IP Core Procesador de redes de Petri Jerárquicas

Orlando Micolini[#1], Emiliano Arlettaz[#2], Sergio H. Birocco Baudino[#3], Marcelo Cebollada[#4]

[#] Laboratorio de Arquitectura de Computadoras, FCEFyN-UNC, Córdoba, Argentina
[1] omicolini@compuar.com
[2] emilianoarlettaz@gmail.com
[3] sbbaudino@gmail.com
[4] mcebollada@gmail.com

Abstract. — La centralización de la sincronización de procesos mediante un IP-Core disminuye los tiempos requeridos para la labor de sincronización. La implementación de un procesador de Petri que ejecuta el formalismo de redes de Petri brinda una forma sencilla y efectiva de modelar, implementar, ejecutar y programar en forma directa sistemas reactivos con un alto grado de paralelismo.

En este artículo se propone el desarrollo de un procesador de redes de Petri Jerárquicas (HPNP) implementado en un IP-Core, el cual, mantiene los beneficios del procesador de Petri, logra una notable reducción en los recursos de hardware utilizados y también la ejecución paralela de los procesos con el procesador de Petri. Esto último es posible gracias a la división de las redes de Petri y la implementación en hardware de un algoritmo que permite la comunicación entre las sub-redes resultantes.

Keywords: IP-Core, Multicore, Petri Net.

1 INTRODUCIÓN

El desarrollo de los procesadores actuales ha dado lugar a la integración de muchos núcleos en un único integrado. Resultando en lo que se conoce como tecnología Multi-Core [1]. Esta apunta a conseguir mejor rendimiento en el procesamiento de tareas paralelas, con los distintos procesos y/o hilos.

Este paralelismo trae aparejado problemas de sincronización y exclusión mutua que se tratan mediante mecanismos de sincronización. Sin embargo el aumento del número de cores y de procesos concurrentes, dificulta en gran medida la implementación de sistemas reactivos con un alto grado de paralelismo [2].

El formalismo de las redes de Petri ha demostrado ser capaz de validar y verificar sistemas paralelos y es apropiado para implementarse en la resolución de problemas inherentes a la sincronización y exclusión mutua.

Existe implementación con hardware usando la matriz de adyacencia, que son programadas en el hardware, como [3, 4]. Estas soluciones no soportan: programación

en tiempo real, programación de prioridades, brazos con peso, brazos inhibidores, plazas acotadas, brazos de reset, detección de interbloqueo, disparos automáticos y redes jerárquicas. También se han implantado con lenguajes de alto nivel como en [5, 6], con las mismas restricciones que las anteriores. Aybar y Altuğ en [7] usa red de Petri temporal, ejecutadas con un lenguaje de alto nivel, también con las mismas restricciones.

En el Laboratorio de Arquitectura de Computadoras (LAC) de la FCEFyN-UNC se desarrolló un procesador, implementando en un IP Core, capaz de ejecutar toda la lógica concurrente de sistemas modelados por redes de Petri tanto de plaza/transición, todos los tipos de brazos, redes temporales, esta última en dos semánticas temporales diferentes [8-11].

Si bien el procesador de redes de Petri permite obtener una reducción notable en los tiempos de sincronización de procesos, dado que evita el uso de software para tal propósito, se ve limitado cuando se abordan sistemas de gran envergadura. Esto es dado que la capacidad de las FPGA usada para implementar dichos procesadores. Sea ha implementado en, una FPGA Spartan-6 un procesador de Petri, de hasta 50 plazas por 50 transiciones, conformando un multi core con un procesador MicroBlaze [10]. Esta limitación emerge por los recursos disponible en la FPGA, para la matriz de Incidencia [12], con la que se implementa el procesador.

Este trabajo aporto una reducción de recursos, dividiendo las redes de Petri en subredes [13]. Esta alternativa permite una reducción en los recursos para modelar e implementar los sistemas y la innovación es que la división se realiza por las transiciones manteniendo la semántica de la red, esta reducción se refleja en la cantidad de elementos necesarios en las matrices de incidencia [12].

