseteq Y, (\bullet t_1 \cap \bullet SY) \neq \emptyset \wedge \exists\ p \in (\bullet t_1 \cap \bullet SY), (weight(p,t_1) + \sum_{t \in SY} weight(p,t) > M(p)$$

Un paso RdP IOPT es máxima. Esto significa que ninguna transición adicional puede ser disparada sin conflicto efectiva con una de transición en la etapa.

Finalmente, definimos una ocurrencia RdP IOPT y la respectiva marca sucesor.

Definición 6: Paso ocurrencia y marcado sucesor

Dada una red $N = (P, T, A, M, weight, ISG, isg, oe, osc)$ y una interfaz de sistema $ICS = (IS, IE, OS, OE)$ entre N y un sistema con estado de entrada $SIS = (ISB, IEB)$, la ocurrencia de un paso Y en la red N devuelve el $N' = (P, T, A, M', weight, ISG, isg, oe, osc)$, igual a la red N, excepto para el marcado M' que viene dada por la siguiente expresión de sucesor:

$$M' = \left\{ \left(p, m - \sum_{t \in Y \wedge (p,t) \in A} weight(p,t) + \sum_{t \in Y \wedge (p,t) \in A} weight(t,p) \right) \in (Px\mathbb{N}o) | (p,m) \in M \right\}$$

Generación de código

Hay que destacar que existen beneficios importantes en la generación de código, los que son:

- Calidad del código escrito manualmente varía según el ciclo de vida de un proyecto. Mientras que, por la mejora del diseño, los códigos generados siempre aumentan la calidad a través del tiempo; una vez que se encontraron errores que se pueden corregir en las plantillas de código.
- La consistencia de los códigos generados automáticamente son coherentes en su estructura, firmas de métodos, nombres de variables y comentarios.
- Los generadores aumentan la productividad por que permite construir códigos en una fracción del tiempo comparado al escrito manualmente. Es un código que se puede construir simultáneamente para la prueba y la ejecución.

Modelo de la generación de código

El modelo adoptado para la generación de código, también nombrado "Código Munger" [126], es la forma más común de generador de código. Un código Munger procesa una o más archivos fuente de código y genera uno o más archivos de salida. En la aplicación lee un archivo de RdP, en el formato PNML [96, 103] y genera un conjunto de archivos ANSI C basado en bloques de código con plantillas predefinidas.

Una de las características claves del proyecto es la compatibilidad con diferentes plataformas y arquitecturas. Esta es la razón para el uso de la norma ANSI C como el lenguaje generado, ya que generalmente esta disponibles en cualquier plataforma de software.

Para construir el código fuente generado por el PnGenerator, representado en el centro de la Figura 14, utiliza información específica acerca de la clase de RdP en uso y elije los bloques. Esta elección la efectúa a partir de un amplio conjunto de bloques de código predefinidos, con el objeto de construir las funciones necesarias y definiciones de tipos de datos.

También, se permiten varias opciones de optimización parametrizadas para el código generado. La Figura 14 ilustra las principales entradas y salidas de la aplicación PnGenerator, junto con una visión de alto nivel de su arquitectura interna.

Bloque Generador

Los bloques generadores (Figura 14) permiten que los bloques de código predefinidos puedan ser modificados fuera de la aplicación.

También simplifica la generación de bloques de código similares. Una etiqueta XML <id> identifica cada bloque. El generador recibe una orden para producir un bloque específico de código junto con la información necesaria. Se procesa el bloque de la plantilla con la información recibida y devuelve su resultado.

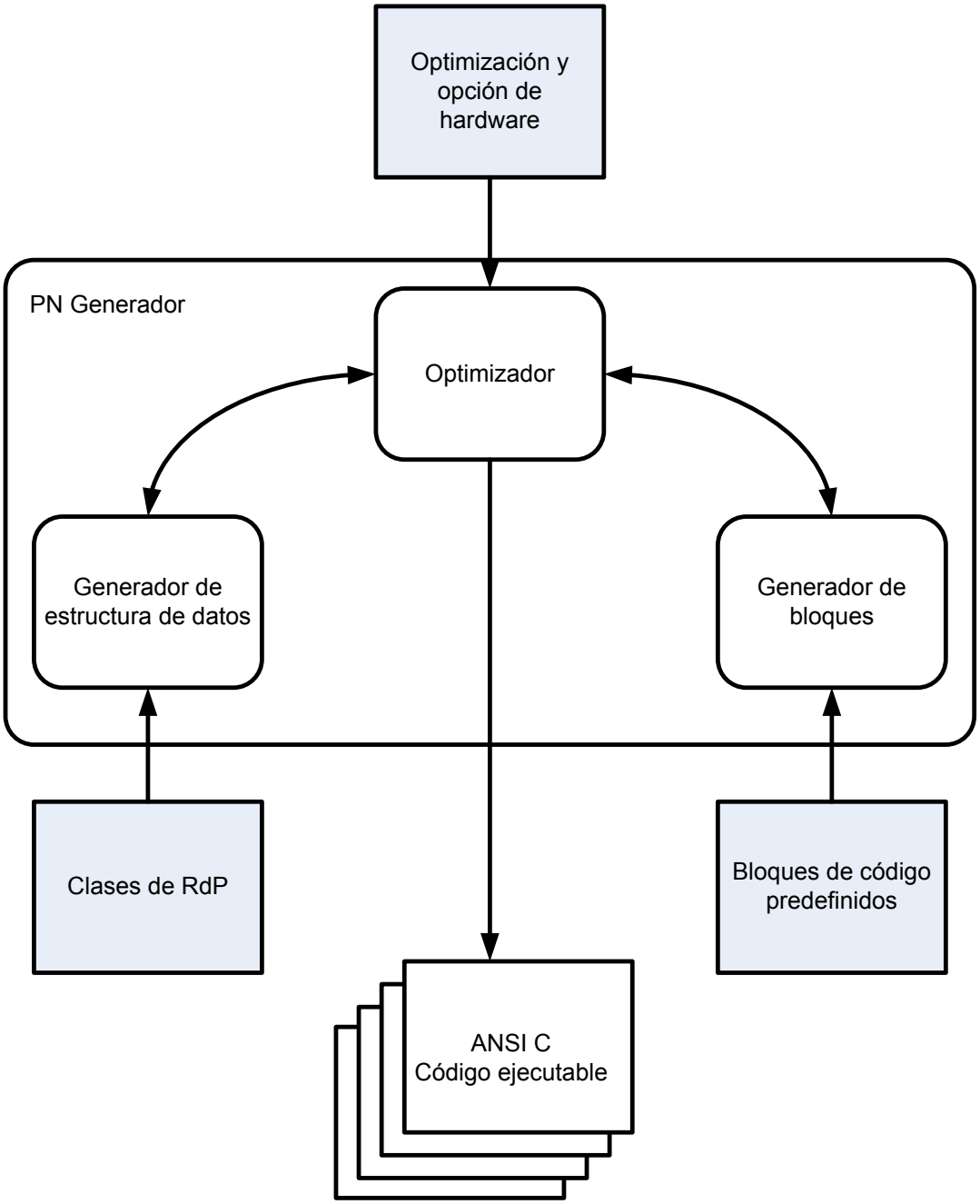


Figura 14: Diagrama del PnGenerator

Ejemplo

En esta sección se han presentado brevemente un modelo y el código generado. El modelo especifica un controlador de acceso a un estacionamiento con una entrada y una salida. Con respecto a la notación utilizada, los eventos de entrada y de salida están representados por triángulos vacíos apuntando hacia y desde las transiciones respectivamente. Las guardas de la señal de entrada se presentan entre corchetes y se subrayan los identificadores de eventos de salida. En este ejemplo, todos los eventos de entrada tienen los mismos identificadores con las respectivas transiciones. Las condiciones de la señal de salida asociadas a los lugares se presentan dentro de bloques. Para la plataforma de implementación se utilizó un micro-controlador de bajo costo, de la familia PIC, con relaciones Input/Output suficientes para la tarea. Esto se complementó con un compilador C integrado en las herramientas de micro-controladores. Para este modelo el código generado tiene alrededor de 900 líneas de código ANSI C (1250 líneas), incluyendo comentarios y utiliza alrededor de 180 bloques generados.

Análisis de resultados

El generador de código presentado crea un código que se puede utilizar para múltiples propósitos. A saber: la ejecución en diferentes plataformas de hardware específicas y la simulación con fines de análisis. La base de datos del generador de secuencias permite una técnica flexible y ampliable para generar códigos a medida, para cada uno de los recursos específicos de la plataforma, teniendo en cuenta las características de cada clase de RdP y modelo.

La actual herramienta está lista para soportar la generación de código en otros lenguajes como VHDL para las implementaciones de hardware, SystemC para los diseños de nivel de sistema y lenguajes de programación de controladores lógicos programables.

Presta especial atención a la resolución de conflictos.

Tiene varias actividades pendientes de añadir como algunas opciones de optimización y la ampliación de bloques de código disponibles para permitir un grano más fino de optimización. Esto deberá permitir más pruebas en relación con los diferentes escenarios para el equilibrio de la memoria / velocidad en cada plataforma y con diferentes tamaños de modelos. En la actualidad, la asociación entre las estructuras de datos y generadores de bloque está codificada en la aplicación del generador. Como proyecto de trabajo futuro presenta que será configurable por el usuario.

Este trabajo [34] muestra cómo ejecutar la traslación de un modelo realizado en RdP IOPT a un programas C, haciendo uso de módulos construidos en C. Aquí también la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones y el disparo, y no permite integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 44, Tabla 45 y Tabla 46.

Tabla 44: Aportes de [34]con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[34]	No	No	Si	Si	No	No	No

Tabla 45: Aportes de [34] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[34]	No	No	Si	Si	No	No

Tabla 46: Aportes de [34] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[34]	No	No	Si	Si	Si	No	Si	Si	No	No

Implementación de Controladores Digitales en C a partir de Modelos de RdP

En el trabajo de [50] se presenta una herramienta para la generación automática de código, con el fin de implementar controladores a partir de modelos de RdP susceptibles de ser desplegadas en plataformas de uso común y lenguajes de programación de alto nivel, tales como C, C + +, y Java. El código generado se vincula con las funciones específicas de la plataforma y el soporte de diferentes tipos de plataformas de ejecución, que van de los micro-controladores de bajo costo para estaciones de trabajo, a los microcontroladores integrados (propiedad intelectual - IP) para ser embebidos en FPGA (Field Programmable Gate Arrays).

El comportamiento del controlador del sistema se modela utilizando RdP IOPT (Input-Output Place-transición Petri Net), que están representados a través notación PNML (RdP Mark-up Language).

Los autores han desarrollado una herramienta para la generación automática de código, se ha alcanzado este objetivo en cooperación con otras herramientas desarrolladas dentro de un marco basado en modelos. Se presenta una aplicación a un sistema de automatización compuesta por un conjunto de controladores distribuidos.

Secciones de interés del trabajo
Las RdP son un lenguaje de especificación abstracta con una representación gráfica muy conocida, especialmente interesante para la construcción de modelos concurrentes. Sin embargo, a pesar de la disponibilidad de numerosas utilidades [103] , las RdP se mantienen relativamente infrautilizadas como herramienta para el desarrollo dirigido por modelos , probablemente debido al número muy escaso de las herramientas de generación de código.

Por lo tanto, las RdP se utilizan sobre todo para el desarrollo de modelos de alto nivel que se utilizan para validar y verificar los sistemas. Algunas de ástas tareas, tal vez implementadas como tarea separada, utilizan lenguajes y herramientas distintas.

Un marco de desarrollo, basado en modelos para el uso de las RdP como el lenguaje de diseño, para la construcción de modelos ejecutables se presenta en [46], como trabajo de referencia. Esto es validado mediante el “token-player Paradigm”, pero la herramienta que aquí se presenta pone de relieve la posibilidad de generar modelos ejecutables para varias plataformas distintas, con diferentes capacidades de memoria y de procesamiento. Más específicamente, se presenta una herramienta para la generación automática de código de control de modelos de RdP. El código generado es susceptible de ser desplegado en plataformas comunes utilizando lenguajes de programación de alto nivel ampliamente usados , tales como C, C + + y Java.

El código generado está vinculado con funciones específicas de plataforma, soportando diferentes tipos de plataformas de aplicación. Estas van desde los micro-controladores de bajo costo para las estaciones de trabajo hasta los IPs microcontroladores (propiedad intelectual) para ser integrado en FPGA (Field Programmable Gate Arrays).

La clase de RdP , descrita en [46] , añade la posibilidad de modelar señales y acontecimientos externos a la clase bien conocida de redes lugar/transición [127] . Por lo tanto, los modelos no son autónomos en su interacción con el medio ambiente y dependen de él. Para ese fin , el lenguaje de marcado de la RdP [128], se amplió con el fin de adaptar la representación de señales de entrada y de salida , como se describe en [46].

También se presenta el flujo de desarrollo que es soportado para el uso de la herramienta propuesta. Finalmente, se analizan los resultados, y las conclusiones, incluyendo temas para trabajos futuros.

Clase de RdP: entrada-salida/ plaza-transición
En esta sección, se presenta brevemente las principales características de la clase de RdP IOPT. La presentación completa , incluyendo la semántica formal y la definición se encuentra en [80], y ha sido presentado en [46, 96, 129]. La clase RdP IOPT es una extensión de las RdP plaza-transición [127]) .

El objetivo es permitir la especificación de la interacción entre el medio ambiente y la red, que es el modelo de un controlador.

Para ese fin, el modelo de controlador especifica señales y los eventos de entrada y de salida. A medida que éstos restringen el comportamiento del modelo, la red se convierte en no autónoma. Existen varias otras clases de redes no autónomas (por ejemplo [102, 130-133]) algunos desde hace bastante tiempo, pero ninguno con resultado de generación de código en el marco de soporte para plataformas específicas utilizando lenguajes de programación de alto nivel de propósito general.

El comportamiento sincronizado implica que el disparo de la red sólo es posible en instantes específicos denominados “tics”, que se define por un reloj global externo. Las RdP IOPT tienen una semántica de paso máximo: cada paso incluye todas las transiciones de forma autónoma habilitados cuyos guardas y entrada de eventos son verdaderas. El paso se ejecuta entre dos tics.

En comparación con las redes de plaza/transición, las redes RdP IOPT tienen las siguientes posibilidades adicionales, que se presentan a continuación en forma breve:

- Arcos de test y arcos con pesos;
- Prioridades en las transiciones;
- Guardas en las transiciones (las guardas son funciones booleanas de las señales de entrada);
- Eventos de entrada y salida a las transiciones;
- Señales de salida en algunas plazas.

Arcos de test tienen la semántica usual de disparo: si una transición está conectada a un lugar a través de un arco de test la transición solo puede disparar si el lugar tiene, al menos, tantos token como el peso de arco de prueba. Sin embargo, cuando se dispara la transición del arco de prueba

no elimina los token. Así que, cuando los arcos regulares y los arcos de prueba son los arcos de salida de una transición específica, no hay conflicto. Existe conflicto cuando son arcos de entrada.

Las prioridades en las transiciones permiten determinismo al disparar las transiciones en conflicto. Generalmente a todas las transiciones en conflicto estructural se les dan un valor de prioridad. Entre las transiciones habilitadas sólo la transición con la máxima prioridad de disparo se dispara y luego el resto. Siempre y cuando se mantenga activada después del disparo de otras transiciones en el mismo conjunto de conflictos con mayor prioridad.

Las guardas de transición son funciones de Boole de las señales de entradas externas. Una transición sólo se puede activar si la guardia se evalúa como verdadera.

Cada transición puede tener una entrada y una salida de evento. Ambos (eventos de entrada y/o salida) son opcionales. Si está presente, el evento de entrada debe ser verdad para que la transición se dispare. El evento de salida se hizo verdadero cuando se dispara la transición.

Cada lugar puede tener un conjunto de reglas. Cada regla tiene tres partes:

1. una función de Boole de la red de señalización,
2. una señal de salida,
3. un valor.

Después de cada paso, para cada regla, si la condición es verdadera la señal correspondiente se le asigna el valor.

A continuación, se presenta cómo generar código ejecutable a partir de un modelo RdP IOPT.

Estrategia de generación de código
El código generado tiene dos componentes de código principales:

1. el modelo es ejecutable independiente de la plataforma (el controlador) ;
2. el código es dependiente de la plataforma.

La primera es una traducción directa del modelo de red, mientras que el segundo se encarga de la comunicación entre el controlador y el medio ambiente.

La Figura 15 describe la arquitectura del modelo. El modelo ejecutable interactúa con el código dependiente de la plataforma, haciendo uso de las interfaces de entrada y de salida.

La comunicación entre el modelo de ejecución y el código dependiente de la plataforma se basa en “Señales y Eventos”. La interfaz de entrada lee las señales de entrada y eventos de los respectivos instrumentos físicos externos. Las señales de entrada se leen y se almacenan en una estructura de datos compartida con el código del modelo ejecutable. Cada módulo tiene la siguiente nomenclatura y descripción:

- I.S.D.S - Estructura de entrada de datos de la señal ;
- I.E.D.S. - Estructura de entrada de datos de eventos ;
- O.S.D.S - Salida de estructura de datos de la señal ;
- O.E.D.S. – Salida de Estructura de datos de eventos ;
- Marcado general. – Generado de marcado ;
- Aux. Marcas. - Auxiliar de marcado ;
- Curr. Marcado. - Marcado actual.

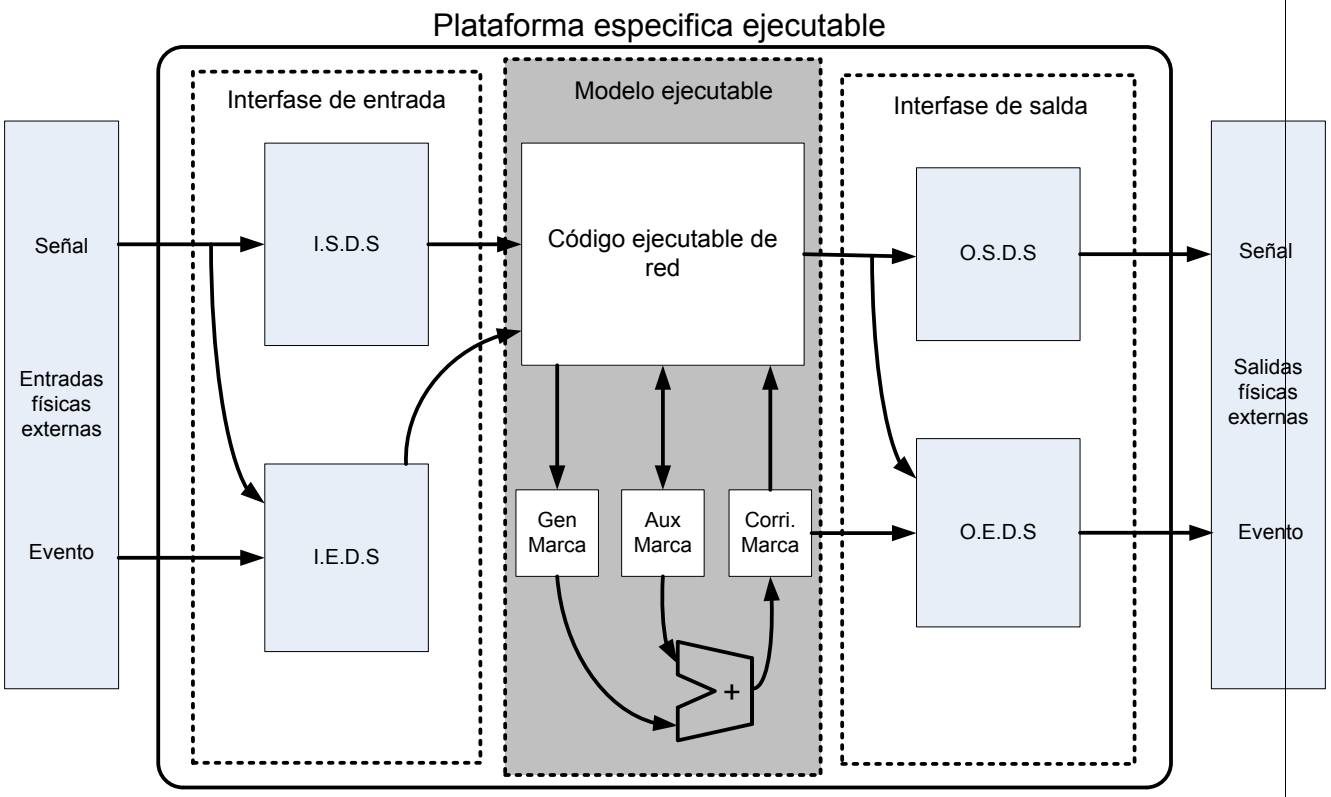


Figura 15: Arquitectura del Framework

El código de la interfaz de entrada define los nuevos acontecimientos mediante la comparación de los valores actuales de las señales con las de la etapa anterior.

De manera similar el código de la interfaz de salida actúa sobre las salidas físicas de la plataforma específica.

Actualmente el código dependiente de la plataforma está escrito manualmente, pero el objetivo del autor es generarlo automáticamente sobre la base de la descripción de hardware de las funcionalidades necesarias.

El algoritmo de ejecución consta de tres partes principales:

1. Lectura de entrada;
2. Procesamiento interno;
3. Actualización de salida.

Partes 1 y 3 son ejecutadas por el código dependiente de la plataforma, mientras que la parte 2 se corresponde con la ejecución de la red.

El comportamiento asociado a cada una de estas tres partes ya se presentó en otros trabajos [34, 47]. Se utilizaron dos estructuras auxiliares que permiten la ejecución coherente de la semántica de RdP:

- El marcado auxiliar, utilizado como una copia inicial del actual marcado, retira los token después de un disparo de una transición; y genera marcado que acumula tokens generados durante una de las etapas de ejecución.

- Al final de la etapa de ejecución, el actual marcado se actualiza con la suma de los registros “Marcado Auxiliar” y genera el nuevo marcado.

En resumen, la ejecución llevada a cabo por la plataforma de implementación opera según las etapas explicitadas en la arquitectura de la Figura 15.

A continuación se presenta una breve descripción de algunos de los algoritmos utilizados por la herramienta de generador de código (sólo para el código C).

La herramienta de generación de código recibe un solo archivo PNML y crea cinco archivos: main.c, functions.h, functions.c, netc.h, y netc.c.

El archivo principal proporciona una interfaz para el usuario. Los archivos incluyen funciones que generan los eventos de entrada de las señales de entrada , las funciones que generan señales de salida de eventos de salida , y las funciones que generan señales de salida sobre la base de marcado actual de la red.

Los archivos Net.c contienen el código de ejecución de red. Incluyen dos funciones principales:

1. **start**, que inicializa todas las variables con los valores iniciales modelo de la RdP ;
2. **run**, que ejecuta un paso de la red.

En la ejecución, un paso de la red, es importante hacer hincapié en:

- Cada lugar está representado por tres variables:
- La primera variable (marcado en la Figura 15) se utiliza para almacenar el número de token en el comienzo de la iteración.
- La segunda variable (auxiliar de marcado en la Figura 15) se inicializa con el número de token en el comienzo de la iteración y es desde donde se eliminan token cuando se consumen, debido al disparo de una transición.
- La tercera variable (marcado generado en la Figura 15) se inicializa con cero y es donde se añaden los token creados.
- Al final de la iteración, se añade la variable “Marcado Generado” a la “Marcado Auxiliar” y es guardado en ¨Marcado Actual¨.
- La estrategia de ejecución de la red adoptada se basa en el análisis y el disparo de las transiciones ordenadas por prioridades conexas (que fueron especificados por el modelador).

Es importante tener en cuenta que las diferentes formas de la lista de transiciones corresponden a diferentes estrategias de resolución de conflictos. El enfoque que se presenta en el uso de prioridades de las transiciones impone a priori la decisión de resolver los conflictos.

Alternativamente, un ordenamiento pseudo-aleatorio significaría una estrategia de resolución justa. Como ya se destacó en [47] , la estrategia se refiere a la resolución de conflictos basada en el orden de prioridades, que según el autor, es muy eficaz para las implementaciones de software .

La estrategia adoptada para la ejecución de pasos, se realiza como una lista de transiciones ordenadas. Las transiciones con mayor prioridad se manejan primero, hasta que los conflictos se resuelven intrínsecamente

A continuación se crean los siguientes elementos: una lista de lugares de origen, una lista de los arcos de origen, una lista de los lugares de destino y una lista de arcos de destino.

Finalmente las condiciones que pretende evaluar para el disparo de la transición incluyen el marcado activado, guarda de la entrada de señal y las evaluaciones de los eventos.

Ejemplo de aplicación
Para ilustrar una aplicación del marco arriba presentado, se utiliza un ejemplo presentado por [102]. El objetivo es producir el controlador para un sistema de automatización integrado por tres coches para el transporte de mercancías (ver Figura 16).

Los coches o vagones se mueven hacia adelante y hacia atrás entre los dos puntos extremos (A y B). Cada vagón tiene su propia trayectoria y velocidad, pero deben empezar a moverse en el mismo momento.

Ellos comienzan a moverse hacia adelante cuando todos están en la posición inicial, los puntos A (i) y se pulsa GO. El movimiento de retroceso se empieza cuando todos los vagones están en su posición final y se pulsa BACK, los puntos B (i) y la parte el retorno se activa.

En este sentido, el controlador del sistema tiene seis sensores de detector de presencia como señales de entrada (tres en la posición inicial A (i) y tres en la posición final B ((i)). Así como Go (para iniciar el movimiento hacia adelante) y BACK (con el fin de iniciar el movimiento hacia atrás).

Como salidas, el controlador tendrá dos conjuntos de señales: una de ellas para controlar los motores de los tres vagones (M (i)), y la otra que indica la dirección del movimiento (Dir (i)), como se muestra en Figura 16.

Con el fin de utilizar las RdP como un lenguaje de especificación de sistema, se presenta el modelo de RdP IOPT en la Figura 17. Este modelo puede ser utilizado para generar automáticamente el código de aplicación para el controlador del sistema.

Sin embargo, si el objetivo es obtener un controlador distribuido, es necesario dividir el modelo RdP IOPT en tres sub-modelos, cada uno de ellos para ser desplegado en un controlador local instalado en cada vagón.

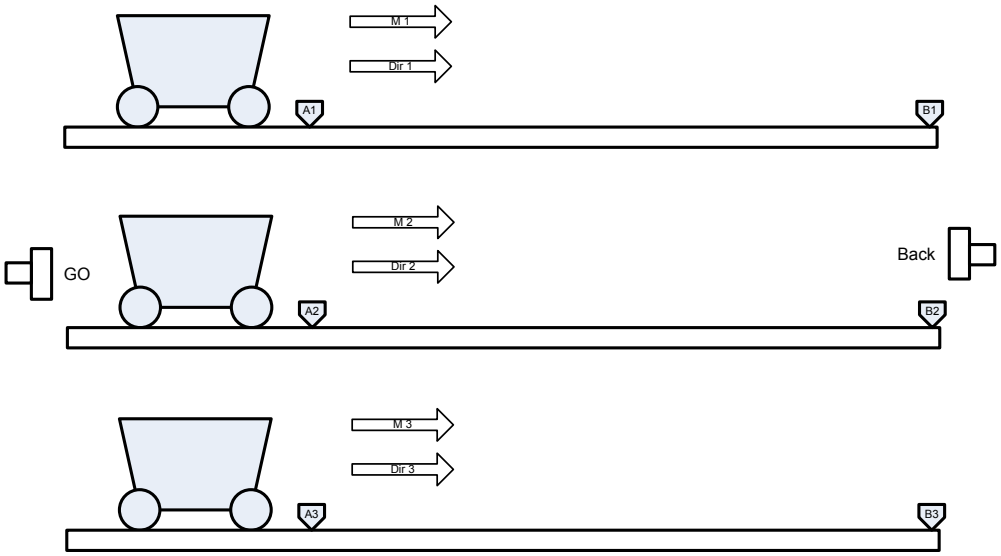


Figura 16: Ejemplo de aplicación: sistema de automatización para el control de tres vagones.

Para lograr este objetivo, podemos utilizar la operación división y herramienta dividir asociados, como ya se ha mencionado en [134].

La Figura 17 identifica la transición de los nodos GO y BACK para ser utilizado como el corte de los conjuntos al aplicar la operación de división.

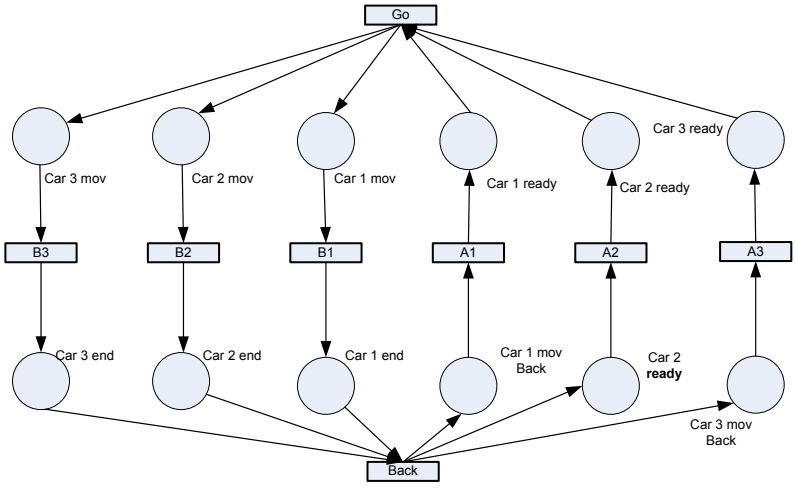


Figura 17: Modelo de red IOPT de la aplicación de ejemplo

Después de aplicar la operación de división, se generan tres sub- modelos, tal como se presenta en la Figura 18. Cada sub-modelo tiene algunas señales dedicadas de entrada y salida (asociados con su propio medio ambiente), así como un conjunto de eventos de entrada y de salida para asegurar la comunicación con otro sub-modelo. Esta comunicación entre los sub-modelos está asegurada a través de canales de comunicación sincrónica dirigidos. Estos canales de comunicación aseguran el disparo sincrónico de un conjunto de transiciones; una de las transiciones en el conjunto involucrado en cada canal de comunicación recibe el atributo de "amo" (master) y es responsable de generar un evento de salida; otra transición que participa en el mismo canal de comunicación recibirá el atributo "esclavo" (slave) y es receptivo al evento de entrada generado por la transición principal.

En la Figura 18 cada canal de comunicación tiene un número asociado único.

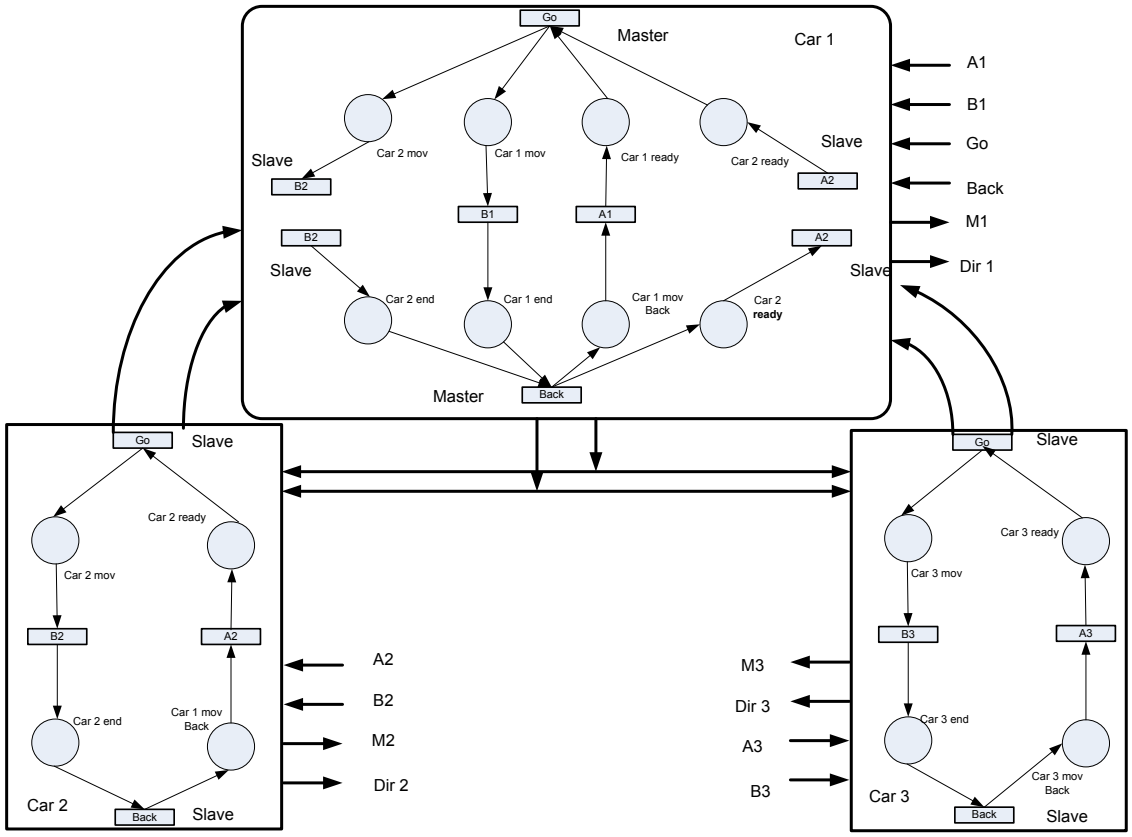


Figura 18: Tres sub-redes conectadas por canales de sincronización dirigidos

El código C asociado con los modelos presentados, tanto para la ejecución centralizada (Figura 5), como para la ejecución distribuida (Figura 18), se generó correctamente usando el generador de código automático.

Hay tres tipos de controladores: uno asociado con el controlador centralizado (sólo un controlador para los tres coches), otro asociado con el coche "maestro ", y el tercero asociado con los coches "esclavos" para el controlador distribuido.

El número de líneas de código generado automáticamente para el controlador centralizado es de 177 líneas (34 líneas de inicialización y 143 de la ejecución). Por otro lado, el número de líneas código generado automáticamente para los controladores distribuidos es de 75 líneas (16 líneas de inicialización y 59 de ejecución) a los coches "esclavo" (coche 2 y 3 pulgadas de coches Figura 17) y 121 líneas (4 líneas de inicialización y 97 de ejecución) para "master" el coche (1 en la Figura 18), respectivamente.

Análisis de resultados

Escribir código a mano puede mejorar y optimizar un sistema específico. Se produce mejor rendimiento en el código de máquina resultante que el producido por herramientas automáticas. Sin embargo, puede tener muchas desventajas, tales como: requiere un número mayor de desarrolladores con una mejor experiencia, un tiempo de desarrollo más amplio, se torna más

susceptible a errores y es difícil de mantener. Podemos compararlo con un lenguaje de bajo nivel, se comporta mejor pero es más difícil de desarrollar.

Por otro lado, la generación automática de código tiene ventajas importantes, el código generado siempre aumenta en calidad con el tiempo, es coherente en su estructura y reglas de estilo, y puede ser producido en una fracción de tiempo.

Al comparar el rendimiento del código escrito a mano y el código que se genera automáticamente es un arma de doble filo. El código producido a mano puede sobresalir en una plataforma específica pero no se adapta fácilmente a una nueva plataforma.

El factor de optimización es especialmente relevante en los sistemas integrados, donde los recursos de memoria y velocidad de ejecución son limitados, lo que requiere un delicado equilibrio entre los mismos.

En las herramientas de generación de código los factores de optimización y la compatibilidad con diferentes plataformas fueron tenidas como objetivos fundamentales. Se utilizaron varias técnicas de optimización, para la exploración de diferentes factores relacionados con la clase RdP IOPT y sus propiedades, opciones del compilador, y las especificidades para la plataforma de implementación. Como ejemplo, se puede hacer referencia a la dicotomía entre el uso de la memoria de datos y el código considerando las especificidades de la plataforma de implementación.

Es importante tener en cuenta que una característica importante del uso de herramientas de generación de código es que es fácil de probar diferentes factores de optimización sin demasiado esfuerzo, tiempo y coste.

Con el creciente número y complejidad de los sistemas de desarrollo para plataformas embebidas, los lenguajes de modelado y las herramientas, el autor busca ampliar con nuevas herramientas mejorar el Ciclo de vida del desarrollo de sistemas (SDLC).

El Lenguaje de modelado de sistemas (SysML) es una notación optimizada para ingenieros de sistemas. Fue creado para mejorar las comunicaciones en todo el SDLC, aumentar la reutilización de los diseños, y que los costos de mantenimiento fueran más bajos [135]. El uso de herramientas de modelado, simulación y ejecución permitiría la verificación inicial de los diseños. Esto traería como consecuencia el aumento de la calidad del software y la reducción de los riesgos en el desarrollo de sistemas.

Los esfuerzos futuros se verán impulsados para mejorar el sistema que se presenta con una herramienta de conversión de modelo a modelo, de SysML a RdP IOPT. Con esta mejora, será posible generar un código de SDLC, permitiendo así la verificación del modelo, la simulación y la ejecución. Ese esfuerzo de desarrollo será apoyado por Eclipse Modeling Framework (EMF), que implementa la mayor parte del Model-Driven Arquitectura (MDA), sólo en especificaciones básicas.

EMF se puede utilizar para soportar una amplia variedad de aplicaciones, incluyendo soporte SDLC, transformaciones de modelos y la integración de sistemas. Esta plataforma es un fuerte candidato para integrar muchas de las herramientas desarrolladas y nuevas funcionalidades en las transformaciones de modelos.

El flujo de desarrollo presentado se basa en el desarrollo dirigido por modelos donde las RdP IOPT tienen el papel principal.

Se han desarrollado varias herramientas. Estas incluyen un editor de RdP IOPT, herramientas que permiten composiciones y descomposiciones de modelo, y varios generadores de código automático. La base común de esas herramientas es la representación PNML para los modelos de RdP IOPT. Esta es una de las ventajas más importantes de este método de desarrollo de sistemas embebidos. La representación PNML proporciona la base para los generadores de código automático y su interoperabilidad con un conjunto de otras herramientas dedicadas al análisis, diseño, verificación, simulación e implementación en plataformas específicas.

Este trabajo muestra las ventajas y la facilidad con que se escribe un programa paralelo si se hace uso de RdP IOPT. La sobrecarga introducida por el software de ejecución de la red es importante, puesto que la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones y el disparo. No permite integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 47, Tabla 48 y Tabla 49.

Tabla 47: Aportes de[50] con respecto a la ejecución

Con respecto a la ejecución							
Referencia [50]	EHS No	EHP No	RIS Si	DU Si	DM Si	DIR No	DIS No

Tabla 48: Aportes de [50] con respecto a la programación

Con respecto a la programación de la red						
Referencia [50]	PD No	LE No	PPSR Si	LEP Si	PHTR No	PSTR No

Tabla 49: Aportes de [50] con respecto a la división de una red

Con respecto a la división de la red										
Referencia [50]	DA No	DT No	DP Si	SEA Si	RPD Si	CMTR No	RD Si	RC Si	EMTR No	RG Si

Diseño de un Controlador usando Stretching en RdP Temporizadas para evitar Interbloqueo

Cabe destacar que este trabajo es esencialmente una extensión del presentado en [136], “Supervisory Controller Design for Timed Petri Nets”.

El método llamado "Stretching” (estiramiento) extiende las RdP a RdP temporizadas. Éstas pueden ser tanto RdP con transiciones controlables como incontrolables. En esta red el estado del temporizado de la RdP se puede representar sencillamente facilitando el diseño de un sistema de supervisión de una RdP temporizada para cualquier propósito.

En este trabajo [137] se diseñó un controlador de supervisión para evitar el bloqueo. Con este método se ha diseñado un controlador para RdP Stretching. Luego, se ha usado este controlador, para obtener un controlador temporizado para la RdP original. En este trabajo, se presentan los

algoritmos para la construcción de los conjuntos de accesibilidad de las RdP Stretching y originales. También, se exponen los algoritmos para obtener el controlador para el cronometrado de las transiciones en la RdP original. Estos algoritmos se han implementado utilizando MATLAB. También, se presentan ejemplos para ilustrar el enfoque introducido.

Secciones de interés del trabajo
Muchos de los sistemas de hoy, y especialmente los sistemas que están compuestos de numerosos subsistemas de diferente naturaleza (es decir, sistemas compuestos por sistemas), son demasiado complejos para ser modelado por ecuaciones diferenciales simples. Estos sistemas pueden ser descritos mejor por ocurrencia de eventos. En tales sistemas, que se llaman sistemas de eventos discretos (DES), la aparición de un evento puede desencadenar la aparición de otros eventos. Una forma de modelar DES es mediante el uso de RdP. Utilizando un modelo de RdP, la dependencia de los acontecimientos según los eventos puede ser descrito fácilmente.

Aunque las RdP se introdujeron primero sin la noción de tiempo, se demostró que el uso de tiempo puede ser necesario para modelar cierto tipo de sistemas DES. Por lo tanto, las RdP temporizada (RPT) se introdujeron para modelar este tipo de sistemas. Por ejemplo: el tiempo puede estar asociado con las transiciones, lugares o arcos de una RdP.

Sin embargo, en DES mayoritariamente el tiempo se usa en la ocurrencia de eventos, por lo que es natural asociar el tiempo con las transiciones de una RdP, ya que estas representan los eventos. Además, el tiempo asociado a las transiciones se puede representar en tres formas básicas: duración, inmediato o intervalos.

En este trabajo, asocia el tiempo con transiciones y considera las duraciones de los disparos, ya que el autor sostiene que el tiempo transcurrido para un evento puede ser mejor descrito por una duración del disparo. Más aun, supone que una vez que la transición se ha habilitado, se puede disparar en cualquier momento (no necesariamente de inmediato), siempre y cuando se mantenga habilitada. Por otra parte, no sólo considera el disparo de las transiciones individuales en un momento dado, sino que también el disparo simultáneo de más de una transición, siempre y cuando estén activadas simultáneamente y no existan conflictos.

Puede que sea necesario diseñar un sistema de supervisión de una RdP para evitar comportamientos no deseados, tales como bloqueo, o para hacer cumplir el comportamiento deseable, como vivacidad, acotación, o reversibilidad. Muestra dos principales enfoques de control para diseño del controlador de RdP, que son:

1. Los estados prohibidos;
2. El enfoque estructural.

En el enfoque de estados prohibidos, hay que construir el conjunto de accesibilidad de la RdP, lo que podría llevar mucho tiempo.

En el enfoque estructural, por lo general se encuentra un controlador “conservador” en lugar de los estados prohibidos, lo que garantiza el diseño del controlador menos restrictivo en la mayoría de los casos. Cabe señalar que, los enfoques de diseño descentralizados se han desarrollado para disminuir las dificultades de construcción de accesibilidad cuando se acercan los estados prohibidos.

En este trabajo, utiliza la cercanía a los estados prohibidos en el diseño del controlador, para evitar el bloqueo.

El trabajo de diseño de control de supervisión de redes se ha iniciado hace relativamente poco. Un enfoque de la síntesis de control para RPT basado en invariantes de predicado especificado fue introducido en [138], política de búsqueda hacia adelante han sido definidas en [139]. Mientras que en [140] se realiza un análisis de planeación. Un controlador basado en observador fue presentado en [141] y [142]. Para diseñar e implementar un controlador para cualquier sistema en general se requiere del conocimiento del estado del sistema. Para una RdP sin límite de tiempo, el vector de marcado, que dá el número de token en cada lugar, es suficiente para representar el estado del sistema. Para una RdP con duraciones de disparo, sin embargo, aparte del vector de marca, también es necesario conocer el estado de los token que se encuentran en transición. En el caso de que se sostenga y se permita las duraciones o intervalos, el estado de los token está residiendo en un lugar y debe ser conocido. Esta propiedad hace que sea muy difícil diseñar un sistema de supervisión de la RdP con duración, en comparación con una RdP sin tiempo. Un enfoque, llamado estiramiento, recientemente se ha introducido para superar esta dificultad [136].

En [136] , sin embargo , se asumió que todas las transiciones de la RdP son controlables. En este artículo, primero se amplía el enfoque de estiramiento para las RdP con duración que pueden tener transiciones tanto controlables como incontrolables. Consecuentemente, se propone diseñar un sistema de supervisión para estas RdP, con el fin de evitar el bloqueo.

El enfoque propuesto para el controlador es el menos restrictivo, y cuando existe, se garantiza la prevención de interbloqueo. Para la implementación del controlador, considera ambos casos cuando las transiciones son incontrolables, observables y no observables por el controlador.

Análisis de resultados
En este trabajo el autor ha ampliado el método de estiramiento, que se introdujo por primera vez en [136] para simplificar la representación del estado de un RdP, a las RdP con duración que pueden tener transiciones tanto controlables como incontrolables

Esta representación permite modificar fácilmente la mayoría de los métodos de diseño de controlador de supervisión que fueron desarrollados para RdP sin tiempo para su aplicación a las RdP con tiempo. De hecho, todos los algoritmos presentados en este trabajo son versiones de la modificación de los algoritmos desarrollados previamente en [143], para las RdP sin tiempo.

Con el uso del estiramiento se ha presentado un enfoque para diseñar un sistema de supervisión de la RdP con tiempo para evitar el bloqueo. El enfoque que se presenta determina el controlador menos restrictivo que garantiza prevención de interbloqueo, siempre que exista un controlador de este tipo.

Aunque el autor ha considerado sólo el diseño del controlador de supervisión para evitar el bloqueo, el enfoque de estiramiento también se puede utilizar para diseñar controladores para otros fines, tales como para hacer cumplir la vivacidad de la conexión, la acotación, y / o reversibilidad. También puede llevarse investigaciones para utilizar este enfoque en el diseño de controladores descentralizados para la RdP con tiempo mediante descomposiciones y superposición [144] u otros medios.

Las redes, consideradas en este trabajo, son aquellas en donde la duración del disparo está asociada con las Transiciones.

Este trabajo muestra los fundamentos para controlar un RdP con tiempo haciendo uso de los eventos con transiciones observables y no observables. También expone que es posible diseñar un controlador (si existe) sólo conociendo el marcado de la red; y si la RdP es con tiempo se requiere también conocer el estado de los temporizadores.

Pero no realiza el controlador o un procesador programado por hardware o software, que role a o use la RdP como programa. Las características que se implementan en [137] son mostradas en las Tabla 50 y Tabla 51.

Tabla 50: Características del tipo de redes implementadas en el PP, en el trabajo de [137]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[137]	Si	No	No	No	No	No	No	No	No	No	No	No	No

Tabla 51: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [137]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[137]	No	No	No	No	No	No	No	No

Aplicación a Programas de Controlador Lógico con Modelo de RdP Coloreada

En este documento [88] se introduce una nueva clase de RdP , denominada “Interpretada Control de color RdP” (CCIPN). Se muestran las ventajas de la modelización con CCIPN para la especificación de los programas de “Logic Controller”.

Se combinan las RdP coloreadas interpretadas (tratadas como un objeto) con la parte de control orientado a objetos. Además de varios artículos, se especifican los token de colores como objetos compuestos que son evaluados dinámicamente, como una parte del objeto. La red se construye a partir de los objetos más simples que están relacionados con entidades VHDL, que se componen por una biblioteca definida por el usuario. Los objetos formales que describen parte de un sistema electromecánico controlado están relacionados con los token de colores de la RdP.

Las RdP coloreadas se componen por enlaces que describen los objetos que se combinan con una RdP de color que especifica el controlador lógico. Ellos se refinan y se combinan en un sistema de RdP completa (modelo total). Puede ser simulado, verificado y sintetizado en el entorno VHDL, colaborando con estas herramientas de diseño experimental.

Esta técnica se ilustra mediante la presentación de una solución al problema del " llenado del tanque". Los métodos propuestos son especialmente útiles en el diseño que tienen aplicación en la industria con “Controlador Lógico” específicos realizados con FPLD (FPGA y CPLD).

Secciones de interés del trabajo

Teniendo en cuenta el creciente interés en las RdP y las técnicas para el desarrollo de controladores industriales con lógica programable, el autor presenta un método de construcción de programa VHDL utilizando el tipo especial de RdP como punto de partida. Se recurre principalmente a los estados locales de sistemas distribuidos y a las transiciones locales entre ellos, en términos de que un concepto de sistema puede ser representado formalmente. Donde cada fase de la actividad de un sistema se asocia con el estado compuesto estructurado en una RdP coloreada.

El controlador que interpreta CPN está evidentemente enfocado a las red de alto nivel con token de colores, introducido por Jensen [92] . Aparte de eso, tiene las ventajas que las RdP son de uso popular en ingeniería, tales como por ejemplo control de las RdP o Grafcet.

Un modelo discreto de controlador lógico se deriva de la RdP y se implementa por ejemplo como una FPGA o un programa virtual de controlador de lógica en programación estructurada [145] .

Una de la principal contribución de este trabajo es el uso de la misma representación gráfica para describir los datos del diseño:

- Para la síntesis de alto nivel y el modelado con VHDL y síntesis automática de VHDL en ASIC (FPLD y FPGA) [90];
- Manual para el diseño de un soporte de herramientas de síntesis lógica (ALTERA o Ciprés, subconjuntos de VHDL o propietarias FPGA. Lenguajes de descripción de hardware, como CUPL o PALASM).

La forma de diseño es experimental. Con las herramientas de síntesis de alto nivel y después del refinamiento con las herramientas de síntesis de la lógica en la misma parte del diseño en un entorno de desarrollo VHDL.

El objetivo de este trabajo es presentar y demostrar un nuevo enfoque de CPN para el diseño del controlador lógico utilizando la coloración del Control de la RdP (un abstracción formal de Grafcet , Grafchart [146] o diagrama de función secuencial) y orientado a objetos como parte controlada.

La aplicabilidad del enfoque se demuestra con el ejemplo presentado por primera vez en [147] y desarrollados en los trabajos de los autores, por ejemplo, en [148] . Ha sido adoptado por otros autores como una ilustración de varias ideas diferentes y metodologías de diseño, por ejemplo en [148] [[147] [149]

Control Interpretado de RdP
El control interpretado de RdP cumple con las siguientes cuatro restricciones:

- Es seguro;
- No mide el tiempo;
- Es determinista;
- Sus posibilidades de entrada y de salida han sido extiendas para ser homogénea con los de la Grafcet.

Análisis de resultados
Es un proyecto en que se han usado la CPN, restringiendo el tipo, para dirigir la ejecución de un programa. Desde el punto de vista de los patrones y componentes para generar código VHDL, el aporte es importante. Las características que se implementan en [88] son mostradas en las Tabla 52 y Tabla 53.

Tabla 52: Características del tipo de redes implementadas en el PP, en el trabajo de [88]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[88]	Si	No	No	No	No	No	No	No	No	No	No	No	No

Tabla 53: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [88]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[88]	No	No	No	No	No	No	No	No

Caso de RdP Entrada-Salida Plaza-Transición (RdP IOPT) y Herramientas asociadas

Las RdP son un formalismor conocido y ampliamente utilizado en diferentes áreas de aplicación. Sin embargo, la falta de herramientas adecuadas que se puedan integrar dentro de los marcos de desarrollo de ingeniería es un gran inconveniente. En este trabajo [46], se identifican, justifican y se hace uso de las características de las RdP para dirigir el diseño con modelado de sistemas de automatización y sistemas integrados, haciendo uso de RdP IOPT introducidas en [47], y su representación mediante e lenguaje "RdP Markup Language" (PNML).

Este trabajo también presenta un conjunto de herramientas asociadas, que están en fase de desarrollo. Esta clase de RdP IOPT, es el vínculo común a través de un conjunto de herramientas en desarrollo, incluyendo un editor gráfico, un analizador de espacio de estado para la verificación de las propiedades, la resolución de conflictos a través de generación automática del árbitro, generadores de código automático y simulador, entre otros. En este sentido, el objetivo principal de la clase de RdP propuesta y las herramientas asociadas es soportar todo el flujo de desarrollo del sistema, desde la especificación hasta la implementación.

Análisis de resultados
Este trabajo es una aplicación del trabajo realizado en [44], y muestra como dirigir, implementar y animar un desarrollo dirigido por un modelo RdP IOPT.

También, se hacen comparaciones con otros desarrollos similares.

Este trabajo [46], muestra cómo se escribe un programa paralelo si se hace uso de RdP IOPT. También, muestra cómo se amplía la capacidad del sistema para comunicarse con el entorno físico, por haber agregado un nuevo tipo de evento. Lamentablemente no se hace uso de programación por componentes y la posibilidad de programar distintas instancias en un componente envolviendo al actuador y/o sensor.

La sobrecarga introducida por el software de ejecución de la red es importante, puesto que la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones y el disparo y no permite integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 54, Tabla 55 y Tabla 56.

Tabla 54: Aportes de[46] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[46]	No	No	Si	Si	Si	No	No

Tabla 55: Aportes de [46] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[46]	No	No	Si	Si	No	No

Tabla 56: Aportes de [46] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[46]	No	No	Si	Si	Si	No	Si	Si	No	Si

Aplicación de un Controlador de RdP Adaptiva en Tiempo de Ejecución

El tiempo de reacción de un controlador es una cuestión fundamental en los sistemas de control de eventos discretos. Las RdP se utilizan ampliamente en este campo. El controlador lee las entradas, ejecuta el control de la RdP y escribe la salida de una manera cíclica. El tiempo de reacción de este controlador depende de la estructura de la RdP, en la secuencia de eventos y en el algoritmo que ejecuta la red. Con el objetivo de minimizar el tiempo de reacción, el autor de este trabajo [150] decidió diseñar un controlador supervisor, que ha denominado controlador de tiempo de ejecución (ETC). El objetivo de la ETC es determinar más rápido, en tiempo real, quien ejecuta el algoritmo de disparo y cambiar el algoritmo de ejecución cuando sea necesario. En el caso de los sistemas de control, esto minimiza el tiempo de reacción del controlador y también minimiza la potencia consumida por el controlador. Una posible aplicación de la técnica es la minimización del tiempo de ejecución de los programas de autómatas programables desarrollados en diagrama de función secuencial (SFC).

Secciones de interés del trabajo

Las RdP son un formalismo muy adecuado para el modelo de sistemas de eventos discretos concurrentes. Se han obtenido respuestas satisfactorias en aplicaciones en campos como las redes de comunicación, sistemas de computadora, sistemas de fabricación de piezas discretas, etc. Los modelos con RdP son a menudo considerados como especificaciones auto-documentados, debido a que su naturaleza gráfica facilita la comunicación entre los diseñadores y los usuarios. Las RdP tienen una fuerte base matemática que permite la validación y la verificación con una amplia gama de propiedades. Además, estos modelos son ejecutables y pueden ser utilizados para la animación y simulación del comportamiento del sistema. También para fines de supervisión, una vez que el sistema está trabajando. El sistema final se puede derivar del modelo de una RdP por medio de técnicas de implementación en hardware y software (generación de código). En otras palabras, se pueden utilizar durante todo el ciclo de vida de un sistema.

En los últimos 25 años, los investigadores han dedicado considerable atención a la implementación de software de RdP ; véase por ejemplo [151] [109] [108]. La implementación de procesos se obtiene como la traducción de un modelo. De un sistema expresado por una RdP, a un sistema real que tiene el mismo comportamiento que el modelo. Una aplicación de software es un

programa que desencadena el disparo de las transiciones de la red. La observación de la evolución de las reglas de marcado juega un papel simbólico en el software. Dependiendo de los criterios, una implementación de RdP puede ser clasificada como compilada o interpretada, secuencial o concurrente y centralizada o descentralizada.

En una implementación centralizada, el papel de los token es ejecutado por una sola tarea, lo que comúnmente se llama el coordinador. Las acciones asociadas se pueden distribuir entre un conjunto de tareas para garantizar la concurrencia expresada por la red (véase, por ejemplo [152] [153] [154]). Por lo tanto, el coordinador actúa como el núcleo del sistema operativo de una aplicación multitarea. En este tipo de aplicación el algoritmo determina qué transición esta habilitada y así determina su disparo en forma determinística. Varios algoritmos han sido propuestos en la literatura como el de fuerza bruta, plaza dividida o transición dividida.

Un análisis de algoritmos de aplicación centralizados se realiza en [155], Brute Force (BF), Enabled Transitions (ET), Static Representing Places (SRP) y Dynamic Representing Places (DRP).Analizando todos los algoritmos y las ideas principales obtenidas en [155]:

La implementación de las transiciones habilitadas, representaciones estáticas y dinámicas de lugares puede conducir a un enorme ahorro en el tiempo de ejecución en comparación con el algoritmo de fuerza bruta.

Si la representación estática de plazas elige las plazas adecuadas, el rendimiento es similar o mejor que la representación dinámica de plazas.

La elección del tipo más adecuado de algoritmo para ejecutar una RdP depende del comportamiento de la RdP (concurrencia efectiva frente a los conflictos efectivos).

En conclusión, el mejor algoritmo para implementar una RdP depende de su estructura y de su comportamiento dinámico (acciones y eventos). La habilitación de las transiciones es más simple en redes con pocos conflictos o con conflictos de tamaño pequeño.

La representación de lugares es mejor en redes con un alto número de conflictos o con conflictos de tamaño medio- grande.

En este trabajo se ha desarrollado una técnica que permite la elección en tiempo de ejecución del algoritmo más adecuado para ejecutar una RdP de acuerdo con el comportamiento observado en cualquier momento. Con este objetivo en mente, se diseñó un controlador supervisor, que se ha denominado controlador de tiempo de ejecución (ETC). El objetivo de la ETC es determinar en tiempo de ejecución qué algoritmo se ejecuta más rápido y cambiar al algoritmo de ejecución adecuado cuando sea necesario.

En el caso del control del sistema, esto minimiza el tiempo de reacción del controlador y también la potencia consumida por el controlador. Una aplicación de la técnica es la minimización del tiempo de ejecución de los programas de controladores lógicos programables desarrollados en lenguaje SFC.

Implementación de RdP Centralizada

Las técnicas de implementación centralizadas [153] están codificadas en una tarea llamada la coordinadora . Lo importante de la caracterización de una implementación centralizada es determinar qué transiciones se activan y poder dispararlas.

Varias soluciones están disponibles para la reducción de los costos de la prueba que determinan la sensibilidad de la transición y, posteriormente, la sobrecarga introducida por el coordinador. Dependiendo de la solución elegida, las técnicas de implementación centralizadas se pueden clasificar en cualquiera de las siguientes clases:

- Enfoques de fuerza bruta. Todas las transiciones son evaluadas para el disparo. En este enfoque, las RdP, se convierten a los lenguajes de programación de controladores lógicos programables como instrucciones.
- Modelo de Evolución de Tránsito Diferido (DT)[156], este algoritmo incluye solamente la prueba de las transiciones que descienden de los lugares marcados, mejorando el funcionamiento del algoritmo de fuerza bruta.
- Para el algoritmo de representación de lugares estáticos y dinámicos. Sólo las transiciones de salida de los lugares marcados se evalúan, para determinar si están sensibilizadas [153]. Cada transición está representada por uno de sus lugares de entrada, llamado "Plaza Representando" y los lugares de entrada restantes se denominan lugares de sincronización. Sólo las transiciones cuyo lugar representando está marcado se consideran como candidatos para disparar.
- Enfoque de transición impulsada. Una caracterización de la habilitación de las transiciones, que no sean el marcado, es suministrada. Y sólo las transiciones totalmente habilitados son consideradas. Este tipo de técnica es estudiada en trabajos como [157] .

Como se dijo en la introducción, se llevó a cabo un análisis de las características de los algoritmos de implementación centralizados de RdP en [155], donde se ilustra que las transiciones habilitadas y la representación de lugares estáticos son los algoritmos que proporcionan mejor rendimiento. Ambos reducen el tiempo de ejecución, en la optimización de la búsqueda de transiciones habilitadas.

En el presente trabajo se ha implementado varios algoritmos que desarrollan las diferentes técnicas de búsqueda de transición habilitadas:

- Fuerza Bruta.
- El modelo de evolución de tránsito diferido.
- El modelo de evolución de tránsito Inmediata.
- Representación Lugares Estática.
- Representación Lugares Dinámico.
- Transiciones Activadas.

ETC ha sido desarrollado teniendo en cuenta todos estos algoritmos. Sin embargo, en este trabajo se presentan los resultados de los mejores que son : ET y SRP [155].

Ejecución del controlador
La elección del tipo más adecuado de algoritmo para ejecutar una RdP depende del comportamiento de la RdP (concurrencia efectiva frente a los conflictos eficaces). El algoritmo de transiciones habilitadas es mejor en redes con pocos conflictos o con conflictos de tamaño pequeño. El algoritmo de representación de lugares es mejor en redes con un alto número de conflictos o con conflictos de tamaño mediano o grande [155].

Con el fin de minimizar el tiempo de ejecución de la RdP, se ha propuesto una implementación adaptativa que elige el mejor algoritmo para ejecutar la red. Nos referimos a esta solución como el controlador de tiempo de ejecución (ETC). La función principal de la ETC es determinar en

tiempo real qué algoritmo ejecuta la RdP más rápido. La ETC ejecuta el algoritmo elegido y calcula el tiempo de ejecución de los otros algoritmos no ejecutados, con el propósito de elegir el mejor algoritmo que ejecuta el sistema controlado. Si es necesario, el ETC cambia el algoritmo.

Evaluación técnica

Plataforma de test

Una biblioteca de RdP ha sido desarrollada para la realización de las pruebas. La biblioteca se basa en ocho modelos de base, que puede ser instanciados en diferentes configuraciones con un parámetro. Algunos de estos modelos son bien conocidos y se utilizan con frecuencia en la literatura.

La biblioteca comprende las siguientes redes:

En la Figura 19 se presenta una RdP con un proceso secuencial (SEQ), con uno a cien estados (n)

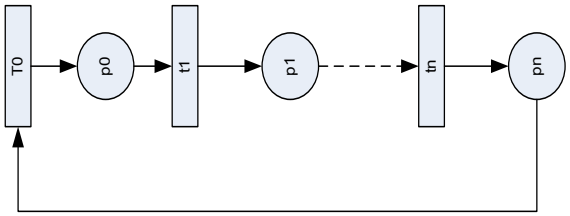


Figura 19: SEQ, RdP con un proceso secuencial

En la Figura 20 se presenta una RdP con uno a cien procesos secuenciales, con 2 lugares por proceso (PAR).

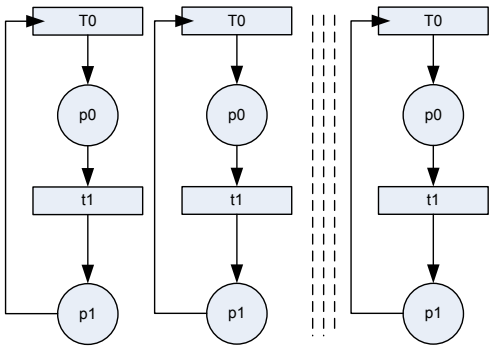


Figura 20: PAR, RdP con uno a cien procesos secuenciales

En la Figura 21 se presenta una RdP con uno a cuarenta procesos secuenciales, con 2 estados cada uno y un recurso común (PR1). Estos pertenecen a la clase s3pr net [158].

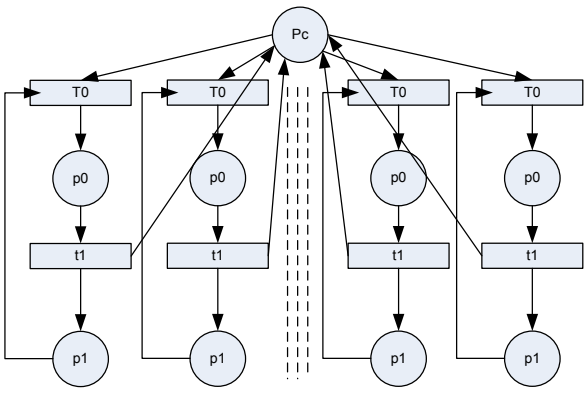


Figura 21: PR1, RdP con procesos secuenciales y un recurso común

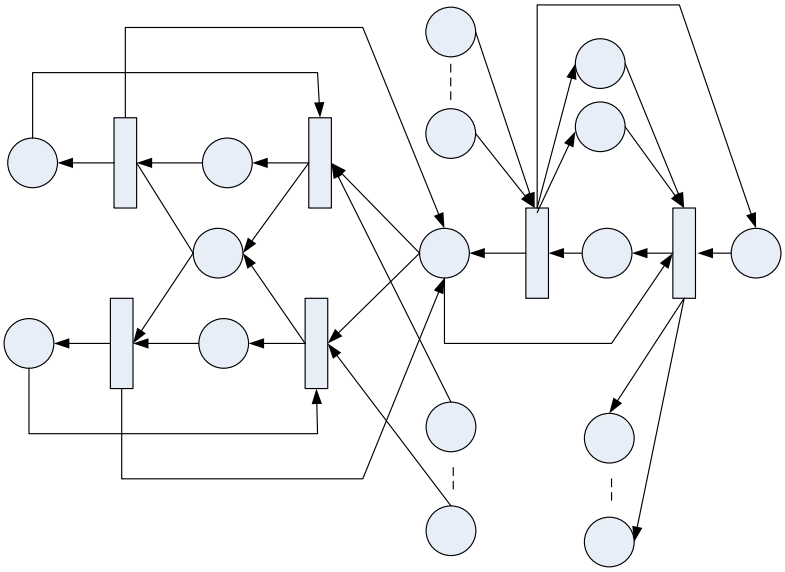


Figura 22: DB, RdP de DB

La Figura 22 representa una RdP de una DB [159].

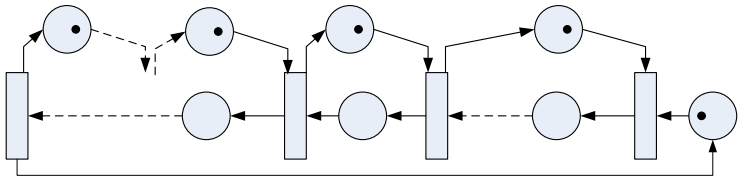


Figura 23: P1R, RdP con procesos secuenciales

La RdP de la Figura 23 es una RdP con 1 proceso secuencial y r (1 …40) recursos (P1R). Estos pertenecen a la clase S3PR net [158].

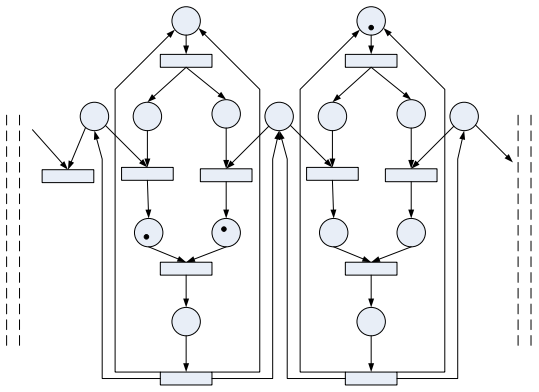


Figura 24: PH, cena de los filósofos modelada con una RdP

La Figura 24 muestra la cena de los filósofos [160] modelada con una RdP (PH) de cinco a cincuenta filósofos.

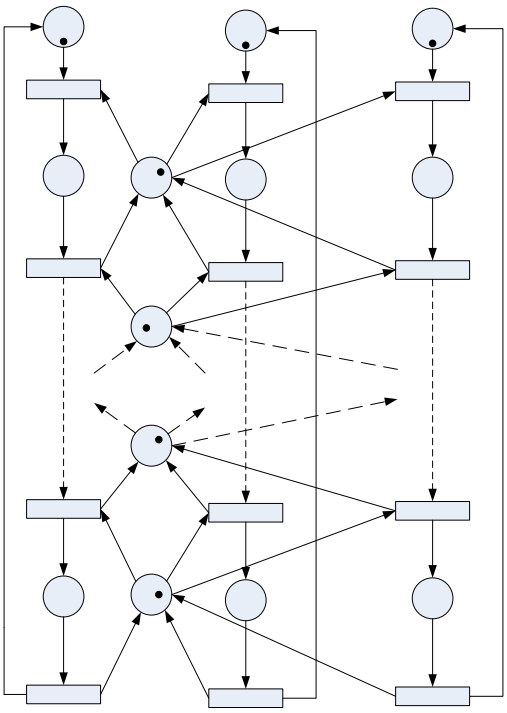


Figura 25: SQUARE, RdP con múltiples procesos secuenciales y recursos compartidos

La Figura 25 muestra una RdP de cinco a quince procesos secuenciales de r estados y r1 recursos comunes (definido por los autores) (SQUARE).

RdP de cinco a sesenta y dos procesos secuenciales de 6 estados y 5 recursos comunes (que no se muestra) (PR5) [158].

Se han implementado las técnicas presentadas en este trabajo en el lenguaje Java utilizando Java con extensión a tiempo real [161] y siguiendo algunas ideas presentadas en [162] [154] . En estas implementaciones, se utilizó el Virtual Machine v2.7 JamaicaVM Tiempo Real Java [163]. El hardware es un ordenador personal con procesador Pentium IV a 1,7 GHz, ejecutando Red Hat Linux 2.4.

Análisis de resultados
En este trabajo se ha desarrollado un algoritmo de disparo para una RdP de tipo adaptativo. El controlador, en tiempo de ejecución, permite elegir en tiempo real el algoritmo más adecuado para ejecutar una RdP. La función principal de la ETC determinará qué algoritmo se ejecuta más rápido. La técnica propuesta es analizada con los dos algoritmos más importantes (desde el punto de vista de rendimiento): las transiciones habilitadas y los lugares.

Sin embargo, el ETC puede trabajar con cualquier algoritmo de aplicación RdP centralizado.

El ETC ejecuta el algoritmo seleccionado y calcula el tiempo de ejecución de otros algoritmos no ejecutados, para decidir el mejor en línea. La ejecución de una RdP sin un algoritmo adecuado puede conducir a un significativo aumento del tiempo de ejecución, junto con una respuesta menos satisfactoria y más lenta en aplicaciones de control. La técnica se ha probado con varias estructuras de red y comportamientos.

Por otra parte el ETC también ha sido probado en una aplicación de control real. La técnica presenta una alta tasa de éxito en la elección del mejor algoritmo de aplicación. La ejecución de la

ETC puede conducir a un enorme ahorro en el tiempo de ejecución. En un ejemplo presentado, el ahorro es de 37 %, pero con otras redes el ahorro puede ser de hasta 70 %.

Por otro lado, el ETC permite una reacción más rápida en los sistemas de control basados en RdP y también minimiza la potencia consumida por el controlador. Una posible aplicación de la técnica es la minimización del tiempo de ejecución de los programas de autómatas programables desarrollados en diagrama de función secuencial.

El autor propone futuros trabajos, que se concentrarán en las siguientes dos ideas:

1. La incorporación de nuevos algoritmos de ejecución RdP en ETC.
2. La mejora de la estimación en tiempo real de los algoritmos de tiempo de ejecución.

Este trabajo [150] muestra y fundamenta la ejecución de las RdP para sistemas de tiempo real, evalúa los algoritmos de disparo y determina cúal es mejor. Pero no hace referencia a la implementación de los tipos de redes y como realizar los disparos. Hay que considerar que la evaluación está realizada para sistemas que se ejecutan en procesadores tradicionales y se basan en lazos.

Las características que se implementan en [150]son mostradas en las Tabla 57 y Tabla 58.

Tabla 57: Características del tipo de redes implementadas en el PP, en el trabajo de [150]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[150]	Si	No	No	No	No	No	No	No	No	No	No	No	No

Tabla 58 Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [150]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[150]	No	No	No	No	No	No	No	No

Configuración de Nodos de Red de Comunicación

En Sistema Embebido especificado por RdP

En este trabajo [52] se propone un conjunto de ecuaciones para configurar los nodos de comunicación de los sistemas en red embebidos (NES), considerados como globalmente asíncronos y localmente sincrónicos (GALS) [164]. El desarrollo de este trabajo utiliza un enfoque basado en el modelo con RdP para especificar sistemas embebidos. Se hace uso de modelchecking [165], que es una herramienta para verificar propiedades, y las herramientas automáticas de generador de código para la implementación del sistema en el hardware y plataformas de software. Propone la caracterización de los nodos de comunicación a través de un conjunto de ecuaciones, es decir, sus registros y buffers acotados, teniendo en cuenta las redes punto a punto, estrella, de bus, y anillo. La configuración automática de los nodos de comunicaciones, para las topologías que se hace referencia y en base a las ecuaciones propuestas, será soportada por una herramienta en fase de desarrollo.

En este trabajo, se presenta un ejemplo de aplicación, donde se utilizó un subconjunto de las ecuaciones propuestas para configurar los nodos de comunicación de una red con topología de anillo.

Secciones de interés del trabajo
Aquí se consideran los sistemas que son integrados en red (NES), compuesta por sistemas embebidos y distribuidos, y los componentes están interconectados por una red, que puede ser un chip de red (NOC), un networkoff chip, o una mezcla de ambos. El énfasis de este trabajo está en la red fuera del chip.

Entre las topologías de red existentes, que pueden ser utilizados para interconectar los componentes, cuatro topologías fueron considerados en este trabajo: punto a punto, estrella, bus y anillo.

En este trabajo se consideró el desarrollo de los sistemas en red integrados que son globalmente asíncronos y localmente sincrónicos (GALS), ya que los componentes de sistemas integrados suelen ser síncronos. Sistemas GALS están compuestos por un conjunto de componentes, cada uno sincroniza con una señal de reloj específica, que es distinta a las señales de reloj de los otros componentes.

Se ha propuesto enfoques de desarrollo basados en modelos para desarrollar sistemas integrados [166] [45] [167, 168], que presentan varias ventajas, a saber, los sistemas son mejor documentados, con menos errores de desarrollo o desarrollados en menos tiempo. Diferentes formalismos de modelado se pueden utilizar en el desarrollo basado en modelos, tales como diagramas de estado, gráficos de estado [169] , y las RdP [80].

Este trabajo utiliza un enfoque de desarrollo basado en modelos para desarrollar sistemas integrados en red. El enfoque considerado utiliza RdP para especificar los sistemas, herramientas de red en sistemas embebidos, modelos de comprobación para probar propiedades del sistema, herramientas automáticas de generación de código para hardware, y el código del software para las plataformas de aplicación. Se utiliza la clase de RdP IOPT [46] ampliadas en [53] para los sistemas GALS .El marco del modelo de comprobación RdP IOPT [170] para comprobar propiedades de sistemas integrados en red. La PNML2VHDL [44] y IOPT2C [50] como herramientas para generar el VHDL y los códigos C de cada componente de la red.

El trabajo presentado contribuye al tratamiento de una nueva herramienta para integrarse en el marco de las herramientas utilizadas en el enfoque de desarrollo mencionado. La nueva herramienta se utilizará para generar automáticamente los nodos de comunicación de la red para diferentes topologías. En este trabajo se propone un conjunto de ecuaciones para configurar un nodo de comunicación específico para redes punto a punto, estrella, bus o topologías de anillo. Todos los valores de las variables de las ecuaciones propuestas se pueden obtener en el modelo de RdP IOPT y en el espacio de estado asociado, que puede ser generada utilizando el marco de modelo de comprobación de [170] .

RdP IOPT extendido para sistemas GALS
Las RdP, de la clase, con entrada salida o "Place- Transición Petri net Input –Output" (IOPT - net) fueron propuestas en [46] para especificar los sistemas de automatización y sistemas embebidos . La clase RdP IOPT es una extensión de la clase Place- Transición Petri net [80] con entradas salidas, y una serie de características adicionales. Las entradas y salidas pueden ser

señales o eventos, lo que permite la especificación de la interacción entre el sistema (controlador) y el medio ambiente.

Dado que la clase RdP IOPT tiene un sincronismo y máximo-paso en la semántica de ejecución, por lo cual es conveniente especificar los sistemas globalmente sincrónicos. La clase RdP IOPT fue empleada en [53] para permitir la especificación de los sistemas distribuidos, que son sistemas GALS .

La clase RdP IOPT se amplió con los tiempos de dominios y canales asíncronos. Los dominios de tiempo se utilizan para asociar cada nodo de RdP con un componente diferente (cada componente es localmente síncrono). Los canales asíncronos se utilizan para especificar los componentes de la interacción. En el trabajo se presenta un modelo RdP IOPT, donde los dominios de tiempo se representan como anotaciones (td: 1, td: 2) y el canal asíncrono está representado por las nubes que conectan a las transiciones mediante flechas discontinuas.

Análisis de resultados
Con las ecuaciones propuestas es posible configurar los nodos de comunicación de los sistemas integrados en red con diferentes topologías y con modelos grandes y complejos.

De acuerdo con la topología de la red, cada componente del sistema integrado puede tener uno o más nodos de comunicación.

El tamaño del nodo de comunicación y el tiempo de la comunicación depende de: el número de componentes, la topología de la red, el número de diferentes eventos que se intercambian entre los componentes, y sobre todo del número de mensajes que pueden estar simultáneamente en la red. Un mensaje puede ir directamente desde el componente de origen al componente de destino (como en una red de punto a punto) o puede ir indirectamente (como en una red en anillo).

El autor propone y tienen en desarrollo una herramienta para configurar y generar los nodos de comunicación, para los sistemas integrados en red con punto a punto, estrella, bus, o anillo de forma automática. Esta herramienta se integrará en el marco de la cadena IOPT-herramienta (disponibles en [79]).

Es un proyecto que ha usado una RdP IOPT, para el diseño automático de un sistema de comunicaciones. Desde el punto de vista de las aplicaciones a sistemas complejos, el aporte es importante.

El controlador es construido por compilación de componentes que se interconectan. Las características que se implementan en [52] son mostradas en las Tabla 59 y Tabla 60.

Tabla 59: Características del tipo de redes implementadas en el PP, en el trabajo de [52]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[52]	Si	No	No	No	No	No	No	No	No	No	No	Si	No

Tabla 60: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [52]

Con respectos a las prioridades, eventos y programación

Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[52]	No	Si	No	Si	No	No	No	No

Circuitos Intra e Inter basados en Componentes de RdP

El presente trabajo [54] tiene como objetivo presentar un entorno de desarrollo que permite la generación automática de código susceptible para la interconexión de los componentes obtenidos como resultado de la partición de un modelo de RdP. Aborda ejecución distribuida utilizando controladores de red (incluyendo microcontroladores y dispositivos FPGAs , así como controladores específicos basado en PLC y de propósito general PCs) . La solución de interconexión propuesta se basa en una solución de red -on-Chip- que permite comunicaciones intra-e inter circuitos bajo el protocolo serial RS- 232 (aunque podría funcionar a velocidades de transmisión más altas). Siendo un protocolo bien aceptado en la industria, la solución propuesta integra módulos existentes que utilizan la interfaz RS232 sin tener que rediseñar todo el sistema de comunicación.

Las plataformas de aplicación utilizadas para las prueba de solución incluyen plataformas reconfigurables Xilinx , concretamente FPGAs Spartan3 y Virtex , así como microcontroladores de bajo costo , a saber Microchip PIC18F4620 y ordenadores de uso general ; PCs industriales , PLCs u otras plataformas con un puerto serie RS - 232 que pueden integrarse fácilmente. Una topología de anillo fue seleccionado para la comunicación. Se presenta un ejemplo en donde, a partir de la RdP modelada que representan el flujo de desarrollo se realiza el diseño e implementación.

Secciones de interés del trabajo
La evolución de los dispositivos de hardware y los recursos disponibles ha hecho posible la integración de varios componentes en un solo chip, utilizando circuitos dedicados o dispositivos reconfigurables. Estos sistemas se llaman “System on Chip” (SoC) o “Programmable-System-on-Chip” (SoPCs) cuando los dispositivos reconfigurables están involucrados. Es común referirse a estas soluciones de SoPCs como SoC.

La creciente complejidad de los sistemas hace que los requisitos de modelado sean cada vez más exigentes y se requiera de la descomposición del sistema en varios subsistemas entre los que interactúan entre sí, permitiendo la reutilización de componentes ya disponibles.

Desde el punto de vista del soporte a la comunicación entre estos subsistemas el uso de conexiones dedicadas no suele ser factible porque, a pesar de ofrecer un mejor rendimiento, ocupa demasiada área o cableado específico que se requeriría cuando los subsistemas considerados son implementados en plataformas heterogéneas. Así se justifica el uso de una red dedicada como una solución para los subsistemas de interconexión, identificada como Network-on-Chip (NoC).

Como se menciona en [171], la idea fundamental de una NoC es aplicar el enfoque por capas, común en las redes de telecomunicaciones e informáticas. Normalmente el modelo de referencia OSI no se sigue exactamente como en la norma, pero es adaptado.

La red puede ser orientada a la conexión (conmutación de circuitos) o sin conexión (conmutación de paquetes). Siempre toma una interfaz para formar el paquete y / o la conexión.

Las topologías propuestas disponibles varían mucho, incluyendo, topologías ad-hoc de interconexión, de malla, malla toroidal , anillo bidireccional , octogonal o árbol [171] . Según Nurmi [171] las más populares propuestas para NoCs existentes incluyen las siguientes: (1) xPipes [172] , (2) Proteo [173] , (3) la Nostrum [174] , (4) SoCbus [175], (5) el SPIN (Red Integrada programable Escalable) Árbol plano [176] , (6) la generalizada Extended Fat Tree (XGFT) [177], (7) en redes CDMA [178] y (8) Philips Æthereal [179].

Por otro lado, el desarrollo basado en modelos (MBD) se utiliza cada vez más por las metodologías de desarrollo actuales de soporte a la integración de los componentes de hardware y software.

Tener un modelo que describe adecuadamente el comportamiento del sistema es una gran ventaja que apoya su desarrollo, documentación y mantenimiento. Enfoques basados en modelos compatibles con la generación automática de código, ahorran tiempo y dinero, además de reducir al mínimo los errores de codificación. La adopción de enfoques basados en el modelo soporta claras mejoras en el flujo de desarrollo del sistema como se indica en [55] y [180].

El proyecto FORDESIGN [79] adhirió a estos objetivos mediante modelos de RdP para especificar el comportamiento del sistema. Una clase de RdP llamadas Input-Output Place-Transición (IOPT), que se describe en [46] , fue desarrollado permitiendo la integración de las señales y los eventos de entrada y salida en el modelo de RdP Place-Transición, que es bien conocido para el diseño de controladores.

Una de las herramientas desarrolladas permite la partición de un modelo IOPT en varias sub-modelos, siendo considerado más tarde como componentes para la ejecución distribuida del modelo (a partir del punto de vista de la implementación).

La adopción de las RdP para el modelado tiene la ventaja de una representación gráfica que funciona como un lenguaje de comunicación entre especialistas de diferentes campos. Permite la descripción de los aspectos estáticos y dinámicos del sistema que está siendo representado, y se beneficia de ser un formalismo matemático que permite el uso de herramientas para el análisis y la verificación.

Además, las RdP satisfacen las necesidades de los sistemas, tales como: la descomposición, la simplicidad, la sincronización y la concurrencia entre las tareas.

Los componentes susceptibles de ejecución distribuida del modelo se obtienen de acuerdo con las reglas de partición propuesto en [134] SPLIT. El uso de la herramienta y herramientas para la generación automática de código C y VHDL, se basan en el lenguaje PNML (RdP Markup Language) [96] para la representación de los modelos IOPT .

El propósito principal de este trabajo es presentar una solución para interconectar los componentes generados, implementados tanto en hardware (VHDL) o en el software (C), en un chip único (como una FPGA), o en un conjunto de dispositivos y sistemas heterogéneos.

La solución de la interconexión se codifica en VHDL en las plataformas de ejecución previstas para la validación de la solución (que incluye a las familias de FPGA Xilinx Spartan-3 y Virtex - II), así como en C para los sistemas externos. Con capacidad para soportar la solución de interconexión deseada (incluyendo ordenadores de propósito general y microcontroladores de bajo costo, por ejemplo el Microchip PIC).

Para poder garantizar la interconexión de las diferentes plataformas, según las necesidades de los componentes, se usa la división de la RdP IOPT.

Análisis de resultados
El objetivo de este trabajo ha sido obtener una solución y luego una herramienta para automatizar la generación de código. Incluso con algunas limitaciones de rendimiento tiene una alta flexibilidad basada en red de comunicación en serie que permite la generación del código de interconexión en VHDL y / o C, basado en los componentes asociados expresadas por modelos RdP IOPT y el soporte en plataformas con arquitecturas heterogénea embebidas de control.

Propone un nuevo chip de red que permita las comunicaciones entre circuitos intra-red basados en protocolo RS-232 serial (para lograr funcionar a velocidades mucho más altas cuando estén dentro del mismo dispositivo). Por lo tanto, las soluciones incluyen el hardware reconfigurable, los microcontroladores de bajo coste de uso común en las tareas de control, así como la interconexión con otros sistemas informáticos que tienen interfaces estándar RS232 o adaptadores compatibles.

Detalla la implementación de un sistema distribuido implementado a partir de un modelo de RdP IOPT, para sistemas embebidos. Desde el punto de vista de las aplicaciones a sistemas complejos el aporte es importante.

Las características que se obtiene en [54] son mostradas en las Tabla 61 y Tabla 62.

Tabla 61: Características del tipo de redes implementadas en el PP, en el trabajo de [54]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[54]	No	Si	Si	Si	No	No	No	No	No	No	No	No	Si

Tabla 62: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [44]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[54]	Si	Si	No	Si	No	No	No	No

Evaluación de Performance para Algoritmos de Ejecución de RdP

En este trabajo [155, 181] se presenta una evaluación del desempeño e interpretación de técnicas para la implementación centralizada de disparos de RdP interpretada y centralizada. Estas técnicas de implementación permiten la traducción de un modelo de sistema expresado por una RdP a un sistema real con el mismo comportamiento que el modelo. Se ha utilizado en diferentes campos de aplicación, tales como controladores lógicos programables.

Se han analizado cuatro técnicas: fuerza bruta, transiciones habilitadas, representación estática de lugares y representación dinámica de lugares. El análisis se ha llevado a cabo a través de una biblioteca de RdP compuesta por modelos conocidos que se pueden ampliar mediante un parámetro. El análisis de los resultados muestra que el rendimiento de los algoritmos depende del comportamiento RdP (concurrencia vs conflictos eficaces). Sin embargo, también se mostró que

la técnica de fuerza bruta puede ser desechada y que la correspondiente a lugares estáticos se comporta mejor que la versión dinámica cuando se realiza la selección de lugares, basada en la información del comportamiento.

Secciones de interés del trabajo
Las RdP son un formalismo que es muy adecuado para el modelado de sistemas de eventos discretos concurrentes. Se ha aplicado satisfactoriamente en campos tales como: las redes de comunicación, sistemas informáticos, sistemas de fabricación de piezas discretas, etc. Modelos con RdP, son a menudo considerados como especificaciones auto documentadas, debido a que su naturaleza gráfica facilita la comunicación entre los diseñadores y los usuarios. Las RdP tienen una fuerte base matemática que permite la validación y verificación de un amplio conjunto de propiedades. Además, estos modelos son ejecutables y se pueden utilizar para animar y simular el comportamiento del sistema. También se pueden utilizar para los propósitos de monitoreo una vez que el sistema está funcionando.

El sistema final obtenido con técnicas de implementación de generación de código ha sido derivado de un modelo de RdP implementando en hardware y software. En otras palabras, se pueden utilizar a lo largo de todo el ciclo de vida de desarrollo de un sistema.

Las implementaciones con RdP han recibido considerable atención de los investigadores en los últimos veinticinco años. La implementación es la traducción de un modelo real a un sistema expresado por una RdP, con el mismo comportamiento que el original. Se ha estudiado en numerosos campos de aplicación, tales como controladores lógicos programables, dispositivos digitales, simulación, sistemas de robótica y otras aplicaciones de software concurrentes.

Sin embargo, este trabajo esta interesado en la aplicación de software. Una aplicación de software es un programa que dispara las transiciones, observando las reglas de evolución del marcado.

Una aplicación se compone de dos partes una de control y una operacional. La parte de control corresponde a la estructura, marcado y reglas de evolución de la RdP. Por otro lado, la parte operativa es el conjunto de acciones y/o códigos de la aplicación, asociados a los elementos de red.

De acuerdo con diferentes criterios, una implementación de RdP se puede clasificar principalmente como compilada o interpretada, como secuencial o concurrente y como centralizada o descentralizada.

Una aplicación es interpretada si la estructura de la red y las marcas se codifican como estructuras de datos. Estas estructuras de datos son utilizadas por una o más tareas llamadas intérpretes para hacer evolucionar la red. Los intérpretes no dependen de la red implementada.

Una aplicación compilada se basa en la generación de una o más tareas cuyo flujo de control corresponde a las evoluciones de la red.

Una aplicación secuencial se compone de una sola tarea, incluso en redes con concurrencia. Este tipo de aplicación es común en aplicaciones cuya parte operativa está compuesta por acciones impulsadas sin tiempo de ejecución significativo. Es el caso de los controladores lógicos programables, ver por ejemplo [109] o [182].

El esfuerzo de investigación en este campo produjo un resultado importante, el Grafcet [183], un lenguaje estandarizado para controladores lógicos programables.

Una aplicación concurrente está compuesta por un conjunto de tareas cuyo número es igual a o mayor que el de la concurrencia en la red real. Ejemplos de implementaciones concurrentes se pueden ver en [152] o en [109] .

En una implementación centralizada la parte de control total es ejecutado por una sola tarea, comúnmente llamado coordinador. La parte operativa de la aplicación puede ser distribuida en un conjunto de tareas para garantizar la concurrencia expresada por la red [152]. De este modo, el coordinador actúa como el núcleo del sistema operativo de una aplicación multitarea.

Para caracterizar una aplicación de sistema centralizado, es importante determinar el algoritmo que habilita las transiciones y realiza los disparos. Aparte de la prueba exhaustiva sencilla de todas las transiciones, hay varias soluciones para reducir los costos de la prueba de habilitación.

El objetivo principal de este trabajo ha sido evaluar las técnicas de implementación actuales de las RdP interpretadas y centralizadas, desde el punto de vista de la sobrecarga introducida por la parte de control. Por esta razón presenta diferentes algoritmos de ejecución RdP que se ejecutarán dentro de un tamaño escalable.

En este trabajo se pretende demostrar que el tiempo de cálculo de la parte de control depende en gran medida de la estructura de la RdP y del algoritmo elegido para ejecutarlo. Analiza el tiempo de respuesta de los algoritmos y propone técnicas que nos permitirán elegir el algoritmo más adecuado para ejecutar una RdP. El objetivo es reducir al mínimo el tiempo de cálculo de la parte de control y de esta manera minimizar el tiempo de reacción del controlador.

Aplicaciones Centralizadas: Estructuras y Algorítmos
Una implementación de una RdP tiene una fuerte dependencia de la interpretación del modelo de red, es decir, los elementos son acciones y el código está relacionado con los elementos de la red. Este trabajo considera las RdP binarias con una interpretación que las asocia con el código de la aplicación, predicados y las prioridades de las transiciones. El disparo es la ejecución de un modelo de transición implementado por una pieza de código.

Se puede considerar tres fases de disparo: desmarcar los lugares de entrada, la ejecución de código y marcado de los lugares de salida. La prioridad de una transición representa la prioridad del código asociado en el acceso a la CPU permitiendo la resolución eficaz de conflictos entre transiciones mutuamente excluyentes. A los efectos de este trabajo, se considera que los predicados más las entradas del sistema a evaluar son siempre verdaderos.

Las técnicas de implementación centralizadas [152] [157] [183] se basan en dos tipos de hilos o tareas.

Los primeros grupos de tipo de tareas implementan la parte operativa del sistema, es decir, el código de aplicación asociado a las transiciones. Una tarea CODE se genera para cada transición de la RdP e incluye el código asociado. Cada tarea CODE hereda la prioridad de la transición.

El segundo tipo corresponde a una única tarea llamada la “Coordinadora”. Es el que hace que la red evolucione con el tiempo. El coordinador establece cuando las transiciones están habilitadas y deben dispararse y ejecuta el código asociado. El coordinador se comunica con las tareas CODE para autorizar su ejecución y tiene la más alta prioridad dentro de la aplicación.

La arquitectura es similar a un sistema operativo en el que el coordinador es el núcleo y las tareas de código son los procesos que se ejecutan dentro. El acceso al procesador está gestionado por el

sistema en tiempo de ejecución en lenguaje de destino, por el núcleo del sistema operativo basado en las prioridades de las tareas de código y el coordinador. El coordinador es un intérprete que funciona sobre una estructura de datos que codifica la RdP [183].

Aparte de la prueba exhaustiva sencilla de todas las transiciones (método de fuerza bruta), existen varias soluciones para reducir los costos de la prueba que permitan reducir la sobrecarga introducida por el coordinador. Dependiendo de estas soluciones, las técnicas de implementación centralizadas se pueden clasificar en una de las siguientes clases [157]:

En el enfoque dirigido por las plazas sólo las transiciones de salida de algunos lugares marcados se ponen a prueba [152]. Cada transición está representada por uno de sus lugares de entrada, que es la representación de la plaza. El resto de los lugares de entrada se denominan lugares de sincronización. Sólo esas transiciones, cuyo lugar está marcado, se consideran como candidatas para disparar. La plaza en representación de una transición puede ser estática o se puede seleccionar de forma dinámica como en [184] (en este caso se denomina activación por lugar).

En la técnica denominada de lugares dinámicos, si un lugar está marcado pero la transición no está habilitada debido a una o varias de las plazas de sincronización que la deshabilitan, cualquiera de los lugares de sincronización no marcados serán elegidos como nuevo lugar de representación. De esta manera podemos reducir el número de lugres de representación.

El enfoque de transición impulsada se caracteriza por que se suministra habilitación de las transiciones, diferentes de la marca, solamente si las mismas están completamente habilitadas. En general, una representación explícita del marcado no es necesaria. Este tipo de técnica es estudiada en trabajos como [109] o [157].

En el presente trabajo se han puesto en marcha cuatro algoritmos en los que se desarrollan las diferentes técnicas de búsqueda de transición habilitadas, los que son:

- Fuerza Bruta (BF).
- Transiciones Habilitadas (ET).
- Representación Estáticas de Plazas (SRP).
- Representación Dinámica de Plazas (DRP).

En una implementación centralizada interpretada, se necesitan una estructura de datos estática que codifique la estructura de RdP y una estructura dinámica que represente el estado o el marcado. En todos los algoritmos probados en este trabajo se ha mantenido una representación explícita del marcado, con el fin de permitir su uso en aplicaciones de control.

Esto implica una diferencia con respecto a la implementación presentada en [109]. Todos los algoritmos comparten la misma estructura de datos básica, que codifican la RdP con diferentes posibilidades de acceso adaptadas a cada técnica.

En el algoritmo se pueden distinguir dos fases:

1. Permite el análisis y la determinación de las transiciones sensibilizadas;
2. La realización del disparo de la transición.

El sistema funciona en dos ciclos y hace uso de dos listas: treatment_list y list_in_formation. El treatment_list contiene las transiciones que son candidatas para disparar, que se analizarán en la primera fase del ciclo de tratamiento. List_in_formation se compone en todo el ciclo de

tratamiento con las transiciones que serán candidatas para el disparo en el siguiente ciclo de tratamiento.

Con el uso de estas dos listas la red se ejecuta por pasos, evitando la aparición de condición de carrera. La diferencia fundamental entre las técnicas de implementación analizadas está en cómo se forma la list_in_formation, es decir las transiciones que se consideran en cada ciclo de tratamiento.

En la técnica de la fuerza bruta, la lista de tratamiento se compone de todas las transiciones de la red.

En la técnica de transiciones habilitadas las siguientes estructuras de datos estarán disponibles:

- Lista de transiciones Habilitadas.
- Lista Tratamiento integrado por las transiciones con todos los lugares de entrada marcadas.
- Lista transiciones Casi Habilitadas. Lista que se completará con las transiciones de salida de los lugares marcados en el disparo de las transiciones.

En las técnicas de SRP y DRP, las siguientes estructuras de datos estarán disponibles:

- Marcado Representando por lista de Lugares y lista de plazas.
- Lista de tratamiento con los lugares marcados -Representación y Plazas de sincronización.
- Marcado Representando lista Lugares siguiente ciclo.
- Marcado de sincronización.
- Lista lugares siguiente ciclo.
- Lista de Formación con los lugares de representación y de sincronización.

Las lista señaladas con el disparo de las transiciones, tienen el fin de resolver los conflictos de despido entre las transiciones que se excluyen mutuamente.

La treatment_list siempre se ordena por prioridad de mayor a menor. De esta manera, la transición habilitada con la prioridad más alta siempre se disparará. El disparo de la transición implica la no-marca de los lugares de entrada (que impide el disparo de otras transiciones en conflicto) y el envío de una autorización para la tarea CODE correspondiente a ejecutar su código.

A continuación, el coordinador entra en la segunda fase de ejecución siendo ésta el fin del disparo de la transición (la tarea CODE completo su ejecución) marcando los lugares de salida.

En el caso de la técnica de fuerza bruta, todas las transiciones están contenidas en el treatment_list y por lo tanto la list_in_formation no es necesaria para esta técnica.

En el caso de las técnicas Coloque impulsada, esas transiciones para las que la representación de Plazas está marcado, se consideran en la fase de pruebas de habilitación.

En estas técnicas, treatment_list está compuesta por las plazas que representan marcas y todas las transiciones representadas son consideradas como candidatas para el disparo. Si un disparo de una transición representada es realizado, el resto de las transiciones representadas no se prueban porque el lugar que representa se convierte sin marcar.

Por último, en la técnica transiciones Habilitadas sólo aquellas transiciones que están completamente habilitadas están contenidas en el treatment_list.

Cada vez que se dispara una transición, las transiciones que descienden de los lugares de salida se ponen a prueba y se incluyen en el list_in_formation si están habilitadas.

Análisis de resultados
En este trabajo el autor ha presentado una evaluación del desempeño de técnicas de implementación interpretados y centralizados de las RdP. Estas técnicas de implementación permiten la traducción de un modelo de sistema expresado por una RdP a un sistema real, que presenta el mismo comportamiento que el modelo. Se han analizado cuatro técnicas: "Fuerza bruta", "transiciones Habilitadas", "Representación de Lugares Estáticos¨ y ¨Representación de Lugares Dinámicos".

Para el algoritmo de representación de lugares se ha desarrollado una biblioteca que tiene como objetivo cubrir una amplia gama de RdP.

De acuerdo con las pruebas realizadas, se ha determinado:

Con la aplicación realizada con el algoritmo de transiciones habilitadas estática y dinámica se obtiene una reducción drástica del tiempo de cálculo en comparación con el algoritmo de Fuerza Bruta.

Si el algoritmo de Representación de Lugares Estáticos elige los lugares adecuados, el rendimiento es similar o mejor que de Representación de Lugares Dinámicos.

La elección del tipo de algoritmo más adecuado depende del comportamiento de la RdP, esto es concurrencia versus conflictos eficaces.

La ejecución de una RdP sin un algoritmo adecuado puede suponer grandes sobrecargas y una respuesta peor, y más lenta, en aplicaciones de control.

Actualmente este grupo trabaja en una técnica para sistematizar el uso de la gráfica de accesibilidad que facilite la selección del algoritmo más adecuado para ejecutar una RdP.

En el proyecto [155, 181] investiga distintos algoritmos para el disparo de una RdP, interpretada y centralizada. Como resultados se obtienen los algoritmos más adecuados para ser ejecutados en distintas condiciones y tipos de redes.

Sin embargo el estudio no aborda la ejecución en una FPGA.

Justifica y fundamenta el desarrollo de sistemas haciendo una traducción directa del modelo del sistema realizado con una RdP a un software ejecutable.

Restringe la evolución del sistema a dos estados para el cálculo, realizando disparos múltiples en un solo ciclo.

Este desarrollo es importante para ser aplicado a la traducción directa a partir de un modelo.

Las características que se obtienen en [155, 181] son mostradas en las Tabla 63 y Tabla 64.

Tabla 63: Características del tipo de redes implementadas en el PP, en el trabajo de [155, 181]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[155, 181]	Si	Si	No	No	No	No	No	No	No	No	No	No	No

Tabla 64: Características de eventos y programación implementadas en el PP, en el trabajo de [155, 181]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[155, 181]	Si	Si	No	Si	No	No	No	No

Distintas Formas en que han sido Divididas las RdP

Según su Interpretación, Diseño y Generación de Espacio de Estados

Arquitectura de hardware de un controlador jerárquico de grandes RdP
En el presente trabajo se propone un controlador por RdP para multiprocesadores [185]. Este ya ha sido aplicado como controlador de sistemas secuenciales en [38], para la automatización de una fábrica.

Los sistemas para la automatización de fábricas se pueden describir usando las RdP. Pero es difícil de describir con un gráfico de RdP cuando el sistema es grande. Por lo tanto, se propone dividir una RdP grande en una principal y muchas sub redes más pequeñas, de manera similar a un programa principal y subrutinas en un lenguaje de programación ordinaria.

Propone operaciones para "llamada a una sub-net" y "retornar de la sub-red" a la arquitectura de un controlador de RdP. Los lugares "Call- lugar" se usan para llamar a una sub-red. Para ejecutar operaciones de alta velocidad usa "llamada de sub-red" y "retornar de la sub-net". Propone una unidad de hardware llamada Unidad de Historia. La implementación de la Unidad de Historia, que es realizada por una memoria accesible en dos dimensiones "modo filas" y "modo columnas", como una memoria asociativa. La unidad de historia puede realizar "operaciones de sub-red" más simples y mucho más rápidas.

Secciones de interés del trabajo

Generalidades del multiprocesador controlado por RdP
Cada lugar, en del grafo, tiene un programa (llamado lugar - programa) para su proceso, que se especifica en PNPL (Petri Net Parallel Programming Language), propuesto en el trabajo [59] .

El multiprocesador consta de un controlador, varios procesadores de trabajo, la memoria común para el control y los datos.

El controlador dispone de cuatro tablas, que mantienen toda la información para controlar el marcado de una RdP, de la siguiente manera:

- Tabla de condiciones de disparo (FCT), almacena la condición para disparar transiciones.
- Tabla de Transferencia Token (TTT), tiene los lugares de transiciones.
- Tabla de atributos de Plaza (PAT), contiene los atributos de cada plaza.

- Tabla de estado de Token (TST), contiene todas las marcas de la RdP, la red principal y las sub-redes.

Cada procesador de trabajo tiene copias de todos los procesos asignados a los lugares. Las operaciones del control son las siguientes:

1) El controlador busca una transición sensibilizada, comparando TST y FCT.
2) El controlador mueve los token a su lugar, usando TTT.
3) El controlador lee los atributos de los lugares de PAT, y si el atributo es normal, entonces escribe el número de lugar en la cola EXQ (cola de ejecución), en la memoria común controlador.
4) Un procesador de trabajo, toma un número de plaza de EXQ y ejecuta el proceso asignado a ese lugar.
5) Cuando un procesador termina el trabajo escribe en EDQ el número de la plaza (cola final).
6) El controlador lee uno de los lugares, el número en EDQ y almacena el token en TST.

Descripción de la RdP jerárquica

Llamada Sub-net

Es dificultoso de manejar una RdP grande, por lo que se la divide jerárquicamente en redes más pequeñas. En este trabajo, se introduce un lugar de llamada "Callingplace" a un sub - red. Las llamadas a una sub-red son similares a las llamadas a subrutinas en lenguaje de programación convencional.

Las plazas “Callingplace” son utilizan para dividir una RdP jerárquicamente en sub-redes. La RdP, se divide en una red principal y una sub-red. Las sub-redes se obtienen cortando la red principal en los puntos de transición.

Se agrega en las sub-redes un lugar “G”, un lugar final (end) “E” y uno inicial (start) “F”. Si la subred tiene más de un arco de entrada o más de un arco de salida, se utiliza lugares globales para llevar un símbolo entre redes.

Cada plaza, excepto las de start y end, tienen un proceso asignado. En el momento en que una plaza adquiere una marca el controlador le indica a los procesadores el proceso asignado según el lugar que esté listo para ser ejecutado por el procesador.

Marca para llamadas a subred

Una llamada a las sub-red se ejecuta de la siguiente manera:

Una llamada a un lugar envía una señal, a una subred, con un nuevo token. Una etiqueta, que es un número, se asigna a la sub-red para identificarla, ya que la misma subred podría ser llamada de otras redes. La asignación del número de etiqueta significa que se utiliza la copia de la sub-red.

La asignación de un token en el lugar de start en la sub-red la inicia.

El token evoluciona con la subred.

Hasta que el token llega a la plaza end.

El token es devuelto a la plaza de llamada, lo que significa que la llamada-lugar ha terminado. Una muestra de la llamada-lugar se escribe en EDQ.

Todas los token que quedan en las plazas de la sub-red deben ser eliminadas. Si la subred había llamado a una sub-red en un nivel inferior, el token de las subredes más interiores debe eliminarse.

Unidad de hardware para llamadas a subred

Unidad de Historia

Se usan las etiquetas para identificar subredes.

La Unidad de Historia tiene las relaciones establecidas con las etiquetas entre sub-redes padre-hijo e hijo-padre.

Las plazas son identificadas por un par tag-número y el número del lugar. La Unidad de Historia comprobará si un lugar es asesinado por el par (tag -number, placenumber) o no. Los lugares son comprobados cuando se extraen de EDQ a la unidad de disparo (Fire Unidad), para evitar firec-hecking y también de EXQ para el empleo de los procesadores, lo que elimina procesos innecesarios.

La Unidad de Historia consiste en Killed Memory Place, CAM (Child padr Memory), Tag Killed Registro y un registro temporal.

La Killed-Memory es una matriz bidimensional (tag, lugar). El bit de (tag, encajes) es 1 si el lugar de la sub-red (etiqueta) es asesinado, y 0 si lo mató.

Para CAT, la memoria es una matriz de dos dimensiones (Child -tag, padre – tag) de bits. El bit de (Child -tag, padre –tag) es 1 si dos hijos -tag sub -net y el padre -tag tiene la relación de una subred llamada (chi1d) y una sub-red del hijo que llama (padre). Es 0 si no tiene ninguna relación. Una palabra que se lee de modo de fila tiene todas las subredes de los antepasados (nombrados como etiquetas de los antepasados).

Killed Tag Register, posee marcas devueltas directamente de las sub-redes.

Análisis de resultados

En este trabajo se propone dividir la red en subredes y dar semántica a las plazas y transiciones. Vemos que se requiere de tipificar las plazas, en algunos casos duplicarlas, muchas operaciones de decisión, tablas de direccionamiento y tablas para mantener los estados, esto último introduce más ciclos para una ejecución.

Al igual que en los trabajos anteriores, dos ciclos de memoria son suficientes para comprobar si la transición es sensibilizada o no. Es lo mismo que informar al proceso en dos ciclos.

Las características que se implementan en [185] son mostradas en las Tabla 65 y Tabla 66.

Tabla 65: Características del tipo de redes implementadas en el PP, en el trabajo de [185]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[185]	No	No	Si	No	Si	No	No	No	No	No	No	No	Si

Tabla 66: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [185]

Con respectos a las prioridades, eventos y programación

Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[185]	No	No	No	No	No	No	No	No

Partición de RdP utilizando las operaciones de división

El objetivo del trabajo [134] es contribuir para el uso de las RdP como el lenguaje de especificación de nivel de sistema en el marco de co-diseño hardware-software de los sistemas integrados, apoyando el modelo de partición del sistema en componentes.

El artículo presenta el conjunto de reglas de la operación de división de RdP sobre la base de la definición de un corte válido y capaz de dividir un modelo de RdP en varias sub-modelos que se comunican a través de canales síncronos. Los sub-modelos generados se asocian con los componentes que se ejecutarán al mismo tiempo y se asignan a los componentes de hardware o de software a nivel de aplicación.

La descomposición del modelo se consigue usando un conjunto de tres reglas. Luego se presenta un ejemplo de división del sistema de automatización, mostrando la aplicación de la división en un sistema.

Secciones de interés del trabajo
El aumento sostenido de los transistores por chip durante las últimas décadas no se ha equilibrado con un aumento idéntico en la productividad del diseñador, debido a la falta de métodos y herramientas adecuadas, dando lugar a lo que se conoce normalmente como la "brecha de productividad". La adopción de un desarrollo basado en modelos puede soportar adecuadamente las mejoras en esta área. Las RdP se encuentran en una posición muy buena para ayudar en este sentido. Lo que pueden aportar, es una sólida semántica y una fuerte base para la verificación de las propiedades del sistema, lo cual es un aspecto clave para aumentar la complejidad del sistema. Por otro lado, teniendo en cuenta el área de diseño de sistemas embebidos, es común la división para hacer frente a los grandes modelos de RdP. También, es común tener la necesidad de atenerse a las limitaciones específicas, es decir, para cumplir con las limitaciones de tiempo real y para adecuar el consumo de energía específico y/o requisitos de costos.

En estas situaciones, se requiere hacer frente a la magnitud del sistema, haciendo descomposición del modelo de sistema en un conjunto de sub-modelos concurrentes, que será ejecutado como componentes concurrentes. Esos componentes pueden potencialmente tener diferentes requisitos en términos de rendimiento y respuesta en tiempo real y ser desplegados en diferentes plataformas. El soporte, la división, tiene que concluir en soluciones más simples de obtener y equilibradas, y tener en cuenta las limitaciones específicas en cuanto a la relación costo/rendimiento.

Este es también el caso, cuando es necesaria la ejecución distribuida del modelo de RdP, lo que también está estrechamente relacionado con los sistemas integrados y el co-diseño hardware-software, donde se necesita asegurar la implementación de la partición del sistema en plataformas de modelo de componentes de software y/o hardware, o en arquitecturas multiprocesadores.

El objetivo es obtener el modelo de RdP como una serie de sub-modelos distribuidos y/o concurrentes. Esto se puede lograr a través de una operación específica, llamada división de la red, que es susceptible de permitir la descomposición de un modelo de RdP en varios sub-modelos que se ejecutarán al mismo tiempo.

El objetivo del presente trabajo es demostrar que el uso de RdP como lenguaje de especificación de nivel de sistema puede "fácilmente" mapear el sistema en varios componentes, si se posee el soporte de las herramientas adecuadas.

La operación a la red genera los sub-modelos que pueden ser considerados como componentes. Así, el objetivo de las tareas propuestas es el de obtener los componentes que pueden ser ejecutados y aplicados como unidades independientes (o componentes).

El documento se estructura de la siguiente manera: Primero se presentan algunos trabajos relacionados y ejemplos motivadores, seguido de una breve presentación de las principales características de la clase de RdP que serán considerados para modelar el sistema, llamado RdP Place-Transición Input–Output (RdP IOPT). A continuación, se centra en la operación de división de la RdP y las reglas asociadas. Finalmente, se presenta un ejemplo de aplicación. Se realizan breves consideraciones acerca de la generación de código y sus conclusiones.

Trabajo relacionado
En la literatura podemos encontrar muchas de obras que tratan de la descomposición de las RdP, la ejecución del modelo distribuido y análisis modular de las RdP. En estos trabajos, por lo general, podemos identificar fácilmente una preocupación común: es obligatorio preservar las propiedades del modelo antes y después de la descomposición de este. En [186] se presentan el modelado y los análisis de algoritmos distribuidos.

En [187] se presenta una propuesta de referencia en la modularidad de las RdP, donde la interacción entre los módulos se logra a través de lugares comunes o transiciones compartidas. Un conjunto de fusión se define para el modelado de datos compartidos, y el conjunto de la fusión de transición se utiliza para modelar acciones simultáneas entre los diferentes módulos.

Un ejemplo típico y muy simple, de modelo de RdP comúnmente utilizado en la literatura es el sistema “senderreceiver” como en [188] . En este caso, la aplicación de los componentes se soporta en el concepto de objetos distribuidos. Para la comunicación entre los componentes se utilizan dos tipos especiales de lugares: lugares de salida y lugares de entrada. Esto significa que tenemos que considerar una semántica especial para ejecutar el modelo y los métodos de verificación comunes no son directamente aplicables. Las reglas para la descomposición del modelo inicial son muy limitadas.

Por otra parte, la propuesta de los autores se refiere a la descomposición del modelo de sistema en un conjunto de componentes concurrentes, utilizando en lo posible los conceptos comunes en las teoría de RdP (con el fin de hacer uso de las herramientas comunes usadas en RdP), permitiendo la mayor flexibilidad al seleccionar el conjunto de corte (los nodos que definen cómo se encuentran sub-modelos).

El objetivo final es soportar la implementación de cada componente en una plataforma diferente, lo que permitiría la ejecución heterogénea y distribuida del modelo del sistema.

Esta solución para la ejecución distribuida tiene como objetivo asegurar que la localidad alrededor de los nodos del conjunto de corte se replique en cada componente con el fin de garantizar la evolución de los componentes de forma coherente con el modelo inicial. La comunicación entre los componentes se asegura a través de disparos sincrónicos de las transiciones (dentro de un conjunto de sincronía de transición).

Discusión sobre Conservación de las Propiedades, o Enfoque Pragmático para Descomponer una Red

La principal diferencia del enfoque propuesto, sobre todos los otros enfoques conocidos, es la actitud pragmática respecto a la conservación de las propiedades, cuando se considera el modelo inicial y el resultado de la descomposición en un conjunto de sub-modelos concurrentes (esto tendrá un fuerte impacto en la definición de como validar los cortes que se usarán). La actitud común dentro de la bibliografía de las RdP, es preservar las propiedades cuando se enfrentan a la descomposición de los modelos de red. Esta es la forma "robusta" para llevar a cabo la descomposición, ya que permite aplicar los resultados de análisis a las transformaciones del modelo. Sin embargo, para ser compatible con la preservación de la propiedad, existen limitaciones que se deben aplicar en la descomposición, lo que limita las posibilidades del corte y la restricción de uso de las clases de red seleccionadas. En términos generales, una de las principales ventajas de este enfoque se relaciona con la "corrección de los enfoques de la construcción". Por otro lado una desventaja importante son las "propiedades de conservación", puesto que no es posible aplicar la descomposición "arbitraria" en algunos modelos complejos que representan, a los llamados, problemas reales de la ingeniería global.

Se ha de señalar que esta descomposición arbitraria podría ser muy recomendable por las limitaciones específicas que viene a partir de las características específicas del sistema. En este sentido, la propuesta reduce el nivel de restricciones que permite la descomposición de los modelos a través de un conjunto arbitrario de corte de nodos (la satisfacción de un conjunto más débil y menos restrictivo de las normas), aunque eso podría tener como consecuencia que las propiedades no se conserven cuando la ejecución corresponda a un modelo distribuido.

Para explicar su propuesta, el autor tiene que identificar claramente dos pasos en la metodología.

La primera de ellas está relacionada con la división del modelo, utilizando la operación de división ; el comportamiento asociado con la ejecución de los sub-modelos resultante es similar al comportamiento observado para el modelo inicial, por lo que el conjunto de sub-modelos se ejecutan dentro de la misma plataforma y la semántica del canal de comunicación síncrona [100] puede ser satisfecha; en este sentido, las propiedades del modelo inicial son mantenidas por el conjunto de sub-modelos .

El segundo está relacionado con la ejecución de los sub-modelos dentro de un conjunto heterogéneo de plataformas (que podría producirse en la implementación del sistema); en esta situación , el paradigma síncrono asociado con las transiciones agrupadas dentro de un conjunto síncrono no se satisface, vamos a potencialmente observar una desviación del comportamiento del modelo inicial frente al comportamiento del conjunto de componentes paralelos; esto tiene como consecuencia que el conjunto de propiedades no se mantienen para el sistema final, las que pueden ser diferentes de las del sistema inicial .

Es necesario hacer hincapié en que el impacto en el comportamiento de esta desviación podría ser mínimo; por lo que es posible determinar anticipadamente (antes de implementar la aplicación), el nuevo conjunto de propiedades que presenta el modelo resultante.

Como cuestión de hecho, a partir de las tres reglas propuestas para producir la división del modelo, dos de ellas conservan propiedades antes y después de la partición, de acuerdo con las reglas de reducción de nodos que se presentan en [7]. Sólo la tercera regla, asociada con la partición a través de una transición de sincronización asegurando las evoluciones de dos o más

componentes, no mantendrá las propiedades. Hay que destacar que esto es muy conocido en la literatura [186]. Sin embargo, el orden relativo de los disparos de transición se mantiene. En resumen, la verificación de la propiedad se debe realizar antes y después de la separación y la implementación de los modelos.

Presentación de la clase de red IOPT
En esta sección se presenta brevemente la clase RdP IOPT, tal como se define en otro trabajo [46]. Esta clase se define como una extensión de la clase las RdP plaza transición (por ejemplo [80]) y su principal objetivo es soportar la construcción de modelos de controladores, lo que significa que los modelos pueden ser ejecutados por un controlador autónomo. A tal efecto, esta clase de redes se extiende de las RdP plaza transición con la especificación de las señales y los eventos de entrada y salida. Además, también permite la especificación de prioridades en las transiciones, y el uso de arcos de prueba. Ambas adiciones (prioridades y arcos de prueba) contribuyen a soportar la resolución de conflictos integrados en el modelo, ya sea directamente o modelado a través de la adición de sub-modelos específicos como árbitros para la solución autónoma de los conflictos.

Los acontecimientos y las señales permiten la especificación de las interacciones entre el modelo del controlador (la red) y el medio ambiente, como en redes interpretadas y sincronizadas [102, 104] , así como en otras extensiones no autónomas a las RdP , con especial énfasis a aplicaciones de automatización de fábrica, por ejemplo [132, 133].

Las señales de salida pueden estar asociadas con el lugar marcado, como en las máquinas de Moore. Para cada transición es posible asociar las prioridades, los dispositivos de seguridad que utilizan señales externas, así como los eventos de entrada y de salida, como en las máquinas Mealy Like. Las RdP IOPT tienen una semántica de paso máxima: cada vez que una transición está activada, y la condición externa asociada es verdadera (el evento de entrada y la guarda de la señal de entrada son verdaderas) la transición se disparó. El paradigma sincronizado también implica que la evolución sólo es posible en instantes específicos en el tiempo llamados tics. Eso es por las implementaciones en hardware, donde los tics podría estar asociados con un reloj global externo, mientras que para las implementaciones de software los tics podrían estar asociado con un ciclo de tratamiento.

Además, una transición habilitada y lista involucrada en un conflicto efectivo con otras transiciones habilitadas y listas solamente es disparada si tiene la prioridad más alta entre las transiciones en ese conflicto (el nivel de prioridad de cada transición dentro de un conjunto de conflictos es única). En la transición habilitada por encima se refiere a las limitaciones en cuanto a la parte autónoma del modelo, mientras que la transición lista también incluye las dependencias de la parte no autónoma del modelo (señales y eventos externos).

Descomposición de RdP mediante la operación división
La operación de división de red mencionada fue propuesto inicialmente en [56] y refinado en el presente documento. Fue implementado en una herramienta, llamada SPLIT, que interactúa con otras herramientas desarrolladas dentro del proyecto FORDESIGN [79, 189] . El objetivo principal del proyecto es contribuir a FORDESIGN para poner las RdP dentro de trabajos de co-diseño de sistemas embebidos. Tomando ventaja de la representación PNML (RdP Mark- up Lenguaje) para los modelos de RdP [128], la interacción entre las diferentes herramientas se realiza a través de archivos de lectura PNML generados. En este sentido, la herramienta que implementa la operación de división de la red se inicia leyendo el archivo PNML del modelo

inicial de RdP y como resultado crea varios archivos PNML. Cada uno asociado a cada sub-red generada (que se ve como un componente del punto de vista de la implementación). Además, la herramienta también crea un archivo en el que se incluyen todas las subredes asociadas a los componentes, en los que se incluye información sobre los conjuntos de sincronización utilizando canales síncronos. La herramienta de SPLIT y algunos ejemplos ilustrativos están disponibles a través del sitio web del proyecto FORDESIGN [79], en herramientas de etiqueta. La operación de división se basa en la definición de un conjunto de cortes válidos, lo que significa un conjunto de nodos a través del cual es posible dividir la red en varias subredes disjuntas.

Tres reglas diferentes se aplican a cada nodo que pertenece a un corte. Se definieron en función del tipo del nodo y sus arcos de entrada. La primera es para el caso cuando el corte del nodo es un lugar y las otras dos para el caso cuando el corte de nodo es una transición. Distinguimos entre las situaciones en las que la transición tiene arcos de entrada a partir de sólo uno o más de un componente.

