Universidad Carlos III de Madrid

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO 2015-2016

TRABAJO FIN DE GRADO

Diseño de un módulo observador para un microprocesador ARM9 en un SOPC

Manuel Peña Fernández

Tutora

Almudena Lindoso Muñoz

Departamento de Tecnología Electrónica

Fecha de entrega	Fecha de defensa
26 de septiembre de 2016	4 de octubre de 2016

A mi familia.

Agradecimientos

En primer lugar quisiera mostrar mi agradecimiento a los profesores y otros miembros del Departamento de Tecnología Electrónica de la Universidad Carlos III de Madrid, por su encomiable valor y sentido del trabajo bien hecho que me han llevado a realizar este Trabajo Fin de Grado. Particularmente a Almudena, por su alta disponibilidad y dedicación en el papel de tutora, enriqueciendo el resultado del trabajo gracias a su ayuda, orientación y experiencia, y por haberme exigido hasta obtener lo mejor de mí.

Gracias a todos mis profesores, desde el primero hasta el último, por enseñarme que estudiar podría llevarme a una gran satisfacción, y forjarme un futuro prometedor.

Gracias a la Universidad Carlos III de Madrid, en cuyo seno he pasado los últimos cuatro años, y aún me quedan fuerzas para continuar otro más.

A mis compañeros de clase, por su apoyo y amistad durante la carrera. En especial a Jorge Plaza, Dragos Andrei Poiana y Jorge Rodríguez, por su especial paciencia conmigo. No hubiéramos conseguido tanto separados como hemos conseguido juntos.

A mis amigos, por seguir ahí, entusiasmándonos al ver cómo poco a poco vamos construyendo nuestra vida. Formáis también parte de esta historia.

A mi familia. Por creer siempre en mí y apoyarme incondicionalmente para que pueda seguir estudiando.

A Patricia, por quererme tanto.

Resumen

En este Trabajo de Fin de Grado se aborda el diseño de un módulo hardware capaz de observar el flujo de ejecución de un microprocesador ARM Cortex-A9, con el que poder detectar errores en su funcionamiento. Para ello ha sido necesario un profundo estudio y comprensión del subsistema ARM CoreSight donde se integra la interfaz de traza utilizada en la observación, así como su posterior configuración y pruebas de funcionamiento.

La necesidad de este desarrollo se enmarca en el creciente problema que suponen los errores transitorios (en inglés *soft errors*) para el funcionamiento de los circuitos digitales. Es un problema tan antiguo como la electrónica, ya que tiene su principal causa en las interferencias producidas por radiación cósmica y electromagnética, y existen técnicas (denominadas "*hardening*" o endurecimiento) para aumentar su tolerancia a fallos desde la época de la carrera espacial.

De un tiempo a esta parte, los grandes avances realizados en las tecnologías de silicio han propiciado notables incrementos tanto en sus prestaciones como en la eficiencia energética asociada, ampliando los sectores de aplicación de los microprocesadores. Sin embargo, la gran complejidad y densidad de integración asociadas, hace a las nuevas generaciones de microprocesadores cada vez más vulnerables a estos errores ya no solamente a nivel aeroespacial, sino también en aplicaciones terrestres.

En cualquier sistema electrónico la fiabilidad es fundamental, y más si se utiliza en una aplicación crítica para la seguridad. Aparece, por tanto, la necesidad de renovar las técnicas de endurecimiento y adaptarlas a las nuevas necesidades, distinguiéndose tres categorías: técnicas hardware, software e híbridas. De ellas, las más efectivas son las técnicas hardware, sin embargo requieren un exhaustivo conocimiento del circuito a robustecer, información que no siempre se encuentra disponible en el ámbito del mercado de la electrónica de consumo; en el que ARM tiene una posición dominante.

Palabras clave: ARM, Cortex-A9, CoreSight, tolerancia a fallos, endurecimiento, radiación, interfaz de traza, errores transitorios.

Abstract

In this Bachelor Thesis, the design of a hardware module capable of observe the execution of an ARM Cortex-A9 is addressed, with the aim of detecting operation errors. A deep study and comprehension of ARM CoreSight subsystem has been needed, that integrates the trace interface used in observations. Also, operation tests and configurations have been done.