En el presente artículo se describe el desarrollo de un procesador de redes de Petri Jerárquicas implementado en un IP-Core descripto en lenguaje Verilog, el cual posee la capacidad de ejecutar sub redes de Petri relacionadas entre sí. Y hace uso del método y el algoritmo para la división y ejecución de redes de Petri Jerárquicas, desarrollados en el Laboratorio de Arquitectura de Computadoras (LAC), que denominaremos “algoritmo de ejecución de las redes de Petri Jerárquicas”.

2 OBJETIVOS

2.1 Objetivo Principal

El objetivo principal de este trabajo es desarrollar e implementar un procesador de redes de Petri Jerárquicas en un IP-Core, haciendo uso del algoritmo de ejecución de las redes de Petri Jerárquicas.

2.2 Objetivos Secundarios

Los objetivos secundarios son:

- Describir brevemente la importancia de dividir las redes de Petri para disminuir recursos.

- Mantener en dos ciclos de reloj la ejecución para la decisión de un disparo del procesador de redes de Petri jerárquicas, que es el costo computacional del procesador de redes de Petri ordinaria.
- Implementar un algoritmo de comunicación entre subredes manteniendo relaciones simples entre las mismas.

3 REDES DE PETRI JERÁRQUICAS

Las redes de Petri pueden ser divididas, lo que facilita su comprensión y reduce los recursos necesarios en la matriz de incidencia.

Estas redes se utilizan para abordar sistemas que por su gran envergadura implican matrices de incidencia importantes. Esencialmente estas matrices tienen muchos elementos nulos. Por lo que podemos considerarla como un caso de "matriz rala" [14] ; los algoritmos para resolver matrices ralas no son aplicables a este caso, puesto que no usan operaciones lógicas simples, complicando su implementación en una FPGA.

Matemáticamente podemos definir a una red de Petri jerárquica como:

$$HPN = \{\{P_1, T_1, I_1^+, I_1^-, C_1, H_1, m_{01}\}, \{P_2, T_2, I_2^+, I_2^-, H_2, C_2, m_{02}\},$$
$$\ldots \{P_n, T_n, I_n^+, I_n^-, H_n, C_n, m_{0n}\}\}$$

El significado de cada término de este conjunto es:

- P_i: es un número finito de plazas no vacías.
- T_i: es un número finito de transiciones, $P \cap T = \emptyset$.
- I_i^++, I_i^-: son las matrices de incidencia positiva y negativa.
- H_i: matriz de arcos inhibidores.
- C_i: es un vector que contiene los valores que representan la máxima cantidad de tokens que cada plaza de la subred puede mantener.
- m_{0i}: es el marcado inicial de la subred.

Donde los conjuntos representados por los términos $\{P_1, T_1, I_1^+, I_1^-, H_1, C_1, m_{01}\}$, $\{P_2, T_2, I_2^+, I_2^-, H_2, C_2, m_{02}\}, \ldots, \{P_n, T_n, I_n^+, I_n^-, H_n, C_n, m_{0n}\}$ son subredes de Petri plaza-transición, resultantes de la división de la red de Petri original.

Estas subredes de Petri son tambien redes de Petri comunes que conforman el sistema. Cada una de estas sub redes son las implementadas como un IP Core que es el procesador de redes de Petri. El cual tiene como responsabilidad de centralizar la sincronización de procesos. La arquitectura de este procesador es la mostrada en la Figura 1.

Por otro lado el conjunto $\{R_1, .., R_n\}$ son las relaciones entre subredes con las demás sub redes, que componen el sistema. Cada una de estas relaciones se expresa con una matriz.

R_i: Matriz de relación de transiciones [15].

En las redes de Petri jerárquicas cada subred que compone el sistema posee un estado particular en un momento dado, esto da lugar a que haya distintas transiciones sensibilizadas por cada subred del sistema en un instante determinado, las cuales se dispararán según un esquema de prioridades. Existen dos tipos de transiciones en este tipo de redes, las transiciones internas y las transiciones de borde [15]. Para que una transición de borde pueda ser disparada es necesario que todas las transiciones de borde que representan a esa única en el sistema original estén sensibilizadas, esto conserva la semántica original de las transiciones.