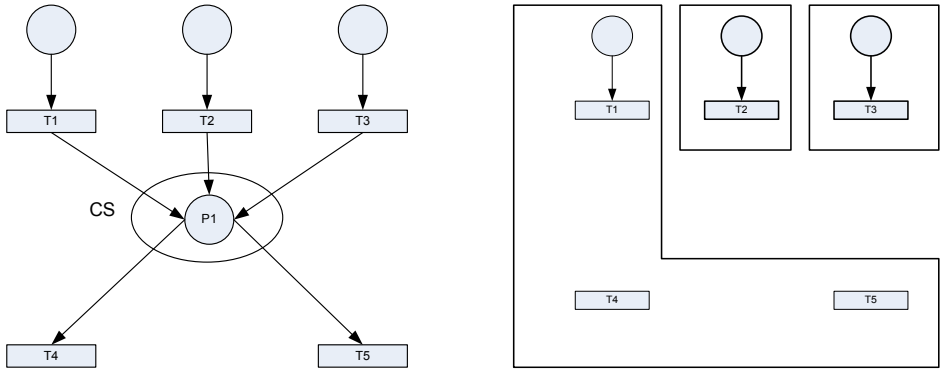
Antes de aplicar las normas para cada nodo, es necesario estar seguros que la eliminación de los nodos del conjunto de corte conduce a conjuntos disjuntos de subredes, y que todos los nodos del conjunto de corte, son elegibles para formar parte del conjunto de corte. Se identificaron como condición de admisibilidad para el corte, que es coherente con las reglas de división propuestas, imponer que los nodos del conjunto de corte y sus pre-sets no pueden involucrarse en cualquier situación de conflicto estructural. El flujo principal de la operación de división se presenta como el “Algoritmo 1”.

Las siguientes sub-secciones están dedicadas a una breve presentación de las tres reglas propuestas para dividir la red. Las normas propuestas se basan en la difusión y en la localidad relevante y asociada a cada nodo de corte, fijado en todos los componentes que están conectados con ese nodo del conjunto de corte específico. Esta estrategia asegurará que la evolución en los diferentes sub-modelos conectados a un nodo, de conjunto de corte específico, sea coherente en todos los sub-modelos (en lo que se refiere a la localidad asociado con el nodo del conjunto de corte).

Como una técnica de soporte para la operación de división, propone utilizar transiciones con un tipo específico de canales de comunicación síncronos conectados. Un grupo de transiciones que tiene unido el mismo canal de comunicación síncrona se dice que establece una sincronía. La semántica asociada con las transiciones en un conjunto sincronía específica es similar a la semántica de fusión de todas las transiciones que pertenecen y establecen la sincronía.

Las transiciones que participan en una sincronía establecidas tienen un atributo adicional, que podría tener uno de dos valores posibles: "maestro " o "esclavo". En una sincronía se establece un solo atributo por transición " maestro", mientras que las otras tienen el atributo "esclavo". Es importante señalar que, en la medida de como el paradigma síncrono es válido, ninguna atención especial se dedica al atributo " maestro / esclavo". Sin embargo, siempre que sea necesario para hacer frente a la ejecución distribuida utilizando plataformas heterogéneas, el atributo "maestro/esclavo " se utilizará para identificar el iniciador de los disparos dentro de las transiciones que pertenecen a un conjunto síncrono.

Regla # 1: un lugar como el corte de nodo
La regla # 1, define el procedimiento para los casos en que el nodo de corte (CS) es un lugar. La Figura 1 ilustra esta regla.



(a) Inicial **(b) Inicial CS**

Figura 26: Grafo inicial e Inicial CS de la regla #1

La Figura 26(a) representa un segmento de red, llamada inicial, donde P1 es el elemento del conjunto de corte CS. La Figura 26(b) representa los resultados de la operación de eliminación de nodo, donde se obtuvieron tres componentes diferentes (identificadas como tres regiones diferentes). No es obligatorio para obtener todos los componentes, aquí elegimos este para representar el caso general.

El requisito mínimo para esta regla es un segmento de red con una entrada y una transición de salida con respecto al lugar que pertenece el nodo de corte, por ejemplo T1, P1, T2. Como se representa en la Figura 26(b), T3, T2 y T5 pertenecen al componente 1, T1 al componente 2, y T4 al componente 3. La Figura 27 representa el resultado de la división de la operación, donde una copia de las transiciones que pertenece a la serie del lugar P1 y que no pertenecen al componente que sostiene la transición /s en el post-conjunto de lugar P1 se añaden dos transiciones en el caso de la Figura 27.

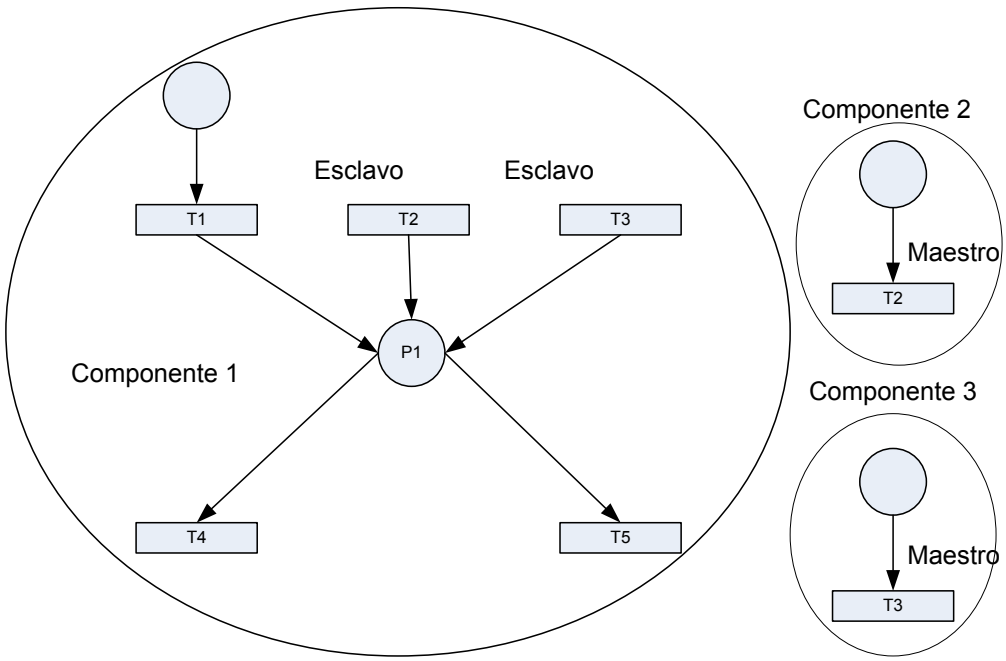


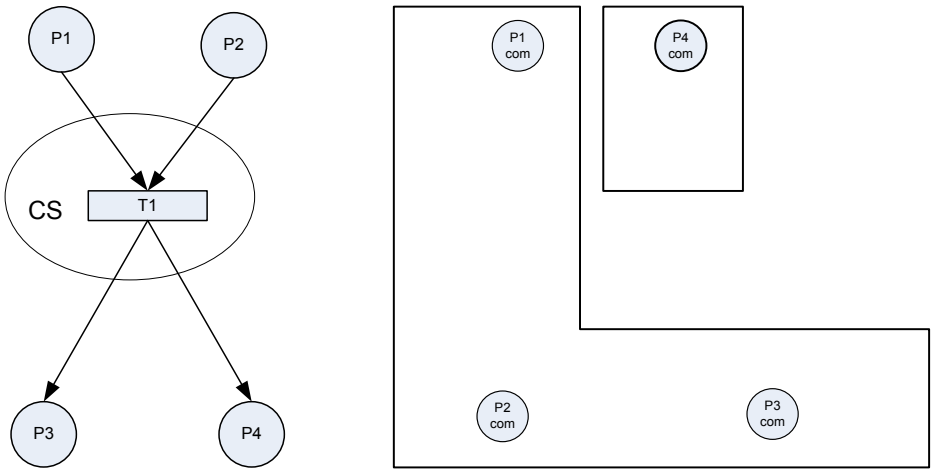
Figura 27: Nuevo gráfico resultado de aplicar la regla #1

Los componentes se comunican a través de los canales sincrónicos asociados con los conjuntos de sincronía. En la Figura 27, se utilizan dos conjuntos de sincronía. A modo de ejemplo, un conjunto de sincronía está compuesto por transiciones T1 (master) en el componente 2, y la copia T1 (esclavo) en el componente 1. La transición "esclavo " no tiene arcos de entrada; la única condición para el disparo se impone por el disparo de la transición "maestro". En este sentido,

podemos decir que el canal síncrono tiene dirección o dependencias, que son sólo en las transiciones "maestro".

Regla # 2: para una transición como cortar

La regla nodo # 2, es para el caso en que el nodo del conjunto de corte sea una transición con arcos de entrada procedente de un solo componente. La Figura 28 (a) representa un segmento de red inicial, donde T1 es el nodo de corte. La Figura 28 (b) representa los resultados de la operación de eliminación de nodo y la Figura 29 muestra el resultado de la división de la operación. El requisito mínimo para esta regla es tener un lugar de entrada y un lugar de salida con respecto de la transición de corte, por ejemplo P1, T1 y P4.



(a) Inicial **(b) Inicial CS**

Figura 28: Grafo inicial e inicial CS de la regla #2

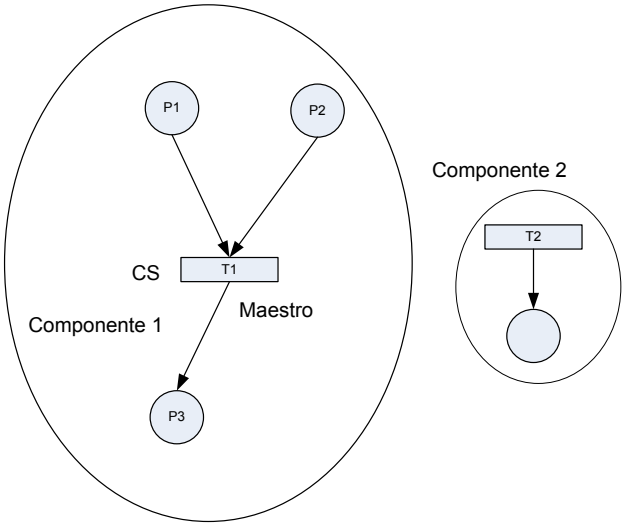


Figura 29: Nuevo gráfico resultado de aplicar la regla #2

Regla # 3: para una transición como nodo de corte

Esta regla es para el caso en que la transición de corte tiene arcos de entrada de los nodos que pertenecen a diferentes subredes después de cortar el nodo. En este caso hay que elegir qué subred recibe la transición con el atributo de maestro del canal de comunicación sincrónica, es el componente "maestro" que pertenece al conjunto de corte con respecto a esta transición.

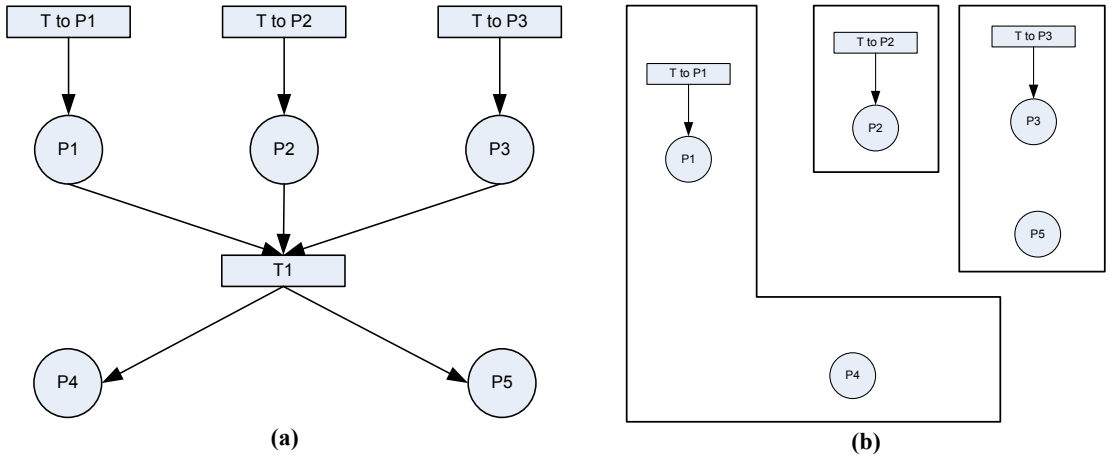


Figura 30: (a) RdP sin dividir, (b) operación de corte por la transición de la RdP

Las Figura 30 y Figura 31 ilustran este corte. La Figura 30 (a) presenta un segmento de red inicial, en el que el nodo de corte es T1. La Figura 30 (b) muestra el resultado de la operación de corte, la obtención de tres componentes. La Figura 31 ilustra el resultado de la operación de división. El componente "maestro " con respecto a T1 fue elegido por ser la sub-red a la que pertenece P1. Esta sub-net (asociado con el componente "maestro") tiene una copia de todos los lugares que pertenecen a la sub.red de T1 que están en una subred diferente.

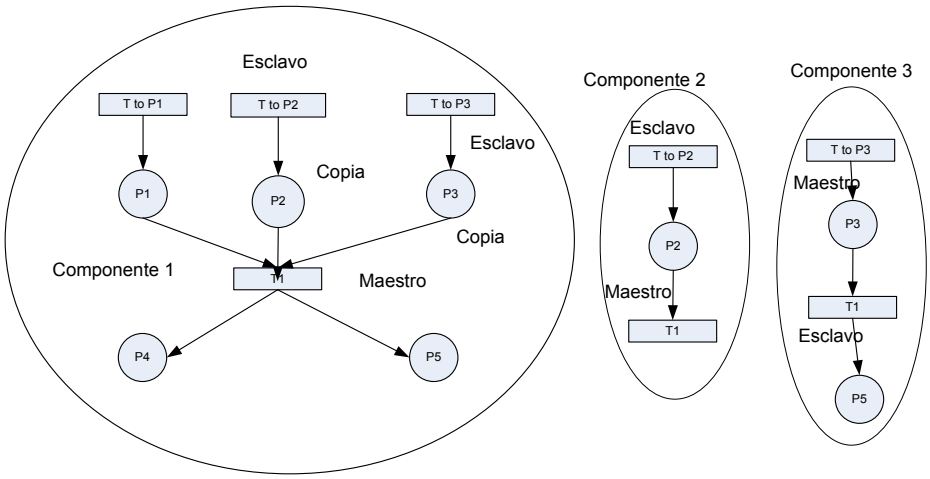


Figura 31: Resultado de la operación de división por la transición

Además, para cada uno de estos lugares es necesario incluir copias de las transiciones predefinidas (que generan tokens a estos lugares). Estas copias de transiciones están asociados con las transiciones iniciales a través de conjuntos de sincronía (son tres en el ejemplo de la Figura 31). El requisito mínimo de esta norma es la existencia de al menos dos lugares preestablecidos de la transición de corte, por ejemplo P1 y P2 de la Figura 31.

Análisis de resultados

Este trabajo realiza la operación de división de red enfatizando como la herramienta de dividir asociada obtiene la división de un modelo de RdP en un conjunto de sub-modelos concurrentes.

La comunicación entre los sub-modelos está asegurada a través de canales síncronos. La herramienta para dividir se integra en el marco del proyecto FORDESIGN [79] , como una herramienta clave para soportar una metodología para el diseño de sistemas embebidos utilizando técnicas de co-diseño de hardware-software y las RdP como el formalismo subyacente. La

herramienta está disponible públicamente a través del sitio web del proyecto denominado FORDESIGN [79]. Junto con el conjunto de otras herramientas desarrolladas que incluyen: un editor gráfico, los traductores a lenguajes específicos como VHDL y C, y un configurador para plataformas específicas de sistemas embebidos.

Para trabajos futuros tendrán en cuenta las implementaciones de las aplicaciones para controladores distribuidos. También, en el futuro dedicarán especial atención a la selección de soporte de comunicación específica que permita la sustitución de los canales síncronos de modelos adecuados (y módulos asociados), que se utiliza cada vez que se requiere de sistemas distribuidos.

Este trabajo [134] muestra las ventajas y como dividir RdP IOPT , haciendo uso de tres reglas. Es importante el análisis de los trabajos previos y la fundamentación.

Se hacen restricciones para la división y la tercera regla no mantiene la semántica. El enfoque se basa en realizar la división para implementar sistemas distribuidos modelados con RdP IOPT. La división requiere de canales síncrono y repetición de transiciones y/o plazas.

Esto último hace que la sobrecarga introducida por el software de ejecución de la red sea importante, puesto que la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones, el disparo y hacer la comunicación. El método no permite integrar distintos tipos de redes. Las características que se implementan son las mostradas en las Tabla 67, Tabla 68 y Tabla 69.

Tabla 67: Aportes de[134] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[134]	No	No	Si	Si	Si	No	No

Tabla 68: Aportes de [134] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[134]	No	No	Si	Si	No	No

Tabla 69: Aportes de [134] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[134]	No	Si	Si	Si	Si	No	Si	Si	No	Si

Extensión de la Operación de División para la Descomposición de RdP de Alto Nivel

En este trabajo [190], se presenta una extensión de una operación de división de RdP para la descomposición de redes de alto nivel, para soportar las implementaciones distribuidas de sistemas embebidos. La operación de división de la red, se realiza a partir del estudio de los

métodos que se resumen en [134], se propuso originalmente para RdP de bajo nivel. Las ventajas del método seleccionado son su mayor flexibilidad en la definición del conjunto de corte y la generación de sub-modelos en interacción con un soporte adecuado para poner en práctica sus canales de comunicación. Las RdP de alto nivel son extendidas para soportar nuevas operaciones y tareas. Finalmente, un ejemplo se presenta para ilustrar la aplicación de la operación de red a un modelo de RdP de alto nivel.

Secciones de interés del trabajo
Varios autores [166, 167, 191, 192] han propuesto enfoques de desarrollo basados en modelos para el desarrollo de sistemas embebidos. Formalismos de modelado como [45, 193]se han utilizado para varios tipos de RdP de alto nivel y RdP de bajo nivel, que son adecuadas para especificar los sistemas de control.

Las RdP de alto nivel son adecuadas no sólo para especificar "el control" de los sistemas, sino también para especificar sistemas con énfasis en "procesamiento de datos", con modelos más compactos. En este trabajo se explota el uso de las RdP de alto nivel dentro de sistemas embebidos y distribuidos.

Para obtener la especificación de un sistema embebido complejo, a través de un modelo de RdP, es conveniente hacer la composición de varios sub-modelos más pequeños con el fin de obtener el modelo global del sistema. La utilización de los mecanismos de estructuración para hacer submodelos de composición [194] son posibles, realizando un enfoque de desarrollo de abajo hacia arriba para luego volver a utilizar los submodelos previamente.

Para optimizar la aplicación (lo que implica mejorar el rendimiento, reducir el consumo de energía o la interferencia electromagnética) de un sistema embebido complejo puede ser necesario considerar una implementación distribuida, compuesta por un conjunto de componentes que interactúen.

Una implementación distribuida, de un sistema embebido, puede ser vista como un sistema embebido distribuido. Un sistema embebido distribuido puede ser visto como una red de sistemas integrados (por componentes) que realizan una o más tareas específicas. Algunos autores, por ejemplo en [134, 195], sostienen que en algunas situaciones es más fácil especificar el comportamiento de un sistema distribuido sin tener en cuenta la distribución final de los elementos del sistema, ya que el enfoque utilizado para desarrollar un sistema embebido o un sistema embebido distribuido puede ser similar.

Este enfoque de desarrollo considera la especificación de comportamiento del sistema global utilizando un modelo único (que se puede obtener basándose en los mecanismos de estructuración), y luego hacer la descomposición del modelo en un conjunto de interacción de sub-modelos, con el fin de soportar una implementación optimizada.

En algunos casos, no es posible saber en las etapas iniciales del proyecto como el sistema debe ser descompuesto, para obtener una implementación optimizada. Sólo después de la puesta en práctica y la prueba, teniendo en cuenta las diferentes particiones, se puede obtener una aproximación de la distribución óptima.

La especificación de un sistema embebido distribuido incluye la especificación de cada componente y la especificación de la interacción entre los componentes. El número de componentes y funcionalidades del sistema se distribuyen entre los componentes, esto puede

depender de varias razones, tales como propósitos de distribución o de optimización geográfica. Los componentes de un sistema distribuido pueden ser tratados geográficamente, en una plataforma de aplicación única o en un "System-on-Chip" (SoC).

Una aplicación distribuida puede reducir el consumo eléctrico y la interferencia electromagnética (si se reducen las frecuencias de operación de algunas partes del sistema) y los costos (si los componentes anteriormente desarrollados son reutilizados).

En este trabajo se extiende un método de descomposición estructural, que se propone en [134] para las RdP de bajo nivel, para soportar la descomposición de las RdP de alto nivel.

Se presenta un resumen de RdP de alto nivel, basado en la norma ISO/IEC15909. Luego se revisa varios trabajos que proponen o que utilizan métodos de descomposición de las RdP, y se presentan las ventajas del método seleccionado.

Posteriormente se presenta el método seleccionado, que es un método de descomposición de RdP de bajo nivel, que soporta la implementación de sistemas embebidos. También detalla las similitudes y diferencias de la aplicación de este método a bajo nivel con las RdP de alto nivel. También, la identificación de las tareas adicionales que se deben realizar con el método extendido para dividir las RdP de alto nivel.

Se propone una extensión para las RdP de alto nivel, para soportar la aplicación del método extendido. Por último, se presenta un ejemplo de aplicación y las conclusiones.

RdP de alto nivel

Hay muchas clases de RdP de alto nivel, tales como Predicado Transición, RdP Coloreadas, RdP estocástico de Alto Nivel, RdP Jerárquica temporizadas, etc. En este trabajo se estudia la aplicación de una operación de división de las redes [134] de alto nivel.

Los documentos internacionales de estándares ISO/IEC 15909-1 e ISO/IEC 15909-2 [196] presentan la definición de las RdP de alto nivel. La norma define las RdP de alto nivel como una extensión de las RdP bajo nivel, introduce anotaciones a Lugares, transiciones, Arcos, RdP y Páginas. Cada lugar tiene un tipo de dato y un conjunto múltiple de token de ese tipo de dato. Las transiciones pueden tener condiciones asociadas (que limitan su disparo). A los arcos se asocian etiquetas que pueden ser variables, constantes o funciones, que determinan que token serán removidos (destruidos) en los lugares de entrada, y cuales se insertan (creados) en los lugares de salida.

En la bibliografía [196] se presentan las estructura básica del paquete de alto nivel que extiende el núcleo del modelo PNML, utilizando un diagrama de clases UML.

Métodos de descomposición de RdP

La descomposición de las RdP ha sido abordada en numerosas obras en las últimas décadas, y se centran en diferentes clases y objetivos. La descomposición puede ser vertical (jerárquica) u horizontales (redes de todas formas separadas en el mismo nivel de abstracción). Su uso puede tener uno o más de los siguientes objetivos:

- mejorar la lectura de los modelos (como sustitución de un modelo grande para un conjunto de sub-modelos más pequeños);
- producir los modelos de reutilización según modelo de comportamiento, a través de la descomposición y métodos de composición;

- simplificar el análisis y la verificación de modelos complejos;
- implementar o producir implementaciones optimizadas.

El autor en las siguientes subsecciones aborda objetivos relacionados con la descomposición RdP, que soportan el análisis e implementación del comportamiento. Una subsección de esta discusión se refiere al método seleccionado e identifica sus ventajas.

Como soporte del análisis de comportamiento

Métodos de descomposición se utilizan sobre todo para soportar el análisis de sistemas, simplificando, entre otras cuestiones, la generación y el análisis de los gráficos de espacio de estado y el cálculo de invariantes de la RdP. Esta sección se refiere a un conjunto de trabajos con énfasis en el análisis del comportamiento de las RdP, utilizando varias técnicas de descomposición.

Se proponen métodos de descomposición jerárquica en varios documentos (por ejemplo, [197, 198]) para mejorar la legibilidad de los modelos y para simplificar el análisis de los sistemas complejos. La descomposición jerárquica proporciona diferentes niveles de abstracción, donde los niveles más altos son una abstracción de los niveles inferiores. Métodos de refinamiento se pueden utilizar para producir el modelo final de RdP.

RdP Estocástico (SPN), a menudo son usadas para el análisis de rendimiento.Se basan en una serie de enfoques de descomposiciones (por ejemplo [199-201]), para reducir la complejidad del análisis.

En [200] se presenta un enfoque de descomposición, para soportar el análisis de grandes RdP estocásticas y la reducción del tamaño de los gráficos de espacio de estado generados. Este enfoque es compatible con la descomposición de un conjunto de tipos de estructuras cercanas e independientes.

Una técnica de descomposición para dividir los modelos de RdP se presenta en [202], lo que permite el análisis de cada sub-modelo resultante por separado, y evita la generación de todo el espacio de estado (que es una tarea que consume tiempo y recursos). La descomposición se puede realizar por división de: (1) transiciones, la creación de conjuntos de lugares (S-componentes); o (2) los lugares, la creación de conjuntos de transiciones (T-componentes). En (1) algunas transiciones son compartidas entre los componentes, mientras que en (2) algunos lugares se comparten entre los componentes. Para crear un sistema compuesto por S-componentes, las transiciones se fusionan, y se obtiene un sistema compuesto por T-componentes, donde los lugares se fusionan.

En [203] se propone un método de descomposición para las RdP con tiempo. El método se utiliza para simplificar el análisis de los sistemas complejos (especificado por las RdP con grandes espacios de estados). Se define la unidad básica de concurrencia (BUC: Basic unit concurrency) y se propone un algoritmo para descomponer el modelo original en un conjunto de certificados de usuario final, que son más fáciles de analizar.

En [204] las RdP también se descomponen en subredes más pequeñas, para evitar la construcción de todo el espacio de estado. Estas subredes tienen que mantener las propiedades específicas del modelo inicial con el fin de soportar la verificación de las propiedades, utilizando una técnica de verificación modular. Otros trabajos también presentan las técnicas de verificación modulares, como en [187], donde los módulos se comunican a través de los nodos compartidos (lugares o

transiciones). La forma de obtenerlos es: (1) revisar las propiedades de comportamiento de la red global, con base en los espacios de estados de los módulos individuales; y (2) la construcción de los invariantes de la red global basada en los invariantes de los módulos individuales.

En [205] se proponen dos algoritmos para descomponer las RdP en redes funcionales. Las redes funcionales resultantes, que conservan las propiedades de la red inicial, tienen lugares de entrada y salida como interfaz. Para obtener el modelo inicial los lugares de interfaz deben fusionarse.

En otro trabajo [206], se utilizó una técnica relacionada con el cálculo de los invariantes de las RdP.

En [207] se presentan dos métodos de descomposición para las redes de elección asimétricas, con fines de diseño y verificación.

Estos métodos preservan las propiedades deseadas (incluyendo vivacidad y marcas limitadas), y verifican las propiedades en las redes resultantes. La descomposición se realiza a través de la división de lugares o de Transiciones.

Un método de descomposición fue utilizado en [208]. Estos autores tienen varios trabajos publicados con métodos de descomposición para RdP que permiten la simplificación del análisis de las RdP (RdP ampliadas con un tipo de lugar que se llama el lugar de restricción de estados). En este trabajo, un método de descomposición se utilizó para los vehículos guiados automáticamente (AVG) en la optimización de enrutamiento. Vehículos con movimiento concurrentes, con restricciones para evitar la colisión, se modelan mediante RdP. El tiempo mínimo de transporte está dado por la secuencia óptima de disparo de las transiciones. Para simplificar el problema de encontrar buenas rutas, los modelos de RdP se descomponen en una serie de sub-modelos. La descomposición se hizo duplicando lugares que están destinados a permanecer en diferentes sub-modelos.

Se presentan dos métodos de descomposición, que son propuestos en [209] para simplificar el análisis de las RdP con estructuras complejas. Las propiedades de la red original pueden ser analizadas de acuerdo con las propiedades del modelo resultante. Los dos métodos están basados en los índices de lugares o transiciones, que se utilizan para analizar la sincronización entre los sistemas o la compartición de recursos entre los sistemas. En uno de los métodos, se consigue un conjunto de subredes a través de la replicación de las transiciones, en el otro método a través de la replicación de los lugares. Para obtener el modelo inicial es suficiente fusionar los nodos replicados.

En [144] se propone un método de descomposición para los modelos de RdP, para ser usados en el diseño de controladores descentralizados complejos para sistemas de eventos discretos. Las partes débilmente conectadas del modelo de RdP se identifican. De la eliminación de estas piezas se debe obtener un conjunto de sub-modelos desconectados. Las partes débilmente conectados se replican, y se introducen en los sub-modelos desconectados, lo que resulta en un conjunto de subredes, que a continuación, están interconectados usando dos transiciones entre cada par de lugares replicados. Este método de descomposición asegura el mantenimiento de propiedades de red limitadas, la reversibilidad y la vivacidad (con una condición adicional) y la simplificación del análisis de los controladores.

En [210] las RdP se descomponen en redes abiertas. Redes abiertas son las RdP con una interfaz compuesta por los lugares de entrada y lugares de salida. Es posible obtener el modelo inicial,

componiendo las redes abiertas, a través de la fusión (interfaz) de los lugares compartidos. Lugares compartidos pueden ser vistos como canales de comunicación entre los componentes. Cada lugar de salida de uno de los componentes, es un lugar de entrada de otro componente y cada lugar de entrada de uno de los componentes es un lugar de salida de otro componente.

Esta es una limitación, ya que cada lugar no puede estar en más de dos componentes, siendo una restricción de los lugares en las particiones. Este método de descomposición se puede aplicar, por ejemplo, para simplificar el análisis de comportamiento, o para obtener servicios web a partir de modelos de procesos de negocio.

Como soporte de la ejecución

Varios trabajos se pueden encontrar en la literatura que propone métodos de descomposición como soporte para:

- Síntesis de circuitos asíncronos;
- Reconfiguración dinámica de hardware;
- Implementación de sistemas distribuidos.

En [211, 212] , se hace la descomposición gráfica de la señal de transición, se trata de una clase de RdP no autónoma. No autónoma en el sentido de tener entradas y salidas que limitan la evolución del modelo. El método de descomposición se hace a través de operaciones admisibles, tales como contracciones de transiciones, o eliminando nodos (lugares o transiciones). La descomposición conduce a un sistema modular ampliado, lo que simplifica la generación de espacio de estado y la síntesis de circuitos asíncronos.

En [213] se propuso una herramienta (DES) para descomponer los gráficos de la señal de transición, y en [214] se proponen cuatro heurísticas para encontrar una mejor partición.

En [215] los procesos de hardware representados por modelos de RdP CDFG, se dividen en una serie de sub-modelos, que luego son utilizados por un mecanismo de hardware virtual para hacer reconfiguraciones en tiempo de ejecución sobre la FPGAs. Este enfoque reduce el área de utilización, porque no todas las partes del proceso se aplican al mismo tiempo.

El desarrollo de aplicaciones concurrentes, que utilizan un lenguaje de alto nivel de RdP a objetos, es considerado en [188]. Para soportar la ejecución distribuida, los modelos son divididos en un conjunto de sub-modelos, en sustitución de los lugares de comunicación por salidas especiales y lugares de entrada, llamados puertos de entrada y salida. Cada token insertado en los puertos de salida se entrega inmediatamente al puerto de entrada del receptor. Este trabajo propone la descomposición de las RdP de alto nivel, para obtener un conjunto de componentes de interacción, que soporte la aplicación.

En [216], un clase de RdP de alto nivel (redes Predicado/Transición extendidas) es utiliza para desarrollar sistemas embebidos de tiempo real y de ejecución paralela. Para soportar implementaciones distribuidas de sistemas complejos se propone un método de partición. La partición utiliza un conjunto de corte de nodos y los duplicados de las partes resultantes. Un método de pre-partición apunta a producir implementaciones de sistemas reduciendo la sobrecarga de comunicación.

Una técnica de sincronización fue propuesto en [217], para transformar una especificación síncrona “sincrónico local” (GALS) en una especificación de sincronismo global, soportando la

implementación como un sistema GALS. Esta operación de partición asegura la preservación de la semántica y conservación del bloqueo.

La interacción entre los componentes se realiza a través de los puertos (circuitos que proporcionan una interfaz entre los elementos síncronos y asíncronos), que puede sintetizarse usando RdP, basada en herramientas de síntesis.

En [134] se propuso un método de descomposición para soportar la implementación distribuida de sistemas embebidos. En los sistemas integrados especificados por RdP IOPT [46], los modelos pueden descomponerse en una serie de componentes que interactuan como sub-modelos, que trata de traducir de forma automática a los códigos de software e implementación de hardware.

La interacción entre los sub-modelos se especifica mediante canales síncronos dirigidos, asegurando que todas las propiedades del sistema se mantengan después de la descomposición. Este método de descomposición se presenta en [134].

Análisis de resultados
La investigación presentada mostró la falta de métodos de descomposición de las RdP de alto nivel, para soportar las implementaciones distribuidas de sistemas embebidos (con énfasis en el "procesamiento de datos"). Las implementaciones distribuidas están compuestas por un conjunto de componentes que interactúan. En este trabajo no se especifica un método de descomposición adecuado para crear conjuntos como sub-modelos, tampoco detalla explícitamente cómo interactúan.

Aunque algunos de los métodos de descomposición propuestos, para reducir la complejidad del análisis de los sistemas, puede ser utilizado para soportar la implementación de sistemas. Sin embargo no proporcionan la información necesaria para implementar la interfaz de comunicación entre los componentes resultantes del sistema.

El método propuesto en [134] permite la descomposición en un conjunto de componentes que interactúan. Se especifica la interacción utilizando canales síncronos dirigidos. La comunicación en los canales síncronos dirigidos no es simétrica, se tiene una transición principal y una o más transiciones esclavas, haciendo a este tipo de canales de comunicación susceptibles de soportar la implementación distribuida del modelo, en oposición a los canales síncronos como propone en [100] y [91] para RdP coloreadas, las que son simétricas por su naturaleza.

Este método [134] tiene mayor flexibilidad y pocas restricciones en la definición del conjunto de corte, que puede estar compuesta por los lugares y transiciones. Como estaba dirigida a las RdP de bajo nivel, el presente trabajo tiene como objetivo concluir acerca de su aplicabilidad e idoneidad (identificando las extensiones requeridas), para soportar la descomposición RdP de alto nivel que se usan en implementaciones distribuidas.

La operación de división de RdP de alto nivel
Teniendo en cuenta que las RdP de alto nivel son una extensión de las RdP de bajo nivel, como se muestra en [196]. La operación de división de red presentada en la sección anterior se puede aplicar a las RdP de alto nivel después de la integración de algunas extensiones específicas.

Tal como se presenta en el apartado anterior, cuando se aplica la operación de división de las RdP de bajo nivel, los arcos se replican con el mismo peso, los lugares se repiten con la misma marca

(número de token) y los conjuntos de sincronía se introducen entre las transiciones maestras y transiciones esclavas (replicadas).

Para aplicar esta operación de separación a las RdP de alto nivel, sus objetos (lugares, transiciones y arcos) también se debe replicar, y los conjuntos de sincronización también deben ser insertados. Pero debido a que los objetos de RdP de alto nivel tienen notaciones distintas, la operación de división que ha de ampliarse para replicar estas notaciones y los conjuntos de sincronía debe ser diferente.

En las RdP de alto nivel cuando se replican lugares, sus tipos y estructuras de datos (marcados) deben replicarse; y cuando se replican arcos, sus anotaciones deben replicarse.

Sin embargo, las propuestas para la sincronía de las RdP de bajo nivel pueden aplicarse para las RdP de alto nivel, ya que ofrece las transiciones esclava del evento generado por la transición principal; mientras que el conjunto propuesto para la sincronía de RdP de alto nivel deben llevar mensajes en lugar de eventos.

Cada mensaje tiene que contener la información requerida por las notaciones de arco que se conectan a las transiciones esclavas, asegurando el flujo de atributos de la entrada a los arcos de salida.

Para ambas clases de RdP de bajo nivel y de alto nivel, la operación de división no replicará las condiciones asociadas a las transiciones, y las características no autónomas asociadas con objetos de RdP. Características no autónomas son las entradas y salidas, que especifican la interacción entre la RdP y el medio ambiente. Las entradas limitan el disparo de las transiciones y se reservan, solamente por asociación, con la transición master. Las salidas tienen un efecto sobre el medio ambiente.

La siguiente sección propone la definición de conjuntos de sincronía de las RdP de alto nivel. Es compatible con los documentos estándares internacionales presentada en [196].

Conjuntos de sincronización para la ampliación RdP de alto nivel
En [218] se define al conjunto de la sincronización de las RdP de bajo nivel. Para aplicar este método de descomposición a las RdP de alto nivel, el conjunto de la sincronía se redefine [218] con respecto a las RdP de bajo nivel.

Un nuevo paquete de UML llamado “DirectedSynchronousChannel” se propone en este trabajo para extender a las RdP de alto nivel. Este nuevo paquete introduce el concepto de conjuntos de sincronización.

Al igual que en [218], un conjunto sincronización es un canal síncrono dirigido que vincula una transición (el maestro) a un conjunto de transiciones (los esclavos). Con el fin de cumplir con la norma, todas las ampliaciones de las RdP deben ser etiquetas o atributos. Una etiqueta o un atributo de un objeto contienen información sobre el mismo. Una etiqueta se muestra como una notación de texto del objeto, mientras que un atributo conduce a cambios de la forma del objeto. Se propone el conjunto de sincronía en el presente documento como una notación de la página (RdP). Esta notación está compuesta por una etiqueta y una estructura. La etiqueta es un texto no estructurado, que se utiliza para presentar alguna información informal sobre el conjunto de la sincronía. La estructura utilizada para presentar la sincronía se establece como un árbol de

síntaxis abstracta en XML, la identificación de la transición principal y las transiciones de los esclavos.

La semántica de una sincronía es fijada para las RdP de alto nivel, por lo que se extiende a la semántica de las RdP de bajo nivel en lo siguiente:

- En una RdP bajo nivel, cada vez que se dispara una transición maestra, el conjunto sincronía propaga un evento para las transiciones esclavas, que se disparará.
- Mientras que en una RdP de alto nivel, cada vez que se dispara una transición maestra, el conjunto sincronía propaga un mensaje (en lugar de un evento) a las transiciones de esclavos. El mensaje lleva los datos requeridos por las anotaciones de arco de las transiciones de esclavos. Como un conjunto "sincronía" es un canal síncrono dirigido, maestro a esclavo cuando las transiciones están sincronizados.

Se aplica el paradigma de retardo cero entre transiciones maestro y esclavo de cada conjunto sincronía.

Análisis de Resultados
Una operación de división, ha sido ampliada para permitir la descomposición de las RdP de alto nivel, se ha aplicado con éxito en una serie de estudios de caso para habilitar su aplicación a RdP de alto nivel.

Se ampliaron las RdP con canales síncronos, que se introducen por la operación de la red maestra. Cuando se aplica a RdP de alto nivel, la división tiene que realizar tareas adicionales con el fin de preservar las propiedades del modelo inicial.

Como trabajo futuro está en desarrollo una herramienta para realizar automáticamente la nueva operación de división.

Este trabajo [190], es un extensión del trabajo presentado en [134] CPN y muestra como dividir RdP IOPT Coloreadas , haciendo una generalización de las tres reglas anteriores. Aquí se profundiza el análisis de los trabajos previos y la fundamentación alcanza también a otros tipos de redes.

También, se hacen restricciones para la división y la tercera regla no mantiene la semántica. La división requiere de canales síncronos con más capacidad de comunicación que el requerido en los anteriores y con repetición de transiciones y/o plazas.

Esto último hace que la sobrecarga introducida por el software de ejecución de la red es importante, puesto que la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones, el disparo y hacer la comunicación. Las características que se implementan son las mostradas en las Tabla 70, Tabla 71 y Tabla 72.

Tabla 70: Aportes de[190] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[190]	No	No	Si	Si	Si	No	No

Tabla 71: Aportes de [190] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[190]	No	No	Si	Si	No	No

Tabla 72: Aportes de [190] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[190]	No	Si	Si	Si	Si	No	Si	Si	No	Si

Composición de Modelo mediante la Reutilización de Módulos basados en RdP

En las últimas décadas, el diseño de controladores de sistemas embebidos se enfrenta a una complejidad creciente y sostenida. Esto trae nuevos retos, de los que se pueden beneficiar los enfoques de desarrollo basados en el modelo. Las RdP son uno de los formalismos adecuados para ser utilizado como lenguaje de especificación del sistema. En este trabajo [219] se propone un método para la composición del modelo, a partir de sub-modelos que representan componentes concurrentes, y confiando en su composición utilizando un enfoque de abajo hacia arriba, por lo que se dá soporte a la reutilización de módulos. Este enfoque de abajo hacia arriba está integrado con un enfoque de descomposición de arriba hacia abajo, del modelo del sistema, utilizando la operación de división, donde se generan sub-modelos asociados a componentes. En este sentido, la técnica propuesta es compatible con un uso equilibrado de arriba hacia abajo y los enfoques de abajo hacia arriba a la modelación de sistemas que utilizan el desarrollo basado en modelos de controladores de sistemas embebidos donde las RdP juegan un papel central.

Secciones de interés del trabajo
El enfoque está basado en el desarrollo del modelo, y tiene un impacto importante en el diseño de sistemas embebidos. En cuanto a los sistemas se vuelven más y más complejos, la adopción de técnicas que pueden mejorar el proceso de desarrollo es de suma importancia.

Una de las técnicas se basa en el uso de modelos que se pueden utilizar como formalismos de lenguaje de especificación de sistema con fines de verificación del mismo, para soportar la generación automática de código y el despliegue en plataformas de implementación.

Las RdP son uno de esos formalismos. Debido a que su representación gráfica, las RdP pueden ayudar a la comunicación entre los equipos de desarrollo. Por otra parte, debido a su fuerte soporte matemático, que puede ser utilizado para las verificaciones automáticas de propiedades. Además, si se tiene en cuenta el diseño del sistema integrado y la comunicación con su entorno es necesario modelar el estímulo externo del sistema, tales como señales de entrada/salida y eventos. Para modelar explícitamente e integrar en el modelo de RdP los estímulos del entorno físico, se propone una nueva clase de RdP IOPT [46]. Esta clase de RdP es una extensión de las RdP lugar/transición [7] que se utiliza ampliamente, es decir, dentro de la modelización de automatización de fábrica y diseño de sistemas embebidos. La representación de las RdP IOPT se han extendido al lenguaje “Markup Language” (PNML) [95]. También se extendió para especificar señales y eventos, así como otras propiedades utilizadas en clase RdP IOPT, de

acuerdo con la sintaxis RELAX NG, como se propone en [46]. Más recientemente, una representación Ecore, equivalente a la gramática RELAX NG se propuso en [46], para la integración a los marcos de desarrollo actuales.

Desde el punto de vista de la ingeniería, otra cuestión importante es la posibilidad de contar con herramientas de generación automática de código. Para la RdP IOPT, se desarrollaron dos generadores de código automático: una plataforma de generación de código C independiente [49], para las implementaciones de software; y otro código de generación de VHDL [220], para implementación de hardware. En el desarrollo de sistemas complejos (o sistemas de automatización complejas), es a menudo insuficiente, o incluso inviable, modelar todo el sistema y asumir una ejecución centralizada. Las soluciones de ejecución distribuidos pueden contemplar diferentes requisitos del sistema, es decir, el consumo de energía, los costos y el rendimiento.

Adicionalmente, se hace posible para tomar ventaja en las técnicas de co-diseño hardware-software. Para ese fin, se propuso un operador de RdP que permite la división del modelo de RdP en un conjunto de sub-modelos[134]. Estos subsistemas de modelos estarán asociados con los componentes de hardware o software, que se implementarán en plataformas finales de ejecución.

En este trabajo, la clase de RdP IOPT se considera como lenguaje de especificación de sistema susceptible de soportar modelado usando de arriba hacia abajo y / o enfoques de abajo hacia arriba, teniendo en cuenta los componentes obtenidos de la operación de división.

Los autores extienden la división de las RdP tal cual la realizarán en [134] y transforman en sub-modelos a los módulos para realizar la división del sistema y luego resolverlo.

Los enfoques de arriba hacia abajo permiten al modelador iniciar el modelo de sistema para generar componentes que se pueden ejecutar al mismo tiempo, mientras que los enfoques ascendentes pueden soportar la composición de modelos de RdP con sub-modelos agregados, produciendo nuevas funcionalidades. De esta manera, la reutilización de modelos se admite explícitamente, que es muy importante desde el punto de vista de la ingeniería.

Análisis de resultados
Este trabajo [219] muestra las ventajas de dividir RdP IOPT, haciendo uso de tres reglas. Es importante el análisis de los trabajos previos y la fundamentación.

La división requiere de canales síncrono y repetición de transiciones y/o plazas.

Esto último hace que la sobrecarga introducida por el software de ejecución de la red sea importante, puesto que la ejecución de la red requiere múltiples ciclos para determinar la sensibilidad de las transiciones, el disparo y hacer la comunicación.

El método permite resolver grandes modelos con una técnica simple y no muy restrictiva. Las características que se implementan son las mostradas en las Tabla 73, Tabla 74 y Tabla 75.

Tabla 73: Aportes de[219] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[219]	No	No	Si	Si	Si	No	No

Tabla 74: Aportes de [219] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[219]	No	No	Si	Si	No	No

Tabla 75: Aportes de [219] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[219]	No	Si	Si	Si	Si	No	Si	Si	No	Si

Ejecución de transiciones de RdP distribuidas y resolución de conflictos a través de la transformación de modelos

Las RdP son un formalismo de modelado adecuado para especificar la concurrencia, haciéndolas apropiadas para modelar sistemas embebidos distribuidos y con procesos concurrentes. Para realizar una implementación distribuida de un sistema especificado por un modelo de RdP, se requiere hacer la partición y distribución del modelo en un conjunto de sub-modelos distribuidos que interactúan. El trabajo [221] presenta un modelo de transformación para permitir la ejecución distribuida y la solución de conflictos en RdP, permitiendo de esta manera la distribución de los procesos sin conflicto. Esta transformación tiene algunas limitaciones e introduce algunos pequeños cambios de comportamiento, de lo contrario no sería posible distribuir conflictos en RdP (se sabe que los conflictos deben ser resueltos a nivel local).

Esta transformación se aplicó a un sistema no-autónomo de RdP, que junto con las herramientas, permite el desarrollo de sistemas embebidos. Las herramientas IOPT (disponible en [79]) incluyen un editor,la comprobación de modelo, el código automático y los generadores de código para las plataformas de software y hardware.

Análisis de resultados
Este trabajo [221] muestra cómo resolver los conflictos que introduce la división propuesta en [134], por lo que las conclusiones son las mismas que en el caso referido.

Distintas interfaces de Comunicación entre el Procesador/Controlador y los Procesos

Diseño de Código para Sistemas Embebidos a partir de Modelos con RdP No Autónomas

En este artículo [47] se presenta un entorno de desarrollo de diseño de sistemas embebidos basado en RdP. El trabajo utiliza una extensión del modelo Plaza/Transición de RdP que permiten la asociación de las señales de entrada externas a las transiciones y la asociación de señales de salida externas a las transiciones, con el fin de cambiar el marcado. Además, la clase proporciona soporte para la especificación de los eventos de entrada y salida. Esta extensión permite el uso de RdP Lugar / Transición para la especificación de los controladores. El documento también muestra cómo generar código ejecutable a partir de esas especificaciones y

los detalles del algoritmo utilizado para la ejecución escalonada del modelo de red. El código de la herramienta y arquitectura del generador se discute brevemente. Finalmente, se presenta un ejemplo del sistema de automatización.

Secciones de interés del trabajo
El presente trabajo, se centra en el diseño de sistemas embebidos. A grandes rasgos, un sistema embebido puede ser visto como un sistema de eventos discretos, con restricciones de tiempo real y capacidades de procesamiento de datos. En este sentido, la selección de un modelo de computación debe empezar por considerar la naturaleza reactiva del sistema embebido. Sin embargo, las limitaciones de procesamiento en tiempo real y los datos también deben ser considerados.

Las RdP son un modelo de computación ampliamente utilizado para la especificación de los controladores de sistemas de eventos discretos, ver por ejemplo [122]. Donde para el modelado de sistemas de eventos discretos por RdP, se proponen algunas extensiones, por ejemplo: el tiempo, la conexión con el entorno a través de la especificación de los eventos externos, dependencias entre transiciones, prioridades, etc.

En este trabajo se utiliza una extensión sencilla e intuitiva a las RdP Plaza/Transición [80, 99], que son las clase de RdP más usadas. Esto permite su uso para el control de la especificación de sistemas de eventos discretos. Nos referimos al tipo de RdP Input -Output Place/net Transición como (IOPT). La extensión IO agrega un conjunto mínimo de anotaciones a las redes de P/T que permiten su conexión con el medio ambiente.

Dicho de otra manera, se extienden los modelos basados en RdP P/T al medio ambiente. Más específicamente, las redes IOPT permiten la especificación de las señales de entradas externas y eventos de entradas externas asociados al disparo de la transición. También se sustenta la especificación de eventos de salidas externas y señales de salidas externas como acciones.

En primer lugar se asocian a las transiciones, mientras que en segundo lugar se asocian a las marcas.

Sintácticamente, las redes IOPT no proporcionan nuevas características a las redes de Plaza/Transición, solamente agregan las conexiones con el medio ambiente.

El planteamiento de fondo corresponde al diseño y la implementación del controlador directo. Esto se identifica como el "enfoque de sistema de automatización" en [122]. También es el método utilizado por [102, 222] y otros, cuando se utilizan RdP sincronizada y RdP interpretadas.

Una clase de RdP "no Autónoma"
Esta sección se define formalmente una clase de RdP no autónoma llamada IOPT. Corresponde a una red P/T extendida con eventos externos y señales. La Definición 7, supone la existencia de un lenguaje que permite especificaciones como expresiones algebraicas, variables y funciones. El conjunto de expresiones booleanas se designa por BE. Var (E) devuelve el conjunto de variables presentes en una expresión E. Para evitar el análisis sintáctico, el lenguaje de etiquetado debe ser el mismo que el generado por la herramienta PnGenerator. Actualmente los autores están utilizando ANSI C.

Definición 7: IO RdP

Entrada/Salida Lugar / Transición RdP, es una tupla N, donde:

$$N = (P, T, A, w, IS, OS, IE, OE, ov, ie, oe, sig, soa)$$

Que satisface la siguientes requisitos:

1) P, es un conjunto finito de lugares.
2) T, es un conjunto finito de transiciones (disjunta de P).
3) A, es un conjunto de arcos tal que A ⊆ (P × T) ∪ (T × P).
4) W, es una función de peso arco definido de A en N0.
5) IS, es un conjunto de señales de entrada.
6) OS, es un conjunto de señales de salida.
7) IE, es un conjunto de eventos de entrada.
8) OE ,es un conjunto de eventos de salida.
9) ov , es una función de valor de salida definido a partir de (OS ∪ OE) en N0.
10) ie , es una función de entrada de evento definido de T a IE.
11) oe es una función de evento de salida definida de T a OE.
12) sig, es una función de protección contra la entrada de señal definida de T a E ⊆ SER donde Var (E) ⊆ ES.
13) soa es una función de acción de salida de señal definida desde P a E ⊆ SER donde Var (E) ⊆ OS

Una marca de una red de N del tipo IOPT es una función M: P → N0. N junto con un marcado M0 (marcado inicial) se llama un sistema de red: $NS = (N, M0)\ o$

$$NS = (P, T, A, w, IS, OS, IE, OE, ov, ie, oe, sig, soa, M0)\ .$$

Las señales de entradas externas se adjuntan a las transiciones ya que las condiciones de eventos del punto 10 o dentro de las expresiones de guarda en la punto 12 (ambos de la Definición 7), mientras que las señales de salida pueden ser modeladas de dos maneras:

- token / lugares, de una manera similar a Moore (por la Definición 7 punto 6);
- señales de salida de eventos producidos cuando los cambios de estado, es decir, cuando se dispara una transición (por la Definición 7 punto 11).

En cuanto a las reglas de disparo de transición, se adoptó el paradigma de sincronizado [222] en el sentido de que cada vez que una transición está activada, y la condición de evento externo asociado es cierta , la transición se dispara .

La evolución del modelo sólo es posible en instantes específicos; el período entre dos disparos consecutivos es el ciclo de tratamiento o un paso. Se utiliza la estrategia de máximo paso, lo que significa que todas las transiciones que están habilitadas y listas son disparadas en el mismo paso.

El controlador y el medio ambiente

Un entorno de desarrollo que soporta IOPT se está desarrollando actualmente. La arquitectura del sistema se muestra en la Figura 32, que se ha descompuesto en tres áreas:

- Interfaz de entrada, centrada en la edición de la aplicación (PnEditor) que se encarga de la interacción gráfica entre el diseñador y el modelo;
- Modelo ejecutable que es la aplicación de generación de código (PnGenerator), es el núcleo del entorno de desarrollo. Es responsable de la generación automática de código, basado en la descripción del modelo de RdP proporcionada por el PnGenerator. En la etapa actual de desarrollo, prevee integrar el analizador y el simulador, pero aún no ha sido implementadas. Además, el código generado actualmente es ANSI C, que es

susceptible de ser utilizado con una amplia gama de plataformas de microcontroladores de bajo costo, para las estaciones de trabajo de alto rendimiento. También prevee la generación automática de código VHDL, como un elemento clave para el apoyo al desarrollo del sistema integrado, utilizando hardware-software y técnicas de co-diseño. En este sentido, el énfasis actual es la implementación del componente PnGenerator. Que es el controlador en que se basa el software, soportado por los microcontroladores específicos;

- La interfaz de salida es la plataforma específica. Se trata de un sistema embebido que recibe el código generado.

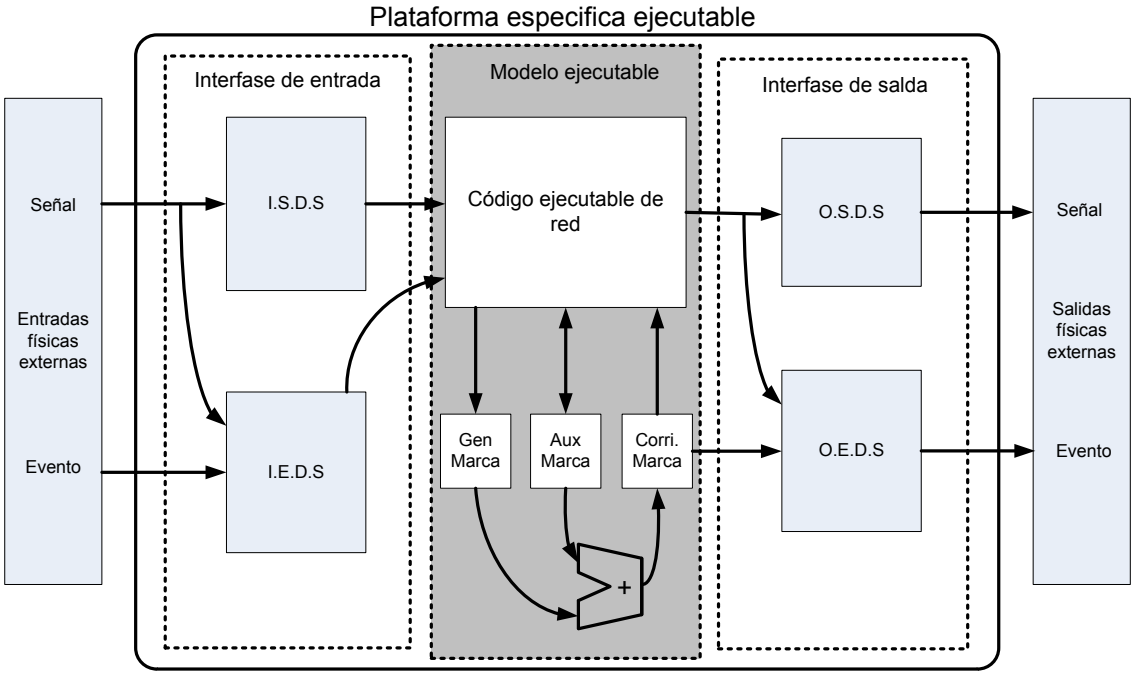


Figura 32: Framwork de ejecución de una IOPN, para la generación automática de código

Análisis de Resultados

Como elemento clave del entorno presentado, destacamos el hecho de que el mismo código ejecutable, generado automáticamente, se puede utilizar con tres diferentes objetivos: para la ejecución dentro de la plataforma en tiempo real; para soportar el análisis de propiedad del modelo (es decir, a través de la construcción del espacio de estado equivalente); y para soportar la simulación interactiva. Desde el punto de vista de la ingeniería, esta son las características principales del entorno de desarrollo.

Es un proyecto que fundamenta la comunicación entre el medio ambiente y las RdP, haciendo uso de eventos y señales.

Justifica y basa el desarrollo de la arquitectura haciendo uso de las RdP IOPT, para el diseño de un sistema embebido. Se expone el paradigma de sincronización y se expone como estas redes se apartan del formalismo de las RdP ordinarias.

Restringe la evolución del sistema a un reloj externo, para realizar disparos múltiples en un solo ciclo.

La arquitectura de este proyecto está en desarrollo, por lo que hay que considerar que muchas de estas características son proyectos futuros.

Este desarrollo es importante para ser aplicado al desarrollo de sistemas embebidos complejos.

Las características que se obtiene en [47] son mostradas en las Tabla 76 y Tabla 77.

Tabla 76: Características del tipo de redes implementadas en el PP, en el trabajo de [47]

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[47]	Si	Si	No	No	No	No	No	No	No	No	No	No	No

Tabla 77: Características del tipo de eventos y programación implementadas en el PP, en el trabajo de [47]

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB
[47]	Si	Si	No	Si	No	No	No	No

Eventos para el Modelado con RdP IOPT para Interacción Humana-Sistema

En este trabajo [223], se presenta un conjunto de tipos de eventos para el modelado de la interacción hombre-sistema, y su uso con RdP IOPT. Los autores han considerado, como evento no autónomo, el cálculo de la variación con respecto a un umbral específico de señales de entrada. Estas variaciones son obtenidas en dos pasos de ejecución consecutivos, que han sido definidos para los modelos IOPT.

Se proponen otros tipos de eventos, lo que permite la definición de un evento no sólo sobre la base de un umbral en el valor de la señal, sino también como un cambio en la variación de la señal. El concepto de eventos retardados considera dos valores en diferentes pasos de tiempo (distintos de los pasos consecutivos). Este concepto se puede incluir con los anteriores tipos de eventos. El concepto de eventos compuestos también es introduce, con el fin de crear eventos dependientes de dos o más eventos en la misma señal, en el mismo paso de tiempo, así como en diferentes pasos de tiempo. Finalmente, un ejemplo se utiliza para ilustrar los conceptos presentados.

Análisis de Resultados

Este trabajo [223], es un aplicación del tipo de eventos que se integran con las RdP IOPT, y muestra como incorporarlos con los ya tratados y su importancia. En todos los trabajos del autor, no se generalizan los eventos que el sistema puede tratar con el fin de programarlos, lo que hace es construir una biblioteca e incorporarla al sistema final como un módulo obtenido de una librería que se compila. Las características que se implementan son las mostradas en las Tabla 78, Tabla 79 y Tabla 80.

Tabla 78: Aportes de[223] con respecto a la ejecución

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[223]	No	No	Si	Si	Si	No	No

Tabla 79: Aportes de [223] con respecto a la programación

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[223]	No	No	Si	Si	No	No

Tabla 80: Aportes de [78] con respecto a la división de una red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[223]	No	Si	Si	Si	Si	No	Si	Si	No	Si

Ecore basado en RdP del tipo RdP IOP para la Definición del Modelo

En este trabajo [224] los autores presentan una nueva meta-modelo de RdP IOPT (entrada-salida Plaza-Transición) y su definición utilizando una gramática RELAX NG. Esta nueva versión introduce cambios en la meta-modelo anterior para garantizar que la representación del modelo sigue las directrices de la Norma Internacional para extender PNML y soporta totalmente la generación automática de espacio de estado. La validación de esta meta-modelo, revisada y el PNTD correspondiente se realizan a través de estudios de casos. Los nuevos conceptos introducidos en la meta-modelo han demostrado su validez y utilidad, en la generación automática de código ejecutable y en el espacio de estados. La corrección de la gramática RELAX NG también fue validada a través de casos de estudio.

Análisis de Resultados

En [224] los autores extienden el lenguaje de para expresar el modelo, pero no aportan nuevas características a como relacionar la red con el controlador o sobre el controlador.

Desarrollo de Sistemas basados en Módulos con RdP

El desarrollo de modelo en el ámbito de los sistemas integrados de hardware y software para el co-diseño ha demostrado ser eficaz para manejar la complejidad del sistema. Varios formalismos de modelado son ampliamente usados dentro de esta área. El trabajo [225] presentado por los autores tiene como objetivo contribuir para el uso de las RdP como el lenguaje de especificación a nivel de sistema dentro del desarrollo, basado en modelos de sistemas embebidos que utilizan técnicas de co-diseño.

El modelado de sistema distribuido en módulos y componentes, listos para ser implementados en hardware o software e instanciados en una plataforma distribuida es una preocupación importante dentro de las metodologías de co-diseño.

Hacer uso de las RdP como el modelado subyacente y las operaciones formales para la división con el fin de descomponer el modelo en varios sub-modelos. Donde los sub-modelos resultantes

pueden ser vistos como componentes y ejecutados en la comunicación paralela, haciendo uso de canales síncronos dirigidos. En este trabajo se presenta la forma de modificar el modelo dividido con el fin de ser considerado como módulos componibles, para dar soporte a la reutilización, que es un factor de impacto clave dentro de la ingeniería.

Las técnicas propuestas se basan en mantener la interfaz del módulo sin cambios, con el fin de soportar la integración del módulo en nuevas situaciones.

Análisis de Resultados
En este trabajo [224] los autores introducen el concepto de componentes para expresar el modelo, pero no aportan nuevas características de cómo relacionar la red con el controlador.

Una Herramienta para la Generación de Estados de RdP IOPT

Este trabajo [226] presenta la herramienta IOPT2SS, utilizada para generar automáticamente gráficos de espacio de estado asociados con RdP IOPT. La nueva herramienta incorpora el carácter no autónomo de las RdP IOPT, donde el disparo de las transiciones se ve limitada por los eventos de entrada y las señales de entrada externa (expresado como guardas de transición). Por otra parte, las transiciones pueden desencadenar la aparición de eventos o señal de salida y el lugar marcado puede activar señales de salida. Para alcanzar el nivel necesario de generación rápida del complejo de estados, en tiempo razonable, la herramienta emplea una estrategia de compilación de creación automática de un programa en C optimizado. Cuando se ejecuta, el programa creará un archivo XML jerárquico que contiene el gráfico de espacios de estado.

El archivo XML de salida se puede utilizar para la comprobación del modelo y análisis de la propiedad. La aplicación usa lenguajes de consulta XML estándar. Por último, el archivo de salida se puede convertir en SVG (Scalable Vector Graphics), lo que permite la visualización gráfica de tamaño pequeño a mediano de los espacios de estado. La nueva herramienta ha sido implementada para el uso en conjunto con otras y se puede utilizar como una herramienta independiente o como un bloque de construcción en los marcos de nivel superior, con una fácil integración en aplicaciones web y entornos de desarrollo integrados.

Análisis de Resultados
En este trabajo [226] los autores introducen una herramienta IOPT2SS, utilizada para generar automáticamente gráficos de espacios de estado asociados con RdP IOPT, pero no aportan nuevas características a como relacionar la red con el controlador.

Antecedentes Previos usados en la Patente

Existen diferentes maneras de implementar las RdP: en software y en hardware. Las primeras pueden ser realizadas a través de lenguajes o desarrollos de hardware específicos como los referenciados en los siguientes párrafos; donde se da la referencia y se especifica las diferencias y desventajas con respecto a este invento.

En los trabajos de Gary A. Bundell [60] “An FPGA Implementation of the Petri net Firing Algorithm”. La diferencia radica en que se implementa solo el algoritmo de disparo de una transición de la RdP por medio de un bloque en una FPGA. Esto es un módulo primitivo, donde falta el resto de los componentes de una RdP y su interconexión para componer el sistema por módulos programables, destacándose que no está implementado sobre la base de la ecuación de estado.

En los trabajos presentados por Hideki Murakoshi y colaboradores [129] "A High Speed Programmable Controller Based on Petrinet", se describe un controlador basado en la tablas obtenidas de la “matriz de Petri”. Este controlador no es programable y se necesitan demasiados ciclos de reloj (más de veinte) para realizar un disparo. Además no permite la programación de prioridades, ni brazos con peso mayor a uno, no tiene brazos inhibidores, ni plazas acotadas, tampoco tiene brazos de reset, no cuenta con detección automática de interbloqueo ni disparos automaticos.

En los trabajos de Sergio C. Brofferio, “A Petri Net Control Unit for High-speed Modular Signal Processors” [17], se ha implementado un controlador compuesto por módulos, no haciendo uso de la ecuación de estado. Esto no permite programar el sistema en tiempo de ejecución, ni manejar prioridades, ni plazas acotadas, tampoco tiene brazos de reset y no detecta en forma automática el interbloqueo ni disparos sistólicos.

En los trabajos realizados por Ramón Piedrafita Moreno y José Luis Villarroel Salcedo [19] "Adaptive Petri Nets Implementation. The Execution Time Controller", se programa un controlador por software haciendo uso de un lenguaje de alto nivel; pero no se ha implementado prioridades, ni brazos inhibidores, ni plazas acotadas, no tiene brazos de reset, no tiene disparos sistólicos y no detecta automáticamente el interbloqueo.

En los trabajos presentados por Xianwen Fang y colaboradores [227] “A Study about the Mapping of Process- Processor based on Petri Nets”, se modelan procesos con una RdP con el fin de evaluar y anticipar el comportamiento del sistema, pero no se usa la ecuación de estado de la RdP como programa. Solamente lo codifica con un lenguaje de alto nivel con el fin de anticiparlo y modelarlo.

En el trabajo de Murakoshi y colaboradores [228] “Memory reduction of fire unit for Petri net controlled multiprocessor”, se presenta un controlador basado en dos matrices que no permite la programación en tiempo de ejecución y se necesitan demasiados ciclos (más de cinco) para realizar un disparo. El sistema además no permite la programación de prioridades, ni brazos con peso mayor a uno, no tiene brazos inhibidores, ni plazas acotadas, tampoco tiene brazos de reset, disparos sistólicos, ni detección automática de interbloqueo.

El trabajo realizado por Wegrzyn y colaboradores [95] "Coloured Petri Net Model of Application Specific Logic Controller Programs" modela y construye un sistema en VHDL con bloques funcionales y no con la ecuación de estado. El sistema resultante no es programable en tiempo de ejecución, no maneja prioridades ni tiene plazas acotadas; tampoco tiene brazos de reset, no tiene disparos sistólicos y no detecta en forma automática el interbloqueo.

El trabajo de [136] “Supervisory Controller Design for Timed Petri Nets”, implementa un nuevo tipo de RdP temporal, ejecutándola en un programa que hace uso de un lenguaje de alto nivel, pero no es una ejecución por hardware. El algoritmo no es programable en tiempo de ejecución y tampoco implementa el manejo de prioridades ni plazas acotadas; tampoco tiene brazos de reset, no tiene disparos sistólicos y no detecta en forma automática el interbloqueo.

Finalmente, el trabajo de Murakoshi y colaboradores "Hardware Architecture for Hierarchical Control of Large Petri Net" [185], es una continuación del explicado en la referencia [228] donde introduce el concepto de red jerárquica para mitigar el problema de tamaño de la matriz. El controlador es basado en dos matrices, pero no permite la programación en tiempo de ejecución y se necesitan demasiados ciclos (más de cinco) para realizar un disparo. Además no permite la

programación de prioridades, ni brazos con peso mayor a uno, no tiene brazos inhibidores ni plazas acotadas; tampoco tiene brazos de reset ni disparos sistólicos y no posee detección automática de interbloqueo.

Todos los trabajos que implementan RdP por hardware lo hacen por módulos, lo que implica programar el hardware cuando cambia la red y no por la ecuación de estado.

También se puede destacar que los controladores de Petri que usan su formalismo programados con un lenguaje de alto nivel no son ejecutados a partir de la ecuación de estado.

Esto genera una serie de impedimentos, a saber: que puedan ser reprogramados en forma directa durante su ejecución, la programación de prioridades, no permite que sus brazos tengan peso mayor a uno ni que posean brazos inhibidores, que las plazas no estén acotadas, también impide programar brazos de reset, no tienen disparos sistólicos y no se detecta en forma automática el interbloqueo en tiempo de ejecución.

Conclusión de las Características Implementadas en los Trabajos Estudiados

Tabla 81: Resumen de los trabajos realizados con respecto a los tipos de redes ejecutadas

Con respecto a los tipos de redes													
Referencia	RO	RPL	APU	APE	AH	AR	Ar	AS	ASH	AFB	TT	TD	RdPJ
[1]	No	No	Si	No	No	No	No	No	No	No	No	No	No
[2]	No	No	Si	No	Si	No	No	No	No	No	No	No	Si
[6]	Si	No	Si	No	Si	No	No	No	No	No	No	No	Si
[19]	No	No	Si	No	Si	No	No	No	No	No	No	No	Si
[21]	Si	No	Si	No	Si	No	No	No	No	No	No	No	No
[22]	Si	No	No	No	No	No	No	No	No	No	No	No	No
[39]	Si	No	No	No	No	No	No	No	No	No	No	No	No
[35]	Si	Si	Si	Si	No	No	No	No	No	No	No	No	No
[97]	Si	No	No	No	No	No	No	No	No	No	No	No	No
[40]	Si	No	No	No	No	No	No	No	No	No	No	No	No
[110]	Si	No	No	No	No	No	No	No	No	No	No	No	No
[124]	Si	No	No	No	No	No	No	No	No	No	No	Si	No
[133]	No	Si	Si	Si	No	No	No	No	No	No	No	No	Si
[155, 181]	Si	Si	No	No	No	No	No	No	No	No	No	No	No
[149]	No	No	Si	No	Si	No	No	No	No	No	No	No	Si
[57]	Si	Si	No	No	No	No	No	No	No	No	No	No	No

Tabla 82: Resumen de los trabajos realizados con respecto a las prioridades, eventos y programación

Con respectos a las prioridades, eventos y programación								
Referencia	PD	EE	TEE	ES	TES	PMV	PTE	DIB

[1]	No	No	No	No	No	No	No	No
[2]	No	No	No	No	No	No	No	No
[6]	No	No	No	No	No	No	No	No
[19]	No	No	No	No	No	No	No	Si
[21]	No	No	No	No	No	No	No	No
[22]	No	No	No	No	No	No	No	No
[39]	No	No	No	No	No	No	No	No
[35]	Si	Si	No	Si	No	No	No	No
[97]	No	No	No	No	No	No	No	No
[40]	No	No	No	No	No	No	No	No
[110]	No	No	No	No	No	No	No	No
[124]	No	Si	No	Si	No	No	No	No
[133]	Si	Si	No	Si	No	No	No	No
[155, 181]	Si	Si	No	Si	No	No	No	No
[149]	No	No	No	No	No	No	No	No
[57]	Si	Si	No	Si	No	No	No	No

Tabla 83: Resumen de los trabajos realizados con respecto la ejecución de las redes

Con respecto a la ejecución							
Referencia	EHS	EHP	RIS	DU	DM	DIR	DIS
[78]	No	No	Si	Si	Si	No	No
[59]	No	No	Si	Si	No	No	No
[49]	No	No	Si	Si	Si	No	No
[106]	No	No	Si	Si	No	No	No
[120]	No	No	Si	Si	No	No	No
[34]	No	No	Si	Si	No	No	No
[50]	No	No	Si	Si	Si	No	No
[46]	No	No	Si	Si	Si	No	No
[134]	No	No	Si	Si	Si	No	No
[190]	No	No	Si	Si	Si	No	No
[219]	No	No	Si	Si	Si	No	No
[223]	No	No	Si	Si	Si	No	No

Tabla 84: Resumen de los trabajos realizados con respecto la programación de la red

Con respecto a la programación de la red						
Referencia	PD	LE	PPSR	LEP	PHTR	PSTR
[78]	No	No	Si	Si	No	No
[59]	No	No	Si	Si	No	No
[49]	No	No	Si	Si	No	No
[106]	No	No	Si	Si	No	No
[120]	No	No	Si	Si	No	No
[34]	No	No	Si	Si	No	No
[50]	No	No	Si	Si	No	No
[46]	No	No	Si	Si	No	No
[134]	No	No	Si	Si	No	No
[190]	No	No	Si	Si	No	No
[219]	No	No	Si	Si	No	No
[223]	No	No	Si	Si	No	No

Tabla 85: Resumen de los trabajos realizados con respecto a la división de la red

Con respecto a la división de la red										
Referencia	DA	DT	DP	SEA	RPD	CMTR	RD	RC	EMTR	RG
[78]	No	Si	Si	Si	Si	No	Si	Si	No	Si
[59]	No	No	Si	Si	Si	No	Si	Si	No	Si
[49]	No	No	Si	Si	Si	No	Si	Si	No	Si
[106]	No	No	Si	Si	Si	No	No	Si	No	Si
[120]	No	No	Si	Si	Si	No	Si	Si	No	Si
[34]	No	No	Si	Si	Si	No	Si	Si	No	No
[50]	No	No	Si	Si	Si	No	Si	Si	No	Si
[46]	No	No	Si	Si	Si	No	Si	Si	No	Si
[134]	No	Si	Si	Si	Si	No	Si	Si	No	Si
[190]	No	Si	Si	Si	Si	No	Si	Si	No	Si
[219]	No	Si	Si	Si	Si	No	Si	Si	No	Si
[223]	No	Si	Si	Si	Si	No	Si	Si	No	Si

Como podemos ver ningún autor implementa todas las características del PP / HPP o ha posibilitado la ejecución de distintas clases de RdP en un mismo PP o HPP. Con respecto a la división de las redes siempre el foco ha sido puesto en dividir una red con el fin de distribuirla, comprenderla o reducir su espacio de estados pero no para implementarla o ejecutarla. En cuanto al algoritmo de ejecución, en la investigación realizada, no se ha encontrado uno que integre a todos los tipos de red considerados en un mismo procesador, tanto para una FPGA como para software. En cuanto a los eventos, los trabajos en RdP IOPT, son muy completos, pero no consideran a la comunicación de IO como un módulo de entrada-salida programable.

Capítulo 3

Determinación de la Arquitectura para integrar el PP a un Sistema SMP

Resumen

En este trabajo se expone el análisis de un mecanismo para mejorar la sincronización y exclusión mutua entre procesos y/o hilos, haciendo uso de las RdP sincronizadas o no autónomas. La metodología usada es la simulación de un sistema SMP que integra un procesador de Petri (PP).

Introducción

La exclusión mutua y la sincronización demandan tiempos importantes en la ejecución de programas que tienen una alta relación de solicitudes de exclusión mutua y/o sincronización con respecto al procesamiento. Actualmente existen distintos componentes para implementar la sincronización y la exclusión mutua, los más usados son: pipes, semáforos, monitores y memoria transaccional. Estos son implementados por software y/o hardware, haciendo uso del sistema operativo e instrucciones específicas del procesador.

En este trabajo se propone, mediante el uso de un simulador de Procesadores Superescalares, determinar en qué lugar de la arquitectura de un SMP es posible introducir un módulo que hace uso de RdP sincronizadas para mejorar el desempeño de sistemas con alta demanda de sincronización y/o exclusión mutua.

Las RdP sincronizadas son adecuadas para el modelado de sistemas que requieren de concurrencia y paralelismo. También es simple representar estados locales y globales del sistema, lo que simplifica el diseño y construcción del sistema informático.

El trabajo se ha dividido en las siguientes secciones: Objetivos y Restricciones, Hipótesis, Desarrollo e Implementación, Simulaciones, Experimentos y Mediciones.

En Desarrollo e Implementación, se explica cómo se logra implementar dicha arquitectura dentro del simulador, cómo se encuentra formada esta arquitectura y el funcionamiento de la misma.

En el apartado de Simulaciones, Experimentos y Mediciones se muestran las tablas y gráficas correspondientes a las simulaciones realizadas con algoritmos que requieren de sincronización y exclusión mutua, donde se evidencia los resultados de las arquitecturas propuestas.

Objetivos

Determinar si es posible implementar e integrar un módulo de hardware PP a un sistema SMP que hace uso de múltiples procesos y/o hilos. Donde la responsabilidad de este módulo es mejorar los tiempos de sincronización y exclusión mutua entre procesos.

Objetivos Secundarios

Estos procesos deben emplear primitivas de sincronización y/o exclusión mutua, y ser integrable al simulador SESC.

Determinar los tiempos de sincronización y exclusión mutua entre procesos que se ejecutan en un SMP.

Hacer comparaciones entre el sistema original y el que hace uso del PP para su evaluación.

Determinar las condiciones y alcance donde se obtienen mejoras con el nuevo módulo.

Restricciones

No introducir y/o modificar el set de instrucciones (ISA) de los procesadores del SMP.

A nivel de programa introducir sentencias equivalentes al uso de las primitivas de sincronización y/o exclusión mutua (para lograr remplazarlas).

Implementar el hardware del nuevo módulo para interconectarse a nivel de bus con las distintas arquitecturas SMP.

Resultados Esperados

Se espera establecer la arquitectura óptima y sub óptima de interconexión del PP al sistema SMP que cumplan con las restricciones planteadas. Así mismo, una vez determinada la arquitectura (mecanismo) con el simulador, se espera obtener una estimación de los tiempos de ejecución de algoritmos fuertemente sincronizados y la cantidad de ciclos (clk) ejecutados a nivel de procesador en distinto casos, comparando los tiempos obtenidos con los procesadores sin el PP y con el PP. Obteniendo las bases para lograr la implementación del procesador.

SESC

SESC es un simulador cuya sigla significa: Simulador SuperESCalar. Éste modela distintas arquitecturas: single processors, CMPs (Comun Memory Processor), PIMs (Processing in Memory) y Thread Level Speculation (TLS). SESC es un proyecto iniciado por José Renau en Urbana-Campaing en el grupo IACOMA.

Modelado del Procesador en SESC

SESC emula la ejecución de las instrucciones, en un módulo con el ISA de un MIPS. El sistema acepta como entrada las instrucciones en orden (binaria) del programa, el MINT (que es un emulador de MIPS) genera las instrucciones. Éste retorna los objetos instrucción a SESC, los cuales son utilizados por el simulador de tiempos. Los objetos contienen la dirección de la instrucción, la dirección de load o strore en la memoria, los registros fuente y destino y las unidades funcionales utilizadas por la instrucción. Todos estos objetos son utilizados por el simulador para determinar el tiempo de cada instrucción que salen del pipeline. Puesto que la ejecución y la medición de tiempos están separadas se simplifica la programación y la depuración.

Componentes del SESC

Describiremos como una instrucción en una aplicación fluye a través del simulador.

Tipos de instrucciones

Instrucciones Estáticas (Static Instructions)

Son instrucciones del tipo R. Un ejemplo de este tipo de instrucción es add r1, r2, r3. Esto es implementado en la clase Instrucción (esesc/src/Libll/Instruction.cpp).

Instrucciones Dinámicas (Dynamic Instructions)

Son instancias específicas de instrucciones estáticas. Cuando cada instrucción estática es ejecutada una instrucción dinámica es creada.

Las instrucciones dinámicas son referenciadas teniendo en cuenta las dependencias entre las mismas.

Emulación

La emulación es controlada por la clase ExecutionFlow (se encuentra en esesc/src/Libll/). La interfaz hacia el nivel superior es a través de la función executePC (). La función executePC () en ejecución llama a exeInst () quien realiza algunos controles, como por ejemplo si la dirección de la instrucción es válida, y luego la ejecuta. Existe () retorna cero si ninguna instrucción pudo ser ejecutada. Un valor distinto de cero si la ejecución de una instrucción fue exitosa (si se trató de un load/store el valor retornado corresponde a la dirección virtual del dato accedido).

Pipeline

En SESC, el objeto GProcessor, un procesador genérico (se encuentra en esesc/ src/libcore/Gprocessor.cpp). Tiene como responsabilidad coordinar la interacción entre las diferentes etapas del pipeline.

La interfaz con el nivel superior al objeto es la función advanceClock (). La función advanceClock () mueve la instrucción por cada etapa del pipeline en cada pulso de reloj. Para la planificación y ejecución hay dos módulos, uno para enteros y otro para punto flotante. Cada uno tiene su propia ventana de planificación (Tomasulo con buffer de reordenamiento .ROB).

Fetch/Decode

Es la primera etapa del pipeline. En él, las instrucciones son tomadas desde la caché de instrucciones (en la configuración típica se leen 4 instrucciones por ciclo).

Predictor de salto

SESC soporta diferentes predictores de salto. La selección y el tamaño se realizan en el momento de ejecución. Desde que la unidad de predicción esta lista en la etapa de búsqueda, es manejada por la clase FetchEngine.

Emisión y planificación

Durante la etapa de emisión, las instrucciones son tomadas desde la cola de instrucciones donde estaban almacenadas, y enviadas a ventanas de planificadores individuales (planificadores de ventanas individuales) en un cluster particular. La planificación (schedule) hace referencia a cuando los operandos de entrada para una instrucción están listos, y la instrucción es planificada para ejecución.

Ejecución

El objeto DepWindow planifica una instrucción para ejecución o asigna punteros a las instrucciones que debe ejecutar primero.

De cada instrucción planificada hay que tener en cuenta los recursos que va a utilizar para su ejecución.

Finalización (Retirement)

En esta etapa de un procesador fuera de orden, las instrucciones que ya fueron ejecutadas, son removidas de la cabecera del buffer de reordenamiento en el orden de ejecución del programa original.

Llamadas a sistema

SESC no provee un sistema operativo, es por esto que el simulador debe capturar las llamadas a sistema y llevarlas a cabo en nombre de la aplicación. Para cada llamada estándar al sistema, SESC la transforma a una función de MINT. Por ejemplo, la función lstat () es redireccionada a la función mint_lstat de MINT.

Thread (Hilo de ejecución) API

- void sesc_init (): Inicializa la librería y arranca el planificador interno.
- sesc_spawn (void (*start routine) (void *), void *arg.long flags): Crea un nuevo hilo de control que se ejecuta concurrentemente con el proceso desde el cual se ejecuta (hilo principal).
- void sesc_wait (): bloquea el hilo hasta que uno de sus hijos termine su ejecución. void sesc_self (): retorna el ID del hilo.
- int sesc_suspend (int pid): suspende la ejecución del hilo con el pid pasado como argumento.
- int sesc_resume (int pid): pasa el hilo suspendido a la cola de ejecución.
- Int sesc_yield (int pid): Explícitamente deja el control de ejecución para el hilo cuyo pid se pasa como parámetro.
- void sesc_exit (): termina la ejecución del hilo actual.

Sincronización API

- void sesc_lock_init(slock t *lock): inicializa el bloqueo lock.
- void sesc lock (slock t *lock): toma el bloqueo de lock.
- void sesc unlock (slock t *lock): libera el bloqueo sobre lock.

Otras primitivas de sincronización

- void sesc barrier init (sbarrier t *barr): inicializa una barrera barr.
- void sesc barrier (sbarrier t *barr, long num proc): ejecuta la barrera barr.
- void sesc sema init (ssema t *sema, int initValue): inicializa un semáforo sema.
- initValue es la cantidad de recursos que contiene el semáforo.
- void sesc psema (ssema t *sema): emite un signal () sobre sema.
- void sesc vsema (ssema t *sema): emite un wait () sobre sema.

Impacto de la Sincronización y Exclusión Mutua en Sistemas SMP con Múltiples Hilos

Específicamente: la cantidad de instrucciones despachadas y el tiempo de procesamiento consumido en sincronización.

Análisis de la Sincronización entre Hilos

Esta medición se realiza usando el algoritmo del Escritor/Escritor. La selección de este algoritmo está motivada en su sencillez y la elevada sobrecarga por sincronización.

En este análisis se logró conocer el tiempo que el procesador utiliza para llevar a cabo la sincronización entre hilos, como ésta impacta en el rendimiento del mismo, y a su vez conocer la relación entre sincronizar y no sincronizar procesos. Las mediciones se llevaron a cabo incrementando la cantidad de elementos que se escriben en un array. La ejecución se realizó en el simulador SESC.

Los resultados de la medición se han organizado en siguiente Tabla 86.

Relación entre Hilos Sincronizados y Sin Sincronizar

Tabla 86. Relación Sincronización vs. No sincronización

Tamaño del array en byte	Sin sincronizar	Con sincronización	Relación CS/SS	%
2	0,15172	0,2625675	1,73060584	73
4	0,25672	0,3675675	1,43178369	43
8	0,46672	0,5775675	1,23750322	24
16	0,88672	0,9975675	1,12500846	13
32	1,72672	1,8375675	1,06419541	6

La relación entre la máxima sincronización, en la cual cada hilo escribe solo una vez el array por sincronización realizada, y dos hilos no sincronizados es arrojada en la Tabla 86 y Figura 33. En ella se observa claramente como la relación decrece a medida que el array compartido aumenta de tamaño, esto se debe a que la cantidad de escrituras aumentan para modificar todo el array compartido en cada ciclo mientras la cantidad de semáforos para sincronizar permanece constante, entonces la sobrecarga en la sincronización de los procesos se tornan despreciables con respecto al tiempo de escritura.

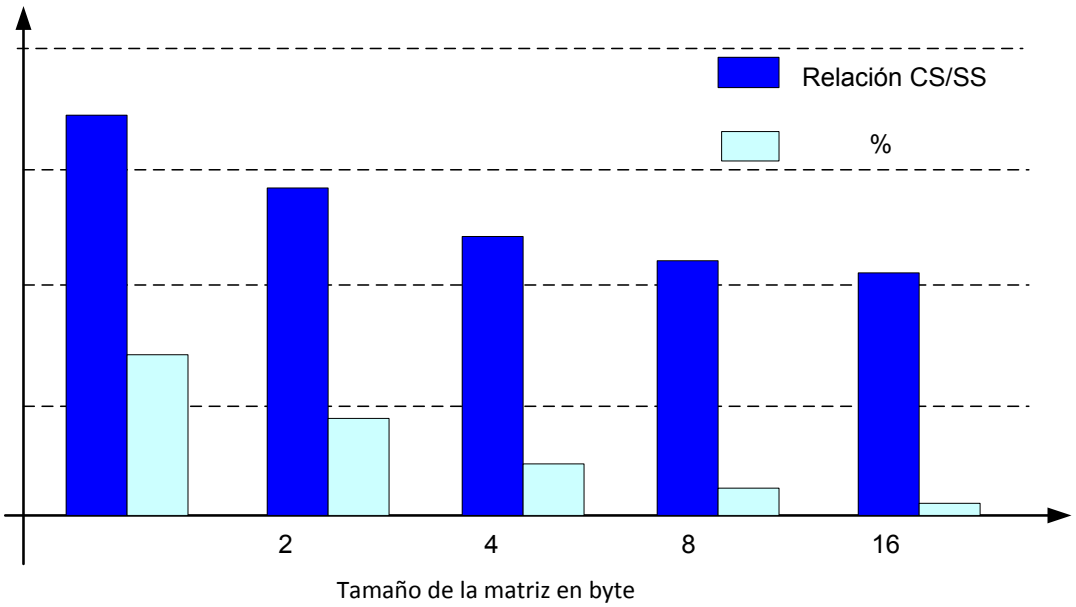


Figura 33: Relación de tiempos de ejecución según el tamaño de la matriz

Esto indica que los métodos de sincronización usados tienen una gran incidencia sobre hilos que realizan gran cantidad accesos a un recurso compartido.

Tabla 87: Tiempos de dos hilos escribiendo con sincronización y sin sincronización

Escrituras por sincronización	Tiempo sin sincronizar (ms)	Tiempo sincronizando (ms)	Relación porcentual CS/SS
1	35,363	38,469	**100,00**
2	35,363	36,976	**51,93**
4	35,363	36,169	**25,95**
8	35,363	35,766	**12,97**
16	35,363	35,565	**6,50**
32	35,363	35,464	**3,25**
64	35,363	35,413	**1,61**
128	35,363	35,388	**0,80**
256	35,363	35,376	**0,42**
512	35,363	35,369	**0,19**
1024	35,363	35,366	**0,10**
2048	35,363	35,365	**0,06**

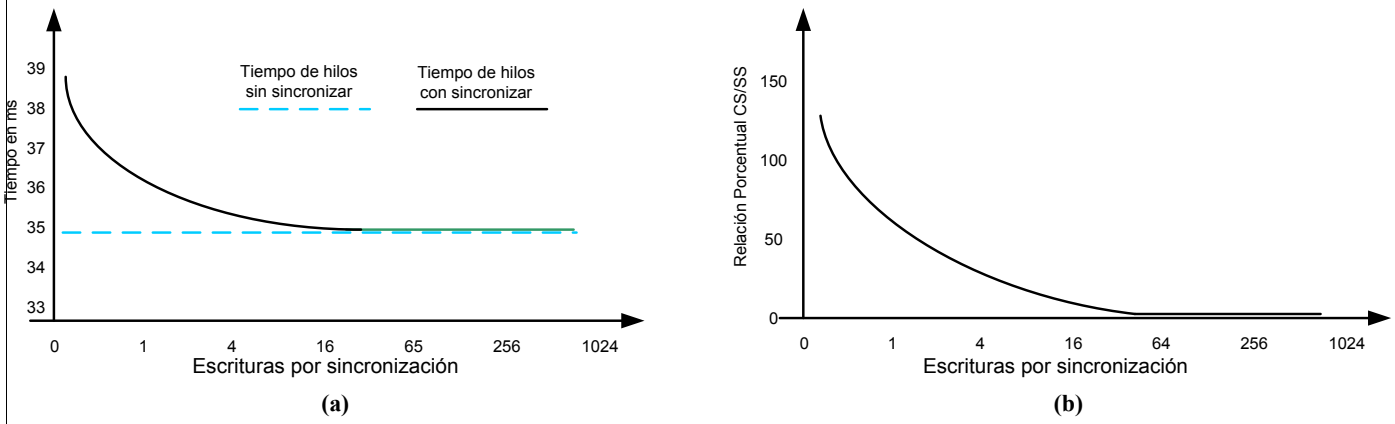


Figura 34: Tiempos con sincronización y sin sincronización entre procesos (hilos) Escritor/Escritor

Las Figura 34 (a) y (b), son el resumen del estudio realizado en donde se observa que la sincronización entre procesos (hilos) de un Escritor/Escritor emplea gran cantidad instrucciones de procesador haciendo incrementar notablemente el tiempo de ejecución. Así también de forma análoga se midió que el tiempo de ejecución y la cantidad de instrucciones ejecutadas disminuyen a medida que disminuye la cantidad de sincronizaciones relativas, como lo muestra la Figura 34 (b).

Teniendo en cuenta los objetivos y condiciones planteados, como así también los resultados obtenidos en el estudio realizado sobre el impacto de la Sincronización, se van a analizar distintas formas de implementar mecanismos de sincronización, y seleccionar el mejor, para llevar a cabo la sincronización entre hilos corriendo sobre diferentes procesadores a nivel de hardware. Con el fin de disminuir los tiempos de ejecución de procesos fuertemente sincronizados.

Para el desarrollo de este trabajo se estudiaron distintos algoritmos que requieren de accesos simultáneos a recursos compartidos, que son fuertemente sincronizados y expresados mediante un grafo de RdP. Estos son:

1. Escritor / Escritor.
2. Productor / Consumidor.
3. Algoritmo de Simulación de una Planta de Embalajes.
4. Algoritmo de Control de un mecanismo de Velocidad Crucero con detección de automóviles.

Las siguientes son las posibles formas que fueron contempladas para llevar a cabo la implementación:

- Nuevas instrucciones;
- Instrucciones de Entrada/Salida;
- Posiciones de memoria compartida;
- Sistema de colas con un procesador de eventos;
- Posiciones de memoria caché compartida.

A continuación se presenta una tabla donde se comparan las distintas opciones para implementar los mecanismos de sincronización, según una serie de condiciones que se deben cumplir.

Tabla 88: Distintas opciones para implementar sincronización con el PP

Alternativas	*No modificar el ISA*	*No requerir operaciones del SO*	*Cambios a nivel de programa mínimos*	*Cambios en hardware con módulos testeados*	*Viabilidad*	*Velocidad*
Crear nuevas Instrucciones	Si	No	No	Si	Si	Muy alta
Instrucciones de Entrada/Salida	No	Si	Si	No	No	Baja-
Sistema de Colas en el Procesador	Si	Si	Si	Si	No	Alta
Posiciones de memoria Compartida	No	Si/No	Si	Si	Si	Muy Baja
Posiciones de memoria caché compartida	Si	Si	Si	Si	Si	Alta

Análisis de las Opciones

Nuevas Instrucciones

Crear un nuevo tipo de instrucción cuya responsabilidad es la de sincronización, implica que se debe modificar el ISA de cada procesador, debiendo así modificar la arquitectura de cada procesador sobre el cual se desee implementar, haciendo esta posibilidad inviable por su falta de generalidad.

Instrucciones de Entrada/Salida

El simulador tiene un módulo de emulación MINT [228] para ejecutar las instrucciones, el cual ejecuta un set de instrucciones de un microprocesador MIPS. Como el ISA de dichos procesadores no posee instrucciones de entrada/salida, esto no es posible de realizar con el simulador y procesador, pero si es posible de implementar en otros procesadores.

Posiciones de Memoria Compartida

Mediante direcciones de memoria compartida entre los distintos procesadores es posible sincronizar los procesos, mediante el empleo de semáforos. Este mecanismo al tener una velocidad de ejecución baja, por que se sincroniza mediante accesos a memoria RAM, disminuye notablemente el rendimiento.

Sistema de Colas con un Procesador de Eventos

Esta opción utiliza la primitiva de monitor mediante el algoritmo de RdP. Mediante la creación de un sistema de colas podemos ir "encolando" los disparos (accesos a una variable compartida para lectura o escritura). Para que luego el procesador vaya quitando de esta cola las peticiones (disparos) con el fin de ejecutarlos. En el caso de no poder llevar a cabo la ejecución el disparo continúa encolado. Esta opción es inviable ya que no es posible implementarla en el simulador, debido a la complejidad de la misma y a como se encuentra construido el simulador en sí.

Posiciones de Memoria Caché Compartida

Consiste en cambiar el algoritmo de comportamiento a una zona de memoria caché, donde su nueva responsabilidad sea almacenar información, que permita sincronizar los procesos empleando RdP. Con esta propuesta se obtiene el mejor tiempo de acceso a los datos compartidos para la sincronización. También respeta el objetivo y las restricciones planteadas en esta tesis. Permite ser implementada en el Simulador SESC, utilizando módulos ya funcionando y con pocas modificaciones.

A partir del análisis de las distintas posibilidades, se optó por simular un sistema con memoria caché compartida, para lo cual se creó una nueva memoria caché. Implementando en esta caché, RdP para hacer la sincronización de los procesos. Esta elección se debe a que cumple con todas las condiciones planteadas. Y a su vez es la alternativa de mayor velocidad (acceso a memoria), obteniendo así un mejor rendimiento del sistema.

Desarrollo e Implementación

Relación entre Eventos y Transición

La Figura 35 ilustra como los eventos son acumulados en un contador. Lo eventos de entrada, que son generados por los procesos, incrementan el contador de eventos de entrada al ser insertados. Los disparos de la transición disminuyen el contador de eventos de entrada e incrementa el de eventos de salida. El contador de salida es disminuido en uno al ser leído por los procesos.

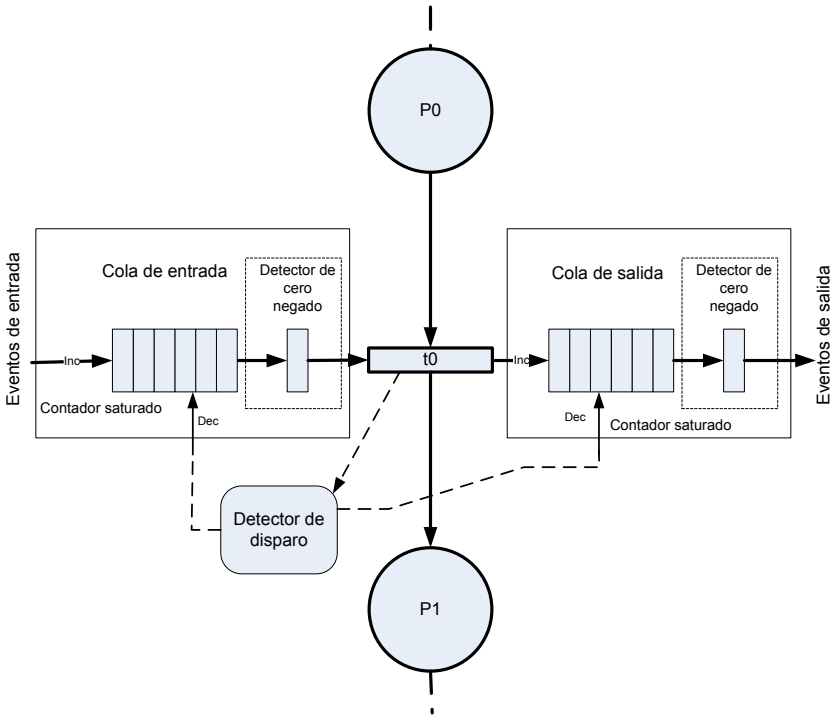


Figura 35: Contadores para colas de eventos

Los contadores de eventos aceptan en forma asíncrona y simultánea o no incrementos y decrementos. Son contadores saturados, en cero por decremento y en valor máximo por incremento.

El procesador ejecuta los disparos de las transiciones como uno-sensibilizado.

Elementos del Algoritmo

- $M_{j+1} = M_j + I\ x\ \sigma$, ecuación de estado para un RdP
- Dónde:
- I matriz de Incidencia
- σ el vector disparo, de las transiciones solicitada la sensibilizada con más prioridad, en el instante de la evaluación (es solo una transición).
- M_0 el vector de marcado actual
- $I^1, I^2, ..., I^n$ las columnas de la matriz I
- S^i el vector que se obtiene del cálculo $S^i = (M_0 + I^i)$, es un vector con el estado que se obtendría de disparar la i-esima transición.
- es el vector, de valores binarios, que indica cuales transiciones están sensibilizadas. Sus componentes E_i se obtiene de aplicar la función $sing(E^i)$ a cada vector E^i
- es cero si alguna de las componentes de S^ies negativo de otra forma es uno.
- Prioridad (E). Retorna un vector binario, de la misma dimensión que E, con un uno en la transición sensibilizada con disparo solicitado de más prioridad, el resto de las componentes son cero.

Algoritmo del PP Simplificado

Ejecución asincrónica y en paralelo

Para cada contador de eventos de entrada:

- Se aceptan todos los pedidos de disparo, por lo que se incrementa el contador de eventos de entrada asociado con la transición por cada evento.
- Se aceptan todos los disparos de la transición, por lo que se disminuye en uno el contador de eventos de entrada asociado a la transición.

Para cada contador de eventos de salida:

- Se aceptan todos los pedidos de lectura, por lo que se disminuye en uno el contador de eventos de salida asociado a la transición.
- Se aceptan todos los disparos de la transición, por lo que se incrementa el contador de eventos de salida asociado a la transición.
- Cada contador tiene como salida un uno si es distinto de cero y un cero si es cero

Cálculo

Se realiza el cálculo continuamente.

1. Se calcula $S^i = M_j + I^i$, se suman todas las componentes en paralelo para obtener el vector S
2. Se calcula $E_i = sing(S^i)$, se aplica sing a todas las componentes en paralelo para obtener el vector E
3. Se determina que transición se puede disparar, haciendo uso de la política de prioridad. Se calcula el valor de $\sigma = Prioridad\ (E)$
4. Se asigna el nuevo estado. Se genera una señal de decremento del contador de entrada asociada a la transición disparada y la señal de incremento del contador de salida asociada a la transición disparada

Los pasos 1, 2 y 3 son calculados en un ciclo, haciendo uso de una función lógica. Las tres acciones indicadas en 4 se realizan en paralelo. Tiene prioridad sobre uno y la asignación del nuevo estado está en exclusión mutua con el paso uno.

En este algoritmo se propone hacer distintos cálculos en paralelo, esto es beneficioso cuando la implementación se realiza en hardware. El paralelismo de ejecución del algoritmo le permite al PP responder en dos ciclos de ejecución si es posible realizar el disparo. Cuando no es posible espera a que la red sincronice para que el disparo sea posible y luego comunica el disparo en un ciclo.

Implementación y Ejecución del Algoritmo de Petri Simplificado

En la matriz I el marcado inicial y las cadenas de disparo se colocan en un archivo como texto plano. La matriz y el vector de marcado inicial se utilizan en el programa que ejecuta el PP, mientras que la cadena es la secuencia de disparo para simular la ejecución. Los disparos son reconocidos por el algoritmo de Petri (que se aloja en caché, actuando como "controlador de caché"). Se ejecuta el algoritmo del PP para determinar que disparados han sido resueltos, obteniendo así nueva marcas. Según el resultado de estas operaciones matemáticas, se determina si es posible acceder o no a los recursos compartidos o si se ha alcanzado la sincronización deseada. Si el disparo no es posible se acumula en la cola de disparos de entrada. Cuando el estado de la red lo admite se ejecuta y comunica a la cola de salida.

Implementación del Mecanismo en el Simulador

Para evaluar las opciones de implementación hay que considerar que el mecanismo de sincronización debe estar lo más cerca posible del punto donde se genera la solicitud y se deben cumplir las restricciones planteadas previamente (Restricciones).

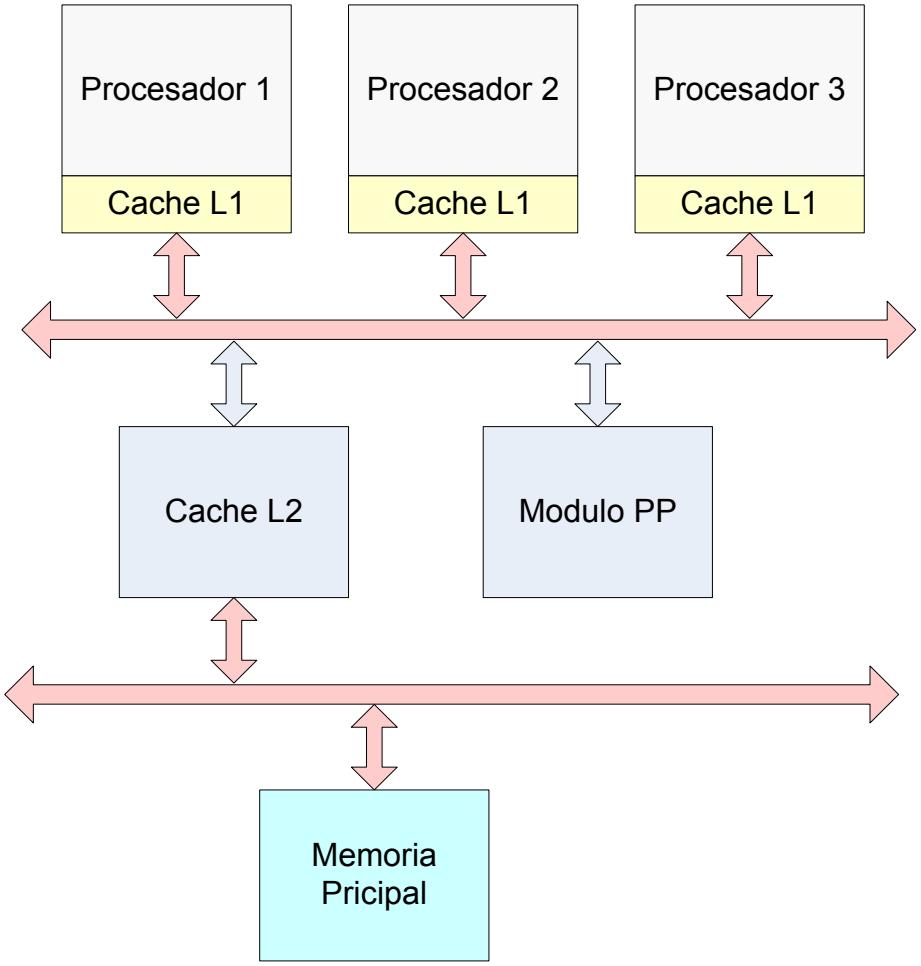


Figura 36: Arquitectura SMP, para colocar el módulo del PP

La Figura 36 muestra una arquitectura SMP, para colocar en ella el módulo del PP debemos considerar que éste debe ser accedido por todos los procesadores y mantener la coherencia. También se considera que el acceso será realizado por todos los procesadores, con una tasa de acceso que depende del algoritmo y el estado de los procesos.

Consideraremos los distintos lugares donde se puede instanciar el módulo:

- Si el módulo PP es instanciado en L1, este debe implementar un mecanismo para mantener la coherencia de todas las colas y el vector de estado, lo que degrada el desempeño del módulo PP. Además, se tiene que hacer una copia de la matriz I en cada L1.
- Si el módulo PP es instanciado en L2, no es necesario implementar un sistema de coherencia, puesto que solo hay una copia del vector de estado, pero la velocidad de acceso al módulo es menor que la del acceso a L1 y no hay degradación por la coherencia.

- Otra alternativa es instanciar el módulo PP con la memoria principal, pero la velocidad de acceso es baja lo que degrada el sistema.

En puntos anteriores se ha explicado el algoritmo para llevar a cabo la implementación, ahora bien se explicará con más detalle a nivel de implementación y ejecución en el simulador.

Solo a los fines de la simulación, porque el simulador lo facilita y los resultados son válidos se instancia el módulo PP como parte de la memoria caché y la política de esta caché es el algoritmo de Petri. Esto es equivalente a instanciar un módulo PP en el nivel L2.

Primero es necesario que la Caché sea capaz de implementar un mecanismo de inicialización el cual permita definir: el marcado inicial, la matriz de incidencia y los vectores de disparos posibles. *Segundo* una vez cargado el estado inicial, el sistema acepta disparo de transiciones. Si los disparos son resueltos, encola el éxito en la cola de salida. Si no son resueltos los encola en la cola de entrada hasta que el algoritmo de Petri los resuelva. Todo este mecanismo de Petri esta implementado dentro de una caché instanciada en un nivel 2.

El algoritmo de Petri se encarga de hacer de "controlador de caché". Esta caché almacena el estado de la red de todo el sistema de procesadores, por lo tanto debe ser compartida por todos los procesadores como una caché L2. Con la diferencia de que no requiere ser conectada a RAM, puesto que la inicialización se realiza antes de la ejecución del programa.

Aprovechando el modo jerárquico que posee el simulador para instanciar la memoria desde un archivo de configuración, creamos la "PetriCaché" que no se asocia a ningún procesador particular, sino que está compartida por todos, conectado directamente a todos los procesadores como lo muestra la Figura 36. El estar compartida por todos los procesadores es indicado a través del parámetro *shared* incluido en la variable *petriSource* dentro del ámbito de configuración del módulo PP que estamos instanciando. La memoria se la etiquetó como *PMemory*, luego en esta sección del archivo se lleva a cabo la configuración de parámetros como *tipo de caché, tamaño, algoritmo de Petri, nivel inferior de memoria*, etc.

En el nivel de memoria inferior no existe ningún bus que conecte a la PetriCaché con la RAM, configurándose este parámetro en la variable *lowerLevel* del módulo PP.

La clase GMemorySystem del simulador es la encargada de instanciar todos los niveles de memoria, es por esto que se declaró el objeto petriSource conectado directamente a todos los procesadores. El objeto es del tipo MemObj, de esta manera posee características similares a todos los módulos de memoria.

De la clase Caché hereda la nueva clase declarada como "PetriCaché", la cual implementa el algoritmo de Petri, que posee varios métodos, que son detallados a continuación:

- void petriCaché:: cargarConfig() , su responsabilidad es inicialización mediante la carga del archivo de configuración del Algoritmo de Petri. Éste contiene: Id, Cantidad de Filas, de Columnas, Matriz de Incidencia, Marcado Inicial de la red, Cantidad de Vectores Disparos, Vectores Disparo.
- clase MemRequest, posee los métodos que controla si la petición de memoria es de dato o instrucción correspondiente a Nivel 1 de caché, o si se está pidiendo un dato de la caché de Petri (dato manejado por el controlador de caché). Según si se pide uno u otro se ejecutan o no aquellos métodos del controlador de caché. En el caso de que se haga una

petición de un dato alojado en la caché de Petri, luego de que se efectúe un disparo (el programa que se ejecuta quiere escribir un dato en una dirección de memoria manejada por nuestro controlador de caché de Petri) se llama al método petriCaché::write(MemRequest *mreq), el cual toma un lock para ejecutar con exclusión mutua el método petriCaché::disparo() este llama al método que multiplica la matriz I por el vector Sigma, multiIxSigma(sigmaID), y con ese resultado se llama al método sumaVector(), el cual suma el vector M_0. En el caso de que se pueda efectuar el disparo se deja constancia de disparo efectivo o realizado, marcándose como Hit (permite verificar en el log que la cantidad de disparos realizados es coherente con la cantidad de disparos que ejecutó nuestro programa).

- Luego de cada disparo realizado se llama al método testdisparo (), el cual se encarga de testear los disparos en las colas de entrada. Verifica qué disparo es ahora posible, puesto que ha cambiado el marcado de la red. Esto se realiza hasta que al recorrer todas las colas de entrada no sea posible realizar un nuevo disparo. Al finalizar los disparos se procede a liberar el lock adquirido.

En la Figura 37 un flujo que detalla de forma sencilla el funcionamiento del algoritmo implementado dentro del simulador, resaltando como se realiza el llamando de los distintos métodos entre sí, los cuales se han descriptos anteriormente.

Para la simulación, la caché de Petri, fue instanciar a nivel de Simulación, por lo que se dificulta el manipular correctamente los hilos que llevaban a cabo la ejecución de las instrucciones del programa, ya que no se tienen las instrucciones en la API del simulador para hacer el correcto tratamiento. Esto es, cuando el disparo no se puede realizar, SESC no es capaz de frenar el hilo correspondiente a nivel de emulación, debido a la forma en que se encuentra creado el simulador, se computan de forma correcta los miss correspondientes a no poder efectuar el disparo, pero igual sigue corriendo el programa cuando debería quedarse "parado", hasta que otro hilo realice el disparo capaz de modificar el marcado de matriz y ahí poder efectuar el disparo del hilo "parado", continuando con la ejecución.

Entonces, la secuencia de disparos, por ejemplo para una exclusión mutua como la presentada en la Figura 39, que en teoría debiera respetar el orden que se muestra en la Figura 38, no se cumple, ya que los hilos de ejecución no se "paran" cuando no se puede ejecutar el disparo.

Las soluciones evaluadas son:

1. **Efectuar una espera activa** por parte de los hilos hasta que se cumplan las condiciones para efectuar el disparo.
2. **Emplear semáforos** (ya que estos si son provistos en la API del simulador) para forzar la detención de los hilos cuando el disparo no es posible, y que el semáforo sea liberado por aquellos hilos que efectúan correctamente los disparos.

La primera solución necesita soporte para hilos por parte de la API, lo cual no es posible. La segunda es la implementada, mediante el empleo de instrucciones especiales provistas por la API del simulador. Se ignora el tiempo empleado en ejecutar los semáforos, (no es tenido en cuenta para computar los tiempos de ejecución), ya que estos semáforos son una "medida para lograr el cometido de la simulación", sujeto a como está programado el simulador. Se quiere medir el tiempo de espera que es una sucesión de miss, hasta que el disparo sea posible. Esta solución está en completa concordancia con el objetivo de esta simulación.

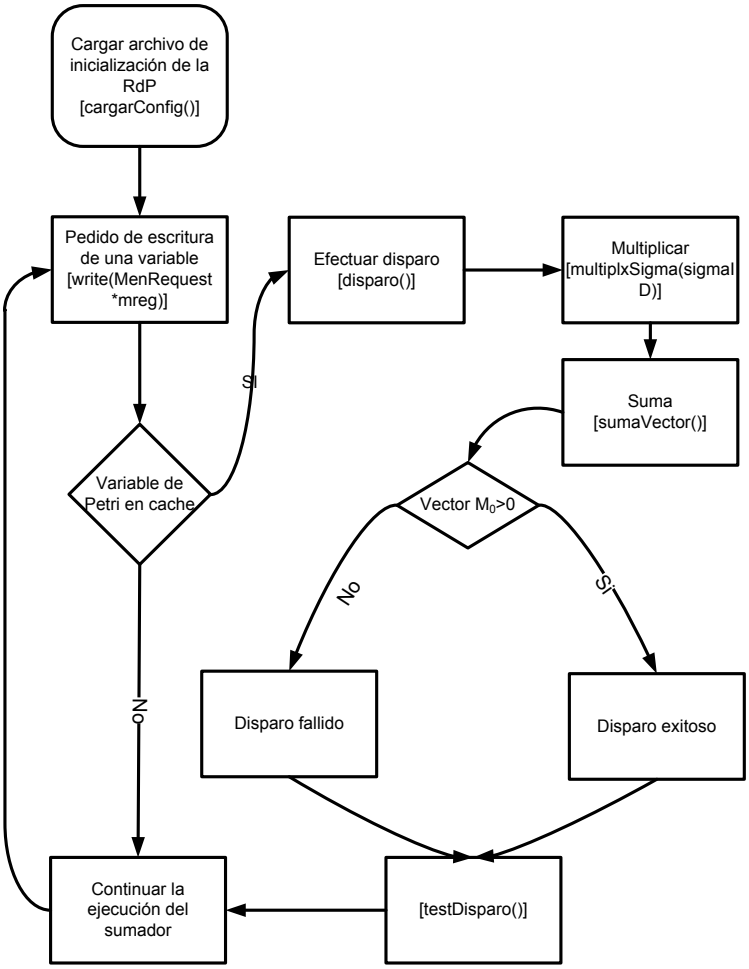


Figura 37: Diagrama de flujo del algoritmo del Petri-Caché

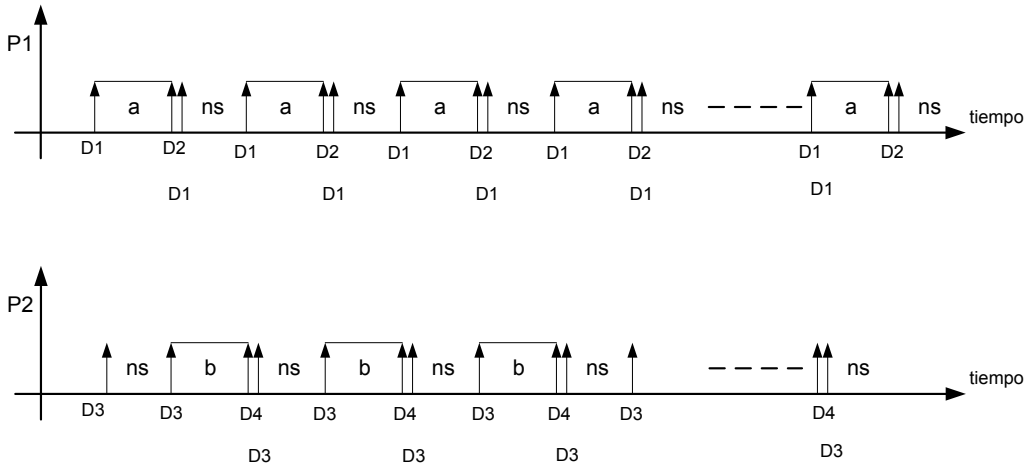


Figura 38: Ejecución de disparos a la RdP, por 2 procesos P1 y P2

Simulaciones y Mediciones

Se implementaron algoritmos que requieren de accesos simultáneos a recursos compartidos y fuertemente sincronizados, expresados mediante una RdP. Estos son: Escritores alternados, Productor / Consumidor, Algoritmo de Simulación de una Planta de Embalajes, Algoritmo de Control de un mecanismo de Velocidad Crucero con detección de automóviles.

Escritores Alternados

Estos dos procesos escriben alternativamente en el mismo buffer. El hilo principal, que corre en el procesador 0, instancia dos hilos, los que se ejecutan en los procesadores 1 y 2 respectivamente. Este par de hilos escriben en un array compartido, las cantidades n de bytes, que es variable. La Figura 39, representa el modelo de RdP de Escritores alternados.

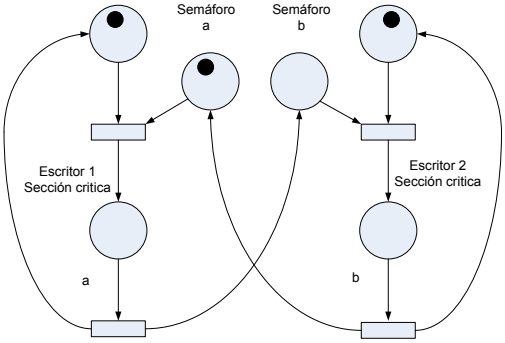


Figura 39: RdP de escrituras alternativas

Para ejecutar el algoritmo de Petri implementado en la caché se requieren la matriz de incidencia y el vector de estado inicial.

Tabla 89: Mediciones de tiempos de escritura en el simulador original y el modificado

Iteraciones	Escrituras en sección crítica (n)	Total de escrituras	Tiempo de ejecución sobre simulador original (ms)	Cantidad de semáforos/disparos ejecutados	Tiempo de ejecución sobre simulador modificado (ms)	Relación %
204800	1	204800	94,823	204800	70,2468	25,92
102400	2	204800	54,888	102400	42,6004	22,39
51200	4	204800	39,476	51200	32,4622	17,77
25600	8	204800	32,821	25600	29,3136	10,69
12800	16	204800	29,493	12800	27,7258	5,99
6400	32	204800	27,592	6400	26,7154	3,18
3200	64	204800	26,641	3200	26,2032	1,64
1600	128	204800	26,225	1600	26,0066	0,83
800	256	204800	26,017	800	25,8008	0,83
400	512	204800	25,913	400	25,8584	0,21
200	1024	204800	25,86	200	25,8322	0,11

En las Tabla 89 y Tabla 90 se presentan los valores medidos de: cantidad de escrituras, tiempo de ejecución del código, sobre el simulador original y sobre el simulador modificado, y la relación porcentual entre estas mediciones de tiempo. Además se indica la cantidad de escrituras realizadas en la sección crítica por cada iteración realizada. También se expresa el total de escrituras

realizadas por cada hilo sobre el array compartido y la cantidad de instrucciones de procesador ejecutadas por ambos simuladores (original y modificado).

En la Tabla 89 se puede apreciar como un algoritmo fuertemente sincronizado, tiene un porcentaje de mejora de 25,9% en los tiempos de ejecución y la Tabla 90 un 37,32% menos de instrucciones ejecutadas en el simulador modificado. En ambas tablas es evidente como al disminuir en uno el valor de los semáforos ejecutados (o sea, incrementar las escrituras realizadas por cada proceso entre sincronización realizada) disminuye la mejora del procesador modificado hasta niveles inferiores al 1%.

Tabla 90: Mediciones realizadas del algoritmo Escritores Alternados

Escrituras en sección crítica (n)	*Cantidad de instrucciones ejecutadas por core en simulador original*	*Cantidad de Instrucciones ejecutadas por core en simulador modificado*	*Mejora Porcentual*
1	*28262376*	*17715276*	**37,32%**
2	*15974426*	*10700916*	**33,01%**
4	*11110365*	*10009605*	**9,91%**
8	*8959924*	*8409504*	**6,14%**
16	*7910174*	*7634964*	**3,48%**
32	*7321114*	*7183494*	**1,88%**
64	*7026174*	*6957364*	**0,98%**
128	*6893884*	*6859474*	**0,50%**
256	*6826104*	*6808894*	**0,25%**
512	*6788944*	*6780334*	**0,13%**
1024	*6763227*	*6758917*	***0,06%***

La Tabla 90 muestra una relación porcentual de la cantidad de instrucciones ejecutadas, comparando el simulador modificado respeto del original. Se aprecia la disminución en las instrucciones ejecutadas que posee el simulador modificado respecto al original. Dicha mejora se encuentra más pronunciada para una cantidad de escrituras por sincronización igual a 1y 2, esto es debido al alto impacto producido por la ejecución de una primitiva en cada ciclo o par de ciclos de escritura.

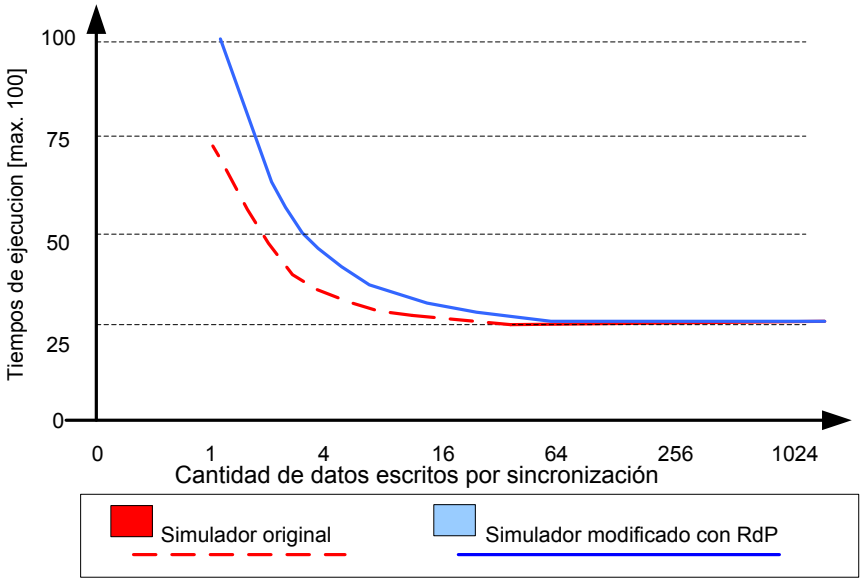


Figura 40: Variaciones de tiempo de sincronización con semáforo y PP

En la Figura 40 se aprecia como para cantidades de escrituras por sincronización superiores a 64, toma valores muy próximos a cero. Esto se debe a que los tiempos o instrucciones empleados para ejecutar los semáforos no representan más de un 1% del tiempo o instrucciones totales, como se había mencionado anteriormente.