This work is related with the rising problem involving soft errors in the digital circuits' normal behavior. This problem is as old as electronics, and has its main cause in cosmic and electromagnetic radiation interferences. Hardening techniques have been developed in order to increase the associated fault-tolerance since the space race.

In the past few years, great advances made in silicon technologies have promoted big growth both in performance as in energy efficiency. Thus, application sectors for microprocessors have been expanded. However, the big complexity and integration density reached make new processor's generations even more vulnerable to soft errors, not only in space but also at ground level.

Reliability is a must in every electronic system, more if it's used for safety-critical applications. Therefore, the need to renew or adapt hardening techniques to new needs, appear. Three different categories can be applied: hardware techniques, software and hybrid ones. Among them, the most effective ones are hardware ones, but they require a deep knowledge about the hardened circuit. This information is not always available in consumer electronics market, in which ARM has a privileged position.

Keywords: ARM, Cortex-A9, CoreSight, fault-tolerance, hardening, radiation, trace interface, soft errors.

Índice

Agradecimientos iii
Resumen iv
Abstract v
Índice vi
Índice de tablas ix
Índice de figuras x
Capítulo 1 Introducción 1
1.1 Planteamiento 2
1.1.1 Objetivo 2
1.1.2 Punto de partida 2
1.1.3 Retos principales 3
1.1.4 Objetivos secundarios y temporización 4
1.2 Estructura de la memoria 5
Capítulo 2 Estado de la técnica 6
2.1 Introducción 6
2.2 Tolerancia a fallos 8
2.3 Interfaz de traza y observación no intrusiva 13
2.4 Sistemas empotrados 14
2.4.1 Sistemas empotrados programables 14
2.5 Alternativas de diseño 16
2.6 Requisitos del sistema 17
Capítulo 3 Sistema empleado 18
3.1 Introducción 18
3.2 Placa utilizada 19
3.2.1 Familia ZYNQ de Xilinx 19
3.2.2 ZYBO 20
3.3 Características del sistema 21
3.3.1 Lógica programable 22
3.3.2 Microprocesador 22

3.3.3 Interfaz de traza 23
3.4 Entorno de desarrollo utilizado 29
3.4.1 Xilinx Vivado 29
3.4.2 Xilinx SDK 30
Capítulo 4 Sistema implementado 31
4.1 Introducción 31
4.2 Configuración de la interfaz de traza 32
4.2.1 Procesador 32
4.2.2 Program Trace Macrocell (PTM) 33
4.2.3 Funnel 37
4.2.4 Replicator 38
4.2.5 Trace Port Interface Unit (TPIU) 38
4.3 Diseño del módulo observador 42
4.3.1 Caracterización de la información de traza 42
4.3.2 Solución propuesta 45
4.3.3 Resultados de síntesis 51
Capítulo 5 Resultados experimentales 54
5.1 Introducción 54
5.2 Software empleado 55
5.3 Experimentos realizados 56
5.3.1 Depuración de las señales internas 56
5.3.2 Comportamiento del módulo diseñado 58
5.4 Conclusiones 64
Capítulo 6 Conclusiones y líneas futuras 65
6.1 Conclusiones 65
6.2 Líneas futuras 67
6.2.1 Línea futura: ampliar el módulo observador 67
6.2.2 Línea futura: perfeccionar el diseño 67
6.2.3 Línea futura: validación del diseño 67
6.2.4 Línea futura: experimentos de radiación 68
6.2.5 Utilizar traza con varios orígenes 68

6.2.6 Línea futura: integración en técnicas híbridas 68
Acrónimos 69
Bibliografía 72
Anexos 76
Anexo A: Planificación y presupuesto 76
i. Descomposición en fases 76
ii. Presupuesto 78