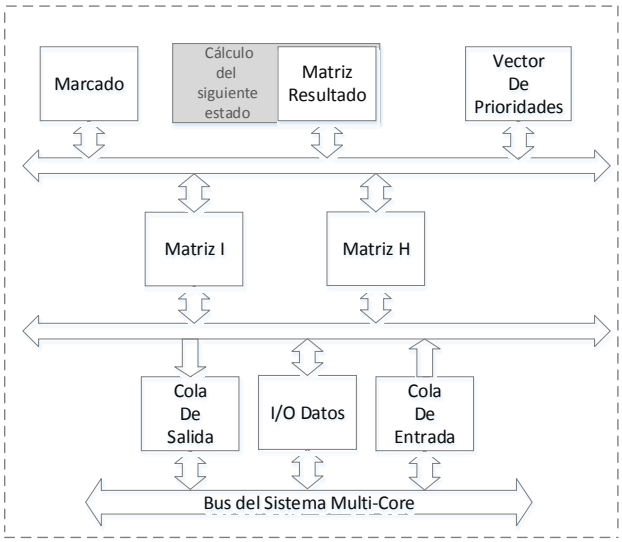

Figura 1. Arquitectura del Procesador de Redes de Petri

En el ejemplo de la **Figura 2**, vemos una transición con tres arcos de entrada y uno de salida.

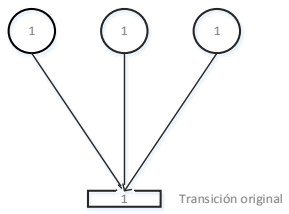

Figura 2. Red de Petri simple sin dividir

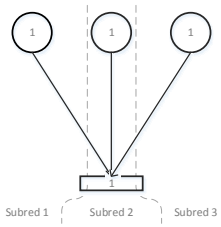

Figura 3. División de red de Petri en subredes

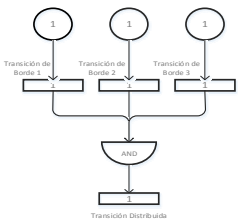

Figura 4. Esquema de sensibilización de una transición distribuida

Observamos aquí que la transición se encontrará sensibilizada si y solo si existe la cantidad necesaria de tokens en las plazas de entrada de la misma. Realizamos ahora, la división de esta red de Petri, como se ha explicado en el algoritmo, resultando las subredes de laFigura 3.

Del corte de la transición, resultan 3 subredes que tienen a esta última como transición de borde, la cual define la relación existente entre dichas subredes. Ahora en el sistema dividido la transición solo podrá dispararse cuando todas las transiciones de borde que la componen estén sensibilizadas, se define al conjunto de estas transiciones de borde como transición distribuida, como se muestra en la Figura 4.

De manera que la transición original del sistema ahora puede ser definida como una transición distribuida, apresada a continuación:

Transición Distribuida = {T. Borde 1, T. Borde 2, T. Borde 3}

Generalizando esta expresión la ecuación de las transiciones de borde sensibilizadas en una subred es:

$$sensibilizados\_borde \; = \; matriz\_relación \; x \; sensibilizados\_signo$$

Luego se obtiene un vector con las transiciones distribuidas que se pueden disparar para cada subred y mediante una máscara de habilitación; que inhabilita aquellas transiciones sin relación, cuya expresión es:

$$sensibilizados\_borde\_correlacionados[i] \\ = \; sensibilizados\_borde[i] \; \; OR \; mascara\_habilitacion[i]$$

Finalmente mediante una operación AND elemento a elemento entre los vectores obtenemos la:

$$sensibilizados\_borde\_correlacionados$$

De cada subred se obtiene las transiciones distribuidas de todo el sistema que están sensibilizadas, esto es:

$$sensibilizados\_distribuidos[i] = \bigwedge_{j=0}^{r-1} sensibilizados\_borde\_correlacionados_j[i]$$

Donde r es la cantidad de sub redes en que se ha dividido el sistema.

La comunicación entre las distintas subredes para el disparo de transiciones de borde se realiza con el algoritmo mencionado. Este último permite mantener la ejecución del sistema respetando el modelo original del mismo, y es implementado en hardware mediante un circuito combinacional logrado a partir de compuertas simples que permiten mantener los beneficios computacionales alcanzados por el procesador de Petri.