Productor / Consumidor

Este es un problema clásico de acceso a recursos compartidos que se arbitra mediante un mecanismo de concurrencia que implemente la exclusión mutua, como el buffer circular de la Figura 41. El proceso productor genera información que es colocada en un buffer y es extraída de éste por un proceso consumidor.

Cuando el buffer circular se encuentra lleno, el productor no puede colocar el dato, hasta que el consumidor retire elementos. Si el buffer se encuentra vacío, el consumidor no podrá retirar elementos hasta que el productor deposite alguno en él.

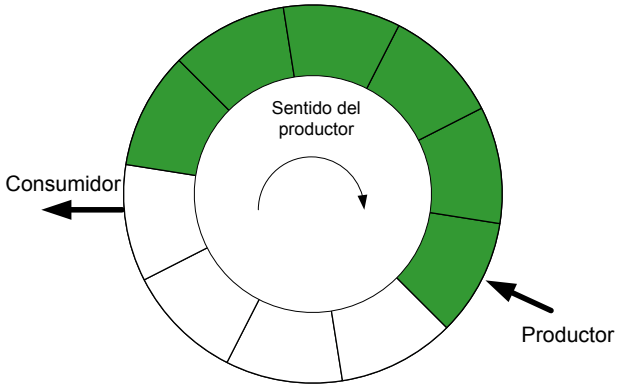


Figura 41: Buffer circular

El hilo principal corre sobre el procesador 0. Instancia seis hilos, que se ejecutaban sobre los procesadores 1, 2, 3, 4, 5 y 6. Tres de estos hilos producen datos y los otros tres hilos consumen datos del buffer compartido, cuyo tamaño se va a ir modificando para la simulación de distintos escenarios.

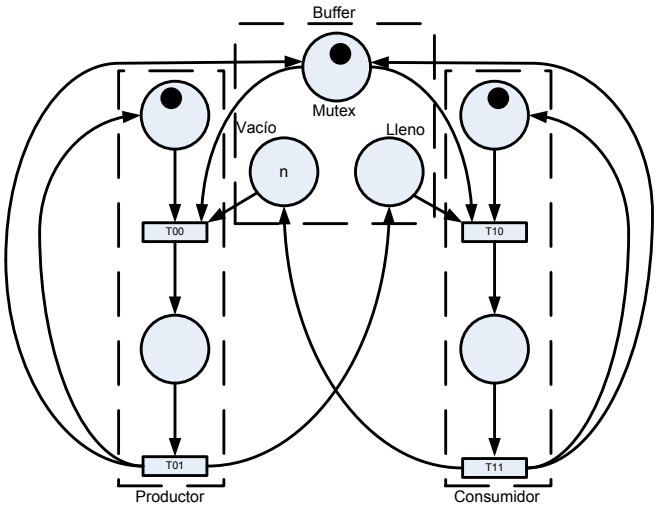


Figura 42: RdP de productor consumidor con buffer limitado

En la RdP de la Figura 42, las transiciones T00, T01, T10, T11 forman el modelo del productor/consumidor. El buffer esta inicializado con n elementos, la plaza denominada vacía indica los lugares vacíos disponibles al proceso productor y la plaza llenos le indica al consumidor la cantidad de elementos que le restan por consumir.

Las Tabla 91 y Tabla 92 muestran los valores obtenidos en la simulación, contienen: tiempo de ejecución del código sobre el simulador original y sobre el modificado, la relación de mejora, la cantidad de producciones/consumiciones realizados, la cantidad de semáforos ejecutados y la

cantidad de instrucciones de procesador ejecutadas por ambos simuladores (original y modificado).

Tabla 91: Tiempos de simulación con el simulador original y el modificado

Cantidad de Prod/Cons	Tiempo de ejecución sobre simulador original(ms)	Cantidad de semáforos/disparos ejecutados	Tiempo de ejecución sobre simulador modificado(ms)	Relación %
500000	198,003	1000000	95,946	51,54
200000	79,203	400000	38,379	51,54
100000	39,603	200000	19,191	51,54
50000	19,702	100000	9,644	51,05
40000	15,762	80000	7,715	51,05
30000	11,822	60000	5,758	51,29
20000	7,882	40000	3,839	51,29
15000	5,912	30000	2,88	51,29
10000	3,942	20000	1,921	51,27

Tabla 92: Interrupciones obtenidas en el simulador original y el modificado

Cantidad de Instrucciones ejecutadas por Core en Simulador Original	Cantidad de Instrucciones ejecutadas por Core en Simulador Modificado	Mejora Porcentual
69689258	52208343	25,08%
27875750	20883343	25,08%
13937914	10441682	25,08%
6918982	5171120	25,26%
5535212	4136942	25,26%
4121373	3073172	25,43%
2747603	2049011	25,43%
2060701	1536922	25,42%
1373816	1024850	25,40%

En la Figura 43 muestra las mediciones de los tiempos de ejecución, de cantidad de producciones y consumiciones realizadas por sincronización.

Podemos ver que los tiempos de sincronización disminuyen a medida que disminuyen la cantidad de sincronizaciones realizadas, en la Tabla 91 se ve esta disminución de 50 veces entre la máxima (1000000 de semáforos) y mínima sincronización (20000 semáforos). Este resultado era lógico de esperar, gracias al estudio realizado con anterioridad.

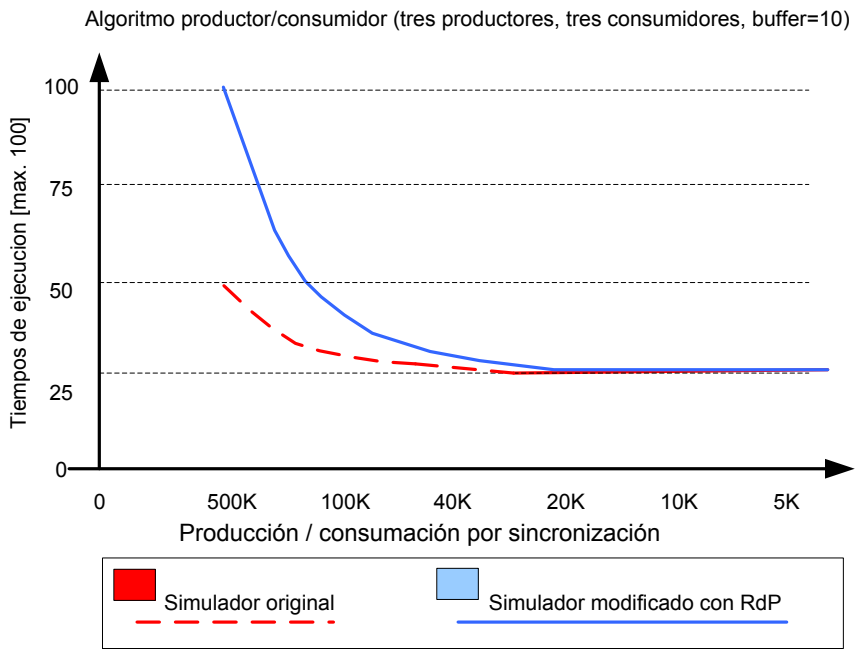


Figura 43: Tiempo de ejecución tiempo de sincronización

Es importante mencionar que existe una disminución de más del 50% de los tiempos empleados en ejecutar el código sobre el simulador modificado (ver Tabla 91) y un 25% menos de instrucciones ejecutadas (ver Tabla 92).

En la Figura 43 se muestran las gráficas comparativas para el simulador original versus simulador modificado, con los resultados arrojados en la simulación del algoritmo productor/consumidor. Ambas curvas tienen valores similares para una cantidad de producciones/consumiciones por sincronización igual a 15000, 10000. Debiéndose esto a que los tiempos empleados para esas cantidades tienden a asemejarse a los valores de tiempo obtenidos sin emplear sincronización, como consecuencia de que el tiempo empleado en sincronizar pasa a ser mínimo por que reduce en 50 veces la cantidad de sincronizaciones realizadas, respecto a la máxima sincronización (ver Tabla 92).

Algoritmo de Simulación de una Planta de Embalaje

Una cadena de fabricación que consiste de una cadena de montaje y está formada por distintas máquinas para procesar las piezas, almacenes y un robot que introduce y extrae piezas de las cintas transportadoras y que carga y descarga máquinas y almacenes. Dependiendo del producto que la fábrica este en ese momento produciendo, cada cadena de montaje sigue distintas secuencias y operaciones.

La Figura 44 muestra una cadena de montaje. El robot recibe una pieza de la cinta A, la carga en la máquina M1 que la procesa de algún modo, y la pieza se almacena tempo-remanente hasta que pueda cargarse en la máquina M2 para otra operación. Entonces se envía a otra cadena de montaje a través de la cinta B. Los lugares de la RdP asociada, en la Figura 45, se corresponden con las siguientes operaciones en p1.: La máquina M1 espera una pieza, p2. El robot carga la máquina M1 con una pieza recibida desde la cinta A, p3. La máquina M1 trabajando, p4. La máquina M1 preparada para descarga, p5. El robot descarga la máquina M1 y almacena la pieza

en el Almacen1 p6. La máquina M2 trabajando, p7. La máquina M2 preparada para descarga, p8. El robot descarga la máquina M2 y envía la pieza a la cinta B, p9. La máquina M2 espera una pieza, p10. El robot carga la máquina M2 con una pieza del Almacen1, A. Piezas preparadas para cogerse desde la cinta A, B. Piezas en la cinta B, D. Número de espacios, libres en el Almacen1 y E: Número de piezas en el Almacen1.

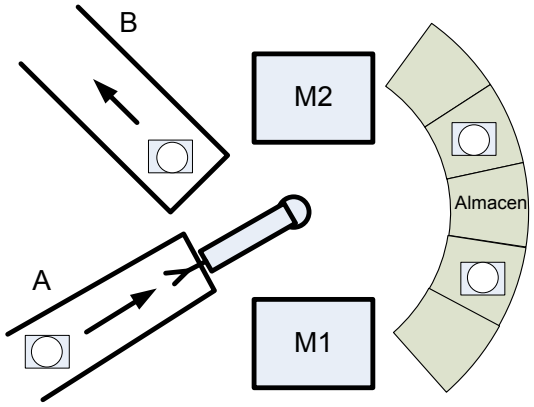


Figura 44: Cadena de montaje

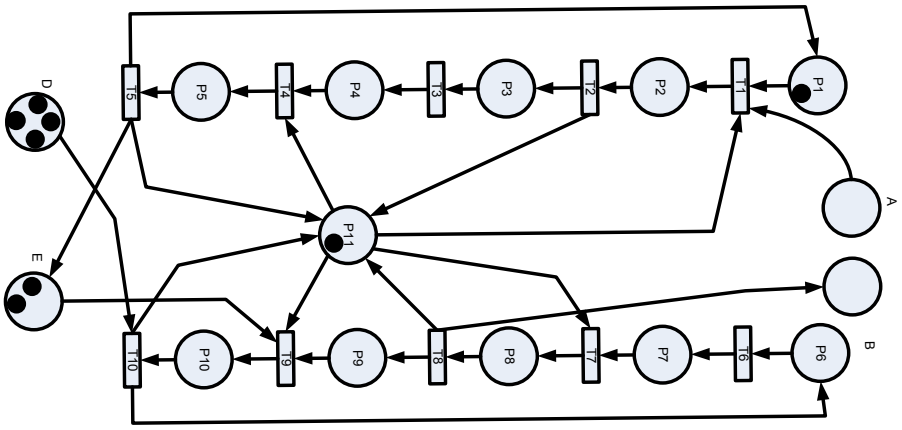


Figura 45: RdP de la planta de montaje

En esta red partimos de la condición donde las máquinas M1 y M2 están esperando a que se les cargue una pieza, y el robot está disponible.

La implementación de este algoritmo se llevó a cabo mediante el empleo de 4 hilos, de los cuales el hilo 0 se encargó de la inicialización de los hilos 1, 2, 3, y los demás hilos ejecutan distintos disparos para llevar a cabo la ejecución de la RdP (Petri-Caché).

En la Figura 44 se presenta la vista de la planta de embalaje y en la Figura 45 el modelo de la planta representado por una RdP.

Las Tabla 93 y Tabla 94 muestran los valores obtenidos en la simulación: tiempo de ejecución del código sobre el simulador original, sobre el simulador modificado y la relación porcentual entre estas mediciones de tiempo. También se indica el tamaño de loop, la cantidad de semáforos

ejecutados y el número de instrucciones de procesador ejecutadas por el simulador original y modificado.

Tabla 93: Tiempos de ejecución del código sobre el simulador original y el modificado

Cantidad de Embalajes	*Tiempo de ejecución sobre simulador original(ms)*	*Cantidad de semáforos/disparos ejecutados*	*Tiempo de ejecución sobre simulador modificado(ms)*	*Relación %*
1000000	*432,002*	*3000000*	*104,002*	***75,93***
100000	*43,202*	*300000*	*10,402*	***75,92***
10000	*4,302*	*30000*	*1,822*	***57,65***
1000	*0,467*	*3000*	*0,208*	***55,46***

Tabla 94: Relación porcentual entre estas mediciones de tiempo del simulador original y el modificado

Cantidad de Instrucciones ejecutadas por Core en Simulador Original	*Cantidad de Instrucciones ejecutadas por Core en Simulador Modificado*	*Mejora Porcentual*
164000251	*53000247*	**67,68%**
16400251	*5300247*	**67,68%**
1620198	*680177*	**58,02%**
171448	*75659*	***55,87%***

La Figura 46 muestra las mediciones de tiempos empleadas para la ejecución del código, con respecto a la cantidad de embalajes realizados por sincronización. Los tiempos de ejecución disminuyen notablemente a medida que disminuyen la cantidad de sincronizaciones realizadas, debido a una disminución en la cantidad de embalajes. Resultado concordante con los estudios anteriores. Se aprecia una disminución en los tiempos de ejecución del código sobre el simulador modificado (con Petri), siendo este tiempo superior al 55%, ver Tabla 93.

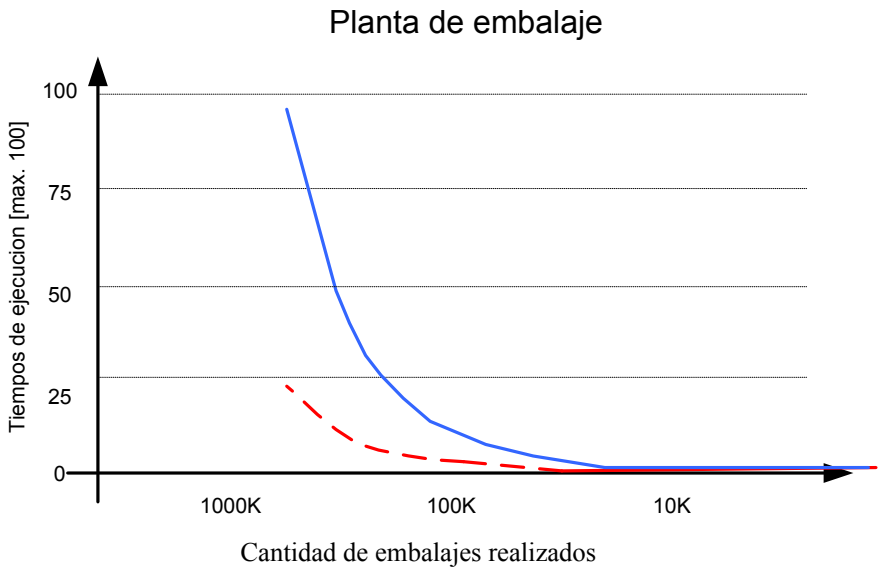


Figura 46: Medición de tiempos de la ejecución del código original y el modificado

Mientras que la cantidad de instrucciones también disminuye en igual medida, ver Tabla 94.

Caso de Algoritmo de Control

Mecanismo de Velocidad Crucero con Detector de Distancia a Objetos para Automóviles
Para este experimento se utilizó el ejemplo expuesto en el trabajo "Petri Nets in Software Engineering" [229] para poder, mediante simulación, medir la eficiencia del módulo sobre un problema real.

En este algoritmo, al igual que en el anterior, sólo se lleva a cabo la sincronización de los hilos, y no la implementación completa de todo el mecanismo de control, que es lo que se desea determinar.

Vamos a considerar un comando de velocidad crucero, con detector de objetos a distancia como se muestra en la Figura 47. Cuando el control de velocidad es encendido (ON), la velocidad del vehículo es fija.

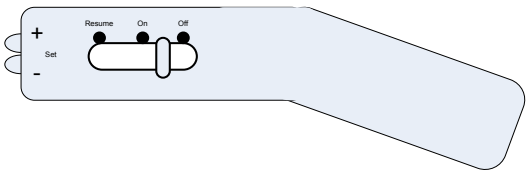


Figura 47: Comando de velocidad crucero

Empleando el botón SET (+ o -) el valor de la velocidad crucero, puede ser incrementado o decrementado por 2 km/h. Si el conductor presiona el freno (Brake) el control de velocidad es suspendido, pudiendo éste pasar a estado activo mediante RESUME y de nuevo a ON. En el estado de suspendido la velocidad actual es comparada con la velocidad almacenada y un zumbador es activado por un segundo si se supera el valor del control de velocidad. El mecanismo de velocidad crucero, puede ser desactivado moviendo el botón a OFF.

En conjunto con el control de crucero se encuentra el mecanismo de advertencia de objeto cercano. Al mismo tiempo que se almacena la velocidad, se activa el mecanismo que mide la distancia al vehículo que se encuentra delante de nosotros. Dicha distancia es comparada con la mínima distancia permitida. Si la distancia medida es menor que la permitida y el sistema de velocidad crucero se encuentra activo, este es suspendido, el vehículo desacelera y el conductor es informado de esto mediante una advertencia (por medio de un LED).

Las transiciones con las que cuenta nuestra RdP de la Figura 48 son:

- t1: Encendido,
- t2: Almacenar velocidad,
- t3 - t4: Incrementar velocidad,
- t5 - t6: Decrementar Velocidad,
- t7: Reactivar el mecanismo de Crucero,
- t8: Frenar,
- t9: Distancia Medida menor a la Mínima permitida / Desacelerar,
- t10: Distancia por debajo de la Mínima,
- t11: Distancia por encima de la Mínima,
- t12 - t13: Control de velocidad Excedido.

Los estados con los que cuenta nuestra red son:

- p0: Apagado (Off),
- p1: Listo (Ready),
- p2: Distancia Peligrosa Activa,
- p3: Control Crucero Activo,
- p4: Aumento de velocidad,
- p5: Disminución de velocidad,
- p6: Suspendido,
- p7: Distancia medida por debajo de la mínima,
- p8: Control de velocidad.

La implementación de este algoritmo en el simulador, se logró, empleando 5 hilos. El hilo 0 hizo la inicialización y la creación de los hilos 1, 2, 3 y 4, que se encargaron de ejecutar de forma concurrente todos los disparos de la red. Las mediciones se llevaron a cabo para distintas cantidades de iteraciones de una secuencia de trabajo del control de velocidad.

Las Tabla 95 y Tabla 96 contienen los valores obtenidos en la simulación del programa, tiempo de ejecución del código sobre el simulador original, sobre el simulador modificado y la relación porcentual entre estas mediciones de tiempo.

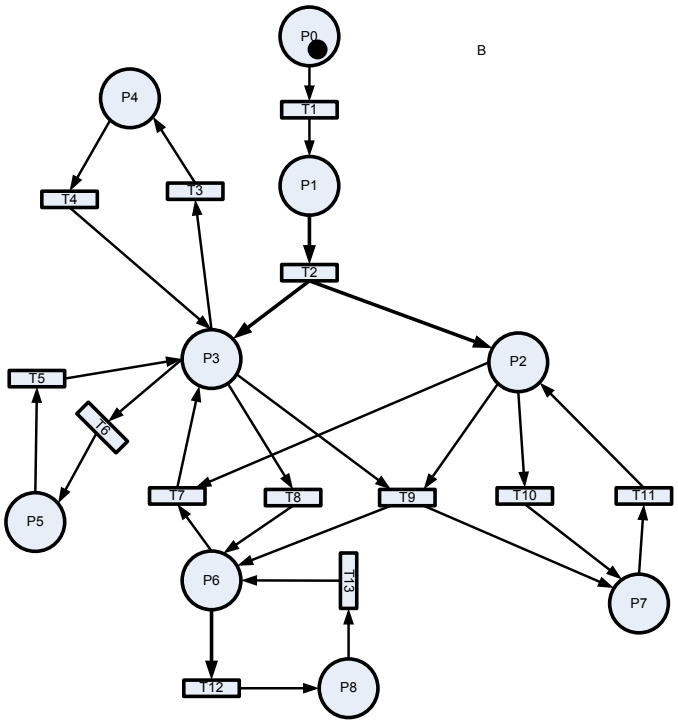


Figura 48: RdP del control de crucero

También se indica el número de iteraciones realizadas, la cantidad de semáforos ejecutados y el número de instrucciones de procesador ejecutadas por ambos simuladores (original y modificado).

Tabla 95: Tiempos medidos según cantidad de iteraciones, con el simulador original y el modificado

Iteraciones	*Tiempo de ejecución sobre simulador original(ms)*	*Cantidad de semáforos ejecutados*	*Tiempo de ejecución sobre simulador modificado(ms)*	*Relación %*
100000	*145,472*	*700000*	*60,354*	***58,51***
10000	*14,592*	*70000*	*6,275*	***57***
1000	*1,458*	*7000*	*0,629*	***56,86***
100	*0,147*	*700*	*0,0642*	***56,32***
10	*0,016*	*70*	*0,00792*	***50,5***

Tabla 96: Tiempos para interrupciones según cantidad de iteraciones del simulador original y el modificado

Cantidad de Instrucciones ejecutadas por core en simulador original	*Cantidad de instrucciones ejecutadas por core en simulador modificado*	*Mejora porcentual*
46913745	*18217625*	**61,17%**
4688523	*1867513*	**60,17%**
468213	*186813*	**60,10%**
46953	*18833*	**59,89%**
4783	*2013*	***57,91%***

En la Figura 49 se encuentran las mediciones de los tiempos empleados para llevar a cabo la ejecución del código. Claramente se observa que estos tiempos disminuyen a medida que es menor la cantidad de ejecuciones (iteraciones), debido a que es menor la cantidad de

sincronizaciones realizadas. Obteniéndose con el simulador modificado (empleando RdP) una disminución superior al 50% en los tiempos de ejecución (ver Tabla 95).

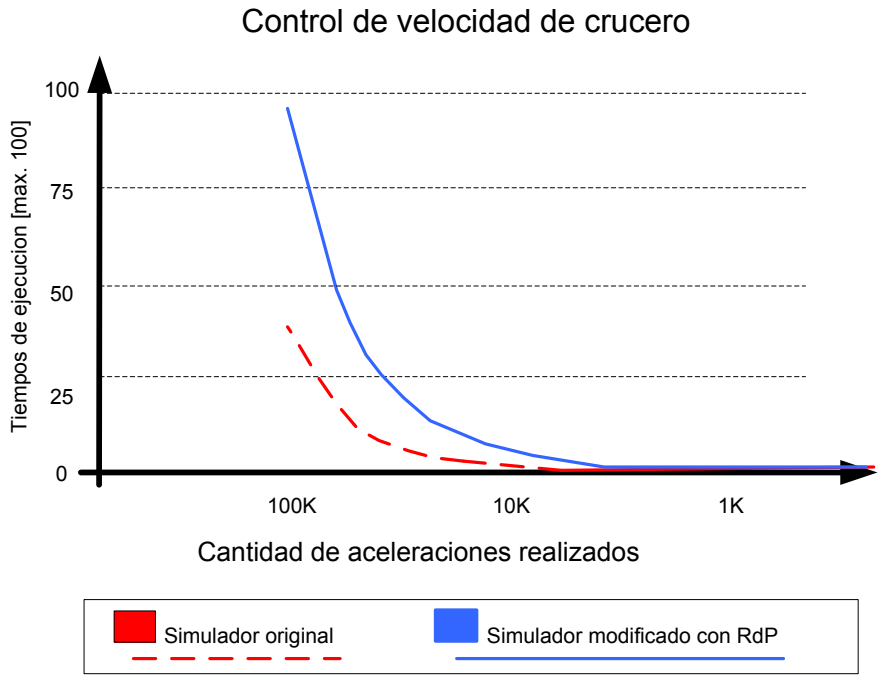


Figura 49: Mediciones de tiempos de ejecución del código, del simulador original y el modificado

Resultados y Conclusiones

En este trabajo, se determinó el costo (tiempo) de procesamiento de la sincronización de procesos. Luego se modificó el simulador SESC con el fin de incluir el algoritmo de Petri (Petri-Caché). Se realizaron corridas que permiten determinar con el simulador: Tiempo de ejecución, Ciclos de reloj empleados, Cantidad de instrucciones ejecutadas por core, Tipo de instrucción ejecutadas, Unidades funcionales utilizadas, Cantidad de Hits y Miss por caché. De esta forma se obtuvo información para determinar el problema de eficiencia de la sincronización entre procesos y proponer una solución. La propuesta es la modificación de la arquitectura SMP agregando un procesador de Petri para la sincronización de procesos en un sistema multi-Core, desarrollando así un concepto de procesador heterogéneo.

Los experimentos realizados con la simulación, donde se incluye esta arquitectura heterogénea, arrojaron disminución por encima del 25% (alcanzando valores del 75%) en los tiempos de ejecución de procesos fuertemente sincronizados, reduciendo también el número de instrucciones ejecutadas en más de un 20% respecto a las arquitecturas SMP. Estas mejoras han sido obtenidas en programas que emplean fuerte sincronización entre procesos. Además debemos remarcar que esta arquitectura es más rápida, ya que tiene mejor respuesta en tiempo real a la hora de sincronizar sin introducir ningún cambio en el ISA.

Por otro lado, es importante mencionar que la realización de los programas para que puedan ser ejecutados en el nuevo procesador son de simples programación, ya que una vez realizado el grafo de la RdP se pueden obtener las matrices que programan directamente al PP y quedan totalmente desacopladas las partes secuenciales de los programas del flujo de ejecución.

Hay que destacar que la carga al PP impacta en los recursos de memoria y procesador pero no sobre el algoritmo de Petri (Petri-Cache) puesto que este sólo atiende a los requerimientos de sincronización y exclusión mutua, es decir que la sobrecarga sobre el módulo está dada por la cantidad de disparos por unidad de tiempo.

Capítulo 4

Sistema Multi-Core Sincronizado por un PP Implementado en una FPGA

Resumen

En este trabajo se desarrolla un módulo IP-Core cuya responsabilidad es ejecutar el algoritmo de Petri para mejorar la sincronización entre procesos que se ejecutan en una arquitectura Multi-Core. En una primera etapa se lleva a cabo el estudio de factibilidad del desarrollo del mismo. Luego se integra el IP-Core en una FPGA, formando parte de la arquitectura (Microblaze), constituyendo un sistema Multi-Core asimétrico. Finalmente se realizan las pruebas para validar la eficiencia de la implementación.

Objetivos

Diseñar e implementar un IP-Core que ejecute el algoritmo de Petri para conformar el PP e integrarlo a un procesador, para obtener una arquitectura Multi-Core heterogénea sobre una FPGA.

Objetivos Secundarios

Permitir al PP la programación de distintas RdP y del vector de estado en tiempo de ejecución.

Instanciar en el Microblaze los programas de prueba al PP desde la memoria principal.

Comunicar el PP con los procesos, a través de eventos.

Implementar en el PP el tratamiento de las siguientes acciones para la comunicación: que el Microblaze envíe y reciba eventos al PP (IP-Core); que el PP procese los eventos, realizando el cálculo del nuevo vector de estado según los eventos recibidos, o encole el evento hasta que sea leído y espere por nuevos eventos.

Restricciones

El PP procesa los eventos y responde a las lecturas de eventos en el orden de dos ciclos reloj.

Materiales y Metodología del Trabajo

Para este trabajo se utilizó una metodóloga experimental, siguiendo un modelo de proceso que combina desarrollo evolutivo y basado en componentes [230, 231].

Se realizaron simulaciones para verificar los módulos y luego se implementaron en una FPGA para validarlos.

En este proyecto se utilizaron las siguientes herramientas: ISE de Xilinx (para la simulación del módulo a lo largo del desarrollo), CoreGenerator de Xilinx (para la creación del código de la memoria externa), XPS de Xilinx (para la creación y configuración del MicroBlaze, así como para la conexión del PP, mediante el bus PLB).

Sistemas Operativos: XilKernel (se instaló en MicroBlaze para la interacción del microprocesador con el exterior) y Windows 7 Ultimate (se utilizó para la configuración del Software XPS).

Resultados Esperados

Mejoras Alcanzadas con este Enfoque

Lograr una mejor performance en el tiempo de sincronización de las arquitecturas Multi-Core.

Obtener un PP independiente de la arquitectura ISA del procesador, con programación directa (vector y matriz) y con posibilidad de conexión a distintos tipos de buses para facilitar la comunicación e integración de software y hardware de arquitecturas heterogéneas.

RdP Extendidas

Una RdP extendida contiene arcos especiales[1, 35, 232], como: arcos inhibidores. Un arco inhibidor es un arco dirigido que deja un lugar Pi para alcanzar una transición Tj. Su fin está marcado por un pequeño círculo, como se muestra en la Figura 50. En esta figura, el arco inhibidor entre P2 y T1 significa que la transición T1 sólo se activa si el lugar P2 no contiene token. El disparo desencadena la eliminación de un token (según el peso) de cada lugar de entrada a T1, con la excepción de P2, y en la adición de una marca (según el peso) a cada lugar de salida de T1.

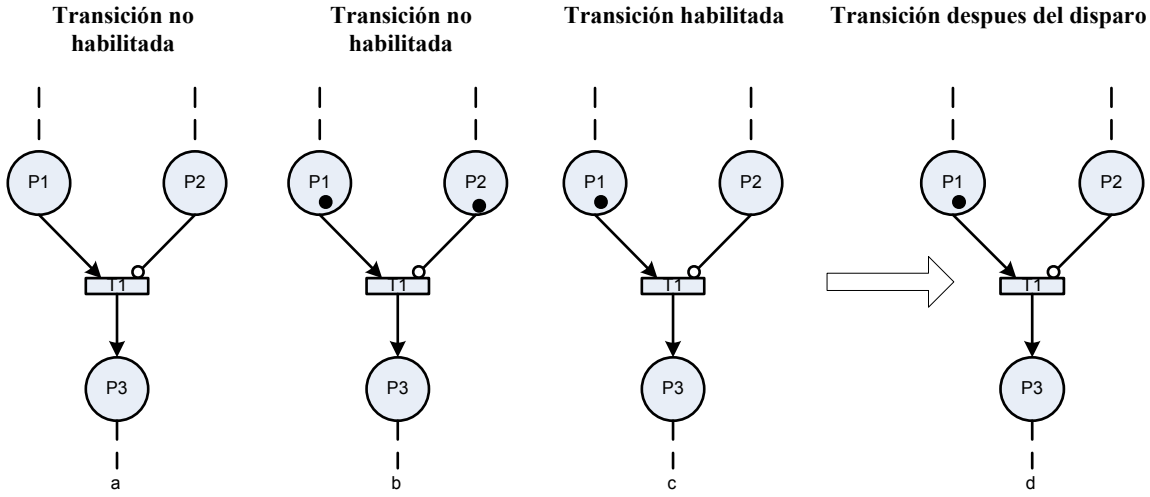


Figura 50: Transición con arco hinividor

En la Figura 50 a, la transición T1 no está activado porque P1 no contiene ningún token. En la Figura 50 b, T1 no está habilitada porque P2 contiene un token. En la Figura 50 c, la transición está habilitada y la marca obtenida después del disparo se muestra en la Figura 50 d.

RdP con prioridad[35]
Este tipo RdP se utiliza cuando queremos elegir entre distintas transiciones habilitadas que han recibido el evento asociado y están en conflicto. Para esto se hace uso de una relación de orden parcial de las transiciones de la red.

En la Figura 51 hay dos estructuras en conflicto, las que son: $\langle P_1, \{T_1, T_2\}\rangle$ y $\langle P_2, \{T_3, T_2\}\rangle$. Si asignamos un orden estricto para cada conflicto, como por ejemplo: $T_2 < T_1$ y $T_4 < T_2 < T_3$ para el la Figura 51, el conflicto es resulto.
Estas transiciones están en conflicto para la marca para la marca $m_1 = (1,1,0,..)$ lo que se expresa: $\langle P_1, \{T_1, T_2\},\ m_1\rangle$ y $\langle P_2, \{T_3, T_2\}\ m_1\rangle$.

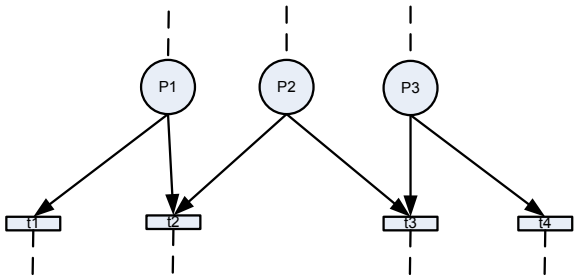


Figura 51: Transiciones en conflicto

Es de notar que las prioridades no pueden ser expresadas solo con una RdP ordinaria; mientras que, el orden de prioridad, si puede ser implementada con brazos inhibidores (u otros brazos).

Poder de Expresión de las RdP

El poder expresivo de las RdP ordinarias es menor [232] que el de las máquinas de Turing.

Esto significa que hay algoritmos que no pueden ser descritos por cualquier RdP clásica.

"No todas las funciones que son computables por una máquina de Turing se puede calcular por una RdP". Para más información sobre Turing-computabilidad consultar [233].

Con el agregado del tiempo a las RdP clásicas, es posible demostrar que tienen la misma potencia de cálculo [232] que las máquinas de Turing. Esto se realiza verificando que cada función-número teórico, que son computable por una máquina de Turing, también son computables por una RdP con tiempo.

Informalmente, una máquina de contador es una máquina MultiStack restringida que puede almacenar un número finito de números naturales, y puede sumar o restar (si el contador no está a cero) a uno o a ninguno de estos contadores. Para más información sobre las máquinas MultiStack, ver [233].

Las RdP extendidas y con prioridad tienen el poder de cómputo de las máquinas de Turing. De ello se desprende que todas las RdP con prioridad se pueden modelar mediante RdP extendidas.

En cuanto a las propiedades de las RdP extendidas, con prioridad, no autónomos y continuas son todas extensiones de las RdP ordinarias. Algunas, aunque no todas las propiedades, de las RdP ordinarias son aplicables a las RdP extendidas.

Relación entre las Funciones del Monitor y las RdP Extendidas y No Autónomas (RdPnA)

El objetivo es guiar la ejecución de un programa según la disponibilidad de los recursos, actuadores y/o sensores, lo que es comunicado a la RdPenA. La sincronizando de la RdP no autónoma (RdPnA), se realiza con el sistema físico, haciendo uso de los eventos que éste genera y los eventos que la RdPnA computa y genera.

Dado que la capacidad de expresión de una RdPnA es la de una máquina de Turing y es posible obtener una RdPnA que modele las restricciones del sistema. Donde el sistema está conformado por el modelo y los eventos que generan y/o reciben los actuadores, recursos y/o sensores; es

posible controlar la ejecución del sistema en forma autónoma. Lo que conforma un mecanismo de abstracción de datos.

Donde la lógica de la secuencia de cada proceso, el estado local de cada parte del sistema y el estado global del sistema, son representados por la RdPnA extendida.

Esta RdPnA implementa la lógica del sistema y los eventos lógicos, estos últimos se relacionan con los eventos físicos.

La responsabilidad de los componentes es transformar los eventos lógicos de la RdPenA en acciones y eventos físicos, y los eventos físicos del sistema físico en eventos lógicos y comunicarlos a la RdPnA.

Los componentes ejecutan códigos secuenciales que controlan los recursos, actuadores y/o sensores (sistema físico). Estos códigos son agrupados en componentes con capacidad de ejecución.

La ejecución secuencial de los componentes genera eventos, que son comunicadas a la RdPnA. Esta red, computa estos eventos y los comunica, por medio de eventos lógicos que son el resultado del cómputo, a los componentes con el fin de sincronizar y coordinar las acciones de todos los componentes.

La única forma de cambiar el estado de la RdP es a través de los eventos de entrada y el tiempo, por lo que el cálculo de los nuevos estados es realizado con datos y procedimientos aislados, que solo son accedidos desde el PP y no son visibles desde fuera del procesador (sólo es posible la lectura).

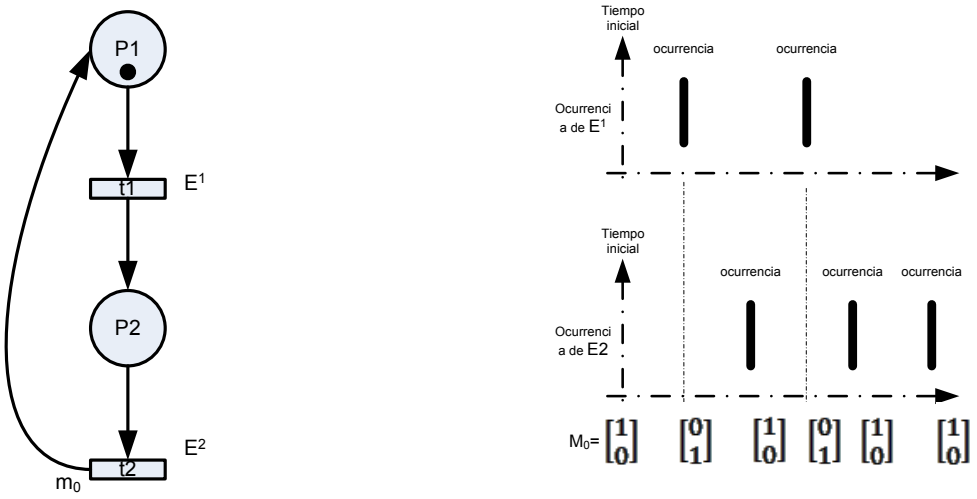
En la interacción entre los módulos y el PP podemos distinguir los siguientes casos:

- La solicitud para iniciar una acción es trasmitida desde el componente a la RdPnA por un evento, cuando el estado de la RdP permite la ejecución le comunica al componente con un evento (que es el resultado del cómputo de eventos)
- Cuando el componente termina una acción, le comunica a la RdPnA para que ésta refleje la terminación como un nuevo estado.
- Si la RdP ha cambiado de estado, por tiempo u algún evento, le comunica al componente por un evento, el nuevo estado.

Aquí destacamos, que al igual que en un monitor los componentes que esperan por una sincronización lo hacen en una cola, por lo que es necesario almacenar los eventos en las colas relacionadas con las transiciones y con las mismas prioridades que las transiciones, hasta que la sincronización sea posible y luego comunicar la sincronización al componente.

Por lo que, cada transición tiene que ser capaz de almacenar los eventos lógicos, o no, según sea requerido.

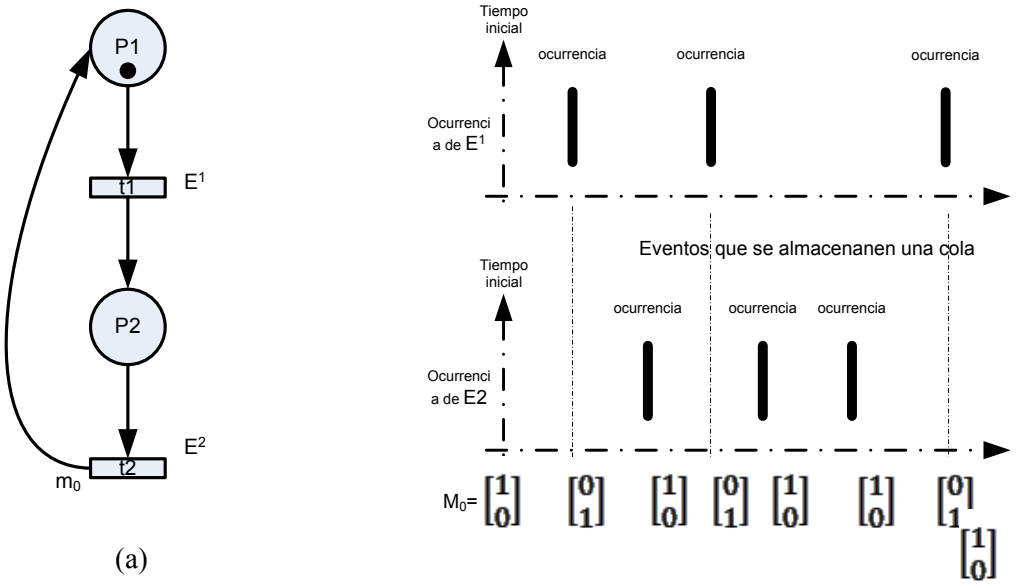
Si la transición esta sensibilizada, con el evento que se le ha asignado realizar el disparo, o hacerlo esperar en una cola de evento hasta que la transición este sensibilizada para realizar el disparo.



(a) (b)

Figura 52: Eventos en RdPnA con transiciones sincronizadas sin cola de almacenamiento.

La última ocurrencia de E^2 es descartada puesto que la red no está sincronizada



(a)

(b)

Figura 53: Eventos en RdPnA con transiciones sincronizadas con cola de almacenamiento.

Luego de que el evento E^2, que es guardado en la cola de T2, ocurre E^1 se dispara T2 por el evento almacenado en su cola.

La Figura 52 (a), muestra una RdPnA sin cola de eventos. El arribo de los eventos E^1 y E^2 se muestran en la Figura 52 (b), En las tres primeras ocurrencias las transiciones están sensibilizadas por lo que son disparadas inmediatamente. El ultimo evento que arriba es E^2 y la red no está sincronizada por lo que se pierde. En la parte inferior de la Figura 52(b), se muestran los cambios de estado.

La Figura 53 (a), muestra una RdPnA con cola de eventos. El arribo de los eventos E^1 y E^2 se muestran en la Figura 53 (b). En las tres primeras ocurrencias las transiciones están sensibilizadas por lo que son disparadas inmediatamente. El último evento que arriba es E^2 y la red no está sincronizada por lo que se almacena en la cola, y cuando la transición es sensible se dispara. En la parte inferior de la Figura 52(b), se muestran los cambios de estado.

La Figura 54, muestra la interacción entre los componentes compuestos por software, sensores, actuadores, recursos físicos, recursos lógicos y el PP. El PP es inicializado con las matrices, vectores de la RdPenA y con el marcado inicial. Los procesos son ejecutados en las CPU del SMP.

Requerimientos del PP

Funcionalidades del PP:

- Compartir el bus para comunicar el PP con una arquitectura SMP.
- Comunicación configurable, compuesta por memoria y registros configurables (de 8,16 o 32 bits.).
- Implementar en el módulo el algoritmo de Petri, programable en tiempo de ejecución. La RdP programada en el controlador, debe ser del tipo ordinaria y prever la implementación de brazos inhibidores.
- Asociar a cada transición con una cola de eventos de entrada programable.
- Asociar a cada transición con una cola de eventos de salida programable.
- Implementar colas de entrada/salida de eventos.
- Decidir el disparo en 3 o menos ciclos de reloj.
- Acceder al módulo de Petri por medio de direcciones específicas.
- Realizar en una FPGA la integración y pruebas.

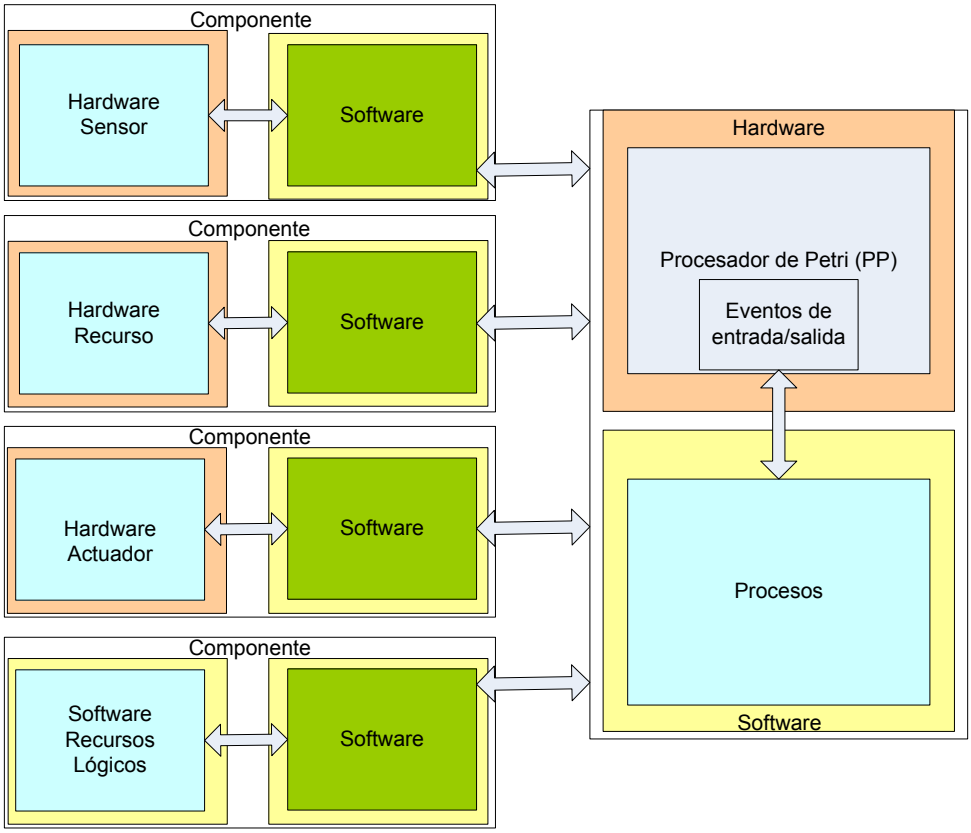


Figura 54: PP interconectado con los componentes

La implementación de los brazos inhibidores y cotas de plaza será prevista en esta iteración para ser agregada en la próxima etapa.

Implementación del Sistema Embebido

Como primera aproximación para la implementación e integración del PP en un SMP, se ha realizado la implementación en una la plataforma de hardware embebido que hace uso de una FPGA Spartan 6 [234]. Con el fin de construir un sistema SMP, se ha instanciado un sistema embebido, compuesto por un procesador MicroBlaze y el PP.

La implementación se realizó haciendo uso de las herramientas Xilinx Platform Studio (XPS) del SDK 13.3, se utilizó el asistente BSB para crear un modelo básico del sistema embebido de forma directa, y luego modificarlo de acuerdo a los requerimientos.

Las características del diseño son las de la familia del dispositivo Spartan 6: Dispositivo: xc6slx16: paquete: csg324, revisión de la placa: B, tipo de Procesador: MicroBlaze Versión 8.20a, frecuencia del procesador: 83.33Mhz, tamaño de memoria local: 64kb, tipo de Interconexión: Processor Local Bus (PLB) y Periféricos: dlmb_cntlr: controlador de memoria de datos, ilmb_cntlr: controlador de memoria de instrucciones, RS232_Uart_1: periférico de comunicación mediante protocolo RS232, xps_timer_0: timer de interrupciones periódicas al procesador.

Diagrama en Bloque del PP

Descripción de los bloques funcionales

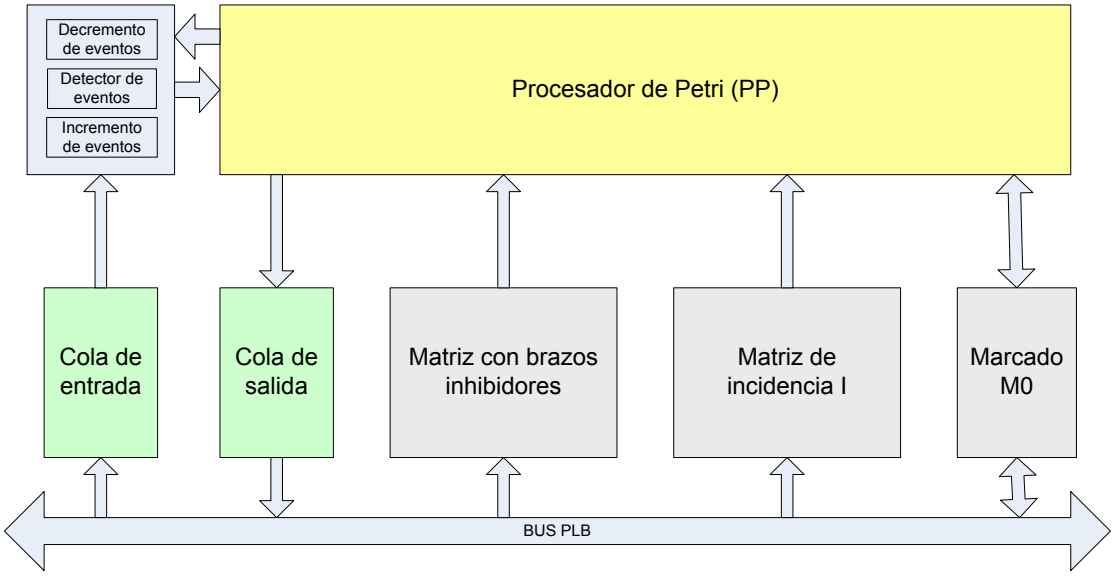


Figura 55: Interconexión del PP con el SMP

Consideraciones para la implementación del IP Core
Para acoplar el PP al sistema embebido, de forma que tenga comunicación con el procesador, se utilizó el asistente CIP del XPS. Con este asistente, se generó un IP-Core que se acopla al sistema mediante el bus PLB. Para esto se reconsidera el rol que desempeña el PP dentro del bus, la cantidad de registros visibles para el software de usuario que están mapeados en memoria, etc.

Una vez generada la plantilla del IP Core, se le ha introducido la lógica RTL descripta en lenguaje VHDL del PP.

Direcciones Mapeadas en Memoria

El IP Core que implementa el PP posee 5 registros de 32 bits visibles para el software de usuario:

- Registro de entrada, para encolar los eventos que disparan las transiciones en una cola FIFO (`XPAR_PETRI_0_BASEADDR`).
- Registro de salida, para leer los disparos que han sido procesados por el disparo de las transiciones del PP y se encuentran almacenados en una cola FIFO (`XPAR_PETRI_0_BASEADDR + 0x1`).
- Registro para escribir la Matriz de Incidencia (`XPAR_PETRI_0_BASEADDR + 0x2`).
- Registro para escribir y leer el Vector de Estado (`XPAR_PETRI_0_BASEADDR + 0x3`).
- Registro para escribir la Matriz Inhibidora (`XPAR_PETRI_0_BASEADDR + 0x4`).

Forma de Escribir las Matrices y el Vector de Estado
Como el bus del PLB es de 32 bits y los datos internos del PP son de 8 bits, los datos del PP se cargarán de a 4 por cada acceso al bus, en la carga de las matrices de incidencia e inhibidora. Es decir, por cada acceso a su posición de memoria desde el software de usuario, se le envían cuatro elementos de 8 bits para ser escritos en la matriz de incidencia. Por ejemplo tenemos la siguiente asignación:

*(`XPAR_PETRI_0_BASEADDR + 0x2`) = 0x01000001;

Esta asignación carga enteros con signo en la matriz, para el ejemplo son: 1, 0, 0 y 1.

Internamente en el procesador, las matrices de incidencia e inhibidora se almacenan como filas las transiciones y en las columnas las plazas. Como lo muestra la Tabla 97, en el ejemplo:

Tabla 97: Disposición de las matrices del PP

	P1	P2	P3	P4
T1	1	5	9	13
T2	2	6	10	14
T3	3	7	11	15
T4	4	8	12	16

La forma de cargar la matriz es secuencial y se accede a posiciones consecutivamente de forma secuencial de arriba hacia abajo, de izquierda a derecha; como lo indica la secuencia numérica de la tabla. Esto hay que tenerlo en cuenta en el software del usuario para inicializar el PP, el siguiente ejemplo muestra la carga de la matriz de incidencia:

Sea una matriz de 4x4, se carga con las siguientes asignaciones:

```
*(XPAR_PETRI_0_BASEADDR + 0x2) = 0x01000001;

*(XPAR_PETRI_0_BASEADDR + 0x2) = 0xFF00FF01;

*(XPAR_PETRI_0_BASEADDR + 0x2) = 0x02010000;

*(XPAR_PETRI_0_BASEADDR + 0x2) = 0x0100FF01;
```

La matriz en el PP será la de la Tabla 98, que se muestra a continuación.

Tabla 98: Ejemplo de carga de matriz en el PP

	P1	P2	P3	P4
T1	1	-1	2	1
T2	0	0	1	0
T3	0	-1	0	-1
T4	1	1	0	1

Los valores escritos en los registros del PP están en hexadecimal y se interpretan como complemento a 2, por esto el 0xFF se representa como -1 decimal.

El vector de estado de la RdP, se almacena con valores de 8 bits, pero con cada acceso a esta dirección de memoria, se transfieren 32 bits por el bus, pero solo se tienen en cuenta los 8 menos significativos, por lo que se cargan los valores de a uno. Estos valores se cargan de izquierda a derecha, siguiendo la secuencia numérica, de la forma que lo muestra la Tabla 99.

Tabla 99: Disposición del Vector de Estado del PP

P1	P2	P3	P4
1	2	3	4

Por ejemplo las asignaciones dan por resultado el vector de estado dentro del PP de la Tabla 100

```
*(XPAR_PETRI_0_BASEADDR + 0x3) =
```

```
0x00000001;

*(XPAR_PETRI_0_BASEADDR + 0x3) =
0x00000003;

*(XPAR_PETRI_0_BASEADDR + 0x3) =
0x00000000;

*(XPAR_PETRI_0_BASEADDR + 0x3) =
0x00000001;
```

Tabla 100: Ejemplo Vector de Estado del PP

P1	P2	P3	P4
1	3	0	1

El la Figura 56, muestra un diagrama de secuencias que representa el proceso de inicialización del PP, desde un proceso que se ejecuta en un procesador tradicional.

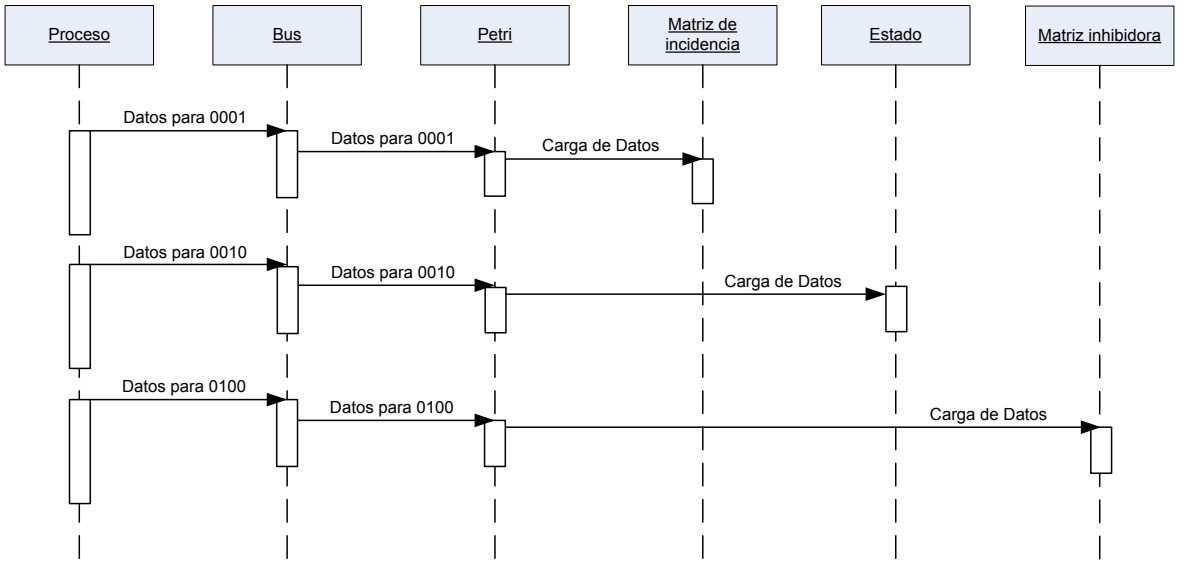


Figura 56: Inicialización del PP

Implementación en Hardware del PP

En esta sección se muestra una implantación en hardware de PP. La implementación ha sido realizada como un IP-Core, e integrada en un sistema SMP, compuesto por mutiles MicroBlaze. El PP ha sido descripto en Verilog, que es un lenguaje de descripción de hardware (HDL). En las secciones que siguen se presentara una arquitectura posible, para ser usada en sistemas embebidos. Finalmente se muestran las pruebas realizadas para evaluar el PP.

Arquitectura del PP

En la Figura 57 se muestra una arquitectura posible para la implementación del PP. Para la ejecución directa de una RdPnA.

En primer lugar, la comunicación con los procesos se realiza con eventos, haciendo uso de las colas. La programación se realiza con las matrices y vectores que son almacenados en el área de datos del PP. Para este caso se hace uso del bus que implementa el protocolo AXI.

En la Figura 57, se muestra el diagrama de la arquitectura de interconexión del PP. En esta Figura 57 se muestran el área de datos y la de cálculo.

La responsabilidad del área de datos del PP es almacenar los datos que se mantienen durante la ejecución del programa y son el programa del PP.

La responsabilidad del módulo área de cálculo del PP es calcular todos los vectores y matrices que requiere el PP para la ejecutar la red (el algoritmo de Petri). Es decir el nuevo estado, consume los eventos de entrada, transiciones sensibilizadas y genera los eventos de salida.

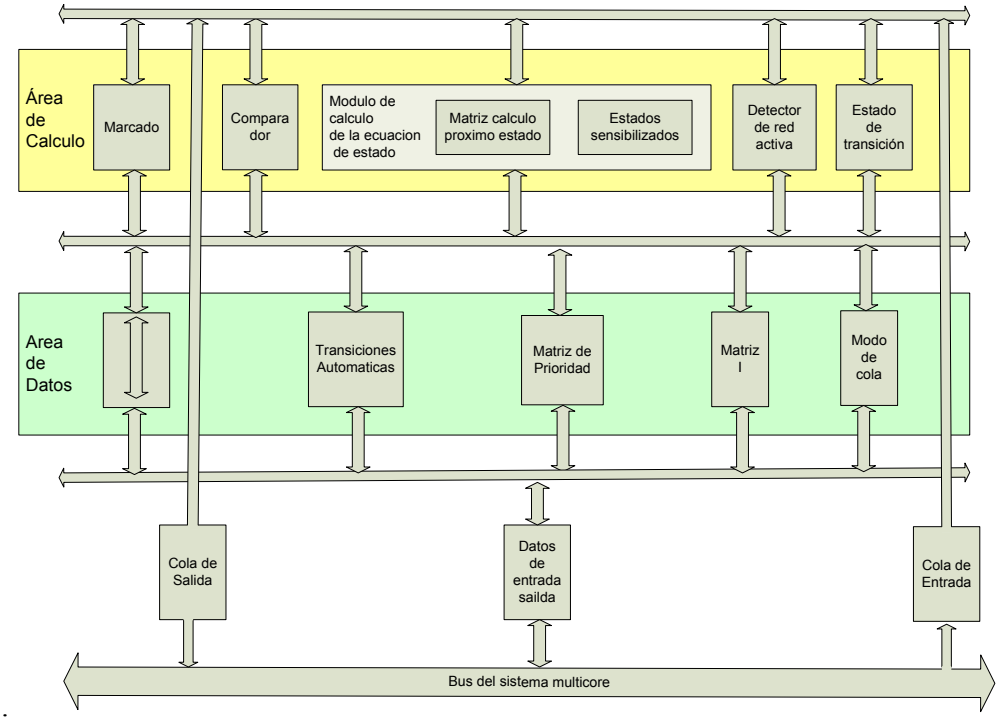


Figura 57: Arquitectura del PP sin brazos inhibidores y cota de plaza

Algoritmo para la Ejecución de RdPnA

Para ejecutar una RdPnA, se hace uso de la ecuación de estado con semántica de servidor sencillo (single server semántica). Para lo cual se detectan las transiciones sensibilizadas, los eventos solicitando disparos, los disparos que sean posibles de ser ejecutados y en caso de serlo, resuelve la ecuación de estado de la red para lograr un nuevo marcado de la misma.

La ecuación de estado de RdP, tiene la siguiente forma:

$$m_{i+1} = m_i + I \cdot (d\ )$$

Donde d es un vector, con uno en las componentes que tienen disparo solicitado. En la Figura 58, se muestra el vector d , que es binario de dimensión $|T|$, indicado como "Vector con el disparo de mayor prioridad".

Este vector es el resultado de realizar la suma entre cada columna (transiciones) de la "Matriz de Incidencia I" (vector con valores enteros con signo de dimensión |P|) y el "Vector de marcado"

(vector con valores enteros con signo de dimensión |P|); la cantidad de operaciones vectoriales que se realizan simultáneamente son |T|. De esta operación resultan "Matriz con Posibles próximos Estados". A estos posibles estados se les aplica la función "Detector de signos" y se obtiene el "Vector con transiciones sensibilizadas", es decir las transiciones que si son disparadas tienen marcas sin valores negativos (ninguna componente negativa en el nuevo vector posible estado).

Para determinar si la transición tiene evento solicitando el disparo, o es automática, se realiza una operación "*or*" entre el "Vector con disparos automáticos" y los "Vector con las solicitudes de disparos (salida de las colas de entrada)".

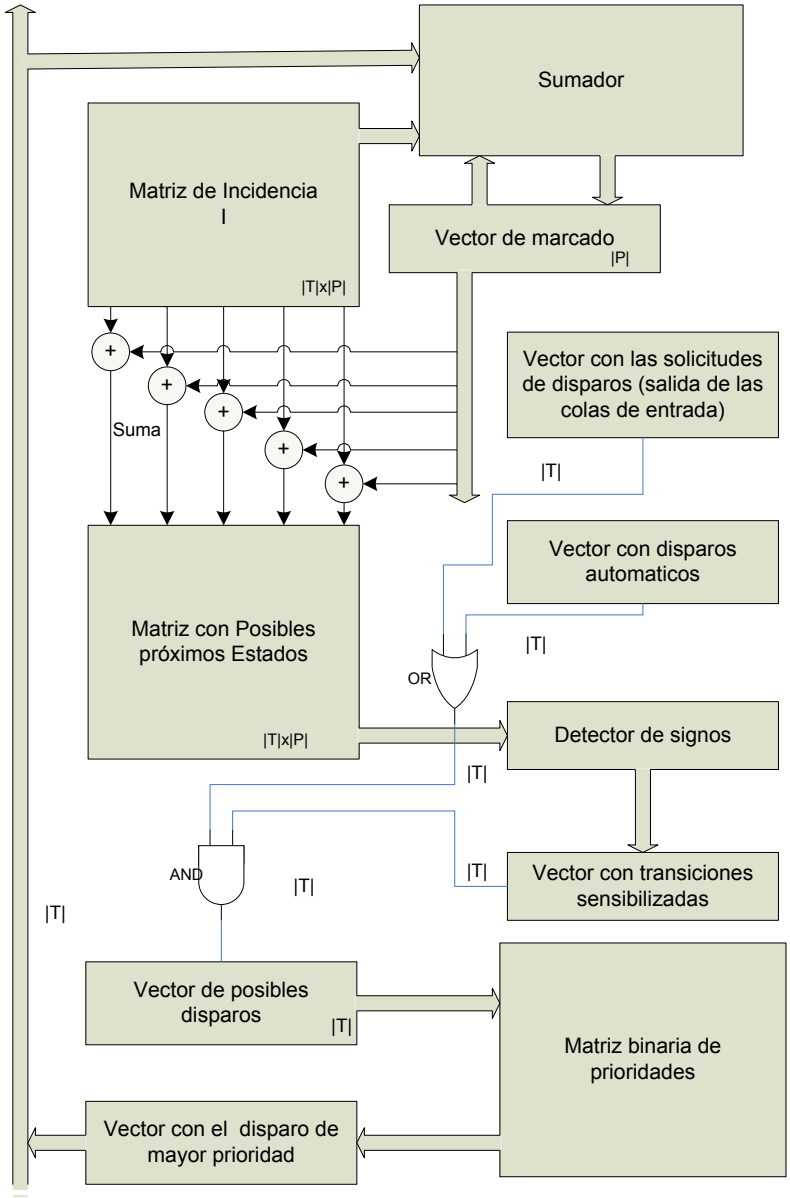


Figura 58: Diagrama en bloque del circuito del PP

Con los valores de estos dos últimos vectores se realiza una operación "*and*" para obtener todas las transiciones sensibilizadas con evento solicitando su disparo (o automática).

Con este vector y la "Matriz binaria de prioridades" se obtiene la transición sensibilizada con eventos de mayor prioridad, para realizar el disparo y obtener la próxima marca.

La Figura 58 muestra un diagrama en bloque de la implementación del algoritmo ejecutado por el PP, para determinar que transiciones se dispara y el nuevo estado.

Colas de Entrada y Salida del PP

La responsabilidad de las colas de entrada/salida es almacenar los eventos, que son las solicitudes de disparo de una transición generada por un proceso (entrada) o la comunicación del PP a un proceso de un disparo realizado (ver Figura 54). Es necesario, que cada cola sea asignada a una transición, puesto que cada cola almacena n eventos con el mismo objetivo, asociar una transición a un proceso.

Para la implementación de la cola es suficiente usar un contador saturado. Se consideran contadores saturados, cuando están en cero para la cuenta mínima y en máximo para la cuenta máxima (todos los bits en 1). También tienen una señal de salida para cuenta mayor que cero y una para máximo, una señal de entrada de incremento y una de decremento de cuenta.

Las colas de entrada, son los receptores y emisores de eventos por el PP, son enviados y recibidos por los procesos.

En la cola de entrada, cada vez que se escribe, el contador de entrada se incrementa en uno, indicando que hay una nueva solicitud de disparo de la transición.

Cuando la transición se dispara, para esto la transición debe estar sensibilizada y tener al menos un evento en su cola, el contador de entrada de la transición se disminuye en uno y el de salida de la transición disparada se incrementa en uno.

En las colas de salidas, se almacenan los eventos generados por el PP, y cada transición está relacionada con una cola de salida. Cada cola almacena los eventos que son producidos por el disparo de una misma transición.

Cada vez que se accede a leer un valor de la cola de salida, el contador se transfiere al bus y se disminuye en uno. Por ser saturado, si es cero no disminuye y si es máximo se puede inferir que se ha perdido al menos un evento (una cuenta).

Esta política ha resuelto el problema de asignar eventos a las transiciones con muy poco agregado de hardware por transición, solo dos contadores saturados.

En el apartadoColas de Entradas se expone con más detalle las funciones, implementación y modos de programación de las colas.

Señales de Control

A continuación se describen algunas de las señales de conexión entre el bus PLB y PP, y sus responsabilidadades:

- **Bus2IP_Clk:** Bus to IP clock. El clk del bus PLB, es el clk del PP. Su responsabilidad es: con cada pulso de este clk el procesador calcula los disparos posibles, y también el manejo de las colas de entrada y salida.

- **Bus2IP_Reset:** Bus to IP Reset. Es también el reset del PP. Su responsabilidad es: se activa el proceso de inicialización del PP, poniendo a cero todos los registros internos y vuelve cero todos los contadores de las colas.
- **Bus2IP_Data:** Bus to IP data bus. Es el bus de 32 bits, en donde se transfieren los datos enviados desde el MicroBlaze a PP. Este bus es leído por el PP para cargar los disparos y la inicialización del módulo, es decir, para cargar la matriz de incidencia, la inhibidora y el vector de Estado.
- **Bus2IP_WrCE:** Bus to IP write chip enable. Esta señal se utiliza para direccionar cuál de los registros del PP son visibles desde el software de usuario, para ser accedido. El PP, captura esta señal en un registro llamado **slv_reg_carga**, para posteriormente ser usada, cuando se trata de encolar un disparo, cargar la matriz de incidencia, cargar la matriz inhibidora o cargar el vector de estado.
- **Bus2IP_RdCE:** Bus to IP read chip enable. Esta señal se utiliza para seleccionar cuál de los registros visibles para el software de usuario se desea acceder para su lectura. Internamente en el PP, esta señal es capturada en un registro llamado **slv_reg_read_sel**, para posteriormente decidir si se quiere leer la cola de salida.

Desarrollo de las Aplicaciones de Prueba

Una vez implementada la plataforma de hardware embebido con el PP, el primer paso para probarla es correr el sistema operativo sobre esta y hacer pruebas básicas, como acceso de escritura y lectura a los registros del PP. Luego con el sistema operativo funcional, se ejecutan aplicaciones multi-hilo que requieren sincronización para acceder a recursos compartidos, y realizar la sincronización con el PP y luego con semáforos, para contrastar su rendimiento.

Implementación del Sistema Operativo

Xilinx, con la herramienta EDK, provee un Kernel con todas las características que necesitamos para desarrollar las aplicaciones embebidas. Este Kernel se llama Xilkernel y se puede compilar dentro de la herramienta de desarrollo de software para los sistemas embebidos llamada SDK. Cuando se compila el Xilkernel se le puede configurar, con los parámetros que determinaran sus características, para ajustar el tamaño (funcionalidades) según los requerimientos.

En nuestro caso se requiere del uso de semáforos, memoria compartida, manejo de hilos, etc. La compilación del Kernel genera una biblioteca que se usa en las aplicaciones y que se linkea con el programa compilado. El resultado es un archivo binario ejecutable en la plataforma embebida (.ELF) que contiene el sistema operativo más las aplicaciones, para ser corrido.

Se compila el Kernel con soporte para el uso de semáforos y el módulo RS-232. Con estos semáforos, se crea una aplicación que hace uso de exclusión mutua entre los hilos para acceder al recurso compartido del módulo RS-232. Como se puede ver en la Figura 59, las impresiones de los hilos se dan en forma correcta.

El siguiente paso, fue probar el acceso a los registros del PP y su funcionamiento. Para esto se desarrolló una RdP simple, con la que se inicializó el PP (matriz de incidencia y el vector de estado). Una vez inicializado el PP, se realizaron disparos sucesivos, que se encolaron en las distintas transiciones de la RdP, y se realizó la lectura de las colas de salida. Comparando los disparos de entrada, el estado de la red y los disparos de salida se pudo verificar el funcionamiento del sistema.

Figura 59: Ejecución de Xilkernel con hilos sincronizados por semáforos.

Una vez realizadas estas pruebas, se validó el funcionamiento del sistema (Procesador, SO y PP). Seguidamente, se continuó con el desarrollo de aplicaciones multihilo que simulen casos de sincronización reales, los que son presentados a continuación.

Desarrollo de las Aplicaciones

Se desarrollaron aplicaciones multihilo, que requieran de una sincronización explícita, con el fin de comprobar la capacidad del PP para las tareas de sincronización. Para la solución de los casos planteados se realizaron tares secuenciales programadas en C, la lógica paralela se modela y se programa con una RdP, como lo muestra la Figura 60 (a) y (b).

En la Figura 60 (a) se representan los hilos, cuando éstos solicitan un recurso compartido encola, en una cola asociada a una transición del PP, un disparo que representa la solicitud de un recurso por el hilo. El PP, que ejecuta la lógica de gestión de sincronización, decidirá si esta condición es válida y si se puede efectuar la toma del recurso, realizando el disparo y colocando el resultado (evento) en la cola de salida asociada a la transición. Este disparo es leído de la cola de salida por un hilo, el cual es el encargado de continuar con la ejecución de sus tareas secuenciales.

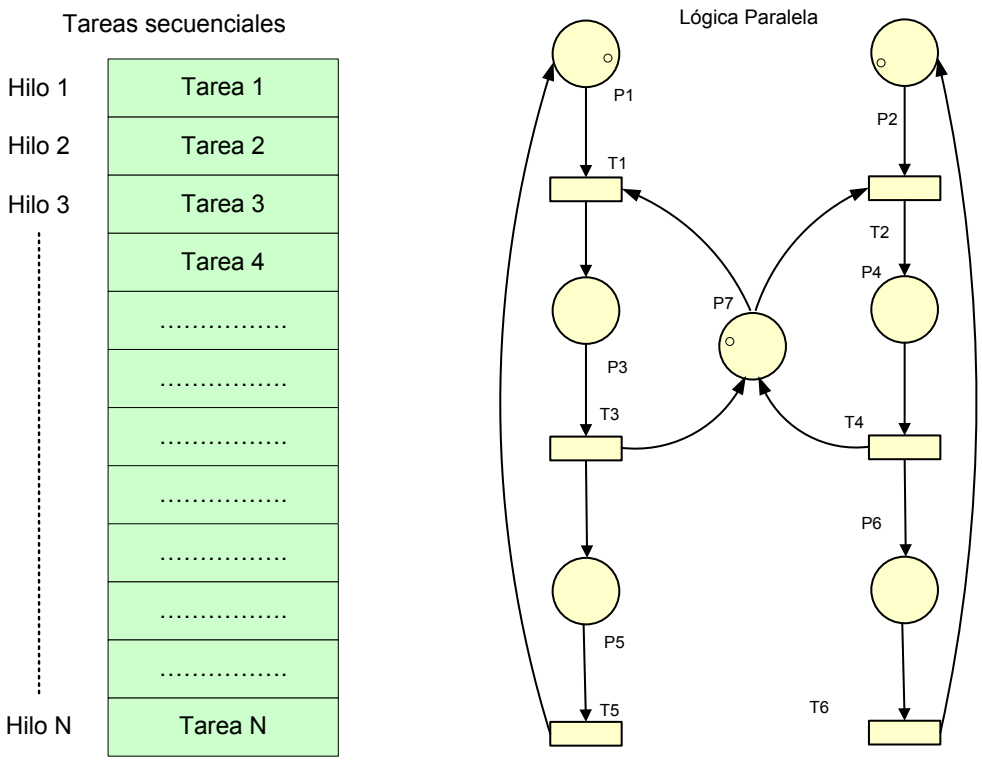


Figura 60: (a) Hilos secuenciales, (b) la lógica paralela de las aplicaciones que gestiona la sincronización

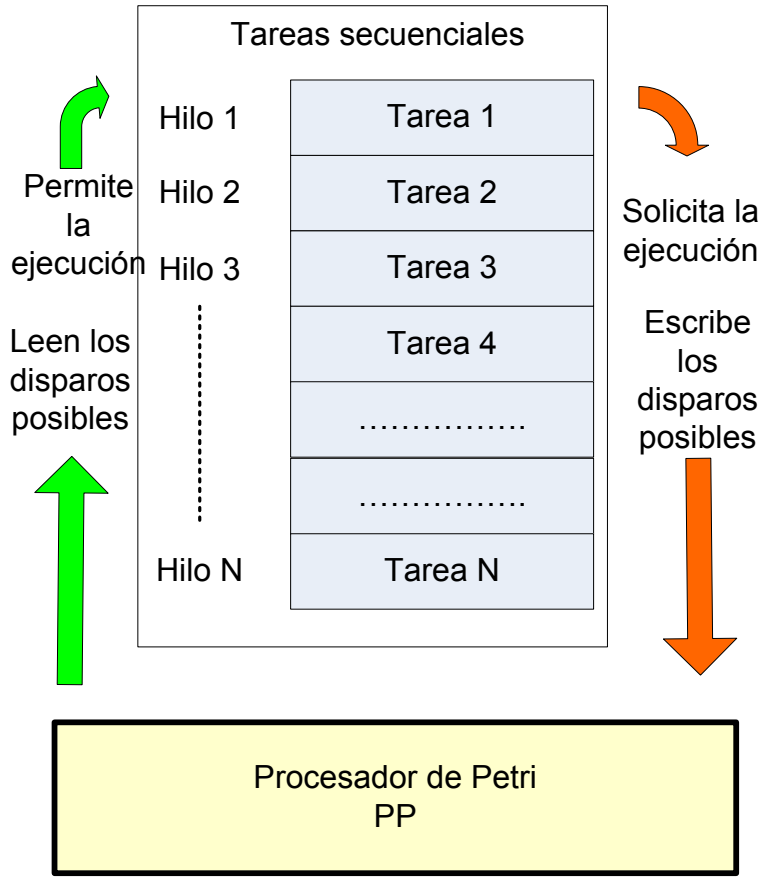


Figura 61: Eventos entre hilos y el PP, y evento entre el PP y los hilos

Estas aplicaciones requieren una fase de inicialización donde se programa al PP y los procesos que han sido compilados como hilos. Esta lógica se describe y modela mediante RdPnA.

Del modelo de gestión del programa paralelo, se obtiene las matrices y vectores que determinan el comportamiento según los requerimientos de sincronización. Una vez realizada esta inicialización, el PP queda programado. El PP puede aceptar disparos para procesarlos y decidir la ejecución según el estado del sistema, generando los eventos de salida y la evolución de la RdP.

La Figura 62 muestra el diagrama de secuencias de programación del PP.

En la Figura 62 podemos ver un hilo principal que crea a los demás hilos. Estos hilos ejecutan las tareas que son sincronizar. Esto se realiza solicitando disparos y consultando la resolución para ejecutar las tareas específicas.

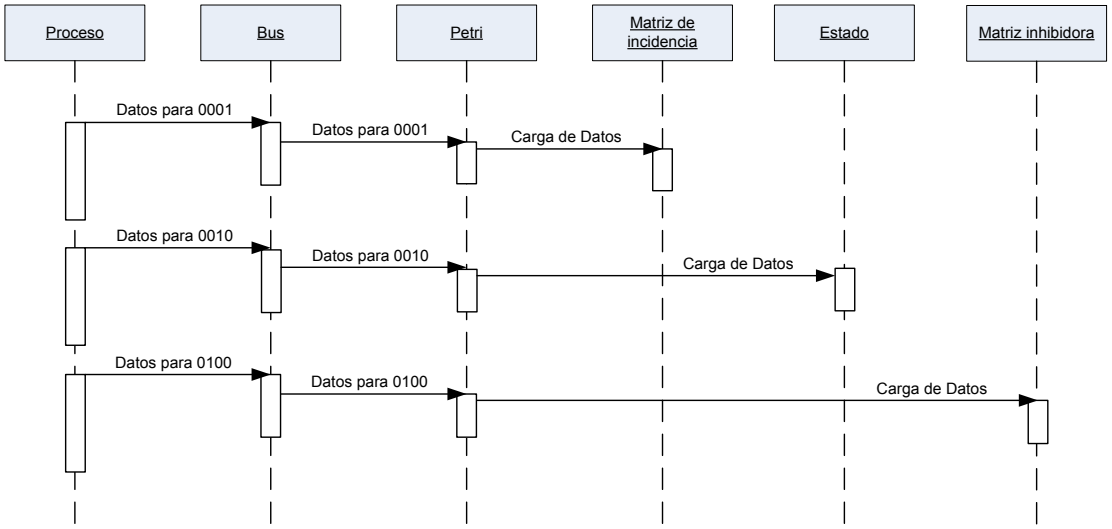


Figura 62: Secuencia de carga de datos al PP

La Figura 63 ilustra el diagrama de secuencia, que muestra la sincronización entre dos hilos que intentan acceder a un recurso compartido, en este caso es el módulo de RS-232.

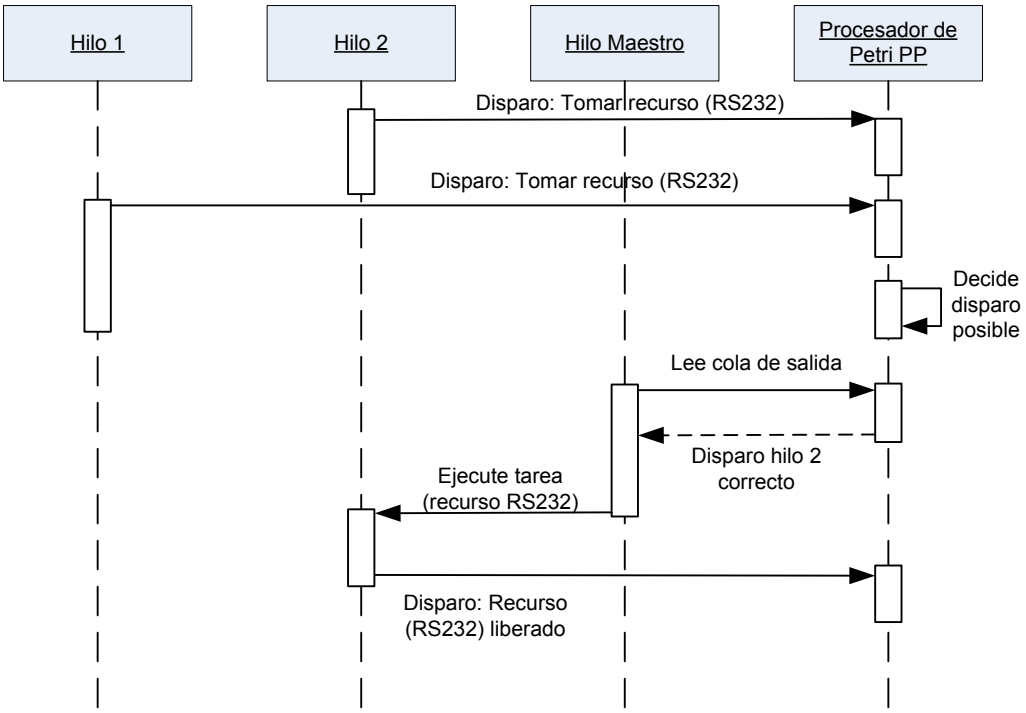


Figura 63: Sincronización de dos hilos, controlada por el PP

Medición del Rendimiento

Para evaluar el rendimiento se tomaron las medidas de tiempo de las aplicaciones resueltas con el PP, luego se realizaron las mismas mediciones en la aplicación resuelta con semáforos con el fin de determinar el impacto del PP. Se usó la plataforma de Xilinx y los clk del SO Xilkernel.

Con las funciones de la API del SO se obtiene la cantidad de clk, esta función es llamada con xget_clock_ticks(). De esta forma se hacen dos llamadas a la función: una al comienzo de la ejecución y otra al final, para luego hacer la diferencia y calcular los clk totales empleados en la ejecución.

Un clk del sistema se cuenta cuando el procesador es interrumpido por un Timer externo. Este Timer externo que interrumpe al procesador periódicamente es el IP Core xps_timer en la plataforma de hardware. La función es proveer una interrupción periódica para que el SO Xilkernel realice las tareas de Scheduling cada cierto intervalo de tiempo. Este intervalo de tiempo se puede modificar durante la compilación del Xilkernel, y por defecto tiene un valor de 10 ms. Para las pruebas se ha dejado el valor por defecto.

Caso de Prueba para Dos Escritores

Se ha tomado un caso con dos hilos independientes, que se comportan como escritores, que acceden en forma concurrente a un recurso compartido, que es la UART RS-232. Cada hilo imprime en pantalla un saludo y su número de identificación, numero de n veces, para luego terminar su ejecución.

El acceso al recurso, módulo UART, requiere más tiempo para imprimir por pantalla que la cantidad asignada por el SO. El acceso a dicho módulo requiere más ciclos de reloj de los que el

sistema operativo Xilkernel le dá a cada hilo como tiempo de ejecución, antes de cambiarlo por otro hilo. Si no hacemos uso de la exclusión mutua, como resultado, vemos que las impresiones de caracteres enviados por ambos hilos se ven intercaladas. En cambio, si sincronizamos para obtener exclusión mutua sobre el recurso compartido, la impresión es correcta. Sólo un hilo tiene acceso al recurso hasta que termine de imprimir, evitando el problema.

Presentamos dos alternativas para la solución, las que son: por software, usando semáforos del SO Xilkernel, y la segunda por hardware, haciendo uso del PP para obtener el correcto acceso al recurso compartido. Posteriormente se ejecutan y comparan las mediciones de tiempo empleadas en la ejecución del programa, con el objeto de determinar el rendimiento del PP, respecto a los semáforos. En la Figura 64 se muestra el modelo de RdP, para dos escritores con prioridad del de la derecha.

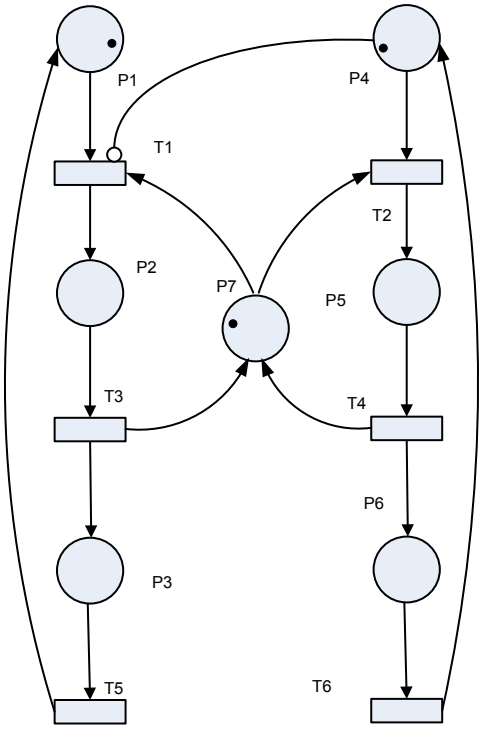


Figura 64: Modelo de RdP con dos escritores

Es posible validar la RdP y determinar, como lo muestra la Figura 65, las matrices de incidencia (a), de brazos inhibidores y (b) el estado inicial del sistema. Con estas matrices y vectores se inicializa el PP. Estas matrices han sido obtenidas con el simulador PIPE [235].

	T1	T2	T3	T4	T5	T6
P1	-1	0	1	0	0	0
P2	1	-1	0	0	0	0
P3	0	1	-1	0	0	0
P4	0	0	0	-1	0	1
P5	0	0	0	1	-1	0
P6	0	0	0	0	1	-1
P7	-1	1	0	-1	1	0

(a)

	T1	T2	T3	T4	T5	T6
P1	0	0	0	0	0	0
P2	0	0	0	0	0	0
P3	0	0	0	0	0	0
P4	1	0	0	0	0	0
P5	0	0	0	0	0	0
P6	0	0	0	0	0	0
P7	0	0	0	0	0	0

(b)

P1	1
P2	0
P3	0
P4	1
P5	0
P6	0
P7	1

(c)

Figura 65: (a) Matriz Incidencia, (b) matriz de brazos inhibidores, (c) vector de estado

El paso siguiente es el desarrollar en el SDK de la aplicación, que consta de dos hilos ejecutores y un hilo para la inicialización.

Los hilos de ejecución, usan el PP o semáforos provistos por el SO Xilkernel para realizar la sincronización entre los distintos hilos. Una vez terminado el trabajo de cada hilo mide la cantidad de clk. En el caso que se usa PP, existe un hilo adicional que es el hilo de inicialización y que realiza una espera activa leyendo el PP.

Las pruebas se repitieron 100 veces para cada caso, obteniendo resultados idénticos en casi todos los casos, puesto que el SO Xilkernel utiliza un Scheduling simple de Round Robin.

La Figura 66 muestra la salida de pruebas realizadas, primero con semáforos y luego con el PP.

La Figura 67 muestra la salida del número de clk del SO para ejecutar el caso resuelto con el PP.

Las mediciones obtenidas en los casos resueltos con semáforos son de 40 clk y para los casos con el PP son de 29 clk. Esto implica un aumento de rendimiento por el uso del PP de un 27,5%.

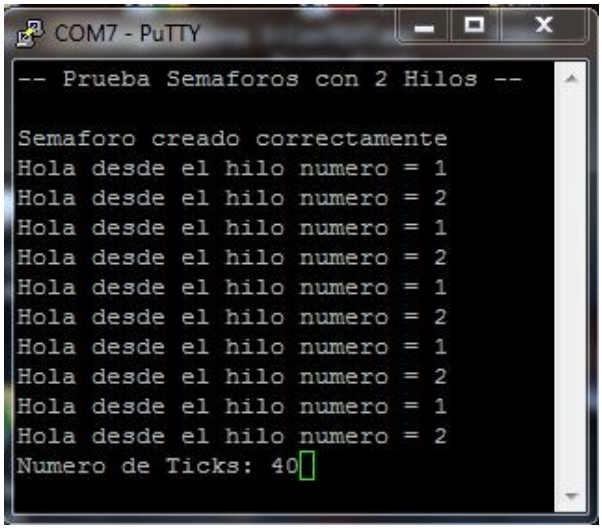


Figura 66: Pruebas con dos hilos y semáforos

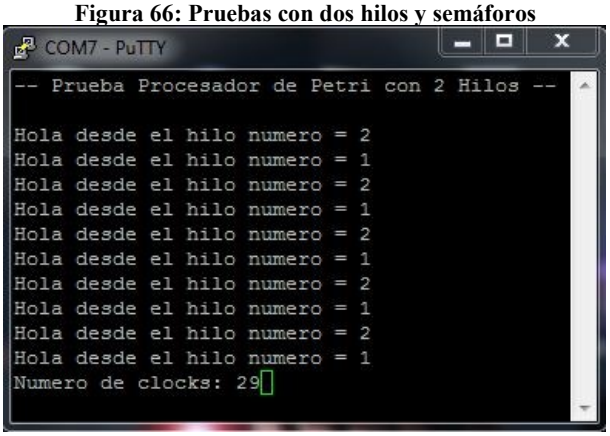


Figura 67: Pruebas con dos hilos y PP

Caso de Prueba para 4 Escritores

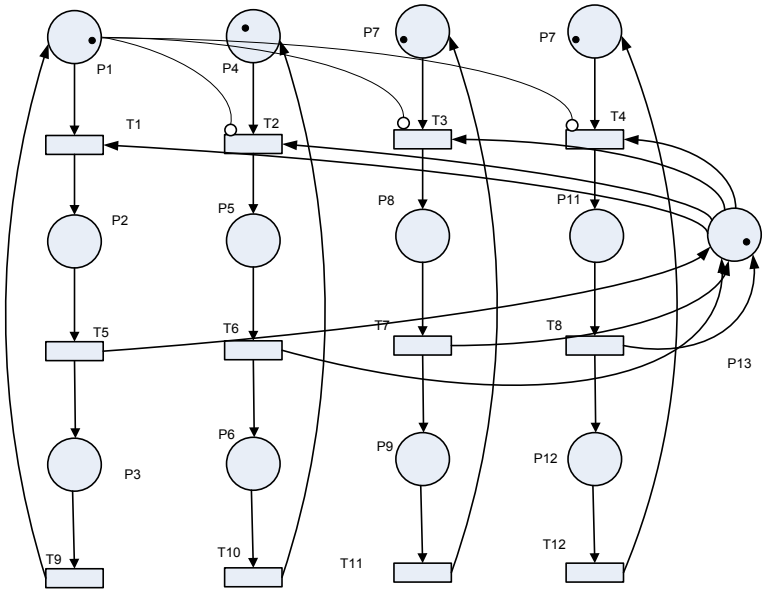


Figura 68: RdP para cuatro escritores con prioridad

Para esta prueba, el caso planteado en el ítem anterior se extendió a 4 escritores sobre el recurso compartido. En este apartado también se implementaron las aplicaciones con semáforos y PP para compararlas. La Figura 68 muestra la RdP que modela el sistema de gestión del recurso.

Las matrices de incidencia e inhibidora son mostradas en las Figura 69 (a) y (b) respectivamente, mientras que el vector de estado se muestra en la Figura 69 (c).

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
P1	-1	0	0	0	0	0	0	0	1	0	0	0
P2	1	0	0	0	-1	0	0	0	0	0	0	0
P3	0	0	0	0	1	0	0	0	-1	0	0	0
P4	0	-1	0	0	0	0	0	0	0	1	0	0
P5	0	1	0	0	0	-1	0	0	0	0	0	0
P6	0	0	0	0	0	1	0	0	0	-1	0	0
P7	0	0	-1	0	0	0	0	0	0	0	1	0
P8	0	0	1	0	0	0	-1	0	0	0	0	0
P9	0	0	0	0	0	0	1	0	0	0	-1	0
P10	0	0	0	-1	0	0	0	0	0	0	0	1
P11	0	0	0	1	0	0	0	-1	0	0	0	0
P12	0	0	0	0	0	0	0	1	0	0	0	-1
P13	-1	-1	-1	-1	1	1	1	1	0	0	0	0

(a)

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
P1	0	1	1	1	0	0	0	0	0	0	0	0
P2	0	0	0	0	0	0	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	0	0
P5	0	0	0	0	0	0	0	0	0	0	0	0
P6	0	0	0	0	0	0	0	0	0	0	0	0
P7	0	0	0	0	0	0	0	0	0	0	0	0
P8	0	0	0	0	0	0	0	0	0	0	0	0
P9	0	0	0	0	0	0	0	0	0	0	0	0
P10	0	0	0	0	0	0	0	0	0	0	0	0
P11	0	0	0	0	0	0	0	0	0	0	0	0
P12	0	0	0	0	0	0	0	0	0	0	0	0
P13	0	0	0	0	0	0	0	0	0	0	0	0

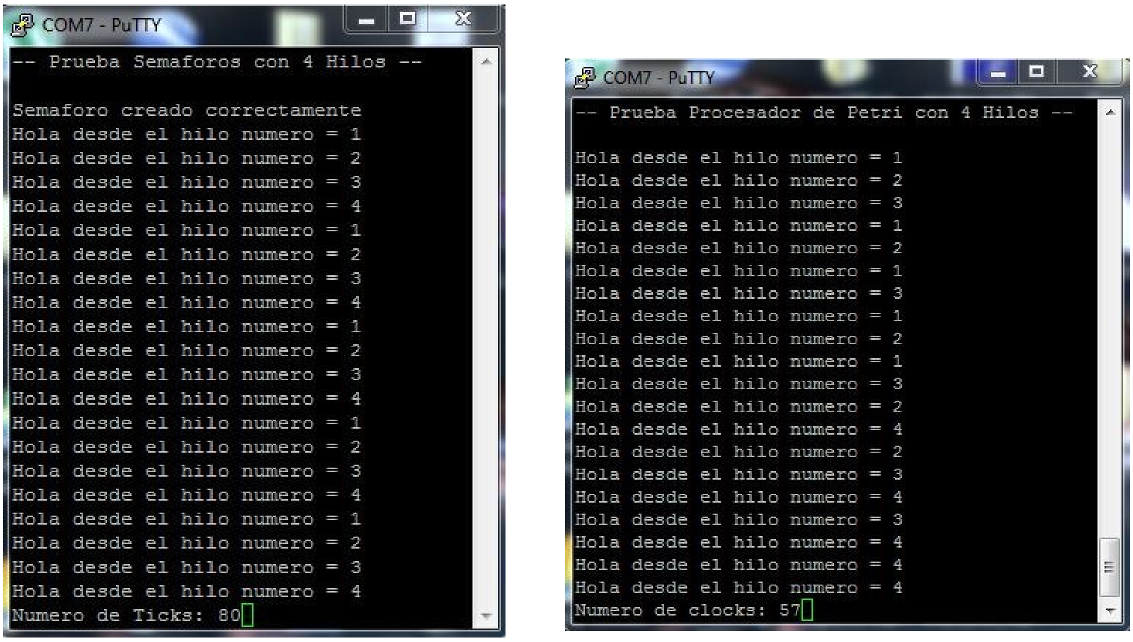
(b)

P1	1
P2	0
P3	0
P4	1
P5	0
P6	0
P7	1
P8	0
P9	0
P10	1
P11	0
P12	0
P13	0

(c)

Figura 69: (a) Matriz incidencia, (b) matriz de brazos inhibidores, (c) vector de estado

Como en el apartado anterior se ejecutó la prueba n veces, y en la Figura 70 se muestran los resultados obtenidos.



(a) **(b)**

Figura 70: Ejecución de cuatro escritores, en (a) con semáforos y en (b) con PP.

En este caso, el número de clk requeridos para la ejecución con semáforos es de 80 clk y para la ejecución con el PP es de 57 Clk. Por lo que la mejora de desempeño usando el PP es de 28,75%.

Caso de Prueba para 6 Escritores

Finalmente se implementó un caso con seis escritores sin prioridad, como lo muestra el modelo de la Figura 71, repitiéndose los mismos pasos y mediciones que en los dos casos anteriores.

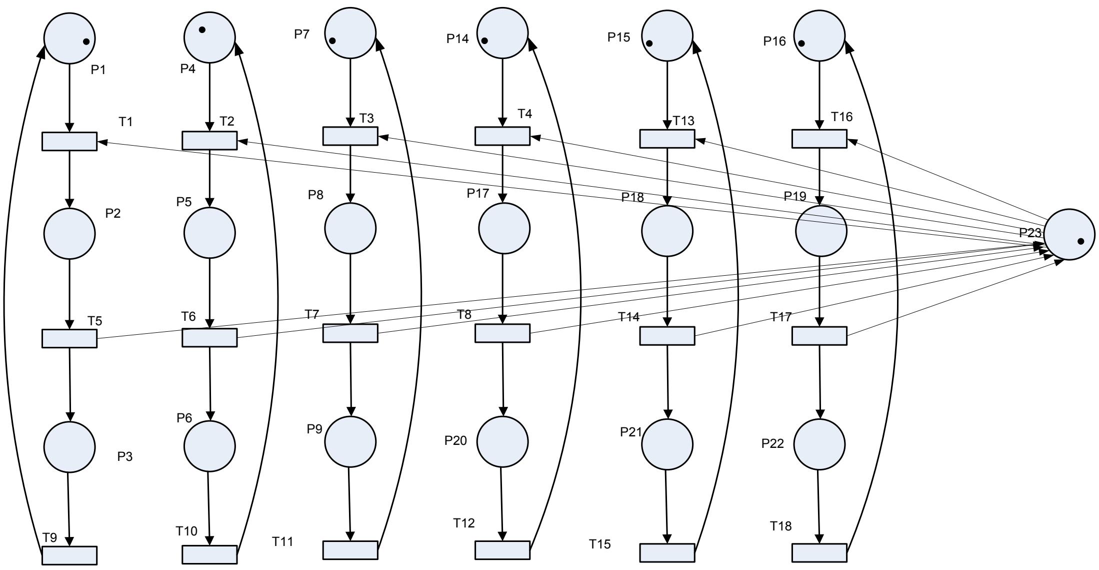


Figura 71: RdP con seis escritores y sin prioridad

Los resultados obtenidos para seis escritores son: 62 clk para la solución con semáforo y 48 clk para la solución con el PP. Esto implica una mejora de desempeño con el PP de 22,58%.

Nota: las pruebas para esta última medición se implementaron y ejecutaron en la placa de desarrollo Atlys con Spartan 6.

Análisis y Resultados

La Tabla 101 muestra los resultados obtenidos para las pruebas realizadas con dos, cuatro y seis escritores, con tiempos medidos en clk del SO. El aumento promedio del desempeño obtenido con el PP respecto al uso de semáforos es del 28%, que son valores similares a los obtenidos en la simulación realizada con el SESC en la sección "Determinación de la Arquitectura para integrar el PP a un Sistema SMP".

Tabla 101: Resultado de las pruebas con 2, 4 y 6 escritores

Prueba	Clk con Semáforos	Clk con Petri	Mejora de rendimiento con Petri
2 hilos	40	29	27,50%
4 hilos	80	57	28,75%
6 hilos	98	76	28,94%

La Figura 72 se muestra la gráfica de los resultados obtenidos. Podemos notar que los tiempos de ejecución se reducen en un 28%.

La Figura 72 muestra el al incremento de rendimiento por el uso del PP para realizar la sincronización. Podemos notar que la mejora de rendimiento se mantiene.

También podríamos suponer que el rendimiento disminuye al aumentar el número de hilos, debido al cuello de botella que se genera por las consultas que realizan los procesos y/o semáforos haciendo uso del bus del sistema. Una solución para este problema sería implementar múltiples bases.

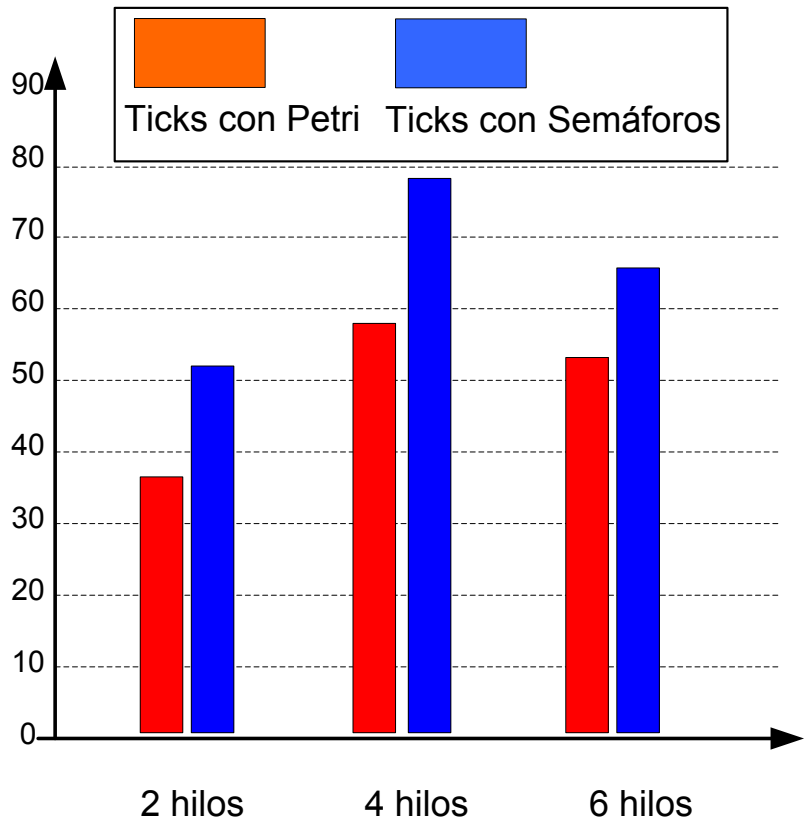


Figura 72: Comparación de tiempos de ejecución con semáforos y PP

Resultados del hardware empleado se muestra en la Tabla 102, donde se encuentran las características del hardware sintetizado para los casos de prueba realizados. Se detallan el tamaño de las matrices de incidencia e inhibidora que se generan dentro del procesador, así como la frecuencia máxima de trabajo que éste soporta una vez implementado. Por otro lado se muestran la cantidad de flip-flop utilizados dentro de la FPGA así como también el número de LUTs usadas para la lógica. Hay que tener en cuenta que estos últimos valores son para el sistema embebido completo y no solo para el PP.

Tabla 102: Resumen síntesis de Hardware

Prueba	Matrices	Frecuencia máxima (MHz)	Flip-Flop	LUTs
2 hilos	7x6	85	2700	4347
4 hilos	13x12	84	4144	6022
6 hilos	19x18	46	7709	19065

En la Figura 73 se grafica el número de flip-flop utilizados en el sistema para almacenar los valores. Éstos crecen de forma proporcional al tamaño de las matrices con el número de hilos. También en la Figura 73 se grafica el número de LUTs utilizados como lógica, vemos que éstos aumentan con el producto entre las plazas y las transiciones, o con el tamaño de las matrices y el número de hilos. Esto se debe a la cantidad de sumadores y comparadores que se sintetizan para realizar las operaciones, y a la complejidad del cableado entre los componentes.

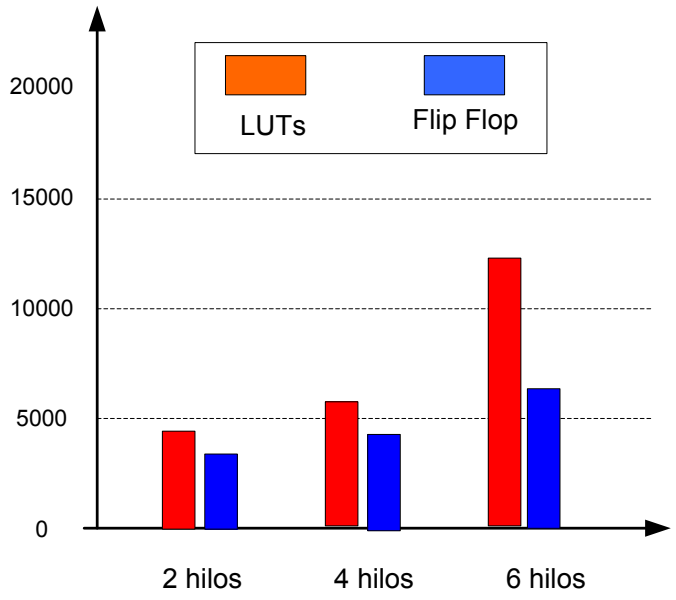


Figura 73: Uso de recursos de FPGA para implementar el PP

Una mejora importante, que se introduce en los próximos PP, es la consulta a las colas de salida. Estas son:

- Consultas activas, leer la cola hasta que se obtenga el resultado (sobrecarga del bus).

- Consultas cediendo el tiempo de ejecución, en el caso de no estar resuelto el disparo ceder el tiempo de ejecución al próximo proceso.
- Por interrupción (sobrecarga por cambio de contexto).

Esto ha sido implementado en los procesadores que se presentan en los próximos capítulos.

Resultados y Conclusiones

Con la realización del trabajo expuesto se logró validar y documentar el PP para RdPnA. Se comprobó su correcto funcionamiento y las ventajas que éste aporta para los sistemas que requieren de sincronización, no sólo en tiempo de ejecución, sino también en cuanto a la forma de resolver problemas de sincronización utilizando las RdP para desacoplar la lógica del programa paralelo de las tareas secuenciales a realizar.

Con el uso del PP se tiene un nuevo e innovador enfoque para resolver problemas que requieran sincronización. El trabajo aquí presentado está en una etapa de desarrollo, por lo que se propone extenderlo a RdPnA temporales y con tiempo, disminuir el uso de recursos de hardware, permitir más opciones a los procesos que esperan por su sincronización e incluir múltiples interrupciones.

Capítulo 5

Procesador de Petri Temporal

Resumen

En este trabajo se extiende el PP a PP con tiempo (PPcT) y temporizado (PPTm), y hacemos uso de los formalismos de las RdP con tiempo para construir y verificar el modelo. En el "Anexo de RdP con tiempo" se expone los formalismos de las RdP con tiempo y temporizadas.

Seguidamente programamos directamente el procesador extendido con los vectores y matrices del modelo, las partes del sistema que fueron modelados quedan programadas sin necesidad de codificación, de la misma manera que lo hicimos en el PP.

Este desarrollo está motivado en los sistemas reactivos de tiempo real, donde se realiza un modelo que represente el sistema con los requerimientos a implementar. Seguidamente hay que realizar la programación, verificación y validación, donde estas etapas generalmente son evolutivas.

Puesto que es deseable que exista una relación directa entre el sistema modelado y el programa a implementar, al menos en forma parcial, es que se implementa el PP con tiempo y temporizado.

La consecuencia esperada son la programación directa y la disminución de los tiempos de desarrollo y ejecución.

Objetivos

El objetivo principal de este trabajo es diseñar, implementar, verificar y validar el funcionamiento de los PP con tiempo (PPcT) y temporizados (PPTm), capaz de ejecutar una de las dos semánticas: *RdP con Tiempo* y *RdP Temporizadas*.

Manteniendo la condición de programación directa entre el modelo y la implementación.

Los objetivos Secundarios

- Evaluar y determinar cómo incorporar el tiempo en el hardware del PP para transformarlo en un PPcT y/o PPTm.
- Rediseñar el procesador PPcT y/o PPTm.
- Implementar PPcT y/o PPTm como un IP-Core en una FPGA.
- Insertar el PPcT y/o PPTm, en una FPGA que implemente un SMP.
- Proveer un mecanismo para programar el PPcT y/o PPTm.
- Implementar un mecanismo de comunicación entre el PPcT y/o PPTm y los procesos.
- Prever la comunicación entre el PPcT y/o PPTm y los procesos del sistema a través de interrupciones.

Desarrollo

PP Diseño e Implementación

En este apartado se diseña e implementan los PPcT y PPTm, como un *IP-Core* que ejecutan RdP *con Arcos Inhibidores, plazas acotadas* y *RdP con Tiempo*.

Arquitectura General del Sistema

La Figura 74 se muestra una interconexión posible del PP, esta arquitectura se deriva del análisis realizado en el capítulo " Sistema Multi-Core Sincronizado por un PP Implementado en una FPGAEl PP" donde se interconecta el PP con la arquitectura SMP conformando un sistema multi-Core heterogéneo.

En la arquitectura mostrada en la Figura 74, se ve la interconexión del PP con los dos cores MicroBlaze de Xilinx, esto ha sido realizado con un bus de comunicación *BUS AXI*.

Seguidamente se expone el diseño y la implementación del PP extendido con tiempo.

Para esto se divide el presente capítulo en dos partes.

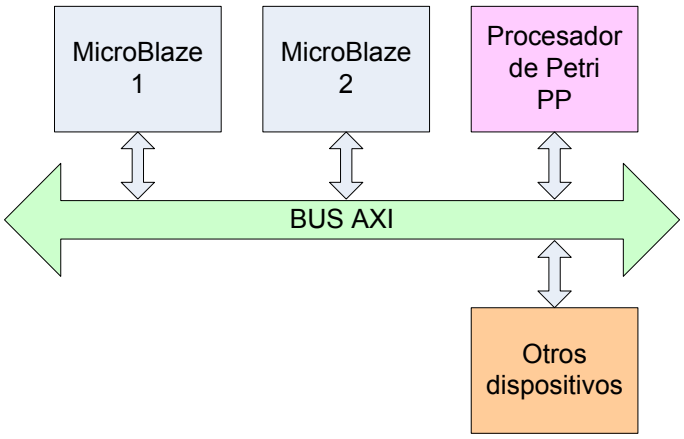


Figura 74: Arquitectura heterogénea, constituida por un PP y múltiples Microblaizer.

En la primera parte, se realiza la refactorización del trabajo presentado en el capítulo anterior. En esta refactorización se rediseñó el sistema para agregar los circuitos de temporización requeridos en los PPcT.

En la segunda parte, se diseña e implementa la arquitectura del PPTm.

Restricciones

Al ser la ejecución del modelo nuestro objetivo final, para la implementación del procesador se debe verificar algunas restricciones para permitir la ejecución, las que son:

- Ejecutar RdP con arcos con peso, RdP ordinarias (arcos de peso enteros).
- Ejecutar RdP con arcos inhibidores, arcos de lectura, stop-watch, stop-watch inhibidores y arcos de reset.
- Ejecutar con RdP plazas acotadas.
- Verificar automáticamente que la red tenga al menos una transición sensibilizada.
- Si sólo consideraremos sistemas secuenciales de código no reentrante el grado de sensibilidad puede ser uno, lo cual puede ser implementado por:
    - Restringiendo a que una de las plazas de entrada de la transición sea uno limitado

    - Por la política de disparo, hasta que no finalice un disparo sensibilizado no se realiza otro.
- Al menos para las transiciones de acción (TA), permitir el disparo de múltiples transiciones, observando las siguientes restricciones.
    - Las transiciones del tipo TA, no pueden estar en conflicto. Esto evita conflictos en la ejecución entre distintas transiciones TA.
    - Si un modelado no puede ser alcanzado por esta restricción, esta podrá no ser considerada como restricción sii la indeterminación es levantada por la política de prioridades de disparo del procesador.
- No incluiremos en nuestros modelos de ejecución (pero el PP las admite):
    - Transiciones sin lugares de entrada,
    - Transiciones que constantemente pueden ser disparadas para introducir marcas en el sistema.
- Las transiciones del tipo AP (transiciones de actividades periódicas) y AS (transiciones de sincronización) podrán tener varias plazas de entrada. De este modo se evita que puedan existir conflictos entre transiciones TA, lo que modela que un sistema se está ejecutando. Las transiciones AP cumplen con $t_{min} = t_{max}$, los que nos permite representar un evento en un instante determinado y además se cumple que $t_{min} > 0$, de otra forma se representará con una transición del tipo AS.

Requerimientos para la Construcción del PP

- Rediseñar el PP del capítulo anterior, para: implementar plazas acotadas brazos inhibidores y de reset.
- Rediseñar el PP para poder programar de recepción de evento de las colas de entrada, en los siguientes modos:
    - Cola de eventos explícita. Es necesario el evento externo, el orden de prioridad y la transición sensibilizada para su disparo
    - Cola de eventos automática. Disparos automáticos de transiciones. Es decir, cuando la transición este sensibilizada y tenga prioridad, sin esperar un evento de pedido de disparo explícito, se dispare.
    - Cola con eventos perenes. Para el disparo, es necesario que la transición esté sensibilizada cuando llega el evento. Si la transición no está sensibilizada, cuando llega el evento, se pierde el evento.
- Rediseñar el PP para poder programar los eventos de salida:
    - El disparo no genera evento de salida, no tiene sentido su lectura.
    - El disparo genera un evento en la cola de salida relacionada con la transición, lo que incrementa el contador de eventos relacionado con la transición. La lectura de la cola de salida realiza las siguientes acciones según su programación:
        - Si se lee solo una cola: cuando la cuenta de eventos es mayor que cero se disminuye en uno la cantidad de eventos y retorna un uno. Si la cuenta es cero retorna un cero y la cuenta de eventos es cero.
        - Si se lee un conjunto de colas (32 o 16 bit) retorna un uno en el bit relacionado con la transición, cuando la cuenta de la cola sea mayor que uno y un cero cuando la cuenta es cero. A todas las cuentas, de cada cola, mayor que cero las disminuye en uno y a las que son cero las mantiene en cero.

Ampliación de la Arquitectura del PP

En la Figura 75, podemos ver el área de datos del PP, estos módulos se inicializan con los valores que programan al PP. Los vectores y matrices que se cargan son la instancia de la RdP, las prioridades de las transiciones y el modo de operar de las colas.

El módulo de cálculo de la ecuación de estado, calcula la sensibilidad de todas las transiciones. Esto lo hace realizando la suma entre cada columna de la matriz de incidencia con el estado actual y luego verificando si todas las componentes (plazas) de cada vector resultante son cero o positivo. Si ninguna de las componentes del vector resultantes es negativa la transición esta sensibilizada por los token. En paralelo calcula la inhibición que producen los arcos inhibidores por el marcado y que el peso de las nuevas marcas no supere la cota de marca. También en paralelo, se realiza una operación *or* entre el eventor solicitando el disparo y los disparos automáticos, con lo que se obtiene un vector con todas las transiciones a las que se les solicita el disparo.

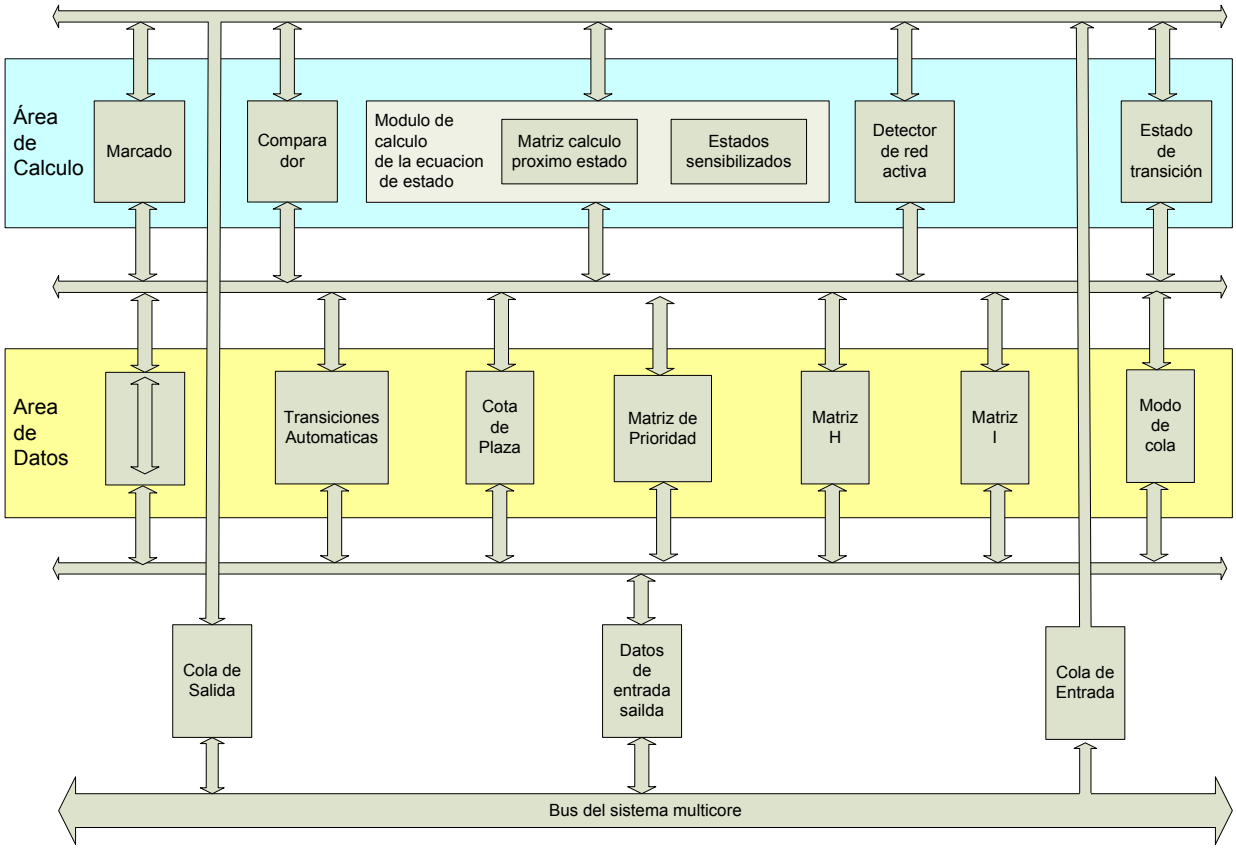


Figura 75: Arquitectura del PP ampliada con brazos inhibidores y cota de plaza

Con estos cuatro vectores se realiza una operación *and* y se obtiene el vector con posibles disparos. A este vector se lo multiplica por la matriz de prioridad y se obtiene sólo una transición que cumple con las siguientes condiciones: es la transición que esta sensibilizada, tiene evento solicitando el disparo, el estado que se obtiene no supera el límite de cota, los brazos inhibidores no impiden su dispara y es la transición de mayor prioridad.

Este vector es el que le indica al sumador que haga la suma entre la columna que se corresponde con la transición y el estado actual, y luego lo almacene en el registro de vector de marca.

El período en el que se realiza todas estas operaciones es en un ciclo de reloj del PP, a esto lo llamamos ciclo del PP. Este ciclo es usado para resetear las solicitudes de disparo (eventos) no-perenne en las colas de entrada.

Arquitectura del PPcT

A continuación exponemos la arquitectura PPcT, esta arquitectura es realizada en bloques que implementan las funciones de la ecuación de estado. Esta ecuación de estado ha sido ampliada a brazos inhibidores y plazas acotadas. Posteriormente, se describe cada una de sus funcionalidades y su interrelación.