Índice de tablas

Tabla 3-1 Comparativa entre depuración y traza 24
Tabla 3-2 Selección de los componentes de traza 28
Tabla 4-1 Configuraciones recomendadas de la TPIU, ARM [19, p. 209] 40
Tabla 4-2 Posibles configuraciones de la TPIU en la ZYNQ, Xilinx [16, p. 727] 40
Tabla 4-3 Paquetes de la especificación ARM PFT Architecture 42
Tabla 4-4 Previsión de aparición de los distintos paquetes de traza 44
Tabla 4-5 Uso de la señal TRACECTL [19, p. 211] 47
Tabla 4-6 Uso de los recursos por el módulo observador 53
Tabla 4-7 Resultados en velocidad del módulo observador, a 150MHz 53
Tabla 5-1 Comportamiento de las señales TRACE_DATA y TRACE_CTL 59
Tabla 5-2 Información de traza extraída del ETB 60
Tabla 5-3 Posiciones del PC obtenidas por simulación 62
Tabla 5-4 Correlación de la información de traza con la ejecución del programa 63
Tabla A-1 Diagrama de Gantt del desarrollo del proyecto 77
Tabla A-2 Presupuesto del proyecto 78

Índice de figuras

Figura 3-1 Productos de la familia ZYNQ [12] 19
Figura 3-2 Placa ZYBO [14] 20
Figura 3-3 Vista general de componentes de una ZYNQ-7000 [16, p. 27] 21
Figura 3-4 Flujo genérico de la información de traza 25
Figura 3-5 Sistema CoreSight completo a bordo de la ZYNQ [16, p. 714] 26
Figura 3-6 Subsistema de traza a bordo de la ZYNQ [16, p. 722] 26
Figura 3-7 Ruta definitiva de la información de traza 28
Figura 3-8 Interfaz principal de la aplicación Xilinx Vivado 30
Figura 4-1 Arquitectura del núcleo Cortex-A9 [16, p. 66] 32
Figura 4-2 Diagrama funcional de la PTM-A9 [26, p. 14] 33
Figura 4-3 Ejemplo de configuración de los recursos de la PTM [27, p. 51] 35
Figura 4-4 Diagrama funcional del Funnel [19, p. 160] 37
Figura 4-5 Diagrama funcional de Replicator [19, p. 154] 38
Figura 4-6 Diagrama funcional de la TPIU [19, p. 186] 38
Figura 4-7 Cronograma de las señales de la TPIU por EMIO [16, p. 728] 45
Figura 4-8 Diagrama de bloques conceptual 46
Figura 4-9 Delimitador de paquetes del módulo observador 48
Figura 4-10 Seguidor de PC del módulo observador 49
Figura 4-11 Diagrama de bloques definitivo 50
Figura 4-12 Diagrama de bloques de Vivado del sistema completo 52
Figura 4-13 Flujo del proceso de diseño en Vivado 52
Figura 4-14 Gráfica de uso de recursos del módulo observador 53
Figura 5-1 Diagrama de flujo del software de prueba 55
Figura 5-2 Captura de las señales de traza inactiva 56
Figura 5-3 Captura de las señales de traza activa 57
Figura 5-4 Simulación del módulo observador 61
Figura 5-5 Comportamiento real del módulo observador. 62

Capítulo 1
Introducción

En la actualidad, cada vez son más numerosos los entornos de aplicación de los microprocesadores. De un tiempo a esta parte, los grandes avances realizados en las tecnologías de silicio han propiciado notables incrementos tanto en sus prestaciones como en la eficiencia energética asociada. Gracias a ello, ahora es posible integrarlos en prácticamente cualquier dispositivo, aumentando su flexibilidad y prestaciones. Uno de los sectores donde más se aprecia este crecimiento es en los dispositivos móviles.

En este ámbito, ARM ha alcanzado una posición predominante, ofreciendo un amplio abanico de arquitecturas que se adaptan a las exigencias de cada escenario. ARM no fabrica sus chips, sino que pone a disposición de los fabricantes los medios necesarios para fabricar los procesadores que diseña, dejando a sus clientes la tarea de seleccionar la configuración final del circuito.