El procesador de redes de Petri en conjunto con el mecanismo para la comunicación de redes de Petri dado por el **algoritmo de ejecución de las redes de Petri Jerárquicas**, hace posible la implementación del procesador de redes de Petri jerárquicas.

4 ARQUITECTURA Y FUNCIONAMIENTO DEL PROCESADOR DE REDES DE PETRI JERÁRQUICAS

La arquitectura general del procesador de redes de Petri Jerárquicas se presenta en la Figura 5.

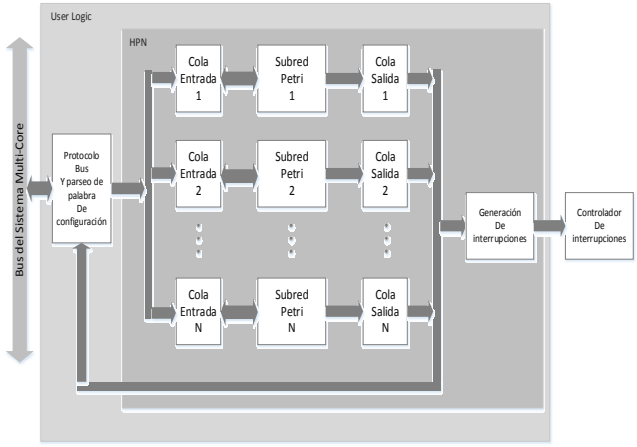

Figura 5. Arquitectura procesador de redes de Petri Jerárquicas

En la parte central de Figura 5 se puede observar los datos referidos a cada subred, donde existe un módulo llamado Subred Petri el cuál calcula todos los vectores y matrices del **algoritmo de ejecución de las redes de Petri Jerárquicas** (AERPJ) referidos a una subred. A este módulo se le adiciona un módulo de cola de entrada y un módulo de cola de salida, los cuales son contadores saturados que llevan la cuenta

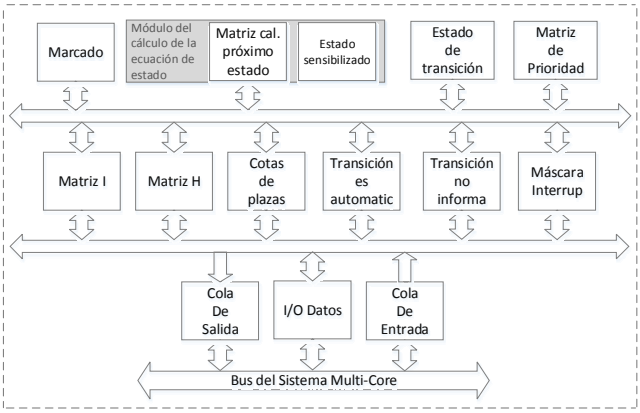

Figura 6. Arquitectura de una subred de Petri individual

de las transiciones solicitadas y las transiciones ya realizadas respectivamente.

El módulo Subred Petri se instanciará tantas veces como subredes hayan resultado de la división de la red de Petri original.

En el HPNP existe un módulo principal denominado HPN (Jerárquica Petri Net), el cual posee todas las instancias de los módulos Subred Petri y sus correspondientes colas de entrada y salida. La función principal de este módulo es decidir que transición distribuida se ejecutará en cada ciclo y de comunicar esta decisión a las subredes. Esto último es necesario para que dichas subredes puedan concluir la ejecución del AERPD.

El módulo HPN es instanciado dentro de un módulo llamado UserLogic. Éste módulo es el encargado de implementar la comunicación del IP Core con el bus, siguiendo el protocolo del bus. También tiene como función traducir (parsear) las señales del bus de datos a la palabra de configuración establecida para el funcionamiento del módulo HPN.

La Figura 6 muestra la arquitectura correspondiente a las subredes del sistema y los distintos elementos que permiten la comunicación y funcionamiento de las mismas.

5 COMUNICACIÓN ENTRE EL IP CORE Y LOS PROCESOS DEL SISTEMA

El objetivo de las etiquetas, que se definen a continuación, es facilitar la comunicación entre el procesador de Petri jerárquico y los procesos.