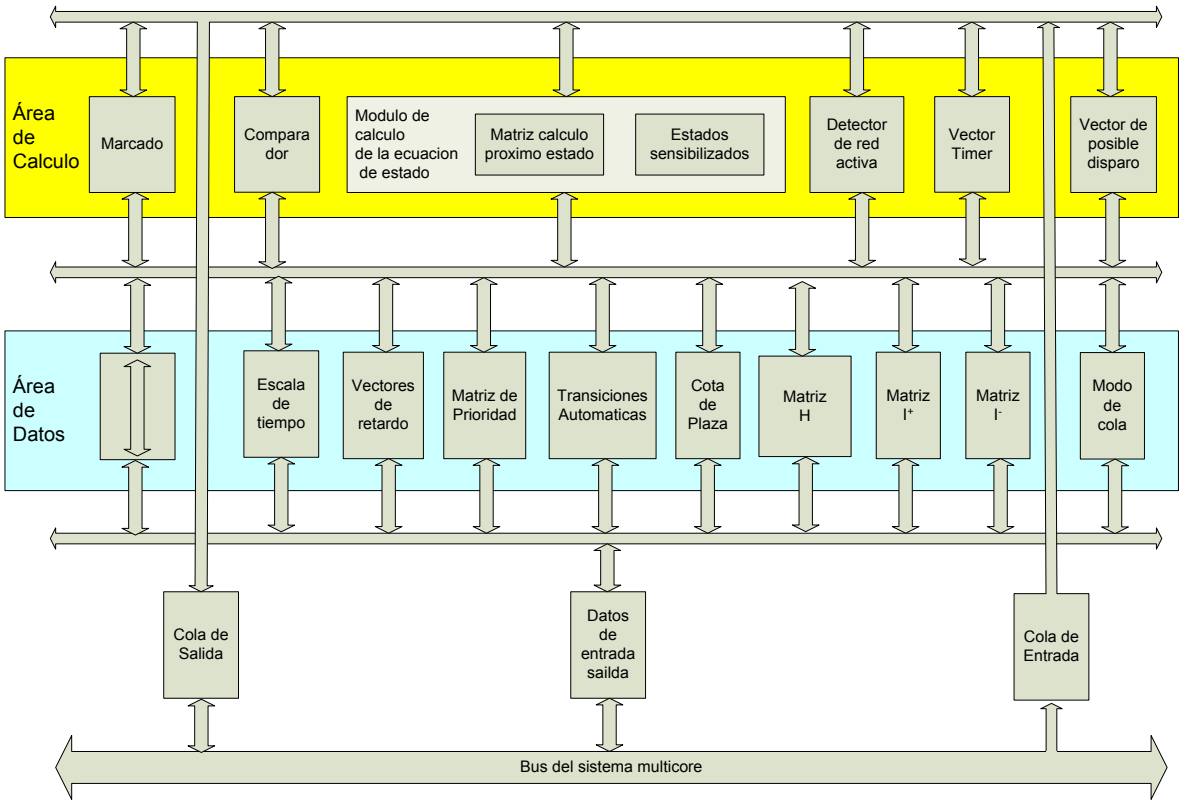


Figura 76: Arquitectura del PPcT

*Matriz **I***

La matriz I, es una matriz que tiene como cantidad de filas el número de plazas $|P|$ y como cantidad de columnas el número de transiciones $|T|$. Cada elemento de esta matriz es un número entero con signo.

```
//Parametros

localparam cant_plazas          = 4;
localparam cant_transiciones    = 2;
localparam tamano_de_elementos  = 8;

//Matriz de Incidencia

reg signed [tamano_de_elementos-1:0] matriz_incidencia [cant_plazas-1:0] [cant_transiciones-1:0];
```

Figura 77: Matriz de Incidencia (en Verilog)

El parámetro *"tamaño_de_elementos"*, define el tamaño en bits de cada elemento de la matriz. La matriz es declarada como ***"signed"***, el *"tamaño_de_elemetos"* igual a "8", como muestra la Figura 77, define un rango entre -128 hasta 127. El valor de cada elemento representa el peso del arco que une una plaza con una transición, para este caso se han tomado ocho bits con signo. No obstante, el tamaño de cada elemento es un parámetro en el programa lo cual hace que su modificación sea simple. La entrada salida del módulo es:

- La entrada a este módulo (inicialización) se realiza en la programación y son los valores de la matriz de incidencia, éstos puede ser reprogramados en tiempo de ejecución.
- La salida de este módulo son los valores de la matriz de incidencia.
- La responsabilidad de este módulo es almacenar los valores de la matriz de incidencia, para el cálculo de los nuevos estados.

*Matriz de **H***

La matriz de inhibición o *H* es una matriz binaria. Por lo tanto, cada elemento de esta matriz es de ***un bit***.

La entrada salida del módulo es:

- La entrada a este módulo (inicialización) se realiza en la programación y son los valores de la matriz de inhibición, estos puede ser reprogramado en tiempo de ejecución.
- La salida de este módulo son los valores de la matriz de inhibición.
- La responsabilidad de este módulo es almacenar los valores de la matriz de inhibición, para el cálculo de los nuevos estados.

```
reg matriz_inhibicion [cant_plazas-1:0] [cant_transiciones-1:0];
```

Figura 78: Matriz de Inhibición implementada en Verilog

Marcado inicial y marcado

Definimos dos vectores, los que son: el ***"marcado inicial"*** que es asignado en la carga de datos y el ***"marcado"*** que es la marca actual de la RdP. Este último es inicializado con el vector *"marcado inicial"* y luego evoluciona con los disparos según la ecuación de estado. La entrada salida al módulo es:

- La entrada a este módulo (inicialización) se realiza en la programación y son los valores del vectores de marcado inicial, estos también puede ser reprogramados en tiempo de ejecución.
- La salida de este módulo son el valor del vector de estado calculado y la marca inicial.
- La responsabilidad de este módulo es almacenar los valores del vector de estado y de la marca inicial, para el cálculo del nuevo estado.

```
reg [tamano_de_elementos-1:0] marcado_inicial [cant_plazas-1:0];

reg [tamano_de_elementos-1:0] marcado [cant_plazas-1:0];
```

Figura 79: Vectores de Marcado implementados en Verilog

Cotas de plazas

Las cotas en las plazas, es un vector donde cada elementos del vector representa el número máximo de token en la plazas. También debemos considerar que cada plaza está limitada por hardware al tamaño de elementos del vector de marcado implementado. Por lo tanto, el vector de cotas, tendrá como tamaño de elementos máximo del vector de marcado implementado, pudiendo limitar la cantidad de tokens en una plaza desde cero hasta este valor.

```
reg [tamano_de_elementos-1:0] cotas_plazas [cant_plazas-1:0];
```

Figura 80: Vector de cota de plaza implementado en Verilog

- La entrada a este módulo (inicialización) se realiza en la programación y son los valores máximos de marca que puede adquirir una plaza, estos también pueden ser reprogramados en tiempo de ejecución.
- La salida de este módulo son los valores máximos de marca que puede adquirir una plaza.
- La responsabilidad de este módulo es almacenar los valores máximos de marca que puede adquirir una plaza, para determinar que las transiciones no pueden ser disparadas por superar la cota máxima de plaza.

Tipos de transiciones de Entrada

Es un vector con dos bits por componente, con tantos elementos como transiciones |T|. Cada componente programa el comportamiento de la transición, donde: el modo "uno" indica que la cola entrega un evento del contador, en modo "dos" disparo automático y no debe esperar el evento y en modo "tres" el evento es no-perenne.

```
reg [cant_transiciones-1:0] transiciones_automaticas;
```

Figura 81: Vector de transición implementado en Verilog

Colas de Entradas

Cada transición tiene una cola de eventos de entrada. Estas colas cuentan los eventos relacionados con una transición que espera a ser ejecutada, cuando la transición esta sensibilizada. La implementación se realiza como lo muestra la Figura 82, con un contador asignado a cada transición. Este módulo es programado para proveer tres modos distintos de comportamiento de la cola, por el registro modo.

Además, existe un valor binario, que es una componente del vector del "Vector con las solicitudes de disparos", que indica si la cola de una determina transición está vacía o tiene algún valor (ver Figura 83). En la Figura 84 se indica cuando se ha sobrepasado la cuenta, en ambos casos se denota como R.

Esta implementación permite insertar o extraer un evento en un ciclo de reloj y además, evaluar todas las solicitudes de disparos en paralelo.

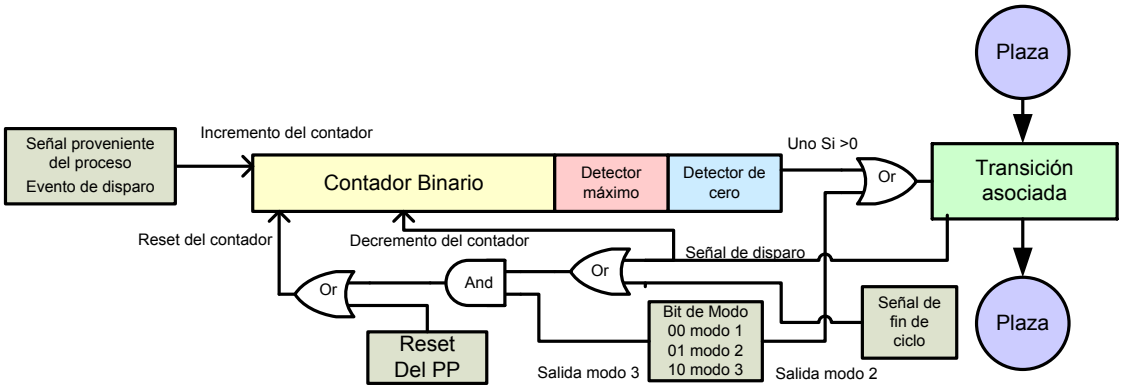


Figura 82: Diagrama en bloque de la cola de eventos de entrada

Cada bit del contador es tomado como entrada en la compuerta *or*, como lo muestra la Figura 83, lo que indica cuando es cero y no hay un evento esperando. Mientras que la Figura 84, muestra la detección de máxima cantidad de eventos, todos uno.

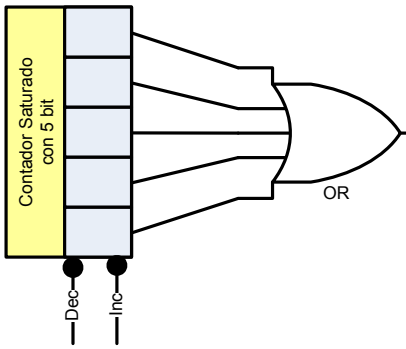


Figura 83: Valor binario que indica que no hay evento en la cola

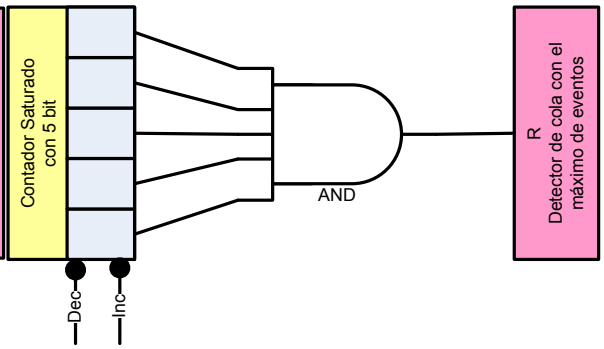


Figura 84: Valor binario que indica que la cola tiene el máximo de eventos

La implementación de estas colas se realizó en un módulo de Verilog, que se usa en las de entrada y de salida de todas las transiciones.

```
module cola_disparos
#(parameter cant_transiciones = 2,      //Cantidad de transiciones de la red de Petri modelada
            tamano_cola       = 5       //Tamano de contador del contador para entrada de disparos
)
(  input clk,
   input reset,
   input [cant_transiciones-1:0] disparo_inc_entrante,
   input [cant_transiciones-1:0] disparo_dec_entrante,
   output [cant_transiciones-1:0] registro_disparos,
   output [cant_transiciones-1:0] colas_llenas
);

   //Registros
      reg [tamano_cola-1:0] counter_cola_disparos [cant_transiciones-1:0]/* synthesis syn_keep = 1 */;


   //Operaciones sobre la cola

end module
```

Figura 85: Cola implementada en Verilog

Este módulo (cola de entrada) tiene tres modos de funcionamiento, estos modos son independientes para cada transición.

Modos de funcionamiento:

- Modo uno: modo de disparo, disparo explícito asociado con la transición. En este modo, se almacena eventos (solicitud de disparo) en una cola hasta que son resueltos por el disparo de la transición asociada.
    - La entrada a este módulo se realiza de dos formas distintas:
        - Por un proceso que genera un evento, solicita el disparo de una transición, incrementando el contador de eventos de esa transición.
        - Por el disparo de la transición asociada a la cola, esto disminuye en uno el contador de eventos.
    - La salida de este módulo un uno si hay eventos solicitando a un disparo y un cero si no los hay.
    - La responsabilidad de este módulo es almacenar los eventos que solicitan disparo, comunicárselos a la transición asociada y descontarlos cuando la transición es disparada. También lanza una excepción cuando alcanza el valor máximo de cuenta.

- Modo dos: disparo automático asociado con la transición. En este modo, se dispara la transición automáticamente cuando esta sensibilizada y no depende de los eventos de los procesos.
    - La entrada a este módulo "no tiene entrada"
    - La salida de este módulo es siempre uno.
    - La responsabilidad es permitir el disparo de la transición asociada una vez que está sensibilizada y su prioridad lo permite.
- Modo tres: disparo no-perenne asociado con la transición. En este modo, se dispara la transición si esta es sensibilizada cuando llega el evento o si en el término de un cálculo se sensibiliza la transición. Luego se borra la solicitud de disparo.
    - La entrada a este módulo es de un proceso que solicita el disparo con la transición asociada
    - La salida de este módulo es un uno mientras se almacena el evento.
    - La responsabilidad es almacenar el evento mientras la transición no es disparada, pero no más de un ciclo de cálculo del procesador.

Cola de Salida

Cada transición tiene una cola de eventos de salida, como lo muestra la Figura 86. Estas colas cuentan los disparos de la transición relacionada. La implementación se realiza con un contador asignado a cada transición. Éste se incrementa cuando la transición se dispara, se disminuye en uno cuando el evento es leído y se pone a cero con la inicialización del PP (reset). Es posible programarlo en modo no informa, cuando no se requiere leerlo.

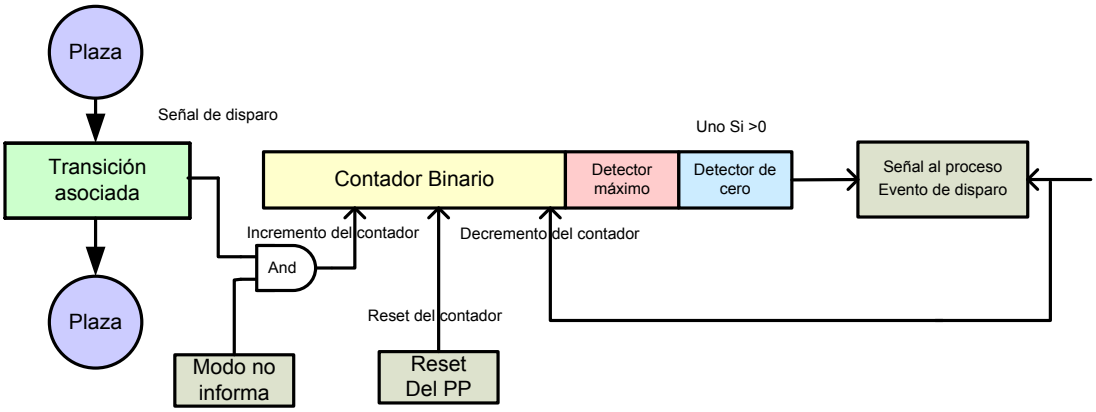


Figura 86: Diagrama en bloque de la cola de eventos de salida

Entrada de datos

La responsabilidad del módulo de entrada de datos es cargar todos los datos que programan al PP para ejecutar la RdP.

Funcionamiento del PPcT

En esta sección, se presentarán los diagramas de secuencia, las palabras de carga, el algoritmo de ejecución del PPcT, la función de implementación del algoritmo, y con el fin de exponer el funcionamiento del PP mostrando algunas secciones de la implementación en Verilog.

Carga de datos

La carga de datos, es la primera acción que se realiza, para programar el PPcT. La carga pueden realizarse en cualquier orden y las estructuras que se almacenan son: la matriz de incidencia, la matriz de inhibición, el marcado inicial, el vector de cotas de plazas, el vector de transiciones automáticas y los tiempos.

El software de inicialización, se ejecuta en un procesador MicroBlaze, para la carga de datos, sean algunos o todos los vectores y/o matrices del PP para programarlo o reprogramarlo.

La Figura 87, muestra el proceso de carga de datos, este diagrama está dividido en dos partes. Una secuencia inicial de reset y una secuencia de carga de los elementos que se requieran para programar el funcionamiento del PP.

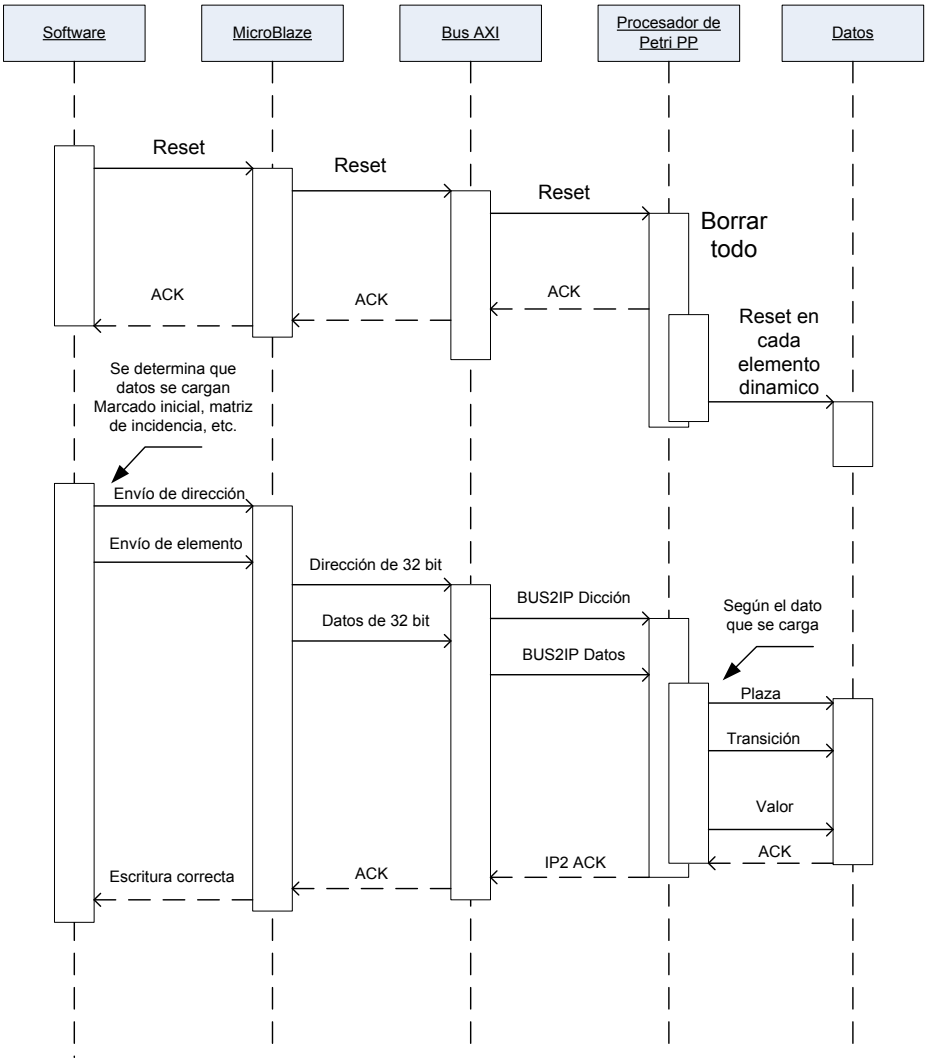


Figura 87: Diagrama de secuencia para la programación del PPcT

Secuencia de reset de datos

La secuencia de reset, se compone de las siguientes etapas.

- El procesador MicroBlaze ejecuta el reset sobre el PPcT.

- El PPcT pone a cero todas las matrices y vectores, almacenados en el PPcT.
- El MicroBlaze reprograma el PP. Secuencia de carga de matrices y vectores.
- Los procesadores MicroBlaze comienzan la ejecución, luego envían y reciben eventos del PPcT, disparando y escuchando las transiciones.

Secuencia de carga de vectores y matrices

El largo de la palabra, para la carga de vectores y matrices, es de 32 bits y ha sido definida de la siguiente manera:

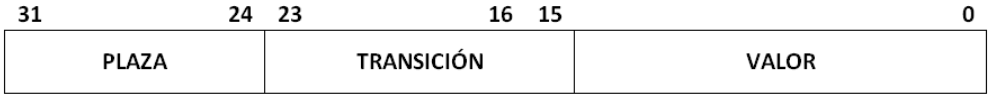


Figura 88: Campos de la palabra para carga de datos en el PPcT

La cantidad de bits, para identificar las plazas y las transiciones son 8. Esto permite una RdP de 256 plazas por 256 transiciones. Estos dos valores identifican la posición en la matriz de incidencia del valor. Mientras que los 16 bits restantes son el valor que se carga.

Procesamiento de los eventos de entrada relacionado con una transición

La palabra utilizada, es la definida en la Figura 88. En la dirección indicada por el campo transición, se indica el número de transición lo que incrementa el contador relacionado con la transición.

Algoritmo para la ejecución de RdP

Para ejecutar una RdP, se hace uso de la ecuación de estado. Para lo cual se detectan las transiciones sensibilizadas, los disparos que sean posibles de ser ejecutados y en caso de serlo, resolver la ecuación de estado de la red para lograr un nuevo marcado de la misma.

La ecuación de estado de RdP con arcos inhibidores, ha sido modificada para la ejecución y tiene la siguiente forma:

$$m_{i+1} = m_i + I \cdot (d \; and \; f(m_i, H))$$

Donde d es un vector binario de dimensión $|T|$, con uno en las componentes que tiene disparo solicitado.

La relación f, depende de: el marcado m_i y la matriz de brazos inhibidores *"H"*. Con esta relación se determina que las transiciones están inhibidas por los brazos inhibidores y la marca.

Esta relación se obtiene multiplicando a la matriz H por un vector binario de dimensión $|P|$, que tiene uno si la marca de la plaza es distinta a cero y cero si es cero. Por lo que, la relación retorna un vector binario de dimensión $|T|$. Con este vector y el vector d , se calcula una función *"and"*, esto inhibe los disparos que no son posibles por el estado y los brazos inhibidores. El resultado del paréntesis es un vector, que indica con un uno, los disparos que no han sido inhibidos por el estado y H.

El resto de la ecuación es la ecuación de estado de las RdP.

La Figura 89, muestra una posible implementación, en diagrama en bloque de la relación anterior, para determinar que transiciones son posibles de disparar.

Descripción de los bloque de la Figura 89

- Matriz de Incidencia.
- Marcado, es el estado de la RdP.
- Detector de Marca, genera un vector de dimensión |P| con un uno en la plaza que tiene marca.
- Matriz de Inhibición.
- Cota de Plaza, es un vector de enteros (8 bits sin signo).
- Matriz con el cálculo de los posibles próximos estados, en una matriz donde cada columna tiene |P| elementos con el cálculo de marcas que resultaría del disparo de la transición, ordenados según el índice de columna. Estos elementos son enteros con signo (8 bits con signo).
- Vector de signos, es binario y tiene dimensión |T|, donde cada elemento es el resultado de realizar una función *or* entre el signo de todas las marcar de un estado posible. Estos valores han sido obtenidos de la matriz con el cálculo de los posibles próximos estados. Con este cálculo, el vector resultante, tiene un uno donde el estado de esa posible marca no es posible; puesto que nos lleva a un estado con marca negativa.
- Registro de transiciones habilitadas por el signo, todos los disparos con estados alcanzables.
- Comparador de cota de plazas, compara el posible estado alcanzado por un disparo, para determinar si hace el disparo sobrepasara el límite de plaza e inhibirlo.
- Registro de transiciones con solicitud de disparo, todas las transiciones con evento de solicitud de disparo o automática.
- Registro de transiciones habilitadas por la cota de plaza, todos los disparos que no superan la cota de plaza.

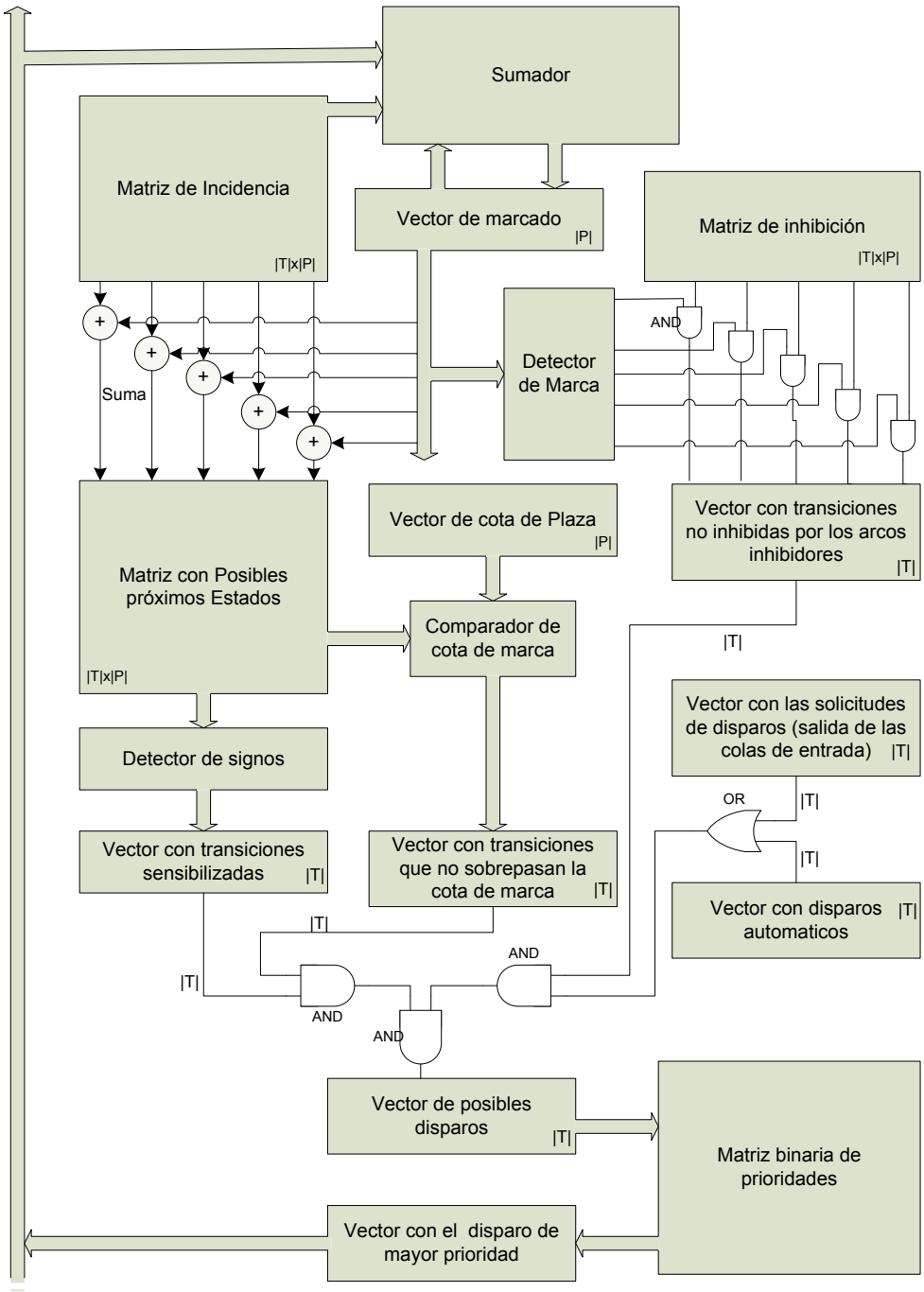


Figura 89: Diagrama en bloque del circuito para determinar disparos de transiciones posibles

Procesador de RdP con Tiempo (PPcT)

Ahora agregamos las estructuras de datos y el hardware necesarios para proveer los mecanismos para ejecutar RdP con Tiempo.

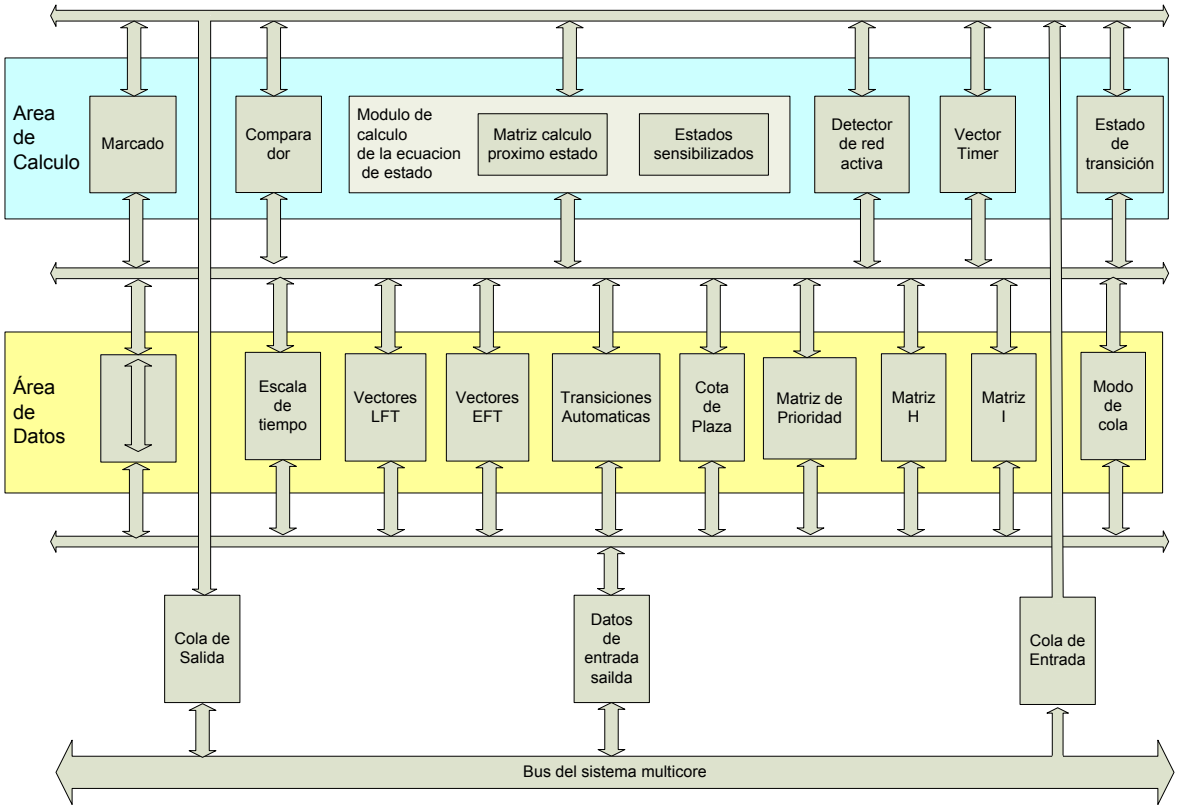


Figura 90: Diagrama de bloque del PPcT

Para esto, agregamos:

- *Vector de Earliest Firing Time (EFT)*, de dimensión |T|.
- *Vector de Latest Firing Time (LFT),* de dimensión |T|.
- *Vector de tiempo,* un registro por transición (estos registros pueden ser leídos por los procesos) para contabilizar el tiempo de sensibilidad que tiene una transición. El vector tiene dimensión |T|.
- *Comparador*, que determina si el registro de tiempo de la transición sensibilizada está en el intervalo de tiempo de disparo, hay |T| comparadores.
- *Escala de tiempo*, que consta de un registro por transición que opera como divisor de cada contador de tiempo, de dimensión |T|.

También se ha agregado un *detector de red activa*, que detecta cuando no hay ninguna transición sensibilizada. La finalidad de este registro es detectar cuando la red está bloqueada, por lo que es muy útil para los procesos de desarrollo y pruebas.

Como se aprecia en la Figura 91, la determinación de las transiciones sensibilizadas es idéntica a la implementada en el PP sin tiempo. La diferencia es que ahora se han agregado los contadores y registros de tiempo (EF y LF). Los registros de tiempo son habilitados por el registro de transiciones sensibilizadas. Las cuentas de estos contadores son comparadas con los registros de LFT y EFT, lo que habilita la ventana de tiempo para el disparo de la transición. El disparo se

realizara siempre y cuando sea requerido por su evento asociado o en su defecto automático, y según el esquema de prioridades.

Los contadores son puestos a cero cuando se realiza el disparo o cuando la transición deja de estar sensibilizada.

Aquí, al igual que en el PP sin tiempo, la salida del registro de posibles disparos es el argumento de la función de prioridades, esta función determina que transición se dispara, en esta solución sólo se dispara una transición.

Algoritmo para la ejecución de RdPcT

Arquitectura del PPTm

A continuación presentamos la arquitectura del PPTm, esta arquitectura es realizada en bloques funciones que implementan la ecuación de estado. Esta ecuación de estado ha sido ampliada con brazos inhibidores y plazas acotadas. El módulo de cálculo de la ecuación de estado es descripto en la Figura 93.

Posteriormente, se describe cada una de los bloques y su interrelación.

Módulo de Cálculo de la Ecuación de Estado

La Figura 92 muestra la arquitectura para el PPTm (cálculo de la ecuación de estado y comunicación con los procesadores), se han incluido todos los bloques para mostrar la relación entre los módulos y sus funciones.

Seguidamente, en la Figura 93 se muestra la arquitectura para el cálculo de ecuación de estado del PPTm. Se describen las entradas, salidas y responsabilidad de los módulos.

- Vector con las solicitudes de disparos
    - La entrada a este módulo son los detectores de cero de las colas de entrada
    - La salida de este módulo es una palabra binaria de tamaño |T| con un cero en las colas vacías y un uno en la cola no vacías.
    - La responsabilidad de este módulo es indicar que cola tiene eventos solicitando disparo.
- Vector con disparos automáticos
    - La entrada de este módulo se realiza en la programación y se coloca un uno en el evento que no es requerido para el disparo y un cero para los eventos requeridos, es una palabra binaria de tamaño |T|. Puede ser reprogramado en tiempo de ejecución.
    - La salida del módulo, que es un registro, mantiene los valores programados.
    - La responsabilidad del módulo es indicar que transiciones no requieren un evento para ser disparadas.
- Vector de posibles disparos
    - La entrada de este módulo es el resultado de realizar una operación *and* entre el vector con transiciones sensibilizadas, vector con transiciones que no sobrepasan la cota de marca y el resultado del cálculo *or* entre el vector con solicitudes de disparo y el vector con disparos automáticos.

- La salida del módulo es un vector binario de tamaño |T| con un uno en los bits relacionados a las transiciones en condiciones de ser disparadas y un cero en las que no se pueden disparar.
- La responsabilidad de este vector es indicar todas las transiciones en condición de disparo.

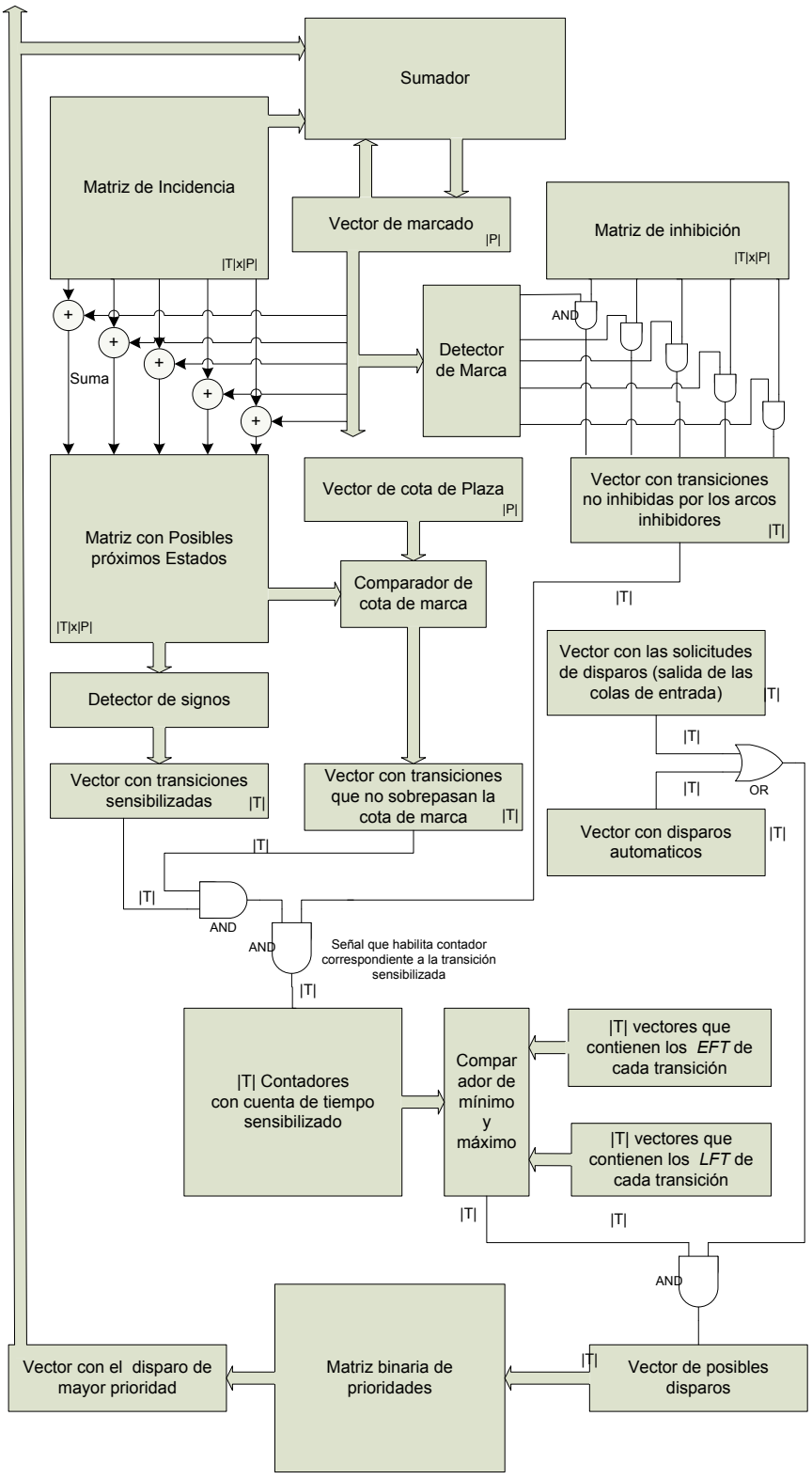


Figura 91: Diagrama en bloque del circuito para determinar que transiciones son posibles de disparar (PPcT)

- Matriz binaria de prioridades
    - La entrada de este módulo se realiza en la programación y es una matriz binaria de |T| x |T|. Puede ser reprogramado en tiempo de ejecución. Las filas de esta matriz tiene un uno sólo en la posición de prioridad de la transición.

- El producto de esta matriz por el vector de posibles disparos ordena los bits del vector, en el orden de prioridad de las transiciones. Donde el bit más significativo es el de más prioridad. Multiplicar la inversa de esta matriz por el vector, resultante del producto descripto en el ítem anterior, devuelve un vector con los bits en el orden de las transiciones.
- Determinar el disparo de más prioridad

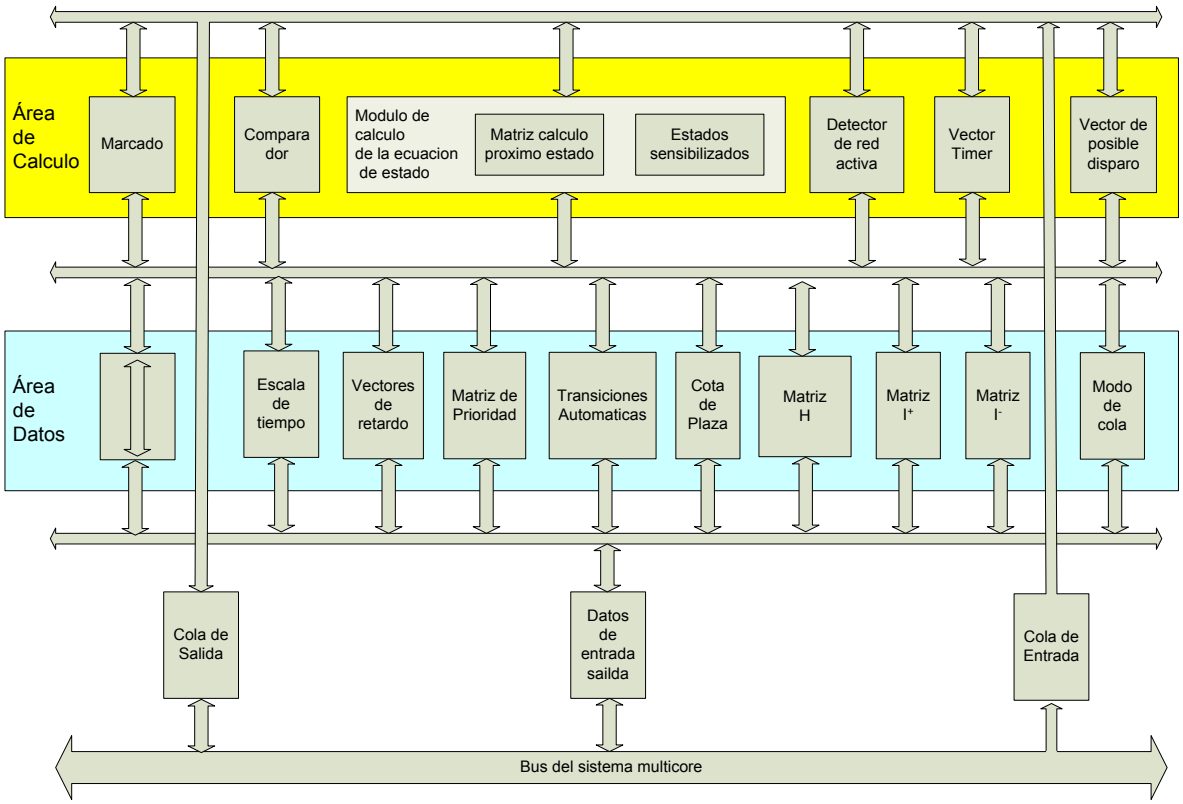


Figura 92: Arquitectura del PPTm

- |T| contadores que contienen las cuentas de cada transición disparadas
  - La entrada a cada contador es un bit de habilitación para la cuenta y una señal de reset para volver la cuenta a cero.
  - La salida son las cuenta de cada contador
  - La responsabilidad de este módulo es realizar la cuenta para los retardos de las transiciones. Cuando esta cuenta es alcanzada se disparan los token de salida y se resetea el contador de retardo.
- Vector binario, es un registro, de dimensión |T| que contiene los contadores que están activos
  - La entrada a este módulo es la salida de la matriz de prioridades con la transición a ser disparada.
  - La salida de este registro son las transiciones que están realizando el retardo.
  - La responsabilidad de este vector es mantener a los contadores contando, el bit con un uno habilita la cuenta y un cero lo resetea.

- Comparador
    - La entrada a este módulo son las salidas de las cuentas de las transiciones que están realizando el retardo.
    - La salida de este módulo es un uno cuando la cuenta es igual al retardo programado. Con esta señal se resetea el bit que habilita cuenta y se realiza la suma para calcular el nuevo estado.
    - La responsabilidad de este módulo es determinar cuándo una transición ha cumplido con el retardo.
- |T| vectores que contienen los retardos de cada transición
    - La entrada de estos vectores se realiza en la programación o en tiempo de ejecución.
    - La salida de estos registros son usados a la entrada del comparador para comparar con la cuenta del retardo.
    - La responsabilidad de este módulo es almacenar el retardo de cada transición.
- Matriz I^-
    - La entrada de esta matriz se realiza en la programación o en tiempo de ejecución.
    - La salida de esta matriz es usada para restar los token de las plazas (estado) cuando comienza el disparo.
    - La responsabilidad de este módulo es almacenar los pesos de los arcos entre las plazas y las transiciones.
- Matriz I^+
    - La entrada de esta matriz se realiza en la programación o en tiempo de ejecución.
    - La salida de esta matriz es usada para sumar los token que salen de las transiciones a las plazas (estado) cuando finaliza el retardo.
    - La responsabilidad de este módulo es almacenar los pesos de los arcos entre las transiciones y las plazas.
- Resta nuevo Estado
    - Las entradas a este módulo son el estado actual y la columna que se corresponde con la transición disparada de la matriz I^-.
    - La salida de este módulo es el valor del nuevo estado (token consumido).
    - La responsabilidad de este módulo es remover los token de las plazas que entra a la transición disparada.
- Suma nuevo Estado
    - Las entradas a este módulo son el estado actual y la columna que se corresponde con la transición disparada de la matriz I^+.
    - La salida de este módulo es el valor del nuevo estado (token generados).
    - La responsabilidad de este módulo es insertar los token de las plazas de salida de la transición disparada.

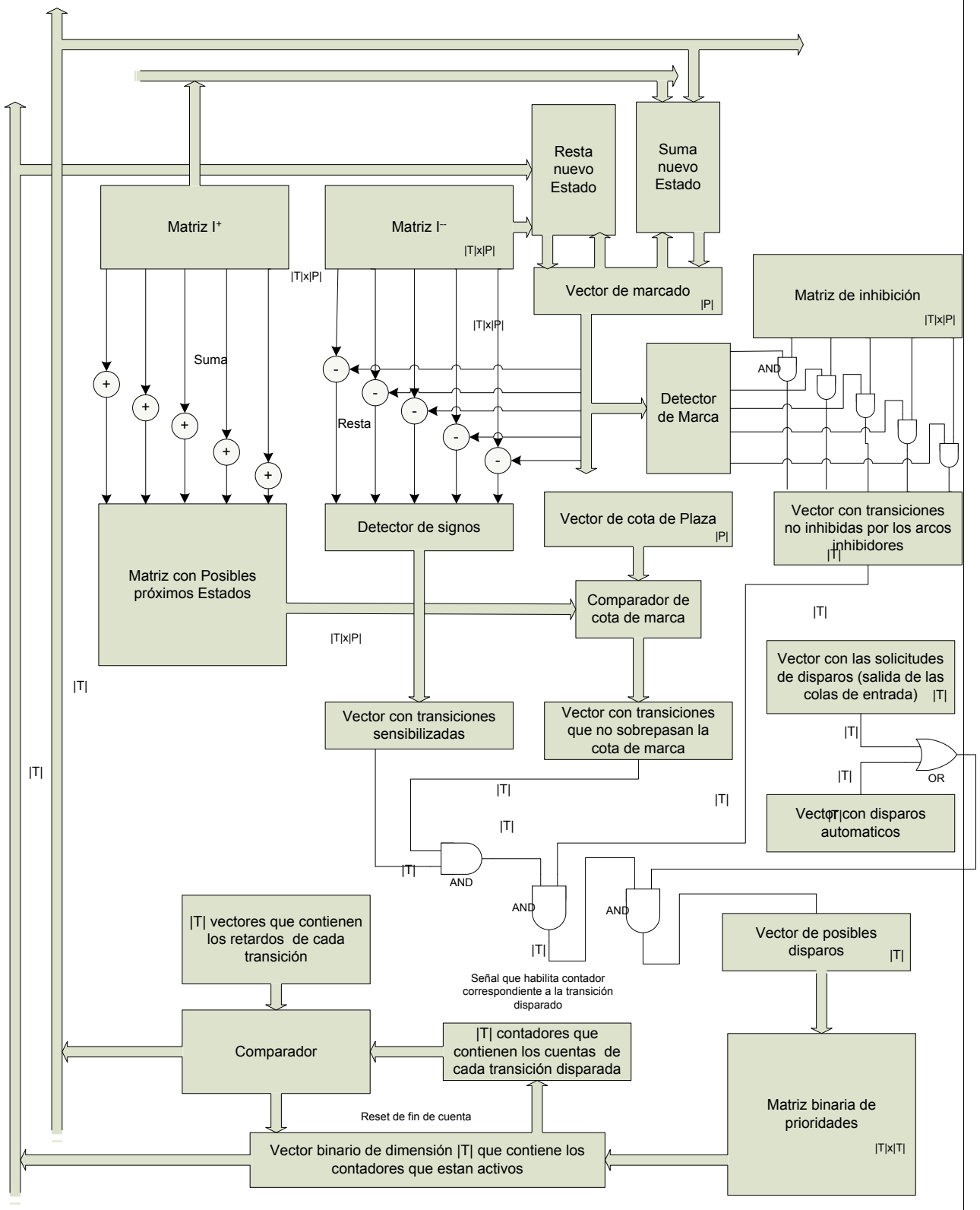


Figura 93: Arquitectura del cálculo de ecuación de estado del PPTm.

Módulo para Disparos Múltiples del PP, PPcT y PPTm

El módulo que se muestra en la Figura 94, ha sido implementado en toda la familia de PP su finalidad es resolver múltiples disparos en una operación.

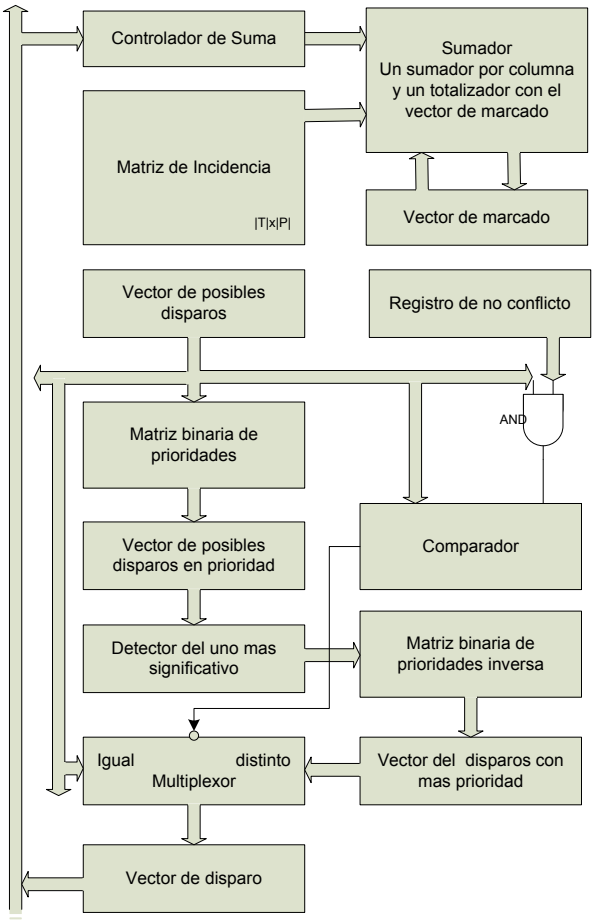


Figura 94: Módulo para disparos múltiples del PP, PPcT y PPTm

Para esto se han introducido nuevos módulos y cambios en otros, a continuación se describen las responsabilidades, entrada y salida de cada módulo nuevo o modificado.

- Registro de no conflicto
  - La entrada de este módulo se realiza en la programación del PP y puede ser modificado en tiempo de ejecución. Este vector está en el área de datos del PP.
  - La salida del módulo son un uno para las transiciones que no tienen conflicto con otra, la dimensión del vector es |T|.
  - La responsabilidad de este vector es indicar que transiciones están en conflicto. Cuando una transición candidata (de posibles disparos) está contenida en este vector, se dispara la de máxima prioridad.
- Comparador
  - Las entradas a este módulo son el vector de posibles disparos y el resultado del cálculo de la operación *and* entre el vector de posibles disparos y el registro de no conflicto.

    - La salida de esta comparación es uno si son iguales, esto indica que no hay una transición en conflicto. Si la comparación es distinta se apaga el bit de conflicto a la salida de la operación and, esto indica que hay una transición con conflicto como candidata a ser disparada.
    - La responsabilidad de este módulo es eliminar el conflicto de transiciones en el disparo de múltiples transiciones simultáneas.
- Multiplexor
    - Las entradas a este módulo son todas las transiciones con posibles disparos, la transición con más prioridad y la salida del comparador.
    - La salida de este módulo es la transición de mayor prioridad si hay conflicto o todas las transiciones posibles de ser disparadas si no hay conflicto de transición.
    - La responsabilidad de este módulo es determinar si el disparo es múltiple si no hay conflicto o solo una transición si hay conflicto.
- Vector de disparo
    - La estrada es el vector que sale del multiplexor.
    - La salida es usada en el controlador de suma, para sumar las transiciones disparadas al estado actual.
    - La responsabilidad de este registro es almacenar las transiciones a ser disparadas en un ciclo.
- Controlador de Suma
    - La entrada a este módulo es el vector de disparo.
    - La salida de este módulo es el control del sumador, se realiza una suma en cada bit con un uno de este vector. Se suman todas las columnas con un uno en el vector con el estado actual.
    - La responsabilidad de este módulo es controlar al sumador para hacer una suma selectiva.
- Sumador, un sumador por columna y un totalizador con el vector de marcado
    - La entrada a este módulo se obtiene del controlador de suma, la matriz de transición y el estado actual.
    - La salida es el nuevo estado actual
    - La responsabilidad del módulo es realizar una suma selectiva, entre las columnas de la matriz de incidencia que se corresponden con las transiciones a disparar y el estado actual. Realiza un disparo múltiple de las transiciones que cumplen con las condiciones de disparo y no están en conflicto.

Este módulo consume excesivos recursos y es implementado si es absolutamente necesario obtener disparos múltiples.

Manejo de Interrupciones

Para este requerimiento, se agrega un registro de máscara de interrupciones al PP con la lógica para generar una interrupción con el disparo de una de las transiciones que pertenezca al conjunto de transiciones que generan interrupción.

Además se especifica un vector que enmascara las interrupciones de algunos disparos entonces, el usuario puede decidir que disparos generarán interrupciones y cuáles no.

La necesidad que motiva este requerimiento, es que el PP puede requerir pocos ciclos para resolver el disparo, una cantidad de ciclos que es del orden a los requeridos para el cambio de

contexto o una cantidad muy por encima de los requeridos para el cambio de contexto. Para este último caso, el PP incorpora la generación de interrupciones.

El uso que se recomienda hacer de estas tres opciones, con el fin de minimizar los tiempos de esperas, es el siguiente:

- Si el tiempo que se requiere para el sincronismo, es menor al orden del tiempo requerido para el cambio de contacto, se consulta la cola de salida hasta que se lee el disparo.
- Si el tiempo que se requiere para el sincronismo, es del orden del tiempo requerido para el cambio de contacto:
    1. Primero se consulta la cola de salida.
    2. Si el disparo no ha sido resuelto se sede el tiempo de ejecución. Si ha sido resuelto indica que hay que continuar.
    3. Se ejecuta 2 hasta que el disparo es resuelto.
- Si el tiempo que se requiere para el sincronismo, es mucho mayor al orden del tiempo requerido para el cambio de contacto, se programa una interrupción y se bloquea el hilo/proceso hasta que ésta se produce.

Arquitectura del Generador de Interrupciones

La Figura 95 muestra el diagrama en bloques del generador de interrupciones.

La descripción de la arquitectura del PP con interrupciones es la siguiente:

- Registro Máscara de Interrupción
    - Su objetivo es determinar cuáles son las transiciones que al dispararse generaran interrupciones.
    - La entrada a este registro se realiza con la inicialización del PP y puede ser reprogramado en tiempo de ejecución. Este se encuentra en el área de datos y tiene |T| elementos binarios. Con un uno se activa la interrupción cuando se realiza el disparo y con un cero se enmascara.
    - La salida de este registro, es una palabra binaria, con la que se hace una operación *and* entre este registro y el vector de disparo para determinar si se genera la interrupción.
    - La responsabilidad de este registro es almacenar las transiciones que generan la interrupción.
- Generador de interrupciones
    - Las entrada a este módulo son el registro de máscara de interrupción y el vector de disparo de mayor prioridad.
    - La salida es un uno cuando se genera la interrupción.
    - La responsabilidad de este módulo es generar una interrupción cuando se produce el disparo de una transición que ha sido programada para generar interrupciones.
- Puerto físico para enviar la señal de interrupción al SMP
    - Para esta implementación, este módulo se encuentra conectado a un controlador de interrupciones del sistema, el AXI Interrupt Controller [236].

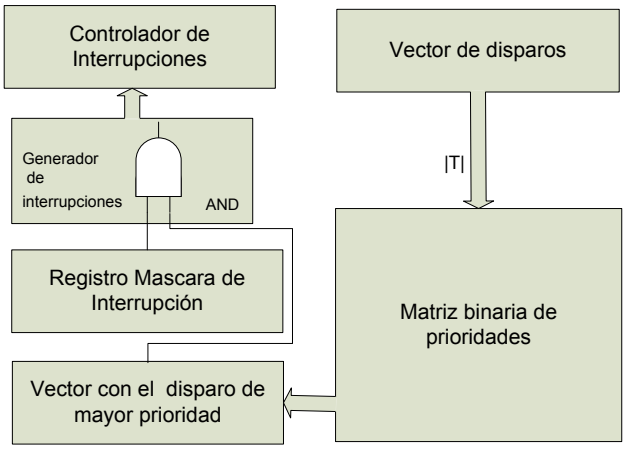


Figura 95: Diagrama en bloque del generador de interrupciones

Funcionamiento del Sistema

En la Figura 96 se muestra el código Verilog que genera las interrupciones.

Cada flanco negativo del reloj verifica si alguna de las transiciones que se han ejecutado produce interrupciones. Cuando esto sucede genera una señal en "1" de un ciclo de duración. Además, determina el índice de dicha transición, para este procesador un valor entre cero (0) y doscientos cincuenta y cinco (255).

```
/*****************************GENERADOR DE INTERRUPCIONES**********************/
    reg [1:0]Interrupt_reg;
    assign Interrupt=(!Interrupt_reg[1])&Interrupt_reg[0];
    integer columnas_intr;
    wire [cant_transiciones-1:0]intr_activadas;
    assign intr_activadas=disparos_ya_ejecutados & intr_mask;
    always@(negedge Bus2IP_Clk)
        begin
            if (Bus2IP_Resetn==1'b0)
                begin
                    num_intr<=0;
                    Interrupt_reg<=2'b00;
                end
            else
                begin
                    Interrupt_reg<={Interrupt_reg[0],(|(disparos_ya_ejecutados & intr_mask))&!intr_reset};
                    for (columnas_intr=cant_transiciones-1 ; columnas_intr>=0 ; columnas_intr=columnas_intr-1)
                        begin
                            if(intr_activadas[columnas_intr]==1'b1) num_intr<=columnas_intr;
                        end
                end
        end
```

Figura 96: Código Verilog para generar las interrupciones

Resultados Obtenidos

Crecimiento del Tamaño del IP-Core con |P| y |T|

La implementación del PPcT que se ha realizado, es parametrizable para cada instancia distinta. Los parámetros son la cantidad de plazas y transiciones máximas que se requieren, y del tamaño de cada elemento.

Para el análisis de crecimiento consideramos al número de plazas |P| y al número de transiciones |T|, con la misma cantidad de elementos. En función de esto el PPcT tendrá el siguiente tamaño.

- Cantidad de plazas |P|, variable
- Cantidad de transiciones |T|, variable
- Tamaño de elementos en bits, variable
- Cantidad de bits colas 5
- Cantidad de bits en los contadores de tiempo 48
- Cantidad de bits para la escala de tiempo 5

Con esta configuración variable de PPcT y un procesador MicroBlaze se han realizado los cálculos de crecimiento y demanda de recursos del sistema, que ha sido implementado en una kit Digilent Atlys. Para esto se han variado los parámetros y se han realizado los gráficos de las Figura 97 y Figura 98.

El kit *Atlys* de Digilent, si bien es pequeño, es adecuado para las pruebas y mediciones propuestas. Esto es, medición del área de la FPGA ocupada para diferentes valores de los parámetros propuestos en el ítem anterior, para obtener distintas instancias del PPcT. Los resultados obtenidos son los siguientes.

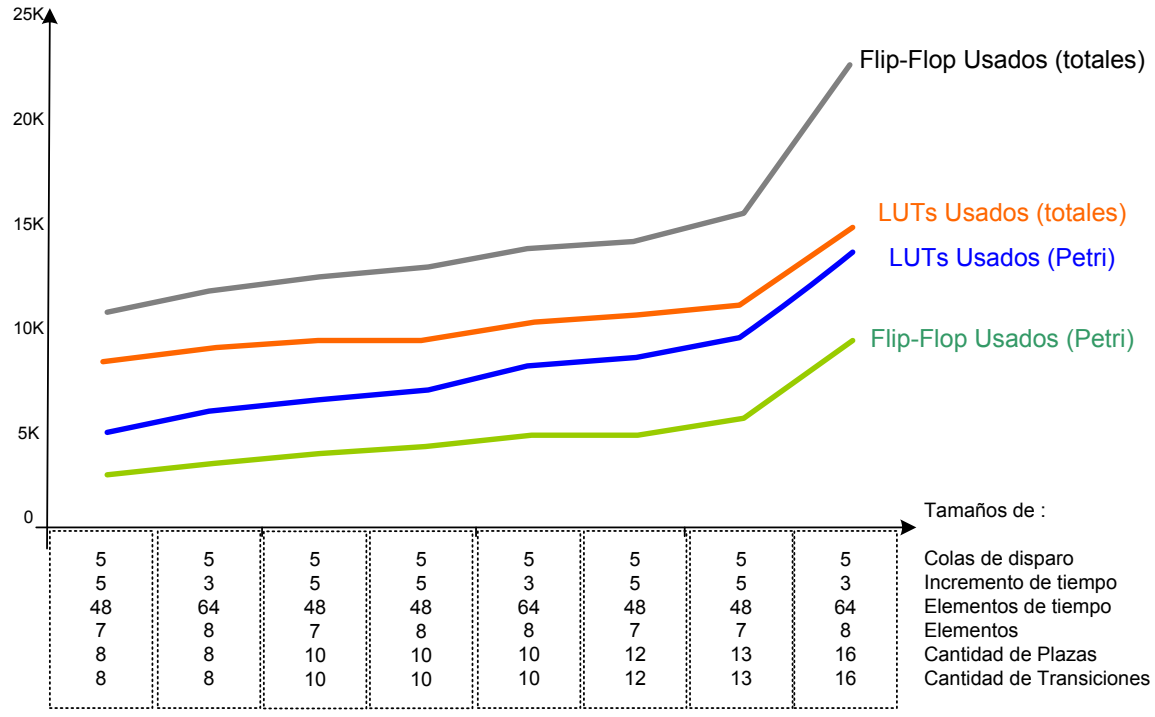


Figura 97: Diferentes instancias del PPcT en el kit Digilent Atlys

Con estos resultados, se observa que una PPcT de 16 plazas por 16 columnas, está en el límite para ser implementada con este en el kit *Atlys*, esto se debe a las siguientes razones:

- La cantidad de celdas lógicas requerida crece en el PP crece con el producto |P|x|T|.
- La FGPA incorpora un procesador MicroBlaze que consume la mayoría de los recursos.

También se realizaron las mismas pruebas con un kit Zedboard, el cual incluye un procesador ARM A9 dual core y una FPGA de mayor capacidad. Los resultados obtenidos en este caso, se muestran en la Tabla 103 y Figura 98.

Tabla 103: Diferentes instancias del PP en el kit Zedboard

Cantidad de transiciones	Cantidad de plazas	Flip-Flop utilizados Total	LUPs utilizados Total	Flip-Flop utilizados Petri	LUPs utilizados Petri
8	8	3117	5216	2683	4448
16	16	8045	15275	7611	14507
32	32	24813	39256	24374	38488
48	48	50781	85172	50347	84404
64	64	85965	128337	84431	127469

Tamaño de elemento	Tamaño de vector de incremento de tiempo	Tamaño de cola de disparo
7	48	5

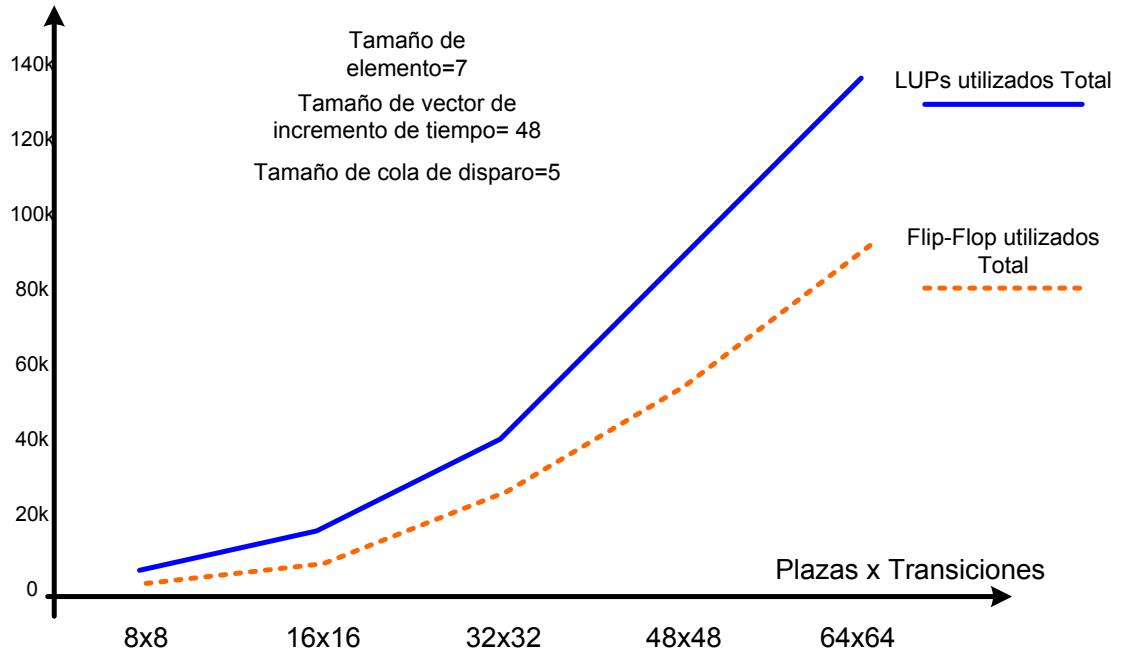


Figura 98: Diferentes instancias del PPcT en el kit Zedboard

En esta última gráfica se ve que el crecimiento está relacionado claramente con el producto |T|x|P|, ya que el procesador está incluido en el sistema y no ocupa recursos de la FPGA.

Medidas de Rendimiento

Para calcular el rendimiento del PPcT, se compara un sistema que resuelve una tarea fuertemente sincronizada con un PP cTy con semáforos, haciendo uso del mismo procesador, para este caso con un MicroBlaze.

Se mide la cantidad de pulsos de reloj (clk) necesarios para procesar los disparos y obtener la respuesta que permita sincronizar diferentes programas.

El modelo usado para realizar la pruebas es: el acceso a una variable compartida por varios hilos que intentan escribirla concurrentemente (productor consumidor).

Secuencia de Ejecución utilizando Semáforos

- El programa ha sido implementado en C, usando semáforos y la secuencia es la siguiente:
    1. Creación del semáforo.
    2. Creación de los hilos.
    3. Se arranca el timer.
    4. Cada hilo intenta solicitar el permiso por parte del semáforo. Al conseguirlo, incrementa la variable compartida. Luego, libera el semáforo.
       Esta operación se repite diez mil (10000) veces.
    5. Una vez que el hilo realizó las diez mil iteraciones, se detiene.
    6. Se detiene el timer.
    7. Se obtiene la marca de tiempo consumido.

Secuencia de Ejecución utilizando el PPcT

- El programa ha sido implementado en C, pero se utiliza el PPcT para la sincronización y la secuencia es la siguiente:
    1. Carga del marcado inicial, la matriz de incidencia, matriz de inhibición, vector de cotas de plazas, vector de transiciones automáticas, vector EFT (Earliest Firing Time), vector de incrementos de tiempo, vector LFT (Latest Firing Time).
    2. Creación de los hilos.
    3. Se arranca el timer
    4. Cada hilo solicita el permiso para escribir la variable al PPcT por medio del disparo de la transición. Luego, espera hasta que su solicitud haya sido ejecutada. Entonces incrementa la variable compartida. Luego, solicita el disparo de otra transición para devolver el token.
       Esta operación se repite diez mil (10000) veces.
    5. Una vez que el hilo realizó las diez mil iteraciones, se detiene.
    6. Se detiene el timer.
    7. Se obtiene la marca de tiempo consumido.

Para realizar las mediciones de tiempo se usó ***"axi_timer"***

Mediciones Realizadas

Estas secuencias, se han implementaron con dos (2), tres (3), cuatro (4) y cinco (5) hilos escritores. Los valores obtenidos son los de la Tabla 104.

Tabla 104: Mediciones con dos (2), tres (3), cuatro (4) y cinco (5) hilos escritores

Ejecuciones	Escritor/Escritor	Escritor/Escritor	Escritor/Escritor	Escritor/Escritor

	2 hilos		3 hilos		4 hilos		5 hilos	
	Petri	Semáforo	Petri	Semáforo	Petri	Semáforo	Petri	Semáforo
10k	5535766	29053589	7831344	40617225	10126872	68686933	12422422	88995594
20k	5535787	29053568	7831321	40617219	10126870	68686984	12422428	88995576
30k	5535774	29053576	7831344	40617232	10126875	68686983	12422417	88995582
40k	5535774	29053584	7831325	40617245	10126886	68686985	12422433	88995587
50k	5535761	29053582	7831321	40617454	10126870	68686984	12422438	88995579
60k	5535785	29053587	7831331	40617236	10126883	68686990	12422432	88995965
70k	5535754	29053573	7831325	40617226	10126892	68686997	12422421	88995598
80k	5535760	29053573	7831321	40617231	10126866	73621638	12422417	88995570
90k	5535768	29053567	7831330	45325461	10126870	68686988	12422421	88995577
100k	5535751	29053587	7831330	55850641	10126881	68686995	12422432	88995958
Valor promedio en cuentas	5535768	29053579	7831329	42611417	10126877	69180448	12422426	88995658
Speed-up por uso de Petri	5,2		5,4		6,8		7,2	
Valor promedio en milisegundos	73,8	387,4	104,4	549,4	135	924	165,6	1195

La Figura 99 muestra los valores promedio en cuenta de ciclos que se obtiene usando el PPcT y semáforos. Mientras que la Figura 100 hace las misma comparación pero en milisegundos.

La Figura 101 muestra la mejora al utilizar el PPcT en vez de semáforos y el aumento que se obtiene con el aumento de la cantidad de hilos. Esto se debe principalmente a que no aumenta la sobrecarga en el PP por el aumento de hilos, éste siempre resuelve la sincronización en dos ciclos.

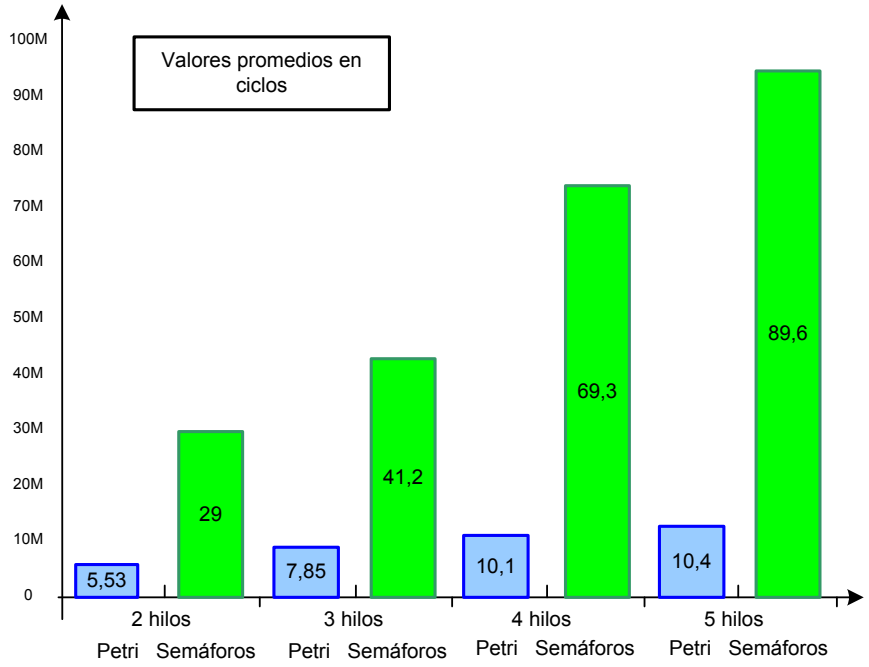


Figura 99: Valor promedio de cuentas en ciclos usando el PPcT y semáforos

Al aumentar la cantidad de hilos, el procesador de PPcT, ve limitado su rendimiento por el uso compartido del bus, en este caso el bus AXI. Pero, el problema es el mismo para acceder a memoria que es lo que hacen los semáforos, lo que se complica con los fallos de caché, etc.

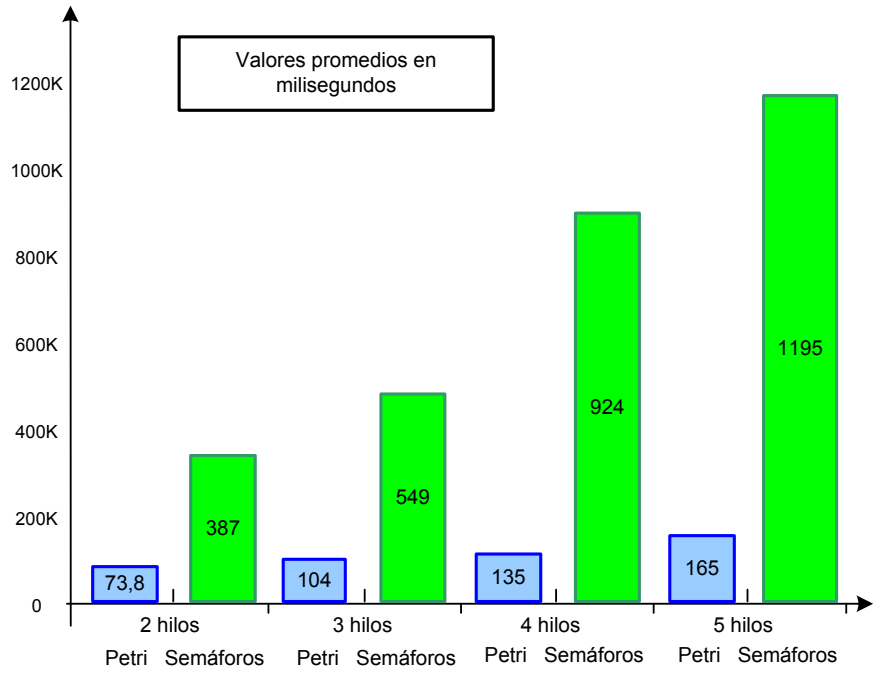


Figura 100: Valor promedio de cuentas en milisegundos usando el PPcT y semáforos

La existencia de más hilos intentando utilizar el procesador de PPcT produce una espera cada vez mayor para obtener el bus y esto, genera que el ritmo de crecimiento de la mejora sea menor.

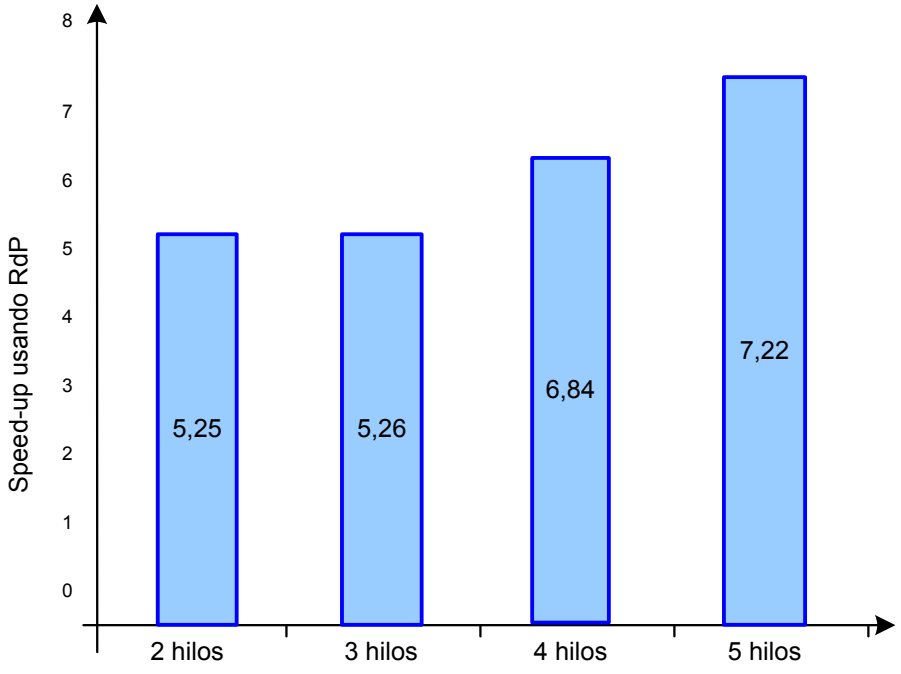


Figura 101: Mejora obtenida utilizando el PPcT frente a semáforos

La Figura 102 muestra como crece el tiempo de ejecución al aumentar el número de hilos. En el caso de los semáforos se observa una curva de crecimiento cuadrático. En cambio, en el PPcT este crecimiento es casi lineal.

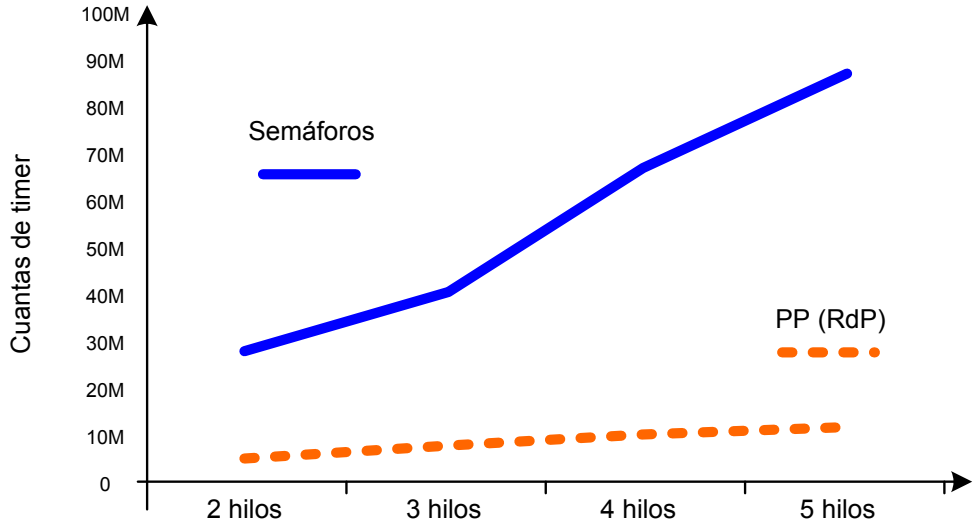


Figura 102: Tiempo de sincronización en función del número de hilos

Arquitectura del PPcT con otros Brazos

Aquí podemos destacar que, según como operan, hay tres tipos de brazos, que son:

1. Brazos que inhiben el disparo ya sea por brazos inhibidores o de lectura.
2. Brazos que detienen la cuenta de tiempo de una transición sensibilizada.
3. Brazos de reset, que retiran todos los toquen de una transición.

Para implementar todos estos brazos en el procesador se requieren una matriz por cada relación, como se lo muestra en la Figura 103. Al resultado obtenido con cada relación se hacen las operaciones: en el caso uno con el vector de transiciones sensibilizadas, el caso dos con el vector que habilita la cuenta de los contadores de tiempo y en el último implementa un módulo que pone a cero la marca de la plaza que se desea resetear.

A continuación se describen las responsabilidades, entrada y salida de cada módulo nuevo o modificado.

- Matriz con brazos de lectura
  - La entrada de este módulo se realiza en la programación del PP y puede ser modificado en tiempo de ejecución. Esta matriz está en el área de datos del PP.
  - La salida de este módulo es la matriz para inhibir un disparo por no haber token en la plaza, no se consume el token.
  - La responsabilidad de esta matriz es indicar que transiciones estan inhibidas por no marca.
- Vector con transiciones inhibidas por los arcos de lectura
  - La entrada a este módulo es un and entre cada columna de la matriz y la salida del detector de marca.
  - La salida es un uno en cada componente del vector que tiene que inhibir el disparo de una transición por una no marca.

- La responsabilidad es calcular las transiciones que no se disparan por una no marca.

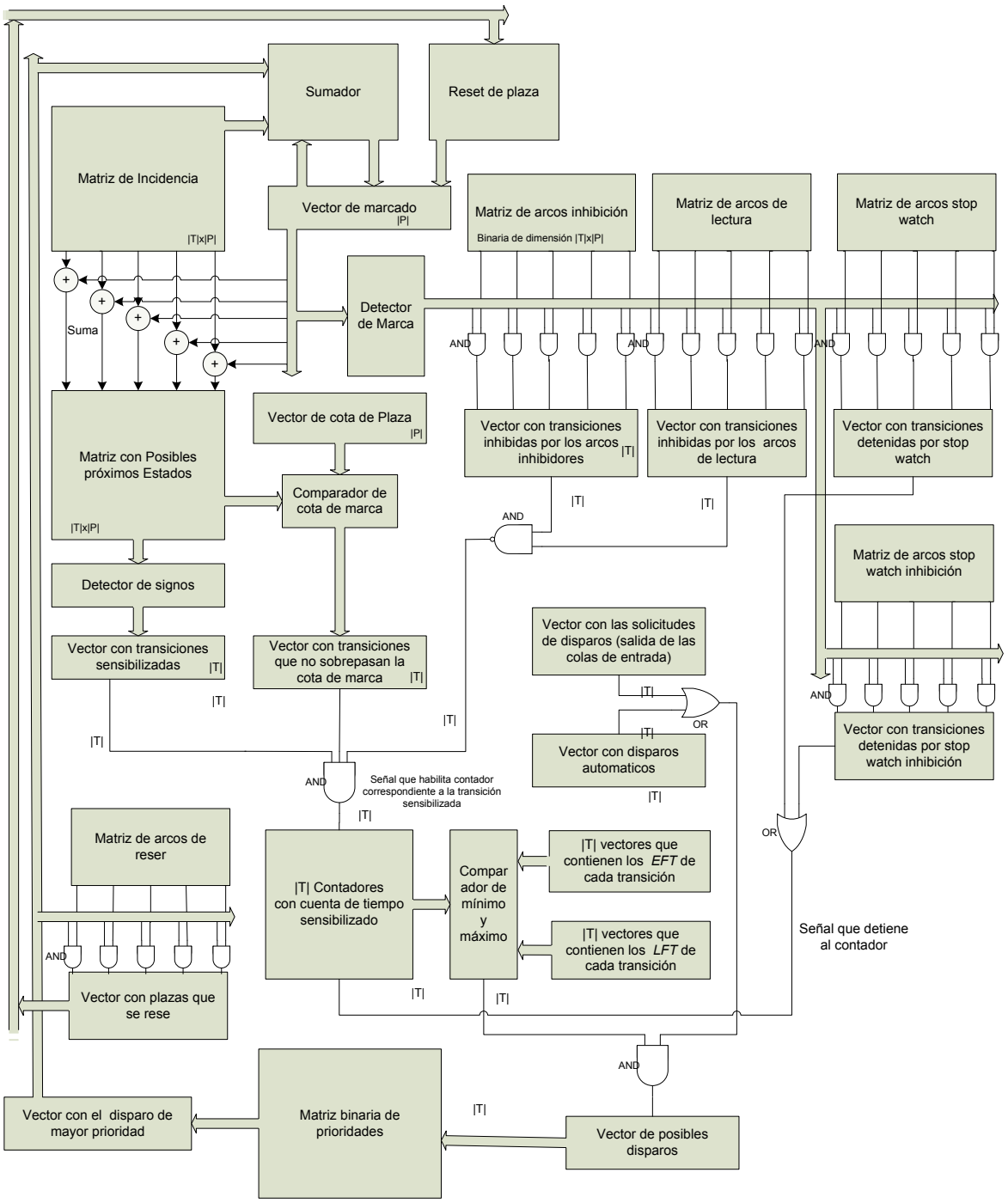


Figura 103: Arquitectura del PPcT con todos los brazos

- Matriz de arcos stop watch
  - La entrada de este módulo se realiza en la programación del PPcT y puede ser modificado en tiempo de ejecución. Esta matriz está en el área de datos del PPcT.
  - La salida del módulo la matriz que inhibe la cuenta por no haber token en la plaza, no se consume el token.

    - La responsabilidad de esta matriz es indicar que transición sensibilizada tiene la cuenta detenida por marca.
- Vector con transiciones detenidas por stop watch
    - La entrada a este módulo es un and entre cada columna de la matriz de arcos stop watch y la salida del detector de marca.
    - La salida es un uno en cada componente del vector que tiene que detener la cuenta de una transición por una marca.
    - La responsabilidad es calcular las transiciones que detiene la cuenta una marca.
- Matriz de arcos stop watch inhibición
    - La entrada de este módulo se realiza en la programación del PPcT y puede ser modificado en tiempo de ejecución. Esta matriz está en el área de datos del PPcT.
    - La salida del módulo es la matriz que inhibe la cuenta por no haber token en la plaza, no se consume el token.
    - La responsabilidad de esta matriz es indicar que transición sensibilizada tiene la cuenta detenida por no marca.
- Vector con transiciones detenidas por stop watch inhibición
    - La entrada a este módulo es un and entre cada columna de la matriz de arcos stop watch inhibición y la salida del detector de marca.
    - La salida es un uno en cada componente del vector que tiene que detener la cuenta de una transición por una no marca.
    - La responsabilidad es calcular las transiciones que detiene la cuenta una no marca.
- Matriz de arcos reset
    - La entrada de este módulo se realiza en la programación del PPcT y puede ser modificado en tiempo de ejecución. Esta matriz está en el área de datos del PPcT.
    - La salida del módulo matriz reset son las transiciones que se resetean si se dispara.
    - La responsabilidad de esta matriz es indicar que transiciones sensibilizadas resetean a la plaza si esta es disparada.
- Vector con plazas que se reset
    - La entrada a este módulo es un and entre cada columna de la Matriz de arcos reset y la salida de las transiciones que se disparan.
    - La salida es un uno en cada componente del vector que tiene que reset una marca.
    - La responsabilidad es indicar que plazas se ponen a cero por reset
- Reset de Plaza
    - La entrada a este módulo es un vector que indica que plaza se reset.
    - La salida es la plaza puesta a cero.
    - La responsabilidad es poner a cero la plaza indicada por el v dector de reset de plaza.

Resultados y Conclusiones

Se ha diseñado, implementado y probado un PPcT y un PPTm que son capaces de:

- Proveer sincronización entre los múltiples procesos con ejecución concurrente en los sistemas SMP. Se ha verificado, para los problemas planteados, que el uso del PPcT permite obtener tiempos de sincronización de hasta 7 (siete) veces más rápido que el uso de semáforos.

La programación del PPcT y PPTm es directa, ya que solo hay que cargar los vectores y matrices del modelo realizado con RdP, en el procesador. Esto significa que es posible modelar un problema con una RdP, matemáticamente validado y verificadas las propiedades y luego ejecutar la parte paralela del programa sin compilar o escribir el código.Asegurando que todas las propiedades y funcionalidades que se cumplen en el modelo, se siguen cumpliendo en la implementación.

Los inconvenientes encontrados son el crecimiento exponencial del procesador con las plazas y con las transiciones. Y la dificultad de distribuir los módulos en la FPGA.

Para abordar estos inconvenientes en el próximo capítulo se desarrolla un PP jerárquico (HPP).

Capítulo 6

Procesador de Petri Jerárquico

Resumen

En capítulos anteriores se ha diseñado, implementado y evaluado un PPcT y un PPTm, pero la dificultad que se encontró es que no se pueden implementar, en una Spartan 3, redes más grandes que 32x32 (plazas por transiciones). Esta limitación, que es debida a la cantidad de recursos necesarios para implementar el algoritmo de Petri en hardware. Todo esto nos motiva a buscar una forma alternativa de expresar el algoritmo de Petri, de manera tal que nos permita construir un PP donde se reduzca la cantidad de recursos necesarios, posibilitando la programación de sistemas con más plazas y/o transición, es decir un mayor |T|x|P|.

Objetivos

Anteriormente se mostró que es posible ejecutar las RdP para controlar la lógica paralela de los sistemas, y además se crearon procesadores que permiten dicha ejecución por hardware. Los sistemas que se pueden controlar a través de este método, están limitados por los recursos de hardware necesarios para la implementación de los procesadores de RdP, es por eso que para este proyecto nos planteamos los siguientes objetivos.

Como objetivo general para el presente capítulo nos proponemos encontrar un algoritmo alternativo que permita la ejecución de las RdP, con implementación por hardware, donde los recursos necesarios sean menores a los empleados en PP y que modele problemas con más plazas y/o transiciones que la red sin dividir.

Objetivos Secundarios

1. Estudiar las distintas maneras de expresar las RdP manteniendo su semántica.
2. Mantener las características implementadas en los PP precedentes.
3. Mantener la programación del PP por vectores y matrices obtenidas del modelo

Desarrollo

Análisis para la Implementación

En los estudios realizados, para las implementaciones de los PP, se ha determinado que los recursos del sistema aumenta con el producto de la cantidad de plazas por la cantidad de transiciones (|T|x|P|), es decir con el tamaño de la matriz.

También se ha observado, que las matrices tienen mayoritariamente elementos que son ceros, esto es debido a que la matriz tiene la capacidad de relacionar cualquier plaza con todas las transiciones y viceversa. También se observa que los unos y menos uno son la otra gran mayoría de elementos de la matriz. Y minoritariamente tiene valores enteros positivos y negativos.

Con el objeto de disminuir la cantidad de recursos se propone una arquitectura de múltiples PP, donde no se repitan las transiciones y/o las plazas, y estén interrelacionados por un sistema que introduzca una sobrecarga menor que la cantidad de elementos que se han ahorrado por la división.

Esto nos permite especializar cada PP, según el tipo de brazos y transiciones, para implementar cada PP según los recursos que se requieran.

Para lograr esto es necesario dividir la red en subredes con la menor cantidad de restricciones posibles. Esto nos conduce a explorar los distintos sistemas de RdP Jerárquicas. Con el objetivo de encontrar como dividir las RdP en un conjunto de RdP donde no se repitan la P y/o T, y que permita interrelacionarlas para constituir una RdP jerárquica equivalente a la original.

En los estudios, para obtener de una RdP una RdP jerárquica, realizados por Kurt Jensen [92] , se señala que una RdP jerárquica conlleva a una organización lógica en bloques para esclarecer el modelo, pero no se propone reducir los recursos necesarios para expresar el modelo y menos aún para ejecutarlo.

El fundamento para disminuir los recursos usando una RdP jerárquica se sustenta en mantener el total de plazas y transiciones del sistema resultante. Supongamos que tenemos una red de 100 plazas y 100 transiciones y logramos dos RdP que conforman una RdP jerárquica equivalente de 50 plazas y 50 transiciones. En el primer caso se requieren 10000 recursos (100x100) y en el segundo caso se requieren 5000 (2x50x50), se ha obtenido un 50% de disminución. Es evidente que habrá una sobrecarga para relacionar a las dos redes que componen la RdP jerárquica.

Por lo dicho anteriormente, nuestro primer objetivo será cómo realizar la división que disminuya la cantidad de recursos y cómo mantener la semántica de la RdP jerárquica resultante. Finalmente trataremos de hacer coincidir cada sub-red con distintos procesadores, con el fin de facilitar la distribución del sistema en la FPGA.

A continuación se detallan distintos enfoques y soluciones para alcanzar el objetivo planteado. Para esto, nos proponemos la división de una RdP simple, que es la de la Figura 104, y posteriormente lo extenderemos a casos más complejos.

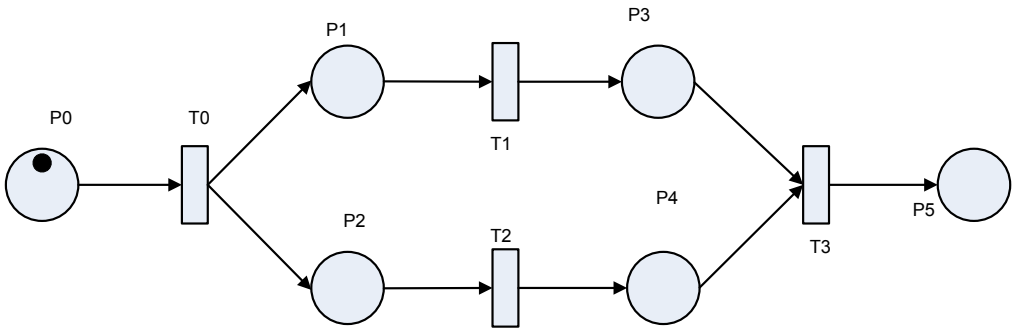


Figura 104: RdP simple, que es tomada como para la división

Esta RdP es la misma que toma Kurt Jensen para realizar el análisis de las RdP jerárquicas.

Implementación basada en plazas puerto y plazas socket

En la alternativa presentada por Kurt Jensen, se toman como límite entre las subredes a las transiciones. Una vez realizada la subred, duplicamos en ésta las plazas de la red global con las que las transiciones de borde tienen relación. En las Figura 105 y Figura 106 se muestra como han sido duplicadas las plazas. Las plazas de la red global que tienen relación con las transiciones de borde, se las denomina plazas socket y las plazas duplicadas internas a la subred se denominan plazas puerto. Luego se establece una relación entre cada plaza socket y su plaza puerto, que ha sido duplicada en la subred. Esta relación implica que ambas plazas mantienen un mismo valor de

marca, como si fueran variables globales del sistema. Por lo cual, si en la red global se modifica una plaza socket, la plazas puerto relacionadas se modifican para mantenerse igual. Lo mismo sucede si se modifica una plaza puerto en una subred, por lo que la plaza socket relacionada adquiere el mismo valor. En la Figura 105 y Figura 106, las plazas p1, p2 y p3 han sido duplicadas y existen en la red global y en la sub-red, por lo que ambas instancias de cada plaza tienen el mismo valor. Esta relación, entre las plazas socket y puerto, nos permite relacionar las sub redes con la red global. Por lo que se mantiene la consistencia del sistema como si la división no existiera.

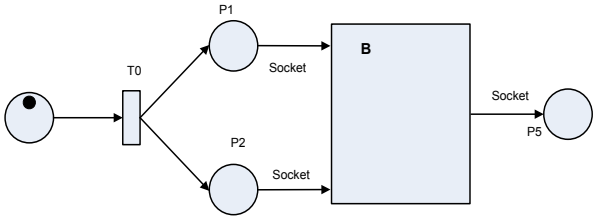


Figura 105: Red global con plazas socket

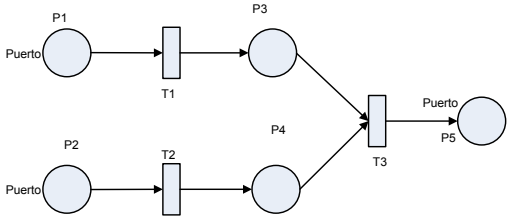


Figura 106: Subred con plazas puerto

Para implementar el sistema, compuesto por la red global y las subredes, en hardware se han considerado tres alternativas, que son:

- Para mantener actualizado el valor de ambas plazas se cablean las plazas socket y plazas puerto, que se corresponden con la misma plaza del sistema original. Para lograr esto se requiere de una compleja red de multiplexores que conecten cada elemento de cada marcado con cada elemento de cada uno de los demás marcados, y mediante palabras de configuración se controlan estos multiplexores. Es decir que por cada plaza de cada subred y de la red global se debería programar una palabra de configuración, para indicar si es una plaza socket, puerto o común, y con qué plaza y de qué otra red está relacionada. Numerosos problemas surgen para integrar las distintas subredes con la red global, lo que hace complicada la implementación, generando sobrecarga en la cantidad de recursos y ciclos para la ejecución.
- Para mantener actualizado el valor de ambas plazas se relacionan como registros separados. Cuando se produce una actualización de uno de estos registros, se realiza la misma actualización en los otros registros relacionados. Para esto se inicializan las plazas relacionadas con la misma palabra, y se implementa un sistema de actualizaciones de los marcados. También debemos considerar el caso de que dos subredes estén relacionadas a través de un socket. Las plazas estarán repetida tres veces en el sistema total: una vez como socket en la red global, y dos veces como puerto, una en cada subred. Esto implica relaciones que no son biunívocas, haciendo que se deban realizar comprobaciones en cascada, siendo el número de estas comprobaciones dependiente de las relaciones. Esto complica la implementación por hardware, generando sobrecarga en los recursos y en la cantidad de ciclos para la ejecución.
- Otra solución contemplada, es mantener un número predefinido de plazas globales que no pertenezcan a ninguna subred ni a la red global. Esto requiere hacer una distinción entre plazas locales y las globales en las matrices de incidencia, resultado,

etc. También implementar un actualizado por separado de dichas variables, complicando el hardware a desarrollar y aumentando la cantidad de ciclos necesarios para el cálculo.

Implementación basada en marcas de la matriz de incidencia
Para dividir las redes, se usan transiciones que sólo salen de la red global y entran en la subred y transiciones que sólo salen de la subred y entran a la red global. De esta forma no se produce el duplicado de ninguna plaza o transición. Para esto se requiere de dos tipos de transiciones de borde, cuya responsabilidades son:

- Transiciones globales, sólo quitan tokens de la red global y los coloca en una subred, por lo que debe ser indicada de qué transición se trata y cúal es lugar al que se conecta.
- Transiciones de la subred, sólo quitan tokens y los depositan en la red global, por lo que debe ser indicada de qué transición se trata y cúal es lugar al que se conecta.

De lo expuesto inferimos que hay dos tipos de transiciones: transiciones comunes o locales que quitan tokens de la red en que se ejecutan y depositan los tokens en la misma red y transiciones de borde que quitan tokens de la red en que se ejecutan, pero los depositan en otra red y lugar a especificar.

Para realizar la implementación por hardware, se requiere agregar la información del tipo de transición, común o de borde. Si es de borde, se especifica cuantos tokens se ponen, en que subred y en qué plaza de la subred. Todo esto requiere de gran cantidad de registros adicionales, aumentando los recursos de hardware para el HPP. También son necesarios muchos ciclos para realizar las actualizaciones de los marcados, debido a que no sólo hay que actualizar concurrentemente el marcado local sino que también el de otras redes.

Implementación basada en transiciones duplicadas
Para esta implementación se duplican las transiciones de borde, que son las responsables de relacionar las distintas subredes con la red global. Son las que tienen la información, en cada red, de cúando se quita o cúando se pone un token. También es necesario especificar la relación entre los pares de transición, entre cada duplicado con su par. Esto es una relación entre cada transición de cada red, con cada transición de cada una de las demás redes. Esto permite determinar que transición se dispara en una subred, cúando se dispara su transición relacionada en otra. De esta forma se mantiene la consistencia del sistema.

La implementación es realizada con una cola de transiciones, que está relacionada con otra, por cada red. Cuando se dispara una transición relacionada, se determina la transición y la red relacionadas, para encolar el disparo de la red correspondiente. Esta solución requiere de múltiples ciclos de reloj y se complica con las transiciones en conflictos. También se complica la determinación del estado global de la red, por lo que el sistema pasa a ser un conjunto de redes relacionadas a través de colas.

Esto hace que la implementación en hardware sea costosa en recursos. Ya que se requiere guardar la relación entre todas las transiciones de todas las redes y la implementación de las colas de transiciones demandando gran cantidad de multiplexores, puesto que es necesario encolar los disparos correctos que dependen de las relaciones.

Estas divisiones están enfocadas en lograr redes más simples de entender, pero no es su objetivo implementar la RdP en hardware, lo que nos motiva a realizar el análisis de la siguiente sección.

División de RdP

La investigación bibliográfica del análisis de división de RdP ha sido realizada en el Capítulo 2 de esta tesis y fundamentalmente éstas han sido realizadas por [134] [56] [190].

Si consideramos que las RdP están constituidas por plazas, transiciones y brazos, a los cuales llamamos elementos. Donde, estos elementos participan de la matriz de la siguiente manera: las transiciones como la cantidad de columnas, las plazas como la cantidad de filas y los brazos son el valores que relaciona una plaza con una columna según un peso y una dirección.

En las RdP vemos que estos elementos se agrupan según subconjuntos, caracterizados por estar fuertemente relacionados entre sí. Es decir, que hay brazos relacionando plazas con transiciones y viceversa. A su vez, estos subconjuntos se relacionan con otros de una manera débil, es decir que hay pocos brazos que relaciona a ambos sub conjuntos. Por esto vemos a la matriz de incidencia, como un caso de "matriz rala" [237] [238] [239].

Como podemos ver en la bibliografía, los algoritmos para resolver matrices ralas no son aplicables a este caso, por los requerimientos impuestos para el desarrollo del PP. Los algoritmos evaluados usan técnicas de compactación y/o punteros para direccionar los elementos y/o no usan operaciones lógicas simples; todo esto demanda recursos y ciclos de máquina, lo que se traduce en excesiva sobrecarga.

Esto nos lleva a definir un nuevo concepto de "**RdP jerárquica con localidad estructural**", esto es: *"dividir las redes por las transiciones agrupando elementos en los subconjuntos con características comunes en cada subred"*. Estos elementos son: arcos con peso uno, arcos con pesos mayor a uno, transiciones con tiempo, transiciones temporizadas y distintos tipos de arcos.

Teniendo en cuenta lo dicho, proponemos reducir la cantidad de recursos en el HPP puesto que cada PP del HPP tendrá sólo los recursos necesarios para ejecutar la subred.

Como se señaló anteriormente, motiva esta división suponer que la red tiene M plazas y N transiciones y es posible dividirla en dos redes, donde cada una tienen la misma cantidad de plazas y transiciones, lo que resulta en:

- Para el sistema sin dividir $\boldsymbol{M * N}$ elementos en la matriz.
- Para el sistema dividido $\boldsymbol{2((M/2) * (N/2)) = (1/2)M * N}$, más una sobrecarga que especifica las relaciones entre las subredes.

Hacemos notar que la cantidad total de plazas y transiciones del sistema se mantiene, por lo que en principio es posible mantener su representación, siempre y cuando seamos capaces de determinar cómo relacionar las subredes. También, la sobrecarga que especifica las relaciones entre las subredes tiene que ser menor a $(1/2)M * N$. Esta diferencia se traducirá en la reducción de recursos. Para este caso, en que la red ha sido dividida en dos, la ganancia máxima es de dos.

Podemos considerar que si se divide la red en J partes, la ganancia máxima teórica sería función de J, menos el costo de interconexión. También nos proponemos determinar la ganancia máxima teórica.

Alternativas para Dividir una RdP

Puesto que nuestro objetivo es dividir a la red para disminuir la cantidad de recursos del PP, podemos considerar siete alternativas para la división. A continuación se muestran estas alternativas y sus implicancias.

División de la red por las plazas externas compartidas. La Figura 107 muestra como cada subred comparte su estado compartiendo una plaza que no pertenece a ninguna subred, esto permite que una misma plaza sea compartida por varias subredes. Para realizar la relación es necesario compartir, por cada brazo, una variable entera, cómo está relacionada y si se escribe o lee.

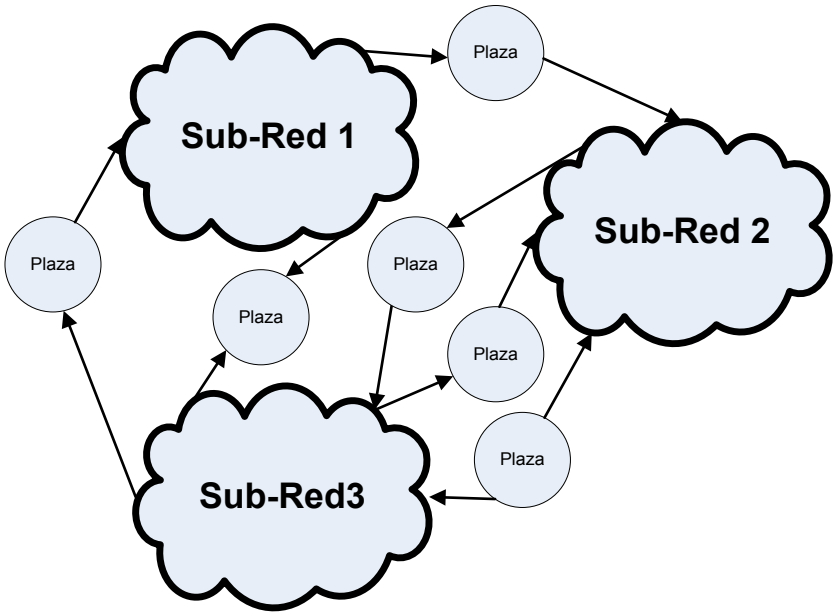


Figura 107: División de RdP, por plaza externa a las subredes. Alternativa 1

División de la red por las transiciones externas compartidas. La Figura 108 muestra como cada subred comparte algunas entradas y/o salidas de token, haciendo uso de transiciones externas a la red, esto permite que una misma transición sea compartida por varias subredes. Para realizar la relación es necesario compartir, por cada brazo, una variable binaria, cómo está relacionada, si se escribe o lee y los valores de marca de la plaza.

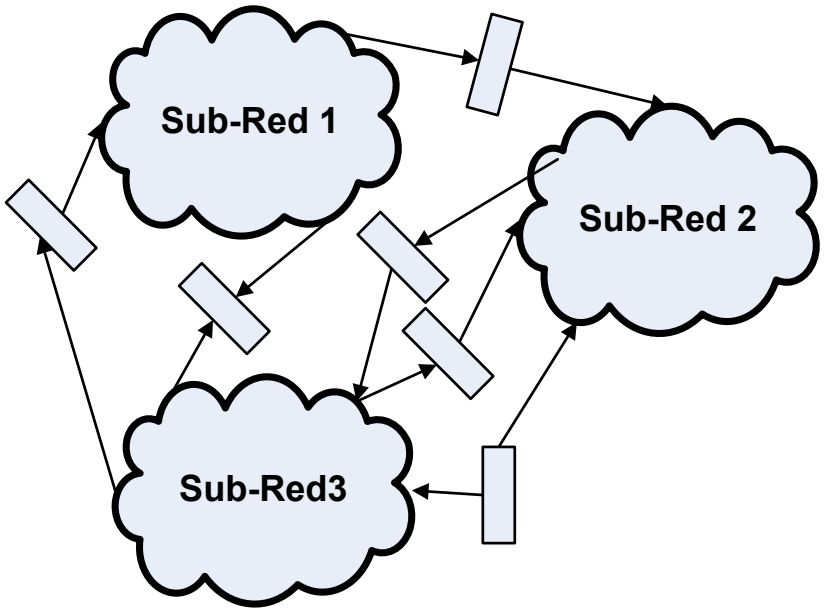


Figura 108: División de RdP, por transición externa a las subredes. Alternativa 2

División de la red por una subred externa compartida. La Figura 109 muestra cómo se usa una subred, llamada red global, que es común a todas las subredes para compartir estados y entrada/salida de token. Esto combina las formas de comunicación entre subredes presentadas anteriormente. Este caso implica combinar los casos anteriores y además manejar la red global.

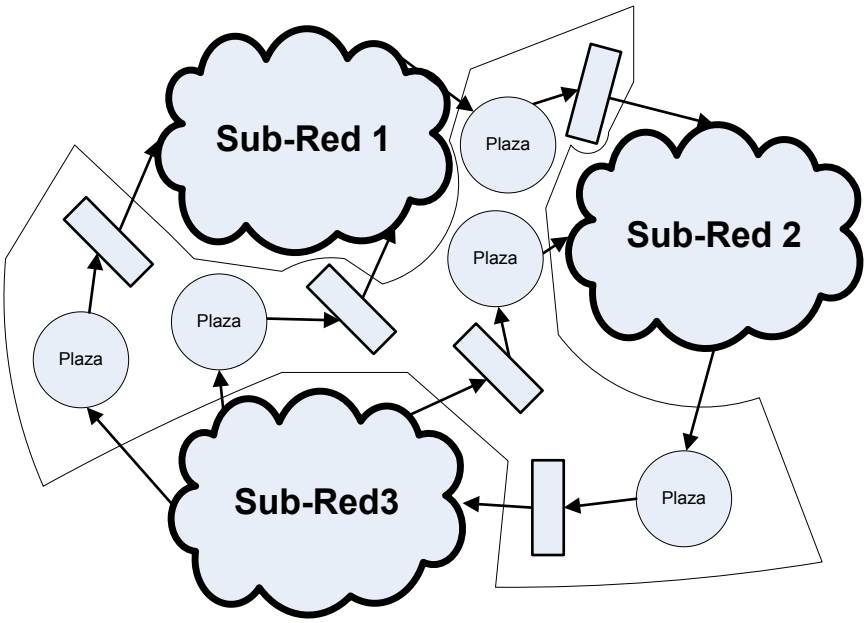


Figura 109: División de RdP, interconectándolas con una RdP externa a las subredes. Alternativa 3

Para realizar la relación es necesario compartir por cada brazo a una plaza una variable entera y por cada brazo a una transición una variable binaria, y cómo están relacionadas y si se escribe o lee.

División de la red por plazas internas compartidas. En este caso las plazas comunes a distintas subredes están repetidas para comunicar su estado, esto se muestra en la Figura 110. Es decir que los valores en las plazas tienen que mantener la coherencia y consistencia. Se trata de valores enteros.

Para esto es posible usar un algoritmo de coherencia entre plazas repetidas.

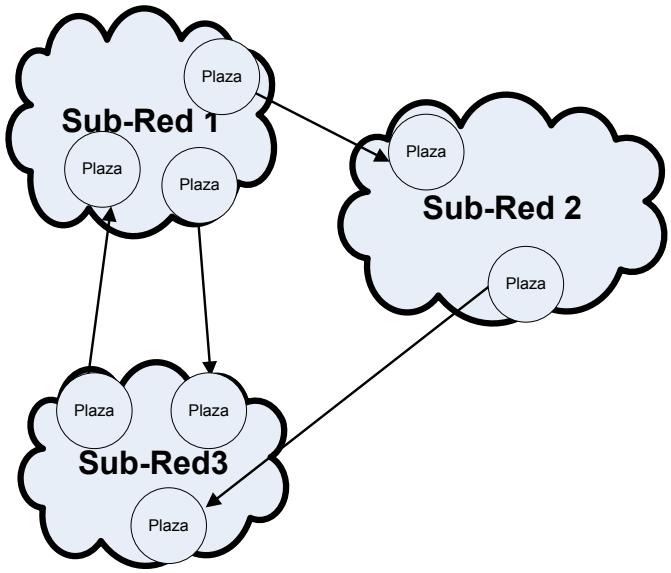


Figura 110: División de RdP, por plaza sin compartir. Alternativa 4

División de la red por transiciones internas compartidas. En este caso las transiciones comunes a distintas subredes están repetidas, para comunicar que están su sensibilizadas Es decir que los valores en las transiciones tienen que mantener la coherencia y consistencia. Se trata de valores binarios. Esto se muestra en la Figura 111.

Para esto es posible usar un algoritmo de coherencia entre transiciones repetidas.

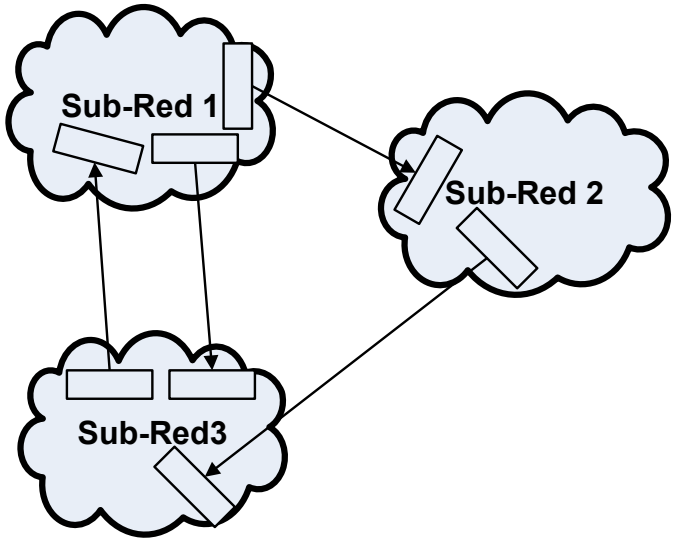


Figura 111: División de RdP: por transición sin compartir. Alternativa 5

División de la red por las plazas internas distribuidas. En este caso las plazas son comunes a distintas subredes y mantienen parte de los token. Para comunicar su estado hay que realizar una suma, es decir que los valores en las plazas comunes tienen que comunicarse a sus brazos relacionados. Se trata de valores enteros. Esto se muestra en la Figura 112.

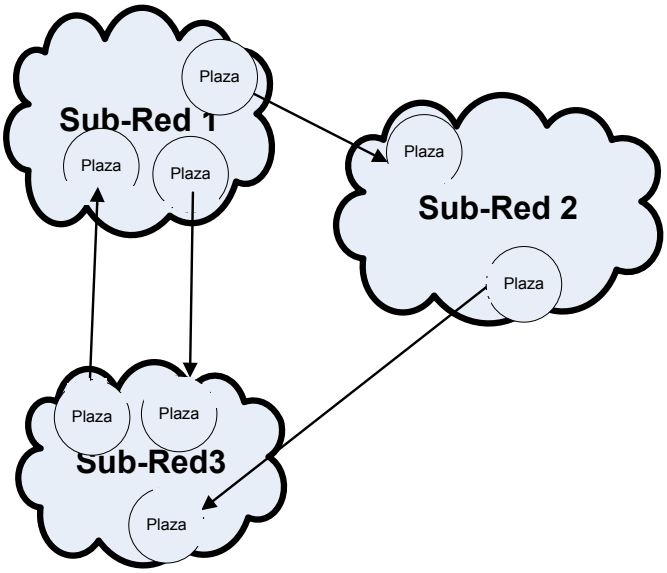


Figura 112: División con plazas comunes. Alternativa 6

Este caso es similar al primero considerado en este apartado con el agregado de una cola de entrada para cada plaza.

División de la red por transición interna distribuida. . En este caso las transiciones son comunes a distintas subredes y mantienen parte de la sensibilidad, para comunicar sensibilidad hay que realizar una comunicación bidireccional y hacer una operación *and*. De esta forma se determina si la transición esta sensibilizada. Esto se muestra en la Figura 113.

Ahora evaluaremos qué forma de división genere menos sobrecarga y acoplamiento.

Para este caso consideramos al acoplamiento como la cantidad de ciclos extras necesarios para decidir el disparo y/o estado por la división de la red. Esto es porque el grado de acoplamiento está directamente relacionado con las relaciones entre subredes. La forma en que se realiza la división impacta directamente en el algoritmo de ejecución del PP. Lo que se busca es que el paralelismo permita determinar si una transición sensibilizada es disparada en dos ciclos de reloj y que los recursos para implementar el PP disminuyan.

Es necesario considerar las tareas a realizar por el PP para evaluar las distintas alternativas. De las implementaciones anteriores realizadas para obtener el PP, se observan las siguientes tareas:

1. Comunicación de un evento entre el proceso y la cola de entrada.
2. Cálculo de transición a disparar.
3. Cálculo del nuevo estado y todos los estados posibles.
4. Comunicación del disparo a la cola de salida.

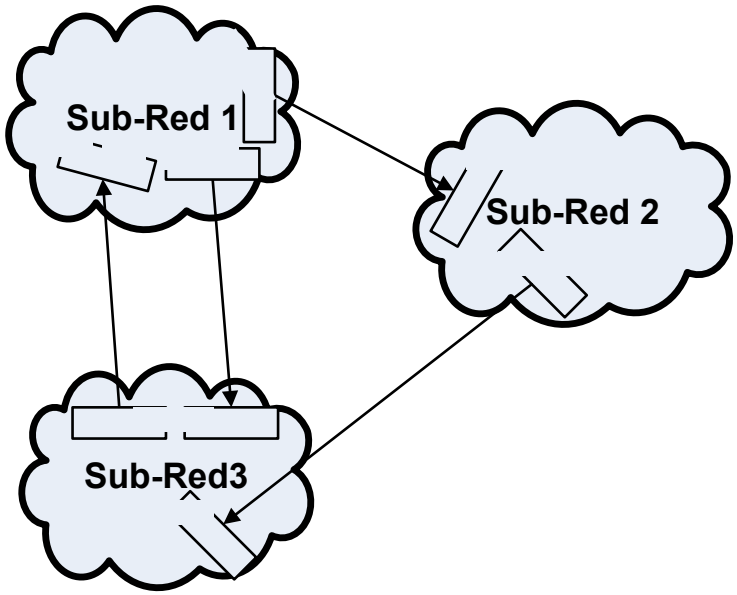


Figura 113: División por transiciones comunes a distintas subredes. Alternativa 7

Existen dos estados de cálculos:

a. Las actividades 1 y 2 que se realizan en el mismo ciclo de reloj.
b. Las actividades 1, 3 y 4 que se realizan en el mismo ciclo de reloj.

Estos dos estados de cálculo, a y b, son disjuntos en el tiempo puesto que no pueden calcularse en el mismo instante ya que en a. se calcula lo que se indica en el punto 2 ,que requiere del estado, y en b. se calcula el nuevo estado.

Consideraciones de las alternativas de división
Primero analizamos las alternativas 1, 2 y 3. Estas tres alternativas consisten en dividir las redes a través de un conjunto de plazas, transiciones o plazas/transiciones. Esta división implica que hay conjuntos de elementos compartidos por las subredes y que son externos a éstas. Si originalmente el estado del sistema se determina en K ciclos, ahora necesitamos por lo menos 2K ciclos para realizar esta tarea. Se necesitan K ciclos para el subconjunto compartido, y K ciclos para las subredes, puesto que para realizar el cálculo se requiere del cálculo de la otra parte de la red. Otro inconveniente en esta alternativa es que al actualizar información de estado en más de un módulo, se requiere ciclos adicionales para el esquema de prioridades.

Ahora consideramos las alternativas 4 y 5. Los elementos que se utilizan como límites para dividir la red aparecen duplicados dentro de cada subred con la que limitan. Luego se debe especificar un vínculo existente entre estos elementos. Este vínculo significa que ambos elementos de las distintas subredes componen la misma plaza o transición. Si existen plazas duplicadas, éstas deben tener siempre el mismo valor, es decir que deben estar cableadas. Si existen transiciones duplicadas, éstas deben ser disparadas de forma simultánea sólo cuando ambas estén sensibilizadas. En el caso de duplicar las plazas se generan costes y complejidades adicionales. Esto es porque los valores de las plazas son enteros, y las operaciones entre ellas son aritméticas. Para cablear las plazas se debe implementar una compleja red de multiplexores que debe ser configurada por el usuario. Otra alternativa es aumentar el número de ciclos de actualización de las plazas. De esta forma surgen los mismos problemas que presentan las alternativas 1, 2 y 3.

La alternativa 6, que consiste en tener valores parciales en cada plaza, no es compleja de implementar puesto que es necesario totalizar para determinar si una transición esta sensibilizada. Esto nos conduce a la alternativa uno.

Ahora consideremos la alternativa 7, la división de la red por transiciones internas distribuidas. Esta alternativa nos permite que las operaciones entre las mismas transiciones, distribuidas en distintas subredes, sean funciones lógicas de un bit por transición. Por lo que sobre esta forma de división se ha realizado diseños y análisis más exhaustivos.

La Tabla 105 muestra una comparación entre los parámetros de interés en las alternativas de la división de las redes con el objetivo de implementar el PP.

Tabla 105: Comparación de ciclos, prioridades y complejidad de las distintas alternativas de división

Alternativa	*Ciclos para determinar nuevo estado*	*Pérdida de la capacidad de prioridad*	*Complejidad de Hardware*
1	**Más de 2**	**Si**	+
2	**Más de 2**	**Si**	+
3	**Más de 2**	**Si**	+
4	**Más de 2**	**No**	+
5	**Más de 2**	**No**	+
6	**Más de 2**	**No**	++
7	**2**	**Si**	=

Para visualizar cómo se realiza la división de las RdP por transiciones internas compartidas, se hace con un enfoque gráfico, que se muestra en la Figura 114. La división ha sido realizada según lo planteado en la alternativa 7. Para esto hacemos dos definiciones, que se exponen en los próximos apartados.

Tipos de transiciones

Transiciones de borde y transiciones distribuidas

El primer tipo de transiciones son las transiciones internas de las subredes. Estas transiciones son las que no fueron cortadas y se tratan como transiciones normales ya que no tienen relación con las demás subredes. En la Figura 114, T2 es interna a la subred 1 y T5 es interna a la subred 2.

El segundo tipo es llamado “transiciones de borde” y surgen como consecuencia de cortar una transición de la red original. Estas transiciones, si bien pertenecen a una subred, son compartidas (dividida) y están relacionadas con otras transiciones de borde de otras subredes. Las transiciones de borde que provienen de una misma transición, en la red original, comparten una relación. La responsabilidad de esta relación es mantener el comportamiento de la transición de la red original (en cuanto a su sensibilidad).

En la Figura 114 la línea punteada establece los límites de cada subred, donde la red ha sido dividida en tres subredes.

También podemos ver en la Figura 114, que las transiciones T0 y T1 son de borde y pertenecen a la subred 1 y subred 3; mientras que las transiciones T3 y T4 son de borde y pertenecen a la subred 2 y subred 3.

Las transiciones de borde, que forman parte de una misma transición en la red original, se sensibilizan cuando todas sus partes están sensibilizadas en cada una de las subredes. Esto es equivalente a hacer una operación *and* entre todos los estados de las transiciones de borde que pertenecen a una misma transición en la red original.

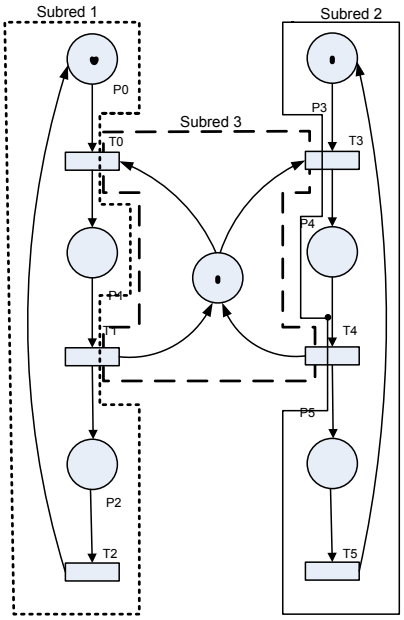


Figura 114: División de una RdP por transiciones internas distribuidas

Por lo dicho anteriormente, definimos como "transición distribuida" al conjunto de transiciones que originalmente constituían la misma transición en la red sin dividir, y ahora se encuentran distribuidas en distintas subredes, es decir que son el conjunto de transiciones que han sido divididas.