Además, el descenso del tamaño de los transistores, en combinación con el aumento de la densidad de integración en los chips permite fabricar procesadores a precios muy competitivos. Sin embargo dada la complejidad que han alcanzado los sistemas electrónicos, cada vez son más vulnerables a interferencias y radiaciones que pueden causar fenómenos físicos que desencadenen un comportamiento inadecuado en los circuitos, pero sin llegar a destruirlos. Se conocen como errores transitorios o *soft errors*, y su impacto en los circuitos digitales actuales no es despreciable. Las técnicas para paliar o prevenir sus efectos se enmarcan en el campo de tolerancia a fallos, y se denominan de endurecimiento (*"hardening"* en inglés), que se refiere a robustecer un diseño para reducir la tasa de aparición de estos errores.

A medida que aumenta la altitud, se alcanzan mayores niveles de radiación cósmica y electromagnética, incrementando la tasa de errores transitorios, y por ese motivo los circuitos de aplicación aeroespacial se fabrican mediante el uso de materiales y procesos altamente especializados que aíslan los componentes de las condiciones externas, incorporando comúnmente estructuras redundantes. Sin embargo estas soluciones no son válidas para un mercado en el que predominan las condiciones de fabricación en masa con bajo coste, bajo consumo y pequeño tamaño, lo que ha llevado al desarrollo de técnicas optimizadas para este nuevo escenario.

En este Trabajo Fin de Grado se aborda el diseño de un módulo hardware capaz de supervisar el flujo de ejecución de un núcleo ARM Cortex-A9 a través de la interfaz de traza estándar de ARM, de manera no intrusiva. Gracias a su pequeño tamaño se podría integrar junto al microprocesador sin que se produjera un incremento del coste de fabricación, y permitiría detectar errores transitorios en toda la gama de microprocesadores más utilizados en la industria de consumo.

1.1 Planteamiento

1.1.1 Objetivo

El objetivo principal es el diseño de un módulo observador externo que permita conocer el flujo de ejecución del software en un microprocesador moderno de altas prestaciones y elevada cuota de mercado, como es la arquitectura ARM Cortex-A9. Se deben cumplir las siguientes condiciones:

- Supervisar el correcto comportamiento del microprocesador sin necesidad de conocer a priori el código ejecutado, reduciendo tanto como sea posible la información necesaria para esta tarea.
- Realizar su función sin acceder a ninguno de los sistemas internos del microprocesador, tan solo mediante la información proporcionada por la interfaz de traza, de manera no intrusiva.
- Operar en tiempo de ejecución, decodificando los paquetes de información tan rápido como son recibidos para no pasar ninguno por alto.
- Requerir un área reducida para su implementación, de manera que pueda integrarse junto al procesador sin suponer un incremento del área total de silicio necesaria ni, por consiguiente, de su precio de fabricación.
- Permitir la compatibilidad con componentes comerciales, también denominados COTS (*Commercial Off The Shelf*).

La enumeración anterior solamente hace referencia al resultado final del trabajo, sin embargo para cumplir con el objetivo principal también es importante analizar el punto de partida del trabajo, identificar los retos presentes en su desarrollo y extraer de ellos los objetivos secundarios que marcarán la progresión hacia su consecución.

1.1.2 Punto de partida

La elección de la temática del trabajo no ha sido algo casual. Ya el Grado en Ingeniería Electrónica Industrial y Automática confiere nociones a través de asignaturas tales como “Programación”, “Fundamentos de Ingeniería Electrónica”, “Electrónica Digital” e “Informática Industrial”, que permiten un acercamiento al mundo de la programación en C, los microcontroladores y el uso de circuitos de lógica programable.

Además, el autor ha demostrado un elevado interés por el campo de la electrónica digital y ha intensificado su formación durante el grado con la elección de asignaturas optativas tales como “Microprocesadores”, “Diseño de Circuitos Integrados”, “Sistemas Electrónicos Digitales” o “Microelectrónica”, obteniendo excelentes resultados.

Ello justifica la idoneidad del autor para afrontar la elaboración del presente Trabajo Fin de Grado.