Con referencia a la comunicación de la transición hacia el proceso, existen 4 posibilidades, estas son: solicitud de disparo de una transición (D), disparo automático de una transición (A), informe del disparo de una transición (I) y que no se informe del disparo (N). Para esto se creó una nomenclatura de etiqueta doble que indica cómo se realizará la programación del IP Core:

Cada una de las letras que forman las etiquetas tiene el siguiente significado:

- D (disparo) → La transición es pedida por el proceso al IP Core y es registrada en la cola de entrada de la subred para ser ejecutada.
- A (automática) → La transición se dispara automáticamente cuando es sensibilizada, no hay una cola para esta transición.
- I (informa) → Cuando la transición se ha disparado, es registrada en la cola de salida de la subred para que informe al procesador que accede al IP Core.
- N (no informa) → La transición, una vez ejecutada, no se encola en la cola de salida de la subred.

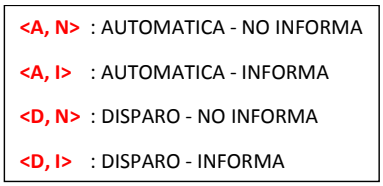

Figura 7. Nomenclatura de etiqueta de transiciones

Estas etiquetas son colocadas en cada una de las transiciones de la red, para determinar cómo se resuelve cada transición.

Estas etiquetas son programadas mediante el uso de una palabra binaria de configuración del IP Core, que es mostrada en la Tabla 1.

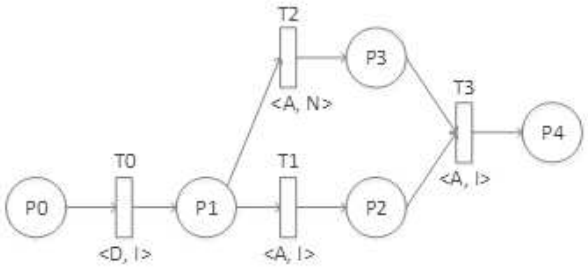

Figura 8. Ejemplo de uso de etiquetas de transición

Mediante esta palabra de configuración se puede controlar completamente el IP Core. Para el caso de las etiquetas de comunicación se usa esta palabra designando un vector para aquellas transiciones que serán automáticas y un vector para aquellas que serán no informadas.

Tabla 1: Palabra de configuración del IP Core

Red	Matriz o vector	Fila	Columna	Valor
5 bits	5 bits	7 bits	7 bits	8 bits

En el caso de las transiciones de borde de las subredes que representan la misma transición distribuida se pueden programar todos los disparos como automáticos o solo uno como explicito, con el fin de evitar ambigüedades; mientras que la etiqueta de informa o no informa puede estar en cualquier subred, dependiendo de la necesidad del programa.

Además también se ha implementado en este IP Core arcos inhibidores, plazas acotadas, arcos de reset, arcos de lectura, semántica temporal, etc. Todo esto haciendo uso de las matrices y/o vectores correspondientes, lo que facilita la programación directa (sin hacer uso de lenguaje alguno, solo los vectores y matrices).

6 CASO DE CANAL MARÍTIMO

En esta sección se presenta un caso, sobre tráfico marino, donde se controla un sistema de tráfico de canal [16], 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 red de Petri de este ejemplo y se determinó la existencia de interbloqueo, lo cual solucionan las dos plazas de control, P15 y P16, para evitar el interbloqueo del sistema.Se muestra en la Figura 9, la red de Petri del canal marítimo con las plazas de control.

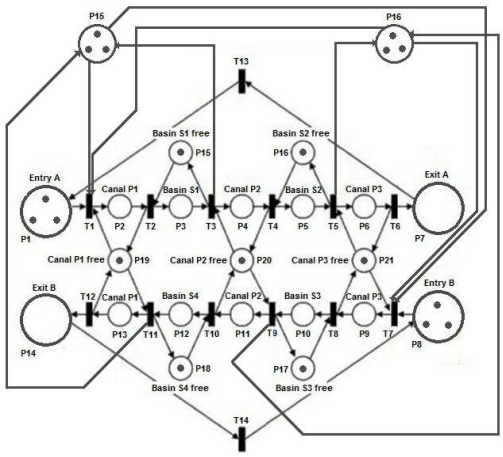

Figura 9. Red de Petri de Canal Marítimo con control de interbloqueo.