"La división de una transición de la red original dá origen a una transición distribuida, independientemente del número de partes en que ésta haya sido dividida."

"Cada parte de la transición dividida dará origen a una transición de borde en una subred distinta."

El número de partes en que la transición, más divida, haya sido dividida es el mayor número transiciones de borde y representa la mínima cantidad de subredes que constituirá el sistema resultante.

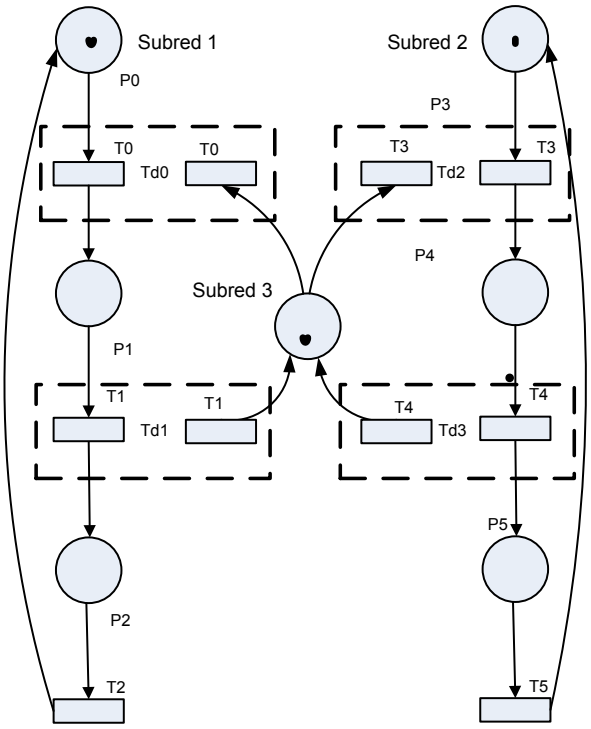


Figura 115: Transiciones distribuidas que resultan por la división en tres subredes

En la Figura 115, se puede ver que se generarán cuatro transiciones distribuidas. Esto resulta de dividir a la red original en tres subredes, para lo que se cortan cuatro transiciones. En la Figura 115 se aprecia las tres subredes resultantes de la división, y las transiciones distribuidas que han sido nombradas Td0, Td1, Td2 y Td3. Cada una de estas transiciones distribuidas está compuesta por dos transiciones de borde.

Lo importante de dividir la red por las transiciones es que se requiere una operación and, y ésta nos permite totalizar múltiples variables con el mínimo esfuerzo de cálculo (lo mismo para una operación or). Para visualizar esto solo hay que comparar con la suma de múltiples variables enteras, que es lo que se requiere para totalizar los token de una plaza distribuida.

La innovación que se introduce en esta manera de dividir la red, es que la comunicación para determinar si la transición está sensibilizada, es bidireccional y solo se requiere una función booleana. La operación, que se realiza para determinar la sensibilidad, es un and lo que implica que la carga de cálculo es mínima y permite fácilmente totalizar las múltiples variables para obtener el resultado.

Podemos ver que existe solo una regla para realizar la división y esta división no altera la semántica de sensibilización, en ninguna de las redes consideradas. Estas son las ventajas con respecto a los resultados obtenidos por las distintas investigaciones que se han considerado en el capítulo 2, [56, 185, 190, 219, 221].

Matriz de relación con transiciones distribuidas

Para especificar la relación existente entre las transiciones de borde en cada subred, se indica que estas forman parten de una determinada transición distribuida. Esta relación se implementa con una matriz binaria por cada subred. Esta matriz es llamada “matriz de relación entre transiciones

distribuidas". Para cada transición de la subred la matriz indica si ésta pertenece a una transición distribuida, la que es de borde. Cada columna de la matriz se corresponde con una transición de la subred, y cada fila se corresponde con una transición distribuida del sistema. Un 1 en la posición a_{ij}, de la matriz, indicará que la transición j de la subred en cuestión es una transición de borde que forma parte de la transición distribuida i. Si se hace coincidir la i con las j la matriz es la identidad y no es requerida.

El algoritmo de ejecución de las RdP jerárquicas basa su funcionamiento en el uso de estas matrices. Es una matriz binaria puesto que la cantidad de recursos que consume es pequeña y permite implementar una función booleana programable para relacionar subredes.

Teniendo en cuenta las transiciones de borde y distribuidas, obtenidas en la división realizada en la Figura 115, se obtienen las matrices de relación con transiciones distribuidas en la Tabla 106 (ver la RdP de la Figura 114). La implementación propuesta también permite mantener desacoplada toda la información de las distintas subredes. Esto conlleva a un aumento del grado de paralelismo (disparo de transiciones en distintas subredes sin conflicto en paralelo) del sistema, posibilitando la ejecución de todas las subredes en paralelo.

Tabla 106: Matriz de relación de la división de la RdP de la Figura 114

Subred 1

	T0	T1	T2
Td0	1	0	0
Td1	0	1	0
Td2	0	0	0
Td3	0	0	0

Subred 2

	T3	T4	T5
Td0	0	0	0
Td1	0	0	0
Td2	1	0	0
Td3	0	1	0

Subred 3

	T0	T1	T3	T4
Td0	1	0	0	0
Td1	0	1	0	0
Td2	0	0	1	0
Td3	0	0	0	1

Análisis Preliminar de Disminución de Recursos

Este análisis se realiza sobre los registros de hardware necesarios para almacenar la información del modelo, es decir para almacenar los elementos de la matriz de incidencia, la matriz de prioridades y la matriz de relaciones entre las subredes. La matriz de prioridades mantiene las prioridades entre transiciones para su disparo. Es binaria y cuadrada, tiene tantas filas o columnas como transiciones tiene la red |T|x|T|. Cada subred tiene una matriz de prioridad que especifica la prioridad de ejecución de cada transición de dicha red. Para el análisis se considera una RdP con arcos de peso mayores a uno, esto implica que lo registros para almacenar los valores de la matriz de incidencia son de 8 bits. Si consideramos una RdP con m plazas y n transiciones, la cantidad de registros necesarios para representar la información de la misma es:

$$PN = 8(mxn) + (nxn)$$

El primer término representa los recursos para almacenar la matriz de incidencia de la red, y el segundo representa los recursos para almacenar la matriz de prioridades.

Ahora consideramos dividir la RdP en p subredes de igual tamaño. Para considerar en el cálculo las transiciones distribuidas generadas por la división, definimos el "factor de acoplamiento" que denotamos con α. Este factor indica el porcentaje de las transiciones de la red original (n) que son cortadas. Para calcular α se debe dividir la cantidad de transiciones cortadas por la cantidad de transiciones totales de la red sin dividir (suponemos que este número es mayor a la cantidad de veces que fue cortada la transición que más ha sido cortada).

$$\alpha = (transiciones\ cortadas)/(transiciones\ totales)$$

Donde α varía entre 0 y 1, y p (número de subredes) es un entero mayor o igual a 1. Teniendo en cuenta lo expuesto, la cantidad de recursos necesarios para expresar la red dividida (HPN) es:

$$HPN = p\left(\frac{m}{p}\frac{n}{p}\right)8 + p\left(\frac{n}{p}x\ \frac{n}{p}\right) + p\left(\propto n\ \frac{n}{p}\right) + \propto n\ \propto n\ +\ TD$$

Los tres primeros términos se multiplican por p, puesto que consideramos el sistema constituído por las p redes. Donde el primer término corresponde a las matrices de incidencia, con 8 bits por elementos, de las p subredes. El segundo término representa las matrices binarias de prioridad de cada subred. El tercer término representa las matrices de relación con transiciones distribuidas de cada subred.

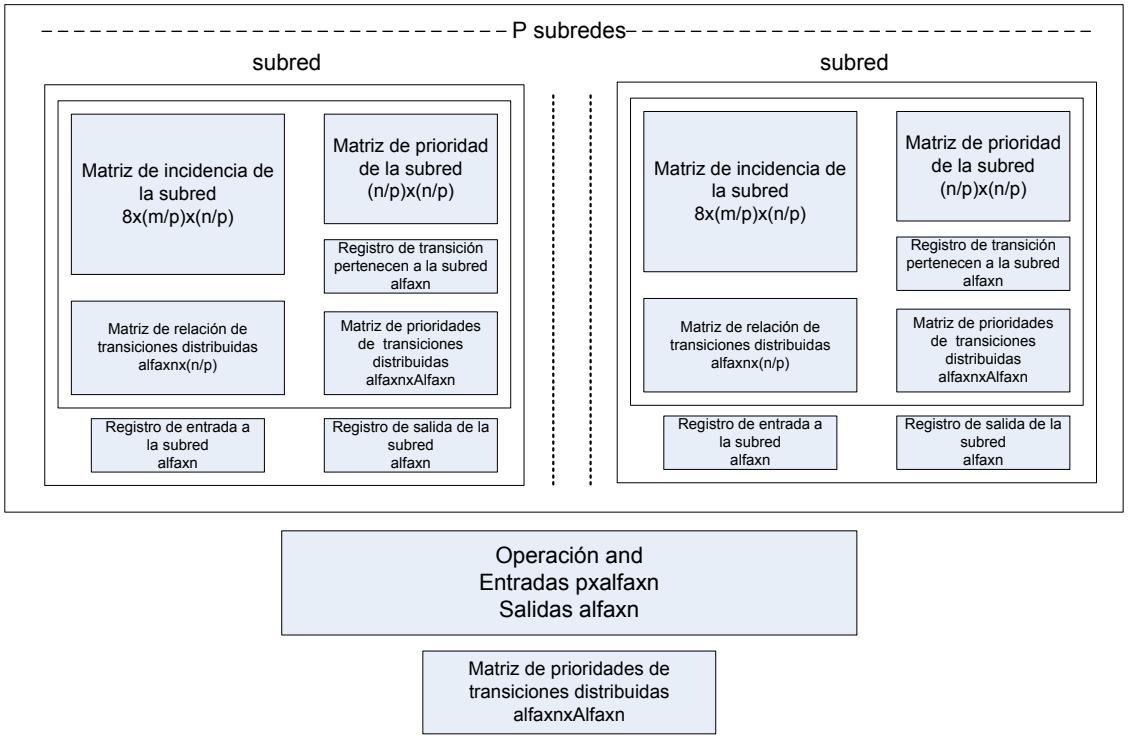


Figura 116: Elementos que intervienen para totalizar los recursos del sistema

Finalmente el cuarto término representa a la matriz de prioridades de transiciones distribuidas. Esta matriz es agregada para posibilitar la programación de las prioridades de las transiciones distribuidas (definidas por el usuario). El quinto y último término representa los recursos adicionales generados por repetir las transiciones divididas. Operando obtenemos:

$$HPN\ = \frac{8mn}{p} + \frac{n^2}{p}\ +\ \alpha n^2\ +\ \alpha^2 n^2\ +\ TD$$

Para obtener TD, es necesario considerar las transiciones repetidas por subred, un registro para inhibir las transiciones de borde que no tienen relación con la subred, un registro para las transiciones de borde de salida de cada sub red y un registro de entrada. Por lo que los recursos requeridos son:

$$TD\ = 8\alpha nm + p\alpha n + p\alpha n + p\alpha n$$

Los términos de TD corresponden a: los recursos que se agregan por repetir las transiciones divididas y los registros que se agregan para calcular la sensibilización de las transiciones de borde.

$$HPN = \frac{8mn}{p} + \frac{n^2}{p} + \alpha n^2 + \alpha^2 n^2 + 8\alpha nm + 3p\alpha n$$

Para obtener la ganancia de recursos, introducimos el factor R como el cociente entre la cantidad de registros necesarios para expresar la RdP dividida, y los utilizados para la RdP sin dividir. Este factor expresa el porcentaje de recursos que utiliza la RdP dividida, tomando como base del 100% los utilizados por la red sin dividir. Operando matemáticamente y suponiendo de que las matrices son cuadradas, es decir m = n, obtenemos la expresión de R:

$$R = \frac{HPN}{PN}$$

$$\frac{HPN}{8n^2} = \frac{1}{p} + \frac{1}{8p} + \frac{\alpha}{8} + \frac{\alpha^2}{8} + \alpha + \frac{3p\alpha}{8n}$$

$$R = \frac{9}{8p} + \frac{9\alpha}{8} + \frac{\alpha^2}{8} + \frac{3p\alpha}{8n}$$

Si G, es la ganancia de recursos como consecuencia de haber dividido la red. Este factor se obtiene restando al 100% el factor R:

$$\mathrm{G} = 1 - \mathrm{R}$$

$$G = 1 - \left(\frac{9}{8p} + \frac{9\alpha}{8} + \frac{\alpha^2}{8} + \frac{3}{8}\frac{p\alpha}{n}\right)$$

La Figura 117, representa la curva de ganancia de recursos, donde n es 50 y la variable independiente es p y en las ordenadas se obtiene el valor de G en porcentaje.

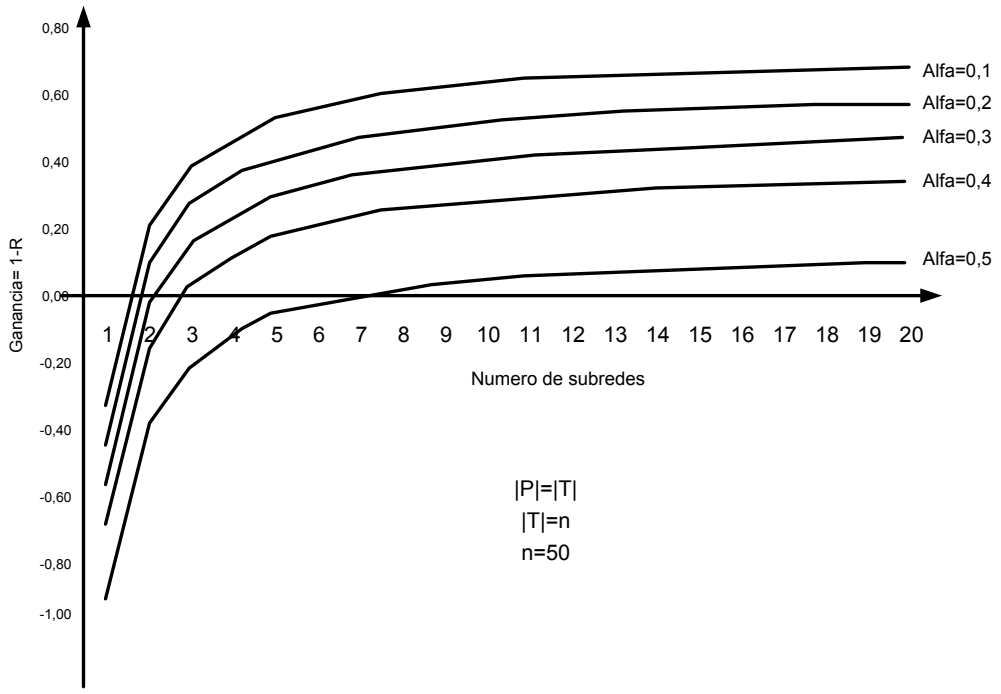


Figura 117: Curva de ganancia de recursos para distintos valores de alfa y con n=50

En la Figura 118 se muestra la gráfica resultante para n= 1000.

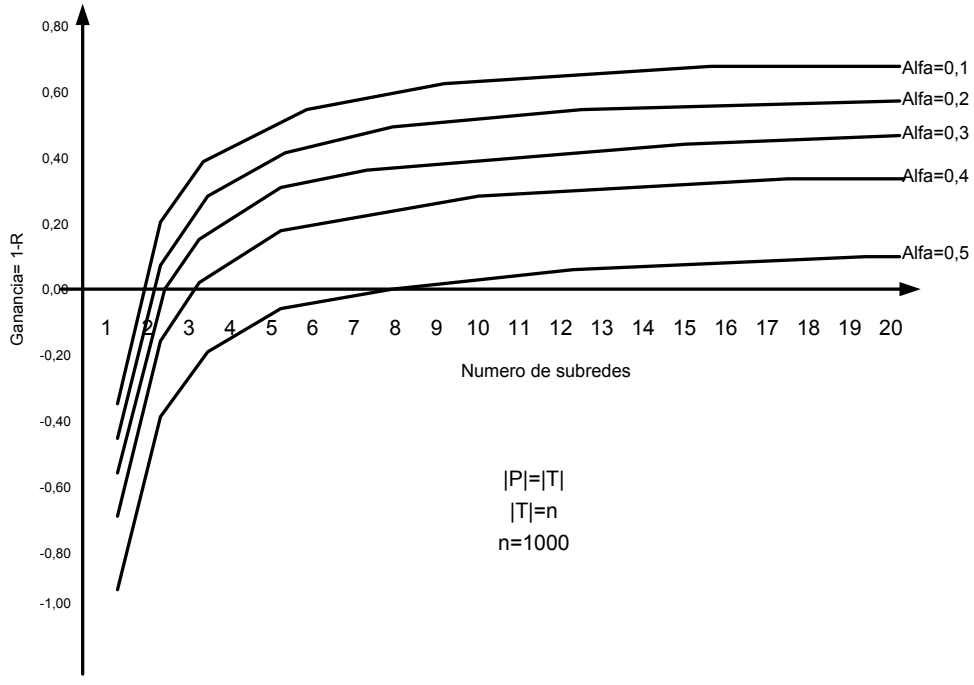


Figura 118: Curva de ganancia de recursos para distintos valores de alfa y con n=1000

De las Figura 117 y Figura 118 podemos ver que mientras más pequeño es el valor de alfa y más grade es n, mayor ganancia podemos obtener. Con alfa=0,01 y n=1000 con 20 divisiones es posible en teoría obtener una ganancia de más del 90%.

Si no dividimos la red no se obtiene ganancia, se obtiene solo pérdida por los módulos que se han introducido y no cumplen función alguna.

En las Figura 117 y Figura 118, el corte de las curvas (distintos alfa) en las abscisa (cantidad de subredes) indican la mínima división con la que se obtiene ganancia, esto ha sido graficado en la Figura 119; este valor también depende de n. Por lo que para un valor de n y alfa hay una cantidad mínima de subredes para obtener ganancia.

Consideraciones para División RdP

Para realizar la disminución de recursos que nos hemos propuesto, dividiendo una RdP de la forma y el modo aquí propuesto, es necesario contestar los siguientes interrogantes:

- ¿Cuánto y cuál es la ganancia de dividir las RdP? Esto ha sido contestado en el apartado anterior y vemos que la ganancia depende de alfa y el valor de n.
- ¿Las RdP resultante son más simples? Vemos que la respuesta es sí, puesto que es posible realizar la división partiendo las transiciones en cualquier número de partes, siempre que la transición sea dividida en un número igual o menor al número de brazos que entran y salen de la transición. Las subredes resultantes son más pequeñas y solo hay una matriz binara de relación.
- ¿Cómo realizar la división? Esta es la respuesta que buscamos en este apartado.

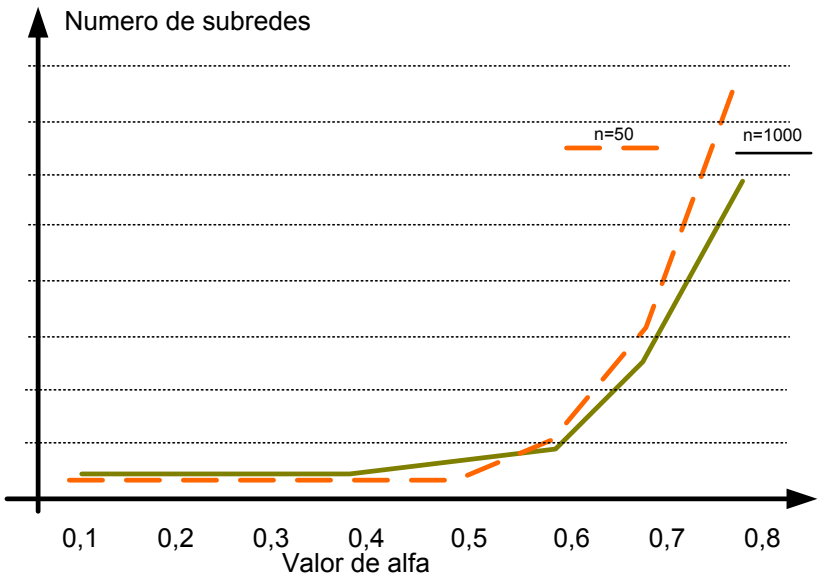


Figura 119: Valores mínimos de división según alfa

La división tiene que expresar y preservar el comportamiento del modelo. Puesto que son RdP no autónomas, es también menester que se mantengan las relaciones con los eventos que disparan las

transiciones. Además las comunicaciones que generan los disparos de las transiciones deben poder ser informadas. Es decir que la división no introduzca ambigüedad en los eventos generados por el disparo de una transición. Finalmente el esquema de prioridades tiene que ser posible de implementar y facilitar la ejecución en paralelo.

La Figura 120, muestra una RdP dividida en subredes, por la división de sus transiciones.

El proceso de dividir la RdP, se realiza cortando transiciones. Estas transiciones cortadas se encontrarán presentes en cada una de las subredes resultantes, la comunicación se realiza por las transiciones de borde, que son los límites de las subredes. Podemos entonces decir que existe una relación entre todas estas transiciones cortadas, que originalmente constituían una única transición, que están comunicadas y se evalúan y disparan de forma simultánea.

En la Figura 120, vemos como las transiciones T3, T4, T8, T10 y T12 son cortadas en dos, mientras que la transición T5 ha sido cortada en tres partes. Esta división dá origen a tres subredes. La primera subred está compuesta por {P0, P1, P8, P9 y P10} y {T0, T1, T2 y T'5}, la segunda subred la componen {P2, P6 y P7} y {T'3, T´4, T'5, T´10, T´12 y T'8}, mientras que la tercera está compuesta por {P3, P4 y P5} y {T´4, T'5, T6, T7 y T'8}.

Hay que notar que el hecho de dividir las redes de esta forma, genera una jerarquía explícita de redes, con un concepto más amplio, que llamamos ¨Redes Jerárquicas Divididas por transiciones¨. La característica es que el conjunto de subredes relacionadas mediante la división de las transiciones distribuidas provoca una redución de la cantidad de recursos.

También puede ser considerada como RdP jerárquica para su facilitar interpretación o una RdP distribuida para ser implementada en distintos dispositivos intercomunicados como se realiza en [50].

Seguidamente se presentan una serie consideraciones para la división de una RdP.

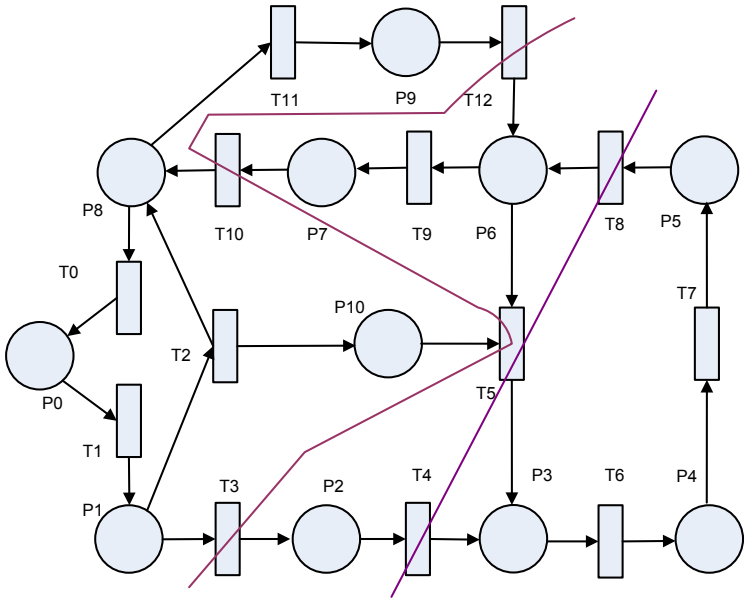


Figura 120: RdP dividida en sub redes, por la división de las transiciones

Consideraciones para obtener Redes Jerárquicas Divididas por Transiciones

Lo primero que observamos, es que sólo existe una red. No hay una red global y subredes, sino que todas las partes de la red o subredes tienen la misma jerarquía y se relacionan entre ellas mediante transiciones de borde. No obstante a cada parte de la red obtenida por la división la llamaremos subred.

1. Las transiciones distribuidas están constituidas por el conjunto de todas las transiciones de borde, que son el resultado de la división de una misma transición de la red general.
2. Para obtener el estado global del sistema consideramos todas las plazas en conjunto, es decir que hay que tener el marcado de cada subred.
3. Debido a la necesidad de mantener la consistencia del sistema, cuando se ejecuta una transición distribuida, se deben ejecutar todas las transiciones de borde de todas las subredes a las que pertenece. Para que esto suceda, todas las transiciones de borde que pertenecen a una misma transición deben estar sensibilizadas.
4. El disparo de una transición distribuida puede generar conflicto en una subred, para que esto no suceda solo se permite la ejecución de una sola transición distribuida por ciclo.
5. Las redes se dividen cortando las transiciones, de modo que las transiciones elegidas como límite entre dos o más redes quedarán partidas, constituyendo una transición de borde para cada una de las redes. Estas transiciones de borde, en cada red estarán relacionadas, constituyendo una transición distribuida.
6. Como cada red mantiene su funcionamiento por separado, se aumenta el nivel de paralelismo en la ejecución del sistema, pudiéndose dar el caso de que todas las redes ejecuten un disparo (en transiciones que no son de borde) en el mismo ciclo.
7. Cuando todas las transiciones distribuidas que pertenecen a una misma transición están sensibilizadas, esta transición está en condiciones de dispararse.
8. Para que el disparo de una transición distribuida sea efectivo, se deben disparar las transiciones de borde relacionadas en todas las redes.
9. La matriz binaria de relación de transiciones distribuidas, indica como las transiciones distribuidas se reagrupan para constituir la transición original en cada subred.

10. Para evitar excesiva comunicación entre las subredes, ante un conflicto, las transiciones distribuidas tienen mayor prioridad de disparo que cualquier transición interna a la red.
11. Si se desea que una transición interna con conflicto a una subred tenga más prioridad en la ejecución que una transición distribuida, se la puede definir como una transición distribuida que solo está relacionada con la red a la que pertenece.

Ejemplo: siete filósofos

A modo de prueba, se evalúa el problema de la cena de los filósofos. Consideramos el caso con 7 filósofos. Para esto dividimos a red de la Figura 121 para obtener la red dividida de la Figura 122 y luego evaluamos la ganancia.

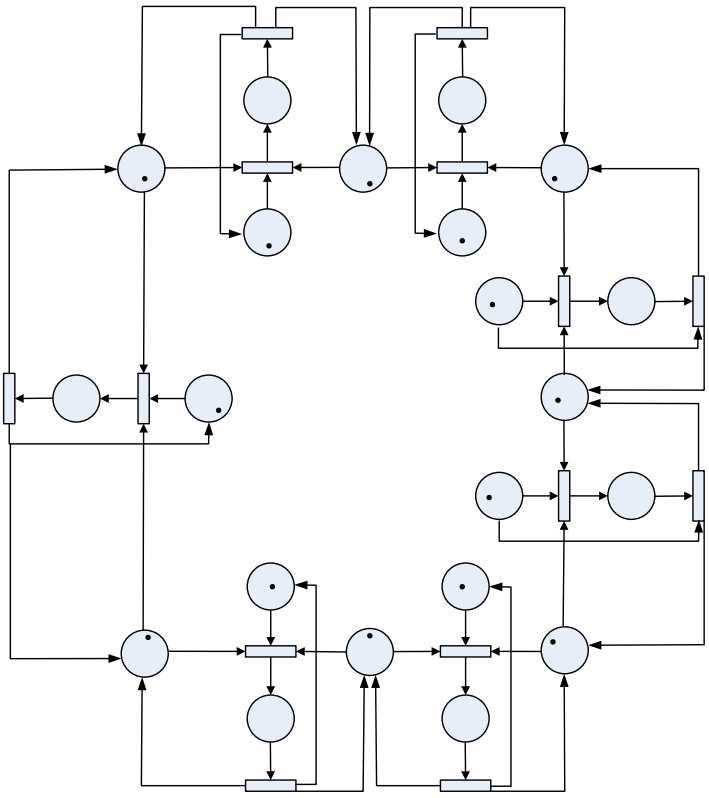


Figura 121: Cena de siete filósofos sin dividir

Esta red tiene 21 plazas y 14 transiciones considerando la matriz de prioridades, requieren 2548 bits, para la implementación.

$$PN = (mxn)8 + (nxn) = 2548\ bits$$

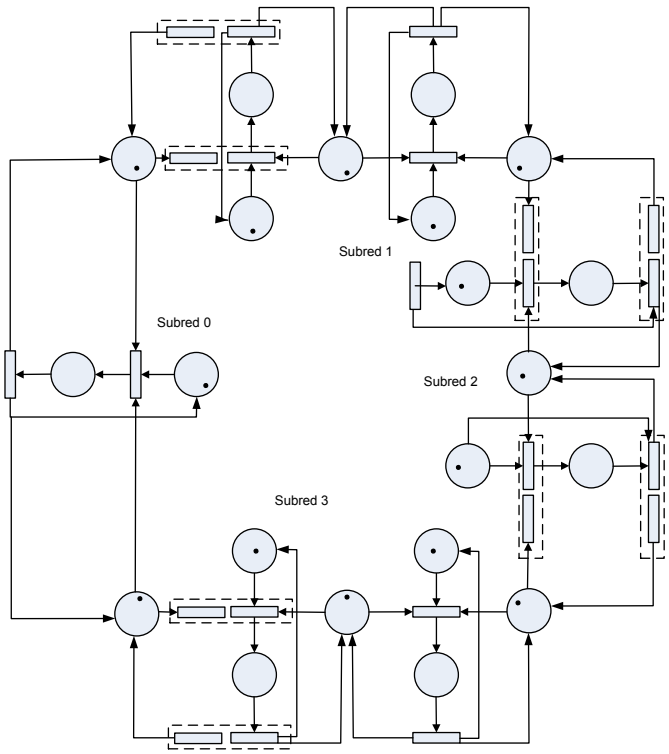


Figura 122: Cena de siete filósofos divididos en cuatro subredes

Dividiendo esta red en 4 subredes, se requieren los recursos que muestra la Tabla 107.

Tabla 107: Recursos para la cena de los filósofos dividida en cuatro subredes

	Plazas	transiciones	Matriz de prioridad	Cantidad de recursos en bit
Red0	4	6	6x6	228
Red1	6	6	6x6	324
Red2	5	5	6x6	324
Red3	6	6	6x6	324
Matriz de relación	Son 8 las transiciones de borde, con cuatro sub redes 8x6x4			192
Matriz de Prioridad de transiciones de borde	8*8			64
			Total de recursos	1426
			Recursos que ocupa la red dividida 1426/2548	56%

La RdP dividida requiere 1426 bits. Por lo la red dividida ocupa el 51% de la red sin dividir, dando una ganancia del 44%.

División de una red con N lectores y un escritor en tres subredes

En la Figura 123, se muestra la RdP que modela a N lectores y un escritor, que ha sido dividida en tres subredes.

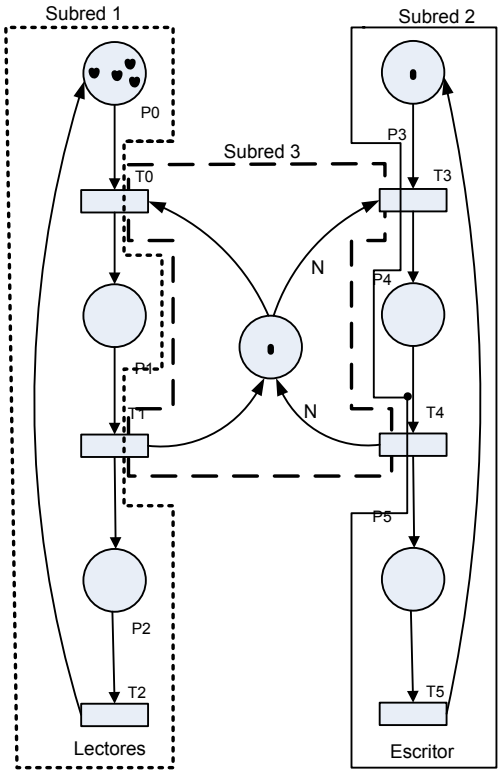


Figura 123: RdP de N lectores con un escritor dividida en tres sub redes

Originalmente se parte de una RdP de 7 plazas y 6 transiciones, para esta RdP se requieren de 372 bits.

$$PN = (mxn)8 + (nxn) = 372$$

Dividiendo ésta como lo muestra la Figura 123, en tres sub redes, los recursos necesarios son 290. Se muestra en la Tabla 108.

Tabla 108: Recursos para una RdP con N lectores y un escritor, dividida en tres subredes

	Plazas	transiciones	Matriz de prioridad	Cantidad de recursos en bit
Red0	3	3	3x3	81
Red1	3	3	3x3	81
Red2	1	4	4x4	64
Matriz de relación	Son 8 las transiciones de borde, con cuatro sub redes 4x4x3			48
Matriz de Prioridad de Transiciones de borde	4*4			16
			Total de recursos	290

Recursos que ocupa la red dividida 290/372	79%

La RdP dividida requiere de 290 bits. Mientras que la RdP sin dividir requiere de 372 bita. Por lo que la división tiene una ganancia del 21%.

División de una red de control

Nos proponemos dividir la red de la Figura 124, en tres subredes para determinar la reducción de recursos.

Con el objetivo de obtener tres redes con una cantidad de plazas y transiciones similares, se ha dividido la red como se muestra en la Figura 124.

La red original tiene 10 plazas y 13 transiciones, esto requiere de 1209 bit para su implementación.

$$PN = (mxn)8 + (nxn) = 1209$$

Los recursos necesarios para las tres subredes se muestran en la Tabla 109 y han sido evaluados de la misma manera que en los casos anteriores.

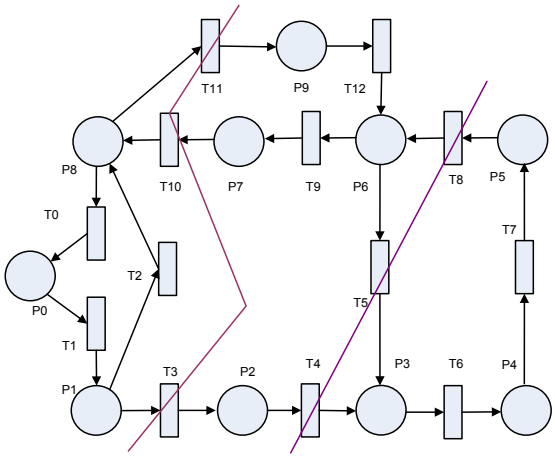


Figura 124: Red de control dividida en tres subredes

Tabla 109: Recursos para una RdP de control, dividida (6,8,5) en tres subredes

	Plazas	Transiciones	Matriz de prioridad	Cantidad de recursos en bit
Red0	3	6	6x6	180
Red1	4	8	8x8	320
Red2	3	5	5x5	145
Matriz de relación	Son 6 las transiciones de borde, con tres sub redes 6x8x3			148
Matriz de Prioridad de transiciones de	6*6			36

borde		
	Total de recursos	829
	Recursos que ocupa la red dividida 290/372	68,5%

Para la red dividida se requieren 829 bit, y por lo que la ganancia es del 31,3%

División de un sistema de control y direccionamiento de sistemas de vehículos guiados automáticamente en tres subredes

Nos proponemos dividir la red de la Figura 125, en tres subredes para determinar la reducción de recursos. Con el objetivo de obtener tres redes con una cantidad de plazas y transiciones similares, se ha dividido la red de la Figura 125,

Esta red posee 14 plazas y 20 transiciones, por lo que requieren 2640 bit para implementarla.

$$PN = (mxn)8 + (nxn) = 2640$$

De la división podemos obtener la Tabla 110, que muestra los recursos necesarios para implementar la red dividida.

Tabla 110: Recursos para una RdP de control, dividida (8,10,7) en tres subredes

	Plazas	Transiciones	Matriz de prioridad	Cantidad de recursos en bit
Red0	4	8	8x8	320
Red1	5	10	10x10	500
Red2	5	7	7x7	329
Matriz de relación	Son 3 las transiciones de borde, con tres sub redes 3x3x10			90
Matriz de Prioridad de transiciones de borde	3*3			9
			Total de recursos	1248
			Recursos que ocupa la red dividida 1248/2640	47%

Para la red dividida se requieren 1248 bits, y por lo que la ganancia es del 52%.

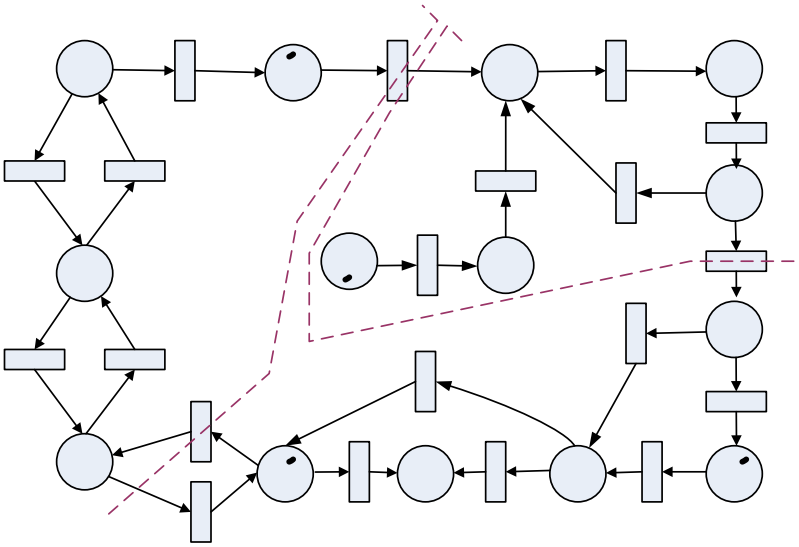


Figura 125: División de la red del sistema de vehículos guiados en tres subredes

Componentes del PP y el HPP

En este apartado presentamos un algoritmo para la ejecución de una RdP dividida. Para la ejecución se requiere de las matrices de incidencia, relación y prioridades, los vectores de estado, colas de disparo, etc. Será implementado como parte del HPP.

En la siguiente sección se definen los elementos necesarios que caracterizan cada una de las subredes de RdP que componen el sistema.

Definición de los elementos que componen las subredes
Al igual que en una RdP es necesario definir a los elementos que componen la red, como se interrelacionan en la subred y como se interrelacionan las subredes para conformar al sistema.

- |P| cantidad de plazas de una subred.
- |T| cantidad de transiciones de una subred.
- |d| cantidad de transiciones de borde.
- S cantidad de subredes que componen el sistema.
- d cantidad de transiciones distribuidas del sistema.

Para que se verifiquen los supuestos que se han adoptado en la división de una RdP, es necesario que se cumpla:

- La suma de las plazas que conforman cada subred sean las plazas de la red.
- La suma de todas las transiciones que conforman cada subred sean las transiciones de la red.
- La cantidad de transiciones de borde no sea menor a la subred con mayor cantidad de transiciones distribuidas.

A continuación se detallan un conjunto de vectores y matrices que componen el HPP. En la Figura 126 se muestra el diagrama en bloque de una subred del HPP y el área común de las

transiciones globales. Este diagrama se repite para cada subred del procesador. Podemos ver que la diferencia con el PP radica en la inclusión del disparo de las transiciones globales.

Matriz de Incidencia de la red (subred).

Módulo etiquetado con (1) en la Figura 126, es la matriz de incidencia de la subred con m plazas y n transiciones. La definición es la misma que la matriz de incidencia de cualquier RdP.

Seguidamente, en la Figura 128 se muestra una subred RdP simple y en la Figura 129 matriz de incidencia.

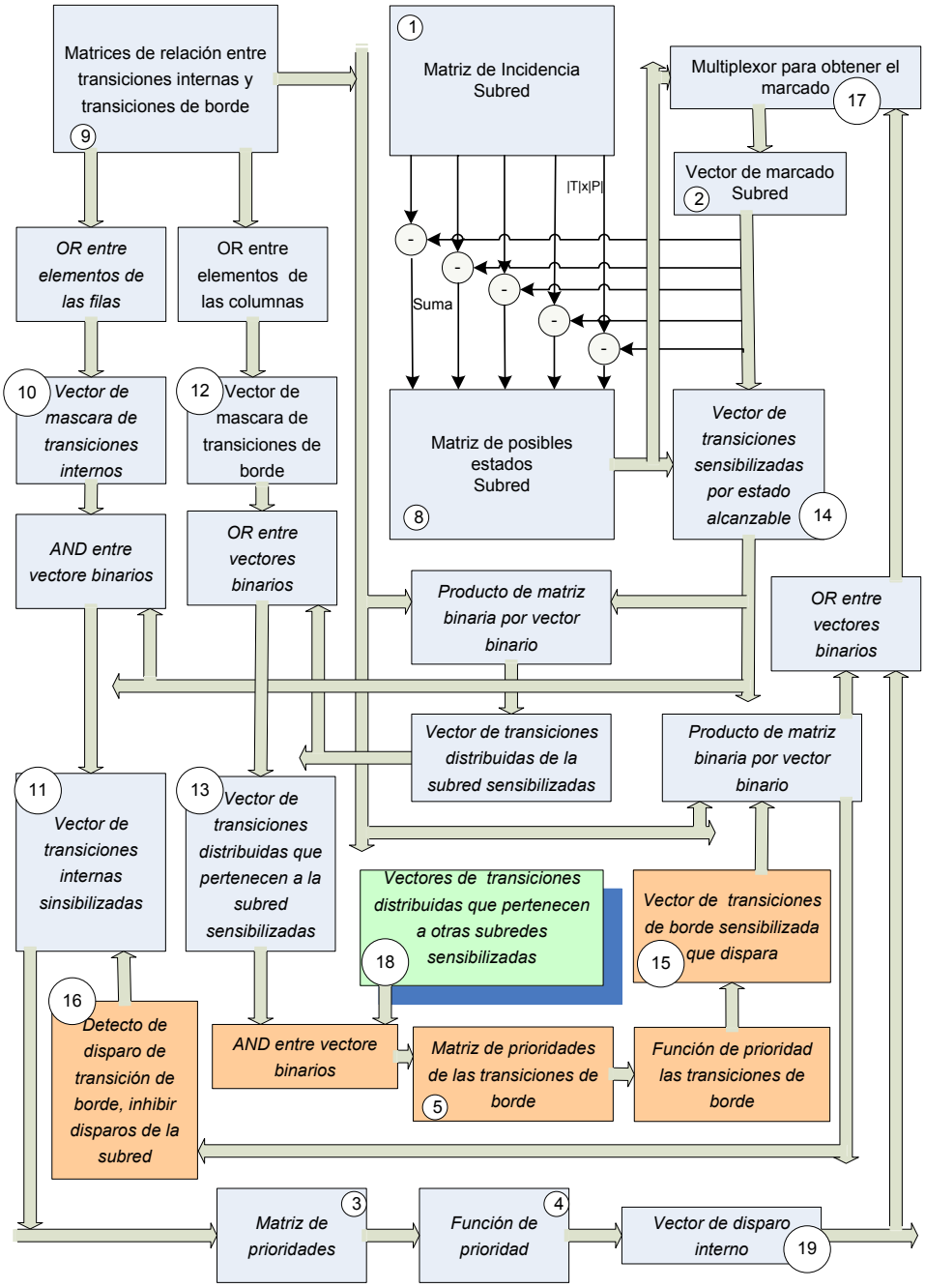


Figura 126: Diagrama en bloque registros e interconexión HPP

Marcado de la red (Subred)

Módulo etiquetado con (2) en la Figura 126. El marcado y el marcado inicial de una subred es el mismo que el de una RdP, un vector de m elementos que indica cuantos tokens hay en cada plaza. Los valores de este vector son enteros positivos. La componente a_i, del vector de marca, es la cantidad de token en la plaza i.

$P0 = 1 \quad P1 = 0 \quad P2 = 0$	$P3 = 1 \quad P4 = 0 \quad P5 = 0$	$P6 = N$
Marcado de la Sub-red 1	**Marcado de la Sub-red 2**	**Marcado de la Sub-red 3**

Figura 127: Marcado inicial de las subredes de la Figura 128

Para la red dividida en subred de la Figura 128, tiene como marcado inicial de cada subred el de la Figura 127. Y la matriz de incidencia de la red completa es la de la Figura 129 mientras que las matrices de incidencia de cada subredes se muestran en la Figura 130 (para la subred 3 se han indicado las transiciones).

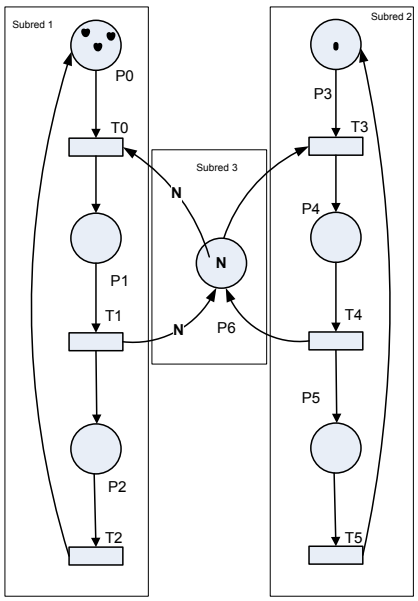


Figura 128: RdP con un escritor y múltiples lectores

$$\begin{bmatrix} & T0 & T1 & T2 & T3 & T4 & T5 \\ P0 & -1 & 0 & 1 & 0 & 0 & 0 \\ P1 & 1 & -1 & 0 & 0 & 0 & 0 \\ P2 & 0 & 1 & -1 & 0 & 0 & 0 \\ P3 & 0 & 0 & 0 & -1 & 0 & 1 \\ P4 & 0 & 0 & 0 & 1 & -1 & 0 \\ P5 & 0 & 0 & 0 & 0 & 1 & -1 \\ P6 & -N & N & 0 & -1 & 1 & 0 \end{bmatrix}$$

Figura 129: Matriz de la RdP de la Figura 128

$$\begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ -1 & 1 & -1 \end{bmatrix} \qquad \begin{matrix} [T0 & T1 & T3 & T4] \\ [-N & N & -1 & 1] \end{matrix}$$

Matriz Subred1 **Matriz Subred2** **Matriz Subred3**

Figura 130: Matrices de las subredes

Matriz de prioridad

El módulo etiquetado con (3) en la Figura 126. La matriz de prioridad, es binaria y de dimensión nxn. La responsabilidad de esta matriz es determinar la transición que será disparada cuando se tenga simultáneamente múltiples transiciones sensibilizadas. Es decir permite determinar, con el fin de disparar, la transición de más prioridad.

Para el ejemplo de la Figura 128, definimos arbitrariamente el orden de prioridad, de las transiciones de la subred 3, que es la única con conflictos, como se muestra en la Figura 131.

$$[T3 \quad T4 \quad T1 \quad T0]$$

Figura 131: Prioridades de las transiciones de la Figura 128

Mientras que el vector de transiciones sensibilizadas, con posibles disparos tiene el orden según el subíndice de la transición, que es el mostrado en la Figura 132.

$$[\boldsymbol{T0} \quad \boldsymbol{T1} \quad \boldsymbol{T3} \quad \boldsymbol{T4}]$$

Figura 132: Transiciones ordenadas por el índice de transición

El objetivo es ordenar las transiciones según su prioridad, que originalmente están ordenadas por el índice de transición, para seleccionar la transición sensibilizada de más prioridad. Esto lo realizamos con una relación booleana, que es programable en tiempo de ejecución. Esta programación, permite prioridad constante de tareas o cambiarlas en función del estado del

sistema. Posibilitando que el PP, implemente algoritmos de planificación basados en prioridades estáticas o dinámicas.

Con este fin definimos la matriz de relación que transforma al vector con el orden de transiciones de la Figura 132, en el vector con el orden de prioridades de transiciones de la Figura 131.

$$vector\ con\ orden\ de\ prioridad = \ vector\ con\ orden\ de\ índice\ x\ matriz$$

El caso considerado se muestra en la Figura 133.

$$[T3 \quad T4 \quad T1 \quad T0] = [T0 \quad T1 \quad T3 \quad T4]x \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Figura 133: Matriz de prioridad para obtener la prioridad programada

Como se ve en la Tabla 132 la matriz de prioridad tiene un uno en cada columna, el uno se encuentra en la fila que coincide con la posición de la prioridad que tiene la transición que se corresponde con el índice de la columna.

Función de prioridad

El módulo etiquetado con (4) en la Figura 126. Ahora la función de prioridad tiene que retornar un uno sólo en la posición de más prioridad. Para esto se hace la siguiente operación booleana.

$$[a1 \quad a2 \quad a3 \quad a4] = [T3 \quad T4 \quad T1 \quad T0]$$

$$f1 = a1$$

$$f2 = a2\ and\ \overline{f1}$$

$$f3 = a3\ and\ \overline{f2}$$

$$f4 = a4\ and\ \overline{f3}$$

Ahora el vector $f = [f1 \quad f2 \quad f3 \quad f4]$ cuando una o más transición esta sensibilizada, tiene solo un uno en la posición de más prioridad, si todas las componentes son cero el vector f es cero.

Si multiplicamos la matriz por el vector f, obtenemos el vector d, este vector tiene un uno en la transición sensibilizada de mayor prioridad y está en el orden de las transiciones, esto se muestra en la Figura 134. El vector d es el que se usa para realizar el disparo.

$$[d1 \quad d2 \quad d3 \quad d4] = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} x \begin{bmatrix} f1 \\ f2 \\ f3 \\ f4 \end{bmatrix}$$

Figura 134: Cálculo de prioridades en el orden de las transiciones

Matriz de prioridad para las transiciones de borde

El módulo etiquetado con (5), es una matriz cuadrada de pxp y binaria. Su responsabilidad es determinar, en el caso de múltiples transiciones sensibilizadas, cúal es la de mayor prioridad para su disparo.

Esta matriz tiene el mismo rol que la matriz de prioridad y se ejecuta exactamente igual en cada subred, excepto que aplica la prioridad sólo sobre las transiciones distribuidas. Como la ejecución de una transición distribuida implica la ejecución simultánea de dos o más transiciones de borde en redes distintas, la capacidad del sistema para ejecutar transiciones distribuidas está restringida a una sola transición distribuida por ciclo. De otra forma, se complica el controlador que se requiere para la ejecución de múltiples transiciones distribuidas simultáneamente. Implica un ciclo de ejecución más para evaluar los conflictos.

Para seleccionar la transición distribuida de mayor prioridad se procede de la misma forma que procede cada subred utilizando su matriz de prioridad.

Función de prioridad de transiciones de borde

El módulo etiquetado con (6) en la Figura 126 provee la función de prioridad de las transiciones de borde y tiene la misma responsabilidad que la función de prioridad de las transiciones de la subred, pero para las transiciones de borde. Retorna un uno sólo en la posición de más prioridad.

Componentes del HPP

A partir de las matrices y vectores, definidos en los párrafos precedentes, se pueden calcular los estados de cada subred y las transiciones sensibilizadas. Todo esto permite determinar que transición se dispara y el nuevo estado. Por consiguiente el algoritmo de ejecución consiste en calcular la evolución de la red, según los eventos externos y el estado de la red.

Para determinar el siguiente estado de la subred de RdP, se utiliza la ecuación de estado

$$m_{k+1} = m_k + I(d_k)$$

Caso de múltiples lectores para ejemplificar el algoritmo

Con el fin de explicar el algoritmo para la ejecución de una RdP jerárquica, haremos referencia a la RdP de múltiples lectores y un escritor. Esta red es la que se muestra en la Figura 135, dividida en 3 subredes.

Las subredes 1 y 2 contienen tres plazas y tres transiciones, mientras que la subred 3 tiene una plaza y cuatro transiciones.

Las transiciones de borde son: Td0, Td1, Td2 y Td3.

La transición de borde Td0 tiene dos transiciones distribuidas, que son T0 de la subred1 y T0 de la subred3. De la Figura 135, se pueden inferir el resto de las transiciones de borde y distribuidas.

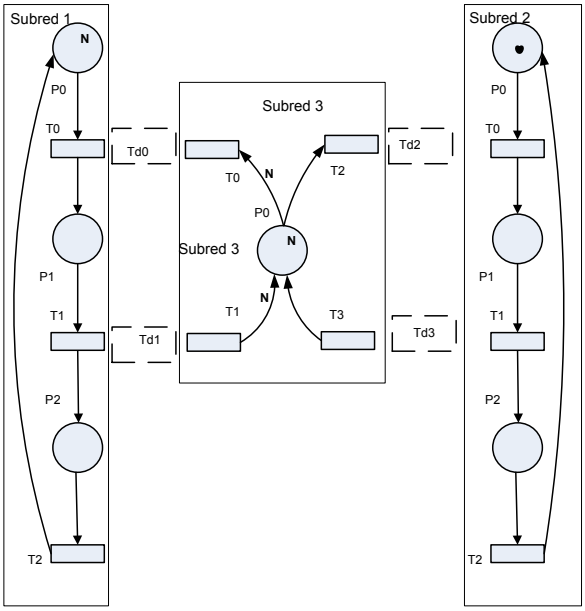


Figura 135: Red jerárquica de múltiples lectores y un escritor

De la Figura 136, vemos cómo determinar si una transición de borde está sensibilizada, para esto realizarmos una operación *and* entre todas las transiciones de borde relacionadas con la transición distribuida.

El valor de Ts1d0, es el mismo que tiene T1 en la subred1, pero este valor pertenece al vector con el que se calcula Td0, que es el usado para calcular transiciones de borde Td0.

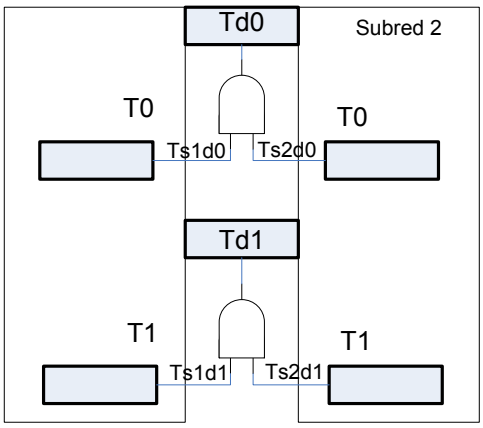


Figura 136: AND entre transiciones distribuidas para determinar si una transición de borde está sensibilizada

Las matrices de incidencia las subredes del Figura 135 son mostradas en la Figura 137.

Subred2 **Subred3**

$$\begin{array}{c|ccc} & T0 & T1 & T2 \\ P0 & -1 & 0 & 1 \\ P1 & 1 & -1 & 0 \\ P2 & 0 & 1 & -1 \end{array} \qquad \begin{array}{c|ccc} & T0 & T1 & 1 \\ P0 & -1 & 0 & \\ P1 & 1 & -1 & \\ P2 & 0 & 1 & - \end{array} \qquad \begin{array}{c|ccc} & T0 & T1 & T2 \\ P0 & -N & N & -1 \end{array}$$

Figura 137: Matrices de incidencia de las subredes del ejemplo.

El marcado inicial de cada sub red del ejemplo de la Figura 135, se muestran en la Figura 138.

Subred1 m0		Sunred2 m1		Subred3 m3	
$P0$	N	$P0$	1		
$P1$	0	$P1$	0	$P0$	N
$P2$	0	$P2$	0		

Figura 138: Marcado inicial de cada subred de la Figura 135

Matriz de posibles estados

El módulo etiquetado con (8) en la Figura 126 es calculado con la ecuación de estado y son los posibles estados, para esto se considera que sólo se puede realizar un disparo a la vez. El marcado alcanzado será la suma del estado actual más la columna que se corresponde con el disparo, por lo que la nueva marca toma un valor distinto según la transición que se ejecute. La matriz se construye considerando que se realizó un disparo para cada transición por cada ciclo de ejecución.

Si la matriz de incidencia de la subred está formada por n columnas:

$$I = |C^0, C^1, \dots, C^n|$$

Donde C^i representa a la columna i. El posible disparo de cada transición de la subred, dará como resultado las siguientes ecuaciones de próximo estado:

- $m_k[t0 > \ m_{k+1} = m_k + C^0$, disparo de la transición T0
- $m_k[ti => \ m_{k+1} = m_k + C^i$, disparo de la transición Ti
- $m_k[tn => \ m_{k+1} = m_k + C^n$, disparo de la transición Tn.

Para determinar cada posible nuevo estado, se suma al marcado actual a la columna de la matriz de incidencia correspondiente a dicha transición. Con la restricción de un disparo a la vez, no es necesario realizar el producto matricial para determinar los nuevos estados, y es posible construir una matriz que contenga todos los nuevos estados posibles de la subred correspondiente a la ejecución de cada transición.

Para determinar qué transición es posible de disparar, se verifica que no tengan token negativo el nuevo estado posible.

Teniendo en cuenta lo anterior, a partir del marcado actual y la matriz incidencia, se define una nueva matriz de resultado, que es una matriz de enteros con signo de dimensión mxn.

Esta matriz tiene todos los posibles estados, que se corresponden a haber disparado la transición una vez la transición con el índice de la columna.

$$Columna\ (j)\ de\ matriz\ resultado = \sum_{i=1}^{n} matriz\ incidencia(i)(j) + marcado\ actual(i)$$

Esta matriz se usa para determinar si una transición está sensibilizada o no. El cálculo de esta matriz es realizada en paralelo, y se hace en forma especulativa para determinar qué disparo es posible antes de que sea solicitado.

Para el ejemplo de la Figura 135, se muestran en la Figura 139 las matrices de posibles estados para cada subred.

Matriz de posibles estados para la Subred1

	$C0$	$C1$	$C2$
$P0$	$N-1$	N	$N+1$
$P1$	1	-1	0
$P2$	0	1	-1

Matriz de posibles estados para la Subred2

	$C0$	$C1$	$C2$
$P0$	0	1	2
$P1$	1	-1	0
$P2$	0	1	-1

Matriz de posibles estados para la Subred3

	$C0$	$C1$	$C2$	$C3$
$P0$	0	0	$2N$	$2N$

Figura 139: Matrices de los posibles estados para cada subred del ejemplo de la figura.

Matrices de relación entre transiciones internas y transiciones de borde

Módulo etiquetado con (9) de la Figura 126. La matriz relación nos permite obtener cuales transiciones locales a una subred están relacionadas con las transiciones distribuidas, es decir que transiciones son de borde. También es posible determinar que transiciones distribuidas del sistema tienen o no relación con cada subred. Esta matriz nos permite calcular dos vectores, que son:

- El vector de las transiciones internas sensibilizadas, que solo dependen del estado interno de la subred. Estas son las transiciones no relacionadas de la subred con otras transiciones.
- El vector de transiciones distribuidas, indica que transiciones son distribuidas y con cual transición de borde está relacionada.

Este último vector es usado para el cálculo de la sensibilidad de las transiciones distribuidas, para lo que se consideran todas las transiciones distribuidas en cada subred en un instante de tiempo.

Para el caso del ejemplo de la Figura 135, en la Figura 140 se muestran las matrices que relacionan a las transiciones distribuidas de cada subred con las transiciones de borde.

Relación entre las transiciones internas de la subred con las transiciones de borde

Para la subred 1
$$[T0 \quad T1 \quad T2] \xrightarrow{R1^{3x4}} [Ts1d0 = T0 \quad Ts1d1 = T1 \quad 0 \quad 0]$$

Para la subred 2
$$[T0 \quad T1 \quad T2] \xrightarrow{R2^{3x4}} [0 \quad 0 \quad Ts1d2 = T0 \quad Ts1d3 = T1]$$

Para la subred 3
$$[T0 \quad T1 \quad T2 \quad T3] \xrightarrow{R3^{4x4}} [Ts1d0 = T1 \quad Ts1d1 = T2 \quad Ts1d2 = T3 \quad Ts1d3 = T4]$$

Figura 140: Relación de las distintas subredes con las transiciones de borde

El resultado del cálculo de la relación es un vector binario que hace coincidir los componentes relacionados, con dimensión igual a la cantidad de transiciones de borde. Este vector, tiene el valor que tiene la transición distribuida que está relacionada en la subred correspondiente, y si no se encuentra en la subred vale cero. En la Figura 141 se muestran las matrices de relación del ejemplo de la Figura 135.

$$R1^{3x4} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R1^{3x4} \quad \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R1^{4x4} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 141: Matrices que relacionan las transiciones internas y transiciones de borde del ejemplo.

Vector de máscara de transiciones internas

El módulo etiquetado con (10) de la Figura 126. Este vector es la máscara para determinar que transiciones son internas a la subred. Es un vector binario de n posiciones. Este vector tiene un uno en la posición correspondiente a aquellas transiciones que son internas de la subred y que no son de borde, es decir aquellas transiciones que sólo dependen del estado local de la subred.

Las componentes de este vector se obtienen con el cálculo de una operación or entre todas las componentes de una fila de la matriz relación, y el resultado es negado. Esto es porque cada fila de la matriz representa una transición local de la subred.

Si se verifica que para una transición, todos los elementos son 0, esto implica que esta transición no tiene ninguna relación con una transición distribuida, es decir que es solo una transición interna de la red.

$$mascara\_internos\ (i)\ = NOT(\bigvee_{j=0}^{p-1} matriz\_relacion(i)(j))$$

En la Figura 142 se muestran los vectores de máscara interna de cada subred del ejemplo de la Figura 135.

Subred1	**Subred2**	**Subred3**
$[0 \quad 0 \quad 1]$	$[0 \quad 0 \quad 1]$	$[0 \quad 0 \quad 0 \quad 0]$

Figura 142: Vector de máscara de subredes internas del ejemplo

Podemos ver que en las subredes 1 y 2 las transiciones T2 de cada una de ellas son internas y que la subred 3 no tiene ninguna transición interna.

Vector de transiciones habilitaciones que no están relacionadas con las transiciones de borde

Es el módulo etiquetado con (11) en la Figura 126. Vector de transición habilitación, es un vector binario de dimensión |T|. El objetivo de este vector es determinarar las transiciones internas que están sensibilizadas y que no tienen relación con las transiciones de borde.

Este vector se obtiene con el cálculo de una operación and entre el vector de máscara interna y el vector de transiciones sensibilizadas.

Para el caso del ejemplo de la Figura 135, el vector de transiciones que no están relacionadas en cada subred es aplicado para el cálculo de las transiciones que se desactivan en cada subred por pertenecer a una transición de borde, esto está mostrado en la Figura 143.

Subred1 $[T0 \quad T1 \quad T2]\ AND\ [0 \quad 0 \quad 1]$

Subred2 $[T0 \quad T1 \quad T2]\ AND\ [0 \quad 0 \quad 1]$

Subred3 $[T0 \quad T1 \quad T2 \quad T3]\ AND [0 \quad 0 \quad 0 \quad 0]$

Figura 143: Cálculo de las transiciones de borde que se desactivan en cada subred del ejemplo.

Vector de mascara de transiciones de borde

Módulo etiquetado con (12) en la Figura 126. Cada elemento de este vector se calcula como una operación or entre todos los elementos de cada fila de la matriz relación de la subred, y al resultado se lo niega. Cada fila de la matriz relación representa una transición distribuida. Por lo tanto si todos los elementos de dicha fila son iguales a 0, significa que la subred RdP no tiene ninguna relación con esta transición distribuida.

$$Vector\ de\ máscara\ de\ transiciones\ de\ borde\ [i] = NOT(\bigvee_{j=0}^{n-1} matriz\_relación[i][j])$$

Este vector pone un uno en la transición distribuida que no existe en la subred, de esta forma no impide el sensibilizado de la transición de borde que no tiene transición distribuida. En la Figura 144 se muestran los vectores para las subredes del ejemplo.

Subred1 $Vector\ de\ máscara\ de\ transiciones\ de\ borde\ [1] = [0 \quad 0 \quad 1 \quad 1]$

Subred2 $Vector\ de\ máscara\ de\ transiciones\ de\ borde\ [2] = [1 \quad 1 \quad 0 \quad 0]$

Subred3 $Vector\ de\ máscara\ de\ transiciones\ de\ borde\ [3] = [0 \quad 0 \quad 0 \quad 0]$

Figura 144: Vector de máscara de transiciones de borde para las subredes del ejemplo.

Vector de transiciones de borde sensibilizadas

El módulo etiquetado con (13) en la Figura 126 es para calcular la transición de borde sensibilizada. Hay que calcular el vector de máscara de transiciones de borde, las transiciones distribuidas sensibilizadas de cada subred y los vectores de máscara de transición de borde de cada subred. Con estos vectores se calculan las transiciones de borde sensibilizadas. Con esto se verifica que cada transición distribuida de cada subred esté sensibilizada, lo que implica que todas la transiciones de borde relacionadas están sensibilizadas.

Para el caso del ejemplo de la Figura 135, el cálculo del vector de transiciones de borde sensibilizadas se muestra en la Figura 145.

$$([Ts1d0 \quad Ts1d1 \quad 0 \quad 0]\ or\ [0 \quad 0 \quad 1 \quad 1])$$
$$and$$
$$([0 \quad 0 \quad Ts2d2 \quad Ts2d3]\ or [1 \quad 1 \quad 0 \quad 0])$$

$$
\begin{gathered}
and \\
([Ts3d0 \quad Ts3d1 \quad Ts3d2 \quad Ts3d3]\ or\ [0 \quad 0 \quad 0 \quad 0]) \\
= \\
([Ts3d0\ and\ Tsd0 \quad Ts3d1\ and\ Ts1d1 \quad Ts3d2\ and\ Ts2d2 \quad Ts3d3\ and\ Ts2d3]
\end{gathered}
$$

Figura 145: Cálculo del vector de transiciones de borde sensibilizadas del ejemplo.

Hay que hacer notar que $Ts3d0$ indica que se trata de la transición distribuida cero y pertenece a la subred 3. Esto ya había sido indicado en la Figura 136.

Vector de transiciones sensibilizadas por estado alcanzable

Módulo etiquetado con (14) en la Figura 126. De la matriz de resultado se puede calcular que transiciones locales de la subred y de borde están sensibilizadas. Para esto se tienen en cuenta el signo de todos los componentes de los estados posibles, ya calculados. Este vector es binario y de n elementos.

Este vector tiene un uno en la posición de la transición que está sensibilizada, según la condición del signo de la ecuación de estado. Dichas transiciones pueden ser tanto internas como de borde. Cada elemento del vector sensibilizado (el signo) se calcula haciendo una operación or entre los bits de signo de todos los elementos de una columna de la matriz resultado. Si se detecta un elemento negativo, el lugar que corresponde a dicha columna en el signo del vector sensibilizado se pone a 0, ya que significaría que no es posible su disparo.

$$
sensibilizados\ signo(i) = \begin{cases} 1 \quad si\ matriz_{resultado[j][i]} \geq 0 \quad para\ j = 0 \ldots (m-1) \\ 0\ en\ \ otro\ caso \end{cases}
$$

Vector de transiciones internas sensibilizadas

El vector de transiciones internas sensibilizadas se obtiene a partir del vector de transiciones sensibilizadas locales y del vector transiciones sensibilizadas internas, que solo contiene los disparos sensibilizados internos, es decir aquellas transiciones que verdaderamente están en condición de ejecutarse sólo por el estado actual de la subred, y que no dependen del estado de otra subred. Es un vector binario y de n elementos (|T|). Este vector contiene un uno en aquellas transiciones de la subred que cumplen la condición para estar sensibilizadas y además no tienen ninguna relación con las transiciones distribuidas, es decir que no son de borde. Este vector se calcula realizando un and entre los elementos de los vectores sensibilizados locales y el vector de transiciones internas.

$$
sensibilizados\ (i)internos = sensibilizados\_signo(i)\ \ AND\ mascara\_internos(i)
$$

Vector de transiciones de borde sensibilizada que dispara (15)

Módulo etiquetado con (15) en la Figura 126. Este vector se calcula multiplicando el vector que indica cúales son las transiciones sensibilizadas de borde, para todas las subredes, por la matriz de prioridad y luego se aplica la función de prioridad. Este vector tiene un uno en la transición de borde sensibilizada de más prioridad, y si no hay transiciones de borde sensibilizadas es cero.

Detección de disparo de transición de borde, inhibición de disparos de la subred (16)

Módulo etiquetado con (16) en la Figura 126. Este detector indica si el vector de transiciones de borde sensibilizado que dispara es distinto de cero, en este caso inhibe el disparo de las transiciones internas de todas las subredes. De esta forma se evitan los conflictos de transiciones de borde y subredes.

Multiplexor para obtener el marcado (17)

Módulo etiquetado con (17) en la Figura 126. El multiplexor elije el nuevo estado de la subred, que está almacenado en la matriz de posibles nuevos estados. Para obtener la señal que controla al multiplexor se realiza una operación or entre las salidas de:

- El producto entre el vector de transiciones de borde sensibilizado con posibles disparos y la matriz de relación entre transiciones internas. Esta salida indica solo una transición interna relacionada con una de borde que se dispara o es cero.
- El vector de disparo interno. Este vector tiene sólo un uno, que es el que indica la transición interna sensibilizada de más prioridad, y que ninguna transición de borde es disparada. En cualquier otro caso tiene un cero.

Vectores de transiciones distribuidas que pertenecen a otras subredes sensibilizadas (18)

Módulo etiquetado con (18) en la Figura 126. Estos vectores han sido calculados por cada subred. Para realizar su cálculo se ha procedido de la misma forma que se ha descripto en el apartado del Vector de transiciones distribuidas que pertenecen a la subred sensibilizada.

Vector de disparo interno (19)

Módulo etiquetado con (19) en la Figura 126. Estos vectores tienen la misma responsabilidad que el vector de transiciones internas sensibilizadas. Éstos son calculados, cada uno en el modulo de la subred que pertenece. Para determinar que transición de borde está sensibilizada se realiza una operación and entre cada componente, como se ilustra en la Figura 136.

Ejecución del HPP

Para la ejecución del HPP se realizan los siguientes pasos en cada ciclo de ejecución del sistema:

Pasos que se ejecutan una sola vez.

1. Para cada subred del sistema calcular el vector de máscara de transiciones internas a partir de las columnas de la matriz relación con transiciones distribuidas de la subred.
2. Para cada subred del sistema calcular el vector de máscara de habilitación de transiciones no relacionadas a partir de las filas de la matriz de relación con transiciones distribuidas de la subred.

Pasos que se ejecutan cada vez que se realiza un disparo o arriba un evento en cada subred.

3. Para cada subred del sistema calcular la matriz resultado, a partir de la matriz de incidencia y el marcado actual.
4. Para cada subred del sistema calcular el vector de transiciones sensibilizadas, según el signo a partir de las columnas de la matriz de resultado.
5. Para cada subred del sistema calcular el vector de transiciones locales sensibilizadas, que en este caso coincide con el vector de transiciones sensibilizadas según el signo.
6. Para cada subred del sistema calcular el vector de transiciones internas sensibilizadas a partir del vector de transiciones locales sensibilizadas y la máscara de transiciones internas.
7. Para cada subred del sistema calcular el vector de transiciones de borde sensibilizadas a partir de la matriz de relación con transiciones distribuidas de la subred y el vector de disparos locales sensibilizados.

8. Para cada subred del sistema calcular el vector de transiciones de borde sensibilizadas correlacionado a partir del vector de transiciones de borde sensibilizadas y la máscara de habilitación de transiciones no relacionadas de la subred.
9. Cada subred del sistema debe enviar su vector de transiciones de borde sensibilizadas correlacionado.

Pasos que se ejecutan, con las transiciones de borde, cada vez que se realiza un disparo o arriba un evento.

10. Calcular el vector de transiciones distribuidas sensibilizadas mediante una operación *and* entre todos los vectores de transiciones de borde sensibilizadas, enviados por todas las subredes del sistema.
11. Generar el vector de disparo distribuido seleccionado, que contiene a lo sumo la transición sensibilizada y distribuida de mayor prioridad, para ejecutarse en este ciclo. Se realiza aplicando el esquema de prioridades seleccionado por el usuario, a partir de la matriz de prioridades de transiciones distribuidas.
12. Comunicar a cada subred la transición distribuida seleccionada para la ejecución en el ciclo actual, enviando el vector de disparo distribuido seleccionado a cada una de las subredes.

Pasos que se ejecutan en cada subred, para determinar que transición se dispara.

13. Cada subred del sistema recibe el vector de borde disparado y con la matriz de relación calcula el vector de disparo distribuido que pertenece a la subred. Esto permite traducir la transición distribuida seleccionada a la transición de borde relacionada de cada subred.
14. Cada subred recibe una señal que indica si se dispara una transición de borde. En caso de ser verdadera, con esta señal se inhibe el posible disparo de una transición de la subred, en caso de ser falso se permite.
15. Para cada subred calcular el vector de disparos sensibilizados de la red a partir del vector de transiciones sensibilizadas internas y el vector de disparo de borde seleccionado de la red. Este vector contiene todas las transiciones locales de la subred, y finalmente es el que se dispara. Como resultado se obtiene una única transición local a cada subred para ser ejecutada en el ciclo actual.
16. En cada subred actualizar el marcado actual correspondiente a la ejecución de la transición seleccionada mediante el vector de disparo y recomienza el ciclo.

Consideración por Eventos entre los Procesos y el HPP

Para la comunicación entre el HPP y los procesos se realizan las siguientes consideraciones:

Comunicación entre procesos por medio de colas de entrada y salida
En el HPP se han implementado las mismas características del PP:

- Caracterización de la cola de entrada relacionada con señal que dispara a las transiciones sensibilizadas. Tipo de cola para el disparo de la transición: automático (A), disparo específico (D), o no-perenne (P).

- Caracterización de la cola de salida relacionada con la señal generada por el disparo de las transiciones. Tipo de cola de salida que comunica el disparo a los procesos: no informa, cola de salida, o interrumpe.

Para implementar estas características se trabaja, al igual que en el PP, con las colas de entrada/salida.

Para la programación de la comunicación por eventos, en relación a las colas de entrada y salida, es necesario hacer las siguientes consideraciones:

Colas de entrada en el HPP

Puesto que las transiciones son distribuidas, por lo que están en distintas subredes, cada transición puede tener una cola de entrada en la subred correspondiente. Esta cola de entrada puede ser de cualquiera de los tipos enumerados en el punto anterior. Esto es debido a que el procesador funciona para cualquier combinación de colas.

Aquí es importante analizar la consecuencia que emerge cuando una misma transición distribuida tiene una combinación de distintos tipos de señales generadas, según se traten a los eventos. Esta configuración impacta en el comportamiento del modelo de la RdP y finalmente en los procesos que el PP conduce.

Desde el punto de vista del modelo se tiene en cuenta la señal equivalente de considerar múltiples señales. Por ejemplo para una transición que ha sido dividida en tres, el análisis se realiza en la Figura 146.

Señal en la cola Ts0d0	Señal en la cola Ts1d0	Señal en la cola Ts2d0	Cálculo a realizar	Señal equivalente
$A0 \equiv 1$	$A1 \equiv 1$	$A2 \equiv 1$	$A0\ and\ A1\ and\ A2 = A$	$A \equiv 1$
$A0 \equiv 1$	$P1$	$A2 \equiv 1$	$A0\ and\ P1\ and\ A2 = P1$	$P1$
$A0 \equiv 1$	$D1$	$A2 \equiv 1$	$A0\ and\ D1\ and\ A2 = P1$	$D1$
$A0 \equiv 1$	$P1$	$D2$	$A0\ and\ P1\ and\ D2$	$P1\ and\ D2$
$P0$	$P1$	$D2$	$P0\ and\ P1\ and\ D2$	$P0\ and\ P1\ and\ D2$

Figura 146: Señal equivalente que resulta de la combinación de señales en una transición distribuida

En el cuadro de la Figura 146, no han sido incluidas todas las alternativas del caso, puesto que se derivan de las presentadas.

Señalamos, que en el caso del diseño realizado del HPP, no han sido incluidas colas en la evaluación de las transiciones globales, puesto que ya están en las distribuidas, y son una redundancia. La inclusión de estas colas puede tener sentido si es necesario realizar alguna operación entre eventos por hardware. En esta tesis este caso no ha sido analizado y/o experimentado.

La innovación que se introduce en esta manera de tratar los eventos en una red dividida, es que permite incorporar más de un evento de entrada por transición y la combinación entre los mismos. En cuanto a la comunicación por evento de salida permite comunicarse con más de un proceso, puesto que es posible generar más de un evento de salida.

Podemos ver que la regla para considerar los eventos, por el HPP, es muy clara y tiene concordancia, tanto por la incorporación de eventos que realiza [35] y las realizadas por [47, 49, 223, 226, 240]. En cuanto a las señales consideradas en las RdP IOPT (que en es una variable binaria que actúa como guarda) es equivalente a un disparo automático (verdadero) o por evento (sin evento) lo que equivale a falso.

La ventaja de este enfoque es que las entradas a las transiciones son programables (también en tiempo de ejecución) y permite que el sistema sea flexible. Mientras que el comportamiento de la señal se deja al componente que envuelve al sensor y/o actuador, ver Figura 54.

Colas de salida en el HPP

Dado que las transiciones son distribuidas, también lo están la cola de salida de cada subred. Esta cola de salida puede ser: no informa, cola de salida, o interrumpe. Esto es debido que las subredes que componen al procesador funcionan en forma independiente, y admiten cualquier combinación de colas.

Aquí hay que prever que una misma transición no genere una misma interrupción, lo que posiblemente lleve al sistema a una condición de error.

La combinación de las salidas puede ser aprovechada para comunicar un mismo evento a distintos procesos, y es útil para cuando la implementación de hardware tiene múltiples buses.

Nomenclatura de las transiciones

Con el fin de identificar, la responsabilidad de las transiciones en el gráfico de las RdP, se ha introducido una etiqueta, que indica el tipo de transición. El primer caracter de la etiqueta indica el comportamiento de la cola de entrada de la transición y el segundo de la cola de salida de la transacción, esto es mostrado en la Figura 147.

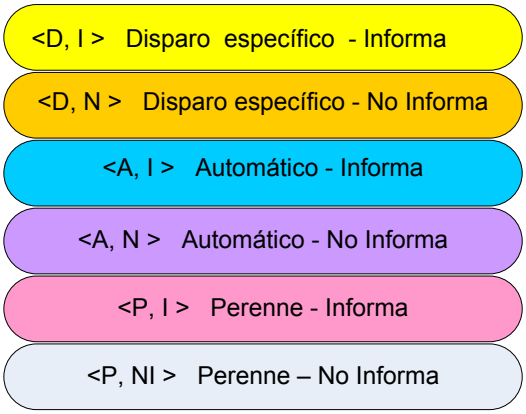


Figura 147: Etiquetas de de transiciones

Para especificar el comportamiento de las transiciones, se agregan tres vectores binarios al conjunto de vectores que programan cada subred del HPP.

Subredes de RdP con arcos inhibidores

Para incorporar los arcos inhibidores al HPP, definimos dentro de cada subred la matriz de arcos inhibidores. Con el detector de marcas y la matriz de arcos inhibidores calculamos el vector con transiciones no inhibidas por los arcos inhibidores. Finalmente, tal cual lo realizamos en el PP, calculamos el vector de transiciones sensibilizadas, pero ahora el cálculo es local de la subred. Se tiene que considerar este nuevo vector para realizar todas las operaciones.

En la Figura 148, se detallan los bloques que se han agregado al HPP para esta funcionalidad. Estos han sido etiquetados con una (H). Podemos notar que a partir de la operación and que se realiza entre el vector de transiciones sensibilizadas por transiciones alcanzables todas las operaciones son las mismas que para el HPP sin brazos inhibidores.

También hay que realizar la siguiente restricción, para el correcto funcionamiento del HPP, la plaza que tiene al estado que inhibe a la transición tienen que pertenecer a la subred. Esto ha sido implementado de esta manera para evitar la comunicación de estados entre subredes.

HPP con cotas de plazas

Para extender el HPP, con el fin de evaluar cotas de plaza, se incorpora el vector de cota de plaza para cada subred. Con este vector y la matriz de nuevos estados posibles se calcula si un disparo puede superar la cota específica. Finalmente se calcula el vector de transiciones sensibilizadas locales de la subred, realizando una operación and elemento a elemento entre los vectores de transiciones sensibilizadas y de brazos inhibidores.

En la Figura 148, se detallan los bloques que se han agregado al HPP para esta funcionalidad, estos han sido etiquetados con una (C). Podemos notar que a partir de la operación and que se realiza entre el vector de transiciones sensibilizadas por transiciones alcanzables, todas las operaciones son las mismas que para el HPP sin cota de plaza.

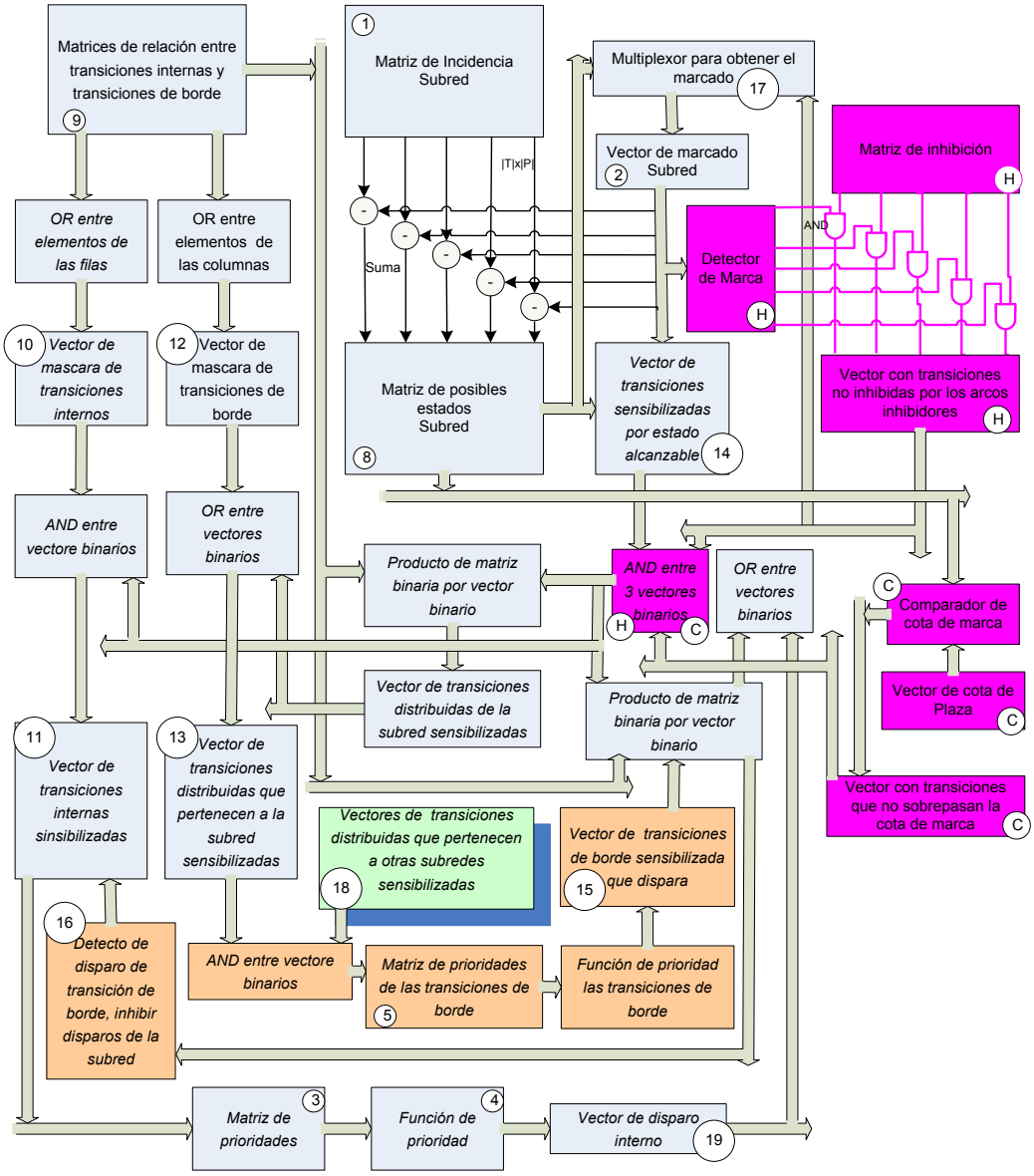


Figura 148: Diagrama en bloque del HPP con arcos inhibidores.

También hay que realizar la siguiente restricción para el correcto funcionamiento del HPP, la plaza que tiene cota de plaza tiene que pertenecer a la subred. Esto ha sido implementado de esta manera para evitar la comunicación de estados entre subredes.

Software para la Simulación del Modelo HPP

Con el objeto de validar el correcto funcionamiento del HPP propuesto, se implementa un software de simulación. El objetivo principal es validar la división de las distintas redes, el funcionamiento e interconexión de los módulos del HPP. Como también, visualizar la ejecución de las transiciones, los estados de cada subred y el estado global del sistema.

Requerimientos del Software de Simulación del HPP

De los modelos de diagramas en bloque y de los criterios de división de RdP, se obtienen los requerimientos funcionales y no funcionales del simulador.

Requerimientos no funcionales

1. Obtener los resultados de la ejecución de una RdP dividida.
2. Soportar la división de una RdP en p partes. Estas partes conforman el sistema original.
3. Programar cada subred del simulador con: la matriz de incidencia, el vector de marca inicial, la matriz de relación con transiciones distribuidas, el vector de cotas de plazas, el vector de disparos automáticos, el vector de disparos no-perenne y la matriz de prioridad.
4. Programar el simulador con la matriz de prioridades de transiciones distribuidas.
5. Especificar todas las matrices y vectores en un texto plano.
6. Especificar el comportamiento y la cantidad procesadores que acceden al HPP de forma concurrente, con un texto plano.
7. Especificar las acciones de cada procesador que accede al HPP, mediante la solicitud de disparo de una transición y/o lectura de un disparo.
8. Lectura del estado de cada transición y plazas en cada subred.
9. Parar y continuar la ejecución del HPP y los procesos

Requerimientos funcionales

1. Implementar una interface gráfica para visualizar los resultados.
2. Visualizar la transición en cada subred y el estado actualizado de cada una.
3. Especificar las rutas y nombres de los archivos con las matrices y vectores de cada subred y las acciones de cada procesador.

Software para la Evaluación del HPP

En la Figura 149 se muestra la ventana principal del simulador. Existen cinco regiones. Estas regiones son: (1) barra de menú, (2) cuadro de especificación de rutas de archivos de configuración, (3) cuadro de solicitud de disparo, (4) cuadro de simulación, y (5) lista de ciclos de simulación ejecutados. Ahora describimos las características y funciones de cada parte del simulador.

Barra de Menú

Se han programado tres menús emergentes:

1. Menú Archivo, permite: crear una nueva simulación, cargar los archivos de configuración especificados, o salir del programa de simulación.
2. Menú Simulación, permite: ejecutar un paso de la simulación, ejecutar un número específico de pasos de simulación, o simular la ejecución paralela de uno o más procesadores.
3. Menú Ayuda, permite: obtener información sobre el software de simulación.

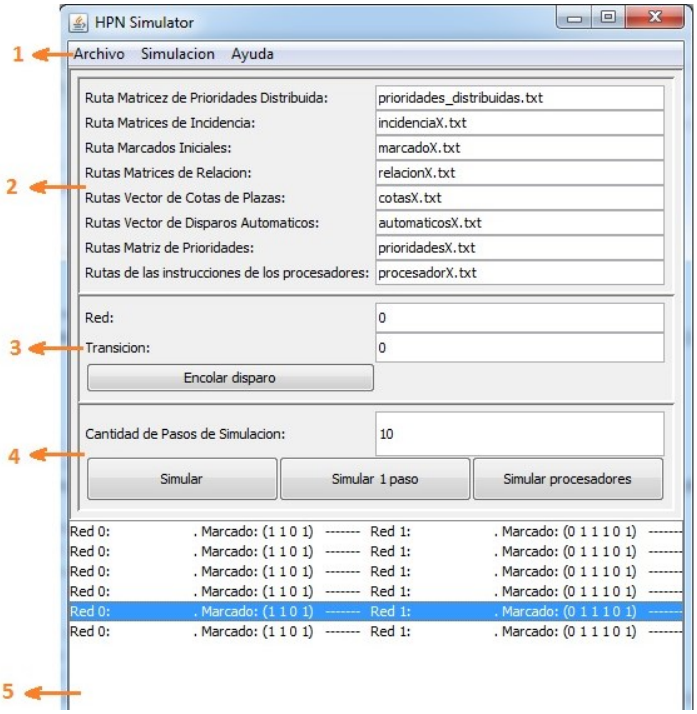


Figura 149: Ventana principal del simulador

Cuadro de especificación de rutas de archivos de configuración

En este cuadro se especifica el nombre y la ruta de los archivos para configurar el simulador. Estos archivos especifican el sistema.

Para la determinación de los vectores y matrices, se crean archivos de texto, separando los elementos de cada fila por cualquier número de espacios en blanco. Los vectores del sistema, se definen como vectores columnas. En la Figura 150 se muestra un ejemplo de un archivo que especifica una matriz de incidencia.

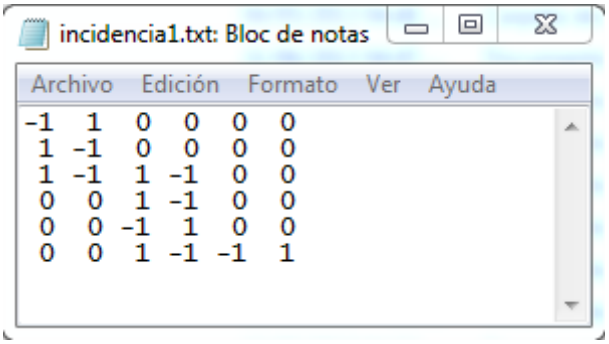
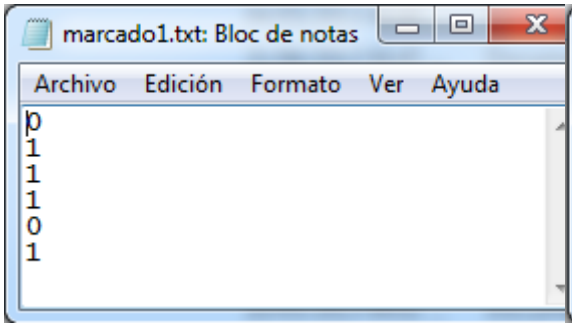


Figura 150: Archivo para especificar la matriz de incidencia

La especificación de un vector de marcado inicial se muestra en la Figura 151.

Por último en la Figura 152 se muestra la ventana para detallar el comportamiento de los distintos procesadores que se relacionan con el HPP. Se puntualiza cada proceso del sistema por un archivo.



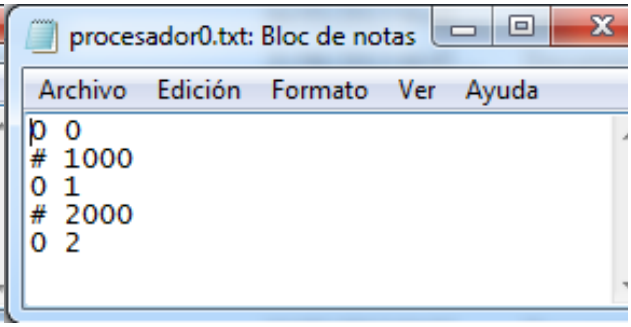


Figura 151: Especificación de un vector de marca **Figura 152: Especificación de comandos de un procesador**

El contenido de cada uno de estos archivos son las secuencias de comandos que cada procesador ejecuta, cuando está corriendo en la simulación. Estos comandos pueden ser de dos tipos:

1. Solicitud de disparo, se especifica con una línea que contiene dos valores enteros separados por cualquier número de espacios en blanco. El primer número corresponde a la subred del sistema donde se solicita el disparo y el segundo valor corresponde a la transición.
2. Espera, el hilo que representa al proceso se queda simulando la ejecución, para lo cual se produce una demora de un número especificado de milisegundos. Este comando se escribe en una línea que empieza con el carácter "#", seguido por un número arbitrario de espacios en blanco, y el número entero son los milisegundos de espera (esto es para simular la ejecución de una acción).

Cuadro de solicitud de disparo

Este cuadro permite configurar la solicitud de ejecución de una transición, independientemente de las solicitudes realizadas por la simulación. Para esto se introduce el número de subred y la transición. Esto detalla el disparo que se quiere encolar en la cola de entrada de la subred. Una vez configurado este campo, se continúa con la ejecución, mostrando el sistema un cuadro de diálogo donde la transición se encoló correctamente.

Una vez encolados los disparos necesarios, el usuario puede continuar controlando la simulación paso a paso. De esta forma verá la evolución de las subredes con los nuevos disparos encolados.

Es importante tener en cuenta que para usar esta característica es necesario que se haya iniciado una nueva simulación.

Cuadro de Simulación

En esta región se tiene el control de la simulación del sistema. Este cuadro tiene el campo "Cantidad de Pasos de Simulación", aquí se especifica el número de ciclos de simulación que se ejecutarán simultáneamente, presionado "Simular".

Este cuadro tiene tres comandos:

- Simular, ejecuta simultáneamente la cantidad de pasos especificados en el cuadro "Cantidad de Pasos de Simulación".
- Simular un paso, ejecuta sólo un paso de simulación.
- Simular procesadores, ejecutan los procesos especificados en los archivos de configuración.

Salida de la simulación ejecutados

Esta salida contiene la lista generada por cada ciclo de simulación que fue ejecutado la última vez que se inicializó la simulación, mediante el botón “Nuevo” o con el atajo “Ctrl + N”.

En la lista se especifica la transición ejecutada por cada una de las subredes del nuevo marcado y para cada subred tras la ejecución de la transición.

Ejemplo de Simulación

Para este ejemplo se usa el caso de 2 escritores sobre una variable compartida. Cuando un proceso accede a esta variable ejecuta una exclusión mutua. El modelo del sistema se realizó con una RdP, como se muestra en la Figura 153.

Para la escritura, el proceso uno solicita el disparo de T0 y el proceso dos solicita el disparo de T3. El permiso de escritura se obtiene por la realización del disparo. Cuando el proceso uno termina de escribir solicita el disparo de T1, mientras que el proceso dos solicita el disparo de T4.

Un token en la P1 indica que el proceso uno está escribiendo, mientras que un token en P4 indica que escribe el proceso dos.

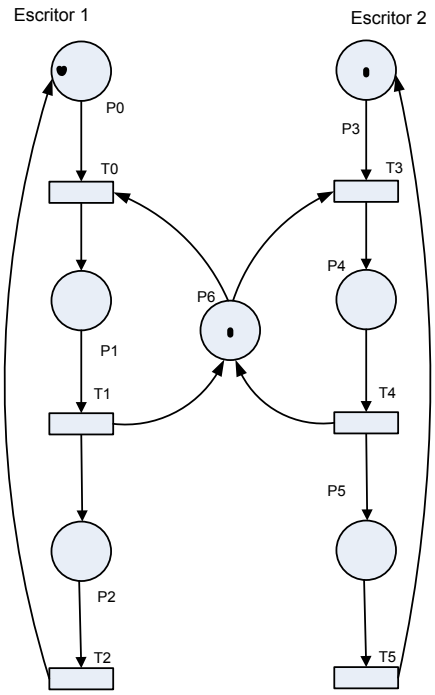


Figura 153: RdP para modelar dos escritores

Para este ejemplo, se divide la RdP en tres subredes. La Figura 154 muestra la división realizada. Con esta división se obtiene una reducción de recursos del 28%.

También se han colocado las etiquetas, para relacionar las transiciones con los procesos y definir el comportamiento de las transiciones.

Hacemos notar que las transiciones de borde tienen más prioridad que las que pertenecen sólo a la subred.

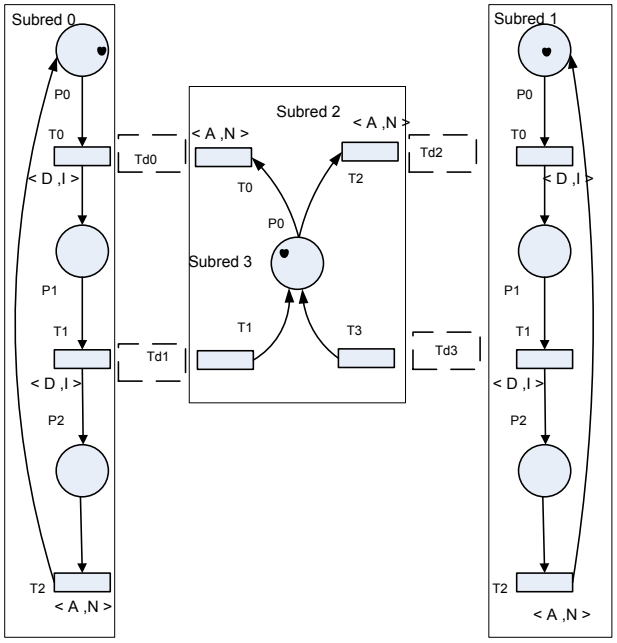


Figura 154: División de la RdP que modela a dos escritores.

Las prioridades de este ejemplo han sido tomadas por el número de la transición y la matriz de prioridad es la identidad en cada subred. Esta matriz es configurada como se muestra en la Figura 155.

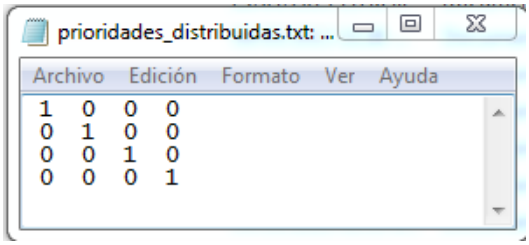


Figura 155: Matriz de prioridad

La programación del simulador, se realizó con archivos de texto que resultan de la RdP dividida y se muestran en la Figura 156, la Figura 157 y la Figura 158.

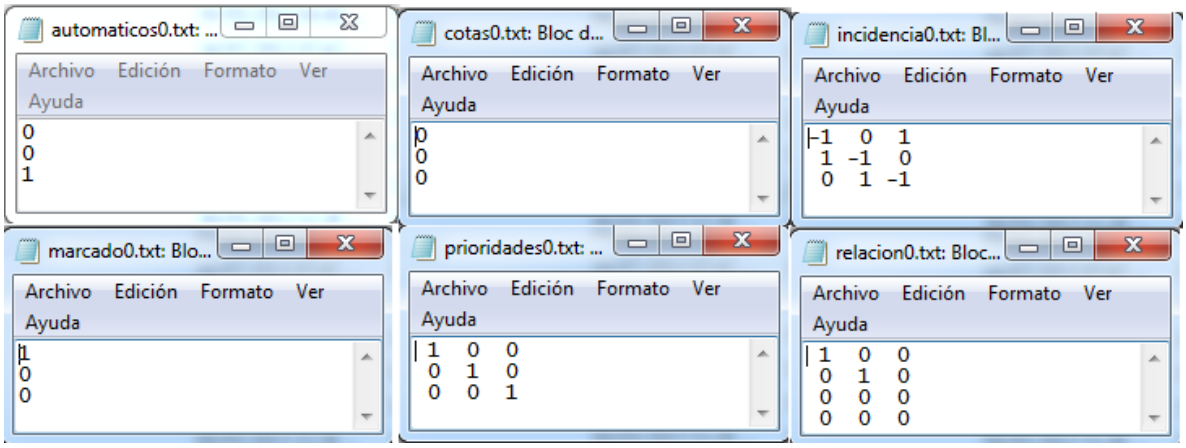


Figura 156: Archivos para programar la subred cero

Figura 157: Archivos para programar la subred uno

Figura 158: Archivos para programar la subred dos

Para este sistema se ha considerado que hay dos procesos de escritura. El proceso cero está relacionado con la subred 0 y el proceso uno está relacionado con la subred 1.

El primer proceso, subred 0, realizará 3 accesos a la variable compartida, y tardará en cada acceso 1000 milisegundos. Por otro lado, el proceso que se corresponde con la subred 1, realizará 6 accesos a la variable compartida, tardando en cada estado 500 milisegundos. El archivo de configuraciones de los procesos es mostrado en la Figura 159.

.

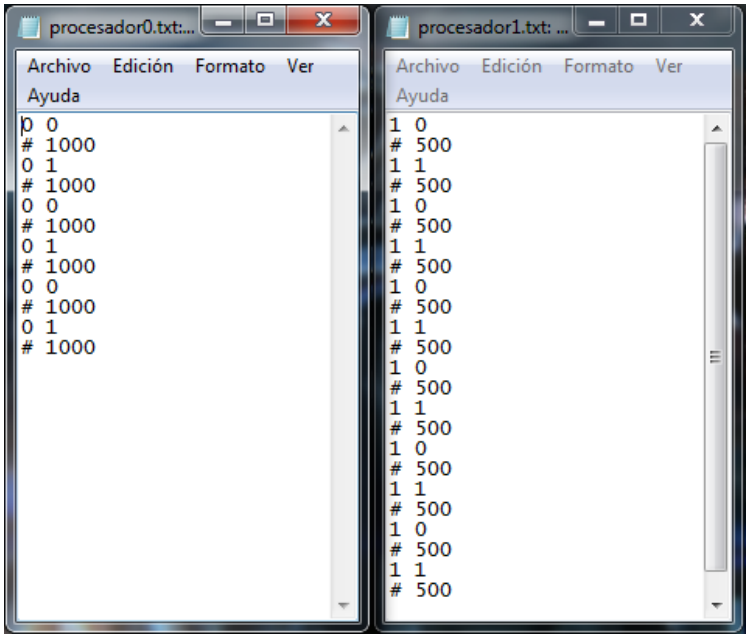


Figura 159: Archivo de comando de los procesos de escritura

Por último, se muestra en la Figura 160, los resultados de la simulación.

En la lista de ciclos de simulación, se ve como el sistema evoluciona. Se verifica que el proceso 1 realiza tres accesos a la variable compartidas (cuando su marcado está en el estado 0 1 0). De la misma manera, el proceso 2 realiza 6 accesos a la variable compartida. También, se verifican el paralelismo de la ejecución de sistema, en la línea marcada en la lista. Las tres subredes del sistema ejecutan una transición en el mismo ciclo. La subred 0 ejecuta la transición dos como una transición interna. La subred 1 ejecuta la transición 0 que es de borde. Por último la subred 2 ejecuta la transición 2 que es de borde. La ejecución de las transiciones de borde de la subred 1 y 2 de forma simultánea, responde al hecho de que en ese ciclo se ejecutó la transición distribuida número 2.

Implementación en Hardware del HPP

En esta sección se muestra una implantación en hardware de HPP. La implementación ha sido realizada como un IP-Core, e integrada en un sistema SMP, compuesto por mutiles MicroBlaze. La interacción entre los procesadores y el HPP se realiza usando el bus del sistema, que para este caso es un BUS AXI.

El HPP ha sido descripto en Verilog, que es un lenguaje de descripción de hardware (HDL). Este lenguaje Verilog facilita la manipulación de arreglos y registros de más de dos dimensiones. Esta

característica simplifica el manejo que se realiza, puesto que el HPP está basado en vectores y matrices.

En las siguientes secciones, se presentará una arquitectura posible, para ser usada en sistemas embebidos. Luego se presentará la convención usada, de las palabras para la programación del HPP. Finalmente se muestran las pruebas realizadas para evaluar el HPP.

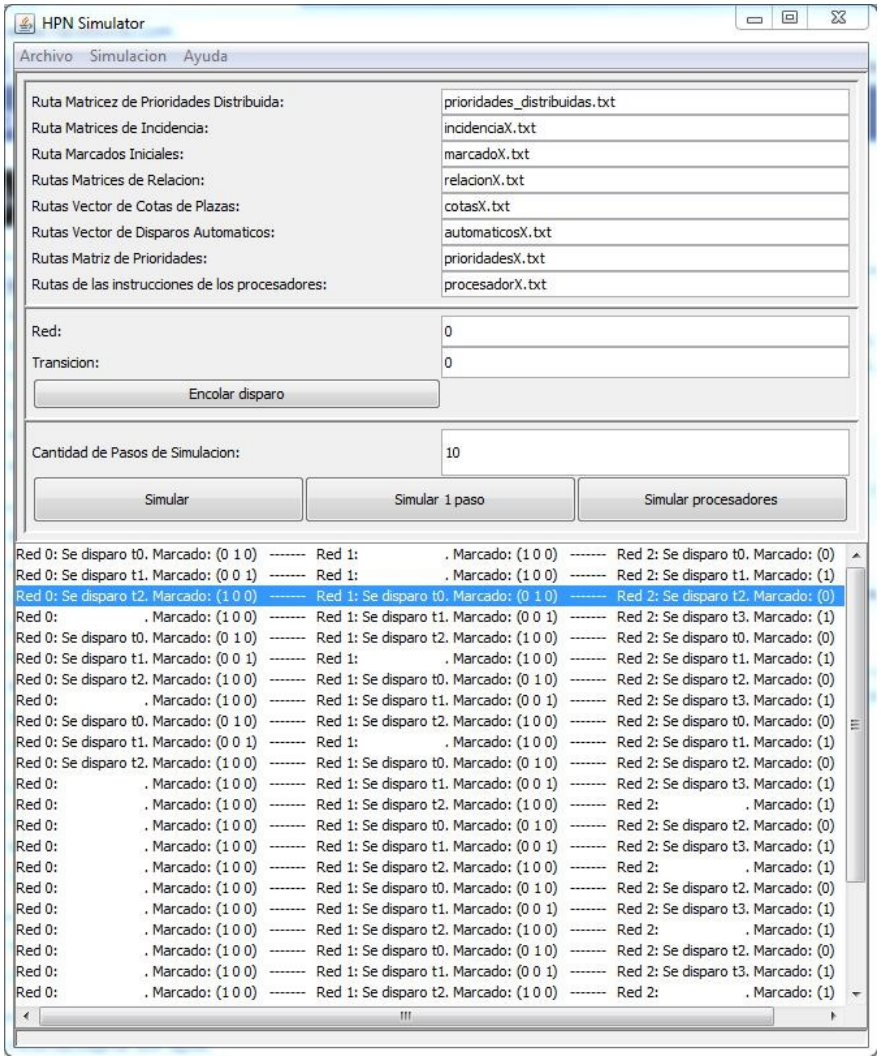


Figura 160: Resultados de la simulación

Arquitectura del Sistema

En primer lugar, es preciso acceder al HPP desde cualquier otro procesador que constituya el sistema, por los que ha seleccionado el protocolo AXI para la interconexión.

En la Figura 161, se muestra el modelo de interconexión entre un HPP, múltiples MicroBlazer y otros dispositivos. Esta interconexión conforma un sistema multi-Core heterogéneo.

Como se observa, el bus de interconexión implementa el protocolo AXI, que facilita la edición de otros dispositivos.

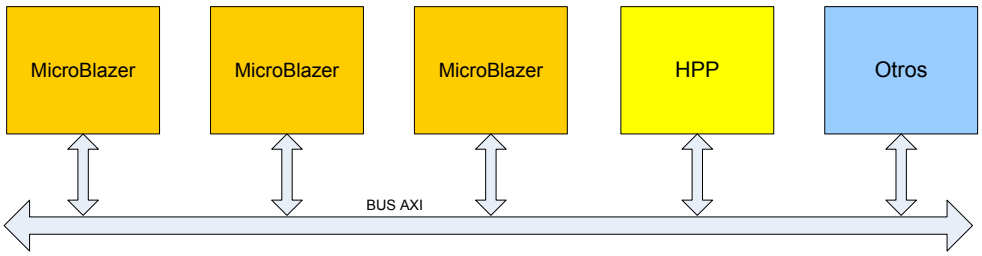


Figura 161: Arquitectura de interconexión del HPP con un sistema SMP

Arquitectura del HPP

Esta sección muestra una arquitectura posible para la implementación del HPP, ver Figura 162. Para la ejecución independiente de cada subred de RdP, se dividió a cada subred en tres áreas como lo muestra la Figura 162. Éstas son: "Área de Comunicación de la subred con los procesadores", "Área de Datos de la subred" y "Área de Ejecución de la subred".

En primer lugar, el área de comunicación de la subred con los procesadores tiene la misma responsabilidad que en el PP. Éstas son las comunicaciones de los eventos, haciendo uso de las colas, y la programación de las matrices y vectores que son almacenados en el área de datos de la subred. Es decir, la comunicación entre cada subred y el sistema conformado por el sistema SMP. Para este caso se hace uso del bus que implementa el protocolo AXI.

La responsabilidad del área de datos de la subred es almacenar los datos que se mantienen durante la ejecución del programa y son el programa del HPP.

Mientras que el módulo área de ejecución de la subred, calcula todos los vectores y matrices que requiere el HPP para la ejecución de cada subred.

Estos módulos son instanciados tantas veces como subredes de la RdP existan en el HPP.

Finalmente, el módulo "Módulo común a todas las subredes" obtiene los vectores de transición distribuidos para calcular el disparo de las transiciones globales. Estos vectores y los disparos son comunicados por el "Bus de interconexión entre procesadores". De esta forma se determina cada nuevo estado del HPP.

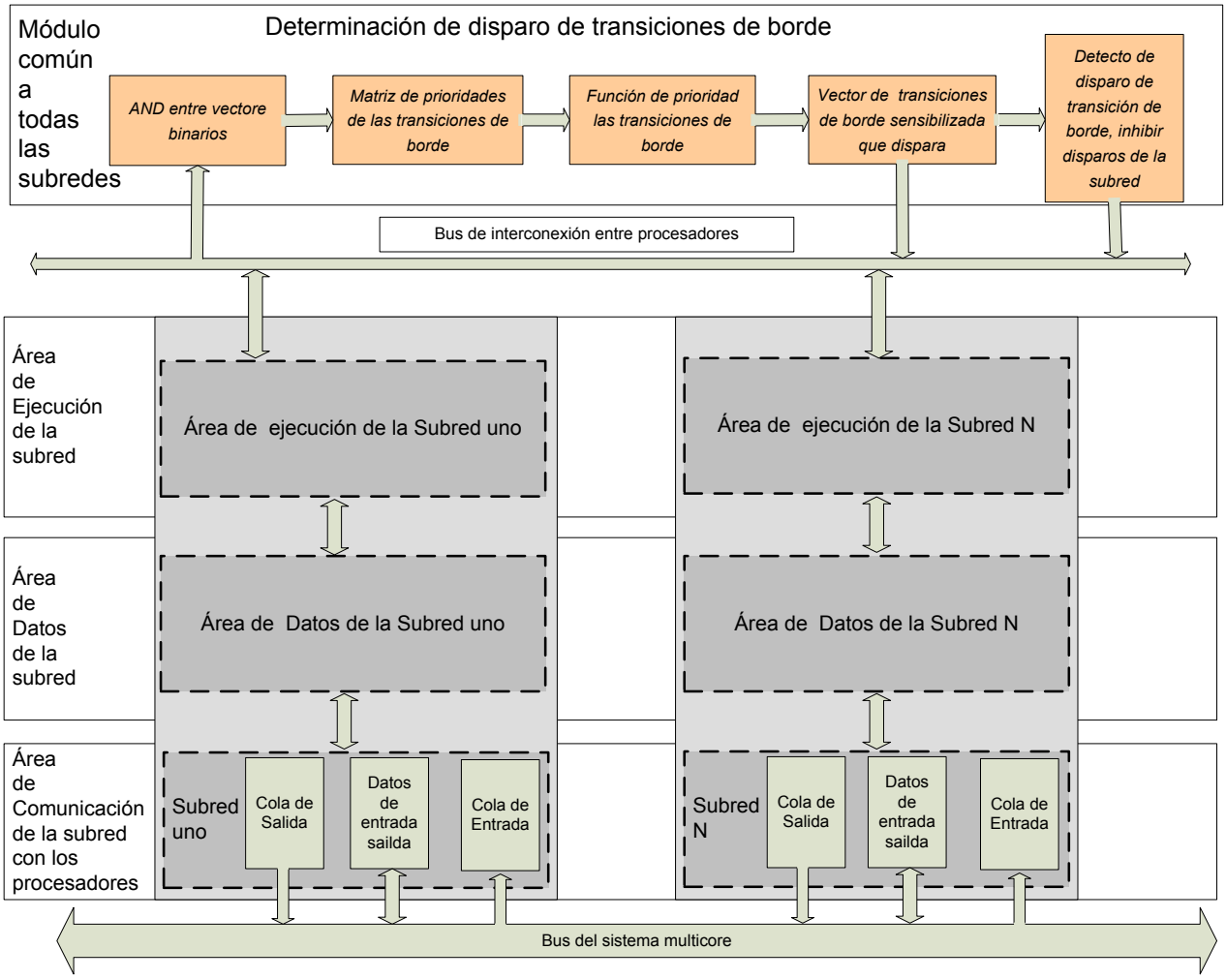


Figura 162: Interconexión del HPP interna y con el bus del SMP

En la Figura 163, se muestra el diagrama de la arquitectura de interconexión del HPP. En ella se resalta las modificaciones y los registros que se agregaron, que son: registros en el módulo de cálculo de la ecuación de estado, cálculo del vector de transiciones de borde sensibilizada que dispara y la matriz de relación entre transiciones.

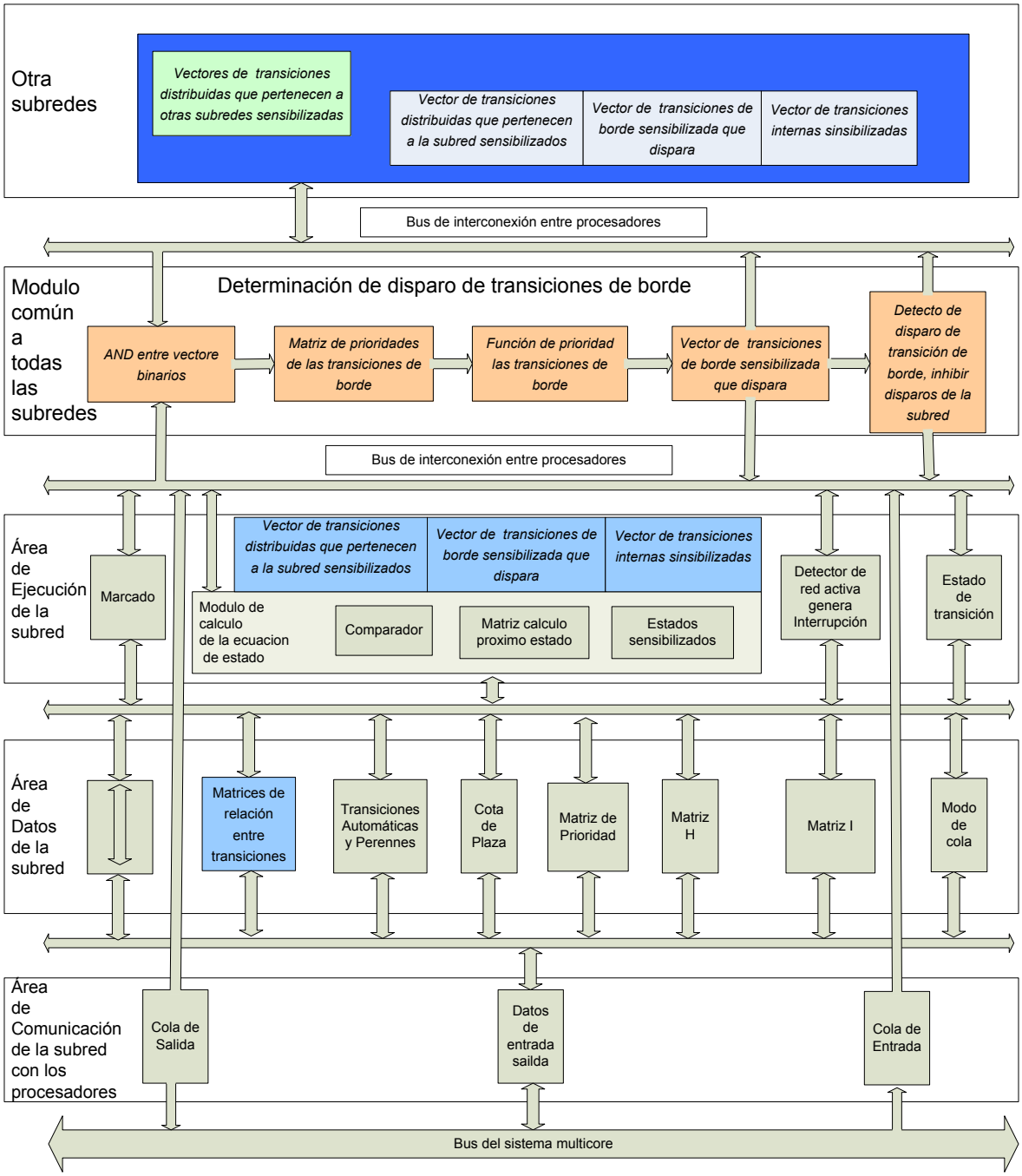


Figura 163: Arquitectura de interconexión interna del HPP

Comunicación entre Procesos y HPP

Para la comunicación entre el HPP y los procesos del usuario, se implementó un registro de configuración de 32 bits. Este registro es enviado a través del bus de datos AXI. Mientras que, para las operaciones de lectura y escritura del HPP se ha mapeado en una serie de registros, que son accesibles desde el software de usuario. Estos registros tienen una dirección base, en el HPP, que es generada cuando se sintetiza el sistema embebido. Todos estos registros tienen una dimensión de 32 bits. Por último, para el manejo de eventos como interrupción, se instanció un

módulo de interrupciones. Este módulo, tiene la capacidad de generar interrupciones por disparo de transición e inactividad.

Registros de escritura del HPP
En este apartado se describen los registros utilizados en la escritura:

Registro de configuración y control, este es el único registro del HPP que es accesible por el usuario. Está mapeado en la dirección base del HPP. Escribiendo la palabra de configuración en este registro, se realizan todas las operaciones de configuración y control necesarias del HPP.

En los apartados que continúan se especifica la palabra de configuración del registro, para programar y controlar el HPP:

Palabra de configuración del HPP

Esta palabra se escribe siempre en el registro de control, que está en la dirección base del mapa del HPP, en el sistema embebido. A continuación se presentan los campos que constituyen la palabra de configuración, que son:

Tabla 111: Campos de la palabra de configuración del HPP

Red	Matriz o vector	Fila	Columna	Valor
5 bits	5 bits	7 bits	7 bits	8 bits

A continuación, se describen la función de cada uno de los campos de la palabra de configuración.

Campo de Red

Este campo identifica la subred, en el proceso de carga de las matrices y vectores, o de solicitud de ejecución de una transición. Puesto que es preciso especificar a qué subred del sistema se está accediendo, el campo es de 5 bits.

También, se usa cuando se realiza una consulta del valor de una plaza o de una transición ejecutada. Puesto que es necesario identificar la subred que se consulta.

Las subredes del sistema se enumeran secuencialmente, a partir de la subred cero, hasta la 31.

Campo Matriz o Vector

Este campo, identifica al módulo que se escribe o lee en el área de datos o a la cola en el área de comunicaciones, y tiene 5 bits.

Por ejemplo, si el campo se corresponde con la "cola de entrada" y se escribe, se estará realizando la solicitud de ejecución de una transición encolándola en la cola de entrada de la subred especificada. Otro ejemplo es que en la palabra de configuración se encuentre la constante que se corresponde con "matriz de incidencia" en el campo matriz o vector. En este caso se estaría accediendo al HPP para cargar un elemento de la matriz de incidencia de la subred.

Campos Fila, Columna y Valor

El valor (dato) define este campo y sus coordenadas son las definidas en los campos fila y columna. A continuación, se enumeran las constantes definidas para el sistema:

- INCIDENCIA, esta palabra de configuración carga un valor en la matriz de incidencia de la subred. El elemento de la matriz a cargar es el indicado por los campos Fila y Columna. El dato que se carga es el indicado en el campo Valor.
- INHIBICION, esta palabra de configuración carga un valor en la matriz de arcos inhibidores de la subred. El elemento de la matriz a cargar es el indicado por los campos Fila y Columna. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- PRIORIDAD, esta palabra de configuración carga un valor en la matriz de prioridades de la subred. El elemento de la matriz a cargar es el indicado por los campos Fila y Columna. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- RELACION, esta palabra de configuración carga un valor en la matriz de relaciones con transiciones distribuidas de la subred. El elemento de la matriz a cargar es el indicado por los campos Fila y Columna. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- PRIORIDADAD, esta palabra de configuración, carga un valor en la matriz de prioridades de transiciones distribuidas del sistema. El elemento de la matriz a cargar es el indicado por los campos Fila y Columna. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- MARCADO, esta palabra de configuración, carga un valor en el vector de marcado inicial de la subred. El elemento del vector a cargar es el indicado por el campo Fila. El dato que se carga es el indicado en el campo Valor.
- COTAS, esta palabra de configuración carga un valor en el vector de cotas de plazas de la subred. El elemento del vector a cargar es el indicado por el campo Fila. El dato que se carga es el indicado en el campo Valor.
- AUTOMATICOS, esta palabra de configuración carga un valor en el vector de transiciones automáticas de la subred. El elemento del vector a cargar es el indicado por el campo Fila. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- No-PERENNE, esta palabra de configuración, carga un valor en el vector de transiciones no-perenne de la subred. El elemento del vector a cargar es el indicado por el campo Fila. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- NO_INFORMADAS, esta palabra de configuración carga un valor en el vector de transiciones que no informan de la subred. El elemento del vector a cargar es el indicado por el campo Fila. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- INTERRUPCIONES, esta palabra de configuración carga un valor en el vector de máscara de interrupciones de la subred. El elemento del vector a cargar es el indicado por el campo Fila. El dato que se carga es el indicado en el bit menos significativo del campo Valor.
- COLA_ENTRADA, esta palabra de configuración solicita la ejecución explícita de una transición (evento), encolándola en la cola de entrada de la subred. La transición a ejecutar es la indicada en el campo Fila.
- CONSULTA_DISP, esta palabra de configuración realiza una consulta para determinar si una transición dada fue o no ejecutada por el HPP. El resultado de la consulta estará disponible en el siguiente ciclo de reloj del sistema, y podrá ser leída desde el registro correspondiente a la posición base del HPP. El campo Red especifica la cola de salida de

la subred que se está consultando. El campo Fila especifica la transición que se está consultando. De ser exitosa la consulta, la transición consultada es removida de la cola de salida correspondiente. Por último, cuando se lee la consulta, el bit menos significativo del campo Valor indica con un 1 si la transición fue ejecutada.