1.1.3 Retos principales

El TFG constituye la última etapa del Grado, y tal y como reza la Ficha Reina debe suponer una verificación del *"nivel conseguido en la adquisición de las competencias por parte del alumno"*, pero pese a ello no debe olvidarse su carácter formativo intrínseco, ya que como resultado de su desarrollo, el alumno también debe adquirir la capacidad de realizar tareas como:

- *"Búsqueda y gestión de información técnica, comercial, etc. relacionada con el proyecto."*
- *"Investigación y autoaprendizaje para proponer la mejor solución posible al problema planteado en el trabajo fin de grado."*

La formación recibida en el Grado debe ser por tanto el vehículo que permita elaborar el Trabajo Fin de Grado, proporcionando el *"Conocimiento en materias básicas y tecnológicas, que les capacite* [a los alumnos] *para el aprendizaje de nuevos métodos y teorías, y les dote de versatilidad para adaptarse a nuevas situaciones"*.

Tal es efectivamente el contexto del presente trabajo, en el que se parte de un bagaje amplio, y a partir de él se pretenden encarar distintas tareas novedosas que proporcionen un aprendizaje avanzado al autor en una disciplina de su interés. Confluyen las siguientes realidades:

- Un marcado carácter investigador, enmarcado en una línea de investigación actual del grupo de Diseño Microelectrónico y Aplicaciones (DMA) de la Universidad Carlos III de Madrid y con una elevada carga en búsqueda y análisis de documentación.
    - Es preciso destacar la dificultad que entraña, ya que el objeto del trabajo es un área prácticamente inexplorada, sobre la que no se ha encontrado ningún estudio ni desarrollo previo en el que apoyarse.
- Aprender a utilizar nuevas herramientas profesionales de diseño por computador de circuitos digitales y sistemas en chip programable (SOPC) que no se incluyen en la docencia del Grado.
- Familiarizarse con sistemas hardware punteros en las aplicaciones desarrolladas.

Así pues, se pueden desgranar las distintas etapas que debe atravesar el desarrollo del trabajo para lograr la consecución de su objetivo, detalladas en el siguiente apartado.

1.1.4 Objetivos secundarios y temporización

Los siguientes objetivos marcarán el desarrollo del trabajo. Se les ha dado una asignación temporal preliminar para estimar el tiempo de realización de cada tarea y poder realizar una aproximación del tiempo total necesario.

- Obtener las fuentes de información necesarias para documentar suficientemente el funcionamiento de los sistemas y subsistemas empleados. Tiempo estimado: 1 semana.
- Aprender a utilizar una herramienta de diseño desconocida hasta el momento. Tiempo estimado: 2 semanas.
- Conocer el concepto de sistema empotrado, así como la metodología de desarrollo asociada. Tiempo estimado: 1 semana.
- Trabajar con un microprocesador de altas prestaciones, desconocido hasta el momento, y familiarizarse con su funcionamiento. Tiempo estimado: 1 semana.
- Localizar, analizar y comprender el funcionamiento de la interfaz de traza a bordo del microprocesador y configurarla acorde con la funcionalidad prevista del módulo observador a diseñar. Tiempo estimado: 3 semanas.
- Diseñar el módulo observador. Tiempo estimado: 1 semana.
- Conectar el módulo una vez diseñado, al microprocesador para obtener el sistema completo. Tiempo estimado: 1 día.
- Implementar el sistema completo en un hardware real que pueda alojarlo, como puede ser una placa de desarrollo. Tiempo estimado: 1 día.
- Probar el funcionamiento del módulo diseñado tanto en simulación como en la aplicación real. Tiempo estimado: 1 semana.
- Diseñar el software ejecutado por el microprocesador para realizar dichas pruebas. Tiempo estimado: 1 semana.
- Analizar los resultados para validar o rechazar el diseño. Tiempo estimado: 1 semana.
- Documentar el proceso de diseño y realizar la memoria. Tiempo estimado: 4 semanas.

Las estimaciones temporales están fuertemente relacionadas, ya que en varios de los casos, la consecución de una condiciona la posibilidad de seguir adelante en el proceso. En el caso de que el tiempo requerido realmente sea mayor que el estimado, es posible se queden las últimas partes del desarrollo sin cubrir.

Según esta primera aproximación, el trabajo podría realizarse por completo en 16 semanas, aproximadamente 4 meses.