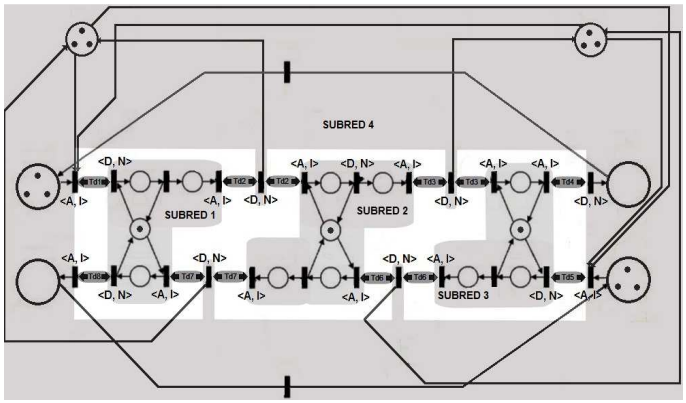

Figura 10. Red de Petri Jerárquica de Canal Marítimo

6.1 Pruebas de rendimiento

Para evaluar las el beneficio de las redes de Petri jerárquicas frente al uso de otros mecanismos, se ha dividido el sistema en 4 subredes, por lo que se requieren 8 transiciones distribuidas que relacionan dichas subredes. A continuación, en la Figura 10, se muestra la división realizada donde cada subred está delimitada por un fondo gris con las etiquetas correspondientes para la programación de cada una de las transiciones de borde.

De la Figura 9 se obtiene que se requieren 266 unidades de recursos para implementar la red de Petri sin dividir (14 transiciones por 19 plazas) y de la Figura 10 se obtiene que son necesarias 130 unidades de recursos (4 subredes con 20, 30, 20 y 60 unidades de recursos, que se obtienen multiplicando las plazas por las transiciones de cada sub red). Con lo que se ha obtenido una disminución de recursos del 51%.

Para la sincronización, al igual que las pruebas realizadas con semáforos, se realizaron distintos casos de prueba. Entre estas se simuló cada estado o plaza como una actividad representada por incrementar 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: semáforos, IP Core HPNP con espera activa y cediendo el procesador (Yield), se corrieron 10 veces tomando como resultado final un valor promedio.

Para llevar el conteo de clocks de cada implementación se utilizó un IP Core "AXI Timer", donde definimos un incremento de rendimiento del uso del IP Core HPNP frente al uso de semáforos, como el cociente entre ambas medidas: $\eta = \frac{T_{sem}}{T_{HPN}}$

Dónde:

- T_{sem} es el tiempo de ejecución para las implementaciones con semáforos.
- T_{HPN} es el tiempo de ejecución para las implementaciones con en IP Core.

Tomando como caso de referencia, el de 32 incrementos por cada actividad, se obtiene la Tabla 2.

Tabla 2: Tiempos de ejecución para cargas de 32 incrementos

Embarcaciones	**Semáforos**	**HPN espera activa**	**HPN 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 clocks del sistema a 25MHz.			

Realizando el cociente para calcular el incremento de rendimiento del uso del IP Core HPNP frente al uso de semáforos, resulta la Tabla 3. Como se puede observar en la Esto teniendo en cuenta una carga de trabajo igual a 32 incrementos de una variable como actividad de los procesos. 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 11, que nos muestra el incremento de rendimiento del uso del IP Core HPNP con espera activa, frente al uso de semáforos.

Tabla 3, el uso del IP Core HPNP (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 como actividad de los procesos. 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 11, que nos muestra el incremento de rendimiento del uso del IP Core HPNP con espera activa, frente al uso de semáforos.

Tabla 3: Incremento de rendimiento para carga de 32 incrementos

Barcos	Semáforos/HPN Activa	Semáforos/HPN Yield
1	1,23	1,23
20	1,39	1,39
40 a 127	1,43	1,42

En la Figura 11, se observa que para valores muy pequeños de carga de trabajo de las actividades en las plazas, no hay ganancia; puesto que, prácticamente la totalidad del tiempo de ejecución corresponde a la sincronización.