- CONSULTA_DISP, la consulta leída se refiere a una transición.
- CONSULTA_PLAZ, esta palabra de configuración realiza una consulta para determinar el valor de una plaza dada en la subred especificada. El resultado de la consulta estará disponible en el siguiente ciclo de reloj del sistema, y podrá ser leída desde el registro correspondiente a la posición base del HPP. El campo Fila especifica la plaza que se está consultando. Por último, cuando se lee la consulta, el campo Valor indica el número de tokens de la plaza consultada.
- CLEAR, esta palabra de configuración realiza un Clear síncrono del HPP. En el ciclo que sigue a la escritura de esta palabra, se pondrán a 0 todos los registros internos del IP-Core. Todas las matrices, vectores, colas de entrada y salida serán puestos a cero. Pese a que el protocolo del bus contempla un reset asíncrono al inicio de la operación del sistema, se recomienda realizar manualmente un Clear al HPP mediante esta palabra de configuración.
- RESTART, esta palabra de configuración realiza una copia del marcado inicial al marcado actual. Esto provoca que todas las subredes de RdP del sistema vuelvan al estado inicial, pero manteniendo los valores cargados en los vectores y matrices.
- RUN, esta palabra de configuración, pone en estado de ejecución (Run o Corriendo) al HPP.
- STOP, esta palabra de configuración pone en estado de Stop o Espera al HPP. En este estado el procesador no ejecutará ninguna transición explícitamente pedida o automática. Sin embargo, se pueden cargar o modificar todos los vectores y matrices que configuran el sistema. Además, se pueden encolar disparos y realizar consultas de plazas o transiciones. Hay que tener en cuenta que si se realiza una modificación en el vector de marcado inicial de alguna subred, antes de poner nuevamente en ejecución al IP-Core mediante la palabra de configuración RUN, hay que enviar la palabra de configuración RESTART para que el marcado inicial modificado sea copiado al marcado actual. Este estado es útil para hacer modificaciones en tiempo de ejecución sobre el sistema.

Los valores o constantes definidos en el campo Matriz o Vector, determinan la función de la palabra de configuración. De esta forma, en función de los valores de este campo, podemos interpretar a la palabra de configuración:

- Carga de una matriz (constantes INCIDENCIA, INHIBICION, PRIORIDAD, RELACION y PRIORIDAD_D). Con estas palabras de configuración se están cargado elementos en las matrices que configuran las subredes. La Tabla 112, muestra la interpretación de los campos.

Tabla 112: Interpretación de los campos de la palabra de configuración para una matriz

Red	Matriz a cargar	Fila	Columna	Valor
5 bits	5 bits	7 bits	7 bits	8 bits

- Carga de un vector (constantes MARCADO, COTAS, AUTOMATICOS, NO_INFORMADAS e INTERRUPCIONES). Con estas palabras de configuración se

están cargando elementos en los vectores que configuran las distintas subredes del sistema. La Tabla 113 muestra la interpretación de los campos.

Tabla 113: Interpretación de los campos de la palabra de configuración para un vector

Red	Vector a cargar	Elemento del Vector	Ignorado	Valor
5 bits	5 bits	7 bits	7 bits	8 bits

- Solicitud de un disparo (constante COLA_ENTRADA). Con esta palabra de configuración se solicita explícitamente la ejecución de una transición encolándola en la cola de entrada de la subred a la que pertenece. La Tabla 114, muestra la interpretación de los campos.

Tabla 114: Interpretación de los campos de la palabra de configuración para un disparo

Red	COLA_ENTRADA	Transición	Ignorado	Ignorado
5 bits	5 bits	7 bits	7 bits	8 bits

Registros de lectura del HPP

Los registros de lectura están mapeados a partir de la dirección base del HPP. A continuación se exponen los registros para la lectura del HPP.

Con la lectura de este registro, se obtiene los resultados de las operaciones realizadas por el HPP, sean consultas de plazas o de transiciones ejecutadas. La respuesta a la consulta estará disponible para ser leída en el ciclo siguiente a la escritura de la consulta.

Para consultar una transición, se leerá en el campo valor un 1 si la transición consultada fue ejecutada, y el HPP la retira de la cola de salida a la que pertenecía. Si la consulta era de plaza en el campo Valor se leerá la cantidad de tokens que tenía la plaza consultada en el momento de realizar la consulta.

La palabra de 32 bits leída en este registro, tiene el significado mostrado en la Tabla 115.

Tabla 115: Registros de lectura del HPP

Red consultada	Tipo consulta	Plaza o transición consultada	Ignorado	Valor de consulta
5 bits	5 bits	7 bits	7 bits	8 bits

Las direcciones de los registros de lectura de colas de salida son definidas en función de la dirección base del HPP más uno más el número de cola (o transición) de la subred. Para este caso se han definido 31 registros de lectura de colas de salida. Cada uno de estos registros, se corresponde con el estado de las colas de salida de las primeras 31 subredes del sistema. En la dirección base + 1 del HPP se lee directamente el estado de la cola de salida de la subred 0, en la dirección base + 2 el estado de la cola de salida de la subred 1, y así hasta llegar a la dirección base + 31 donde se leerá el estado de la cola de salida de la subred 30.

Los registros leídos son de 32 bits, correspondiéndose cada bit con una transición de la subred accedida. De esta forma el bit menos significativo de la palabra leída representa el estado de la transición 0 de la subred en cuestión, y el bit más significativo representa el estado de la

transición 31 de la subred. Cada vez que se lee una de estas colas de salida, se resta una unidad de cada transición encolada en la cola de salida accedida, puesto que se están consultando de forma simultánea todas las transiciones de la subred.

Un 1 en el bit correspondiente a una determinada transición, significa que dicha transición fue disparada por el HPP, por el contrario un 0 indica que dicha transición no fue ejecutada.

El ancho de 32 bits del bus de datos, nos limita la cantidad de transiciones que pueden ser consultadas de esta forma. Por esto y para este caso, cada subred de RdP del sistema tiene un número máximo de 32 transiciones que pueden ser consultadas leyendo directamente las colas de salida del sistema.

La utilización de esta característica para consultar todas las transiciones de una subred de forma simultánea, es útil en la implementación del driver [241], para manejar con una sola lectora 32 transiciones.

Interrupciones del HPP
Para poder manejar eventos con retardos por encima de los tiempos necesarios para los cambios de contexto, en forma eficiente, se incorpora un módulo de interrupciones en el HPP.

En esta implementación, se ha definido una interrupción general para todo el HPP y una específica por cada subred.

Los eventos que provocan las interrupciones son los disparos de transiciones. Por defecto cada transición que ejecuta el sistema provoca la interrupción general y la interrupción propia de la subred a la que pertenece. Para filtrar, los disparos de las transiciones que no provocan interrupción, se programa la máscara de interrupciones.

La interrupción general es provocada cuando el procesador ejecute cualquier transición no enmascarada, independientemente de la subred.

Para implementar estas características, y que el sistema embebido pueda reconocerlas, se conecta un controlador de interrupciones. Este controlador, se comunica con los procesadores, también es necesario que el software las configurare, habilite y atienda.

En la Figura 164, se muestra el detalle de la conexión realizada en la implementación del sistema de interrupciones.

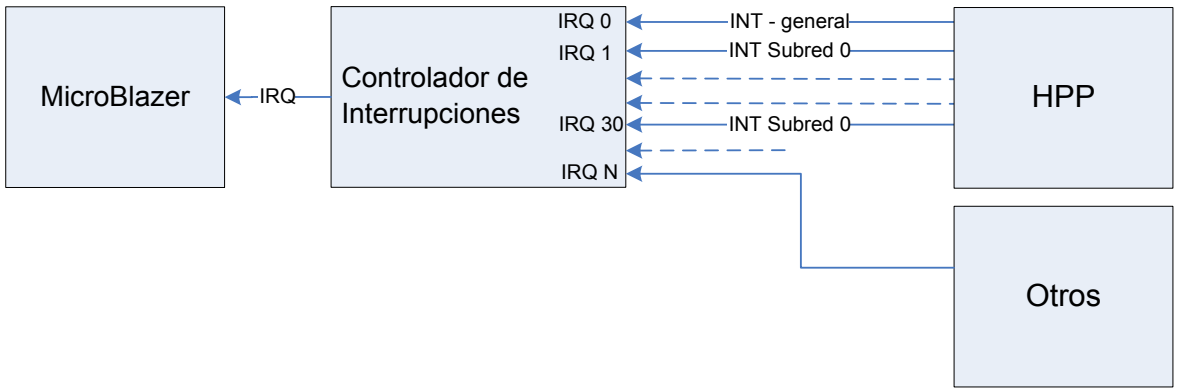


Figura 164: Esquema de interconexión de las subredes, el controlador de interrupciones y el MicroBlazer

Evaluación del Hardware del HPP

Cumplida la implementación del hardware y librería, para la ejecución y programación del HPP, se realizó la evaluación del sistema. El factor de mayor importancia, a evaluar en este apartado, es el crecimiento del HPP. Esto se realizó con distintas cantidades de plazas, transiciones, transiciones distribuidas y subredes.

Los resultados, de las distintas instancias sintetizadas del HPP, han sido graficados, lo que permite evaluar y proponer distintos aspectos y modificaciones del HPP. También se han realizado pruebas de funcionamiento y evaluación de desempeño.

Para esto se han medido los tiempos, con el HPP y semáforos, todo corriendo sobre el sistema operativo Xilkernel. En las siguientes secciones se detallan las pruebas, resultados y conclusiones obtenidos.

Estudio de Crecimiento del Hardware

Para todas las mediciones y pruebas se ha usado el Kit de desarrollo de Digilent Atlys, que está basado en una FPGA Spartan-6 LX45.

Con la implementación del HPP, se determinan los recursos y rendimiento del sistema, para lo cual contestamos las siguientes preguntas:

- ¿Cuánta es la demanda de recursos, según el número de transiciones |T|, plazas P|, transiciones distribuidas y subredes?
- ¿Por la división del sistema en subredes, se obtienen otros beneficios?
- ¿Cuál es la cantidad de plazas y transiciones óptimas o sub-óptimas que deberían tener las subredes del sistema para aprovechar mejor el hardware?

Para cada instancia sintetizada se obtiene: la cantidad de LUTs, los registros que emplea la FPGA y la frecuencia máxima de operación del HPP. Estos valores son tabulados y graficados, para el análisis de los resultados.

Para distinguir las características de cada instancia, se las llama de la siguiente manera: subredes x plazas x, transiciones x y transiciones distribuidas.

Así por ejemplo si tenemos un HPP con 4 subredes, cada subred con 6 plazas, 8 transiciones y 7 transiciones distribuidas, la etiqueta del HPP es 4x6x8x7.

Variación del número de subredes

Como primera prueba dejamos fijo en 8 el número de plazas y transiciones de cada subred, así como también el número de transiciones distribuidas del sistema, y fuimos variando la cantidad de subredes entre 2 y 8. La Tabla 116, muestra los resultados obtenidos para las distintas síntesis. Posteriormente, en la Figura 165, se grafican los resultados de la tabla.

Tabla 116: Recursos en función de las subredes, con plazas, transiciones y transiciones de borde constantes

Subredes	IP-Core HPP	Registros	LUTs	Frecuencia máx. (MHz)
2	2x8x8x8	2179	7010	34.892
3	3x8x8x8	3227	10856	33.958
4	4x8x8x8	4275	14383	34.203
5	5x8x8x8	5323	17618	33.834
6	6x8x8x8	6371	19669	34.334
7	7x8x8x8	7419	20939	33.697
8	8x8x8x8	8467	27865	34.225

La Figura 165, muestra que el crecimiento de registros y LUTs, manteniendo la cantidad de elementos de cada subred constante. En las absisas se muestra elnúmero de subredes y elementos por subred.

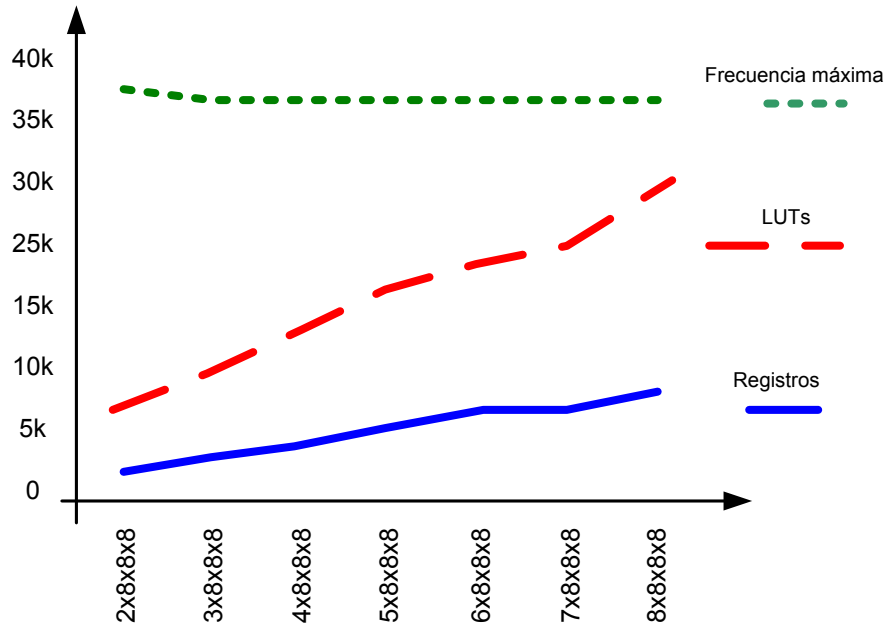


Figura 165: Crecimiento de los LUTs y frecuencia con el aumento de subredes

Si bien no se ha realizado un refinamiento en la síntesis para obtener la máxima frecuencia, ésta se mantiene entre 33 MHz, para todas las instancias.

Variación de los recursos con el número de transiciones distribuidas
Ahora, instanciamos distintos HPP, mantenemos constante el número de subredes en 4, el número de plazas y transiciones en ocho, para cada subred.

Tabla 117: Recursos en registros y LUTs según como varían las transiciones distribuidas

Transiciones de borde	IP-Core HPP	Registros	LUTs	Frecuencia máx. (MHz)
1	4x8x8x1	3988	13769	47.759
2	4x8x8x2	4023	13477	43.066
3	4x8x8x3	4060	13877	40.158
4	4x8x8x4	4099	14054	38.373
5	4x8x8x5	4140	14077	36.967
6	4x8x8x6	4183	14209	36.449
7	4x8x8x7	4228	14323	36.453
8	4x8x8x8	4275	14383	34.203

En la Figura 166 y la Tabla 117 se han obtenido variando la cantidad de transiciones de borde entre 1y 8.

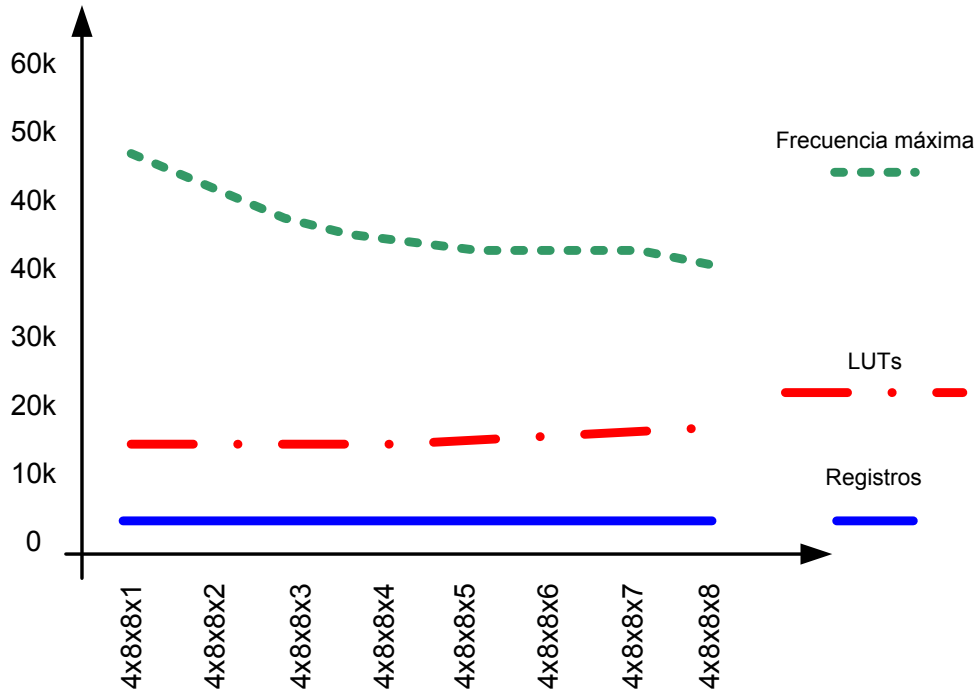


Figura 166: Recursos en registros, LUTs y frecuencia según varían las transiciones distribuidas

Como vemos en la Figura 166, los recursos empleados en todas las instancias varían muy poco. Esto es debido a que toda la lógica, para solucionar las transiciones distribuidas, ha sido

realizada con vectores y funciones binarias (no hay operaciones entre enteros). El agregado de una transición distribuida suma una fila en la matriz de relación con transiciones distribuidas de cada subred. Puesto que estas matrices son binarias y las operaciones necesarias para obtener las transiciones sensibilizadas son and y or, el aumento de recurso es impactado por el aumento de estas matrices y operaciones.

Para este caso no se ha realizado un refinamiento exhaustivo en la síntesis, para obtener la máxima frecuencia, pero aquí claramente se observa que baja la frecuencia de operación. En promedio se disminuye a 1,7MHz por cada transición distribuida que se incorpora. Esto se debe al algoritmo para determinar la sensibilidad de la transición distribuida a ejecutar, ya que se requiere de una lógica en cascada para seleccionar una única transición a ejecutar. De estos resultados podemos esperar que mientras mayor es el grado de acoplamiento del sistema, menor será la frecuencia máxima a la que podrá operar el HPP.

Variación de recursos variando el número de plazas de cada subred del HPP
Ahora se mantiene constante el número de subredes en 4, el número de transiciones de cada subred en 8 y la cantidad de transiciones distribuidas en 8. Se varía el número de plazas de cada subred entre 1 y 18. La siguiente Tabla 118 muestra los datos obtenidos de las distintas síntesis instanciadas del HPP.

Tabla 118: Variación de recursos y frecuencia del HPP, variando sólo las plazas

Número de plazas por subred	IP-Core HPP	Registros	LUTs	Frecuencia max. (MHz)
1	4x1x8x8	865	3779	60539
2	4x2x8x8	1225	4380	59756
3	4x3x8x8	1842	6973	42715
4	4x4x8x8	2210	7345	43896
5	4x5x8x8	3123	11507	33993
6	4x6x8x8	3507	12400	33950
7	4x7x8x8	3891	13304	34007
8	4x8x8x8	4275	14383	34203
9	4x9x8x8	4660	15523	33771
10	4x10x8x8	5065	17435	35077
11	4x11x8x8	5516	16816	33691
12	4x12x8x8	5833	17596	33150
13	4x13x8x8	6217	18758	33064
14	4x14x8x8	6646	20182	32642
15	4x15x8x8	7043	20387	31129
16	4x16x8x8	7431	20975	30621
17	4x17x8x8	7760	21887	33664
18	4x18x8x8	8171	22335	28235

Figura 167 y la Tabla 118 muestran la variación de recursos según varían las plazas. Que la variación sea lineal se debe a que cada plaza se traduce en una fila adicional de la matriz de incidencia y resultado, un lugar en el vector de marcado inicial y actual, un lugar en el vector de cotas de plazas, etc. Todos estos elementos no son binarios, sino que son elementos representados por 8 bits. Por esto el incremento es significativo en número de registros. Mientras que, el aumento de las LUTs es aún mayor que el de registros, ya que se generan sumadores, comparadores, restadores, etc.; cada uno de 8 bits por cada plaza adicional.

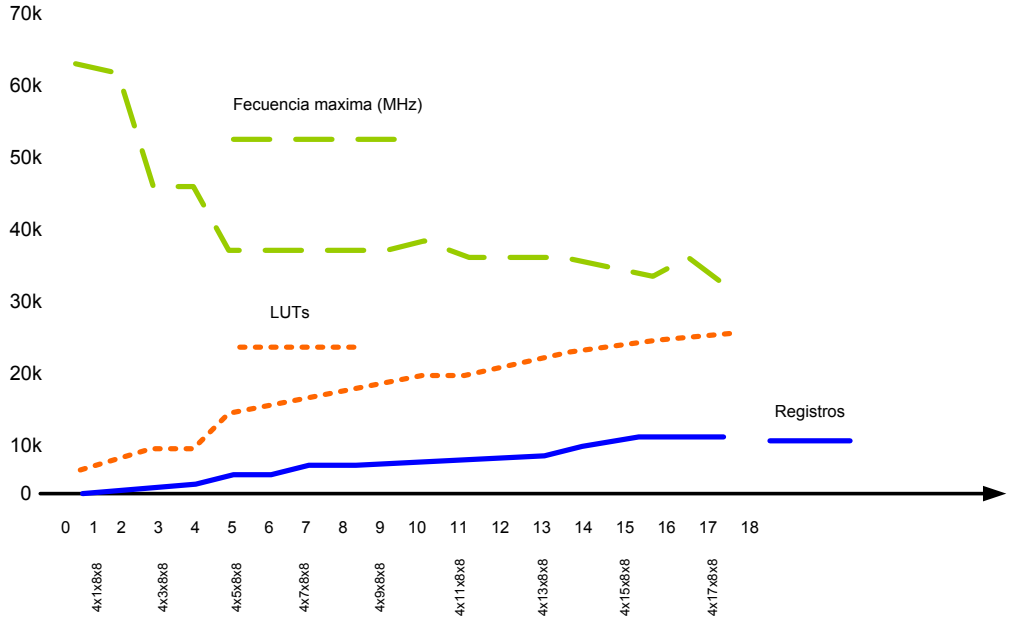


Figura 167: Variación de recursos y frecuencia del HPP, variando sólo las plazas

Con respecto a la frecuencia máxima de operación, vemos que ésta decrece de forma rápida al comienzo, pero luego se mantiene constante alrededor de 32MHz a medida que aumentamos la cantidad de plazas de cada subred. Esto se debe a que las plazas no pasan por el proceso de selección que sufren las transiciones distribuidas, y por ende la lógica que se genera para éstas se puede realizar en paralelo, haciendo que la frecuencia no disminuya de forma significativa con el número de plazas.

Variación de los recursos con la variación del número de transiciones de cada subred

Ahora se mantiene constante el número de subredes en 4, el número de plazas de cada subred en 8 y la cantidad de transiciones distribuidas en 8. Se varía el número de transiciones de cada subred entre 1 y 18. La Tabla 119 muestra los recursos necesarios para las distintas instancias del HPP.

Tabla 119: Variación de recursos y frecuencia del HPP, variando solo las transiciones

Numero de transiciones por subred	IP-Core HPP	Registros	LUTs	Frecuencia máx. (MHz)
1	4x8x1x8	1203	2470	65904
2	4x8x2x8	1611	3824	54259

3	4x8x3x8	2043	5756	43083
4	4x8x4x8	2473	7471	40226
5	4x8x5x8	2934	8793	35734
6	4x8x6x8	3371	10116	35752
7	4x8x7x8	3819	12347	34773
8	4x8x8x8	4275	14383	34203
9	4x8x9x8	4627	14549	32421
10	4x8x10x8	4979	15243	34009
11	4x8x11x8	5379	16961	33571
12	4x8x12x8	5762	17947	33179
13	4x8x13x8	6168	18437	32675
14	4x8x14x8	6532	18947	33716
15	4x8x15x8	6856	20030	31673
16	4x8x16x8	7225	20537	30137
17	4x8x17x8	7488	21364	31461
18	4x8x18x8	7907	21716	29062

Como se muestra en la Tabla 119 y la Figura 168 la variación de recursos y los cambios de frecuencia con el aumento de las transiciones son muy similares al del aumento de plazas en las subredes. El aumento de recursos es levemente inferior en el aumento del número de transiciones en comparación al aumento de plazas, debido a que nuevas transiciones generan algunos elementos que son binarios y no requieren 8 bits. Tal es el caso de las columnas adicionales en la matriz de relación con transiciones distribuidas, los elementos en los vectores de transiciones automáticas, las transiciones que no informan, la máscara de interrupciones, etc. Todo esto hace que el número de registros y LUTs necesarios para aumentar la cantidad de transiciones sea levemente inferior a los necesarios para aumentar el número de plazas en las subredes.

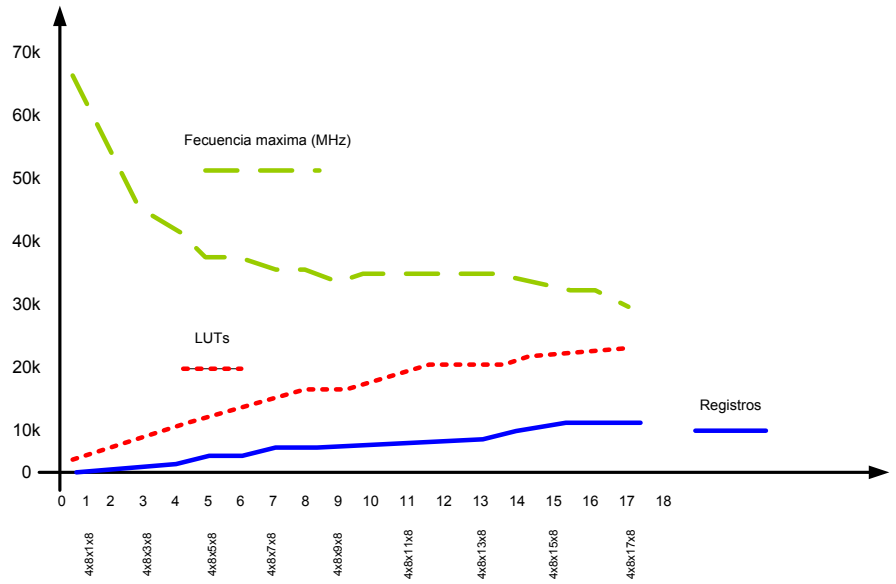


Figura 168: Gráfico de la variación de recursos y frecuencia del HPP, variando sólo las transiciones

Observaciones de la variación de recursos con los parámetros en la implementación del HPP
Las pruebas han sido realizadas en una plataforma de desarrollo de Digilent Atlys. Esta contiene una FPGA Spartan-6 LX45.

Los resultados obtenidos si no se hace la división, en cuanto a la variación de recursos en función de plazas y transiciones del HPP, son comparables a los obtenidos en la implementación del PP

Pruebas para Determinar el Desempeño

En esta sección se realizarán pruebas del funcionando del HPP, en un sistema embebido, con el fin de comparar el rendimiento de éste frente a la sincronización realizada con semáforos. Las pruebas han sido implementadas y corridas un el sistema operativo Xilkernel.

Las primeras mediciones realizadas tienen como objetivo determinar la cantidad de ciclos de reloj requeridos para las operaciones básicas como "or", "and", "suma de un registro", de sincronización, etc. Todo esto en el sistema Digilent Atlys / Xilkernel. Luego se midieron los tiempos del HPP para realizar distintas operaciones.

Finalmente se realizaron mediciones de tiempos totales para la ejecución de aplicaciones con sincronización, entre 4 hilos utilizando semáforos y el HPP para comparar el desempeño de los distintos sistemas.

En la medición de tiempos se ha empleado un IP-Core, que es AXI_TIMER, que se acopla al sistema embebido y permite medir los tiempos por hardware.

El AXI_TIMER, tiene un contador interno que se incrementa con cada clk del sistema. Desde el software de usuario se puede poner a cero, pausar, leer, etc. en cualquier momento. Esto permite medir exactamente cuántos clk del sistema transcurren entre dos lecturas.

Tiempos de ejecución de instrucciones en Xilkernel
En primer lugar medimos el tiempo de ejecución de instrucciones simples y tiempos para los accesos a variables en memoria. Posteriormente medimos los tiempos para realizar cambios de contexto en el sistema operativo. Seguidamente se presentan las pruebas y los resultados obtenidos.

Ejecución de instrucciones

Utilizando el AXI_TIMER, medimos la cantidad de clk que tarda la ejecución del conjunto de instrucciones elementales de la Tabla 120 . El código empleado para la medición es el de la Figura 169.

```
//Se inicia la cuenta de clock del sistema.

XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x0);

XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x00000000);

XTmrCtr_LoadTimerCounterReg (XPAR_AXI_TIMER_1_BASEADDR, 0);

ControlStatus = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0);

XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR,    0,    ControlStatus    &
(~XTC_CSR_LOAD_MASK));
```

```
clocks_inicial = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR, 0);

xil_printf("\r\n\nClocks inicial: %d\r\n\n", clocks_inicial);

XTmrCtr_Enable(XPAR_AXI_TIMER_1_BASEADDR, 0);


//Instrucciones a medir el tiempo.

//Se cuenta la cantidad de clocks.

clocks_final = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR,0);

xil_printf("Clocks final: %d\r\n\n\n", clocks_final);
```

Figura 169: Código empleado en la medición de los ciclos de clk que se requieren para su ejecución

La Tabla 120 muestra los resultados obtenidos de las mediciones:

Tabla 120: Tiempos en número de clk para instrucciones elementales que se ejecutan en Xilkernel

Prueba	Instrucción	Numero de clk
Escritura de un registro de Hardware	*ptrRegistro = 5;	10
Lectura de un registro de Hardware	j = *(ptrRegistro);	13
Operación AND	i = j & NET_MASK;	5
Operación OR	i = j | NET_MASK;	5
Operación Shift	j = j << NET_OFFSET;	6
Suma con 1 registro	i = j + 1;	5
Suma con 2 registros	i = j + k;	11
Suma con 2 inmediatos	i = 2 + 5;	3
Sobrecarga de 1 iteración for	for(i = 0; i < 1; i++){};	19
Sobrecarga de 1 iteración while	while(i != 0) {};	4

Cambios de contexto

Puesto que es necesario determinar la cantidad de clk de reloj que se requieren en el cambio de contexto, con el fin de determinar qué estrategia seguir en la consulta de que transición ha sido resuelta, en este apartado se mide cuantos clk de reloj requieren un cambio de contexto.

Para esto se ha implementado una aplicación que tiene un hilo principal y dos hilos que ejecutan un bucle y la instrucción Yield (), ésta cede el procesador al otro hilo.

Las ejecuciones se realizan de la siguiente manera: se crea los dos hilos de prueba y se duerme el hilo principal, mediante Join (), hasta que los dos hilos terminen de ejecutarse. Luego se muestran los resultados de tiempo total, realizando un número variable de cambios de contexto. La Figura 170 muestra el código implementado para las operaciones del hilo principal.

```
//Se suspende la ejecución del hilo principal hasta que hayan terminado
todos los hijos.

for (i = 0; i < N_THREADS; i++) {
```

```
    ret = pthread_join(worker[i], (void*)&result);
}[150]
xil_printf("Clocks final: %d\r\n\n\n", clocks_final);
```

Figura 170: Código del hilo principal para la medición de cambio de contexto

El código de la Figura 171, es que ejecutan los dos hilos de prueba para la medición del cambio de contexto.

```
//Funcion_Principal_Hilo_1-------------------------------------------------------
-
void* hilo_run_1(void *arg)
{
    int i;
    //Se inicia la cuenta de clock del sistema.
    XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x0);
    XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x00000000);
    XTmrCtr_LoadTimerCounterReg (XPAR_AXI_TIMER_1_BASEADDR, 0);
    ControlStatus = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0);
    XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0, ControlStatus &
(~XTC_CSR_LOAD_MASK));
    clocks_inicial = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR, 0);
    xil_printf("\r\n\nClocks inicial: %d\r\n\n", clocks_inicial);
    XTmrCtr_Enable(XPAR_AXI_TIMER_1_BASEADDR, 0);
    for(i = 0; i < ITERACIONES / 2; i++) {
        yield();
    }
    pthread_exit(NULL);
    return NULL;
}
//-------------------------------------------------------------------------------
-
//Funcion_Principal_Hilo_2-------------------------------------------------------
-
void* hilo_run_2(void *arg)
{
    int i;
    for(i = 0; i < ITERACIONES / 2; i++) {
        yield();
```

```
    }
    //Se cuenta la cantidad de clocks.
    clocks_final = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR,0);
    pthread_exit(NULL);
    return NULL;
}
```

Figura 171: Código que ejecutan los hilos que hacen el cambio de contexto para la prueba

Con la ejecución del código de las Figura 170 y Figura 171, variando la cantidad de cambios de contexto, se obtuvo los valores mostrados en la Tabla 121. Los valores son promedios de retardo para cada cambio de contexto y están medidos en ciclos de reloj.

Tabla 121: Tiempos de cambio de contexto

Número de cambios de contexto	Número de clk promedio por cambio de contexto
1	936
10	862
100	776,86
1000	768,35
10000	767,98
100000	767,94
1000000	767,93

Tiempos de las funciones del HPP

Ahora medimos los tiempos de acceso para lectura y escritura del HPP. Esto lo realizamos usando la misma plataforma y herramientas que en los apartados anteriores.

Petición de disparo

La solicitud del disparo de una transición, programada como explícita, se realiza escribiendo la palabra de configuración correspondiente en el registro de la dirección base del HPP. El resultado es un evento en la cola de la transición asociada.

El código para realizar la petición de un disparo es el siguiente (subred cero, transición cero).

```
*ptrRegistro = DISPARO(0, 0);
```

Esta operación tiene una demora de 10 clk en el sistema operativo Xilkernel.

Consulta de disparo individual

Para una consulta de la ejecución de una transición en particular, primero se debe escribir la palabra de configuración en el registro de la dirección base del HPP y a continuación se lee la respuesta en el mismo registro. Luego se aplica una máscara, para obtener el campo de Valor. Si por ejemplo queremos determinar si fue la transición 0 de la subred 0 quien ha sido ejecutada, utilizamos las funciones de la librería, escrita para el manejo del HPP. El código resultante es:

```
*ptrRegistro = CONSULTA_DISPARO(0, 0);

i = *(ptrRegistro) & TRANSITION_MASK;
```

Estas operaciones tienen una duración de 24 clk en el sistema operativo Xilkernel.

Consulta de cola de salida

Para hacer una consulta sobre el estado de las primeras 32 transiciones de una subred, sólo leemos el registro de la dirección base del HPP, más un desplazamiento que es el número de subred a consultar (+1 para la subred 0, +2 para la subred 1, etc.). El código resultante para realizar una consulta de la cola de salida de la subred 0 seria:

estado_cola = *(ptrRegistro + 1)

Esta operación tiene una duración de 13 clk en el sistema operativo Xilkernel.

Interrupciones generadas por el HPP

Las interrupciones generadas por el HPP, como respuesta al disparo de una transición, es usada para la comunicación entre un proceso y el HPP.

En este apartado se mide el tiempo, que se tardan el sistema operativo Xilkernel, desde que comienza a ejecutar las rutinas de interrupción y retornar de las mismas.

Para realizar las pruebas, programamos una RdP que genera una interrupción por el disparo de la transición 0. Seguidamente solicitamos por el software de usuario la ejecución de dicha transición y ponemos a correr el IP-Core TIMER_AXI. Por último medimos la cantidad de clk que transcurren hasta cuando se comienza a ejecutar la rutina de interrupción, y hasta que se retorna de dicha rutina. En la Figura 172 mostramos la RdP utilizada para la medición de tiempos de interrupción y en la Figura 173 las matrices y vectores de programación.

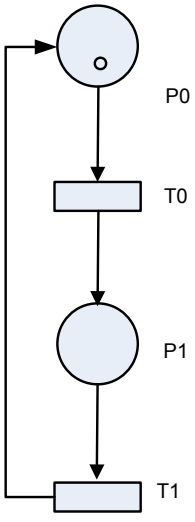


Figura 172: RdP para la medición de tiempos de INT

Matriz de incidencia

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

Marcado inicial

$$[1 \quad 0]$$

Matriz de prioridad

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Transiciones automáticas

$$[0 \quad 1]$$

Figura 173: Matrices y vectores de la RdP

Los resultados obtenidos de las ejecuciones de prueba son:

- Tiempo para reconocer y empezar a ejecutar una rutina de interrupción: 153 clk.
- Tiempo para reconocer, entrar y salir de la rutina de interrupción: 314 clk.

Mediciones de tiempos de sincronización para comparar HPP con semáforos

Para realizar las mediciones, se ha desarrollado una serie de pruebas. Estas pruebas consisten en entrar y salir de una sección crítica. El manejo en exclusión mutua de la sección crítica se programa con semáforos, que son nativos del sistema operativo Xilkernel. Posteriormente se implementa el mismo algoritmo usando HPP, con distintas políticas de espera. Por último utilizamos las interrupciones que genera el IP-Core para realizar la exclusión mutua.

Para este caso, definimos el tiempo de sincronización como el tiempo que tardan los distintos mecanismos para entrar y salir de una misma sección crítica sin hacer otra cosa. Para el caso de las pruebas realizadas con semáforos utilizamos un semáforo binario y hacemos un Wait (), para entrar en la región crítica, y para salir de la región crítica, otro hilo, ejecuta Post (). El código en lenguaje C utilizado se muestra en la Figura 174.

```
//Se inicia la cuenta de clock del sistema.

XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x0);

XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x00000000);

XTmrCtr_LoadTimerCounterReg (XPAR_AXI_TIMER_1_BASEADDR, 0);

ControlStatus = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0);

XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR,    0,    ControlStatus    &
(~XTC_CSR_LOAD_MASK));

clocks_inicial = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR, 0);

xil_printf("\r\n\nClocks inicial: %d\r\n\n", clocks_inicial);

XTmrCtr_Enable(XPAR_AXI_TIMER_1_BASEADDR, 0);


//Se realiza la exclusion mutua.

for(i = 0; i < ITERACIONES; i++) {

    sem_wait(&semaforo);


    sem_post(&semaforo);

}

//Se cuenta la cantidad de clocks.

clocks_final = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR,0);

xil_printf("Clocks final: %d\r\n\n\n", clocks_final);
```

Figura 174: Cliclos de clk para entrar y salir de una sección crítica con semáforos.

Para la medición con el HPP, se usa una modelo de RdP.

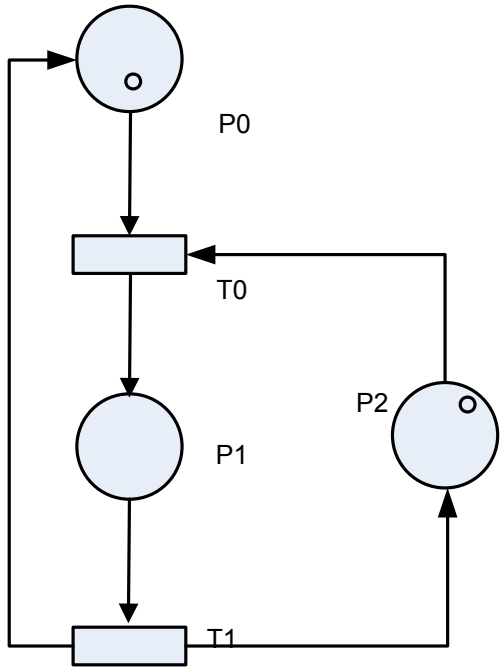


Figura 175: RdP para medir tiempo de sincronización en sección crítica

Matriz de incidencia

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Marcado inicial

$$[1 \quad 0 \quad 1]$$

Matriz de prioridad

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Transiciones automáticas

$$[0 \quad 0]$$

Figura 176: Matrices y vectores para programar el HPP en la realización de la medición de tiempos

Como podemos ver Figura 175, después de ejecutar t0, entramos en la región crítica. Si otro proceso quisiera entrar en su región crítica, no podría realizarlo hasta que el proceso actual libere el token de P2 y ejecute la transición t1 para salir de la sesión crítica.

La primera prueba se realizó con una política de espera activa sobre el HPP. Es decir que una vez solicitada la ejecución de una transición, el proceso se queda en un bucle consultando hasta que la transición es ejecutada. El código para la medición de tiempos en sección crítica con HPP es el de la Figura 177.

```
//Se inicia la cuenta de clock del sistema.

XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x0);

XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x00000000);

XTmrCtr_LoadTimerCounterReg (XPAR_AXI_TIMER_1_BASEADDR, 0);

ControlStatus = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0);

XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR,    0,    ControlStatus    &
(~XTC_CSR_LOAD_MASK));

clocks_inicial = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR, 0);

xil_printf("\r\n\nClocks inicial: %d\r\n\n", clocks_inicial);

XTmrCtr_Enable(XPAR_AXI_TIMER_1_BASEADDR, 0);


for(j = 0; j < ITERACIONES; j++) {
```

```
    //Se realiza la exclusion mutua.
    *ptrRegistro = DISPARO(0, 0); //Se solicita la exclusion mutua.

    do {
        *ptrRegistro = CONSULTA_DISPARO(0, 0);
        i = *(ptrRegistro) & TRANSITION_MASK;
    }
    while(i != 1);

    //Se entro en exclusion mutua.
    *ptrRegistro = DISPARO(0, 1); //Se libera la exclusion mutua.

    do {
        *ptrRegistro = CONSULTA_DISPARO(0, 1);
        i = *(ptrRegistro) & TRANSITION_MASK;
    }
    while(i != 1);

    //Se salio de exclusion mutua.

}

//Se cuenta la cantidad de clocks.
clocks_final = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR,0);
xil_printf("Clocks final: %d\r\n\n\n", clocks_final);
```

Figura 177: Código para la medición de tiempos en sección crítica con HPP

La siguiente prueba consiste en medir los tiempos, donde la espera se realiza haciendo uso de una interrupción.

En el sistema operativo Xilkernel, el cual utiliza el estándar POSIX, no tiene una forma nativa de dormir hilo. El esquema para el control de la sección crítica se basa en una variable global, cuyo valor es controlado en la rutina de ejecución, que es llamada cuando el HPP ejecuta una transición. La Figura 178 muestra la rutina para la medición de los tiempos de ejecución usando INT y en el HPP.

```
//Se inicia la cuenta de clock del sistema.
XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x0);
XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_1_BASEADDR, 0, 0x00000000);
XTmrCtr_LoadTimerCounterReg (XPAR_AXI_TIMER_1_BASEADDR, 0);
ControlStatus = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR, 0);
XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_1_BASEADDR,    0,    ControlStatus    &
(~XTC_CSR_LOAD_MASK));
clocks_inicial = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR, 0);
xil_printf("\r\n\nClocks inicial: %d\r\n\n", clocks_inicial);
XTmrCtr_Enable(XPAR_AXI_TIMER_1_BASEADDR, 0);

for(j = 0; j < ITERACIONES; j++) {

    //Se realiza la exclusion mutua.

    *ptrRegistro = DISPARO(0, 0); //Se solicita la exclusion mutua.

    while(exclusion != 1) {};

    //Se entro en exclusion mutua.

    *ptrRegistro = DISPARO(0, 1); //Se libera la exclusion mutua.

    while(exclusion != 0) {};

    //Se salio de exclusion mutua.

}

//Se cuenta la cantidad de clocks.
clocks_final = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_1_BASEADDR,0);
xil_printf("Clocks final: %d\r\n\n\n", clocks_final);
 //Funcion que se ejecuta cuando se produce la interrupcion de la Red 0---------
-
void interrupcion_net_0()
```

```
{
    int i;

    *ptrRegistro = CONSULTA_DISPARO(0, 0);
    i = *(ptrRegistro) & TRANSITION_MASK;
    if(i) {
        exclusion = 1;
        return;
    }
    *ptrRegistro = CONSULTA_DISPARO(0, 1);
    i = *(ptrRegistro) & TRANSITION_MASK;
    if(i) {
        exclusion = 0;
        return;
    }
    return;
}
//-------------------------------------------------------------------------
-
```

Figura 178: Código para la medición de tiempos de interrupción usando el HPP

Para los tres casos de prueba elaborados, corrimos las aplicaciones midiendo los tiempos y variando la cantidad de veces que se entra y sale de la sección crítica a través de un bucle controlado por una constante que denominamos ITERACCIONES.

La Tabla 122, se muestran los resultados obtenidos en promedios de clk del sistema por iteración.

Tabla 122: Tiempos para entrar y salir de una sección crítica usando semáforos, HPP e INT

Tiempos en sección crítica			
	Semáforos	HPN espera activa	HPN Interrupciones
1 iteración	320	97	781
10 iteraciones	314,9	90,9	774,9
100 iteraciones	313,19	90,09	774,9
I10000 iteraciones	313,5	90,07	774,26
100000 iteraciones	313,39	90,1	774,57
1000000 iteraciones	313,38	90,11	774,59

En la Figura 179, se muestra la gráfica de tiempos, donde se ha hecho uso de las distintas alternativas planteadas. El HPP, presenta una mejora del 71% promedio de frente a el uso de semáforos.

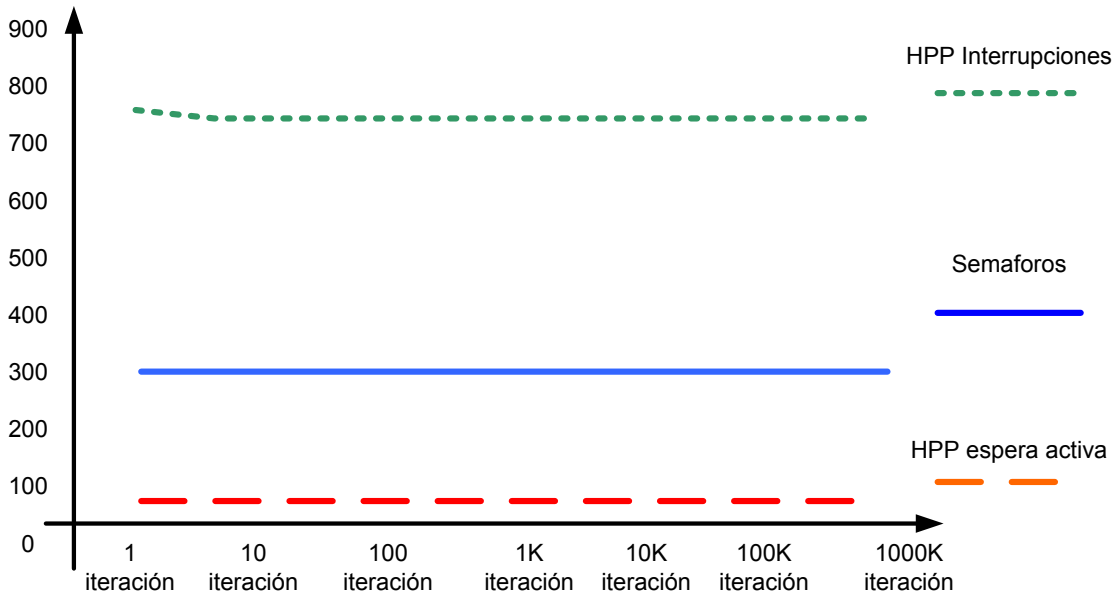


Figura 179: Tiempos de acceso a sección crítica, con semáforos, HPP e INT

Hay que destacar que si bien los tiempos de interrupción son altos, son muy útiles para cuando se tienen que hacer esperas del orden de dos o más veces en los tiempos de cambio de contexto. Puesto que son grandes y el proceso consume toda la cuota de tiempo y de todos modos hace el cambio de contexto.

Ejemplo de sincronización

Ahora implementamos una aplicación con 4 hilos que acceden a una misma variable escribir. Esto implica que el acceso se realiza en exclusión.

Este problema resuelve de tres maneras distintas:

1. Con semáforos del sistema operativo.
2. Usando el HPP con una política de espera activa.
3. Haciendo uso de interrupciones para generar la exclusión mutua con el HPP.

El modelo a implementar es el de la Figura 180. Se ha dividido en cinco subredes. Esta división favorece la comunicación, puesto que cada proceso se vincula con una subred. El sistema tiene 12 transiciones del sistema, 8 se cortan generando 8 transiciones distribuidas. El factor α de acoplamiento es de 8/12 (α = 0,66). Con esto, y teniendo en cuenta que se dividió la red original en 5 subredes, la disminución de recursos para la implementación del sistema es de 0,4978.

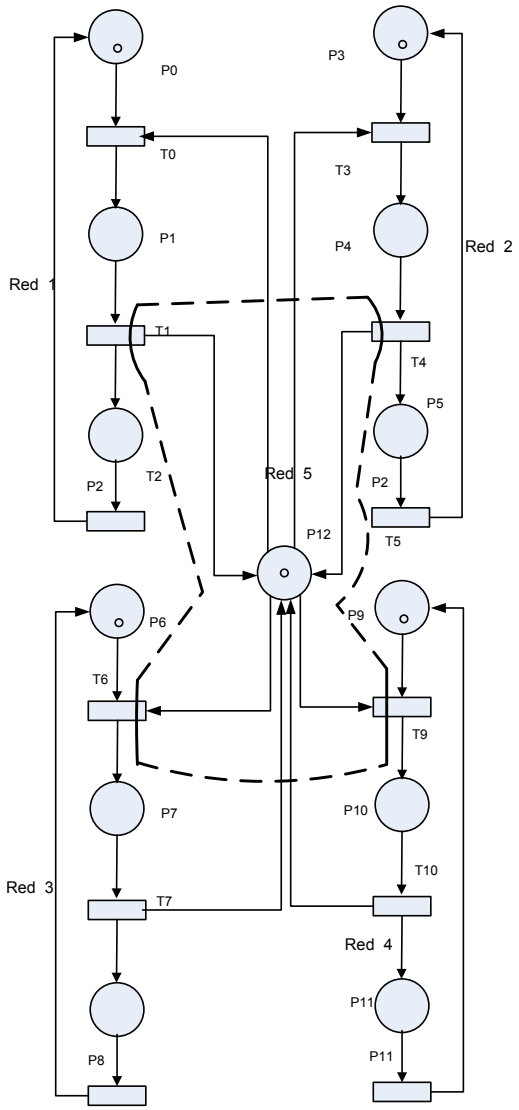


Figura 180: Modelo de 4 escritores, dividido en 5 subredes

Las subredes 1, 2, 3 y 4 se corresponden con los 4 hilos de la aplicación. La subred número 5 contiene la plaza que sirve como mutex (semáforo binario) para realizar la exclusión mutua al acceso de la variable compartida. De esta forma podemos configurar cada interrupción individual de una subred para que atienda las acciones correspondientes a cada hilo. Pero como mencionamos anteriormente, esta implementación no la realizamos de esta forma porque no existe una forma explícita de dormir un hilo en el sistema operativo Xilkernel.

Tabla 123: Tiempos para cuatro escritores implementados con semáforos, HPP e INT

Cuatro escritores sobre una variable (cantidad de clk)			
	Semáforos	HPP espera activa	HPP Interrupciones
1 iteración	4693	4678	5430
10 iteraciones	7515	5574	13022
100 iteraciones	35685	14484	88892
1000 iteraciones	317385	103584	847592
10000 iteraciones	3137527	994584	8509093

Tabla 124: Tiempos promedios de cuatro escritores sobre una variable

Cuatro escritores sobre una variable (promedio de clk por acceso)			
	Semáforos	HPP espera activa	HPP Interrupciones
1 iteración	4693	4678	5430
10 iteraciones	752	557	1302
100 iteraciones	357	145	889
1000 iteraciones	317	104	848
10000 iteraciones	314	99	851

Las implementaciones de prueba semáforos, HPP con espera activa e INT se corrieron, variando el número de accesos a la variable compartida entre 1 y 10000 escrituras. Se muestran en la Tabla 123. En todos los casos medimos la cantidad de clk, y se promediaron por iteración. A continuación presentamos los resultados obtenidos.

La gráfica de los promedios obtenidos, cantidad de clk por iteración, para 1000 iteraciones se muestra en la Figura 181.

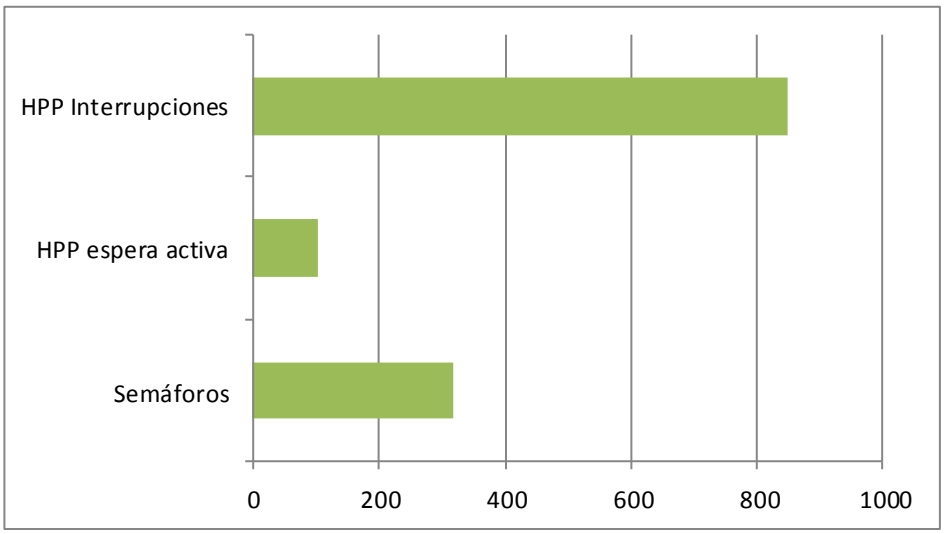


Figura 181:Tiempos de sincronización para cuatro escritores con semáforos, HPP e interrupciones

Como podemos ver las mejoras relativas obtenidas con el uso del HPP con espera activa, son similares a las obtenidas en las pruebas de tiempo de sincronización. Esto se debe a que esta aplicación tiene una carga muy baja de trabajo y un alto grado de sincronización. En otro tipo de aplicaciones donde la sincronización no es tan importante frente a los tiempos totales de ejecución, las ganancias de tiempo por el uso del HPP no son tan significativas.

Caso de canal marítimo

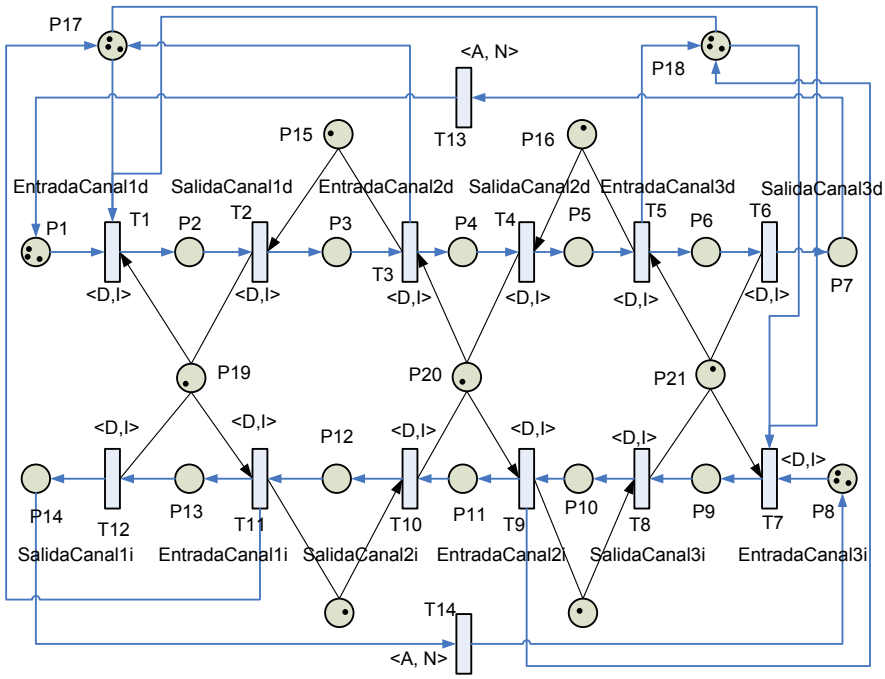


Figura 182: RdP del canal marítimo con las plazas de control

En esta sección se presenta el caso de tráfico marino, para controlar un sistema de tráfico de canal [242]. Se ha elegido este caso por el gran acoplamiento y la dificultad para realizar la división en subredes. Se realizó un estudio de los sifones críticos presentes en la RdP, de este ejemplo, y se determinó la existencia de interbloqueo lo cual es solucionado con las dos plazas de control P15 y P16, que evitan que se vacíen los sifones.

La RdP que modela el canal, con las plazas de control, se muestra en la Figura 182.

Pruebas de rendimiento

Para evaluar el beneficio de las RdP jerárquicas frente al uso de semaforos, se ha dividido el sistema en 4 subredes, y se requiere de 8 transiciones distribuidas que relacionan dichas subredes. A continuación, en la Figura 183, se muestra la división realizada. Cada subred está delimitada por una línea de color, también se han incluido las etiquetas en cada transición para la programación.

Para la sincronización del HPP, al igual que las pruebas realizadas con semáforos, se realizaron distintos casos de prueba. Se simuló cada estado o plaza como una actividad representada por el incremento de una variable en 4, 8, 16, 32 y 128 veces. Asimismo, se varió el número de embarcaciones que circulan a través de los canales de 1 a 127.

Cada una de las pruebas, para los distintos valores mencionados, se realizaron sobre 3 implementaciones distintas: semáforos, HPP con espera activa y cediendo el procesador. Se corrieron 10 veces tomando como resultado final un valor promedio.

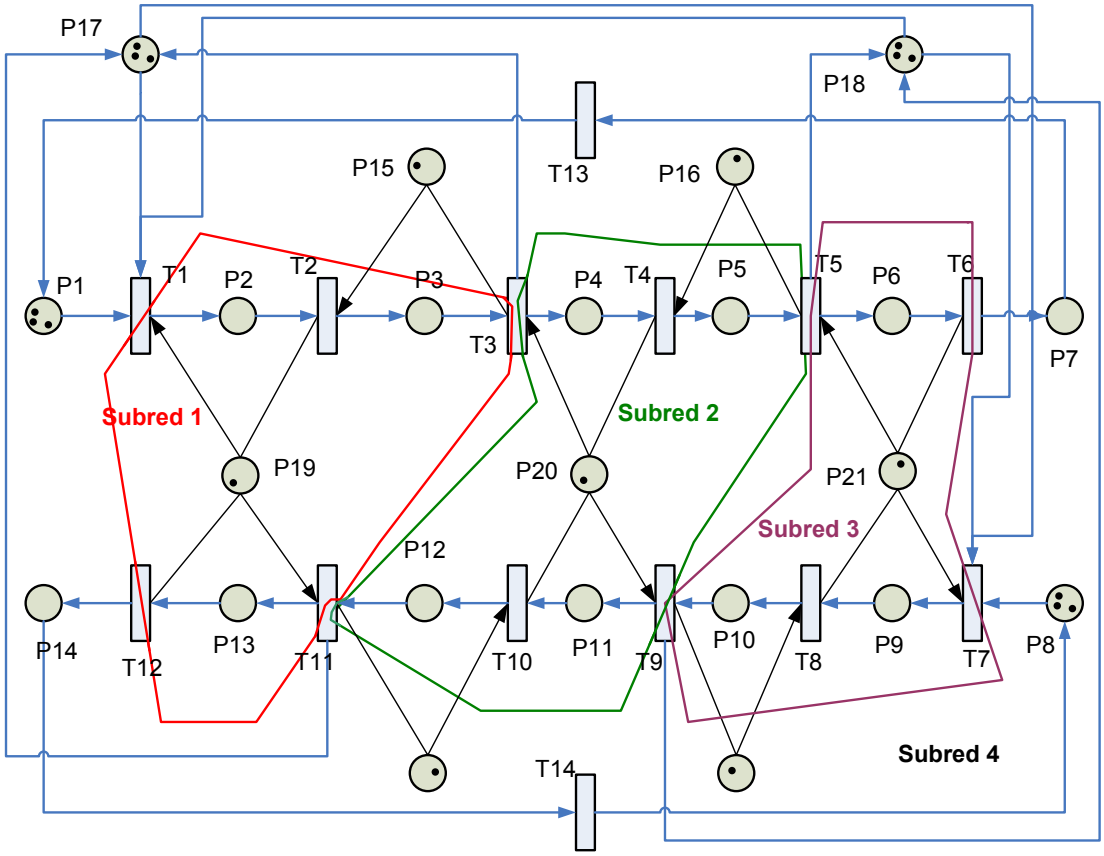


Figura 183: RdP dividida del canal marítimo (subredes)

Para llevar el conteo de clk, de cada implementación, se utilizó el IP-Core "AXI Timer". Definimos el rendimiento, por el uso del HPP frente a los semáforos, como el cociente entre ambas medidas:

$$\eta = \frac{T_{sem}}{T_{HPN}}$$

Dónde:

- T_{sem} es el tiempo de ejecución para las implementaciones basadas en semáforos.
- T_{HPN} es el tiempo de ejecución para las implementaciones basadas en el HPP con espera activa.
- T_{HPNY} es el tiempo de ejecución para las implementaciones basadas en el HPP haciendo uso de yield()

Tabla 125: Tiempos de ejecución para cargas de 32 incrementos

Carga de trabajo de 32 incrementos por actividad			
Embarcaciones	**Semáforos**	**HPP espera activa**	**HPP Yield()**
1	15831	10433	14556
20	233120	196569	134536
40	462342	343231	335665

60	695345	434234	498342
80	911233	621123	667534
100	1178365	800973	816567
120	1372344	959545	943278
127	1485664	1015354	1134671
Nota: valores totales medidos en cantidad clk del sistema a 25MHz.			

Tomando como caso de referencia el de 32 incrementos por cada actividad se obtiene el rendimiento por el uso del HPP en comparación con el uso de semáforos. Los resultados obtenidos son los de la Tabla 125.

Como se puede observar en la el uso del HPP (exceptuando el caso para 1 embarcación), aporta una mejora en los tiempos totales de ejecución del 43% para el caso de una espera activa y del 42% para el caso de la implementación cediendo el procesador. Esto teniendo en cuenta una carga de trabajo igual a 32 incrementos de una variable para cada actividad de los procesos.

En la Tabla 126 el uso del HPP (exceptuando el caso para 1 embarcación) aporta una mejora en los tiempos totales de ejecución del 43% para el caso de una espera activa y del 42% para el caso de la implementación cediendo el procesador. Esto teniendo en cuenta una carga de trabajo igual a 32 incrementos de una variable para cada actividad de los procesos.

Tabla 126: Rendimiento para carga de 32 incrementos

Carga de trabajo de 32 incrementos por actividad		
Embarcaciones	Semáforos/HPN Activa	Semáforos/HPN Yield
1	1,23	1,23
20	1,39	1,39
40	1,43	1,42
60	1,44	1,41
80	1,43	1,43
100	1,44	1,42
120	1,44	1,42
127	1,44	1,42

Tomando los casos de prueba para cargas de trabajo de 4, 8, 16, 32, 64 y 128 incrementos de una variable, y con 100 embarcaciones, obtenemos la Figura 184, que nos muestra el incremento de rendimiento del uso del HPP con espera activa, frente al uso de semáforos.

En la Figura 184 se observa que para valores de carga de trabajo muy pequeños no hay ganancia; puesto que, prácticamente la totalidad del tiempo de ejecución no corresponde a la sincronización. Un valor de mejora esperado es el que ha sido medido en los trabajos de simulación de [243, 244]. Están en el orden del 44% al 23%. En nuestro caso, estos valores se corresponden con cargas de trabajo de 32 y 64 incrementos.

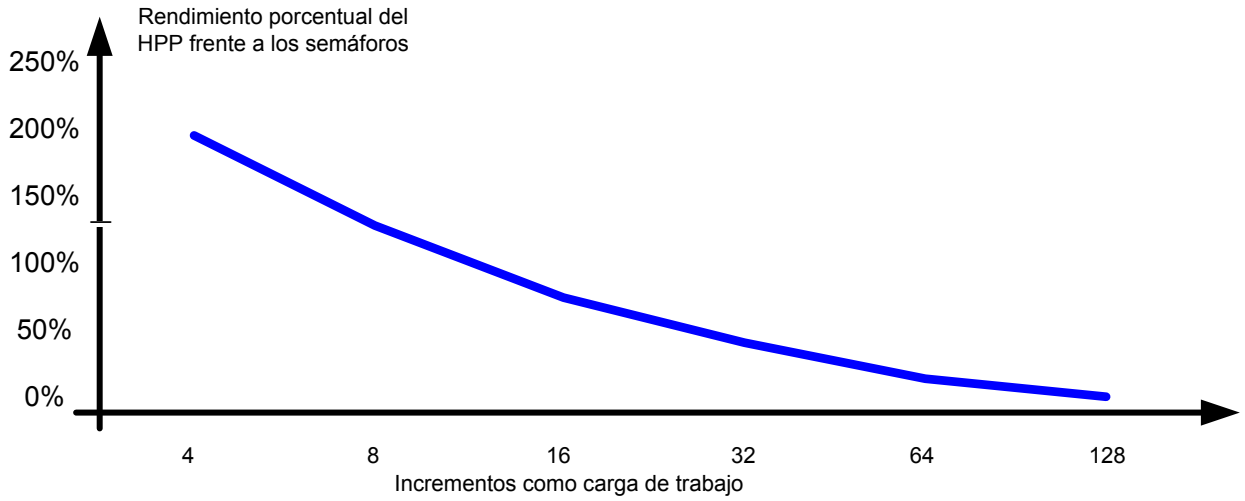


Figura 184: Mejora con el uso del HPP frente al uso de semáforos, según varía la carga de trabajo

Caso de división de RdP del modelo de una celda flexible

Para ilustrar el método de división propuesto, en esta sección se expone el caso del modelo de RdP propuesta en [2], con el objetivo de ser ejecutada en un HPP. Esta red modela la lógica de una celda de fabricación flexible y el caso fue seleccionado por el alto grado de complejidad y acoplamiento que posee.

La celda de fabricación está compuesta por 4 máquinas y 3 robots que realizan 3 procesos de fabricación distintos. En la Figura 185, la RdP que ha sido obtenida de la bibliografía [245], muestra la división que hemos realizado para hacer las pruebas (en tres subredes).

La división de la red se realizó con los siguientes criterios: minimizar el número de divisiones de las transiciones, y además repartir el número de plazas y transiciones en forma equitativa entre las subredes. Para este caso se dividieron 6 transiciones generando 6 transiciones distribuidas. Por otro lado la red original tiene 20 transiciones, por lo que el factor de acoplamiento resulta:

$$\alpha = 6/20 = 0.3$$

Utilizando la ecuación obtenida para calcular la ganancia de recursos por dividir la red, vemos que ésta está dada por:

$$G = 1 - 1/3 - 0.3/9 - 2(0.32)/9 - 0.3/3 = 0.51$$

De esta forma se obtiene una reducción del 51% de los recursos de hardware necesarios para implementar el modelo.

Las 3 subredes obtenidas están relacionadas entre sí a través de las 6 transiciones distribuidas. En la Figura 186 se muestran las 3 subredes resultantes junto con las transiciones distribuidas del sistema. Las flechas grises muestran las relaciones entre las transiciones de borde de cada subred.

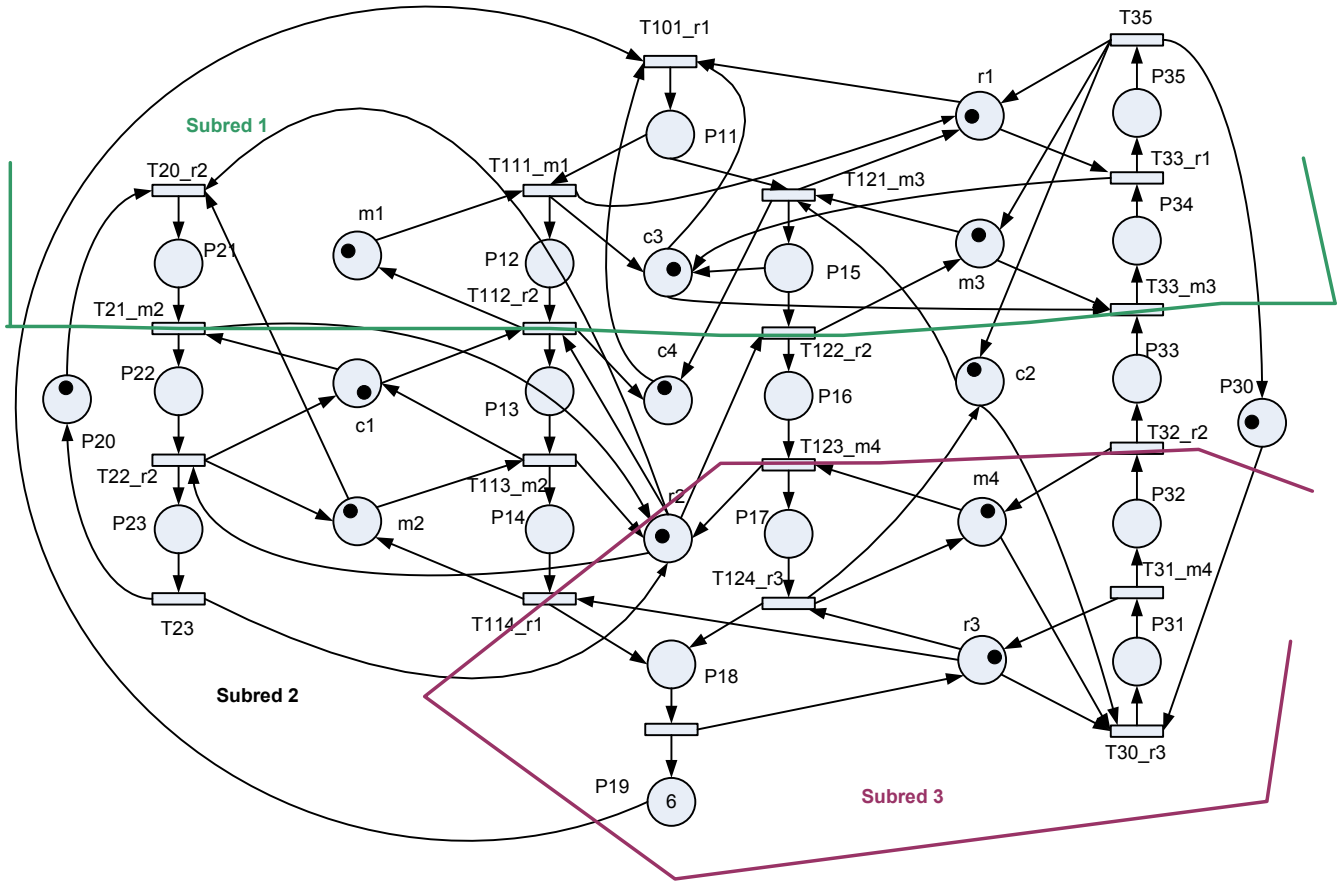


Figura 185: División para la celda de fabricación en 3 subredes

En la Figura 187 se muestran las matrices de relación con las transiciones distribuidas resultantes.

Resultados y Conclusiones

En este capítulo se propone una nueva manera de expresar las RdP. Esta manera permite disminuir la cantidad de información necesaria para expresar el modelo. Como consecuencia de esto se pueden implementar con menos recursos de hardware sistemas con más plazas, transiciones, tipos de transiciones, arcos y tipos de arcos.

Según los resultados obtenidos, en las pruebas realizadas usando HPP para sistemas concurrentes y paralelos, este procesador de eventos por hardware aporta mejoras en el desempeño. Esto se traduce en menores tiempos de ejecución de los programas y reducciones de recursos de hardware, en comparación con el PP, que van entre el 40% y el 60%.

La mejora en el desempeño, con respecto al tiempo, se hace más significativa a medida que las aplicaciones poseen un mayor grado de sincronización. Para una aplicación con poca sincronización, este aumento de rendimiento no es significativo, pero aun así el uso del HPP nos facilita la implementación del sistema directamente a partir de la RdP dividida, que es el modelo creado para validar el sistema. Esto disminuye los tiempos de codificación. Además mantiene la versatilidad de las RdP para modelar sistemas concurrentes y paralelos, lo que hace que el sistema resultante sea flexible ante cambios en la implementación.

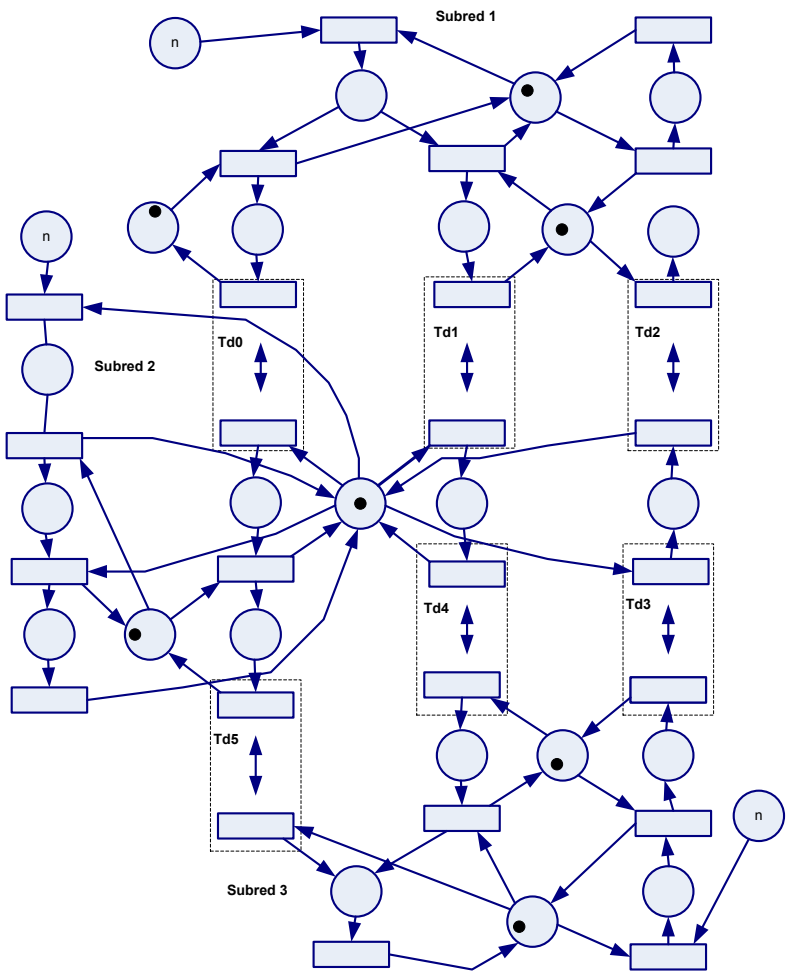


Figura 186: Se muestran las tres subredes junto con las transiciones distribuidas del sistema.

Subred 1

$$\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0 \end{pmatrix}$$

Subred 3

$$\begin{pmatrix} 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&1&0&0&0&0&0\\ 1&0&0&0&0&0&0 \end{pmatrix}$$

Subred 2

$$\begin{pmatrix} 0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0&0&0&0 \end{pmatrix}$$

Figura 187: Matrices de relación

Esta propuesta de división tiene ventajas adicionales que son:

- Permite múltiples disparos de la RdP simultáneamente. Cuando no es global al menos una por subred
- Si la división de la red se realiza generando localidad estructural, es decir agrupando en cada subred: los brazos inhibidores, la parte de la red que tenga arcos con peso más de uno, las transiciones con tiempo, las transiciones temporizadas, etc. es posible instanciar una subred con elementos en común. Esta agrupación, redunda en una

disminución de recursos, puesto que cada subred implementa sólo el tipo que requiere.

Capítulo 7

Procesador de Petri (PP) Comunicación y Código de Procesos

Resumen

Actualmente la ejecución paralela es necesaria en todos los sistemas, ya que en general éstos están formados por múltiples procesadores [11, 15]. Esto nos motiva al estudio, investigación y desarrollo de soluciones de sistemas concurrentes y paralelos. Hasta ahora se han usado las RdP como herramienta gráfica con una base matemática y formal, para modelar diferentes estados de los sistemas reactivos y las posibles transiciones entre ellos [1]. Dicho modelo gráfico se corresponde a una ecuación de estado, por medio del cual se definirá el siguiente estado del sistema según el estado actual y los eventos que se generan en el entorno. En los capítulos precedentes hemos ejecutado estas redes haciendo uso del PP o del HPP. El procesador es programado con las matrices y vectores del modelo, con lo cual, el modelo y la implementación han resultan equivalentes.

En este capítulo se presenta el desarrollo de un software para dar soporte a un PP que interactúa con un sistema real. Este software facilita solución de la concurrencia y del paralelismo de los sistemas modelados con RdP. Esta propuesta es una aproximación para el desarrollo de sistemas concurrentes y reactivos que han sido modelados con RdP.

Objetivos

Hasta ahora hemos diseñado e implementado una familia de PP y un simulador. Con el fin de ejecutar el modelo realizado con la RdP de un sistema concurrente.

Puesto que en los modelos realizados con RdP, no se programan efectivamente las acciones que realizan los procesos (ejecución de las acciones), dado que a estas acciones las realizan los procesos que se ejecutan en los procesadores que componen al SMP.

Y con este fin de ejemplificar estas cuestiones, en este apartado se realiza una aproximación de cómo se realiza la codificación de las acciones y su relación con el PP.

El objetivo principal es obtener una secuencia de pasos para construir los procesos (el código) requerido por las acciones que son dirigidas por un PP, de un sistema modelado por una RdP orientada a procesos (POPN) [2].

Antecedentes

El PP es un componente encargado específicamente de la ejecución de la RdP, que es el modelo del sistema. Este procesador puede estar implementado tanto en hardware como en software.

En esta tesis, se han desarrollaron múltiples procesadores de RdP, los que han sido implementado en IP-Core, para la ejecución del HPP [92] mediante hardware, y un software de simulación de un procesador de RdP jerárquicas [246], el cual se utilizó en este trabajo como componente para la ejecución de la RdP.

Ahora estamos interesados en determinar la forma de generar o construir el código que requiere el PP, para ejecutar distintas acciones y/o lecturas y/o escrituras en un sistema (sensores y/o actuadores).

En [34, 50, 106] se desarrolla una herramienta para la generación de código automático para sistemas concurrentes usando RdP, no contemplando la ejecución paralela. Esencialmente, se realiza la codificación del programa incluyendo la RdP y las acciones. Mientras que en este trabajo buscamos mantener la programación directa (solo con el modelo) del PP y las acciones programarlas como secuenciales y dirigidas por el PP.

Estas estrategias permiten usar las facilidades del PP, para programar el modelo, que son: distintos tipos de brazos, transiciones inmediatas y transiciones temporales; la decisión del disparo se realiza en dos ciclos de reloj y en paralelo, la programación de la prioridad es directa y la ejecución es paralela. Toda la programación puede ser realizada en tiempo de ejecución. Con respecto a las etiquetas de las transiciones, nuestro sistema incluye el disparo regular, automático o no-perenne en la entrada, y el informe y no informe en la salida. La programación de la etiqueta no requiere código, sólo un vector.

Etiquetas de Transición

El PP, para comunicarse con los procesos, requiere de un mecanismo que especifique el comportamiento de cada transición en el sistema. Por tal motivo, en el capítulo anterior se creó una nomenclatura de etiquetas. La Figura 188 ilustra las convenciones utilizadas para programar las transiciones del PP.

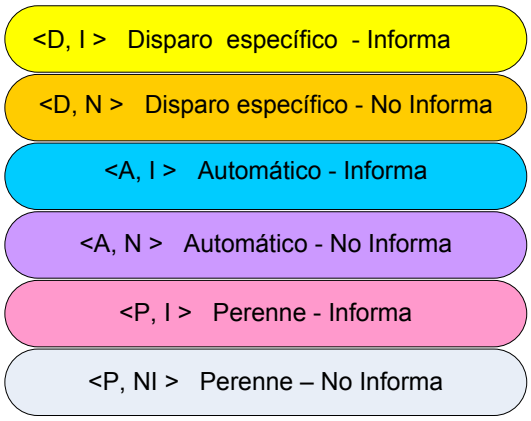


Figura 188: Etiquetas de las transiciones del PP

Con el fin de ejemplificar el uso de estas etiquetas desde un proceso, hacemos las siguientes consideraciones.

Procesos que se relacionan con las colas de entradas

Las etiquetas de transiciones, para definir la el comportamiento de la cola de entrada en relación con los procesos son:

- D, disparo específico (D).
    - Esta etiqueta es requerida por el proceso cuando es necesario una sincronización, es decir que: las plazas que tienen brazos que entran a la transición tienen los token suficientes (transición sensibilizada) y existe un evento que hace el requerimiento del disparo, la transición es no autónoma.

    - Esto implica que un proceso o acción debe comenzar cuando se resuelva el disparo.
    - Por ejemplo: entrar a una sección crítica, esperar que un actuador este en una posición, esperar que un robot o máquina este libre, etc.
- Disparo, automático (A).
    - Esta etiqueta es requerida por el proceso cuando es necesario que se envíe un mensaje que el PP tiene un determinado estado local. Es decir: las plazas que tiene brazos que entran a la transición tienen los token suficiente y no se requiere de un evento, la transición es autónoma.
    - Esto puede usarse cuando un proceso o acción debe comenzar por un estado local del PP.
    - Por ejemplo: si un robot y las máquinas están desocupadas, y existe material para comenzar el proceso.
- Disparo, no-perenne (P).
    - Esta etiqueta es requerida por el proceso cuando es necesario realizar una acción que no se ha hecho y si está hecha no se hace nada.
    - Esto implica que si el PP tiene el estado adecuado cuando llega el evento, la transición se dispara por el evento; pero si el estado no es el adecuado el evento se pierde.
- Por ejemplo: si se desea prender una luz que esta prendida y se intenta prenderla (presionando la llave) la luz queda prendida y el evento se pierde (intento presionando la llave).

Procesos que se relacionan con las colas de salida

Las etiquetas de transiciones, para definir la el comportamiento de la cola de salida en relación con los procesos son:

- Disparo informado, disparo que informa a través de la cola (I).
    - Esta etiqueta es requerida por los procesos que necesitan se les informe que ha sucedido una sincronización o un estado ha sido alcanzado.
    - Es decir que una transición se ha disparado.
    - Esto implica que un proceso o acción está esperando, por este evento, para comenzar.
    - Por ejemplo: entrar a una sección crítica, esperar que un actuador esté en una posición, o esperar que un robot o máquina esté libre, etc.
- Disparo No informa.
    - Esta etiqueta es requerida por que ningún proceso requiere que se le informe del cambio de estado del PP.
    - Esto implica que un proceso o acción que ha terminado, cambia el estado de la red, pero el disparo de la transición no es escuchada por un proceso.
    - Por ejemplo: si un proceso abandona una sección crítica, no es necesario que lo informe por la transición de salida, ya que se sensibiliza la transición de entrada.

La cola de entrada tiene la función de almacenar las transiciones cuyos disparos se solicitaron explícitamente para ser ejecutados, y proveer a la red la información que necesita para determinar que transiciones se encuentran pendientes en la cola. La cola de salida tiene la función de

almacenar las transiciones ejecutadas por la red para que los procesos puedan en cualquier momento consultar su estado.

Es importante notar que las etiquetas de transición son binarias. Es decir, que se requieren tres bits para generar las programaciones posibles de cada etiqueta, descriptos en la Figura 188, de esta forma se define el comportamiento de cada transición con baja complejidad. Esto resulta en que las transiciones son simples de programar y en la implementación del procesador se requieren sólo funciones booleanas.

Mientras que los actuadores y/o sensores son envueltos por un software para conformar un componente que se comunica con el PP por los eventos.

Procesos

Para el presente trabajo, abordaremos solo los sistemas modelados con RdP orientadas a procesos POPN [2].

Para este modelo cada componente (plaza, transición o token) tiene semántica, esta es:

- Hay plazas que representan un recurso o una operación.
- Las marcas iniciales representan el número de máquinas en una planta, el número de piezas que puede procesar una máquina o el número de robots, etc.
- Las plazas que no representen a un recurso puede ser el número de trabajos que se deben realizar.
- Las plazas que representan una operación, en este estado se realiza una operación.
- Las transiciones que representan el inicio o bien el fin de una operación, proceso o evento.

En el modelado del sistema, se especian los procesos y se relaciona los componentes físicos con los componentes de la RdP, esto es lo que dá la semántica al POPN.

Es deseable determinar la cantidad de procesos (o hilos) y si existen otros objetos en el sistema.

En cuanto a los tipos de objetos, diferenciamos entre \procesos runnable" (adoptando el termino \runnable" del lenguaje Java), que son los hilos de ejecución del sistema, y el resto de objetos son los que representan a los recursos del sistema. Además, es necesario especificar en los obetos las transiciones que están relacionadas con cada uno de los procesos.

Por restricción de diseño en una POPN, una transición no puede formar parte de más de un proceso, es decir un solo proceso puede pedir el disparo o esperar el informe de una misma transición (esta restricción no es indispensable en el uso del PP).

Para el desarrollo del componente y la generación de código surge la necesidad de definir la cantidad de procesos que forman parte del sistema.

Por lo tanto, se deben especificar la cantidad de procesos del sistema, sus responsabilidades y tipos. Se han diferenciado: “procesos runnable”, que definen la dinámica y orden de ejecución del sistema, y los “objetos” que representan a los recursos del sistema. En el código resultante del sistema, los procesos “runnable” se convierten en hilos ejecutables y los de tipo “no runnable” en objetos.

Modelado de RdP Orientada a Procesos (POPN)

Introducción

Para el funcionamiento de los sistemas de fabricación automatizado (AMS), es importante explotar la flexibilidad del sistema. La expectativa de flexibilidad del sistema, está dada por la potencialidad de la producción de un conjunto de piezas por la planta. Donde hipotéticamente las máquinas, robots, entradas y salidas de piezas no producen interferencia.

Para esto en las últimas décadas, el modelado, análisis, simulación y control de la AMS se han convertido en temas emergentes y ampliamente estudiados. Por esto es necesario el modelo, para realizar el análisis, simulación y control de la AMS.

Con un POPN es posible modelar un AMS directamente. Para lo cual se utilizan plazas que representan operaciones. Estas plazas describen la parte secuencial de las operaciones de cada pieza que procesa el sistema. También los recursos son representados por plazas, con las que se describen las necesidades de las operaciones. Los modelos POPN pueden describir los procesos de fabricación clara y fácilmente, y son ampliamente usados.

La complejidad de la AMS, está dada principalmente por el intercambio de recursos. Sobre la base de los recursos adquisición para cada operación, los AMS se clasifican en tres clases:

1. La asignación de recursos para la operación del sistema requiere una sola unidad de un solo recurso [247],
2. Los sistemas de montaje / desmontaje [248] [249] y
3. La asignación de recursos conjuntivo y disyuntivo para cada operación del sistema puede requerir un número arbitrario de unidades de un arbitraria conjunto de recursos [250].

Aquí usamos la clase 1 de modelado de POPN con la diferencia que el modelo resultante es la parte paralela del programa ejecutado, para luego discutir las propiedades y resultados.

Método de Modelado

En el modelado de sistemas de eventos discretos (DES) con RdP, a menudo los lugares y los token se utilizan para las condiciones, y las transiciones se utilizan para la ocurrencia de eventos.

Ahora veremos cuáles son las características de los modelos POPN y las consideraciones a tener en cuenta para el diseño con POPN.

Características de los modelos con POPN
Es sabido que el modelado con POPN permite obtener modelos de procesos de producción en forma directa con detalles y de simple comprensión.

Sin embargo, cuando el tipo de piezas u operaciones es grande, el modelo resultante es grande y en general, aumenta la complejidad para el análisis y control.

Por lo tanto, un tema importante es cómo realizar el análisis de los modelos resultantes.

También es conocido que la dificultad en la operación de un AMS proviene de lo limitado de los recursos. Donde los recursos del sistema son compartidos por un número plazas/transición y se utilizan en una manera mutuamente excluyente.

Teniendo en cuenta, los sifones e invariantes de plaza, es fácil verificar que cuando un sifón marcado en la condición inicial se vacía el sistema se interbloquea.

En [2] se muestra que POPN es una forma sencilla de modelar una AMS. En un modelo POPN el intercambio de recursos puede ser descrito por la PME ("parallel mutual exclusion") y de las SME (sequential mutual exclusion). La ausencia y la existencia de sifones con mal comportamiento IBS (ill behaved siphon in a Petri net), en un modelo POPN, determina el interbloqueo del modelo. La relación entre el ISB y RC (circuito de recursos) es usado para determinar el comportamiento de la red.

Con los sifones que se denominan sifón con mal comportamiento (SII), es posible examinar el modelo POPN, cuando el sistema alcanza un marcado M tal que se bloquea y el sifón se vacía, queda determinada la causa.

Además, del método de modelado POPN, también se usa el modelado con RdP orientado a recursos ROPN (resource-oriented Petri net).

La estructura que resultan de modelar con ROPN son más compactas (menos plazas y transiciones) que las que resultan de modelar con POPN. Esto se debe a que el modelo con ROPN se realiza con RdP coloreadas. En el modelado, de los procesos de producción, un circuito de proceso de producción en ROPN corresponde a un sifón de mal comportamiento en POPN. Sin embargo, en el modelado de los procesos de manejo de materiales, los sifones de mal comportamiento en POPN se pueden evitar mediante el modelado ROPN. En general, ROPN puede disminuir significativamente la complejidad, esto es la cantidad de plazas transiciones y brazos, lo que facilita el análisis y control del sistema.

Consideraciones para realizar el modelado con POPN
Para el modelar un sistema AMS con POPN, y teniendo en cuenta la interpretación de plazas, transiciones y token introducida previamente, es necesario realizar las siguientes consideraciones:

1. Identificar las actividades y los recursos necesarios para la producción de cada producto.
2. Identificar la relación de precedencia de las actividades.
3. Para cada actividad:
    a. Crear y etiquetar un lugar que representa el estado de esa actividad.
    b. Añadir una transición (actividad de inicio) con un arco (s) de salida a la plaza (s). El disparo de la transición inicial representa inicio de la actividad o proceso.
    c. Añadir una transición (actividad fin) con un arco de entrada (s) desde el lugar (s) actividad. El disparo de la transición de parada representa la finalización de la actividad y también puede representar el inicio de la siguiente actividad.
    d. En general, la transición de parada para una actividad es la misma que la transición de inicio para la siguiente actividad.
    e. Múltiples token en la plaza indican que la actividad tiene multiplicidad. Por ejemplo, dos tokens podrían representar dos partes siendo almacenados en el mismo depósito.
4. Para cada elemento del producto, crear y etiquetar un lugar. Este lugar representa el estado de la orden de trabajo. Unir la plaza con un arco (s) de salida a la transición de inicio (s) de la primera actividad del elemento y un arco de entrada (s) a partir de la transición de parada (s) de la última actividad del producto. Los token en este lugar indican el número de órdenes de trabajo para el producto a realizar por el sistema.

5. Para cada actividad, en el orden de actividades:
    a. Crear un lugar si no se ha creado.
    b. Crear y etiquetar un lugar para cada recurso que debe estar disponible cuando inicia la actividad.
    c. Conectar todos los lugares, que representan disponibilidad de recursos, a la transición de comienzo de la actividad.
    d. Crear arcos de salida para conectar la transición de parada, después de la actividad a cualquier lugar de recursos que representan los recursos que se dispongan, éstos son liberado tras la finalización de la actividad.
6. Especifique el marcado inicial del sistema.

De lo expuesto decimos, que el modelado por el método de POPN de AMS, contiene tres tipos de lugares, que son: (1) lugares para las operaciones, (2) lugares de los recursos, y (3) lugares que representan el estado de las piezas pedidas para el trabajo.

Flujo de Trabajo para el diseño del software
La Figura 189 expone el flujo de trabajo del proceso de diseño, a modo de ejemplo, puesto que es preciso realizar varias iteraciones entre las distintas etapas. Este flujo es:

- Definición de los requerimientos del sistema a desarrollar.
- Modelo del Sistema.
- Determinación de: plazas, transiciones, sincronización, flujo de ejecución, tiempos, eventos y relación de eventos con transiciones.
- Modelar y refinar el sistema (edición, árbol de alcanzabilidad, sifones, trampas, invariantes, etc.).
- Verificación del modelo, realizando todos los análisis estructurales y dinámicos.
- Definición, codificación y código de control de los procesos del sistema según su responsabilidad (nombre, tipo, transiciones que lo componen).
- Definición y codificación de las actividades de los procesos. Se completan las tareas específicas del problema en particular.
- Validación del correcto funcionamiento del software final.

Es importante, para obtener un buen resultado, validar la RdP antes de comenzar con la definición y codificación de los procesos. Puesto que, si la RdP no se corresponde con los requerimientos, el sistema será erróneo. También es importante verificar que no exista interbloqueo, inanición, etc., esto es importante para obtener un sistema correcto.

Figura 189: Modelo de desarrollo

Caso de Aplicación

Tráfico Marino

En este apartado se estudia el problema y solución del canal marino [242], planteados en el capítulo anterior.

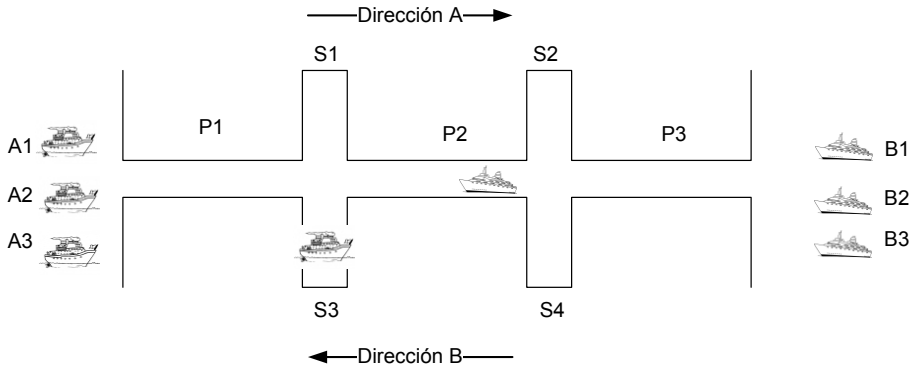


Figura 190:Problema del canal marino

Modelo POPN

Para este caso se ha tomado la RdP de la bibliografía [242] y se le ha realizado unas modificaciones en el manejo de los sifones que introducen plazas de restricción para evitar el interbloqueo. Dando como resultado la RdP de la Figura 191. Posteriormente se han verificado invariantes, vivacidad, interbloqueo e inanición.

Para este caso se ha creado un proceso por cada barco, cada uno de estos procesos es dirigido por la RdP de la Figura 191, que es la red con la que se ha realizado la simulación y verificación del sistema. Las plazas P0 a P5 representan a los barcos que avanzan en la dirección A. Mientras que las plazas P6 a P11, representan a los barcos que van en la dirección B.

El disparo de las transiciones (entradaCanal1d, salidaCanal1d, entradaCanal2d, salidaCanal2d, entradaCanal3d, salidaCanal3d) causan movimientos de tokens entre las plazas P0 a P5, que se corresponde con acción que provoca la circulación de barcos hacia la derecha. El disparo de las transiciones (entradaICanal3i, salidaICanal3i, entradaICanal2i, salidaICanal2i, entradaICanal1i, salidaICanal1i) causa movimientos de tokens entre las plazas de P6 a P11, que se corresponde con acción que provoca la circulación de barcos hacia la izquierda.

Todas las etiquetas de transiciones son del tipo < D; I >, excepto las transiciones que retornan los token a las plazas de inicio de ambos sentidos, que son <A; N> puesto que no es necesario informar al sistema y sólo se requiere cambiar el estado de la red.

En el resto de las transiciones, el pedido de disparo de una transición representa un sensor accionado para pedir “el permiso” y continuar avanzando. Mientras que la recepción del disparo, es el otorgamiento de permisos para avanzar al próximo canal o dársena.

Cada barco o proceso, que avanza a la izquierda o derecha, solicita disparos y espera informes del PP que ejecuta RdP.

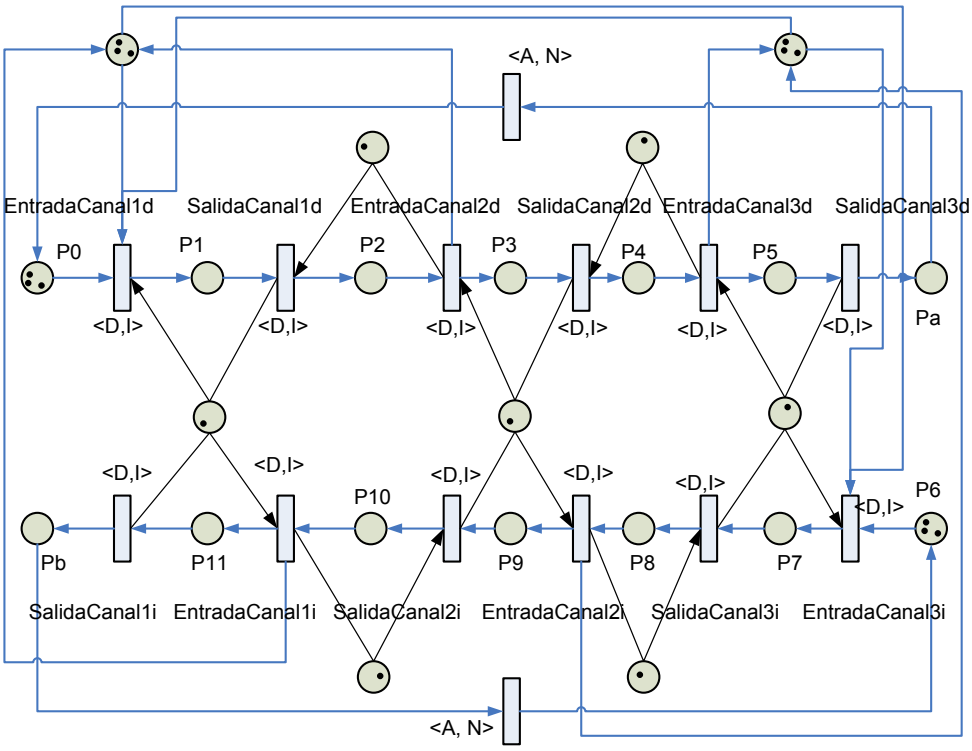


Figura 191: RdP del canal maritímo

Procesos

La RdP que controla el tráfico en el canal está diseñada como una POPN. Donde cada barco está representado por un proceso. La implementación en Java son objetos (hilos) runnable. Estos barcos tienen movimientos que son descritos por uno de los invariantes de transición. El estado local es un token en una de las plazas, que no son recursos o restricciones.

La máxima cantidad de hilos, es igual a la máxima cantidad de barcos en el sistema. Para este caso son seis, lo que resulta de los invariantes de plaza.

La ejecución secuencial de cada barco o hilo es mostrada en la Tabla 127.

Tabla 127: Transiciones que ejecutan y escuchan los hilos para moverse por el canal

Transición (etiqueta)	D/I	D/I	D/I	D/I	D/I	D/I
Barcos que se mueven a la derecha	Entrada Canal1d	Salida Canal1d	Entrada Canal2d	Salida Canal2d	Entrada Canal3d	Salida Canal3d
Transición (etiqueta)	D/I	D/I	D/I	D/I	D/I	D/I
Barcos que se mueven a la izquierda	Entrada Canal3i	Salida Canal3i	Entrada Canal2i	Salida Canal2i	Entrada Canal1i	Salida Canal1i

Clases obtenidas

Las matrices de la RdP de la Figura 191 han sido obtenidas del simulador TINA [251] , y son las usadas para la programación del PP.

Las clases generadas son:

- Iniciador.java
    - Responsable de instanciar tres barcas en cada sentido.
- BarcoADerechaCodGen.java
    - Responsable de representar un barco que se mueve a la derecha. El objeto es creado con la secuencia “Barcos que se mueven a la derecha”.
- BarcoAIzquierdaCodGen.java
    - Responsable de representar un barco que se mueve a la izquierda. El objeto es creado con la secuencia “Barcos que se mueven a la izquierda”.

Resultados obtenidos
El sistema obtenido es un controlador para dirigir el tráfico en el canal. Este controlador permite circulación de tráfico en paralelo y aprovecha en forma concurrente los recursos del canal (dársenas y canales), evitando el interbloqueo y la inanición.

La ejecución se realizó con seis barcos, tres barcos corriendo por el canal a la derecha y tres barcos a la izquierda.

Con el fin de evaluar el sistema resultante, a cada barco se le asignó un tiempo de recorrido completo de 2500ms. Este tiempo es el que el barco tarda en recorrer los tres canales y las dos dársenas sin espera. Y cuando un barco sale del canal regresa a la plaza de espera, para esperar su turno de reingresar.

Los valores obtenidos para una simulación con seis barcos durante 19 segundos son los mostrados en la Tabla 128.

Tabla 128: Tiempos de recorrido de los barcos en el canal

Tiempo total de simulación en milisegundos (ms)	19077 ms
Cantidad de barcos	6
Total de recorridos realizados	33
Tiempo promedio de un recorrido	2270 ms
Tiempo de recorrido máximo	3510 ms
Tiempo de recorrido mínimo	2509 ms

El tiempo de recorrido promedio de los barcos, circulando en el canal con el sistema de control, supera en un 2,7 % y el throughput del sistema, para los 33 recorridos es 4,32 veces, (2500x33)/19077. Estos 33 recorridos en 19077 ms, implica que en el canal durante la ejecución hay 4,45 barcos circulando simultáneamente.

Caso de Aplicación

Celda de Manufactura Flexible

En este apartado se estudia y soluciona el caso de la “celda de manufactura flexible” o “Sistema de manufacturación robotizado” presentado por [2].

Este sistema de manufacturación consta de tres robots, cuatro máquinas y tres tipos diferentes de piezas a procesar, como se observa en la Figura 192.

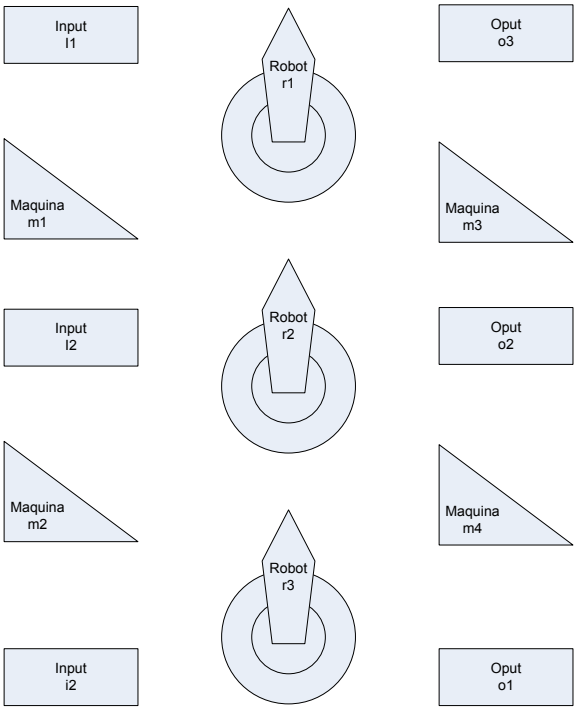


Figura 192: Celda de manufactura flexible

Los robots retiran las piezas de tres inventarios, I1, I2 e I3, y las colocan en las máquinas para su procesamiento. Cuando finaliza el proceso, las retiran y las colocan en las salidas, O1, O2 y O3. Los robots también trasladan las piezas de máquina en máquina en las distintas etapas del proceso. Por lo que, para la manufactura, cada tipo de pieza debe seguir una trayectoria de procesamiento distinta, como se muestra en la Figura 193.

Las trayectorias de las piezas son:

-
- $Pieza\ 1(alternativa\ 2): i1 \xrightarrow{r1} m3 \xrightarrow{r2} m4 \xrightarrow{r3} 01$
- $Pieza\ 2: i2 \xrightarrow{r2} m2 \xrightarrow{r2} o2$
- $Pieza\ 3: i3 \xrightarrow{r3} m4 \xrightarrow{r2} m3 \xrightarrow{r1} o$

La RdP, que se muestra en la Figura 194, modela el sistema (POPN); la RdP ha sido modificada de la presentada en la bibliografía con el fin de evitar interbloqueo.

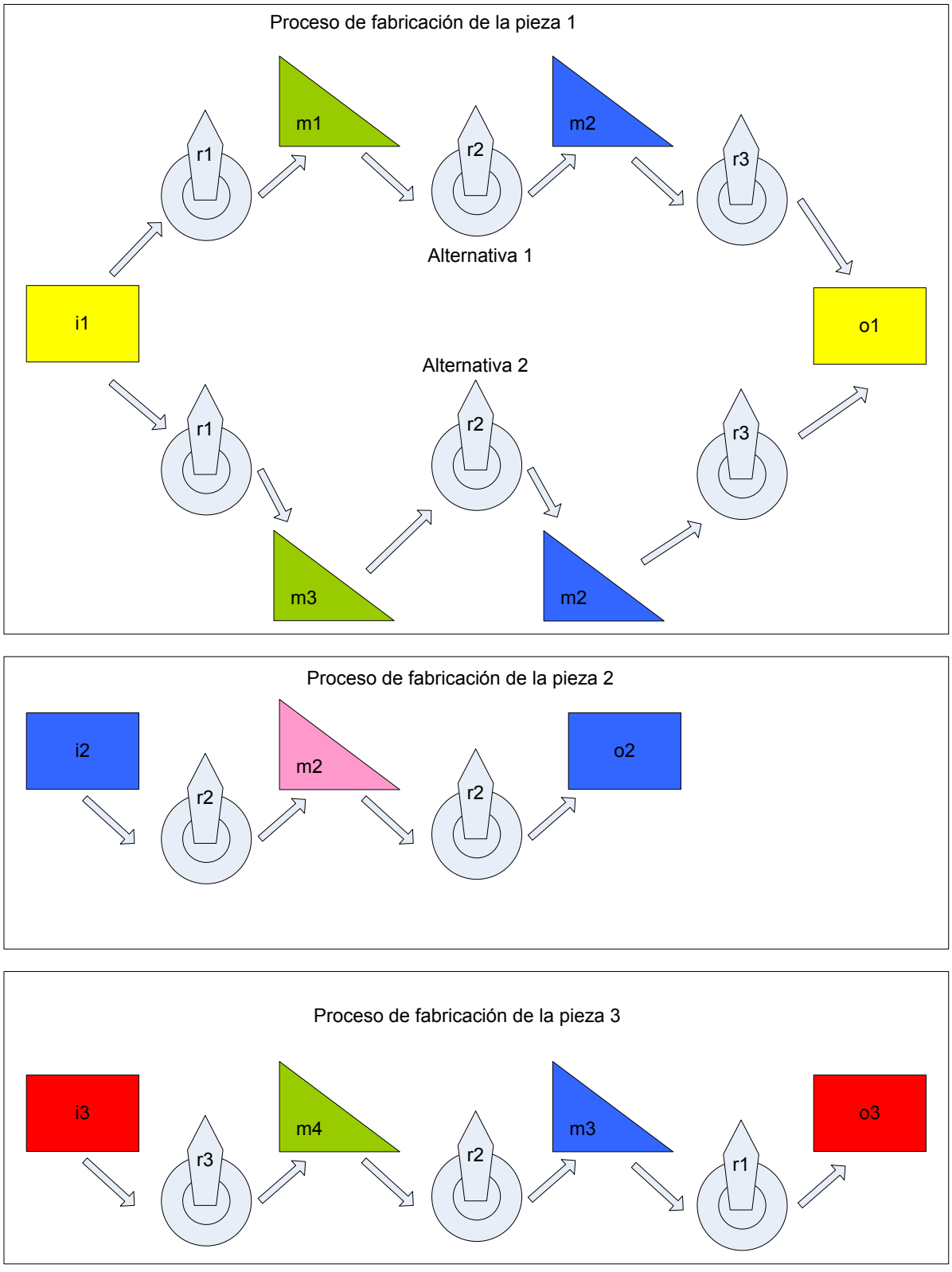


Figura 193: Recorrido de las piezas en la celda flexible

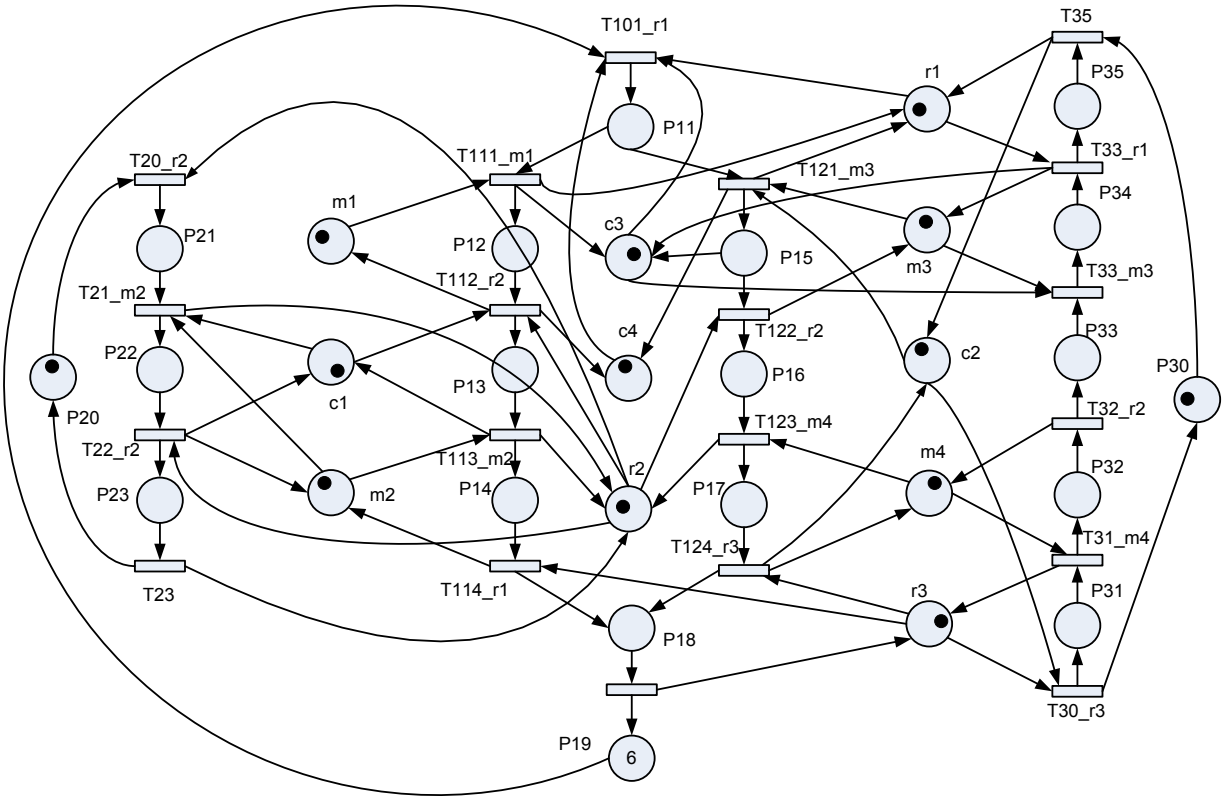


Figura 194: RdP que modela el sistema de celda flexible

Modelo POPN

La RdP en la Figura 194 modela el sistema de manufacturación robotizado. La Tabla 129 muestra las actividades que se realizan en cada plaza, las condiciones de las transiciones y sus etiquetas.

La secuencia de disparos de las transiciones, para fabricar la pieza 1, son dos: alternativa 1 (T101_r1, T111_m1, T112_r2, T113_m2, T114_r3, T102) y alternativa 2 (T101_r1, T211_m3, T122_r2, T123_m4, T124_r3, T102). Las dos alternativas comparten la primera y la última transición. Estos movimientos de tokens completan el circuito, entre las plazas de p10 a p18, que se corresponde con las etapas para fabricar la pieza 2. La Tabla 129, muestra las dos secuencias de disparo, plazas y etiquetas.

Tabla 129: Secuencia de actividades, condiciones y etiquetas para procesar la pieza uno.

Proceso de fabricación de la pieza 1 secuencia uno		
Estado	Disparo	Etiquetas
P10 Input i1, listo	T101_r1 Toma Robo r1 y máquina m1	<A, P10>
P11 Decide que alternativa, según el estado	Dispara en este caso T111_m1	<R1, P11_1>
P12 Material en la máquina m1	T112_r1 Libera el robot r1	<M1, P11_2>
P13 Procesa m1 a la pieza 1	T113_m2 Toma la máquina m2 y el robot r2	<R2, P11_3>
P14 Procesa m2 a la pieza 1	T114_r3 Toma el robot r3	<M2, P11_4>
P18 Coloca la pieza en o1	T102 Libera el robot r3 y la máquina m2	<R3,N>

Proceso de fabricación de la pieza 1 secuencia dos		
Estado	Disparo	Etiquetas
P10 Input i1, listo	T101_r1 Toma Robo r1 y máquina m3	<A, P10>
P11 Decide que alternativa, según el estado	Dispara en este caso T121_m3	<R1, P12_1>
P15 Material en la máquina m3	T122_r2 Libera el robot r1	<M2, P12_2>
P16 Procesa m3 a la pieza 1	T123_m2 Toma la máquina m2 y el robot r2	<R2, P12_3>
P17 Procesa m2 a la pieza 1	T124_r3 Toma el robot r3	<M4, P12_4>
P18 Coloca la pieza en o1	T102 Libera el robot r3 y la máquina m2	<R3, N>

La secuencia de disparos de las transiciones (T20_r2, T21_m2, T22_r2, T23) realizan los movimientos de tokens entre las plazas de p20 a p23, que se corresponde con las etapas para fabricar la pieza 2, esto lo muestra la Tabla 130.

Tabla 130: Secuencia de actividades, condiciones y etiquetas para procesar la pieza dos.

Proceso de fabricación de la pieza 2		
Estado	Disparo	Etiquetas
P20 Input i2, listo	T20_r2 Toma Robo r2 y máquina m2	<A, P2_1>
P21 Material en la máquina m2	T21_m2 Libera el robot r2	<R2, P2_2>
P22 Procesa pieza 2	T22_r2 Toma el robot r2	<M2, P2_3>
P23 Saca pieza de la máquina m2 y la pone en o2	T23 Librea el robot r2	<R2, N>

La secuencia de disparos de las transiciones (t30_R3, t31_M4, t32_R2, t33_M3, t34_R1, t35) realizan el movimientos de tokens entre las plazas de p30 a p35, que se corresponde con las etapas para fabricar el producto 3, esto se muestra en la Tabla 131.

Tabla 131: Secuencia de actividades, condiciones y etiquetas para procesar la pieza tres

Proceso de fabricación de la pieza 3		
Estado	Disparo	Etiquetas
P30 Input i1, listo	T30_r3 Toma Robo r3 y máquina m4	<A, P3_1>
P31 Material en la máquina m2	T31_m4 Libera el robot r3	<R3, P3_2>
P32 Procesa pieza 2	T32_r2 Libera máquina m4	<M4, P3_3>
P33 Saca pieza de la máquina m2 y la pone en o3	T33_m3 Toma Robo r2 y máquina m3	<R2, P3_4>
P34 Saca pieza de la máquina m2 y la pone en la máquina m3	T34_r1 Librea el robot r1 y la máquina m2	<M3, P3_5>
P35 Saca pieza de la máquina m3 y la pone en o3	T34_r1 Librea el robot r1	<R1, N>

Las plazas de m1 a m4 y de r1 a r3 modelan la disponibilidad de los recursos del sistema. Las plazas (c1, c2, c3, c4) son los supervisores para evitar que se vacíen los sifones y prevenir el interbloqueo del sistema.

Las Tabla 129, Tabla 130 y Tabla 131 muestran las etiquetas relacionadas con cada transición. Los nombres indican tareas o eventos asociados para facilitar la lectura, pero en general, todos los pedidos de disp1aros de las transiciones se efectúan cuando un robot o una máquina terminan la tarea en curso, y todos los informes de disparo de las transiciones indican cuando se debe comenzar la próxima etapa de producción. Las etiquetas de las transiciones que comienzan un proceso de producción son de disparo automático, es decir que se van disparando cuando se sensibilizan. Esto permite obtener la mayor performance de producción del sistema y automatizar la inicialización.

Procesos
De la RdP que modela a la celda y de las características del modelado con POPN, podemos inferir que a un recurso le corresponde un objeto "no runnable" por cada recurso del sistema y a un producto le corresponde un proceso "runnable", por cada proceso de fabricación. Es posible reducir la cantidad de hilos, según las restricciones impuestas por las plazas de control de sifones. Puesto que nunca puede haber sifones vacíos y las marcas en los sifones están en los circuitos de recursos, que son restringidos por las plazas de control. Esto nos permite identificar etapas, secuencia de transiciones y plazas, que son independientes en la producción. En esta etapa no puede haber más de un token en un conjunto de plazas que están en una secuencia, esto equivale a decir que no puede haber más de un hilo por etapa independiente. Una etapa independiente, en una línea de fabricación, es aquella que se puede ejecutar en paralelo con otras etapas independientes de la misma línea. Esto nos permite definir un proceso "runnable" por cada etapa independiente (para determinar la máxima cantidad de hilos del sistema).

Por lo tanto hay siete procesos "no runnable" que se corresponden a las máquinas m1, m2, m3 y m4 y los robots r1, r2 y r3. Por otro lado se determinaron ocho procesos "runnable" correspondientes a:

- Producto 1: un proceso para iniciar la producción, cuatro procesos para la alternativa 1 con cuatro etapas independientes y un proceso para la alternativa 2 con cuatro etapas no independientes.
- Producto 2: un proceso para las tres etapas no independientes.
- Producto 3: un proceso para las cinco etapas no independientes.

Clases obtenidas
Las matrices de la RdP de la Figura 194, han sido obtenidas del simulador TINA, y son las usadas para la programación del procesador o simulador.

Las clases generadas son:

- Main.java, clase esponsable de inicializar el sistema.
- Máquina1.java a Máquina4.java, clases responsables y encargadas de controlar las tareas a realizar por las máquinas de la celda flexible, según la acción que le corresponda a cada producto. Además, se encarga de comunicarse con el PP para solicitar los disparos de transiciones que dan inicio o terminación a la tarea.

- Robot1.java a Robot3.java, son clases responsables de controlar las tareas a realizar por los robots de la celda de fabricación flexible, según la acción que le corresponda a cada producto. Además, se encarga de comunicarse con el PP para solicitar los disparos de transiciones que dan inicio o terminación de la tarea.
- Producto10.java, es una clase runnable responsable de controlar las acciones para fabricar el producto uno, que corresponde a la primera etapa de producción que corresponde a la plaza P10. Esta clase realiza las siguientes acciones: empieza en el estado P10, disparo de la transición T101_r1, para esto se comunica con el PP, hace el disparo y espera la confirmación que se haya realizado. Después de la confirmación llama al robot1 para que realice la tarea de esta primera etapa de producción.
- Producto11e1.java a Producto11e4.java, son clases runnables, responsables de controlar las acciones a realizar en cada etapa de producción para el producto 1 alternativa 1. Estas clases se comunican con el PP para esperar el informe de disparo de las transiciones T112_r1, T112_m2 y T114_r3, que correspondan al producto 1. En cada plaza llama al componente (máquina o robot) que realice la tarea de la etapa de producción que continúe en el proceso de producción. En este caso, las acciones a realizar en las plazas P11, P12, P13 y P14, son independientes entre ellas; se generó una clase individual para cada etapa, esto permite la ejecución paralela.
- Producto12.java, es una clase runnable responsable de controlar las acciones para fabricar el producto 1 alternativa 2. Esta clase se comunica con el PP para esperar el informe de disparo de las transiciones que correspondan al producto 1-2, y luego llamar al componente (máquina o robot) que realice la tarea de la etapa de producción que continúe en el proceso de producción.
- Producto2.java, es una clase runnable responsable de controlar las acciones a realizar en cada etapa de fabricación del producto 2. Esta clase se comunica con el PP para esperar el informe de disparo de las transiciones que correspondan al producto 2, y luego llamar al componente (máquina o robot) que realice la tarea de la etapa de producción que continúe en el proceso de producción.
- Producto3.java, es una clase runnable responsable de controlar las acciones a realizar en cada etapa de producción para el producto 3. Esta clase se comunica con el PP para esperar el informe de disparo de las transiciones que correspondan al producto 3, y luego llamar al componente (máquina o robot) que realice la tarea de la etapa de producción que continúe en el proceso de producción.