1.2 Estructura de la memoria

La presente memoria constituye la documentación de todo el proceso de desarrollo del Trabajo Fin de Grado, comenzando por una introducción que contiene información relevante sobre el planteamiento del problema y el punto de partida, hasta finalizar el documento las conclusiones del autor y presentando líneas futuras acordes con los progresos realizados. Cuatro capítulos intermedios permiten desplazarse desde una hasta las otras:

2. Estado de la técnica. Permite ubicar el problema abordado en la realidad actual del desarrollo de la electrónica digital y comprender las distintas tendencias en el campo de aplicación estudiado. Incorpora los apartados necesarios para plantear las alternativas existentes en el diseño así como los criterios utilizados en la elección de unas u otras.
3. Sistema utilizado. Tras la elección de una solución concreta, se presentan los recursos utilizados en su desarrollo tales como herramientas informáticas, así como el hardware específico y sus características globales y específicas para la aplicación en estudio.
4. Sistema implementado. Se sintetiza la información presentada en el apartado anterior y se enfatiza en los puntos clave de la configuración para detallar justificadamente el proceso de diseño realizado.
5. Resultados experimentales. La realización de distintas pruebas sobre el sistema obtenido permite determinar si su comportamiento es el esperado, así como demostrar que tiene la funcionalidad deseada.

En la última parte se incluyen nuevos apartados que ordenan información relevante sobre el documento:

- Índice de acrónimos, utilizados durante el documento, para facilitar su consulta.
- Bibliografía, que engloba la totalidad de documentación consultada para la elaboración del Trabajo Fin de Grado, y que se cita en esta memoria.
- Anexos, que incorporan información relevante sobre el proyecto tal como la distribución del tiempo y la planificación del mismo, así como sus atribuciones económicas.

Capítulo 2
Estado de la técnica

2.1 Introducción

Los circuitos digitales están presentes en una gran cantidad de áreas de la sociedad actual: informática, telecomunicaciones, medicina, transporte, defensa, industria, investigación, espectáculos, y un largo etcétera. En cada caso, el uso de la electrónica permite resolver los problemas característicos asociados, gracias a la facilidad de acceso a la tecnología necesaria, unido a una gran oferta de soluciones de numerosos fabricantes.

La tendencia actual es que cada vez más soluciones electrónicas se implementan a través de sistemas basados en microprocesadores, y está propiciada por un incremento notable de sus prestaciones de un tiempo a esta parte gracias a los avances en las tecnologías de semiconductores, que permiten diseñar sistemas:

- Flexibles, y fácilmente actualizables, que se pueden adaptar a cualquier situación a través del programa ejecutado, así como ampliar sus cualidades a través de una actualización de software.
- Rápidos y capaces computacionalmente.
- Baratos y de bajo consumo.
- Fáciles de mantener dada la estandarización de los lenguajes, la disponibilidad de personal cualificado es muy amplia.

Estas características los hacen muy interesantes para el desempeño de tareas críticas de seguridad, en ejemplos como la administración de insulina en un paciente diabético o el control de estabilidad de una aeronave. Sin embargo, todos los sistemas electrónicos están afectados por fallos, y sería una temeridad ignorar esta realidad. Es mejor aceptar que ocurrirán y tratar de evitar que sus consecuencias sean un problema:

Existe una disciplina que estudia los orígenes y los tipos de fallos que se producen en los sistemas tecnológicos, tanto desde el punto de vista del control automático, como de la electrónica subyacente. Este es, entre otros, el campo de estudio del grupo de Diseño Microelectrónico y Aplicaciones (DMA) de la Universidad Carlos III de Madrd: la tolerancia a fallos.

En este capítulo se desarrollan los conceptos y técnicas de tolerancia a fallos en circuitos digitales, analizando su clasificación y haciendo hincapié tanto en sus causas como en sus consecuencias, sobre todo en lo referente al campo de fallos transitorios donde se enmarca este trabajo. Para ello, se ha realizado una exhaustiva enumeración de los subtipos de fallos transitorios así como de las técnicas más extendidas para su gestión.

Posteriormente se introduce el concepto de interfaz de traza, y se justifica su empleo en este trabajo. El uso de esta interfaz constituye una de las corrientes de investigación más recientes y prometedoras en el ámbito de técnicas de endurecimiento hardware, y que forma parte de las líneas de investigación del Grupo de Diseño Microelectrónico y Aplicaciones del Departamento de Tecnología Electrónica de la Universidad Carlos III de Madrid.