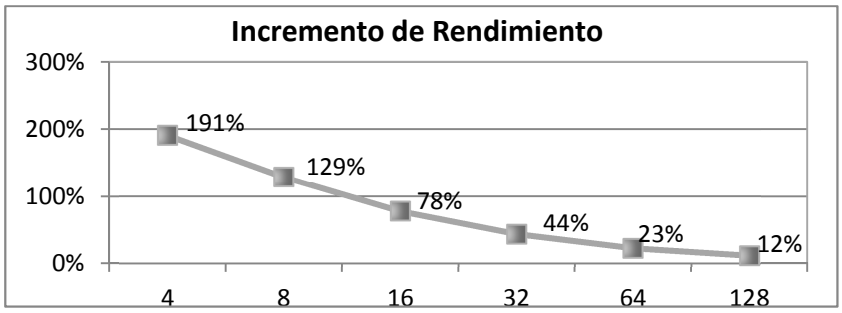

Figura 11. Mejora con el uso del IP Core HPNP, frente al uso de semáforos.

Un valor de mejora esperado, es el que ha sido medido en los trabajos de simulación de [8, 10], está en el orden de 44% a 23%. En nuestro caso, estos valores se corresponden con cargas de trabajo de 32 y 64 incrementos.

7 Conclusión

Mediante los resultados obtenidos en las pruebas realizadas usando el IP Core HPNP, para sistemas concurrentes y paralelos, se ha obtenido un incremento de rendimiento significativo, que se traduce en menores tiempos totales de ejecución de los programas y reducciones de recursos que van entre el 40% y el 60%. Este incremento de tiempo se hace más significativo a medida que las aplicaciones poseen un mayor grado de sincronización. Por el contrario, para aplicación con poca sincronización, este aumento de rendimiento no es porcentualmente significativo, pero aun así el uso del IP Core HPNP nos permite implementar el sistema directamente a partir de la red de Petri sin necesidad de programación adicional, a partir de las matrices y vectores del modelo creado para validar el sistema, lo que evita la codificación. Además mantiene la versatilidad de las Redes de Petri para modelar sistemas concurrentes y paralelos, lo que hace que el sistema resultante sea flexible ante cambios de requerimientos. Hay que hacer notar que seste procesador mantiene los dos siclos para resolver un disparo.

En este momento se está trabajando en elaborar un algoritmo automático que minimice los recursos y las distancias para optimizado el sistema resultante, y que también permita explotar esta división para distribuir en la FPGA el procesador con el fin de mejorar la velocidad de reloj.

Referéncias Bibliográficas

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. P. M. Micolini Orlando, Gallia Nicolás A., Alasia Melisa A., "Procesador de Petri para la Sincronización de Sistemas Multi-Core Homogéneos," *CASECongreso Argentino de Sistemas Embebidos*, pp. 3-8, 2012.
9. I. F. J. Nonino, C. R. Pisetta, O. Micolini, Member, "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.
10. N. G. M. Pereyra, M. Alasia and O. Micolini, "Heterogeneous Multi-Core System, synchronized by a Petri Processor on FPGA," *IEEE LATIN AMERICA TRANSACTIONS*, vol. 11, pp. 218-223, 2013.
11. J. N. y. C. R. P. Orlando Micolini, "IP Core Para Redes de Petri con Tiempo," *CASIC 2013*, pp. 1097-110, 2013.
12. M. Diaz, *Petri Nets Fundamental Models, Verification and Applications*. NJ USA: John Wiley & Sons, Inc, 2009.
13. K. J. L. M. Kristensen. (2006). *Coloured Petri Nets Modelling and Validation of Concurrent Systems*. Available: http://www.daimi.au.dk/~tjell/abm2006
14. D. A. Tacconi, Lewis, F.L, "A new matrix model for discrete event systems: application to simulation," *Control Systems, IEEE*, vol. 17
15. 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.
16. I. P. Ned eljko Peri, " An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net," *ATKAAF*, 2006.