Resultados obtenidos

Las clases obtenidas han sido completadas con los métodos que realizan las acciones. Por ejemplo, una clase cuando recibe la conformación del disparo de la transición T20_r2, se le indica al robot r2 que saque una pieza de input i1 y la coloque en la máquina m2. Todas estas acciones son métodos que solo tienen códigos secuenciales.

El sistema obtenido es un controlador que controla una celda de fabricación flexible. Este controlador permite la realización de tareas en paralelo y aprovecha en forma concurrente los recursos del sistema (input, output, máquinas y robot), evitando el interbloqueo y la inanición.

En la ejecución, de las transiciones que inician un proceso para producir una pieza, son de disparo automático, permitiendo iniciar la producción de un nuevo producto cada vez que se encuentren sensibilizadas las transiciones, lo que evita demoras.

Con el fin de verificar el sistema, a cada máquina se le asignó una demora de 1000ms como tiempo de ejecución en cada etapa, para todas las piezas. En cambio, para los robots, tanto para poner o sacar una pieza se programó una demora de 100ms, corresponde al traslado de un objeto de un punto al otro. De esta forma, el tiempo de producción mínimo de los productos 1 y 3 es de 2300ms y el tiempo mínimo del producto 2 es de 1200ms. Estos tiempos mínimos se determinaron considerando que el producto no sufra demoras por falta de recursos.

Con este modelo se logró una corrida de 3 minutos 11 segundos y un total de 255 piezas.

Para cada línea en particular se obtuvieron los valores que muestra la Tabla 132.

Tabla 132: Resultados de tiempos de fabricación y throughput en la celda flexible

Producto	total	Tiempo medio en milisegundos	Throughput Producto/t-mínimo
Producción 1-1	85	3187	1,02
Producción 1-2	46	2312	0,55
Producción 2	86	2112	0,54
Producción 3	38	2366	0,46
Total	255		2,57

De la Tabla 132, se observa que los totales de producción de 1-2 y 3 son más bajos que los totales obtenidos para los productos 1−1 y 2. Esto es debido a la mutua exclusión que impone la máquina m3, que impide el paralelismo y disminuye la producción de estas líneas.

En el caso de los productos 3 y 1-2, se observa poca interferencia debido a que entre ellos no tiene exclusión mutua por el uso de dos máquinas. Es decir, que cuando comienza el proceso se dispone de las máquinas M3 y M4 sin demoras. Además solo se comparte los robots R1 y R3 con el producto 1-1. Por otro lado, el producto 2 tuvo el mayor incremento, debido a que utiliza dos veces al robot R2, el cual es muy requerido.

El throughput total obtenido es de 2,57 es decir que se comparten los recursos en las tres líneas y se pierde solo 14% de la capacidad de fabricar en forma individual cada pieza.

También observamos que el producto 1-1 obtuvo un throughput cercano al doble que el resto, lo que es consecuencia del paralelismo entre sus etapas de producción. En cambio, en los productos 1 -2, 2 y 3 no existe este paralelismo entre etapas, resultando en un throughput mucho menor.

También hay que destacar que se han hecho mediciones cambiando las prioridades con el fin de modular la producción de cada línea. Para el cambio de las prioridades sólo se ha tenido que modificar la matriz de prioridades.

Esto nos ha permitido evitar inanición en el proceso de producción 1-2, puesto que es el que más comparte recursos. Esta matriz se encarga de establecer prioridades de disparo a las transiciones para el disparo y ha sido cambiada en tiempo de ejecución para obtener una cantidad específica de piezas.

Resultados y Conclusiones

Este desarrollo se realizó con diferentes casos de estudio, de los cuales sólo se exponen dos.

Se muestra que es posible relacionar los procesos con el modelo del sistema concurrente y paralelo, modelado con una RdP que se ejecuta en el PP.

Además, de manera sencilla, con las etiquetas de transición se vincularon los procesos y recursos reales del sistema con los hilos y objetos del programa. Estas etiquetas son la relación entre los procesos y el PP. La vinculación soluciona problemas reales y complejos de forma simple y directa.

De acuerdo con los casos resueltos, vemos que el desarrollo es claro y la programación sólo requiere de acciones secuenciales de objetos que son recursos o actividades. Se obtuvieron resultados exitosos con el código secuencial que refleja la secuencia que conduce la RdP, al cual sólo se le completaron las tareas que ejecutan acciones específicas de los casos.

También, los tiempos de desarrollo se redujeron para los casos abordados.

Se ha programado a la RdP en forma directa en el PP, y se ha obtenido los objetos e hilos requeridos por el programa. Lo cual simplifica la verificación del sistema obtenido.

Capítulo 8

Resumen de Resultados, Conclusiones, Contribuciones y Líneas Abiertas

Introducción

En este capítulo, presentaremos un resumen de los resultados, la conclusión y las contribuciones principales y, por último, las líneas de investigación abiertas.

A continuación sintetizaremos la propuesta de la tesis respecto del PP y HPP**,** en el contexto de nuestro planteo y con respecto de los trabajos vinculados; seguidamente, las conclusiones y las contribuciones principales detallando los aportes más significativo de nuestra propuesta; finalmente, en la sección líneas de investigación abiertas, presentaremos distintas líneas de investigación que permitirán continuar con el trabajo desarrollado en esta tesis.

Resumen de Resultados

Al comenzar la investigación para realizar esta tesis se planteó como objetivo principal el desarrollo de dos procesadores, basado en RdP con distintos tipos de brazos y sus extensiones temporales, para la ejecución directa de las matrices y vectores. Estos procesadores son aplicados a sistemas embebidos con alta concurrencia, reactivos y de tiempo real, esencialmente en la ejecución del modelo, la comunicación con el entorno y la generación de código.

Antes de la elaboración de esta tesis ya se conocían trabajos relacionados, los que han sido citados a lo largo del texto, que incluyen la codificación para la ejecución de RdP y las RdP con tiempo, pero éstos en su mayoría han sido usados para el análisis de las propiedades del modelo. También existen trabajos previos sobre implementación de RdP para dirigir la ejecución de procesos.

Lo novedoso en esta tesis es la realización de una familia de procesadores que permiten la ejecución de los algoritmos modelizados con RdP heterogéneas. Asimismo se sitúa en un entorno uniforme que tiene a las RdP como formalismo base y las utiliza tanto para el modelado, análisis e implementación del sistema objetivo.

A lo largo de este trabajo se ha reconocido que la magnitud era demasiado grande para una tesis doctoral, motivo por el cual nos enfocamos en construir la familia de procesadores, concentrándonos en la obtención directa de código, la relación del procesador con su entorno, la división de la red para obtener paralelismo, con reducción de recursos.

Por lo dicho en el párrafo precedente y con el objeto de enmarcar la tesis en un entorno adecuado, en el Capítulo 2 y Anexos A al F, se hizo un estudio de antecedentes y fundamentos para abordar las distintas etapas que se requerían para el desarrollo de la familia de procesadores. Estos son: el procesador/controlador y sus fundamentos, la interconexión del procesador con el entorno, los algoritmos de disparo de las transiciones, la división de la red y los tipos de red que soporta el procesador.

Así mismo, se introdujeron los formalismos necesarios en los anexos: “RdP con tiempo”, “Monitores” y “RdP no-autónomas”. También se ha incluido una breve descripción de los IP-

Core, en el anexo “IP-Core”. En resumen son una serie de conceptos necesarios sobre implementación de RdP, que han sido nesesarios para el desarrollo de la tesis.

En la parte final del Capítulo 2, se realizó una tabla que muestra las características implementadas por distintos investigadores con respecto al procesamiento y/o generación de software a partir de modelos realizados con RdP y podemos ver que son muy pocas las características por ellos implementadas, sin embargo éstas han sido logradas en este trabajo.

En el Capítulo 3 se estableció el costo para obtener la sincronización entre procesos y los mecanismos para la ejecución concurrente. También se determinó dónde incluir en la arquitectura de un SMP el PP o HPP para conformar una arquitectura heterogénea y lograr un mejor desempeño (respuesta en tiempo real) sin modificar el ISA. Donde los modelos de PP y HPP resultaron de simple programación.

En el Capítulo 4 se validó la simulación realizada en el Capítulo 3 construyendo el PP para RdPnA. Se comprobó su correcto funcionamiento y las ventajas que éste aporta para los sistemas que requieren de sincronización, no sólo para mejorar el tiempo de ejecución, sino también en cuanto a la forma de resolver problemas de sincronización que utiliza las RdP y desacopla la lógica del programa paralelo de las tareas secuenciales a realizar.

En el Capítulo 5 se han diseñado, implementado y probado un PPcT y un PPTm que son capaces de:

- Proveer sincronización entre los múltiples procesos con ejecución concurrentemente en los sistemas SMP. Se ha obtenido, para los problemas planteados, tiempos de sincronización de hasta 7 (siete) veces más rápido que con el uso de semáforos.
- La programación del PPcT y PPTm es directa, ya que solo hay que cargar los vectores y matrices del modelo realizado con RdP, en el procesador.

Asegurando que todas las propiedades y funcionalidades que se cumplen en el modelo, se siguen cumpliendo en la implementación.

También se ha ampliado el PP con plazas acotadas, brazos: inhibidores, de lectura, stop-watch, stop-watch inhibidores y de reset.

Los inconvenientes encontrados en esta etapa de la implementación son el crecimiento de los recursos de hardware que lo hacen con el producto plazas transiciones.

En el Capítulo 6 se ha logrado disminuir la cantidad de información necesaria para expresar el modelo con una RdP. El hecho de dividir las redes de esta forma, genera una jerarquía explícita de redes, con un concepto más amplio, que llamamos ¨Redes Jerárquicas Divididas por transiciones¨. El conjunto de subredes relacionadas mediante la división de las transiciones distribuidas provoca una redución teórica de la cantidad de recursos que puede alcanzar hasta un 80%.

Los resultados en las mejoras de tiempo, obtenidos en pruebas realizadas con el HPP frente a los semáforos, son del 20% al 37% (son idénticos a los valores obtenidos con el PP). Mientras los recursos de hardware se reducen entre el 40% y el 60% en comparación con los necesarios para implementar el PP.

La mejora en el desempeño, con respecto al tiempo de ejecución, se hace más significativa a medida que las aplicaciones poseen una mayor demanda de sincronización. Para aplicación con

poca sincronización, este aumento de rendimiento no es significativo, pero aún así el uso del HPP facilita la programación directa a partir de la RdP, que es el modelo creado para verificar el sistema. Esto disminuye los tiempos de codificación; además mantiene la versatilidad de las RdP para modelar en entornos concurrentes y paralelos, lo que hace que el sistema resultante sea flexible y se facilite su mantenimiento.

Esta propuesta de división tiene ventajas adicionales que son:

- Permite múltiples disparos simultáneos de la RdP, sin más hardware.
- Si la división de la red se realiza agrupando en cada subred los brazos inhibidores, la parte de la red que tenga arcos con peso más de uno, las transiciones con tiempo, las transiciones temporizadas, etc. Es posible instanciar varias subredes con elementos en común para cado caso. Con estas subredes, se obtiene una disminución adicional de recursos puesto que cada subred implementa sólo el tipo de característica que se requiere.

Para esta familia de procesadores, a diferencia de otros enfoques, se ha mantenido un método de traducción de software para programar el procesador ejecutando sólo la ecuación de estado lo que ha sido resuelta con funciones booleanas. Esto último ha facilitado configurar y ampliar el PP o HPP incorporando distintos brazos y tiempo, según se lo requiera. También se ha mantenido, en toda la familia, dos ciclos para disparar una transición y comunicar el disparo.

En el Capítulo 7 se muestra como relacionar (comunicar) el modelado (POPN) con una RdPnA que se ejecuta en el PP o HPP. Las etiquetas programables de transición vinculan los procesos y recursos reales del sistema (componente) con los hilos y objetos del programa. Estas etiquetas son la relación entre los procesos y el PP o HPP.

Los casos tratados muestran cómo la vinculación soluciona problemas reales y complejos de forma simple y directa. El desarrollo que resulta es claro y la programación sólo requiere de acciones secuenciales de objetos, que son recursos o actividades encapsuladas en componentes.

Se obtuvieron resultados exitosos con el código secuencial que refleja la secuencia que es conducida por la RdP (PP o HPP). A este código sólo se le completaron las tareas que ejecutan acciones específicas de los casos.

También, los tiempos de desarrollo se redujeron para los casos abordados. Puesto que, se ha programado la RdP en forma directa en el PP o HPP, y se han establecido criterios para obtener los objetos e hilos requeridos por el programa.

Conclusiones

Destacamos la creación de una familia de procesadores de soft-Core que hemos llamado “Procesador de Petri” (PP), “Procesador de Petri Jerárquico” (HPP) con tiempo y temporizado, que ejecutan métodos formales (RdP de bajo nivel) facilitando el desarrollo de sistemas reactivos, concurrentes y de tiempo real en sistemas embebidos. Esta familia permite combinar distintos tipos de RdP para obtener un sistema jerárquico y/o distribuido y como consecuencia reducir los tiempos de respuesta en sistemas reactivos y la cantidad de hardware utilizado. Todo esto redunda en el aumento de la fiabilidad del sistema resultante, ya que soporta la validación de requerimientos funcionales y temporales, y la detección de errores en etapas tempranas del ciclo de vida de desarrollo. Ya que el código ejecutable de todos los modelos se obtiene en forma directa y exacta.

Esta familia de procesadores ha sido la motivación de esta tesis, en la cual, se ha logrado mantener el formalismo de las RdP con el fin de facilitar su uso en todo el ciclo de desarrollo. Además se ha tratado en profundidad y solucionado el problema que ocasiona el aumento de recursos. Y haciendo uso de un nuevo método de división, que hemos llamado "Redes Jerárquicas Divididas por Transiciones" y la "Localidad Estructural" de la RdP, la reducción de recursos lograda es mayor.

También hemos puesto énfasis en cómo relacionar el PP y el HPP con el medio ambiente, lo que se ha solucionado con las colas de salida y entrada, evitando una cola FIFO que introduciría una sobrecarga de tiempo y recursos.

A partir de la familia de procesadores creada, diseñada e implementada podemos extraer las siguientes conclusiones para su uso y aplicación:

1. La inclusión de estos procesadores como soporte en una metodología de desarrollo permite obtener sistemas robustos. Dado que las RdPnA, con y sin tiempo, admiten el modelado de un gran número de situaciones propias de los sistemas concurrentes, reactivos y de tiempo real, de manera sencilla, ya que es un formalismo fácil de comunicar dada su naturaleza gráfica.
2. Facilitan la generación del código paralelo en forma automática, o casi automática, a partir del modelo. Los errores en los sistemas resultantes serían como consecuencia de fallas del modelado.
3. Los cambios en las etapas de desarrollo y/o mantenimiento pueden ser convertidos en código que los mismos procesadores (RdP, prioridades, vinculación con los eventos, etc.) ejecutan inmediatamente. Es tan sencillo como cambiar las matrices y vectores.
4. El procesador admite la ejecución de sistemas descentralizados conformados por múltiples PP interconectados, reconociendo las actividades (procesos) concurrentes que conforman el sistema, a través de las redes jerárquicas divididas por transiciones.
5. El HPP es capaz de soportar mecanismos de comunicación entre los procesos, proveyendo diferentes primitivas síncronas y asíncronas. Todo ello para una clase muy amplia de diferentes RdP con y sin tiempo, sin las limitaciones de los trabajos expuestos en el Capítulo 2.
6. No menos importante, ha sido mantener la programación de esta familia de procesadores de manera simple e implementar los mismos casi exclusivamente con funciones booleanas, mientras la semántica de todas las RdP consideradas se mantienen aún realizando la división.
7. El código generado para cualquier procesador de esta familia es independiente de los procesadores que colaboran (SMP).
8. Los procesadores que pueden colaborar no requieren ser modificados (ISA).

Contribuciones

1. Se ha transformado una arquitectura multi-Core (SMP) en una arquitectura heterogénea. Para esto se ha determinado el lugar donde incorporar el PP o HPP sin alterar el ISA del resto de los core. Se ha propuesto una arquitectura óptima y otras subóptimas, según los tiempos de respuesta para acceder al PP, también se han establecido límites de aplicación del PP.

2. Se ha construido el hardware de una familia de procesadores, con un IP-Core en una FPGA, integrado a un arquitectura multi-Core, que ejecuta distintos tipos de RdP simultáneamente. Se ha obtenido mejoras en los tiempos de respuesta para soluciones que requieren de alta relación entre la sincronización y el cálculo. Los tipos de redes ejecutadas por el PP y el HPP son: RdP ordinarias, RdP Plaza/Transición, RdP temporales, RdP temporizadas, RdP extendidas (con brazos: reset, inhibidor, de lectura, stop-watch, stop-watch inhibidores y plazas acotadas), RdP distribuidas y la combinación de distintos tipos de RdP. También se ha construido un simulador del PP y del HPP que permite desarrollar y probar sistemas en configuraciones multi-Core sin el procesador respectivo.
3. Se ha conectado al procesador con los procesos, esto se ha logrado programando: los tipos de eventos que soporta el procesador (perenne, no-perenne y simultáneo), la relación entre los procesos y las transiciones (por medio de etiquetas), y un driver que integra al procesador con el sistema operativo (descripto en [241]).
4. La programación de la familia de procesadores es directa, a partir de matrices y vectores que representan a las RdP. Las matrices son: de incidencia, tiempos, brazo reset, brazos inhibidores y lectores, brazos stop-watch y stop-watch inhibidores y prioridades. Los vectores son: cota de plaza, relación de la RdP con los eventos y su comportamiento. Todo esto sin realizar ninguna compilación, permitiendo la programación de distintos tipos de RdP en un mismo procesador sin interferencia de las distintas semánticas, combinar distintos tipos de redes según se requiera, reprogramar dinámicamente al procesador, las prioridades, la relación entre los eventos y los procesos, el comportamiento de los eventos de entrada y salida, y el comportamiento de las transiciones.
5. Se ha logrado disminuir la cantidad de recursos que requiere el PP en la FPGA, para lo cual se ha introducido un nuevo método de división de las RdP jerárquicas y una semántica de disparo de transición equivalente a la semántica convencional que permite distribuir el PP en una FPGA y/o interrelacionar múltiples PP para obtener un HPP.
6. El HPP tiene la capacidad para resolver, concurrentemente por conjuntos, múltiples disparos lo que disminuye los tiempos de consulta y decisión.
7. Los procesadores permiten que los programas ejecutados cumplan con los formalismos de las RdP extendidas y sincronizadas, y los resultados de su ejecución sean determinísticos.
8. Se ha logrado que la implementación por hardware del procesador tenga tiempos de respuesta de dos ciclos por consulta (entre la solicitud de un disparo y la respuesta incluyendo la comunicación).

La familia de procesadores diseñados ha sido patentada, obteniéndoles una patente y dos más están en trámite.

Líneas de Investigación Abiertas

A partir de la tesis realizada, se abren distintas posibles líneas de investigación asociadas que no fueron consideradas en la investigación de la tesis pero que ameritan ser tenidas en cuenta en futuros trabajos.

Básicamente las líneas de investigación abiertas están encaminadas a: ampliar la capacidad, del PP y del HPP, y a profundizar en aspectos como la generación automática de códigos.

Podemos destacar:

1. Ampliar la capacidad del procesador en cuanto a la cantidad de plazas transiciones, a más de 1000. En la investigación realizada se ha logrado un HPP con 160 plazas y 160 transiciones, esto ha sido limitado por dos cuestiones, que son: mantener el número de ciclos de disparo de una transición en dos y usar una FPGA de un tamaño muy limitado (Spartan-6).
2. Ampliar las características del procesador para soportar técnicas de tolerancia a fallos, por ejemplo disparos transaccionales. Lograr que el PP realice disparos reversibles es útil para aplicaciones transaccionales y para la verificación de redes.
3. Ampliar el procesador para soportar RdP bien formadas [1] y atender el tratamiento de situaciones complejas con modelos más compactos.
4. Ampliar el Framwork presentado en [252], para que soporte la generación automática de código y patrones de diseño.
5. Incluir el PP en un sistema operativo para sistemas embebidos.
6. Implementar un simulador de PP haciendo uso de SSE (Streaming SIMD Extensions).
7. Hacer un estudio comparativo de consumo energético entre soluciones con el PP y sin el PP.

Referencias Bibliográficas

[1] M. Diaz, *Petri Nets Fundamental Models, Verification and Applications*. NJ USA: John Wiley & Sons, Inc, 2009.

[2] M. Z. Naiqi Wu, *System Modeling and Control with Resource-Oriented Petri Nets*. Boca Raton, FL, 2010.

[3] P. Pawlewski, *Petri Nets: Applications*. Vukovar, Croatia, 2010.

[4] J. Campos, "Evaluación de Prestaciones de Sistemas Concurrentes Modelados con Redes de Petri," *Departamento de Informática e Ingeniería de Sistemas–Universidad de Zaragoza ago/04.[19] K. Jensen "Coloured Petri Nets–Basic Concepts, Analysis Methods and Practical Use,* vol. 1.

[5] B. K. Choi and D. Kang, *Modeling and Simulation of Discrete Event Systems*: John Wiley & Sons, 2013.

[6] K. A. D'Souza and S. K. Khator, "A survey of Petri net applications in modeling controls for automated manufacturing systems," *Computers in Industry,* vol. 24, pp. 5-16, 1994.

[7] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE,* pp. Vol. 77, No. 4, pp. 541-580, 1989.

[8] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. NJ: Prentice Hall PTR Upper Saddle River, NJ, USA, 1981.

[9] S. Benardin, "Building Stochastic Petri Net models for the verification of complex software systems," *PHD Paper, Torino,* 2002.

[10] C. Rivera, "Utilización de redes de Petri para la elaboración de una interfaz de usuario," Universidad Michoacana de San Nicolás de Hidalgo, México2000.

[11] M. Domeika, *Software Development for Embedded Multi-core Systems*. 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA Linacre House, Jordan Hill, Oxford OX2 8DP, UK, 2008.

[12] P. Mehta, "Unleash the Power of Multi Core Using a Platform Approach," *Multicore Expo, March,* 2006.

[13] A. Vajda, "Multi-core and Many-core Processor Architectures," in *Programming Many-Core Chips*, ed: Springer, 2011, pp. 9-43.

[14] T. Co. (2015). *Adapteva*.

[15] R. A. B. David R. Martinez, M. Michael Vai, *High Performance Embedded Computing Handbook A Systems Perspective*. Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Massachusetts, U.S.A.: CRC Press, 2008.

[16] W. Wolf, *High-performance embedded computing: architectures, applications, and methodologies*: Morgan Kaufmann, 2010.

[17] B. G. L.-M. Traonouez, C. Jard, D. Lime and O. H. Roux3, "Symbolic Unfolding of Parametric Stopwatch Petri Nets," *Universita di Firenze, ENS Cachan & INRIA, IRISA, Ecole Centrale de Nantes & Universite de Nantes, IRCCyN,* 2003.

[18] A. Agarwal and M. Levy, "The kill rule for multicore," in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*, 2007, pp. 750-753.

[19] O. Micolini, E. Arlettaz, S. H. Birocco Baudino, and M. Cebollada, "IP Core Procesador de redes de Petri Jerárquicas," in *XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Argentine Symposium on Technology (AST)(Buenos Aires, 2014)*, 2014.

[20] J. W. S. Jeff MageeandJeff Kramer, *Concurrency—State Models & Java Programs,*, 2nd Edition ed. The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England,Telephone (+44) 1243 779777, 2006.

[21] M. M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, Second Edition ed., 2006.

[22] M. R. Albizu, D. P. J. T. González, and D. O. G. Alonso, "Especificación de Sistemas Reactivos Distribuidos utilizando Estelle Síncrono," Citeseer, 2002.

[23] E. Bainomugisha, A. L. Carreton, T. Van Cutsem, S. Mostinckx, and W. De Meuter, "A survey on reactive programming," in *ACM Computing Surveys*, 2012.

[24] D. A. P. John L. Hennessy, *Computer Architecture A Quantitative Approach*, Fourth Edition ed.: Denise E. M. Penrose, 2007.
[25] H. Nilsson, J. Peterson, and P. Hudak, "Functional hybrid modeling," in *Practical Aspects of Declarative Languages*, ed: Springer, 2003, pp. 376-390.
[26] J. Peterson and G. Hager, "Monadic robotics," in *ACM SIGPLAN Notices*, 1999, pp. 95-108.
[27] J. Peterson, P. Hudak, A. Reid, and G. Hager, "FVision: A declarative language for visual tracking," in *Practical Aspects of Declarative Languages*, ed: Springer, 2001, pp. 304-321.
[28] M. Sperber, "Developing a stage lighting system from scratch," in *ACM SIGPLAN Notices*, 2001, pp. 122-133.
[29] A. C. Giorgio Bruno, Gianpaolo Macario and Marco P. Pescarmona, "Scheduling hard real time systems using high-level Petri nets," in *Application and theory of Petri nets 1992*, ed Berlin: Springer Verlag, 1992, pp. Volume 616/1992, 93-112.
[30] C. G. a. M. P. Miguel Felder, "High-Level Timed Petri Nets as a kernel for executable specifications," in *Real-Time Systems*, ed: springerlink, 1993, pp. Volume 5, Numbers 2-3, 235-248.
[31] P. M. G. del Foyo and J. R. Silva, "Using time Petri nets for modelling and verification of timed constrained workflow systems," in *ABCM Symposium Series in Mechatronics*, São Paulo, 2008, pp. 471-478.
[32] B. Berthomieu and M. Diaz., "Modeling and verification of time dependent systems using time Petri nets," in *Software Engineering, IEEE Transactions on*, 1991, pp. 259 - 273.
[33] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Massachusetts Institute of Technology Cambridge, MA, USA ©1974 Massachusetts1974.
[34] R. Pais, Barros, J.P. ; Gomes, L., "A Tool for Tailored Code Generation from Petri Net Models," *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on,* vol. 1, pp. 857-864, 2005.
[35] R. David and H. Alla, *Discrete, continuous, and hybrid Petri nets*. Springer Science & Business Media, 2010.
[36] G. B. a. E. Vicario, "Compositional validation of time critical systems using communicating time Petri nets," *IEEE transactions on Software Engineering,,* pp. 969-992, 1995.
[37] F. Anzai, Kawahara, N. ; Takei, T. ; Watanabe, T. ; Murakoshi, H. ; Kondo, T. ; Dohi, Y., "Hardware Implementation of a Multiprocessor System Controlled by Petri Nets," *IEEE Control, and Instrumentation,* vol. 1, pp. 121 - 126, 1993.
[38] H. S. Murakoshi, M. ; Ding, G. ; Oumi, T. ; Sekiguchi, T. ; Dohi, Y., "Memory reduction of fire unit for Petri net controlled multiprocessor," *IEEE, Industrial Electronics, Control and Instrumentation,* vol. 3, pp. 1961-1966, 1991.
[39] H. Murakoshi and Y. Dohi, "Petri net based high speed programmable controller," *Trans. Society of Instrument and Control Eng,* vol. 27, pp. 466-473, 1991.
[40] H. Murakoshi, Y. Dohi, and T. Sekiguchi, "Proposal of Division Strategy for Large Petri Net Controller," in *Emerging Technologies and Factory Automation, 1992. IEEE International Workshop on*, 1992, pp. 122-127.
[41] Y. Dohi, E. Nomura, T. Shimoda, and H. Murakoshi, "Petri Net Controller with Hardware to Avoid Deadlocks," in *Industrial Electronics, Control, and Instrumentation, 1996., Proceedings of the 1996 IEEE IECON 22nd International Conference on*, 1996, pp. 457 - 462.
[42] H. S. Murakoshi, M. ; Ding, G. ; Oumi, T. ; Sekiguchi, T. ; Dohi, Y., "A high speed programmable controller based on Petri net," *IEEE, Industrial Electronics, Control and Instrumentation,* vol. 3, pp. 1966 - 1971, 1991.
[43] H. S. Murakoshi, M. ; Ding, G. ; Oumi, T. ; Sekiguchi, T. ; Dohi, Y., "A high speed programmable controller based on Petri net," *IEEE, Industrial Electronics, Control and Instrumentation,* vol. 3, pp. 1966 - 1971 1991.

[44] L. B. Gomes, J.P. ; Lima, P., "From Petri net models to VHDL implementation of digital controllers," *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE,* pp. 94 - 99, 2007.
[45] L. Gomes, J. P. Barros, A. Costa, R. Pais, and F. Moutinho, "Towards usage of formal methods within embedded systems co-design," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, 2005, pp. 4 pp.-284.
[46] L. B. Gomes, J.P. ; Costa, A. ; Nunes, R., "The Input-Output Place-Transition Petri Net Class and Associated Tools," *Industrial Informatics, 2007 5th IEEE International Conference* pp. 509 - 514, 2007.
[47] J. B. L. Gomes, and R. Pais., "From Non-Autonomous Petri net Models to Code in Embedded Systems Design.," *Second International Workshop on Discrete-Event System Design (DESDes'04),,* 2004.
[48] L. Gomes, "On conflict resolution in Petri nets models through model structuring and composition," in *Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on*, 2005, pp. 489-494.
[49] F. P. Rogério Campos-Rebelo, Filipe Moutinho, Luís Gomes, "From IOPT Petri nets to C: an Automatic Code Generator Tool," *IEEE,* pp. 390-395, 2011.
[50] R. Pais, Barros, J.P. ; Gomes, L., "From Petri net models to C implementation of digital controllers " *Emerging Technologies and Factory Automation. ETFA 2005. 10th IEEE Conference on,* 2010.
[51] J. Barros, L. Gomes, R. Pais, and R. Dias, "From Petri nets to executable systems: an environment for code generation and analysis," 2004.
[52] J. e. P. Filipe Moutinho, Lu´ıs Gomes, "Configuring communication nodes for networked embedded systems specified by Petri nets," *Industrial Electronics (ISIE), 2013 IEEE International Symposium on,* pp. 1 - 6, 2013.
[53] F. M. a. L. Gomes, "Asynchronous-Channels and Time Domains Extending Petri Nets for GALS Systems," *Springer Boston,* 2012.
[54] R. C. Ferreira, A. ; Gomes, L. , "Intra- and inter-circuit network for Petri Nets based components," *Industrial Electronics (ISIE), 2011 IEEE International* pp. 1529 - 1534, 2011.
[55] L. Gomes, J. M. Fernandes, and I. Global, *Behavioral modeling for embedded systems and technologies: applications for design and implementation*: Information Science Reference, 2010.
[56] A. Costa and L. Gomes, "Petri net Splitting Operation within Embedded Systems Co-design," in *Industrial Informatics, 2007 5th IEEE International Conference on*, 2007, pp. 503-508.
[57] H.Murakoshi, "An Improvement of Petri Net Controlled Multiprocessor," *Trans. of IEICE,* 1990.
[58] H. Murakoshi, "Control of Multi-Microprocessor Based on Petri-Net," *Trans, of IEICE,* vol. 70, pp. 1285-1293, 1987.
[59] Y. YNakamura, T. Taniguchi, H. Murakoshi, and Y. Dohi, "Implementation of a Petri net based parallel programming language PNPL on workstation," in *Industrial Electronics, Control, and Instrumentation, 1993. Proceedings of the IECON'93., International Conference on*, 1993, pp. 127-132.
[60] G. A. Bundell, "An FPGA Implementation of the Petri net Firing Algorithm," The University of Western Australia1997.
[61] J. L. Peterson, "Petri Nets," *Computing Surveys,* vol. 9, pp. 244-252, 1977.
[62] P. A. Laplante, "REAL-TIIME Systems design and analysis," *IEE,* 1993.
[63] D. Kumar and S. Harous, "An approach towards distributed simulation of timed petri nets," in *Simulation Conference, 1990. Proceedings., Winter*, 1990, pp. 428-435.
[64] G. Chiola and A. Ferscha, "Distributed simulation of timed Petri nets: Exploiting the net structure to obtain efficiency," in *Application and Theory of Petri Nets 1993*, ed: Springer, 1993, pp. 146-165.
[65] B. Bütler, R. Esser, and R. Mattmann, "A distributed simulator for high order Petri nets," in *Advances in Petri Nets 1990*, ed: Springer, 1991, pp. 47-63.

[66] M. Gourgand and D. Hill, "Petri nets modelling on transputers with OCCAM 2," in *Proc. of the European Simulations Systems Conf.: Intelligent Process Control and Scheduling Discrete Event Systems*, 1990, pp. 1143-1147.
[67] G. A. Bundell and W. L. Ang, "A Petri net based system modelling tool with TEG subsystem model," in *Proc. 2nd Australasian Conf. on Parallel and Real-Time Systems*, 1995, pp. 147-154.
[68] G. A. Bundell, "Eigen-properties of the information state-space," in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, 1995, pp. 812-817.
[69] M. Aumiaux, "Methodology for the implementation of a petri net or GRAFCET in programmable logical circuits," *Rairo-Automatique-Productique Informatique Industrielle-Automatic Control Production Systems,* vol. 23, pp. 521-542, 1989.
[70] J. B. Morris, G.A. ; Tham, S., "A re-configurable processor for Petri net simulation," *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on,* 2000.
[71] S. Brofferio, "A Petri Net Control Unit for High-speed Modular Signal Processors," *Communications, IEEE Transactions on,* vol. 35, pp. 577-584, June 1987.
[72] J. Allen, "Computer architecture for digital signal processing," *Proceedings of the IEEE* pp. 852 - 873, 1985.
[73] H. T. Kung, "Why systolic architectures?," *Computer,* vol. 15, pp. 37-46, 1982.
[74] S. S. Patil and T. A. Welch, "A Programmable Logic Approach for VLSI," *Computers, IEEE Transactions on,* vol. C-28, pp. 594-601, 1979.
[75] S.-Y. Kung, K. Arun, R. J. Gal-Ezer, and D. Bhaskar Rao, "Wavefront array processor: Language, architecture, and applications," *Computers, IEEE Transactions on,* vol. 100, pp. 1054-1066, 1982.
[76] B. I. Dervisoglu and D. J. Criscione, "A Hard Programmable Control Unit Design Using VLSI Technology," *Computers, IEEE Transactions on,* vol. 100, pp. 800-810, 1981.
[77] D. I. Moldovan, "On the analysis and synthesis of VLSI algorithms," *Computers, IEEE Transactions on,* vol. 100, pp. 1121-1126, 1982.
[78] F. G. Moutinho, L. , "From models to controllers integrating graphical animation in FPGA through automatic code generation," *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on* 2009.
[79] FORDESIGN. (2014). *Formal Methods for Embedded Systems Co-Design*.
[80] W. Reisig, "Petri nets: an Introduction.," *Springer-Verlag Berlin Heidelberg,* 1985.
[81] N. Marranghello, J. Mirkowski, and K. Bilinski, "Synthesis of synchronous digital systems specified by Petri nets," in *Hardware Design and Petri Nets*, ed: Springer, 2000, pp. 129-150.
[82] M. A. J.M. Fernandes, A.J. Proenca, "VHDL generation from hierarchical Petri net specifications of parallel controllers," *lEE Proceedings: Computers and Digital Techniques,* 1997.
[83] M. E. Soto, Pereira, "Implementing a Petri net specification in a FPGA using VHDL," *Proceedings of the International Workshop on Discrete-Event System Design,* 2001.
[84] M. Westergaard and K. B. Lassen, "The BRITNeY suite animation tool," in *Petri Nets and Other Models of Concurrency-ICATPN 2006*, ed: Springer, 2006, pp. 431-440.
[85] L. Gomes and J. Lourenco, "Petri nets-based automatic generation GUI tools for embedded systems," in *Human System Interactions, 2008 Conference on*, 2008, pp. 269-274.
[86] L. G. Joao Lourenco, "Animated Synoptic Generator Framework for Input-Output Place-Transition Petri Net Models," *ICATPN'2008,* 2008.
[87] M. A. Wegrzyn, M, "Hierarchical Approach for Design of Application Specific Logic Controller," *Industrial Electronics, 1999. ISIE '99. Proceedings of the IEEE International Symposium on* vol. 3, 1999.
[88] W. Marek, W. Pawel , A. Marian, and J. L. Monteiro, "Coloured Petri Net Model of Application Specific Logic Controller Programs," in *ISIE´97*, Guimargees, Portugal, 1997, pp. 158-163.

[89] E. M. J. M. S. A. Pérez, *Progrommable Logic Devices and Logic Controllers*, 1996.
[90] M. A. A. Marek Wegrzyn, Joao L. Monteiro, "The application of reconfigurable logic to controller design," *Control Engineering Practice,* 1998.
[91] K. J. L. M. K. LisaWells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *Int J Softw Tools Technol Transfer (2007),* 2007 2007.
[92] K. Jensen and L. M. Kristensen, "Coloured Petri Nets Modelling and Validation of Concurrent Systems," *Springer* 2009.
[93] R. D. H. ALLA, "Petri Nets for Modeling of Dynamic Systems. Survey," *Elsevier Science,* 1994.
[94] L. Ferrarini, "An Incremental Approach to Logic Controller Design with Petri Nets " *IEEE Transactions on Systems, Man, and Cybernetics,* 1992.
[95] J. S. WG19, "WD 19509-2, Software and Systems Engineering, High-level Petri Nets - Part 2: Transfer Format,," 2005.
[96] S. C. J. Billington, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber, "The Petri Net Markup Language: Concepts, Technology, and Tools," *Springer-Verlag Berlin Heidelberg,* vol. 2679, pp. 483–505, 2003.
[97] L. G. Jo˜ao Paulo Barros, "Operational PNML: Towards a PNML Support for Model Construction and Modification," *Workshop on the Definition, Implementation and Application of a Standard Interchange Formatfor Petri Nets; Satellite workshop at the International Conference on Application and Theory of Petri Nets,* 2004.
[98] M. M. James Clark (2001). *RELAX NG Specification*.
[99] W. R. J Desel, "Place/transition Petri nets," *Springer,* 1998.
[100] S. Christensen and N. D. Hansen, *Coloured Petri nets extended with channels for synchronous communication*: Springer, 1994.
[101] M. Heiner, R. Richter, and M. Schwarick, "Snoopy: a tool to design and animate/simulate graph-based formalisms," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, p. 15.
[102] M. Silva, "Las Redes de Petri en la Informática y la Automática," *Editorial AC. Madrid,* 1985.
[103] Informatik. (2012). *Petri Nets Tools Database Quick Overview*. Available: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
[104] R. David and H. Alla, "Petri nets and Grafcet: tools for modelling discrete event systems," 1992.
[105] H. Alla and R. David, "A modelling and analysis tool for discrete events systems: continuous Petri net," *Performance Evaluation,* vol. 33, pp. 175-199, 1998.
[106] W. L. Weizhi Liao "Automatic concurrent Program Generation from Petri nets," *International Symposium on Distributed Computing and Applications to Business, Engineering & Science,* pp. 34 - 39, 2013.
[107] R. A. Nelson, , and L. M. S. Haibt, P.B., "Casting Petri Nets into Programs," *Software Engineering, IEEE Transactions,* vol. 9, pp. 590 - 602, 1983.
[108] G. Bruno and G. Marchetta, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *Software Engineering, IEEE Transactions* pp. 346 - 357, 1986.
[109] D. Taubner, *On the implementation of Petri Nets*: Springer, 1988.
[110] X. H. Weili Yao, "Mapping Petri Nets to Parallel Programs in CC++ " *Computer Software and Applications Conference, COMPSAC '96., Proceedings of 20th International,* pp. 70 - 75, 1996.
[111] X. He, "Translating hierarchical predicate transition nets to CC++ programs," *Information and Software Technology,* pp. 475– 488, 2000.
[112] W. Y. Xudong He, "Translating hierarchical predicate transition nets to CC++ program skeletons," *Computer Software and Applications Conference, 1997. COMPSAC '97.,* pp. 60 - 65, 1997.

[113] H. Reza, S. Endapally, and E. Grant, "A model-based approach for testing gui using hierarchical predicate transition nets," in *Information Technology, 2007. ITNG'07. Fourth International Conference on*, 2007, pp. 366-370.
[114] E. A. Golenkov, A. S. Sokolov, G. V. Tarasov, and D. I. Kharitonov, "Experimental Version of Parallel Programs Translator from Petri Nets to C++," *LNCS,,* pp. 226–231, 2001.
[115] S. Philippi, "Automatic Code Generation From High Level Petri Nets For Model Driven Systems Engineering," *Software[J],* vol. 79, pp. 1444–1455, 2006.
[116] S. Philippi, "Automatic code generation from high-level Petri-Nets for model driven systems engineering," *The Journal of Systems and Software,* vol. 79, pp. 444–1455, 2006.
[117] D. Xu, "A tool for automated test code generation from high-level petri nets," *Proceeding Petri nets 11 Proceedings of the 32nd international conference on Applications and theory of Petri Nets,* pp. 308-317, 2011.
[118] X. U. Weizhi Liao, Xi"an ; Tianlong Gu, "Optimization And Control Of Production Systems Based On Interval Speed Continuous Petri Nets," *Systems, Man and Cybernetics, 2005 IEEE International Conference,* vol. 2, pp. 1212 - 1217, 2005.
[119] F. Balduzzi, A. Giua, and G. Menga, "First-Order Hybrid Petri Nets: A Model for Optimization and Control," *IEEE Transactions on Robotics and Automation,* pp. 382 - 399, 2000.
[120] F. DiCesare, M. R. Gile, and P. T. Kulp, "A Code Generation Tool for Simulation and Control, Based on Colored Petri Nets," *Systems, Man and Cybernetics, IEEE International Conference* vol. 4, 1995.
[121] F. DiCesare, M. R. Gile, and P. T. Kulp, "A code generation tool for simulation and control, based on colored Petri nets," in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, 1995, pp. 3063-3068.
[122] L. E. Holloway, B. H. Krogh, and A. Giua, "A Survey of Petri Net Methods for Controlled Discrete Event Systems," *Discrete Event Dynamic Systems: Theory and Applications,* vol. 7, 1997.
[123] www.informatik.uni-hamburg.de/TGI/PetriNets. (2013). *Complete Overview of Petri Nets Tools Database*.
[124] W. Reisig, *Understanding Petri Nets, Modeling Techniques, Analysis Methods, Case Studies*. Berlin, Germany: Springer-Verlag Berlin Heidelberg 2013, 2013.
[125] PNML. (2006). *http://www.pnml.org/*.
[126] J. D. Herrington. (2007, 02-01). *Code munger & Simple grammar for code generation eclipse plugin Akrogen*.
[127] J. D. W. Reisig and G. R. (Eds.), "Lectures on Concurrency and Petri Nets Advances in Petri Nets," *Springer-Verlag Lecture Notes in Computer Science,* 2005.
[128] S. C. J. Billington , Kees Van Hee , Ekkart Kindler , Laure Petrucci , Reinier Post , Christian Stehno , Michael Weber, "The Petri Net Markup Language Concepts, Technology, and Tools," *Lecture Notes in Computer Science. Springer,* 2003.
[129] I. S. I. I. 15909-2, "Software and Systems Engineering – High-level Petri Nets Part 2," ed, 2005.
[130] S. Roch, "Simultaneity in signal-event systems," in *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA '99. 1999 7th IEEE International Conference on*, 1999, pp. 281-285 vol.1.
[131] K. Venkatesh, Z. MengChu, and R. J. Caudill, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system," *Industrial Electronics, IEEE Transactions on,* vol. 41, pp. 611-619, 1994.
[132] G. F. M. Minas, "Editing, Visualizing, and Implementing Signal Interpreted Petri Nets," *Proceedings of the AWPN,* 2000.
[133] H.-M. Hanisch and A. Luder, "A Signal Extension for Petri Nets and its Use in Controller Design," *Fundamenta Informaticae,* vol. 41, pp. 415-431, 2000.
[134] A. Costa and L. Gomes, "Petri net partitioning using net splitting operation," *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference* pp. 204 - 209, 2009.

[135] F. U. T. SITES. (2014). *Managing embedded systems complexity with SysML - This how-to on SysML describes the design of a fire control system using Model Driven Development*.
[136] A. Aydın and I. Altu˘g "Supervisory Controller Design for Timed Petri Nets," in *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, Los Angeles, CA, USA, 2006, pp. 59-65.
[137] A. Aydın and I. Altu˘g, "Deadlock Avoidance Controller Design for Timed Petri Nets Using Stretching," *IEEE Systems Journal,* vol. VOL. 2, pp. 178-189, June 2008.
[138] H. C. a. H. M. Hanisch, "Control synthesis of timed discrete event systems based on predicate invariance," *EEE Trans. Syst., Man Cybern. B, Cybern.,* 2000.
[139] K. Kobayashi, K. Inoue, and T. Ushio, "LLP supervisory control with timed Petri net models in mobile robots," *Proc. IEEE Int. Conf. Syst., Man, Cybern,* 2001.
[140] X. H. D. Xu and Y. Deng, "Compositional schedulability analysis of real-time systems using time Petri nets," *IEEE Trans. Softw. Eng,* 2002.
[141] C. S. A. Giua and F. Basile, "Petri net control using event observers and timing information," *Proc. IEEE Conf. Dec. Control, Las Vegas,* 2002.
[142] A. Giua, C. Seatzu, and F. Basile, "Observer-based state-feedback control of timed Petri nets with deadlock recovery," *IEEE Trans. Autom. Control Systems, IEEE,* 2004.
[143] H. Apaydın, A. Manay, A. Aybar, and A. ˙Iftar, "A program for analysis and control of Petri nets," *Proc. IEEE Int. Conf. Comput. Cybern., Vienna, Austria,* 2004.
[144] A. Aybar and A. Iftar, "Overlapping decompositions and expansions of Petri nets," *Automatic Control, IEEE Transactions on,* vol. 47, pp. 511-515, 2002.
[145] M. A. M. Adamski, J.L. , "Declarative specification of system independent logic controller programs," *Industrial Electronics, 1996. ISIE '96., Proceedings of the IEEE International Symposium on* vol. 1, 1996.
[146] G. f. G. Corporation. (2015). *Lund University*.
[147] T. Kozlowski, E. L. Dagless, J. M. Saul, M. Adamski, and J. Szajna, "Parallel controller synthesis using Petri nets " *IEE Proc.-Cornput. Digit. Tech.,* 1995.
[148] A. Amroun and M. Bolton, "Synthesis of controllers from Petri Net descriptions and application of ELLA," *Proc. IMEC-IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design,* pp. 57-74, 1989.
[149] A. M. P. João M. Fernandes , Alberto J. Proença , Jo~ao M. Fern, "Concurrent Execution of Petri Nets based on Agents," *Citeseer,* 1995.
[150] R. Piedrafita Moreno and J. L. Villarroel Salcedo, "Adaptive Petri Nets Implementation. The Execution Time Controller," in *Workshop on Discrete Event Systems Göteborg*, Sweden, 2008, pp. 300-307.
[151] J. L. C. Briz, J.M. ; Silva, M., "Simulation of Petri Nets and Linear Enabling Functions," *IEEE internacional conference,* vol. 2, pp. 1671- 1676, 1994.
[152] J. M. Colom, M. Silva, and J. L. Villarroel, "On software implementation of petri nets and colored petri nets using high-level concurrent languages," *Proc of 7th European Workshop on Application and Theory of Petri Nets,* 1986.
[153] R. Valette, M. Courvoisier, J. M. Bigou, and J. Albukerque, "A petri net based programmable logic controller," *Proc. of IFIP Conference on Computer Applications in Production and Engineering,* 1983.
[154] R. P. Moreno and J. L. V. Salcedo, "Petri Nets and Java. Real-Time Control of a flexible manufacturing cell," *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on,* pp. 1246 - 1253, 2006.
[155] R. P. Moreno and J. L. V. Salcedo, "Performance Evaluation of Petri Nets Execution Algorithms," *Systems, Man and Cybernetics, 2007. ISIC. IEEE* pp. 1400 - 1407, 2007.
[156] A. Hellgren, M. Fabian, and B. Lennartson., "On the execution of sequential function charts," *Control Engineering Practice,* 2005.
[157] J. L. Briz, "Técnicas de implementación de Redes de Petri," Ciencia de los ordenadores, Univ. Zaragoza, Zaragoza, 1995.

[158] J. Ezpeleta, J. M. Colom, and J. Martinez, "A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems," *EEE Trans on Robotics and Automation,* vol. 11, pp. 173–184, 1995.
[159] K. Jensen, ": W. Brauer, W. Rdsig, and G. Rozenberg, editors, Petri Nets: Central Models and Their Properties," *Springer-Verlag,* 1987.
[160] E. W. Dijkstra, "Hierarchical ordering of sequential processes," *Acta Informática,* vol. 1, pp. 111-127, 1971.
[161] G. Bollella, B. Brosgol, P. Dibble, and S. Furr, *The Real-time Specification for Java* 2000.
[162] R. P. Moreno and J. L. V. Salcedo, "Implementation of time petri nets in real-time Java," *ACM New York,* 2006.
[163] A. GmbH. (2013). *JamaicaVM, java for sistem embebed*.
[164] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. Thesis Computer Science., Stanford Univ., CA. , 1984.
[165] E. A. Emerson, "The beginning of model checking: A personal perspective," in *25 Years of Model Checking*, ed: Springer, 2008, pp. 27-45.
[166] A. P. B. Schatz, F. Huber, and J. Philipps, "Model-based development of embedded systems," *Springer LNCS,* 2002.
[167] C. Bunse, H.-G. Gross, and C. Peper, "Applying a Model-based Approach for Embedded System Development," *Proceedings of the 33rd Euromicro Conference on Software Engineering and Advanced Applications,* 2007.
[168] D. B. de Niz, G. ; Rajkumar, R., "Model-Based Development of Embedded Systems: The SysWeaver Approach," *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium,* 2006.
[169] D. Harel, "Biting the Silver Bullet: Toward a Brighter Future for System Development," *Computer,* 1992.
[170] F. Pereira, F. Moutinho, and L. Gomes, "Model-checking framework for embedded systems controllers development using IOPT Petri nets," in *Industrial Electronics (ISIE), 2012 IEEE International Symposium on*, 2012, pp. 1399-1404.
[171] J. Nurmi, "Network-on-chip: A new paradigm for system-on-chip design," in *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, 2005, pp. 2-6.
[172] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *Circuits and Systems Magazine, IEEE,* vol. 4, pp. 18-31, 2004.
[173] D. Sigüenza-Tortosa, T. Ahonen, and J. Nurmi, "Issues in the development of a practical NoC: the Proteo concept," *Integration, the VLSI Journal,* vol. 38, pp. 95-105, 2004.
[174] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone-a communication protocol stack for networks on chip," in *VLSI Design, 2004. Proceedings. 17th International Conference on*, 2004, pp. 693-696.
[175] D. Wiklund and D. Liu, "SoCBUS: switched network on chip for hard real time embedded systems," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 8 pp.
[176] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proceedings of the conference on Design, automation and test in Europe*, 2000, pp. 250-256.
[177] H. Kariniemi and J. Nurmi, "Reusable XGFT interconnect IP for network-on-chip implementations," in *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, 2004, pp. 95-102.
[178] X. Wang and J. Nurmi, "An on-chip CDMA communication network," in *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, 2005, pp. 155-160.
[179] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE,* vol. 22, pp. 414-421, 2005.
[180] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*: springer, 2009.

[181] R. Piedrafita and J. L. Villarroel, "Performance Evaluation of Petri nets Centralized Implementation. The Execution Time Controller," *Discrete Event Dynamic Systems,* vol. 21, pp. 139-169, 2011.
[182] J. Villarroel, "Integración informática del control de sistemas flexibles de fabricación," *Ingeniería Eléctrica e Informática Zaragoza: Universidad de Zaragoza,* 1990.
[183] F. García and J. Villarroel, "Modelling and Ada implementation of real-time systems using time Petri Nets," in *Proc. of the 21st IFAC/IFIP workshop on real-time programming. Gramado-RS, Brazil. November*, 1996.
[184] R. Valette and B. Bako, "Software implementation of Petri nets and compilation of rule-based systems," in *Advances in Petri nets 1991*, ed: Springer, 1991, pp. 296-316.
[185] M. S. Hideki, K. N. Tatsumasa, D. O. Yasunori, F. Anzai, N. Kawahara, T. Takei, and T. Watan Abe, "Hardware Architecture for Hierarchical Control of Large Petri Net," *IEEE,* pp. 115-126, 1993.
[186] W. Reisig, "Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets " *Springer,* 1998.
[187] S. Christensen and L. Petrucci, "Modular analysis of Petri nets," *The computer journal,* vol. 43, pp. 224-242, 2000.
[188] G. Bruno, R. Agarwal, A. Castella, and M. P. Pescarmona, "CAB: an environment for developing concurrent application," in *Application and Theory of Petri Nets 1995*, ed: Springer, 1995, pp. 141-160.
[189] L. Gomes, J. Barros, and A. Costa, "Petri Nets Tools and Embedded Systems Design," 2007.
[190] F. G. Moutinho, L. , "Extending a net splitting operation for decomposition of high-level Petri nets," *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society* pp. 6120 - 6125, 2012.
[191] D. de Niz, G. Bhatia, and R. Rajkumar, "Model-Based Development of Embedded Systems: The SysWeaver Approach," in *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, 2006, pp. 231-242.
[192] L. Gomes and J. Lourenco, "Rapid Prototyping of Graphical User Interfaces for Petri-Net-Based Controllers," *Industrial Electronics, IEEE Transactions on,* vol. 57, pp. 1806-1813, 2010.
[193] C. Rust and B. Kleinjohann, "Modeling Intelligent Embedded Real-Time Systems using High-Level Petri Nets," in *Proceedings of the Forum on Design Languages, FDL*, 2001.
[194] L. Gomes and J. P. Barros, "Structuring and composability issues in Petri nets modeling," *Industrial Informatics, IEEE Transactions on,* vol. 1, pp. 112-123, 2005.
[195] A. Girault and C. Ménier, "Automatic production of globally asynchronous locally synchronous systems," in *Embedded Software*, 2002, pp. 266-281.
[196] L. Hillah, E. Kindler, F. Kordon, L. Petrucci, and N. Treves, "A primer on the Petri Net Markup Language and ISO/IEC 15909-2," *Petri Net Newsletter,* vol. 76, pp. 9-28, 2009.
[197] J. Clempner, "A hierarchical decomposition of decision process Petri nets for modeling complex systems," *International Journal of Applied Mathematics and Computer Science,* vol. 20, pp. 349-366, 2010.
[198] A. Karatkevich and G. Andrzejewski, "Hierarchical decomposition of Petri nets for digital microsystems design," in *Proceedings of the International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science-TCSET*, 2006, pp. 518-521.
[199] H. H. Ammar and S. M. R. Islam, "Time scale decomposition of a class of generalized stochastic Petri net models," *Software Engineering, IEEE Transactions on,* vol. 15, pp. 809-820, 1989.
[200] G. Ciardo and K. S. Trivedi, "A decomposition approach for stochastic Petri net models," in *Petri Nets and Performance Models, 1991. PNPM91., Proceedings of the Fourth International Workshop on*, 1991, pp. 74-83.
[201] Y. Li and C. M. Woodside, "Complete decomposition of stochastic Petri nets representing generalized service networks," *Computers, IEEE Transactions on,* vol. 44, pp. 1031-1046, 1995.

[202] G. Berthelot, "Transformations and decompositions of nets," in *Advances in Petri Nets '86*, Germany, 1987, pp. 359–376.
[203] J. Lee, "Decomposition of Petri nets using the transitive matrix," in *Proceedings of the second IEEE International Conference on Systems, Man and Cybernetics (SMC'02)*, 2002.
[204] K. Klai, S. Haddad, and J.-M. Ilié, "An Incremental Verification Technique using Decomposition of Petri Nets," 2002.
[205] D. Zaitsev, "Decomposition of Petri nets," *Cybernetics and Systems Analysis,* vol. 40, pp. 739-746, 2004.
[206] D. A. Zaitsev, "Decomposition-based calculation of Petri net invariants," in *Proceedings of Token based computing Workshop of the 25-th International conference on application and theory of Petri nets, Bologna, Italy*, 2004, pp. 79-83.
[207] L. Jiao, "Decomposition of Nets and Verification in terms of Decomposition," in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, 2005, pp. 804-809.
[208] T. Nishi and R. Maeno, "Decomposition of Petri Nets for Optimization of Routing Problem for AGVs in Semiconductor Fabrication Bays," in *Automation Science and Engineering, 2006. CASE'06. IEEE International Conference on*, 2006, pp. 236-241.
[209] Q. Zeng, "Two symmetrical decomposition methods for structure-complex Petri nets and their applications," in *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, 2007, pp. 1101-1106.
[210] S. Mennicke, O. Oanea, and K. Wolf, "Decomposition into open nets," in *CEUR Workshop Proceedings, AWPN*, 2009, pp. 29-34.
[211] W. Vogler and R. Wollowski, "Decomposition in asynchronous circuit design," in *Concurrency and Hardware Design*, ed: Springer, 2002, pp. 152-190.
[212] W. Vogler, "Improved decomposition of signal transition graphs," *Fundamenta Informaticae,* vol. 78, pp. 161-197, 2007.
[213] B. Kangsah, R. Wollowski, W. Vogler, and J. Beister, "DESI: A tool for decomposing signal transition graphs," in *3rd ACiD-WG Workshop*, 2003.
[214] D. Wist, W. Vogler, and R. Wollowski, "STG Decomposition: Partitioning Heuristics," in *Application of Concurrency to System Design (ACSD), 2011 11th International Conference on*, 2011, pp. 141-150.
[215] P. S. Nascimento, P. R. Maciel, M. Lima, and R. Sant'ana, "A Partial Reconfigurable Architecture for Controllers based on Petri Nets," in *Integrated Circuits and Systems Design, SBCCI 2004. 17th Symposium on*, Inf. Center, Fed. Univ. of Pernambuco, Recife, Brazil 2004, pp. 16 - 21.
[216] J. Tacken, C. Rust, and B. Kleinjohann, "A method for prepartitioning of petri net models for parallel embedded real-time systems," in *In Proc. of the 6th Annual Australasian Conference on Parallel And Real-Time Systems (PART'99*, 1999.
[217] S. Dasgupta and A. Yakovlev, "De synchronisation Technique Using Petri Nets," *Electron. Notes Theor. Comput. Sci.,* vol. 245, pp. 51-67, 2009.
[218] A. K. H. d. Costa, "Petri net model decomposition-a model based approach supporting distributed execution," 2010.
[219] A. Costa, L. Gomes, and J. P. Barros, "Model composition by reusing Petri net based modules " *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society* pp. 6132- 6137, 2012.
[220] A. Costa, L. Gomes, J. P. Barros, J. Oliveira, and T. Reis, "Petri nets tools framework supporting FPGA-based controller implementations," in *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, 2008, pp. 2477-2482.
[221] F. G. Moutinho, L., "Towards distributed execution of Petri net conflicts through model transformation," *Industrial Technology (ICIT), 2013 IEEE International Conference on* 2013.
[222] R. D. a. H. Alla, *Petri Nets & Grafcet; Tools for Modelling Discrete Event Systems*, 1992.

[223] R. C. Campos-Rebelo, A. ; Gomes, L., "Events for human-system interaction modeling with IOPT Petri nets," *Human System Interaction (HSI), 2013 The 6th International Conference on,* 2013.
[224] J. M. Ribeiro, F. ; Pereira, F. ; Barros, J.P. ; Gomes, L., "An Ecore based Petri net Type Definition for PNML IOPT Models," *Industrial Informatics (INDIN), 2011 9th IEEE International Conference,* pp. 777- 782 2011.
[225] A. G. Costa, L. ; Barros, J.P., "System development using Petri net based modules," *Industrial Informatics (INDIN), 2011 9th IEEE International* pp. 768- 774 2011.
[226] F. Pereira, F. Moutinho, L. Gomes, J. Ribeiro, and R. Campos-Rebelo, "An IOPT-net State-Space Generator Tool," *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on* pp. 383- 389 2011.
[227] F. Xianwen , X. Zhicai, and Y. Zhixiang, "A Study about the Mapping of Process-Processor based on Petri Nets," *Anhui University of Science and Technology,* 2006.
[228] H. b. LIP6. (2009, 2014). *PNML.org*.
[229] K. schafft Zukunft, "Petri Nets in Software Engineering," *Arbeitsberichte - Working Papers,* 2004
[230] P. P. Chu, "FPGA Prototyping By Verilog Examples: Xilinx Spartan-3 Version," *Wiley-Interscience,* 2008.
[231] C. Szyperski, D. Gruntz, and S. Murer, *Component Software*: Addison Wesley, 2002.
[232] L. Popova-Zeugmann, "Time and Petri Nets," *Springer,* 2013.
[233] J. E. H. R. M. J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation " *Prentice Hall,* 2006.
[234] Digilent. (2008). *Digilent Nexys Board Reference Manual*.
[235] Sarahtattersall. (2014, 30 de enero). *PIPE 5*.
[236] Xilinx, "LogiCORE IP AXI INTC (v1.02a)," 2012.
[237] S. Pissanetzky, *Sparse Matrix Technology*. Bariloche, Argentina: Centro Atómico Bariloche, 1984.
[238] S. Kestury, J. D. Davisz, and E. S. Chungz, "Towards a Universal FPGA Matrix-Vector Multiplication Architecture," *Field-Programmable Custom Computing Machines (FCCM), IEEE 20th Annual International Symposium on,* 2012.
[239] G. H. Golub and C. F. V. Loan, *Matrix Computations* Johns Hopkins, 2012.
[240] F. Pereira and L. Gomes, "Automatic synthesis of VHDL hardware components from IOPT Petri net models," in *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, 2013, pp. 2214-2219.
[241] O. M. A. M. A. M. E. Maxiniliano, "Driver para Multicore Heterogéneo con Procesador de Petri," *V congreso de microelectronica aplicada,* 2014.
[242] I. P. Nedjeljko Peri, " An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net," *ATKAAF,* 2006.
[243] J. Nonino, I. Furey, C. R. Pisetta, and O. Micolini, "Analysis for the Integration Feasibility of OpenSPARC T1 and a Petri Nets Processor to Form a System with Hardware Synchronization Capability," *IEEE Latin America Transactions,* vol. 11, pp. 60-64, 2013.
[244] J. N. y. C. R. P. Orlando Micolini, "IP Core Para Redes de Petri con Tiempo," *CASIC 2013,* pp. 1097-110, 2013.
[245] N. Wu and M. Zhou, "Process vs resource-oriented Petri net modeling of automated manufacturing systems," *Asian Journal of Control,* vol. 12, p. 267 280, 2010.
[246] E. A. Orlando Micolini, Sergio H. Birocco Baudino, y Marcelo Cebollada, "Reducción de recursos para implementar procesadores de redes de Petri," *en Jaiio 2014,* 2014.
[247] M. A. Lawley, "Deadlock avoidance for production systems with flexible routing," *Robotics and Automation, IEEE Transactions on,* vol. 15, pp. 497-509, 1999.
[248] M. P. Fanti, G. Maione, and B. Turchiano, "Design of supervisors to avoid deadlock in flexible assembly systems," *International Journal of Flexible Manufacturing Systems,* vol. 14, pp. 153-171, 2002.

[249] E. Roszkowska, "Supervisory control for deadlock avoidance in compound processes," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on,* vol. 34, pp. 52-64, 2004.
[250] J. Park and S. A. Reveliotis, "Liveness-enforcing supervision for resource allocation systems with uncontrollable behavior and forbidden states," *Robotics and Automation, IEEE Transactions on,* vol. 18, pp. 234-240, 2002.
[251] LAAS/CNRS. (2013). *TINA webpage*.
[252] O. Micolini, M. F. Caro, I. Furey, and M. Cebollada, "Generación de Código de Sistemas Concurrentes a partir de Redes de Petri Orientadas a Procesos," in *XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Argentine Symposium on Technology (AST)(Buenos Aires, 2014)*, 2014.
[253] J. C. a. P. Chretienne, "Timed Petri net schedules," *pringer Verlag, Berlin,,* vol. 340, pp. 82-84, 1988.
[254] A. C. Ramirez, J. Silva., M., "On optimal scheduling in deds," *IEEE Int. Conf. on Robotics and Automation,* pp. 821-826, 1993.
[255] C. H. a. A. Munier., *Cyclic scheduling problem: An overview. In P. Cheretienne*, 1995.
[256] I. M. J.M. Proth, " Planning and scheduling based on Petri nets," *Petri Nets in Flexible and Agile Automation,* pp. 109-148, 1995.
[257] C. Ramchandani, " Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. ," Massachussets Inst. of Technology, PhD, Massachussets, 1974.
[258] J. Sifakis, "Performance evaluation of systems using nets," *Springer-Verlag Berlin Heidelberg,* vol. 84, pp. 307-319, 1979.
[259] J. E. C. y. N. Roussopoulos, "Timing requirements for time-driven systems using augmented Petri nets," *EEE transactions on Software Engineering,* vol. 9, pp. 603-616, 1983.
[260] H. M. Hanisch, " Analysis of place/transition nets with timed arcs and its application to batch process control," *Springer Verlag, Berlin,,* vol. 691, pp. 282-299, 1993.
[261] K. M. Sacha, "Real-time software specification and validation with transnet. Real-Time," pp. 153-172, 1994.
[262] P. M. Merlin, "A Study of the Recoverability of Computing Systems," PhD Thesis, 75–11026, University Microfilms, , University of California, 1974.
[263] M. K. Kurt Jensen Jonathan Billington, "Transactions on Petri Nets and Other Models of Concurrency III," *Springer-Verlag Berlin Heidelberg* vol. Lecture Notes in Computer Science 5800, 2009.
[264] D. M. Frank Heitmann. (2005). *Petri Nets World*.
[265] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze, "A unified high-level Petri net formalism for time-critical systems," *IEEE transactions on Software Engineering,* p. 160{172, 1991.
[266] M. K. Molloy., "Performance analysis using stochastic Petri nets," *IEEE Transactions on Computers,* vol. 39, pp. 913-917, 1982.
[267] G. B. M. Ajmone Marsan, G. Conte, S. Donatelli, G. Franceschinis, "Modelling with Generalized Stochastic Petri Nets," *Wiley,* 1995.
[268] F. Bause and P. Kritzinger, "Stochastic Petri Nets - An Introduction to the Theory -," *Universit¨At Dortmund D-44221 Dortmund, Germany,* 2002.
[269] P. M. MERLIN and D. J. FARBER, "Recoverability of communication protocols Implications of a theoretical study," *IEEE Transactions on Communications,* pp. vol. 24(9), pp 1036–1043, 1976.
[270] R. Di Giovanni, "Petri nets and software engineering:," in *International Conference on Application and Theory of Petri Nets*, Paris, France, 1990, pp. 123-138.
[271] P. R. Muro, J. A. Banares, and J. L. Villarroel, "Knowlege representation oriented nets for discrete event systems applications," *IEEE Transactions on Systems Man and Cibernetics,* 1998.
[272] F. G. Izquierdo, "Modelado e Implementación de Sistemas de Tiempo Real Mediante Redes de Petri con Tiempo," Universidad de La Rioja Servicio de Publicaciones, La Rioja , España, 2003.

[273] C. E. L. Thomas H. Cormen, Ronald L. Rivest, Clifford Stein, *Introduction to algorithms* Third Edition ed. Massachusetts London, England, 2009.
[274] N. D. Jones, L. H. Landweber, and Y. E. Lien, "Complexity of Some Problems in Petri Nets," *Theoretical Computer Science,* vol. 4, pp. 277–299, 1977.
[275] E. Greenbaum, "Open source semiconductor core licensing," *Harv. JL & Tech.,* vol. 25, p. 131, 2011.
[276] G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, 2000.
[277] M. F. Peter A. Buhr, "Monitor Classification," *ACM Computing Surveys* vol. 27, pp. 63-107 1995.

Anexo A

RdP Temporales (RdPTp)

Introducción

En este apartado analizaremos el comportamiento de los sistemas en los que el tiempo surge como un importante parámetro, el que es continuo o discreto y cuantificable. Los sistemas informáticos a abordar son concurrentes, interactúan con el entorno (reactivos) y dependen del tiempo, como los de control, comunicaciones, monitorización, robótica, electrodomésticos, aeroespacial, comunicaciones digitales, etc.

En estos sistemas el tiempo es un parámetro determinante. Por ejemplo, en los sistemas que implementan protocolos de comunicación, se requiere de mecanismos de reconfiguración después de la pérdida de un mensaje o un cambio de topología de red, esto generalmente se basan en los retardos.

En principio, la especificación temporal, la podríamos realizar en las plazas, las transiciones, los arcos y/o los token; y esta especificación podría ser determinística o estocástica.

Distintos formalismos de RdPTp

Cada abordaje sobre las distintas semánticas de las RdP, se corresponde con distintas problemáticas y técnicas de modelado.

Si consideramos en asociar una duración al disparo de las transiciones, no es posible modelado de actividades interrumpibles, lo que es muy restrictivo para sistemas de tiempo real. No obstante, en el modelado, análisis y planeación de sistemas de fabricación y logística esto simplifica la tarea y son generalmente suficientes, podemos encontrar abundantes datos al respecto en [253] [254] [255] [256].

El formalismo de Ramachandani [257] de RdP temporizadas, asocia una duración constante a las transiciones, también lo hace [29].

Hay semánticas que asocian el intervalo de tiempo a las plazas como [258] [259]. Mientras que otras semánticas introducen: intervalo de duración o token con una marca de tiempo como en las RdP coloreadas [92].

Hay dos modelos de RdP, donde se asocia un tiempo a los arcos que unen los lugares con las transiciones, donde los arcos se abren o cierran durante un instante específico, estos conceptos son introducidos por [260] y [261].

RdPTp

Definición 8: Notaciones

Sea Σ un conjunto finito (o alfabeto). Σ^* especifica un conjunto finito de palabras sobre Σ. Si $w = a_1 \dots a_n$. es una secuencia de largo w , denotado por $|w| = n$. Además usamos $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$, donde $\epsilon \notin \Sigma$, siendo ϵ la palabra vacía. B^A representa una aplicación de A a B. Si A es finito y |A|=n, un elemento de B^A es un vector en B^n.

Los operadores que usamos sobre el vector A^n son $+, -, <\ y\ =$.

Números: donde $\mathbb{N}$ el conjunto de números naturales, $\mathbb{Q}$ el conjunto de números racionales, $\mathbb{R}$ el conjunto de números, $\mathbb{B}$ el conjunto son los booleanos $(0, 1)$ y $\mathbb{R}_{\geq 0}$ denota el conjunto de los reales no negativos.

Un valor v sobre un conjunto de variable X de es un elemento $\mathbb{R}_{\geq 0}^{X}$. Para $v \in \mathbb{R}_{\geq 0}^{X}$ y $d \in \mathbb{R}_{\geq 0}$, $v + d$ representa el valor definido por $(v + d)(x) = v(x) + d$ o el valor $\forall\ x \in X, v(x) = o$.

Un intervalo I es un $\mathbb{Q}_{\geq 0} - intervalo$ de $\mathbb{R}_{\geq 0}$ sii su lado izquierdo pertenece a $\mathbb{Q}_{\geq 0}$ y su lado derecho pertenece a $\mathbb{Q}_{\geq 0} \cup \{\infty\}$.

Denotamos a$I^{\downarrow} = (x| \ni y\, I \wedge\ y\ \geq x)$, es la clausura descendente de I.

Denotamos $\mathcal{I}(\mathbb{Q}_{\geq 0})$ al conjunto $\mathbb{Q}_{\geq 0} - intervalo$ de $\mathbb{R}_{\geq 0}$

Transición temporizada de Sistemas (TTS)

La descripción de la evolución de los sistemas temporizados puede ser discretas, continuas o una combinación de ambas.

Definición 9: Sistemas de transición temporizada (TTS)

Un sistema de transición temporizados (TTS) sobre el conjunto de acciones Σ_ϵ es una tupla $S = (Q, q_0, \Sigma_\epsilon, \longrightarrow)$ dónde:

- Q es un conjunto de estados,
- q_0 es el estado inicial,
- Σ_ϵ es un conjunto finito de acciones disjuntas de $R_{\geq 0}$,
- $\rightarrow \subseteq Qx(\Sigma_\epsilon \cup R_{\geq 0})xQ$ es un conjunto de arcos. Si $(q, e, q') \in \rightarrow$, escribimos $q \xrightarrow{e} q'$, para una transición $q \xrightarrow{d} q'$ con $d \in R_{\geq 0}$, el valor d representa un intervalo de tiempo.

Haremos las siguientes presunciones sobre las TTS:

- Tiempo determinístico: si $q \xrightarrow{d} q'$ y $q \xrightarrow{d} q''$ con $d \in R_{\geq 0}$, luego $q' = q''$,
- $0 - retardo$: $q \xrightarrow{0} q'$
- Retardo aditivo: si $q \xrightarrow{d} q'$ y $q' \xrightarrow{d'} q''$ con $d, d' \in R_{\geq 0}$, luego $q' \xrightarrow{d+d'} q''$
- Continuidad: si $q \xrightarrow{d} q'$, luego para cualquier $d', d'' \in R_{\geq 0}$ tal que $d = d' + d''$, existe un q'' tal que $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$

A partir de estas preposiciones, podemos decir que una corrida ρ de largo $n \geq 0$, puede ser escrita como una secuencia de transiciones finitas:

$$\rho = q_0 \xrightarrow{d_0} q_0' \xrightarrow{a_0} q_1 \xrightarrow{d_1} q_1' \xrightarrow{a_1} q_2 \xrightarrow{d_2} \ldots q_n \xrightarrow{d_n} q_n'$$

Cuando acciones discretas se alternan con intervalos de tiempos (estos pueden ser nulos), es posible escribir:

$$\rho = q \xrightarrow{d_0 a_0 \ldots d_n} q'$$

La secuencia atemporal ρ es la palabra de Σ^* obtenida de concatenar las etiquetas:

$$a_0 a_1 \ldots a_{n-1}.$$

Mientras que el retardo es:

$$retardo(\rho) = \sum_{i=1}^{n} d_i.$$

Definición 10: Bisimiliridade temporizacion débil (w.r.t.)

Sea dos TTS $S_1 = (Q_1, q_0^1, \Sigma_\epsilon, \longrightarrow_1)$ y $S_2 = (Q_2, q_0^2, \Sigma_\epsilon, \longrightarrow_2)$ y $\approx$ una relacion binaria sobre $Q_1 x Q_2$. Ahora podemos escribir $q \approx q'$ para $(q, q') \in \approx$.

$\approx$ es una bisimiliridad temporizada débil entre S_1y S_2 si:

$q_0^1 \approx q_0^2$
$Si\ q_1 \overset{a}{\rightarrow}_1 q'_1$ donde $a \in \Sigma \cup \mathbb{R}_{\geq 0}\ y\ q_1 \approx q_2\ luego\ \ni\ q_2 \overset{a}{\rightarrow}_2 q'_2$ tal que $q'_1 \approx q'_2$, por el contrario si $q_2 \overset{a}{\rightarrow}_2 q'_2$ con $a \in \Sigma \cup \mathbb{R}_{\geq 0}\ y\ q_1 \approx q_2$ luego $\ni\ q_1 \overset{a}{\rightarrow}_1 q'_1$ tal que $q'_1 \approx q'_2$.

Extensión de tiempo en las RdP

Las RdP con tiempo se introdujeron en [262] para ampliar las RdP, esto se realizó introduciendo limitaciones de tiempo sobre los disparos de las transiciones. Un intervalo de tiempo está asociado con cada transición; es decir, un cronómetro implícito es asociado con cada transición activa, y mide el tiempo transcurrido desde la última vez que se sensibilizó.

Una transición habilitada puede ser disparada si el valor cronometrado asociado a la transición tiene un valor en el intervalo de la transición (el valor del intervalo ha sido predeterminado y asociado a la transición). Las siguientes definiciones formalizan estos principios.

Definición 11: RdPTp

Una RdPTp (etiquetada) con tiempo es una tupla $N = (P, T, \Sigma_\epsilon, \bullet(.), (.)\bullet, M_0, \Lambda, I)$, donde :

- P es un conjunto de plazas finito
- T es un conjunto de transiciones finito, donde $T \cap O = 0$;
- $\bullet(.) \in (\mathbb{N}^P)^T$ es el mapa de incidencia de entrada de las transiciones
  $(.)\bullet \in (\mathbb{N}^P)^T$ es el mapa de incidencia de salida de las transiciones
  - $\bullet t$ es el conjunto de plazas de salida de las transiciones
    $$\bullet t = (p \in P \mid \bullet t(p) > 0)$$
  - $t \bullet$ es el conjunto de plazas de entrada de las transiciones
    $$t \bullet = p \in P / t \bullet (p) > 0$$
- $M_0 \in \mathbb{N}^P$ es la marca inicial
- Λ: T $\rightarrow \Sigma_\epsilon$ es la función de etiquetado

- $I: T \rightarrow \mathcal{I}(\mathbb{Q}_{\geq 0})$ el intervalo de disparo asociada con cada transición.

La red de la Figura 195 ilustra la representación gráfica de un RdP con tiempo. Cada transición tiene etiqueta e intervalo de tiempo. Por ejemplo la transición t1 se ha etiquetado con L1 y su intervalo de tiempo es [1, ∞ [.

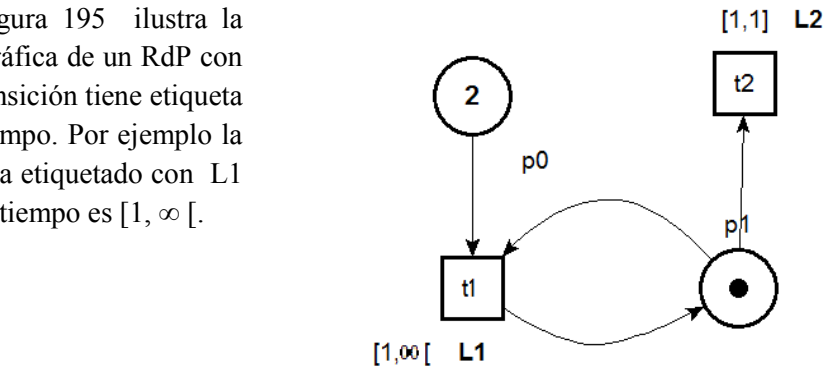


Figura 195: Representación gráfica de RdPcT

Semántica de las RdPTp

La semántica de la RdPcT se dá en términos de TTS. Un marcado M de un RdP con tiempo es una asignación en $\mathbb{N}^P$ y $M(p)$ es el número de token en lugar p. Una transición t está sensibilizada en un marcado M si y sólo (sii) si $M \geq \bullet t$. Denotamos por $En(M)$ el conjunto de transiciones habilitadas en M. Para decidir si una transición t puede ser disparada, es necesario conocer por cuánto tiempo la transición ha estado habilitada: si esta cantidad de tiempo se encuentra dentro del intervalo I (t) la transición t pueden ser disparado, de lo contrario no es posible. La medida del tiempo en la transición sólo puede avanzar si la transición permanece sensibilizada y no es inhibida por un brazo.

Sea $v \in (\mathbb{R}_{\geq 0})^{En(M)}$ la medida de tiempo, tal que su valuación, el valor $v(t)$ es el tiempo transcurrido desde que la transición t fue sensibilizada.

El estado de la RdP con tiempo (red N) es un par $(M, v(t))$. Un estado válido de un RdP con tiempo es $(M, v(t))$ donde se cumple $\forall\, t \in En(M), v(t) \in I(t)^{\downarrow}$. Decimos que ADM (N) es el conjunto admisibles de marcas.

En la definición de la semántica de un RdPTp, tres tipos de políticas se fijan, estas son:

Definición 12: Política de selección

Se refiere a la elección del próximo evento para ser disparado (programado). Para una RdPTp tiempo (y también los autómatas temporales), esta elección es no determinista. Para que la selección sea determinística hay posibles alternativas, como: usar prioridades, probabilidades, etc.

Definición 13: Política de servicio

Se refiere a la posibilidad de instancias simultáneas varias veces un mismo evento. En el contexto de las RdP, esto se formaliza por el grado de sensibilidad de una transición. Aquí, para el PP, adoptamos la política de un solo servidor (solo una instancia de un disparo por transición, para

todos los estados).

Definición 14: Política de memoria

Se refiere a la actualización de los temporizadores cuando se produce un paso discreto. La cuestión clave en la semántica es definir cuándo reinicia el cronómetro que mide el tiempo transcurrido desde la última sensibilización de la transición. Esto sólo puede ocurrir cuando se dispara la transición. Decimos que $\uparrow habilita\,(t',M,t) \in \mathbb{B}$ es verdadero si t' está recién sensibilizado por el disparo de la transición t en la marca M , y falso en caso contrario.

Definición 15: Ecuación de estado de una RdPcT

Sea una marca M y $t \in En(M)$. El disparo de t genera una nueva marca definida por la ecuación de estado: $M' = M - (\bullet t + t \bullet)$.

Ahora tres semánticas son posibles, las que son:

Definición 16: I es la semántica intermedia

I: es la semántica intermedia y consiste de dos pasos, los que son:

1. Consumir los token de entrada en $\bullet t$
2. Producir los token de salida en $t \bullet$

El hecho de que una transición t' está recién habilitada en el disparo de una transición $t \neq t'$ es definido como el marcado intermedio $M - \bullet t$. Cuando se dispara la transición t está es habilitada nuevamente cualquiera que sea la marca intermedia. Denotamos por $\uparrow habilita\,(t',M,t)$ el predicado recién habilitada en este caso. Esta asignación se define por:

$$\uparrow habilita_I\,(t',M,t) = (t' \in En(M -\bullet t + t \bullet)) \wedge (t' \notin En(M -\bullet t) \vee (t = t'))$$

Definición 17: A es la semántica atómica

A: es la semántica atómica que consiste en que el disparo de una transición se logra por un paso atómico. El mapa que corresponde es:

$$\uparrow habilita_A\,(t',M,t) = (t' \in En(M -\bullet t + t \bullet)) \wedge (t' \notin En(M) \vee (t = t'))$$

Definición 18: PA es la semántica atómica persistente

PA: es la semántica atómica persistente que consiste en que el disparo de una transición se logra también por un paso atómico, la diferencia con A es que el valor de la sensibilización, la que ahora es:

$$\uparrow habilita_{PA}\,(t',M,t) = t' \in En(M -\bullet t + t \bullet) \wedge (t' \notin En(M))$$

Hay que tener en cuenta que la relación:

$$(\uparrow habilita_{PA}) \Rightarrow (\uparrow habilita_{A}) \Rightarrow (\uparrow habilita_{I})$$

Pero, como veremos esto no implica ninguna relación de inclusión entre los diferentes comportamientos. Definimos ahora la semántica de un RdPcT, que se parametriza por la elección del predicado de $\uparrow habilita$.

Definición 19: Semántica de una RdPTp.

Sea $s \in (I, A, PA)$. La $s - semántica$ de una RdPcT $N = (P, T, \Sigma_\epsilon, \bullet(.), (.)\bullet, M_0, \Lambda, I)$ es un sistema de transición de tiempo TTS:

$S_N = (Q, q_0, T, \rightarrow)$, donde

$Q = ADM(N)$

$q_0 = (M_0, 0)$, y

$\rightarrow \in Qx(\Sigma_\epsilon \cup \mathbb{R}_{\geq 0})xQ$, Consiste en la relación, según sean las transiciones discretas o continuas:

- $\forall (M, v) \in ADM(N), \forall t \in En(M)\ s.t.\ v(t) \in I(t)$,la relación discreta para la transición es definida por:

$$(M, v) \xrightarrow{A(t)} (M - (\bullet t + t \bullet), v')\ donde\ \forall t \in En(M - (\bullet t + t \bullet))$$
$$v'(t) = \begin{cases} 0\ si\ enable_s(t', M, t) \\ v(t)\ en\ otro\ caso \end{cases}$$

- $\forall (M, v) \in ADM(N), \forall t \in \mathbb{R}_{\geq 0}, s.t. \forall t \in En(M), v(t) + d \in I(t)^{\downarrow}$, la relación continua para la transición es definida por:

$$(M, v) \xrightarrow{d} (M, v + d)$$

Ahora podemos escribir $(M, v) \xrightarrow{w}$ para enfatizar que la secuencia de transiciones w puede disparar S_N desde (M, v).

Si la duración del disparo w es cero, entonces la transición es inmediata, de $duración(w) = 0$, esto puede ser extendido a una secuencia de disparos.

El conjunto de marcas de N es alcanzables si:

$$alcanzable(N) = (M \in \mathbb{N}^P | \in (M, v)| \in (M_0, 0) \xrightarrow{w} (M, v))$$

Una red N es acotada sii

$$\ni un\ entero\ B\ tal\ que\ \forall M \in alcanzable(N), \forall p \in P, M(p) \leq B$$

El ejemplo de la Figura 195 ilustra las diferencias entre semánticas. La secuencia $(M0, 0) \xrightarrow{1t_1t_1}$ es solo para la semántica PA, pero no para los demás ya que el segundo disparo de t_1 debe implicar un retraso de al menos 1 unidad de tiempo. La secuencia $(M0, 0) \xrightarrow{1t_1t_2}$ es una secuencia que corresponde a las semánticas PA y A, pero no para la semántica I, $t_1$consume (y produce de

nuevo) el token en lugar p_2 esto debe implicar un retraso de al menos una unidad de tiempo antes de que t_2 pueda dispararse. La secuencia $(M0, 0) \xrightarrow{1 t_1 1 t_2}$ es válida para la semántica de I, pero no para los demás ya que después de los disparos t_1, t_2 no está activado y el tiempo no puede transcurrir.

La semántica intermedios I es la más común.

Sin embargo conocemos dos patrones significativos, en los sistemas de eventos discretos, donde las otras semánticas son más apropiadas.

Considere la posibilidad de la red de la Figura 196, donde los modelos de un componente cuyo estado está marcada por un observador a fin de reaccionar (por el disparo de la transición t).

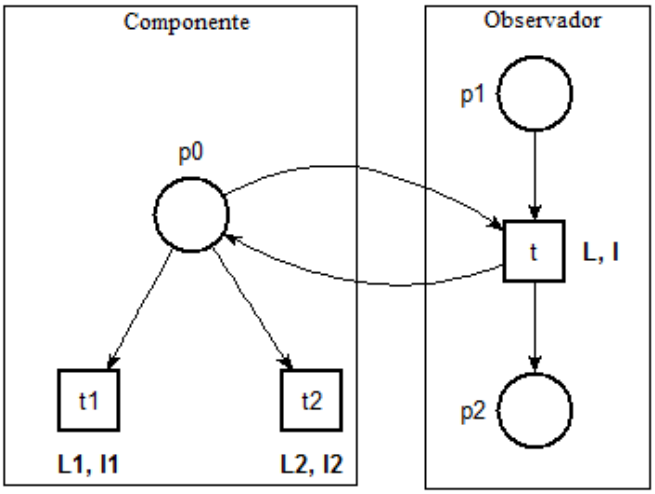


Figura 196: Modelo de un componente cuyo estado está marcado por un observador a fin de reaccionar

El observado no interfiere con el comportamiento de los otros componente si usamos la semántica A y PA mientras que la semántica I reinicializa el contador de tiempo de las transiciones t_1, t_2.

La red de la Figura 197 modela un cliente en espera de información, que es producida por un productor (en una unidad de tiempo) la que luego se envía a cada cliente (en forma instantánea). Los dos disparos de t se llevan a cabo al mismo tiempo sólo con la semántica PA.

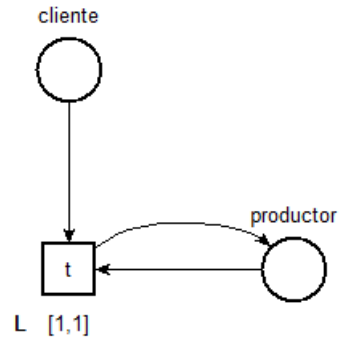


Figura 197: Modelo un cliente en espera de información

Inclusión de las semánticas

Ahora establecemos relaciones de inclusión entre los tres semántica para RdPcT.

Con el fin de aliviar las figuras, las transiciones se llenan en negro cuando su intervalo de disparo es $[0, 0]$ y omitimos su etiqueta cuando es igual a ϵ.

Definición 20: Relación entre la semántica A y la I

Sea N una RdP con tiempo, con semántica intermedia I. Existe una TPN $\overline{N}$ con semántica atómica A que está débilmente cronometradas bisimilar a N. El tamaño de $\overline{N}$ es lineal w.r.t. el tamaño de N. Además, si N es acotada entonces $\overline{N}$ es acotada.

Sea N una RdP con tiempo con semántica atómica A. Existe una RdP $\overline{N}$ con persistencia atómica PA que está débilmente cronometrados bisimilar a N. El tamaño de $\overline{N}$ es w.r.t. lineal el tamaño de N. Además, si $\overline{N}$ es acotada entonces N es acotada.

Inclusión estricta

La siguiente proposición muestra que el poder expresivo de las RdPcT depende de la semántica elegida, incluso en el caso acotado.

Sea N una RdPTp con tiempo limitada, semántica atómicas persistentes PA, tal que no hay una RdP con tiempo (incluso sin límites) con semántica atómicas bisimilar a N.

Resultado de equivalencia de las RdPTp acotadas con intervalos superiores cerrados

Debido a la inclusión establecida en la definición anterior, ahora limitamos el estudio a la RdPTp acotadas, con intervalos superiores cerradas, es decir, con intervalos
$[a, b], [a, \infty[,]a, b]\ o]\ a, \infty[$.

Definición 21: Equivalencia de las RPT acotadas con intervalos superiores cerrados

Sea N un RdPTp acotado con intervalos superiores cerradas y con semántica atómica persistente PA. Existe una RdPTp $\overline{N}$ limitada con la semántica intermedia I, que es débilmente cronometrada bisimilar a N. El tamaño de $\overline{N}$ es w.r.t. lineal con el tamaño de N y el logaritmo como cota.

Introducción del tiempo en RdP

Podemos encontrar distintas formas de introducir el tiempo en las RdP, algunas de estas son:

- En las transiciones o token: asociando una duración constante a las transiciones [257].
  Utilizar redes de alto nivel enriquecidas con tiempo [29].
  Los token tienen asociado una estampa de tiempo (timestamp) y otros valores [263].
  En los arcos, se asocia el tiempo a los arcos que unen las plazas con las transiciones [260] [261].
- En las plazas, se asocia la duración a cada lugar, deshabilitando la capacidad de la marca para sensibilizar por un tiempo [258] [259].
- Otras son asociar funciones a las transiciones y condiciones a los arcos [261].

RdP con tiempo de alto nivel

Las redes Entorno/Relación (Environment / Relationship nets) o (ER), son RdP de alto nivel, donde los tokens son relacionados con el entorno, es decir, funciones que asocian valores a las variables [264].

Los token son relacionados con el entorno por medio de un ID y un conjunto de valores V , esto lo que realiza la función: $ID \rightarrow V$, siendo $ENV = V^{ID}$ el conjunto de todo el entorno.

Las *Time ER* o *TER* [265] constituyen un formalismo más general que la RdP con tiempo, siendo estas últimas un caso particular de las TER.

En forma sintética, las TER son similares a las *Color Petri Net* CPN, donde los token son una tupla, estos token tienen una componente llamada *timestamp* en la variable *chronos*. La regla de disparo es similar a las CPN, siendo el disparo condicionado por el valor de *chronos* de las marcas de las plazas de entrada, y produce los token de salida con la variable *chronos* de salida, según la expresión asociada con la transición. El valor de la variable *chronos* de los token de salida de una transición para un mismo disparo es lo mismo, y no toman valor inferior al mayor de los valores del *chronos* de las marcas de entrada.

De esta manera, las transiciones pueden ser disparadas en distintos modos dependientes del color de las marcas que consumen. Si el número de colores posibles es finito, las CPN tienen el mismo poder expresivo que las redes P/T, y mediante un algoritmo de desplegado (unfolding) es posible obtener una red P/T equivalente a un CPN determinada.

Elección de un modelo

Hay básicamente tres mecanismos que han sido desarrollados como extensión de las RdP ordinarias, los cuales reducen el indeterminismo con respecto al instante de disparo, que son:

- RdP con Tiempo (RdPcT) [262] [32], donde se asocian dos números enteros a cada transición, que representan, respectivamente el primer instante en el que la transición puede ser disparada, y el último instante en el que puede ser disparada (ambos con respecto al instante en el que la transición ha sido sensibilizada) por lo que estas transiciones pueden ser interrumpibles. En este tipo de redes el disparo no toma ningún tiempo y es atómico. La definición de tiempo, es más general puesto que hay dos valores asociados con cada transición: $t\_min \ y \ t\_max$ y estos valores pueden existir en los siguientes intervalos: $0 \geq t\_min \ \geq \ t\_max \geq \infty$.
  Supongamos que una transición es sensibilizada (regla de disparo de las RdP ordinarias) en el instante de tiempo $t = \theta$, esta expresión nos indica que la transición no puede ser disparada antes del instante $(\theta + t_{\min})$ y tampoco después de $(\theta + t\_\max)$, mientras que las reglas de disparo de la RdP se cumpla.
- RdP Temporizadas (RdPTm) [257], se basa en la asociación de una duración, que es el tiempo que toma el disparo, es in-interrumpible (el modelo puede aceptar valores de intervalo temporal racionales o reales). El disparo puede ser considerado como tres momentos atómicos, los que son:
  - Cuando la transición se sensibiliza, se quitan los token de las plazas que entran a la transición.
  - La segunda instancia corresponde con el intervalo de tiempo que la transición permanece inactiva, no se permite que la transición se sensibilice nuevamente, aún cumpliendo con las condiciones de sensibilización.

    - Finalmente el instante atómico donde la transición coloca los token en las plazas de salida y puede ser sensibilizada nuevamente si cumple con las condiciones de sensibilización.
    - Estas redes, y modelos similares, se utilizan principalmente para análisis de sistema de producción y rendimiento.

- RdP Estocásticas (RdPE) [266] [267] [268], introducen un parámetro que es una estimación estocástica del instante de disparo, no serán implementadas en esta tasis puesto que nuestro objetivo está en un procesador para sistemas de tiempo real, pero consideraremos la extensión de nuestro procesador a este caso.

Usando estas redes se pueden abordar problemas de sistemas reconfigurables, protocolos de comunicaciones [262] , [269] como así también los anteriormente mencionados.

Capacidad de modelado del Formalismo Seleccionado

Puesto que nuestro objetivo principal es la ejecución de las RdP con una semántica temporal, que modela al sistema de tiempo real, nos ocuparemos de la seleccionar del tipo de red para estos modelos, según las restricciones planteadas.

En los modelos de sistemas de tiempo real se debe representar de manera precisa aspectos como:

- Eventos a los que el sistema responde:
    - internos
    - externos
- Patrones temporales de estos eventos:
    - Periódicos
    - Aperiódicos
- Acciones que desencadenan estos eventos:
    - Tiempos para general la acción
    - Tiempo de respuesta
    - Tiempo de cómputo
- Interrelación entre las acciones:
    - Sincronización
    - Comunicación
    - Relación de precedencia

La RdP con semántica temporal, que hemos presentado (RdPcT y RdPTm), nos proporciona las primitivas necesarias para su modelado[32]:

No es objetivo de esta tesis la aplicación de metodológicas de diseño en el modelado de sistemas de tiempo real, por lo que los modelos serán construidos directamente en términos de RdP. No obstante, hay trabajos que integran los formalismos de las RdP a metodologías de diseño de sistemas de tiempo real como [270] [271].

Tipos de Transiciones. Unidades de ejecución

Según se ha expresado en los párrafos anteriores, para el modelado de sistemas de tiempo real mediante RdPTp usaremos las mismas técnicas que para RdP, excepto algunas medicaciones relatadas con las transiciones que incluyen al tiempo.

En el modelo de redes de RdPTp [32] todas las transiciones son del mismo tipo, con la misma funcionalidad. Pero, en un sistema de tiempo real hay distintas situaciones que debemos modelarlas mediante una transición. Para mostrar en nuestros modelos los diferentes roles que representa una transición, distinguimos entre tres tipos de transiciones [272] que, junto con sus lugares de entrada, conformarán los elementos de modelado o unidades de ejecución básicos para construir los modelos de nuestro interés, y estas son:

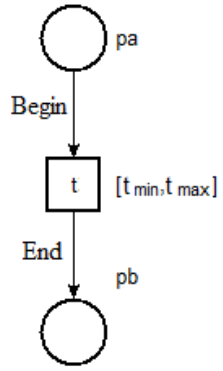
- **Transiciones de tiempo como acción (TA)**. Estas transiciones y sus lugares de entrada representan una actividad ejecutada por el sistema (código en ejecución, movimiento, envió de un mensaje, etc.). Esta acción comenzará a realizarse en el momento en que la transición es sensibilizada, es decir, todos sus lugares de entrada estén marcados. Este tipo de transiciones modela la parte operativa del sistema y están etiquetadas con dos valores temporales $[t_{min}, t_{max}]$, según hemos definido para las RdPT.
  En el modelo de la Figura 198 (a), el significado de los valores está asociado al tiempo de la acción. En el mejor caso esta demora el tiempo mínimo y en el peor el tiempo máximo. La finalización de la acción está representada por el disparo de la transición. Por tanto, la ejecución de la actividad asociada ocupa un tiempo entre $[t_{min}, t_{max}]$ y puede ser interrumpible.

- **Transiciones de actividad periódica (AP)**. Son transiciones asociadas a actividades como un timeout o la activación periódica de un proceso. Estas transiciones modelan la parte de control y de supervisión temporal del sistema. Al igual que las transiciones TA, están etiquetadas como un intervalo temporal $[t_{min}, t_{max}]$. Pero ahora representan el instante a partir de cuando se ha sensibilizado la transición y se se producirá el disparo.
  El disparo del modelo de la Figura 198 (b) de este tipo de transiciones representa la ocurrencia del evento temporal y por tanto provocará que se desarrollen acciones de control en el sistema.
- **Transiciones de sincronización (AS)**. Son el resto de transiciones, y no tienen significado temporal explícito asociado. Estas transiciones al igual que Time modelan la supervisión y el control del sistema.
  El disparo, del modelo de la Figura 198 (c) será inmediato, lo que supone un intervalo temporal implícito $[0, 0]$. Son usadas para modelar sincronizaciones y tareas de control sin requisitos temporales. El disparo de este tipo de transiciones conduce a solo a cambios de estado del sistema y/o es utilizado para sincronizar actividades, modelar eventos no temporales, excepciones, etc. Estas transiciones pueden tener asociados eventos externos que condicionen su disparo.

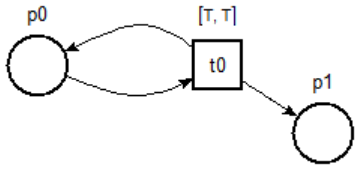
A cualquiera de las transiciones definidas, le podemos asignar una prioridad, y su objeto es:

- AS y AP, deshacer conflictos entre varias transiciones sensibilizadas (prioridad de control) .

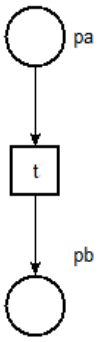
- TA, la planificación del sistema y se utiliza en la ejecución del mismo (prioridad de acción).



(a) RdPcT, valores de tiempo asociados a la transición



(b) El disparo de esta Transiciones representa la ocurrencia del evento temporal



(c) RdP con disparo inmediato

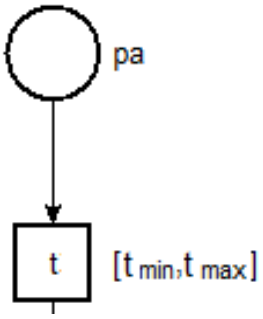
Figura 198: RdP con tiempo

Modelado de situaciones habituales

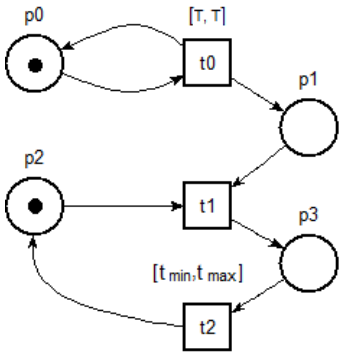
Con el fin de mostrar la capacidad de modelado de las RdPcT se han seleccionado un conjunto de situaciones habituales en sistemas de tiempo real [272], que muestran escenarios frecuentes.

En la Figura 199 (a) se muestra una RdPT que representa un código secuencial, con tiempo de ejecución mínimo t_{min}, y tiempo de ejecución máximo t_{max} .

La Figura 199 (b) muestra una RdPT con activación periódica, el sistema se activa después de cada intervalo de tiempo T. La ejecución del código realiza en una sección crítica p_3. Con tiempo de ejecución mínimo t_{min}, y tiempo de ejecución máximo t_{max}.



(a) RdPcT con tiempo mínimo y máximo



(b) RdPT con activación periódica

Figura 199: Situaciones habituales en sistemas de tiempo real

La Figura 200 muestra una RdPcT con activación aperiódica, el sistema se activa después de un tiempo s y antes de ∞. . La ejecución del código realiza en una sección crítica p_3. Con tiempo de ejecución mínimo t_{min}, y tiempo de ejecución máximo t_{max}.

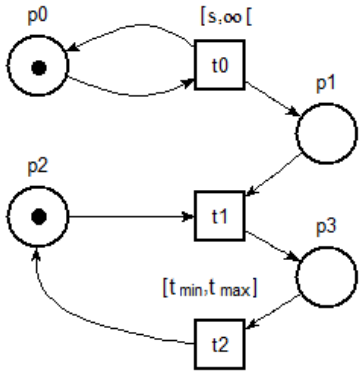


Figura 200: TdPcT con activación aperiódica

La Figura 201 muestra una RdPT con tome-out, las transiciones t_o y t_e se activan simultáneamente. El disparo de t_e en el intervalo de sensibilización, representa la terminación normal del código. Si el código no termina en este intervalo, en el instante T (que es posterior a t_{max}), se genera una señal de time-out por el disparo de t_o .

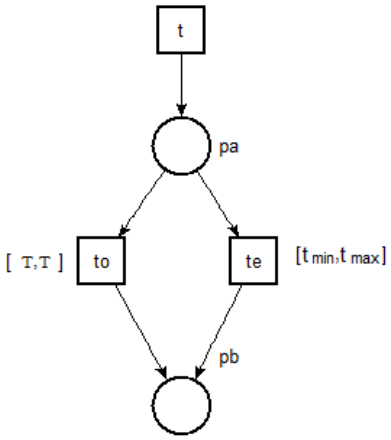


Figura 201: RdPT con time-out

La Figura 202 muestra una RdPT con transferencia asincrónica de control, al activarse la transición t_e , se disparara en el intervalo t_{max}, t_{maz} de sensibilización, lo que representa la terminación normal del código. Pero si t_7 es sensibilizado antes del disparo de t_e, se disparará t_7, lo que representa una interrupción del sistema.

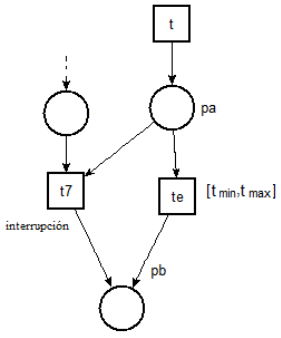


Figura 202: RdPT con transferencia asincrónica de control

La Figura 203 muestra una RdPT con time-out en comunicaciones, al activarse la transición t_e , se disparará en el instante T, lo que representa el tiempo de espera para el arribo de una comunicación (time-out), el arribo del mensaje es representado por el disparo de t_7.

Lo que debe suceder antes de T para una correcta terminación.

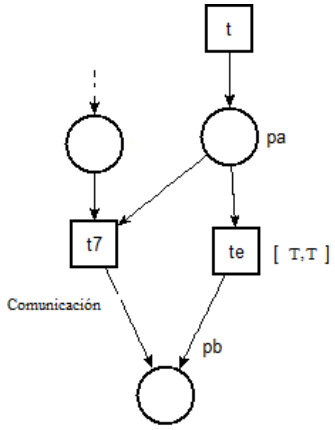


Figura 203: RdPT con time-out en comunicaciones

RdP para el modelado de sistemas de tiempo real

Puesto que nuestro objetivo es el diseño y construcción de un PP que interprete distintos formalismos (RdPcT y RdPTm), para implementar aspectos de control y temporales de sistemas de tiempo real. Nos ocuparemos brevemente de aspectos relacionados con los modelos de construidos directamente en términos de RdP. Para un estudio detallado del diseño con estos modelos podemos ver [270] o [271].

RdP que incluyen el tiempo

Definición 22: Time RdP

RdP con tiempo (RdPcT), es definida por la tupla $\{P, T, Pre, Post, M_0, IS\}$ donde $\{P, T, Pre, Post, M_0\}$ es una RdP y $IS: T \to Q^+ x\ (Q^+ \cup \{\infty\})$ es la función estática del intervalo de tiempo.

La función IS relaciona cualquier transición t de una red con un valor limitado en el intervalo $0 \geq t_{min} \geq t_{\max} \geq \infty$

donde $t_{min}\ es\ denotado\ como\ date\ of\ earlier\ firing\ , SMin(t)$

y

$t_{max}\ es\ denoptado\ como\ static\ date\ of\ later\ firin, SMax(t)$

En la Figura 204 podemos ver un ejemplo de cómo se especifica una RdPcT

Si la regla de disparo se cumple en una transición t, éste sólo se podrá realizar en un intervalo de tiempo relacionado con la transición, recuerde que este intervalo comienza en el momento en que se habilitó la transición.

La Figura 204 nos muestra la red marcada y los intervalos de tiempo, las transiciones pueden ser disparadas al estar sensibilizadas dentro del intervalo de tiempo.

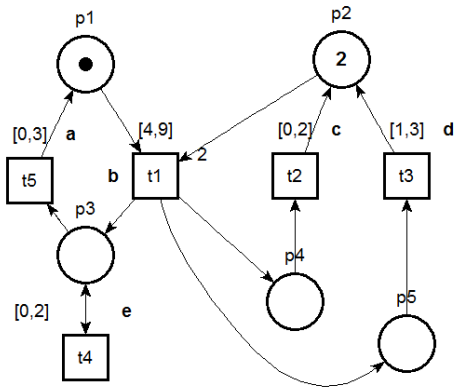


Figura 204: RdP con tiempo (RdPcT)

Este intervalo "dinámico" se expresa como una aplicación, que coincide en cualquier transición t con el intervalo de tiempo $I\,(t)$, en este intervalo puede dispararse la transición. Los límites inferior y superior, del intervalo $I\,(t)$ para una transición t, se conocen como el momento de disparo antes (denotado $DMín\,(t)$) y el momento de disparo posterior (denotado $DMax\,(t)$) de la transición.

Otra interpretación se puede hacer para cuando se tiene múltiples marcas que habilitan una transición con múltiples tiempos, esto será abordado más adelante.

Estado y Regla de Disparo

Definición 23: Estado de una RdPcT

El estado de una RdPcT se define por la expresión $E = (M, I)$ donde:

M es la marcación de la RdP,

I es una aplicación del intervalo de disparo.

Por lo que si tenemos dos estados: $E = (M, I)$ y $E' = (M', I')$, podemos interpretar que el cambio de estado obedecen a dos eventos distintos, los que son:

- Disparo de una transición: $E \xrightarrow{t} E'$
- Paso del tiempo: $E \xrightarrow{\tau} E'$

Definición 24: Estado inicial de una RdPcT

El estado inicial de una RdPcT se define como: $E_0 = (M_0, I_0)$, donde:

M_0 es la marcación inicial de un RdP

I_0 Concuerda con cada marcación habilitada y el intervalo de disparo estático $IS(t)$.

Cualquier transición no habilitada se corresponde con un intervalo vacío, no hay cuenta.

Supongamos la red de la Figura 205, dónde la marca inicial $M_0 = (2, 0, 1)$, $I_0 = (\#, 0, 0, 0)$ (en carácter # indica que esa transición no está sensibilizada), los ceros indican que aún no ha transcurrido tiempo, puesto que es el estado inicial. La tupla (M_0, I_0) se corresponde con el estado inicial del sistema.

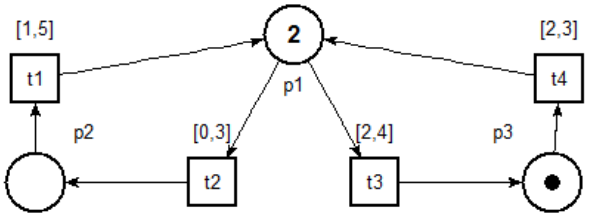


Figura 205: Estado inicial de una RdPcT

Definición 25: Regla de disparo de una RdPcT

Una transición t puede dispararse en el instante θ para una marca $E = (M, I)$ si se cumple:

La transición t está habilitada por lo que se cumple $M \geq Pre(t)$

θ cumple para la transición t : $\theta \geq Dmin(t)$, tambien

θ cumple para la transición t : $\theta \leq DMax$

por la marca M

$$\forall\, k, M \geq Pre(k) \Rightarrow \theta \leq DMax(k)$$ [4]

Definición 26: Transición en conflicto

Dos transiciones $t\ y\ t'$ están en conflicto para una marca M cuando ambas son habilitadas y comparten al menos una plaza p, y se cumple que:

$$M(p) < Pred(p, t) + Pred(p, t')$$

Definición 27: Cambio de estado

El disparo de una transición habilitada t en un instante θ, para una marca $E = (M, I)$ a una marca $E' = (M', I')$ está determinado por:

La marca M' es resultado de la operación $M' = M - Pre(t) + Post(t)$

El nuevo intervalo de disparo $I'(k)$, para todas las transiciones K, es definido por:

- Los k que no están sensibilizado en $M', I'(k) = \emptyset$

[4] Una transición t es habilitada por múltiples marcas si para un marcado M se cumple $M \geq k, Pred(t)\ para\ un\ entero\ k > 1$

- Los k distintos de t que están sensibilizados en M y no están en conflicto en t para M, tendremos $I'(k) = [max(0, DMin(k) - \theta),\ DMax(k) - \theta]$, si $DMax(k)\ no\ es\ finito\ I'(k) = [\max(0,\ DMin(k) - \theta, \infty[$
- Para todos los otros casos $I'(k) = IS(k)$

Supongamos la red de la Figura 206, dóndes las Transiciones $t_1\ y\ t_4$ estan sensibilizadas con la marca inicial $M_0 = (0, 1, 1)$ por lo que $I_0 = (0, \#, \#, 0)$, puesto que aún no ha transcurrido tiempo. El estado inicial será: $E_0 = ((0, 1, 1), (0, \#, \#, 0))$, después de 1.3 unidades de tiempo, si no hay disparo será: $E_1 = ((0, 1, 1), (1.3, \#, \#,\ 1.3))$ y después de 1 unidad de tiempo más ser:

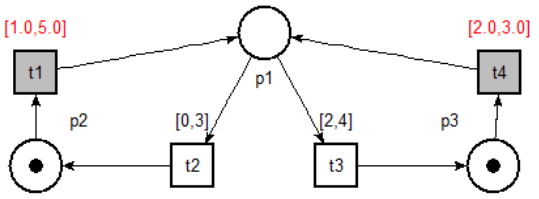


Figura 206: RdPcT con transiciones sensibilizadas (rojo)

$E_2 = ((0, 1, 1), (2.3, \#, \#,\ 2.3))$. Si ahora disparamos t_4, el estado será: $E_3 = ((1, 1, 0), (2.3, 0, 0,\ \#))$.

Podemos representar a estos cambio de estados como: $E_0 \xrightarrow{1.3} E_1 \xrightarrow{1.0} E_2 \xrightarrow{t_4} E_3$

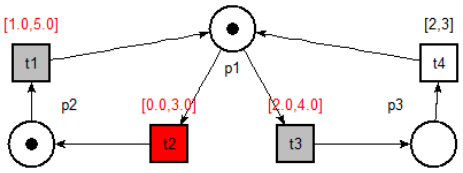


Figura 207: RdPcT con transiciones t4 disparada

En otras palabras, a las transiciones no habilitadas para la nueva marca M´ le corresponde un intervalo vacío; a las transiciones que permanecen habilitadas después del disparo, en el instante θ (transición de M a M´) hay que desplazarles el intervalo de tiempo en θ, manteniendo la restricción de tiempo no negativo; todas las transiciones que aparecen habilitadas en M´ reciben el intervalo de disparo estático. Notar que si t permanece habilitado después de su disparo, éste recibirá el intervalo estático como nuevo intervalo.

Conjunto de estados, secuencia de disparos

La definición de las reglas de disparo realizada, nos muestra que es posible alcanzar un estado a partir de una secuencia de disparos y un intervalo de tiempo establecido por los intervalos de disparos, lo que podemos establecer como el par (s, u) donde s es la secuencia de disparos y u es la secuencias de intervalos de tiempos según se ha definido en el apartado anterior.

Representar las operaciones de una RdPcT por un grafo de alcanzabilidad, como lo hicimos en las RdP, es generalmente imposible [1], puesto que el tiempo es continuo y las transiciones pueden ser disparadas en cualquier momento por lo que tendremos infinitos sucesos para esta regla de disparo. Por lo cual, definiremos la **clase estado**, con el fin de obtener una representación finita de este conjunto infinito de estados.

Aunque los estados definidos, para las RdPcT, tienen generalmente infinitos sucesores, no existe una clase específica de tiempo de RdPcT donde cada estado pueda tener un número finito de sucesores, y por el cual el grafo de estados sea definido. Por lo que definimos redes en las que el intervalo estático asociado con cada transición es $[0, \infty[$, para estas redes, tenemos la siguiente propiedad:

Definición 28: Isomórfico de grafo de alcanzabilidad de la RdPcT

Dada una TRdP $\{P, T, Pre, Post, M_0, IS\}$, si el IS asociado a cada transición tiene como intervalo de disparo $[0, \infty[$, se cumple que el grafo de alcanzabilidad de la RdPcT es isomórfico con el grafo de la RdP marcada $\{P, T, Pre, Post, M_0\}$. Eso nos permite ver a las RdPcT, intervalos de tiempo son $[0, \infty[$, como RdP.

Dominio de disparo

El estado de una RdPcT está definido por $E = (M, D)$ donde M es la marca y D el vector de referencia del dominio de disparo, con un componente asociado a cada disparo sensibilizado por M. Donde el elemento de orden i del conjunto D es el intervalo de disparo asociado con la transición habilitada por el elemento de orden i de M. Estos dominios pueden ser expresados como el conjunto de soluciones de sistemas de inecuaciones lineales con una variable asociada a cada transición habilitada, como sigue:

En la RdPcT de la Figura 204, el estado inicial E_0 se expresa por la tupla (M_0, D_0), dónde:

> $M_0: p_1(1), p_2(2)$, lo que nos dice que la plaza uno es marcada por un token y la plaza dos por dos token.
>
> $D_0: es\ el\ conjunto\ de\ soluciones\ de\ la\ desigualdad\ 4 \leq t_1 \leq 9$, lo que nos dice que la variable t_1 es asociada con la transición t_1.

Es decir que la única transición habilitada es t_1, si el disparo de t_1 se realiza en un intervalo de tiempo relativo θ_1 entre $[4, 9]$, del estado E_0 alcanzaremos el estado $E_1 = (M_1, D_1)$, donde:

> $M_1: p_3(1), p_4(1), p_5(1)$.
>
> $D_1: es\ el\ conjunto\ de\ soluciones\ de\ la\ desigualdad\ en\ las\ transiciones\ sensibilizadas$
>
> (t_2, t_3, t_4, t_5), por lo que las desigualdades son:

$$0 \leq t_2 \leq 2$$

$$1 \leq t_3 \leq 3$$

$$0 \leq t_4 \leq 2$$

$$0 \leq t_5 \leq 3$$

Aquí θ_1 no aparece, puesto que la única transición sensibilizada por M_0 fue t_1 y θ_1 es su tiempo relativo y en M_1 no está habilitada ninguna transición que estuviera habilitada en M_0 .

Si en el estado E_1 disparamos la transición t_2 en el instante θ_2, se alcanzará el estado E_2, que podemos expresar como:

M_2: $p_3(1), p_2(1), p_5(1)$.

D_2: $transiciones\ sensibilizadas$ (t_3, t_4, t_5), por lo que las desigualdades son:

$\max(0, 1 - \theta_2) \leq t_3 \leq 3 - \theta_2$

$0 \leq t_4 \leq 2 - \theta_2$

$0 \leq t_5 \leq 3 - \theta_2$

El parámetro θ_2 que aparece en la desigualdad es una constante, y depende del instante en que se disparo t_2 por lo que es un real en el intervalo $[0, 2]$. Por lo que los estados siguientes a E_1 son infinitos, puesto que para cada valor de θ_2 queda definido un estado distinto.

Caracterización del comportamiento de una RdPcT

Clase de Estado

Hasta ahora hemos considerado alcanzar un determinado estado desde un estado inicial, según una secuencia de disparos (s, u), también podemos considerar al conjunto de todos los estados que pueden ser alcanzado por los intervalos de tiempos u y la secuencia de disparos s.

Definición 29: Clase de estado

Una clase de estado está asociado con cualquier secuencia de transiciones sensibilizadas desde el estado inicial, la clase de estado asociada con la secuencia s se define como un par (M, D) en la que M es el marcado alcanzado desde el marcado inicial por el disparo de la secuencia s y D caracteriza los dominios de disparo de todos los estados alcanzables desde el estado inicial mediante la secuencia s.

El dominio de disparo de la clase de estado puede expresarse como el conjunto de soluciones del sistema de in-ecuaciones lineales formado por la desigualdad con una variable asociada con la transición sensibilizada por el marcado. Para cada transición se expresan las relaciones entre momentos de disparo diferentes.

Intuitivamente, el dominio de disparo de una clase abstrae el momento relativo del disparo de la transición.

Una clase de estado se verá como un par, $C = (M, D)$, donde:

- M es una marca
- D es un dominio de disparo, es decir el conjunto de soluciones de la inecuación lineal $A_t \leq b$ en la cual A es una matriz, b es un vector donde la i-esima variable t_i del vector t y está asociado con la i-esima transición habilitada por la marca M.

Transiciones entre clases de estado

Una transición t es disparable en una clase de estado $C = (M, D)$ sii se cumplen las siguientes condiciones:

- t está habilitada por la marca M
    - Es la condición de sensibilización de las RdP ordinarias

- El dominio D tiene un vector en el que la componente de la transición tiene un valor menor o igual a la componente relacionada a la transición habilitada por M
    - Esta condición expresa la condición del disparo por los límite de tiempo

La segunda condición se cumple si la inecuación del sistema tiene solución para la transición t, lo cual se expresa de la siguiente manera:

- $A \,.\, \bar{t} \leq b$
- $t_i \leq t_f, \forall\, t_i \neq t_f$

El cálculo de la nueva clase $C' = (M', D')$ se realiza como se indica a continuación:

1. La nueva marca M'es calculada con la ecuación de estado de las RdP ordinarias.
2. El dominio D' es determinado con los cuatros pasos que se describen a continuación:
    a. El sistema de inecuaciones $A \,.\, \bar{t} \leq b$ es actualizado según la condición de transición sensibilizada, la transición t es sensibilizada si el sistema de in-ecuaciones tiene solución.
    b. Las variables asociadas con transiciones t en conflictos son eliminadas según las restricciones del sistema.
    c. En este sistema reducido, que consiste de transición a disparar que coincide con la variable ti. En este sistema simplificado, cualquier variable $t_j, con\ j \neq i$ se sustituye para eliminar variable de t_i.
    d. En este nuevo sistema, una nueva variable es introducida para cada nueva transición habilitada, obligando a pertenecer al intervalo de sensibilización estática de la transición asociada. Ahora nuevas transiciones son habilitadas por M' que no estaban habilitadas en $M - Pre(t)$, tal cual t es habilitada en M'.

En la etapa (a), el dominio de partida se reduce al conjunto de vectores con transiciones sensibilizadas en el intervalo de tiempo disparable, el vector t que primero puede ser disparado, entre transiciones habilitadas.

En el paso (b) se eliminan las variables correspondientes a las transiciones distintas de t y conflictivas con t; dicha eliminación no modifica los intervalos de disparo de las transiciones restantes, ni las posibles relaciones entre los momentos de despido de dichas transiciones.

El conjunto de soluciones del sistema determinada en la etapa (c) puede ser visto como el dominio de disparo de las transiciones (distinto de t) que se mantuvo habilitado en la disparo de t, expresado como el nuevo origen del tiempo, el momento en que la transición t fue disparada.

La etapa (d), simplemente introduce intervalos de disparo de las transiciones recién habilitadas, siendo igual a sus respectivos intervalos estáticos. Si t queda habilitado en su propio disparo, entonces se considera como recién habilitado.