A continuación se contemplan las distintas formas de implementar un sistema que permita desarrollar las técnicas de observación mediante la interfaz de traza, centrando la atención en los sistemas integrados con microprocesador (empotrados), para comparar sus prestaciones con otras opciones y justificando su empleo en este trabajo.

La información provista en cada uno de los apartados anteriores desemboca en un abanico de soluciones alternativas, que deben ser valoradas para justificar la elección de unas u otras. Presentar de forma clara y sintetizada las distintas alternativas de diseño es necesario para poder decantarse entre ellas.

Por último, a partir de las diferentes alternativas, es preciso delimitar las especificaciones que requiere el diseño que se desea realizar, para lo que se han planteado distintos requisitos, que permiten analizar y justificar el empleo de la solución finalmente elegida.

2.2 Tolerancia a fallos

Se conoce como tolerancia a fallos las técnicas que otorgan a los sistemas la capacidad de recuperar su estado de funcionamiento tras un error, manteniendo la calidad del servicio que ofrecen. Esta característica es fundamental cuando el sistema electrónico estudiado realiza una operación crítica, en la que la fiabilidad es estrictamente necesaria.

El concepto de fiabilidad alude al grado de inmunidad del sistema frente a los fallos, sin embargo no debe malentenderse con la ausencia de ellos: un sistema con baja tasa de fallos, pero que no dispone de mecanismos para recuperarse de los mismos no es un sistema fiable; siendo más fiable un sistema con mayor tasa de fallos, pero que ha sido endurecido para recuperarse de un alto porcentaje de ellos.

Es por tanto, igualmente importante detectar la aparición de fallos en el sistema, como corregir sus consecuencias.

Se define fallo [1] como la manifestación de un error. Se distinguen distintos tipos atendiendo a diferentes criterios:

En relación a la fase del ciclo de producto en el que se han introducido los errores:

- Diseño. Errores de especificación.
- Fabricación. Errores en materiales o fallos de las máquinas involucradas.
- Operación. Ocasionados por un uso incorrecto del dispositivo.

En relación a su causa:

- Humanos. Originados en alguna de las fases tanto de diseño, fabricación o uso en las que intervienen personas.
- Naturales. Fallos producidos por el entorno, sin participación del ser humano.

En relación con las fronteras del sistema:

- Internos. Su origen se encuentra dentro del propio dispositivo. Un ejemplo es el desgaste.
- Externos. Se producen desde el exterior.

En relación a la parte del circuito afectada:

- Hardware. Anomalías en el nivel físico del circuito, ya sea en la parte digital o analógica.
- Software. Errores en el código ejecutado por un procesador.

En relación con la duración de sus efectos:

- Fallo permanente en el circuito: indica un defecto en algún componente del hardware o software a bordo del mismo que inutiliza su funcionamiento.
- Fallo intermitente: es el preámbulo de un fallo permanente. El error subyacente aún no ha inutilizado el sistema por completo, y se manifiesta en períodos de duración limitada no periódica.
- Fallo transitorio: manifestación de errores producidos por efectos ambientales relacionados con entorno de funcionamiento del circuito digital, como puede ser ruido electromagnético o radiación cósmica [2], presente en cualquier lugar. Se conocen como *soft errors* [3]. Si bien es cierto que cuando las partículas cargadas de la radiación cósmica impactan en un circuito pueden desencadenar un fallo físico, su efecto más habitual es una única alteración puntual en su funcionamiento, sin producir daños permanentes en el circuito.

  Se puede catalogar un fallo transitorio como un error de operación en la parte digital del hardware, de origen natural y externo.