Puesto que M_0 es el estado inicial la clase C_0 se corresponde con E_0

La Figura 208 muestra una RdPcT con estado inicil $M_0: p_1(1), p_2(2)$, donde la transición t_1 está sensibilizada, lo que implica que podrá ser disparada en el intervalo $4 \leq\ t_1 \leq 9$.

Es decir que el momento de disparo puede estar en este intervalo y es expresado por:

θ_1 entre $[4,9]$. Este disparo nos lleva a la clase C_1 que consiste de un único estado E_1.

Donde $M_1: p_3(1), p_4(1), p_5(1)$, y las transiciones sensibilizadas son (t_2, t_3, t_4, t_5). Disparando t_2 desde la clase C_1, alcanzamos la clase C_2, donde su marca es:

$$M_1 = M_1 - Pre(t_2) + Post(t_2)$$

Para calcular C_2 aplicamos los cuatro pasos enunciados anteriormente.

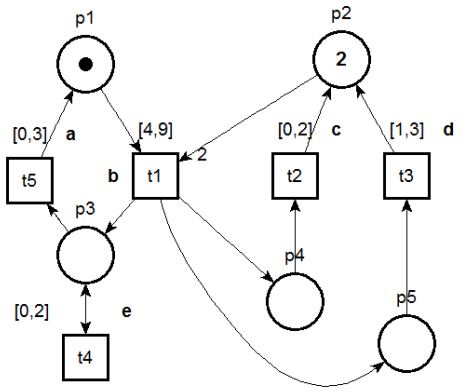


Figura 208: RdPT con estado inicil M_0

Paso **a**

$0 \leq t_2 \leq 2$ $1 \leq t_3 \leq 3$ $0 \leq t_4 \leq 2$ $0 \leq t_5 \leq 3$	$t_2 \leq t_3$ $t_2 \leq t_4$ $t_2 \leq t_5$

Paso **b**

No hay transición en conflicto con t_2, por lo que los dos pasos tienen el mismo sistema de inecuaciones, $D_2(a) = D_2(b)$

Paso **c**

Por sustitución de variables obtenemos el siguiente sistema de inecuaciones:

$0 \leq t_2 \leq 2$ $1 \leq t_2 + t_3 \leq 3$ $0 \leq t_2 + t_4 \leq 2$	$0 \leq t_2 + t_5 \leq 3$ $t_2 \leq t_2 + t_3$	$t_2 \leq t_2 + t_4$ $t_2 \leq t_2 + t_5$

Eliminando t_2 obtenemos $D_2(c)$

$0 \leq t_3 \leq 3$ $0 \leq t_4 \leq 2$	$0 \leq t_5 \leq 3$ $t_4 - t_3 \leq 1$	$t_5 - t_3 \leq 2$

Paso **d**

No hay nueva transición habilitada, por lo que: $D_2(d) = D_2(c)$. Ahora la clase C_2 está determinada por:

$M_2: p_2(1), p_3(1), p_5(1)$

D_2: es el conjunto de soluciones en (t_3, t_4, t_5)

$0 \leq t_3 \leq 3$ $0 \leq t_4 \leq 2$	$0 \leq t_5 \leq 3$ $t_4 - t_3 \leq 1$	$t_5 - t_3 \leq 2$

El ejemplo de la Figura 209, muestra donde hay un generador períodico de eventos, para realizar distintas tareas en region crítica. En los párrafos que continuan se analizan los distintos estados, invariantes y clases con el fin de verificar el modelo.

state 0

props p1 ps

trans t1/1

estado 1
props p1 pm*w ps
trans t1/1 t2/2

estado 2
props p1 p3 pm*w
trans t1/2 t3/3

estado 3
props p1 p4 pm*w
trans t4/1 t1/3

tr t1 [10,10] p1 -> p1 pm

tr t2 pm ps -> p3

tr t3 [2,4] p3 -> p4

tr t4 [1,2] p4 -> ps

pl p1 (1)
pl ps (1)

invariantes de plaza
p1
p3 p4 ps

invariante de transición
t1 t2 t3 t4

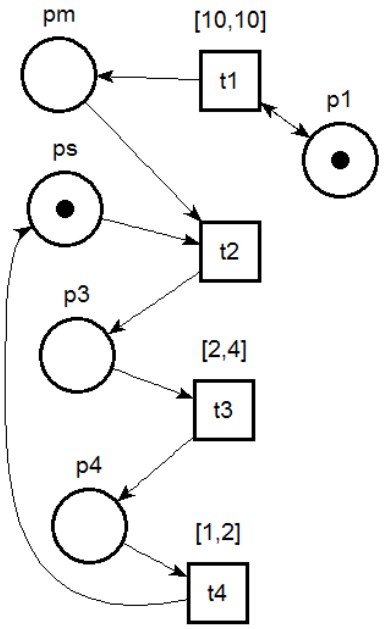


Figura 209: RdPcT con generador periódico de eventos

path de 0 a 1 incrementa la marca enumeracion parcial.

2 clase(s), 1 Transicióne(s)

estado 0
marc: p1, ps
clock vector: t1 = 0
domin:o de disparo 10 <= t1 <= 10
estado 1
marcado: p1, pm, ps
clock vector: t1 = 0, t2 = 0
domini: 10 de disparo <= t1 <= 10, 0 <= t2

Igualdad de clases

Dos clases son iguales sii su marcado y dominio respectivo son iguales. La comparación de dos dominios es equivalente a comparar conjuntos de soluciones de dos sistemas de inecuaciones lineales con variables idénticas. Esta comparación es en general costosa, pero se puede realizar de manera eficiente en caso específico, como se explica a continuación.

Definición 30: Dominio de disparo

Un dominio de disparo, en cualquier estado de clase, se puede expresar como el conjunto de soluciones de un sistema de desigualdades lineales, con un máximo de dos variables por la desigualdad, con la siguiente forma general:

$$a_1 \leq t_i \leq b_i \; para \; cualquier \; i$$

$$t_j - t_k \leq c_{jk} \; para \; cualquier \; j, \; donde \; k \neq j$$

Donde t_i es la variable asociada al marcando habilitada de la transición i de la clase y a_1, b_i y

c_{jk} son constantes (donde b_i y c_{jk} puede ser infinito).

Para el cálculo de la ruta más corta, se puede utilizar el algoritmo de Floyd-Warshall [273], con complejidad temporal $O\ (n^3)$ y espacial $O\ (n^2)$ (n es el número de vértices del gráfico). Además este algoritmo nos permite verificar la consistencia del sistema de inecuaciones. El sistema s es consistente si la gráfica de restricción asociada no contiene ningún ciclo de peso negativo, es decir, si para cualquier vértice x se cumple $D(x,x)\ \geq 0$.

Grafo de la clase

Podemos construir un grafo que represente la relación de clases alcanzables de estado definidos por la regla de disparo; donde sus vértices son las clases de estado, donde está contenida la clase inicial, existe un arco marcado t con origen C y con extremo C' si y sólo si la transición t es disparable y como resultado de la clase C se alcanza la clase a C'.

De la clase de estado se desprende que cualquier secuencia de disparos desde el estado inicial corresponde a un camino gráfico cuyo origen es la clase inicial, la existencia de un camino gráfico marcado con s entre la clase inicial y un vértice C requiere que exista al menos una secuencia de disparo s e intervalos u tal que (s,u) pueda alcanzarse. Debe tenerse en cuenta que el gráfico de la clase no permite determinar directamente la definición del conjunto de intervalos alcanzables entre dos clases, pero sí el conjunto de secuencias de disparo. Sin embargo, tales intervalos de tiempos pueden lograrse mediante una adaptación del método.

Grafos de Marcas y Grafos de Clases

En un grafo de clases pueden existir varias clases con la misma marca pero con diferentes dominios de tiempo. Esto simplemente significa que el tiempo de la transición relacionada representa un comportamiento y la diferencia es expresada por la clase.

Lo cual podemos ver una TRdP, suponiendo una única marca, si observamos el efecto de los diferentes intervalos de tiempo asignados a las transiciones; donde la primera se corresponde con el intervalo $[0,\infty[$, y luego partiendo del estado inicial es posible disparar cualquier secuencia de transiciones.

Las Figura 210 muestra una TRdP muy simple, si consideramos que no hay restricciones temporales:

$$IS(t_1) = IS(t_2) = [0,\infty[$$

Obtendremos la alcanzabilidad de la red.

0 : p1 p2

1 : p1*w p2

2 : p1 p2*w

3 : p1*w p2*w

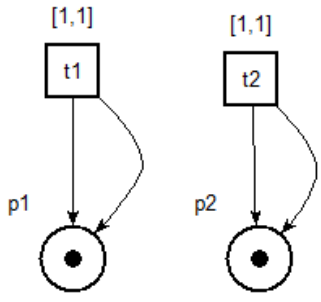


Figura 210: RdPcT sin restricciones temporales

En este ejemplo de la Figura 211, mediante la combinación de los intervalo [1,1] y las dos transiciones, se obtiene un gráfico con tres clases alcanzables.

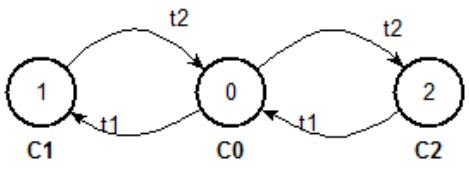


Figura 211: Grafo con tres clases alcanzables.

La clase C_0 se corresponde con la marca inicial, sin disparo, y las clase C_1 y C_2 que se alcanzan según se dispare t_1 o t_2 respectivamente. Aquí se puede ver que ciertas secuencias de disparo ya no son posibles.

Por ejemplo, a partir de la clase inicial, t1 o t2 no puede ser disparado más de una vez consecutivamente, y, de cualquier estado, t1 (t2, respectivamente) nunca puede haber sido disparado más de dos veces sin disparar t2 (t1, respectivamente).

Las inecuaciones correspondientes a cada clase resultan:

Clase C_0	Clase C_1	Clase C_2
Marca: p1 p2	Marca: p1*w p2*w	Marca: p1*w p2*w
Dominio:	Dominio:	Dominio:
$1 <= t1 <= 1$	$1 <= t1 <= 1$	$0 <= t1 <= 0$
$1 <= t2 <= 1$	$0 <= t2 <= 0$	$1 <= t2 <= 1$

Figura 212: Inecuaciones de las tres clases

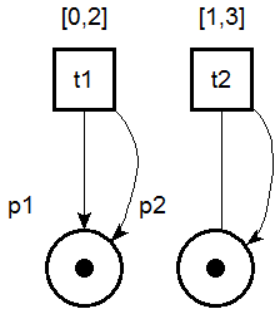


Figura 213: RdPcT con restricción temporal

Finalmente, la red que se muestra en la Figura 213, tiene asociando el intervalo [0,2] con t1, y [1,3] con t2, establece un número de clases y un comportamiento que es diferente al anterior, el que podemos ver en la Figura 214.

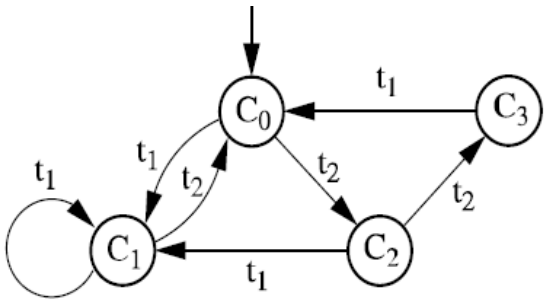


Figura 214: Grafo de clases de la RdPcT

Clase C_0	Clase C_1	Clase C_2	Clase C_3
Marca: p1 p2	Marca: p1*w p2*w	Marca: p1*w p2*w	Marca: p1*w p2*w
Dominio:	Dominio:	Dominio:	Dominio:
$0 <= t1 <= 2$	$0 <= t1 <= 2$	$0 <= t1 <= 1$	$0 <= t1 <= 0$
$1 <= t2 <= 3$	$0 <= t2 <= 3$	$1 <= t2 <= 3$	$1 <= t2 <= 3$

Análisis utilizando el gráfico de la clase de estado

En el análisis del comportamiento de un sistema dependiente del tiempo, donde éste interviene de diferentes maneras según se lo especifique.

Si suponemos que es necesario el tiempo para la correcta especificación de un sistema, en el conjunto de restricciones o afirmaciones del sistema, el tiempo debe aparecer como una variable. Cuando el sistema está representado por una RdPcT, tales propiedades se expresan como propiedades del conjunto de clases, la red y la secuencia de transiciones. De todas las propiedades de las RdPcT, consideramos que son importantes las siguientes:

a) Invariantes del conjunto de marcas alcanzables por la red, tales como: restricciones de exclusión mutua o la ausencia de estados bloqueados.
b) Propiedades de un conjunto de secuencias de disparos, tales como vivacidad y terminación.
c) Restricciones orientadas al tiempo, como límites por el tiempo en la alcanzabilidad de una marca, o la falta en el tiempo de una sincronización.

Propiedades del tipo (a) o (b) en general se pueden definir en un gráfico de estados de clase tal como se ha definido en el apartado anterior. Los invariantes verificados mediante un examen exhaustivo de marcado en las clases del estado. Estas propiedades se verifican con las secuencias de disparos en el conjunto de caminos de los gráficos de la clase.

La mayoría de las restricciones enumeradas en (c) se reducen a verificar la falta de tiempo en las secuencias específicas, con un determinado origen o la extremidad, con una duración impuesta, o bien una ruta restringida.

Marcado accesible

Se trata de determinar si es posible alcanzar una marca M' desde una marca M en una RdPcT.

Denotamos el conjunto de marcas que se puede llegar a partir de su marca inicial por $R\ (M)$. Lo cual es enunciado en la siguiente definición:

Definición 31: Alcanzabilidad de una TRdP

El problema de la accesibilidad a una marca a partir de un marcado inicial es indecidible en una RdPcT [274].

Límite

Recordemos que una RdP está acotada si el marcado de cualquier lugar supone un límite superior. En cuanto a la accesibilidad de un marcado con una restricción de tiempo de RdP se reduce a demostrar tal propiedad de las redes de arco inhibidor y es indecidible.

Definición 32: Límite de una RdPcT

El problema de la cota en la una marca de un TRdP es indecidible [274] .

Si consideramos un tiempo acotado para la RdPcT es posible probar que el número de clases de estado es finito y el conjunto de dominios de disparo para un tiempo finito de una RdP es finito, y no se requiere que la red sea acotada.

Definición 33: RdPcT tiempo finito

El número de clases de estado de una RdPcT es finito si y sólo si el tiempo está delimitado por la red.

Definición 34: Uno, condición para que RdPcT sea acotada

Una RdPcT es acotada si no se cumple que para cualquier par de clases de estado $C = (M, D)$ y $C' = (M', D')$ se cumple:

C' es alcanzable desde C;
$M' > M$

La siguiente condición suficiente es más débil que la anterior

Definición 35: Dos, condición para que RdPcT sea acotada

Una RdPcT es acotada si no se cumple que para cualquier par de clases de estado $C = (M, D)$ y $C' = (M', D')$ se cumple:

C' es alcanzable desde C;
$M' > M$
$\mathrm{D}' = \mathrm{D}$

Definición 36: Tres, condición para que RdPcT sea acotada

Una TRdP es acotada si no se cumple que para cualquier par de clases de estado $C = (M, D)$ y $C' = (M', D')$ se cumple:

C' es alcanzable desde C;

$$M' > M$$
$$D' = D$$
$$\forall p, M'(p) > M(p) \Longrightarrow M'(p) max_{t \in T}(Pre(p,t))$$

La última condición expresa que cualquier lugar, cuya marca está incrementanda entre M y M', contiene un número de marcas al menos igual al mayor número de arcos entrantes a las próximas transiciones.

Propiedades de grupo de marcas o secuencias de disparo

Cuando la propiedad de red acotada ha sido mostrada para una RdPcT, verificando cualquier propiedad en relación con las marcas o secuencias de disparo se hace posible un examen exhaustivo de la gráfica de clases de estado; puesto que las propiedades de una RdP delimitada se analizan utilizando su gráfica de marcas alcanzadas. En particular, las propiedades de ausencia de estancamiento y cuasi-vivacidad, que se define simplemente para las RdP, se pueden verificar para cualquier tiempo acotado, utilizando las gráficas de clase del estado. Invariantes o propiedades específicas relativas a las marcas alcanzadas o secuencias de disparo alcanzables se demuestran de la misma forma.

Análisis dependientes del tiempo, la existencia de secuencias temporales

La verificación de las propiedades puramente temporales, como la accesibilidad de un dato que marca a otro dentro de un intervalo de tiempo determinado, o una configuración en función del tiempo de duración de la secuencia de disparo, es importante de considerar.

Un gráfico de la clase de estado por lo general no nos permite completar con éxito este tipo de análisis a través de un simple examen de las clases o de disparos. Sin embargo, un intervalo de disparos independiente puede ser fácilmente asociado con cualquier arco, que es el intervalo relativo de la transición relevante que se dispara; tal intervalo se puede extraer de las desigualdades que representan el dominio de disparo de la clase. Sin embargo, para una secuencia de disparo dada, el conjunto de las secuencias temporales alcanzables con una secuencia de transición, tal como soporte, no puede deducirse a partir de intervalos asociados de esta manera con las secuencias de transiciones, en realidad, los momentos de disparos relativos son generalmente interdependientes.

Esto no significa que los dominios de disparo no tienen relación con los tiempos alcanzables, sino simplemente no podemos interpretarlos de la manera que se muestra anteriormente. Para obtener la existencia de secuencias de tiempo, se muestra que, para cualquier secuencia de disparo σ comenzando por una transición disparable, podemos construir un sistema de inecuaciones en la que cada solución θ determina una secuencia de tiempos realizable ($\sigma,\ \theta$). La técnica consiste en la aplicación de la regla de la prosecución de las clases, pero sin eliminar, en la etapa (c), la variable correspondiente a la transición disparada. En lugar de eliminar a t_i como variable, se renombra como una nueva variable θ que se interpreta como el momento del despido relativo a la transición t_i, apareciendo en términos de un parámetro de una clase y la próximas clases en la secuencia considerada. De este modo, mediante el cálculo extendido de estas clases, a lo largo de una secuencia de disparo dado σ , la última clase lograda contiene tantos parámetros $\theta_1, \theta_2, \ldots \theta_n$ como el número de transiciones disparadas en la secuencia, y cualquier solución θ del sistema proporcionará la secuencia de tiempos.

Sumando las limitaciones de tiempos adicionales en tales variables, podemos centrarnos más en la existencia o la imposibilidad de secuencias de tiempo específicas. En particular es fácil de obtener la determinación de la secuencia más corta o más larga, con un origen determinado.

Semánticas relacionadas con el grado de sensibilización

El grado de sensibilización de una transición es el número de veces que una transición puede ser disparada, como máximo, concurrentemente. Esta noción modifica la definición de la semántica de disparo. La cantidad de disparos se resuelven en una la transición en un cálculo, esto es: la semántica de servidor sencillo (único) o múltiple. Cuando no se considera el disparo simultáneo y concurrente de las Transiciones de la red, se trata de semántica de servidor sencillo (single server semantics). Sin embargo, si el grado de sensibilización de una transición es mayor que uno, es posible considerar la ejecución concurrente de la misma transición. Esta es una semántica de servidor múltiple (multiple server semantics)

Semántica de tiempo de disparo o de tiempo de sensibilización

Son las dos formas más difundidas de definir el tiempo asociado a una transición, lo que impacta en la forma de progreso del marcado, según se conserve o no la atomicidad del disparo:

- El parámetro temporal indica el tiempo necesario para sacar las marcas de las plazas de entrada, que sensibilizan el disparo, y colocarlas en las plazas de salida de la transición, esto se realiza en forma atómica. *Este tiempo se conoce como tiempo de sensibilización.*
- La transición puede ser considerada como no atómica, con tres fases, las que son: sacar las marcas de las plazas de entrada que sensibilizan a la transición, realizar el disparo y colocar las marcas en las plazas de salida de la transición. Estas operaciones no son atómicas y tienen una duración. *Se conoce como semántica de duración*.

Formas de especificar el tiempo asociado a las Transiciones

En general las RdPTp son usadas para modelar, por lo que el tiempo representa la duración de una actividad, esto puede depender de diversos factores como distintas variables u ocupación del sistema, etc. Por lo que es posible usar:

- Timestamp en los token
- Colores en los token
- Un intervalo asociado a la transición, que especifique el tiempo máximo y mínimo para realizar la actividad (RdPcT)

Este último caso es el más simple de representar y analizar, y permite realizar análisis cuantitativo de ejecución como el tiempo máximo de ejecución, el peor caso, etc.

Este convencionalismo es el que adoptaremos para nuestro trabajo y es el correspondiente a las *Time Petri* Nets definido por [262] [269].

Semánticas de tiempo débil y fuerte

Para reducir el indeterminismo de cuando una transición sensibilizada debe ser disparada las RdPcT, se espesifica un intervalo de tiempo en el cual el disparo debe o puede producirse.

- Decimos que las RdPTp tiene semántica de tiempo fuerte (strong time semantic) cuando un disparo sensibilizado está obligado a realizarse en este intervalo
    - Esta política de disparo es la apropiada para el modelado de sistemas de tiempo real [265]
- Decimos que las RdPTp tiene semántica de tiempo débil (weak time semantic), cuando un disparo sensibilizado no está obligado a realizarse, pero si lo hace, debe ser en el intervalo especificado.
    - Esta política es apropiada cuando se trata de modelar una acción que puede ocurrir solamente durante un período de tiempo, pero que no necesariamente debe ocurrir.

RdP con Tiempo (RdPcT)

En este trabajo se adopta la definición propuesta en [32]

Definicion 37: RdP con Tiempo (RdPcT)

Una RdPcT, es definida por la tupla $\{P, T, Pre, Post,\ M_0,\ IS\}$

- donde $\{P, T, Pre, Post,\ M_0\}$ es una PN y
- $IS: T \rightarrow\ Q^+ x\ (\ Q^+ \cup\ \{\infty\}\ )$ es la función estática del intervalo de tiempo.

Q^+ *es el conjunto de los números racionales positivos junto con el cero*

La función *IS* asocia a cada transición un par: $IS(t_i) =\ (\propto_i, \beta_i)$ que define un intervalo temporal, y se verifica que:

$$0\ \leq \propto_i \leq\ \infty\ , 0\ \leq \beta_i\ \leq\ \infty\ , y\ \propto_i \leq\ \beta_i\ si\ \beta_i \neq\ \infty\ o\ \propto_i <\ \beta_i\ si\ \beta_i =\ \infty$$

Regla de Disparo

Si una transición t_i es sensible en el instante θ_i y continúa sensible durante el intervalo $(\propto_i, \beta_i)$, podrá ser disparada durante este intervalo de tiempo: $(\theta_i + \propto_i, \theta_i + \beta_i)$, La semántica de este disparo es del tipo de tiempo fuerte, el disparo se debe producir en la ventana de tiempo mientras la transición está sensibilizada, por lo que las marcas permanecen en los lugares de entrada durante el tiempo necesario. Una vez que se produce el disparo, atómico, éstas no consumen tiempo.

Casos frecuentes de intervalos:

- Intervalo preciso, $(\propto_i, \propto_i)$, para transiciones con tiempo de sensibilización fijas
- Intervalo sin restricción temporal $(0, \infty)$, como las del formalismo de las RdP. Por lo que inferimos que las RdP son un caso particular de las RdPTp

Definiciones:

- Intervalo estático de disparo $(\propto_i, \beta_i)$
- $\propto_i, earlier\ firing\ \ SMin(t)$ es el instante de disparo más cercano (EFT)
- $\beta_i, later\ firin\ SMax(t)$ es el instante de disparo más lejano (LFT)
- IS es la función estática del intervalo de tiempo

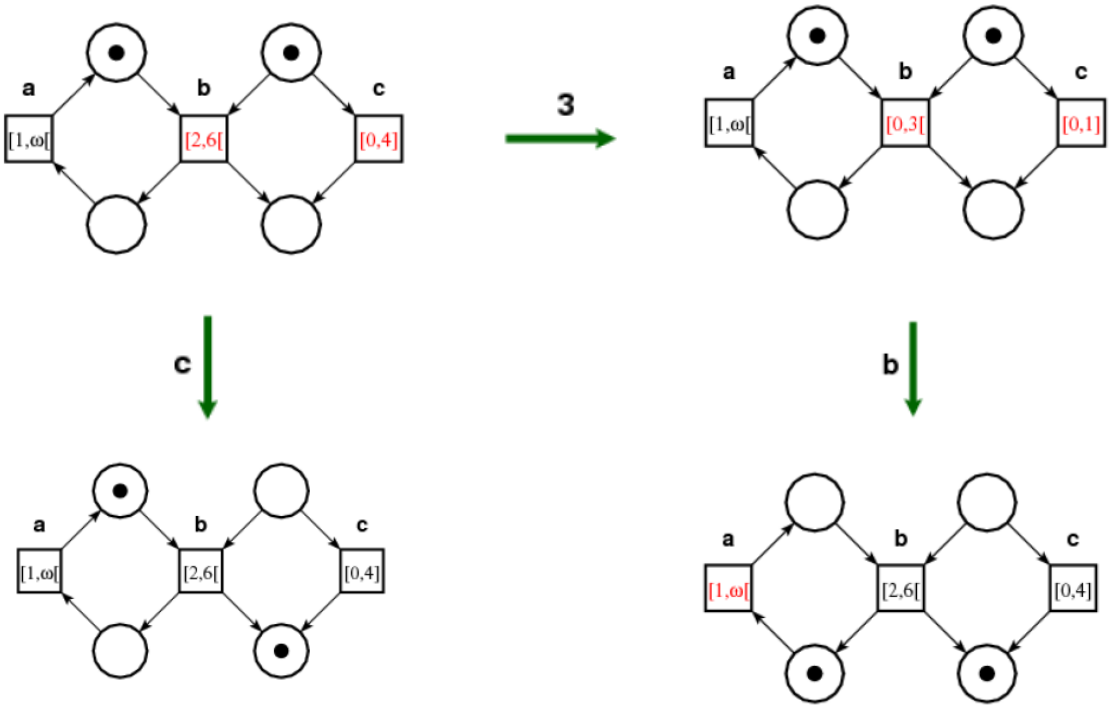


Figura 215: Regla de disparo de RdPcT

Estado

Siendo que el tiempo interviene en la regla de disparo, éste se reflejará en el estado de la RdPcT, es decir su marcado, ya que se debe registrar la cantidad de tiempo que una transición ha estado sensibilizada, puesto que es necesario para decidir la posibilidad de disparo de una transición sensibilizada.

Por lo que definimos el estado de una RdPcT como un par S = (M; IS), donde M es el marcado e IS es un vector de pares de todos los posibles intervalos de disparo de todas las transiciones sensibilizadas por el marcado M e IS es el vector, con la misma dimensión de T. Cuyas componentes son el tiempo transcurrido desde la sensibilización de la transición al instante ɵ de todos los posibles intervalos de disparos.

Conclusiones

Con respecto a la semántica

Con las políticas, nos hemos referido:

- a la elección del próximo evento (para disparar una transición).
- a la posibilidad de instancias simultáneas donde se produzca un mismo evento.
- a la actualización de los temporizadores cuando se produce un paso discreto.

Es de esperar poder programar el procesador con cualquiera de estas políticas y de ser posible realizar distintas programaciones con distintos subconjuntos de transiciones y plazas.

Si bien hemos elegido dos de las semánticas (RdPcT y RdPTm) es importante investigar la posibilidad que el procesador implemente más de una simultáneamente.

Las etiquetas son fundamentales para obtener una relación formal que vincule la RdPT con las acciones a ejecutar por el programa y con los eventos. Esta relación tiene que ser lógica para que la validación del modelo concuerde con el sistema implementado, por esto se ha extendido el modelo con RdPnA.

Anexo B

IP-Core

IP-Core (Intelectual Property Core)

Los IP-Core, o bloque IP son bloques de hardware (secuenciales y/o combinacionales), idealmente reutilizables, de lógica o esquemático, para ser usados en proyectos con FPGAs y/o ASICs, los cuales han sido diseñados y testeados con un propósito específico. Los IP-Core pueden ser licenciados por distintos proveedores.

Sus características son: funcionalidades pre-definidas, tamaño (son relativamente pequeños en relación al sistema donde se integran), tienen un uso específico y pueden ser estandarizados, independizándolo de la herramienta de desarrollo. Otras características con las que se clasifican los IP-Core son: la dependencia tecnológica, verificaciones o predictibilidad y la reutilización en diferentes aplicaciones y tecnologías.

Categorías de IP-Core

Las categorías son: HardCores, SemiRigidos y Soft Cores.

Los **Hard Cores**, son optimizados para una tecnología específica y no pueden ser modificados por el diseñador que los utiliza. El diseño del core tienen una capa predefinida incluida en la arquitectura. Son usados para aplicaciones plug&play, son muy predictibles (los tiempos son fijos) y fiables una vez implementados. Pero no son flexibles, ni portables y poco configurables.

Los **Cores Firmes** o semirrígidos, también incluyen información del mapeo a nivel de compuertas y el código fuente, el que es visible para el diseñador; haciéndolos parcialmente configurables. Podemos encontrar estos IP-Core diseñados y probados en diferentes tecnologías, ofreciendo una buena predictibilidad en cuanto a performance de tiempos de funcionamiento y área utilizada, pero deben ser usados en las configuraciones (placas) del mismo fabricante

Los Soft-Core o Cores Blandos**,** son flexibles, configurables y se expresan con una netlist (lista de compuertas e interconexiones) o como código HDL, por lo que son independientes de la tecnología. Su desventajas son: menos predecibles en cuanto a la implementación y suelen tener un mayor costo de hardware y tiempos, y menor desempeño de procesamiento.

Las Licencias de los IP-Cores se especificados según su funcionalidad. Los más difundidos son diseños de microprocesadores (soft-CPUs), los cuales se licencian como sistemas programables por funcionar como la "base" de muchos sistemas embebidos.

También es común licenciar IP-Core que cumplan la funcionalidad de controladores de periféricos, tales como USB, PCI, SDRAM, etc. Estas interfaces requieren lógica digital y analógica para controlar las comunicaciones y voltajes fuera de la FPGA.

Por último, los cores "cableados" (llamados así por no ser programables por software) son licenciados para funciones específicas, tales como decodificación de audio, video, GPU, etc.

Proveedores de IPs

El rango de desarrolladores y licencias varían desde personas individuales hasta corporaciones multinacionales. Tanto los desarrolladores como los fabricantes de chips, se localizan por todo el mundo, aunque es menos común encontrar desarrolladores en países donde haya leyes débiles para la protección del robo de propiedad intelectual. De todas formas, al momento de conseguir los IP-Cores suelen pertenecer a alguna de las siguientes categorías:

Código abierto, ofrecen una amplia variedad de diseños, en diferentes lenguajes, por lo general bajo licencias GPL o similares. Uno de los ejemplos más relevantes es OpenCores.org

Proveedores, estos mantienen catálogos de múltiples vendedores, proveyendo servicios de búsqueda y comercialización.

Soft-Cores [275], Un Software microprocesador o también llamado Soft-CPU es un microprocesador que puede ser implementado completamente utilizando un semiconductor programable (suficientemente grande). Los soft CPU son particularmente destacados por su alto nivel de configuración, pudiendo agregar y quitar funcionalidades (módulos) en base a las necesidades específicas del proyecto.

El objetivo de embeber un CPU en un chip, es integrar un CPU con la lógica necesaria sobre la FPGA. A grandes rasgos la CPU se encarga de la ejecución de programas secuenciales y cálculos, y la lógica de la FPGA de las interfaces y el procesamiento paralelo, facilitando el diseño e interconexiones, entre otras cosas.

Los principales factores en la selección de un Soft-CPU son: costos de implementación, Performance, Herramientas, Sistemas Operativos.

Los costos son complejos de determinar, ya que no solo incluye el costo del microprocesador en sí mismo, sino también los costos de desarrollo de software y de hardware, incluso factores como reutilización de código, flexibilidad, portabilidad, curva de aprendizaje, etc.

La performance, es una característica que puede variar en cada implementación y dependiendo de las características inherentes u opcionales al CPU, tales como tipos y tamaños de caché, buses, controladores, etc.

Las herramientas de diseño y desarrollo también deben ser consideradas, evaluando características como disponibilidad, facilidad de uso, compatibilidades, debugger, etc.

Es deseable utilizar un sistema operativo para proveer una capa de abstracción para el software, reduciendo los esfuerzos de desarrollo en muchos casos. Algunas de las consideraciones en la elección pueden ser las latencias de interrupciones, tamaño del kernel, servicios disponibles, e incluso soporte de componentes como USB, algoritmos de encriptación, conexiones Wireless, etc.

Las soft-CPUs son provistas por diferentes entidades. Tienen la ventaja de estar optimizadas para la tecnología de la misma fábrica a un costo relativamente bajo. Además suelen disponer de múltiples periféricos para agregar y de herramientas de desarrollo amigables. Por otro lado, el código es cerrado, y se depende de una licencia y proveedor/tecnología específicos.

Los principales ejemplos son de los fabricantes Xilinx y Altera. Xilinx provee los softcores Picoblaze (8 bits), MicroBlaze (32 bits) y Altera provee el Nios II (32bits) en tres configuraciones (economy, standard and fast).

Los Proveedores de IP, son empresas dedicadas al desarrollo de IP-Cores, lo que hace que tengan un buen soporte y documentación, así como también que no sean específicos a una tecnología o fabricante. Las licencias son variables en cada caso, algunas veces dependientes de la aplicación del core, como es el caso del LEON3, desarrollado por Aeroflex Gaisler, el cual está disponible (bajo GPL) para desarrollos no comerciales.

Existe una Comunidad Open Source, donde los cores tienen código abierto y/o libre, y se dispone de gran cantidad de IPs, pero muchos de estos desarrollos son individuales, los cores no siempre están completos, la calidad y la documentación es variable. El ejemplo más sobresaliente es el OpenRISC1200 creado por la comunidad opencores.org.

Anexo C

Monitores

Introducción

Los monitores son mecanismos de abstracción de datos (que representan recursos en forma abstracta), con variables que caracterizan el estado del recurso. Los monitores tienen como finalidad sincronizar y comunicar sistemas informáticos que cooperan haciendo uso de memoria compartida [276].

La programación de un monitor es modular, por lo que se divide al monitor en módulos, procedimientos o funciones con el fin de simplificarlo, hacerlo más legible y mantenible. Esto resulta en un conjunto de procedimientos y datos relacionados que interactúan entre sí para alcanzar un objetivo, que ocultan los datos haciéndolos locales al módulo.

Partes de un Monitor

Un monitor tiene cuatro componentes: inicialización, datos privados, procedimientos del monitor y cola de entrada.

- Inicialización: contiene el código a ser ejecutado cuando el monitor es creado.
- Datos privados: contiene los datos y procedimientos privados, que sólo pueden ser usados desde dentro del monitor y no son visibles desde fuera.
- Procedimientos del monitor: son los procedimientos que exporta el monitor, para ser llamados desde fuera del monitor.
- Cola de entrada: contiene a los hilos que han llamado a algún procedimiento del monitor pero no han podido adquirir permiso para ejecutarlos aún.

Gestión de un Monitor

Un monitor no es un hilo o proceso, por lo tanto no tiene hilo o proceso de control. Esto implica que las rutinas de monitorización son ejecutadas por el hilo o tarea que utiliza al monitor.

Gestión de las colas del monitor: cuando un proceso activo ejecuta uno de los procedimientos exportados del monitor, se dice que "*está adentro del monitor*", el mecanismo de acceso al monitor garantiza que sólo un proceso o hilo activo esté en el monitor. Si hay un procedimiento en el monitor y otro trata de ingresar, este último se bloquea en la cola de entrada al monitor, y cuando un procedimiento abandona el monitor, selecciona el que está primero en la cola y lo desbloquea, si no hay ninguno el monitor queda libre.

Sincronización de un monitor: si un proceso está en el monitor y no obtiene el recurso no puede seguir su ejecución, por lo que se requiere un mecanismo que lo bloquee hasta que el recurso esté disponible y pueda ser desbloqueado; este mecanismo es implementado con variables de condición, estas son accesibles sólo desde el monitor, y con estas variables se realiza la gestión de sincronización. Las primitivas Wai y Signal bloquen y desbloquen, los procesos, en las colas asociadas a las variables de condición del monitor. Una alternativa, condición implícita, es eliminar Signal y especificar con Wai una condición o aserción en el monitor. Si esta condición es falsa el proceso se bloquea y cuando es verdadera se desbloquea.

Condición de sincronización: este concepto hace referencia a la situación en la cual es necesario coordinar la ejecución de procesos concurrentes, la cual sucede cuando un objeto compartido está en un estado inadecuado para ejecutar una operación determinada. Cualquier proceso que intente realizar esta operación sobre el objeto, debe retrasar su ejecución hasta que el estado del objeto cambie por las operaciones de otros procesos. Por lo tanto podemos definir la condición de sincronización como la propiedad requerida por un proceso para no atender un evento hasta que otro proceso haya finalizado una acción determinada. Es responsabilidad del programador del sistema solicitar al monitor el recurso y/o sincronización y devolver el recurso y/o comunicar la sincronización.

Variable de condición: son para detener la ejecución de un proceso que no puede proseguir su ejecución en forma segura. Cuando la condición no es satisfecha, el proceso es suspendido en una cola asociada con esta variable. Se dice que es una cola puesto que puede haber más de un proceso esperando por la condición. El proceso continua cuando satisface la condición, que es representada por una variable de condición. Se requiere de una variable de condición por cada condición que bloquea un proceso. Estas variables son locales al monitor y gestiona una cola por condición.

Política de colas, corresponde al orden en que los procesos son sacados de las colas de espera. Esta política, sobre las colas de condición, no tiene efecto sobre la lógica del programa, es decir sobre el funcionamiento con o sin colas de prioridad. Sólo impacta en el desempeño del monitor; por ejemplo, en el tiempo de respuesta, el tamaño de la cola, etc.

Clasificación de Monitores[277]

A priori podemos dividir a los monitores en dos grupos, en función de si la operación Signal es explícita o implícita. Dentro de estas dos categorías hay distintas sub-categorías según sea la semántica de la operación Signal.

Signal explícita

Para este caso, la clasificación está relacionada con las distintas prioridades de las colas de del monitor, que son las prioridades de: la cola de entrada (E_p), las colas de espera por condición (W_p) y las cola de Signal (S_P). Estos tres casos representan 13 distintas alternativas. Solo 6 de los 13 casos son de implementación práctica, esto es debido a inanición potencial y/o al aumento incontrolable de procesos bloqueados en las colas, cuando las variables de condición son verdaderas. Los 6 casos de interés son:

- $E_p = W_p = S_p$
- $E_p = W_p < S_p$,
- $E_p = S_p < W_p$
- $E_p < W_p = S_p$
- $E_p < W_p < S_p$
- $E_p < S_p < W_p$

En el caso de que dos colas tengan igual prioridad y se deba elegir por una, la elección es arbitraria. Con este enfoque, una tarea genera la señal para una condición, cada vez que la condición es verdadera; la comprobación es responsabilidad de la tarea. Este es un enfoque cauteloso para el control de concurrencia, donde las tareas señaladas y el señalizador no pueden hacer ninguna suposición sobre el orden de ejecución, incluso dentro del monitor (determinismo).

Por otra parte, hay que señalar que, las prioridades son relativas entre las colas y no de los procesos; esto permite gestionar la distribución de recursos, que son responsabilidad del monitor.

Signal automática
En este tipo de monitor el Wai está acompañado de una expresión condición, que computa variables y variables locales del monitor. Si se permite sólo las variables del monitor en la expresión condicional, el monitor es llamado un monitor de señal automática restringido.

Cuando se desbloquea un monitor Signal automática, en general, la búsqueda de la siguiente tarea a ejecutar puede ser costosa porque, en el peor de los casos, se trata de re-evaluación las expresiones condicionales de todas las tareas en espera. Desde una expresión condicional potencialmente puede depender de variables locales de una tarea, como los parámetros formales de una rutina de entrada de monitor, por lo que se accede al contexto local de una tarea para evaluar su expresión condicional. La forma más sencilla de lograr esto es despertar las tareas de la cola de espera, una por una, para que cada proceso evalue su expresión condicional. Si la evaluación de la condición es verdadera se procede; de lo contrario, de nuevo se bloquea en la cola de espera y permite que otra realice la evaluación. El problema con este enfoque es que se producen muchos cambios de contexto antes de proceder.

Una implementación alternativa es tener toda la copia de las variables locales de la tarea que espera, en un área de datos global de todo el contexto local necesaria para evaluar la expresión condicional y proporcionar un puntero al código para su prosecución, de modo que se puede acceder por cualquier tarea en espera.

Sin embargo, la creación de los datos necesarios, la búsqueda y evaluación de la lista es a la vez complicado y el tiempo de ejecución caro. No tenemos conocimiento de cualquier aplicación que tenga este enfoque.

Tanto en este aplicación y el más simple descripto anteriormente, el tiempo que se necesita para encontrar la siguiente tarea a ejecutar es determinado por el número de tareas en espera y el costo de re-evaluar las expresiones condicionales.

Los monitores de señal automática restringidos tienen aplicación mucho más eficiente. Puesto que todas las variables en las expresiones condicionales son variables del monitor, y por lo tanto no dependen del contexto de las tareas individuales. Las expresiones condicionales pueden ser evaluadas de manera eficiente por la tarea que está a punto de desbloquear el monitor. La eficiencia por señal puede mejorada si las expresiones condicionales representan condiciones distintas; si dos o más tareas están esperando la misma condición, sólo necesita ser evaluada una vez.

Dado que sólo las variables de monitorear están permitidas en una expresión condicional, el tiempo que se tarda en encontrar la siguiente tarea a ejecutar está determinado por el costo de re-evaluación de las expresiones condicionales. Por lo tanto, en comparación con los monitores de señal automática generales, existe el potencial de ahorro en la determinación de la siguiente tarea para ejecutar el tiempo de ejecución significativo.

El esquema de categorización del monitor es aplicable a los monitores de señal automática. Sin embargo, no hay colas de condición y si cola comunicador. Por lo tanto, cuando se desbloquea la siguiente tarea activa es, o bien una tarea de llamada o una tarea en la cola de espera, a la espera de su expresión condicional para evaluar a verdadero. Estas posibilidades son:

- $E_p < W_p$
- $E_p > W_p$

En este caso también se descarta la alternativa en que la cola de entrada tiene prioridad sobre una cola interna.

Conclusiones

Podemos pensar al PP como un monitor, donde la diferencia fundamental es que este no tiene la capacidad de suspender a los hilos/procesos cuando no se cumplen las condiciones.

Con respecto a las colas el PP puede programr las prioridades, con lo que podemos alcanzar todas las prioridades propuestas para los monitores.

Con respecto a la evaluación de las condiciones el PP realiza todas las evaluaciones en paralelo ya que la condiciones lógicas están expresadas en las en la RdP y los estados locales representan al estado de estas variables.

La exclusión mutua para "entrar al monitor" es cumplida por el PP, puesto que la manera de implementar la comunicación entre el PP y los procesos es a travéz de la cola de entrada y esto se realiza con el uso de un BUS.

Anexo D

RdP No Autónomas

En el trabajo de [35] se describen las RdP Autónomas, estas permiten un enfoque cualitativo, ahora bien las extensiones de las RdP permiten describir lo que ocurre, y además cúando y porqué sucede. Estas RdP modelan el sistema cuyos disparos son sincronizados con los eventos externos y / o cuyas evoluciones son dependientes del tiempo (cúando).

En una RdP sincronizada, una transición puede ser disparada sólo una vez en un instante dado. Mientras que en una "RdP sincronizada extendida ", más general, se basa en el hecho de que una transición que es q-sensibilizada puede ser disparada q veces en el mismo instante.

Ahora presentamos la interpretación de las RdP sincronizadas autónomas para el control de sistemas de eventos discretos. Estos modelos son tratados como casos especiales de RdP sincronizadas. Como consecuencia, todas las propiedades que se muestran para RdP sincronizadas son relevantes para estos modelos.

RdP sincronizadas

En una RdP autónoma, la transición puede ser disparada si está habilitada, pero no sabemos cuándo o porqué va a ser despedido. En una RdP sincronizada, un evento es asociado con cada transición, y el disparo de esta transición se producirá cuando:

- La transición está habilitada,
- Se produce el evento asociado.

Los eventos externos corresponden a un cambio en el estado del mundo externo (incluyendo el tiempo); por oposición, un cambio en el estado interno, modificando el marcado, se llama evento interno. Debemos destacar que la ocurrencia de los eventos no tiene duración.

Definición 38: RdP sincronizada

Una RdP sincronizada es una triple {R, E, Sync} tal que:

- R es un RdP marcada,
- E es un conjunto de eventos externos,
- Sync es una función del conjunto T a las transiciones de $R\ a\ E\ \cup \{e\}$ en el que e es el evento que ocurre siempre (es el elemento neutro del monoide E^*).

Principio

Sea un conjunto de eventos externos $E = \{E^1, E^2, \ldots\}$. La notación E^i corresponde con el "nombre" de un evento externo. La notación E_j se corresponde con el evento asociado con la transición T_j.

Comportamientos y conceptos básicos

Los tres ejemplos que se dan en la Figura 216 ilustran los principales conceptos de la RdP sincronizada. Los ejemplos más complicados que vamos a examinar (por ejemplo, el caso de un conflicto) tienen una interpretación que está relacionada con estos conceptos.

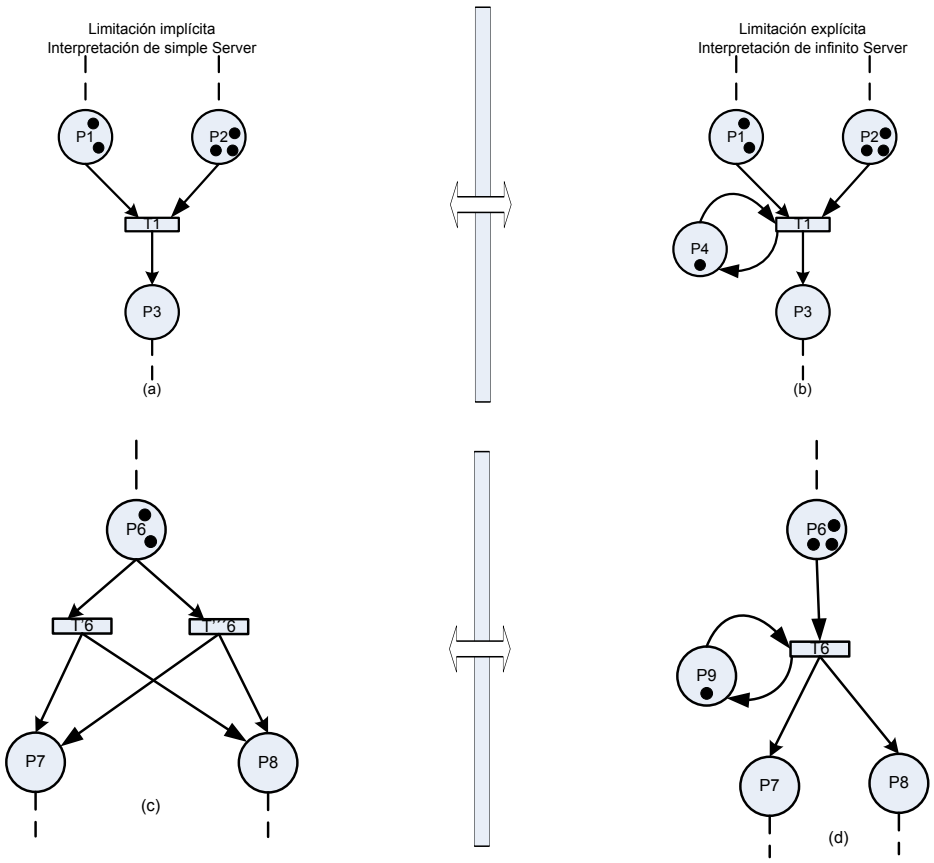


Figura 216: Semántica de simple server frente a semántica de multiple server

En la Figura 217 a, el evento externo E^3 está asociado con la transición T1. En esta figura, la transición T1 se dice que es receptiva al evento E^3, ya que está habilitada. Se convertirá en sensibilizada cuando se produce el evento E^3, por lo que será disparada de inmediato (ver el diagrama de tiempos en la Figura 217).

En la Figura 217 b, la transición T2 es receptiva al evento E^1, ya que está habilitada. Se dispara cuando se produce evento E^1. Por otro lado, la transición T3 no se dispara, a pesar de que se sincroniza con E^1, ya que no está sensibilizada cuando se produce E^1 (no es receptivo a este evento).

En la Figura 217 c, la transición T4 es receptiva al evento E^2, ya que está habilitada. De hecho, podemos observar que la transición T4 es 2-habilitada por lo que puede ser disparada dos veces. Por lo tanto, cuando se produce el evento E^2, se disparó dos veces en un instante atómico. Esto se ilustra en el diagrama de tiempos.

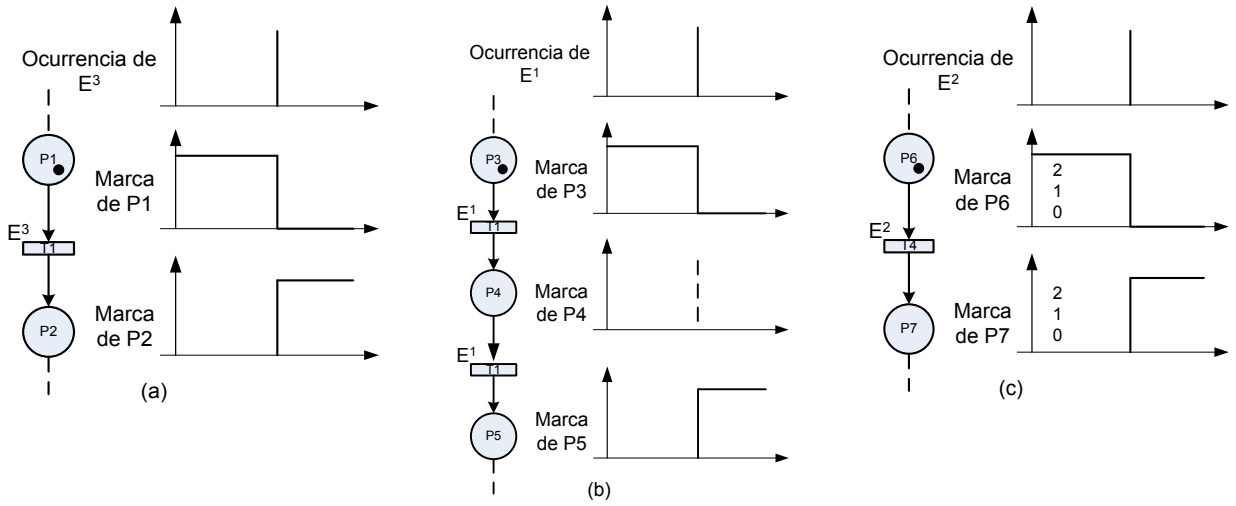


Figura 217: Disparo de una transición sincronizada

Observación

En una RdP autónoma, una transición podría ser calificada con indiferencia por el habilitado o disparo. Este no es el caso para un RdP sincronizada. Una transición está habilitada cuando cada uno de sus lugares de entrada contiene suficientes marcas (por lo menos un token para un RdP ordinaria). Si está habilitada, es disparada en la ocurrencia del evento asociado a la transición. Con la ocurrencia del evento el disparo es inmediato, con la posible excepción que exista un conflicto que no permite que todas las transiciones habilitadas sean disparadas.

La Figura 218 (a), se muestra una RdP sincronizada con un marcado inicial $m0 = (1, 0)$. Donde ll tiempo inicial está representado por una línea de ordenada de rayas. El conjunto de acontecimientos externos es $E = \{E^1, E^2\}$. La Figura 218 b muestra la evolución de las marcas cuando la secuencia de eventos es $Z = E^2E^1E^1E^2E^1$. Para el marcado inicial, sólo la transición T1 está habilitada. Esta transición es receptiva al evento E^1. La RdP sincronizada es por lo tanto sólo receptiva a este evento, lo que significa que la ocurrencia de cualquier otro evento no afectará el marcado. Puesto que, el primer evento de la secuencia Z, que es E^2, no altera nada. Tan pronto como se produce el evento E^1, T1 se dispara, lo que resulta en la marca (0, 1). La única transición habilitada, después del disparo de T1, es T2 que es receptiva al evento E^1. Cuando este evento se produce (después de la segunda ocurrencia de E^1 que no tuvo ningún efecto), T2 será disparado y así sucesivamente. El conjunto de posibles evoluciones está representada por la gráfica de marcas en la Figura 218 c. Por cada disparo de una transición, hemos indicado después de la barra / el evento cuya ocurrencia ha provocado el disparo.

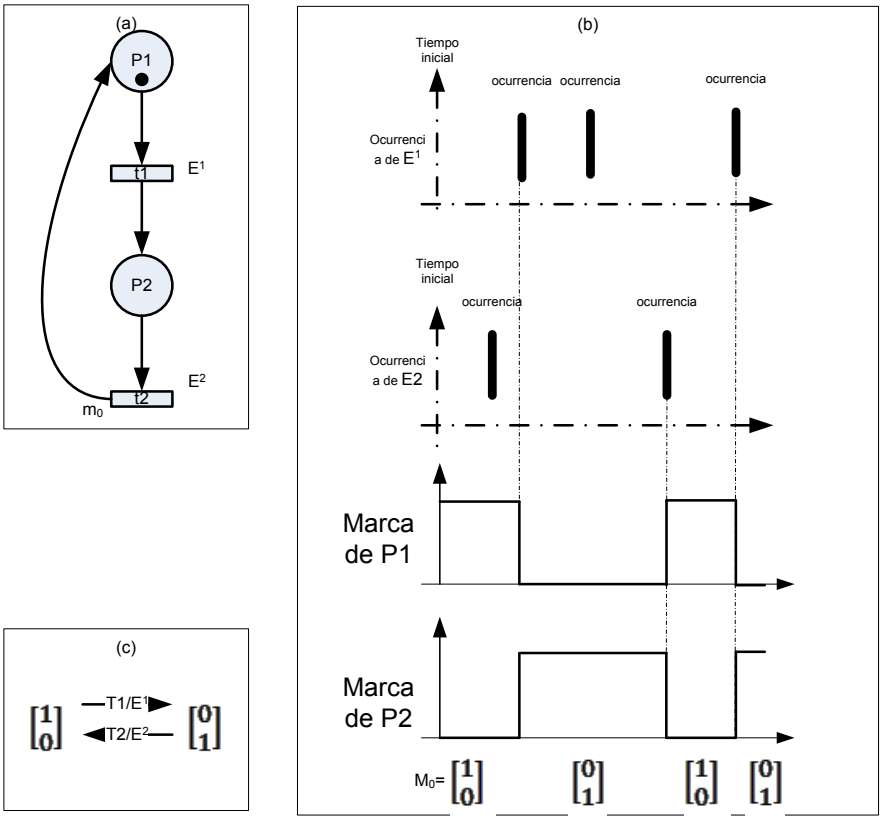


Figura 218: Ejemplo de funcionamiento de una RdP sincronizada

El mismo evento puede estar asociado con varias transiciones de un RdP sincronizada (en la Figura 217b ya se dió un ejemplo de esto). Consideremos la RdP sincronizada de la Figura 218. Para el marcado $M0 = (1, 0)$, la transición T1 está habilitada y es receptora al evento E^3. Cuando se produce este, T1 se dispara y se traduce en la marca (0, 1). Entonces la transición T2 se activa y es receptiva al evento E^3. Su disparo en la segunda aparición del E^3 , lo que nos lleva de nuevo a la marca inicial. Esta evolución se ilustra en la Figura 219 b, y la gráfica de las marcas se representa en la Figura 219 c.

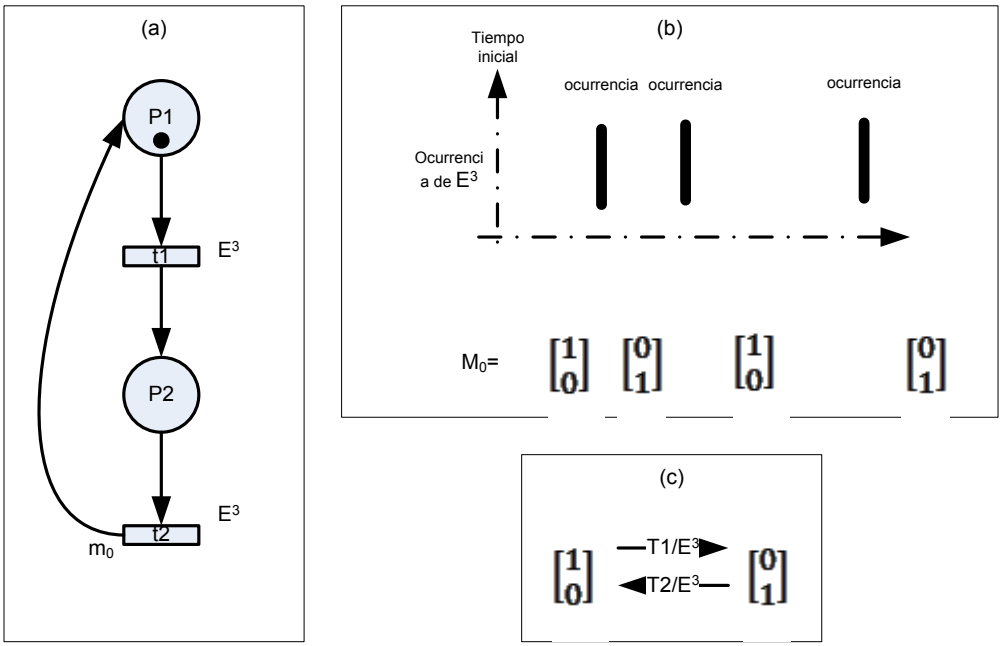


Figura 219: Un mismo evento asociado con una misma transición

Ahora introducimos un nuevo tipo de evento, que no es un evento externo. Este es el evento que ocurre siempre, que escribiremos como e.

En la Figura 220, el evento e se asocia con la transición T2. Esto significa que cuando la transición T2 está habilitada, será receptiva a este evento y por lo tanto se dispara de inmediato, ya que este evento está "ocurriendo siempre". Para el marcado inicial $m0 = (1, 0)$ de la Figura 220 a la RdP sincronizada es receptivo al evento E^3. Cuando se produce este caso, T1 se activa, lo que resulta en la marca (0, 1) para los que T2 está sensibilizado, y esta transición es receptiva al evento e, se dispara inmediatamente y volvemos a la marcar (1, 0). Luego esperamos a la siguiente aparición del E^3 con el fin de evolucionar de nuevo.

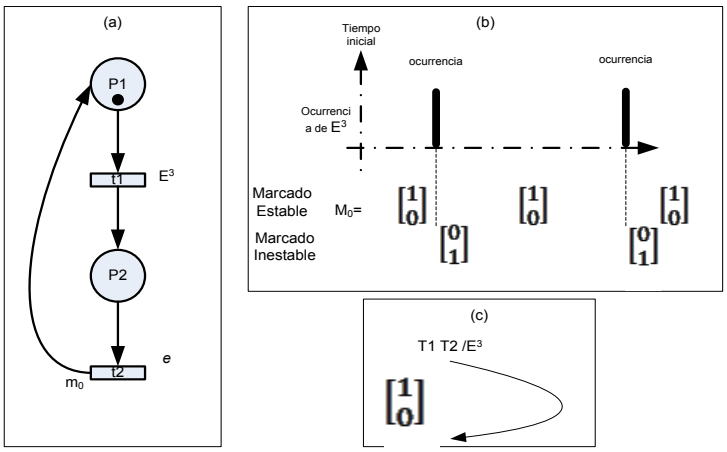


Figura 220: El evento *e* ocurre siempre

En este ejemplo vemos que el marcado (1, 0) es estable, mientras que la marca (0, 1) es inestable porque existe una transición que es receptiva a e, para esta marca. Por lo tanto, cuando la marca es $M0 = (1, 0)$ y el evento E^3 se produce, no sólo una transición se dispara, sino también la secuencia de transición T1T2. Se dice que se reiteró el disparo sobre ocurrencia del evento E^3.

Figura 220 b ilustra esta evolución y la Figura 220 c representa la gráfica de las marcas alcanzables. Observamos que este RdP sincronizada sólo tiene un marcado estable, mientras que la misma RdP sin sincronización tiene dos marcas alcanzables.

Transición inmediata
Introduzcamos ahora el concepto general de la transición inmediata.

Definición 39: IO Petri Net

En una RdP no autónoma, una transición inmediata es una transición que se dispara q veces tan pronto como es q-habilitada, para cualquier q> 0 (o posiblemente no suceda si hay un conflicto real entre esta y otro/otras transiciones).

De acuerdo con esta definición, si Tj es una transición inmediata, para cualquier marca m tal que q (Tj, m)> 0 es inestable. Para una RdP sincronizada, es equivalente a decir que una transición es inmediata o que se sincroniza con el suceso e.

Eventos Simultáneos
Sobre la base de la explicación anterior, ahora vamos a especificar cuando se produce un conflicto real. Primero denotamos por X un conjunto de eventos simultáneos. Por ejemplo, si $E^1, E^2, E^3\ y\ E^4$ son eventos independientes, ahora los pares independiente $E^5\ y\ E^6$, donde $E^5 = (E^1 + E^2)$ y $E^6 = (E^1 + E^3)$, son compatibles, es decir, pueden ocurrir simultáneamente; para el conjunto de eventos $E = \{E^1, E^2, E^3, E^4, E^5, E^6\}$, los siguientes conjuntos de eventos simultáneos se pueden obtener: $X^1 = \{E^1, E^5, E^6\}$, $X^2 = \{E^2, E^5\}$, $X^3 = \{E^3, E^6\}$, $X^4 = \{E^4\}$. Otro ejemplo: E^7 denotar que el evento ocurre después de cada minuto y E^8 que ocurre después de cada hora; es evidente que no son independientes, y para el conjunto de eventos $E = \{E^7, E^8\}$, de los conjuntos $X^5 = \{E^7\}$ y $X^6 = \{E^7, E^8\}$ se pueden obtener los eventos simultáneos.

Notación: Eventos y conjunto de eventos
Sea $X^j = \{E^1, E^2\}$ un conjunto de dos sucesos compatibles. Esto indica que el producto de dos eventos es un evento, es sinónimo de decir "el conjunto X^j de eventos se produce" o "el evento $E^1\ y\ E^2$ se produce. "Por lo tanto, cuando un conjunto de eventos contiene sólo un acontecimiento, no hay ambigüedad si decimos" el evento "en lugar de" el conjunto de los acontecimientos ", si escribimos $X = E^i$ en lugar de $X = \{E^i\}$, $X = e$ en lugar de $X = \{e\}$.

Definición 40: Conflicto real de eventos

Se produce un conflicto real cuando el conjunto de eventos X^h se producen simultáneos y si

1) el marcado de la RdP sincronizada es m, y
2) hay un conflicto general $K^G = < Pi, \{Tj\}, m >$ tal que todas las Transiciones en el conjunto {Tj} se sincronizan con los acontecimientos de X^h.

Una RdP sincronizada es tal que ya sea un evento externo o el evento que ocurre siempre e (es decir, un elemento de $\{e\}$) está asociado con cada transición. Una RdP sincronizada se dice que está totalmente sincronizada si ninguna de sus transiciones está asociada con el elemento e.

Secuencia de disparo Elemental

Varias transiciones pueden ser simultáneamente disparadas con la aparición de un conjunto de eventos simultáneos X (este conjunto X puede ser un subconjunto de E o el evento e). El disparo simultáneo de estas transiciones se denomina secuencia de disparo elemental (EFS); la notación para disparos simultáneos de una secuencia de disparo elemental se representa por una secuencia entre corchetes. Un vector característico se asocia con una secuencia de disparo elemental. Por ejemplo, si $s = [(T_1)^2\, T_2 T_4]$ es una secuencia de disparo primaria de una RdP que contiene cinco transiciones, su vector característico es $s = (2, 1, 0, 1, 0)$. Se denota por $T\,(X, m)$ el conjunto de transiciones receptivas de los eventos en X, para la marca **m**.

Definición 41: Secuencia de disparo elemental (EFS)

La secuencia S_k es una secuencia de disparo elemental (EFS) con respecto a un conjunto de eventos simultáneos X para una marca m, si cumple con los siguientes tres condiciones.

1) Todas las Transiciones en S_k pertenecen a $T\,(X, m)$.

2) $W\, s_k \leq m$.

3) No hay una secuencia s_h que cumple las condiciones 1 y 2 de manera que $s_h \gtrdot s_k$.

Observación

La condición 3, de la Definición anterior, es una generalización y una formalización del principio intuitivamente que se presenta en la Figura 217 c.

Las Figura 221 a, b, y c, son partes de RdP sincronizados en el que se supone que las otras transiciones no son receptivas de los eventos considerados.

La Figura 221 a, el marcado es M1 = (2, 3, 1,...), T1 es 2-habilitado y T2 es 1-habilitado. Para la ocurrencia del evento E1, ambas transiciones son sensibilizadas, por lo tanto, T (E1, M1) = {T1, T2}. Puesto que hay suficientes marcas para disparar estas transiciones en función de su grado de habilitación, es decir, no hay un conflicto real, para EFS que es $s1 = [(T_1)^2 T_2]$.

En la Figura 221 , al presentarse el evento E2, el conjunto de transiciones sensibilizadas es $T\,(E2, m2) = \{T_3, T_4\}$. La transición T_3 es 2-habilitada y T_4 es 3-habilitada. Dado que no hay suficientes marcas en P5, que permitan disparar estas transiciones en función de su grado de habilitación, es decir, hay un conflicto real, hay varias EFS posibles, a saber, S2 = [(T3) 2T4], S3 = [T3 (T4) 2], y S4 = [(T4) 3].

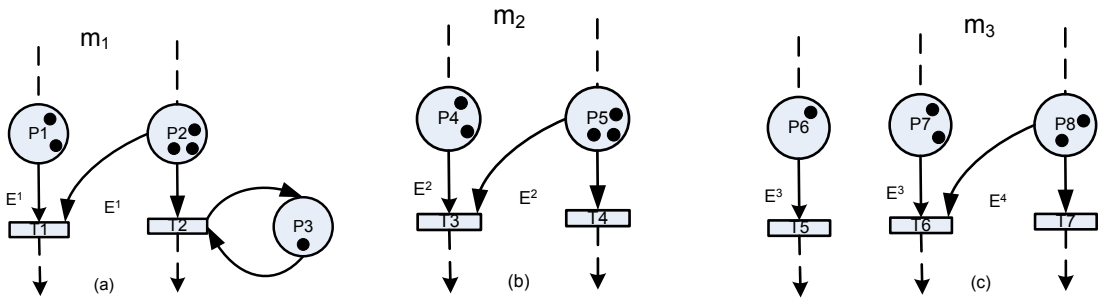


Figura 221: Secuencia elemental de disparos

Figura 221 c. Al presentarse el E3, el conjunto de transiciones sensibilizada es T (E3, m3) = {T5, T6}. No hay conflicto real: el único EFS es S5 = [T5 (T6) 2]. Sin embargo, si se producen acontecimientos E3 y E4 simultáneamente (lo cual es posible si no son independientes), existe un conflicto real. Para el conjunto de eventos simultáneos X1 = {E3, E4}, el conjunto de transiciones sensibilizada es T (X1, m 3) = {T5, T6, T7}. Los posibles EFS son S6 = [T5 (T6) 2], S7 = [T5T6T7] y S8 = [T5 (T7) 2]; transición T5 es en cada uno de EFS, ya que no está involucrado en el conflicto real entre T6 y T7.

En la Figura 222 vemos que son posibles distintos EFS para una marca dada m y una serie de eventos simultáneos X. Vemos, en la Figura 222 a, que m4 que es un marcado de la RdP sincronizada y nos preguntamos ¿Cuáles son los posibles EFS?, dada la serie de eventos simultáneos, por ejemplo X2 = {E1, E2}. Si con R1 denotamos el RdP inicial (Figura 222 a).

De acuerdo con la primera condición en la Definición 41, todas las transiciones en una EFS pertenecen a T (X2, M4) que es un subconjunto de las transiciones sincronizadas por E1 o E2; a continuación, todas las transiciones que no están sincronizados por E1 o E2, se cancelan, también sus arcos de entrada y salida. De acuerdo con la segunda condición en la Definición 41, las EFS dependen sólo de la incidencia de entrada de la matriz W; a continuación, se cancelan todos los arcos de una transición a un lugar (que se corresponde a W +).

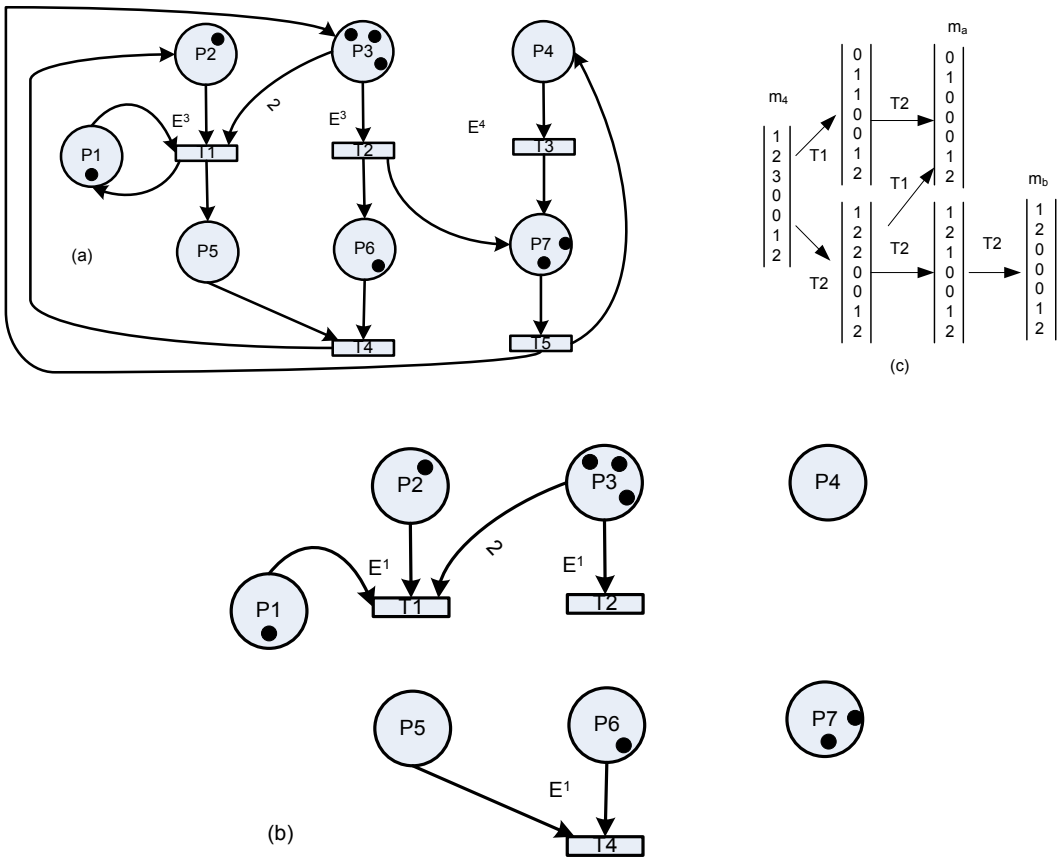


Figura 222: Secuencias elementales sensibilizadas, (a) RdP R1 con marca m, (b) RdP modificada R2 con X={E1,E2}, (c) grafo de marcas R2

Después de estas cancelaciones, con la RdP R2 de la Figura 222 se obtiene el diagrama de la Figura 220 b.
Ahora vamos a construir la gráfica de las marcas de la PN R2, ver Figura 220 c.

De acuerdo con la tercera condición en la Definición 41, una EFS corresponde a cada detención marcado en la Figura 220 c. Para nuestro ejemplo, dos puntos muertos se encuentran: ma correspondiente a Sa = [T1T2] y mb correspondientes a Sb = [(T2) 3].

El algoritmo ilustrado en la Figura 222 siempre converge en un número finito de pasos. Como cuestión de hecho, por la construcción, la matriz de incidencia de salida de la RdP R2 está vacía.

Por lo tanto, cada disparo transición m Tj m 'en R2 es tal que $m \geqslant m'$; Dado que el marcado de R2 inicial es finito, su gráfica de las marcas es necesariamente finita (y no contiene ningún circuito).

Si la gráfica de marcas de R2 contiene un único punto muerto, se dice que la secuencia de disparo elemental correspondiente a una EFS máxima.

Disparos iterados
Vamos a considerar la secuencia del despido iterado sobre la ocurrencia de un evento externo Ei. En un caso general, en lugar de un evento Ei, un conjunto de eventos simultáneos Xj puede ser considerado. Sin embargo, con el fin de simplificar la lectura, supondremos implícitamente que todos los eventos externos son independientes, por lo tanto, que dos de estos eventos no pueden ocurrir simultáneamente. Vamos a especificar cuando queremos mencionar el caso de eventos posiblemente simultáneos.

La iteración que produce un disparo sobre la ocurrencia de un evento externo Ei, consiste en los disparos de EFS sobre la aparición de Ei, posiblemente seguido por el disparo de uno de más de EFS sobre la aparición de e. Esto se ilustra en la Figura 223.

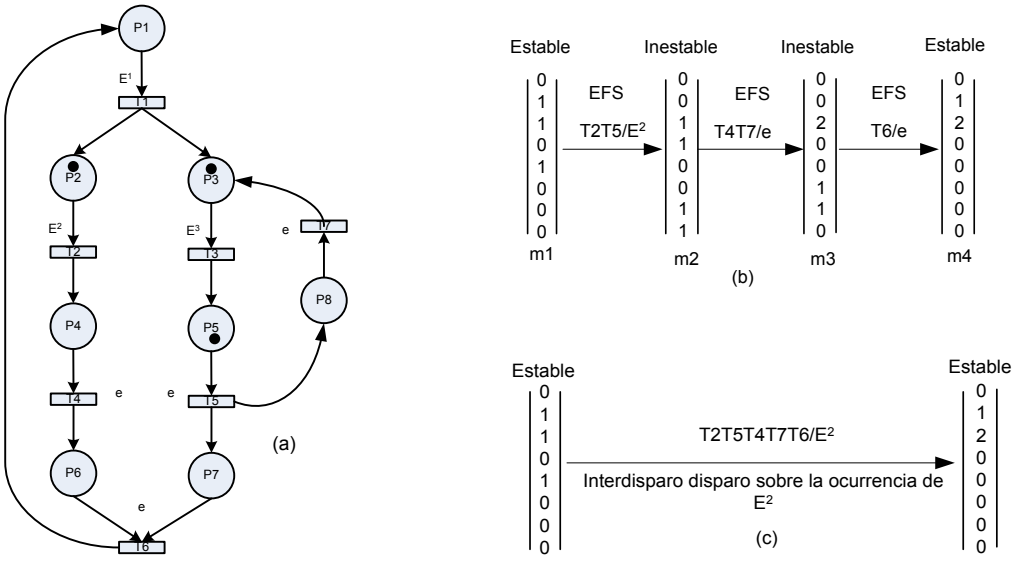


Figura 223: Iteración de disparos para la ocurrencia E^2

El marcado actual, M1 = (0, 1, 1, 0, 1, 0, 0, 0), la RdP sincronizada es receptiva a los dos eventos externos E2 (las transiciones T2 y T5 son receptivas al mismo evento) y E3 (es receptado por la transición T3). Como consecuencia de la ocurrencia del evento E2, la secuencia de disparo primaria [T2T5] se dispara, resultando así en la m2 marcado = (0, 0, 1, 1, 0, 0, 1, 1). Para este

marcado, la RdP sincronizada es receptiva al evento e (transiciones T4 y T7), lo que significa que esta marca es inestable. Una vez que las EFS [T4T7], han sido disparaas, se obtiene la marca m3 = (0, 0, 2, 0, 0, 1, 1, 0) para los que el PN sincronizada sigue siendo receptiva al evento e (T6 de transición). Una vez que la EFS T6 ha sido disparada, se obtiene el marcado m4, esta marca es estable.

La RdP sincronizada es ahora receptiva a los eventos externos E1 (transición T1) y E3 (T3 transición). La evolución de la marca se ha llevado a cabo antes de que ocurra uno de estos eventos. Así pues, los disparos iterados sobre la aparición de E2 se han traducido en un cambio de marcado M1 al marcado M4. La secuencia de disparo fue [T2T5] [T4T7] T6, es decir, T2 y T5 simultáneamente, a continuación, T4 y T7 de forma simultánea, a continuación, T6. Podemos escribir de m1 [T2T5] por los disparos [T4T7] T6 / E 2 a m4 o, porque no es sólo una posible cadena de EFSs: m1 E2 a m4.

El siguiente algoritmo explica cómo debe entenderse el comportamiento en tiempo real de un RdP sincronizada: la evolución se especifica cada vez que un nuevo evento externo se produce. Para una simulación, dada una secuencia de tiempo ordenada de eventos, una variante de este algoritmo se da en Observación.

Anexo E

Estudio de Recursos para la Implementación del PPcT

Para abordar problemas de mayor envergadura, se requiere ejecutar sistemas que cada vez más complejos, si estos sistemas están modelados con RdP el número de plazas y de transiciones será mayor. Para esto, es necesario un PP capaz de soportarlas, esto se traduce en un aumentando del número de recursos de la FPGA. Esencialmente estos recursos son el número de flips-flops (almacenar estados y registros) y el número de LUTs (la lógica del sistema, como multiplexores, sumadores, comparadores, etc).

Con el fin de analizar como varía el crecimiento del PPcT se sintetiza el PP para distintos tamaños de la matriz de incidencia en la plataforma Atlys de Digilent. Para esto se instanciaron distintos PP con matriz de incidencia de 2x2 (plazas y transiciones) hasta 64X64. El tiempo requerido para la síntesis del PP (en una PC con 4GB de RAM de 1333Mhz y una CPU i7 Q720) es aproximadamente de 2 horas y media.

Para configurar los elementos del PP, mostrada en la Figura 224, se tomó como constantes del PP los siguientes componentes:

```
parameter tamano_de_elementos                = 6;  //Tamano de los elementos de los arreglos, medido en bits
parameter tamano_cola                        = 5;  //Tamano de contador del contador para entrada de disparos
parameter tamano_vector_incremento_de_tiempo = 5;  // Cantidad de bits para expresar el numero de unidades en las que se in
parameter tamano_de_elementos_tiempo         = 32; //Tamano de los elementos de los arreglos de cuenta de tiempo, medido en
```

Figura 224: Parámetros para la síntesis del IP-Core procesador de RdP Temporales

La Tabla 133 muestra los resultados de los recursos requeridos en la síntesis para cada instancia de matriz.

Tabla 133: Resultados de los recursos requeridos en la síntesis del PP para distintas matrices I

Implementada con	Tamaño Matriz	Slice Registers	Slice LUTs	LUT Flip Flop pairs	Frecuencia maxima (MHz)
Spartan-6 FPGA- XC6SLX45	2x2	340	710	849	81.409
	3x3	556	1144	1383	76.512
	4x4	803	1529	1865	74.355
	5x5	1083	2012	2491	69.534
	6x6	1394	2507	3136	69.681
	7x7	1737	3134	3913	60.464
	8x8	2112	3657	4553	58.314
	9x9	2520	4302	5366	57.614
	10x10	2959	4909	6154	56.201
	11x11	3430	5578	7001	55.002
	12x12	3933	6363	7982	54.349
	13x13	4468	7105	8956	53.861
	14x14	5035	8050	10086	53.377
	15x15	5634	8792	11276	53.362
	16x16	6265	10355	13206	51.290
	17x17	6929	11772	14677	50.984
	18x18	7624	12904	16309	49.785
	19x19	8351	15238	19252	50.782

	20x20	9110	16142	20098	45.734
	21x21	9901	16734	22276	35.470
	22x22	10724	19775	24140	49.697
	23x23	11579	21331	26529	49.654
	24x24	12466	23093	28235	49.518
	25x25	13385	25560	31519	49.449
	26x26	14336	26703	32617	49.509
	27x27	15319	23372	29265	49.764
	28x28	16334	26325	32743	48.901
	29x29	17381	26529	33183	48.254
	30x30	18460	28856	36180	48.681
	31x31	19571	29782	37176	48.396
	32x32	20714	32028	39957	48.210
	33x33	21884	33766	42552	48.949
	34x34	23091	36000	47312	47.929
	35x35	24330	37602	47464	48.024
	36x36	25601	38919	49045	48.742
	37x37	26904	41646	53386	50.485
	38x38	28239	44030	56807	47.287
	39x39	29606	46071	59207	51.164
	40x40	31005	49217	63692	50.093
	41x41	32436	48963	63371	48.946
	42x42	33899	52476	68164	48.243
	43x43	35394	54766	70223	46.943
	44x44	36921	56307	73659	48.214
	45x45	38480	59861	77572	45.436
	46x46	40071	61509	80453	48.569
	47x47	41694	64354	84339	47.435
	48x48	43349	69837	91595	48.606
	49x49	45036	71674	93926	48.386
	50x50	46755	73384	98026	45.640
	51x51	48506	76243	100939	47.596
	52x52	50289	79203	104987	47.516
	53x53	52104	82084	107585	47.533
	54x54	53951	84912	112418	46.278
Spartan-6 FPGA-XC6SLX150	55x55	55830	88248	117342	46.336
	56x56	57741	90863	119567	43.844
	57x57	59684	94581	126849	46.290
	58x58	61659	96429	128305	43.782
	59x59	63666	100013	135937	42.375
	60x60	65705	100192	134827	41.866
	61x61	67776	105633	141786	43.158
	62x62	69879	108751	146562	44.117
	63x63	72014	113512	153119	40.746
	64x64	74181	117293	154590	43.702

Con esta Tabla 133 se ha obtenido el gráfico de la Figura 224 , esto facilita realizar una comparación entre los Flip-Flops y LUTs utilizados para cada instancia de la matriz.

Es decir que con esta placa, se puede sintetizar un procesador de RdP como máximo de 29 plazas y 29 transiciones (29x29) debido a que se consumen todas las LUTs disponibles. Por otro lado, asumiendo que se contará con un número infinito de LUTs, se podría sintetizar un procesador de hasta 54 plazas y 54 Transiciones (54x54).

Hay que notar que se usan muchas más LUTs dentro de las celdas lógicas de la FPGA que los Flip-Flops que estas poseen, haciendo que del total de Celdas utilizadas, aproximadamente el 50% de ellas tengan su Flip-Flop sin utilizar, ocupando solamente las LUTs. Trabajando en reducir la lógica necesaria para el procesador se podría mejorar este aspecto, produciendo un mejor aprovechamiento de los recursos disponibles.

En el siguiente gráfico, se puede apreciar el porcentaje de Celdas que ocupan su LUT, pero no su Flip-Flop.

PPcT jerárquico

En este apartado se sintetiza un HPP con varios PP simples y pequeños trabajando juntos para conformar un sistema con más plazas y transiciones. Esto facilita la optimización de los recursos de hardware a la vez que aumenta el tamaño de la matriz. Por ejemplo, si se desea implementar un procesador con 64x64 (plazas x transiciones) es posible resolver el sistema con 8 procesadores de 8x8 en una configuración jerárquica. Otras alternativas serian un HPP con 16 PP de 4x4, o 4 PP de 16x16, etc.

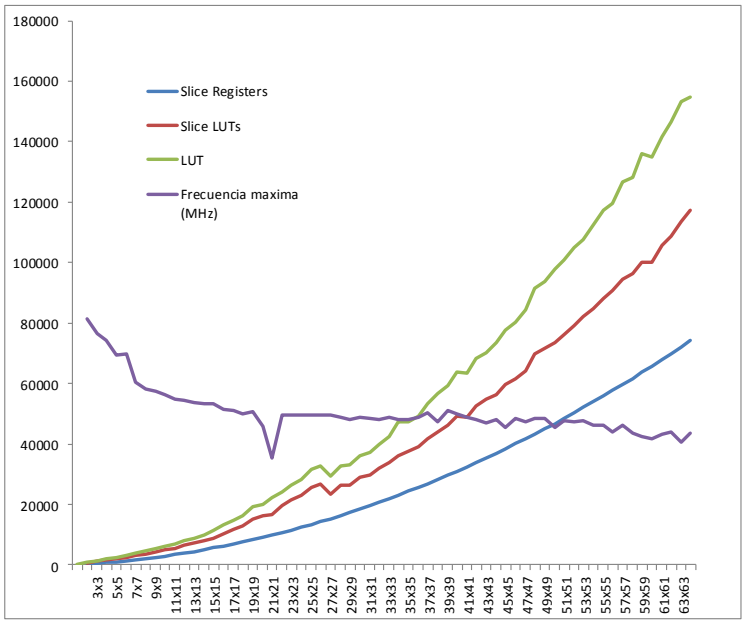


Figura 225: Resultado de síntesis del IP-Core procesador de RdP Temporales

Para analizar la propuesta se ha construido la Tabla 134, donde la columna “procesadores” expresa la cantidad de PP (todos iguales) con la cantidad de plazas x transiciones de cada uno.

Mientras que los recursos obtenidos de cada síntesis han sido expresados en valores absolutos y en porcentaje, tomando como base de 100% los recursos que utiliza el procesador de 64x64.

Tabla 134: Disminución de recursos con la división del IP-Core procesador de RdP Temporales

Procesadores	Flip-Flops	Porcentaje de Flip-Flops	LUTs	% LUTs	Registros	Porcentaje de Registros
1 - 64x64	74181	100%	117293	100%	4096	100%
2 - 32x32	41428	55,85%	64056	54,61%	2048	50%
4 - 16x16	25060	33,78%	41420	35,31%	1024	25%
8 - 8x8	16896	22,78%	29256	24,94%	512	12,50%
16 - 4x4	12848	17,32%	24464	20,86%	256	6,25%
32 - 2x2	10880	14,67%	22720	19,37%	128	3,13%

Graficando los valores absolutos la Tabla 134 se obtiene la Figura 225, que muestra la disminución de recursos dividiendo el PP original para obtener el HPP equivalente, todo para RdPcT.

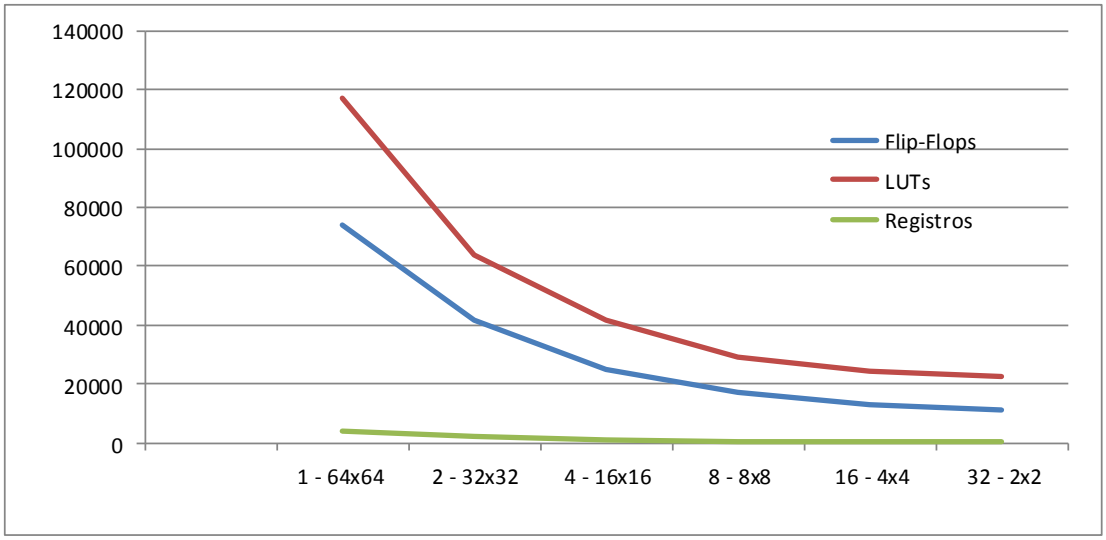


Figura 226: Disminución de recursos con la división del IP-Core procesador de RdP Temporales

Mientras que la gráfica en porcentajes ha sido realizada en la figura.

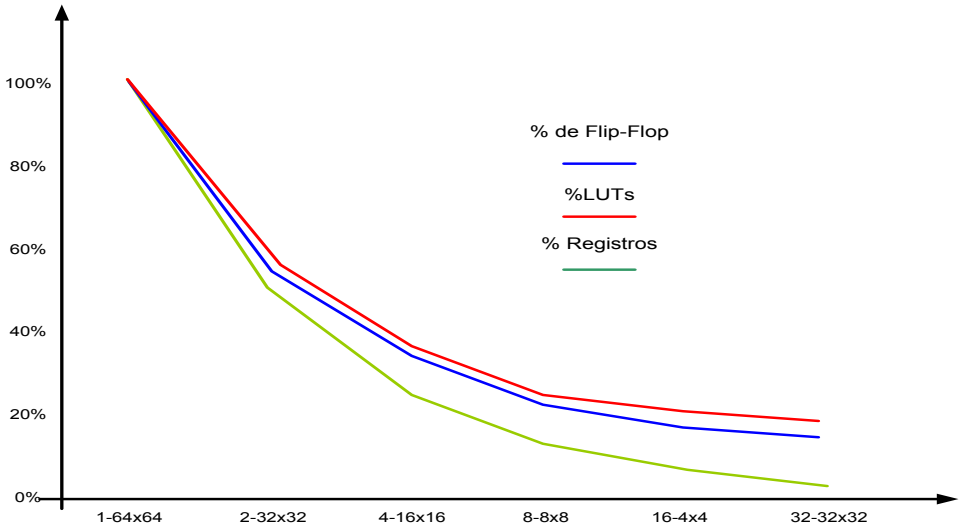


Figura 227: Recursos del HPP dividiendo el PPcT de 64x64 en porcentajes

Se ha construido, con una Spartan-6-XC6SLX150, un HPP con 16x16x16 y otro de 4x32x32.

Componentes de Hardware sintetizados

Recursos para sisntetizar los sumadores requeridos por la matriz de incidencia
Para esto se genera un sumador por cada elemento de la matriz de incidencia. El sumador tiene los bits necesarios para soportar los elementos de la matriz de incidencia y el marcado. Para obtener la matriz de próximos estados posibles se les suma a cada columna de la matriz de incidencia el vector de marcado (traspuesto), ver la línea 393 de la Figura 228.

```
for (filas=0 ; filas<cant_plazas ; filas=filas+1) // Recorro filas
   begin
      resultado[filas][columnas] = marcado[filas] + matriz_incidencia[filas][columnas]; //Ecuacion
      // Verificacion habilitacion por signo - Ecuacion de matriz
         sign_matrix [columnas][filas] = resultado[filas][columnas][tamano_de_elementos-1];
```

Figura 228: Sumadores para la matriz próximos estados

El resultado obtenido es que los sumadores más los multiplexores son el 11,56% de las LUTs utilizadas en el sistema.

Recursos para sisntetizar los sumadores de las colas de entrada y salida
Para esto se genera un sumador por cada transición y por cada una de las colas. La cantidad de bits de sumador es igual al tamaño de los contadores de las colas de entrada y salida. En cada disparo (evento asociado con la transición) el contador se incrementa en 1. Luego cuando la transición es ejecutada se disminuye en 1 y se aumenta el contador de la cola de salida. Esto es mostrado en la línea 502 (colas de entrada) y 538 (colas de salida) de la Figura 229.

```
for (index_entrada=0 ; index_entrada<cant_transiciones ; index_entrada=index_entrada+1)
   begin
      //Incremento cola entrada
         if ({disparo_entrada_incrementa[index_entrada] , disparo_entrada_decrementa[index_entrada]} == 2'b10) //Debe incrementar
            begin
               counter_cola_disparos_entrada[index_entrada] <= counter_cola_disparos_entrada[index_entrada] + 1'b1;
            end //FIN if(disparo_entrante[index_entrada] == 1'b1)
```

Figura 229: Sumadora de las colas de entrada y salida

El resultado obtenido es que los sumadores más los multiplexores son el 2,66% de las LUTs usados en el sistema.

Recursos para sisntetizar los restadores de las colas de entrada y salida

Para esto se genera 1 restador por cada transición y por cada una de las colas. La cantidad de bits del restador es igual a los que tienen las colas de entrada y salida. Con la elección de la transición a ser disparada se disminuye en uno el contador. También cuando se lee el evento en la cola de salida se disminuye en uno el contador de la cola de salida. Esto se muestra en las línea 507 (cola de entrada) y 543 (cola de salida).

```
//Decremento cola entrada
   else if ({disparo_entrada_incrementa[index_entrada] , disparo_entrada_decrementa[index_entrada]} == 2'b01)
      begin
         counter_cola_disparos_entrada[index_entrada] <= counter_cola_disparos_entrada[index_entrada] - 1'b1;
      end //FIN if(disparo_dec_entrante[index_entrada] == 1'b1)

//Decremento cola salida
   else if ({disparo_salida_incrementa[index_salida] , disparo_salida_decrementa[index_salida]} == 2'b01)
      begin
         counter_cola_disparos_salida[index_salida] <= counter_cola_disparos_salida[index_salida] - 1'b1;
      end //FIN if(disparo_salida_decrementa[index_salida] == 1'b1)
```

Figura 230: Restadores de colas de entrada y salida

Los resultados obtenidos para los restadores más los multiplexores necesarios son el 2,66% de las LUTs del sistema.

Recursos para sintetizar los sumadores que actualizan las marcas temporales

Para esto se genera 1 sumador por cada transición. La cantidad de bits de contador es igual a la de los elementos temporales. Este sumador, en cada ciclo, actualiza las marcas temporales, incrementando los contadores de tiempo de cada transición. Se trata de un sumador saturado, esto se utiliza con un comparador de mayor o igual. La Figura 231 muestra en las líneas de código 609 y 615 como se describen los sumadores.

```
if (t_sensibilizadas[index2] == 1'b1) //Transicion sensibilizada
   begin
      if (vector_tiempo_sensibilizacion[index2]+vector_incremento_de_tiempo[index2] >= {tamano_de_elementos_tiempo{1'b1}} )
         begin
            vector_tiempo_sensibilizacion[index2] <= {tamano_de_elementos_tiempo{1'b1}};
         end
      else
         begin
            vector_tiempo_sensibilizacion[index2] <= vector_tiempo_sensibilizacion[index2]+vector_incremento_de_tiempo[index2];
         end
   end
```

Figura 231: Código de los sumadores que actualizan las marcas temporales

Los recursos necesarios para implementar los sumadores más los multiplexores son el 9,61% de las LUTs del sistema.

Recursos necesarios para sintetizar los comparadores de cotas de plazas

Para esto se genera 1 comparador de mayor por cada elemento de la matriz de incidencia. El número de bits del comparador es el número de bits de los elementos de la matriz de incidencia y el vector de marcado. Para esto se genera una matriz binaria de habilitación según las cotas. El código de la comparación se realiza en la línea 399 del código de la Figura 232.

```
// Verificacion de habilitacion por cotas - Creacion de matriz
   if (resultado[filas][columnas][tamano_de_elementos-1]==1'b0 && resultado[filas][columnas] > cotas_plazas[filas])
      begin
         limit_matrix [columnas][filas] = 1'b0; // Cota SI superada
      end
   else  limit matrix [columnas][filas] = 1'b1; // Cota NO superada
```

Figura 232: Implementación de los comparadores para la cota de plaza

Los recursos necesarios para implementar los comparadores más los Multiplexores son el 3,64% de las LUTs del sistema.

Recursos necesarios para sintetizar los comparadores para el menor tiempo de disparo
Para esto se genera un comparador menor o igual por cada transición. Los bits necesarios para el comprador son los elementos de tiempo. Cuando se cumple este tiempo mínimo, recién ahí la transición queda habilitada para ser disparada. El código se puede ver en la Figura 233, línea 434.

```
/*REDES TEMPORALES*/
//Determinacion del vector de habilitacion de transiciones segun las "marcas temporales"
   for (k=0 ; k<cant_transiciones ; k=k+1) //Recorro transiciones
      begin
         if (vector_tiempo_sensibilizacion[k]>=vector_EFT[k] && vector_tiempo_sensibilizacion[k]<=vector_LFT[k]) t_enable_time[k] = 1'b1;
         else t_enable_time[k] = 1'b0;
      end
```

Figura 233: Código de descripción para los comparadores del menor tiempo

Los recursos necesarios para los comparadores más los multiplexores son el 17,14% de las LUTs del sistema.

Recursos necesarios para sintetizar los comparadores para el mayor tiempo de disparo
Para esto se genera un comparador menor o igual por cada transición. Los bits necesarios son igual a los del comparador de los elementos de tiempo. Cuando se cumple el tiempo máximo la transición queda definitivamente habilitada para disparar. La figura muestra el código del comparador en la línea 434.

```
/*REDES TEMPORALES*/
//Determinacion del vector de habilitacion de transiciones segun las "marcas temporales"
   for (k=0 ; k<cant_transiciones ; k=k+1) //Recorro transiciones
      begin
         if (vector_tiempo_sensibilizacion[k]>=vector_EFT[k] && vector_tiempo_sensibilizacion[k]<=vector_LFT[k]) t_enable_time[k] = 1'b1;
         else t_enable_time[k] = 1'b0;
      end
```

Figura 234 : Código de los comparadores de mayor tiempo

Los recursos necesarios para los comparadores más los multiplexores son el 17,14% de las LUTs del sistema.

Recursos necesarios para sintetizar los comparadores de señal de red activa
Para esto se genera 1 comparador de mayor por cada transición. Los bits necesarios son igual a los del comparador de los elementos de tiempo. Las comparaciones se realizan en la línea 454 de la Figura 235.

```
/***************************SEÑAN DE RED ACTIVA*********************/

   reg [cant_transiciones-1:0] red_activa_reg;
   wire red_activa;
   integer indice_activa;
   always @(*)
      begin
         for (indice_activa=0 ; indice_activa<cant_transiciones ; indice_activa=indice_activa+1)
            begin
               if (vector_tiempo_sensibilizacion[indice_activa] < vector_LFT[indice_activa]) red_activa_reg[indice_activa] = 1'b1;
               else red_activa_reg[indice_activa] = 1'b0;
            end
      end

   assign red_activa = &red_activa_reg;
```

Figura 235: Código de los comparadores para red activa

Los recursos necesarios para los comparadores más los multiplexores son el 8,44% de las LUTs del sistema.

Recursos necesarios para sintetizar los comparadores que actualizan las marcas temporales
Para esto se genera 1 comparador menor o igual por cada transición. Los bits necesarios para el comprador son igual al tamaño de los elementos de tiempo. La Figura 236 muestra donde se realiza el comparador, linea de código 609.

```
//Actualizo marcas temporales
   if (t_sensibilizadas[index2] == 1'b0) vector_tiempo_sensibilizacion[index2] <= {tamano_de_elementos_tiempo{1'b0}};
   if (t_sensibilizadas[index2] == 1'b1) //Transicion sensibilizada
      begin
         if (vector_tiempo_sensibilizacion[index2]+vector_incremento_de_tiempo[index2] >= {tamano_de_elementos_tiempo{1'b1}} )
            begin
               vector_tiempo_sensibilizacion[index2] <= {tamano_de_elementos_tiempo{1'b1}};
            end
         else
            begin
               vector_tiempo_sensibilizacion[index2] <= vector_tiempo_sensibilizacion[index2]+vector_incremento_de_tiempo[index2];
            end
      end
```

Figura 236: Código de los comparadores para actualizar las marcas temporales

Los recursos necesarios para el comparadores más los Multiplexores son el 5,54% de las LUTs del sistema.

Análisis de elementos de Hardware

El análisis previo se ha realizado sobre un PPcT con las características mostradas en la Figura 237.

```
parameter cant_plazas                      = 3;  //Cantidad de plazas de la red de Petri modelada
parameter cant_transiciones                = 2;  //Cantidad de transiciones de la red de Petri modelada
parameter bits_cant_transiciones           = 1;  //Numero de Bits necesarios para representar el numero cant_transiciones
parameter tamano_de_elementos              = 6;  //Tamano de los elementos de los arreglos, medido en bits
parameter tamano_cola                      = 5;  //Tamano de contador del contador para entrada de disparos
parameter tamano_vector_incremento_de_tiempo = 5;  // Cantidad de bits para expresar el numero de unidades en las que se i
parameter tamano_de_elementos_tiempo       = 32; //Tamano de los elementos de los arreglos de cuenta de tiempo, medido e
```

Figura 237: Parámetros del IP-Core sintetizado para obtener los recursos de implementación

Los valores des LUTs necesarios son muy dependientes de la configuración mostrada en la Figura 237.

La Tabla 135 muestra los resultados obtenidos. Resume los elementos medidos en porcentaje de LUTs, en la medición se incluyen los multiplexores (que manejan estos elementos), el enrutamiento de los datos, los sumadores, restadores y comparadores.

Tabla 135: Distribución de recursos para la implementación del PPcT en un IP-Core

Hardware	LUTs utilizadas en %
Sumadores para la matriz resultados	11,56%
Sumadores para las colas de entrada y salida de Transiciones	2,66%
Restadores para las colas de entrada y salida de Transiciones	2,66%
Sumadores para actualizar las marcas temporales	9,61%
Comparadores de cotas de plazas	3,64%
Comparadores para el menor tiempo de disparo permitido	17,14%
Comparadores para el mayor tiempo de disparo permitido	17,14%
Comparadores para la señal de red activa	8,44%
Comparadores para actualizar las marcas temporales	5,54%
TOTAL	78,39%

La totalización de los elementos usados en el análisis es el 78,39% de las LUTs utilizados en la síntesis del PPcT. Este análisis es la base para realizar el refinamiento que conduzca a la reducción de recursos necesarios en la implementación del PPcT.

Anexo F

Arquitectura del PP para Brazos con Peso Uno (PPpU)

La arquitectura general del PPpU es esencialmente la presentada en el Capítulo 4, la diferencia principal radica en la matriz de incidencia.

Para este caso se aprovecha que los elementos de esta matriz tienen ceros, unos, menos uno lo que demanda menos recursos.

Mientras que, la comunicación con los procesos se realiza con eventos, haciendo uso de las colas ya empleadas en el PP, por lo que las colas y su programación son las mismas. También, el bus de comunicaciones es el mismo que implementa el protocolo AXI.

En la Figura 238, se muestra el diagrama de la arquitectura modificada (Figura 57) de interconexión del PPpU. En el área de datos se ha modificado la matriz de incidencia ahora se requiere $I^+ e\ I^-$. Mientras que, en el área de cálculo se ha modificado el módulo de cálculo del próximo estado. En la Figura 238se ha rotulado las áreas modificada con "Módulo modificado".

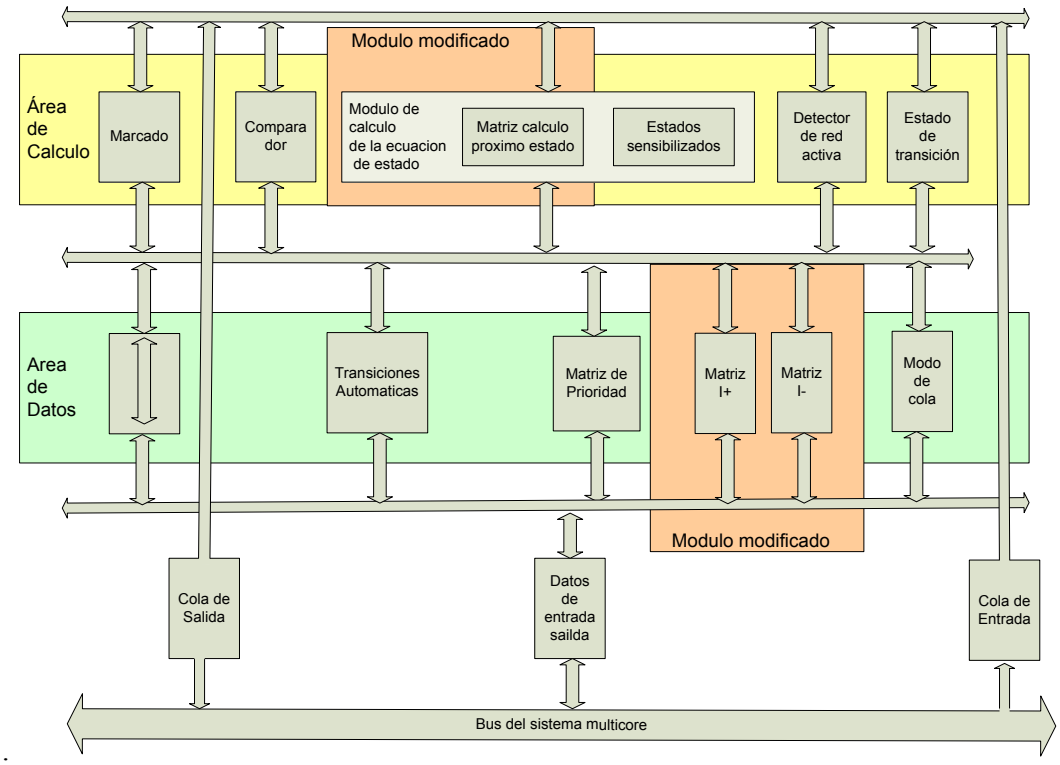


Figura 238: Arquitectura del PPpU

Algoritmo para la ejecución de una RdPnA con arcos de peso uno

El algoritmo es el esencialmente el mismo que se empleó en el PP, la diferencia radica en que no es necesaria la suma de las plazas con la columna de transición o la resta cuando se realiza el disparo, ahora solo se requiere un incremento en uno (en lugar de la suma) o un decremento en

menos uno (en lugar de la resta). Mientras que para detectar si es posible el disparo es necesario una xor (en vez de una comparación entre enteros).

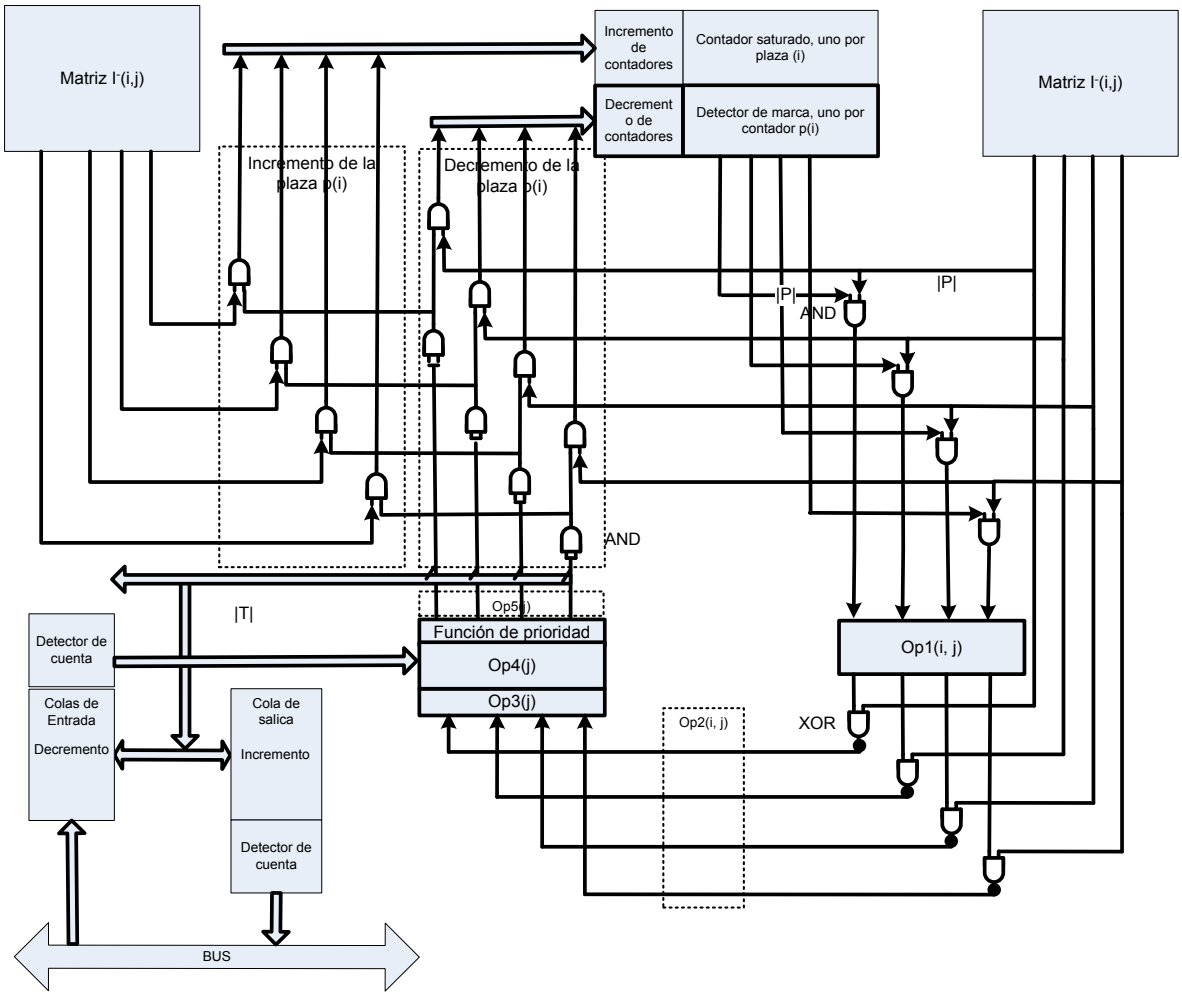


Figura 239: Diagrama en bloque de la lógica del circuito del PPpU

Las funciones booleanas que se requieren para los cálculos intermedios son:

- $op1(i,j) = I^{-}(i,j)\ and\ \ detector\ de\ marca\ de\ \ plaza\ p(i), \forall\ i,j$. Es una matriz binaria de dimensión |T|x|P|.
- $op2(i,j) = not(I^{-}(i,j) xor\ op1(i,j), \forall\ i,j$. Es una matriz binaria de dimensión |T|x|P|.
- $op3(i) = \bigwedge_{\forall j} op2(i,j)$. Es un vector binario de dimensión |T|.
- $op4(i) = \ op3(i)\ and\ detector\ de\ cuenta\ de\ cola\ entrada\ (i)$. Es un vector binario de dimensión |T|
- $op5(i) = función\ de\ prioridad\ (op4(i))$. Es el vector de disparo y tiene solo una componente con un uno, que es la de más prioridad.
- $Decremento\ de\ la\ plaza\ p(j) = op5(i)\ and\ I^{-}(i,j)$. Si es uno se disminuye en uno el contador que representa la plaza j.
- $Incremento\ de\ la\ plaza\ p(j) = op5(i)\ and\ I^{+}(i,j)$. Si es uno se incrementa en uno el contador que representa la plaza j.
- $Incremento\ de\ las\ cola\ de\ salida\ se\ realiza\ para\ los\ uno\ del\ vector\ op4(i)$
- $Decremento\ de\ las\ cola\ de\ entrada\ se\ realiza\ para\ los\ uno\ del\ vector\ op4(i)$

La Figura 239, muestra un diagrama en bloque de la implementación del algoritmo ejecutado por el PPpU. Este determina que transiciones se disparan y el nuevo estado.

La implementación y modos de programación de las colas son los mismos que los expuetos en los Capítulos 4 y 5, ya que no se ha cambiado para el PPuP.

Recursos para la implementación del PPuP

Esencialmente los recursos son el número de flips-flops (almacenar estados y registros) y el número de LUTs (la lógica del sistema, como multiplexores, sumadores, comparadores, etc).

Con el fin de analizar como varía el crecimiento del PPuP se sintetiza para distintos tamaños de la matriz de incidencia en la plataforma Atlys de Digilent. Para esto se instanciaron distintos PP con matriz de incidencia de 2x2 (plazas y transiciones) hasta 64X64.

La Tabla 136 muestra los resultados de los recursos requeridos en la síntesis para cada instancia de matriz.

Tabla 136: Resultados de los recursos requeridos en la síntesis del PPuP para distintas matrices I

Tamaño Matriz	Slice LUTs	Slice Registers	LUT Flip Flop pairs
2x2	102	227	255
3x3	167	366	415
4x4	241	489	560
5x5	325	644	747
6x6	418	802	941
7x7	521	1003	1174
8x8	634	1170	1366
9x9	756	1377	1610
10x10	888	1571	1846
11x11	1029	1785	2100
12x12	1180	2036	2395
13x13	1340	2274	2687
14x14	1511	2576	3026
15x15	1690	2813	3383
16x16	1880	3314	3962
17x17	2079	3767	4403
18x18	2287	4129	4893
19x19	2505	4876	5776
20x20	2733	5165	6029
21x21	2970	5355	6683
22x22	3217	6328	7242
23x23	3474	6826	7959
24x24	3740	7390	8471
25x25	4016	8179	9456
26x26	4301	8545	9785

27x27	4596	7479	8780
28x28	4900	8424	9823
29x29	5214	8489	9955
30x30	5538	9234	10854
31x31	5871	9530	11153
32x32	6214	10249	11987
33x33	6565	10805	12766
34x34	6927	11520	14194
35x35	7299	12033	14239
36x36	7680	12454	14714
37x37	8071	13327	16016
38x38	8472	14090	17042
39x39	8882	14743	17762
40x40	9302	15749	19108
41x41	9731	15668	19011
42x42	10170	16792	20449
43x43	10618	17525	21067
44x44	11076	18018	22098
45x45	11544	19156	23272
46x46	12021	19683	24136
47x47	12508	20593	25302
48x48	13005	22348	27479
49x49	13511	22936	28178
50x50	14027	23483	29408
51x51	14552	24398	30282
52x52	15087	25345	31496
53x53	15631	26267	32276
54x54	16185	27172	33725
55x55	16749	28239	35203
56x56	17322	29076	35870
57x57	17905	30266	38055
58x58	18498	30857	38492
59x59	19100	32004	40781
60x60	19712	32061	40448
61x61	20333	33803	42536
62x62	20964	34800	43969
63x63	21604	36324	45936
64x64	22254	37534	46377

Con esta Tabla 136 se ha obtenido el gráfico de la Figura 240, también se han graficado los resultados obtenidos para el PPcT, esto facilita la comparación y podemos ver que ahora es posible realizar procesadores con 50% más de plazas y transiciones.

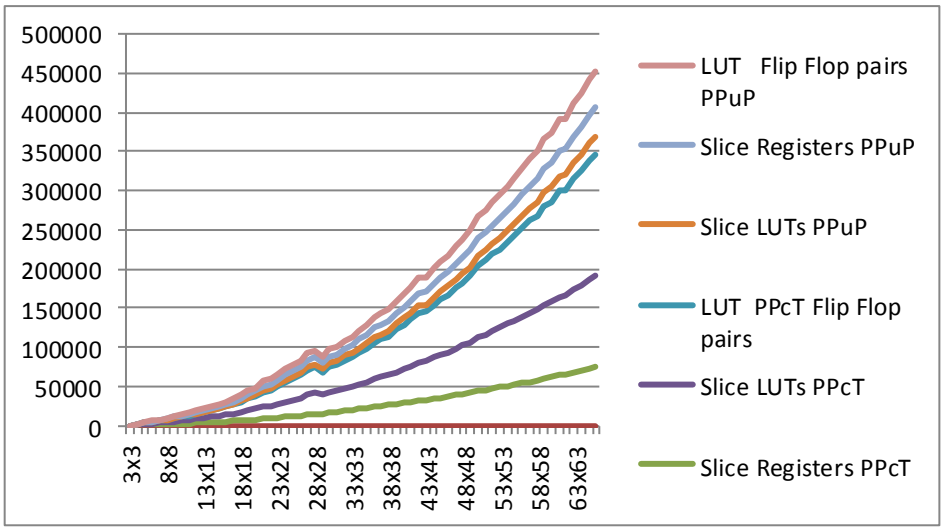


Figura 240: Curva de comparación entre recursos necesarios parz el PPcT y el PPuP

Anexo G

Acrónimos

AMP	Multi-core asimétricos
AMS	Sistemas de fabricación automatizado
AP	Transiciones de actividades periódicas
AS	Transiciones de sincronización
ASIC	Aplication-specific integrated circuit
AVG	Vehículos guiados automáticamente
BF	Fuerza bruta
CCIPN	Controlador de CPN interpretadas
CCM	Control Común de Memoria
CMOS	Semiconductor complementario de óxido metálico
CMPs	Comun Memory Processor
CODE	Transición que se corresponde con un código ejecutable
CPI	Ciclos por instrucción
CPN	Rdp coloreadas
CPN-Tool	Simulador de rdp coloreadas
DB	Base de datos
DES	Discrete-event simulation
DRP	Representación de plaza dinámica
DT	Transición diferida
EF	Counter Earliest Firing Time (ppct)
EFS	Secuencia de disparo elemental
EFT	Vector de Earliest Firing Time (rdpct)
ET	Transición habilitada
ETC	Controlador de tiempo de ejecución
EXQ	Cola de ejecución
FCT	Tabla de condición de disparo
FFT	Transformada rápida de Fourier
FPGA	Field programmable gate array
GALS	Globalmente asíncronos y localmente sincrónicos
HDL	Lenguaje de especificación de hardware (Hardware Description Language)
HPP	Procesador de Petri Jerárquico
IC-Core	Bloque IP, unidad reutilizable de diseño lógico, cell, o el diseño de chip
ILP	Paralelismo a nivel de instrucción
IOPT	Input-output Petri Net
IP	Propiedad intelectual
ISA	Conjunto de instrucciones de un procesador
ISB	Ill-behaved siphon in a rdp
ISTA	Integración y prueba de sistemas de automatización
LC	Controlador lógico
LCA	Matriz celular lógica
LF	Counter Latest Firing Time (ppct)
LFT	Vector de Latest Firing Time (rdpct)

LUTs	Lookup tables
MBD	Model based design
MDA	Model-driven arquitectura
MID	Model-implementation description
MINT	Emulador de MIPS
MIPS	Microprocessor without Interlocked Pipeline Stages
MPI	Interfaz de Paso de Mensajes (Message Passing Interface)
NES	Sistemas que son integrados en red (embebidos y distribuidos)
NOC	Net On Chip
P1R	Rdp con procesos secuenciales
PAR	Rdp con uno a cien procesos secuenciales
PAT	Tabla de Atributo de plaza
PCI	Bus de Peripheral Component Interconnect
PE	Elementos de proceso
PH	Cena de los filósofos modelada con una rdp
PIMs	Processing in Memory
PLC	Controlador lógico programable
PME	Mutual exclusión paralela
PNML	Petri net markup language
PNPL	Petri Net lenguaje de programación paralelo
PNSFZ	Especificación de Formato de rdp 2
POPN	Rdp orientadas a procesos
POS	Terminal punto de venta (Point of sale)
PP	Procesador de Petri
PPcT	PP con tiempo
PPTm	PP temporizado
PR1	Rdp con procesos secuenciales y un recurso común
PVM	Máquina virtual paralela (parallel virtual machine)
RC	Circuito de recursos
RdP	Redes de Petri o Red de Petri
RdPcT	Rdp con tiempo
RdPE	Rdp Estocásticas
RdPTm	Rdp temporizado
ROB	Buffer de reordenamiento en el algoritmo de Tomasulo
ROPN	Resource-oriented rdp
RPT	Rdp temporizada o con tiempo
SDLC	Ciclo de vida de desarrollo de sistemas
SEQ	Rdp con un proceso secuencial
SESC	Simulator Superscalar processor
SFC	Diagrama de función secuencial
sii	Si y solo si
SII	Sifón con mal comportamiento
SIMD	Single instruction, multiple data
SME	Mutual exclusión secuencial
SMP	Arquitectura simétrica de multiprocesadores
SO	Sistema operativo

SoC	System on Chip
SoPCs	Programmable-System-on-Chip
SPN	Rdp Estocástico
SQUARE	Rdp con múltiples procesos secuenciales y recursos compartidos
SRP	Representación de plaza estática
SSE	Extensión al grupo de instrucciones MMX para arquitecturas X86
SVG	Scalable vector graphics
SysML	Lenguaje de modelado de sistemas
TA	Transiciones de acción
TLS	Thread level speculation
TST	Tabla de estado Token
TTS	Transición temporizada de Sistemas
TTT	Tabla de Transferencia Token
VHDL	Combinación de VHSIC (High Speed Integrated Circuit) y HDL (Hardware Description Language)
w.r.t.	Bisimiliridade temporizacion débil
XNF	Xilinx netlist format
m_0	Marcado inicial
$\mathbb{N}$	Conjunto de los números naturales
$\mathbb{Q}$	Conjunto de los números racionales
$\mathbb{R}$	Conjunto de los números reales
$\mathbb{R}_{\geq 0}$	Conjunto de los números reales no negativos
$\mathbb{B}$	Conjunto de los booleano
$\|P\|$	Cantidad de plazas
$\|T\|$	Cantidad de transiciones
$\|T\|x\|P\|$	Cantidad de columnas y filas de la matriz de incidencia

Anexo H

Índice de Tablas

Anexo I

Índice de Figuras

Anexo J

Indice de Definiciones