  En el marco de este trabajo se van a desarrollar técnicas para robustecer sistemas electrónicos frente a fallos transitorios, por lo que es conveniente detallar los distintos subtipos:

  - *Single-Event Upset*, SEU. Ocurren cuando la partícula interfiere con el funcionamiento de un biestable en un registro o memoria, modificando su valor hasta el siguiente ciclo de reloj.
  - *Single-Event Transient*, SET. En este caso, una partícula impacta en un transistor perteneciente a lógica combinacional, originando un pulso erróneo. En las tecnologías nanométricas, la duración de este pulso es comparable al retardo de una puerta, por lo que su efecto se puede propagar a través del circuito y llegar a almacenarse en uno o varios registros.
  - *Multiple-Bit Upset*, *Multiple-Cell Upset* (MBU/MCU). La creciente miniaturización del tamaño de los transistores posibilita que el efecto de una misma partícula afecte a más de una señal combinacional o registro. Cuando la partícula afecta solamente a lógica combinacional se denomina MBU, así como cuando todos los biestables afectados pertenecen a un único registro. El término MCU está reservado a los efectos producidos por una sola partícula en diferentes registros.

- *Single-Event Functional Interrupt*, SEFI. Cuando el impacto se produce sobre un registro crítico que afecta a un sistema de control, además de alterar su valor, se produce una interrupción en su funcionamiento. Esto ocurre por ejemplo cuando afecta a la lógica de la señal de reset. En los casos anteriores, el efecto del error transitorio puede no manifestarse, sin embargo en este caso siempre se produce un fallo, y la única solución es reiniciar la aplicación.
- *Single-Event Latch-up*, SEL. Se refiere al efecto del impacto de la partícula sobre circuitos con tecnología CMOS (*Complementary Metal-Oxide-Semiconductor*) con sustrato, cuando la carga generada activa los transistores bipolares parásitos existentes entre las distintas zonas dopadas originando un camino entre alimentación y masa que provoca un cortocircuito. Se trata del fallo más peligroso, ya que puede inutilizar permanentemente el dispositivo afectado, siendo su única solución desconectarlo de la alimentación. Sin embargo, también se puede prevenir utilizando tecnología CMOS de substrato aislado que elimina los transistores parásitos, aunque dado sus elevados costes de fabricación solamente se utiliza en aplicaciones espaciales críticas.

En el contexto del presente Trabajo Fin de Grado, un fallo es cualquier comportamiento anómalo en el microprocesador utilizado. Los fallos más comunes son los que afectan a:

- Datos del programa. Si el impacto de la partícula produce un cambio en una variable del banco de datos, los resultados de las operaciones realizadas serán incorrectos, como también lo serán las acciones realizadas en base a esos datos.
- Flujo del programa. Si ocurre en un registro de control, como el PC (*Program Counter*) o el *stack pointer*, se pierde el control sobre el flujo, pudiendo quedar el microprocesador bloqueado en un bucle infinito.

Se han descrito múltiples estrategias que se pueden enmarcar dentro de tres grandes grupos, a saber técnicas basadas en software, hardware o híbridas:

- Técnicas basadas en software. El código ejecutado por el microprocesador se modifica para aumentar su robustez frente a fallos transitorios.
  - Para control de datos: se aplica lo que se denomina redundancia a nivel de código, ya sea modificando directamente el ensamblador o utilizando herramientas de transformación con un código de alto nivel. Las técnicas más utilizadas son la duplicación de instrucciones, duplicación de variables [4] y el uso de multiprocesos redundantes (*multithreading*).
  - Para control de flujo, también llamadas *control-flow checking* (CFC): se introducen en el código nuevas instrucciones persiguiendo el objetivo de que el procesador pueda discernir si ha ocurrido un error que pueda afectar al flujo de ejecución y evitarlo. Hay dos tendencias diferentes:
    - Firmas: El código se divide en *Branch-free Blocks* (BBs), en español bloques libres de salto, esto es, un conjunto de instrucciones que el procesador siempre a ejecutar de forma simultánea, siendo la única que puede implicar un salto, la última. A cada BB se le asigna una firma única, usualmente un número primo. La información de las firmas se utiliza para detectar anomalías en el flujo del programa analizando el orden de ejecución de los distintos BBs y comparándolo con una serie de datos previamente generados, los cuales aumentan el consumo de memoria del sistema [5].
    - Saltos invertidos (*Inverted Branches*): Esta técnica duplica todas las condiciones de salto para evaluar cada una en dos ocasiones. En la primera se evalúa si el resultado de la primera condición es positivo, y en la segunda, si el resultado de la misma condición negada es negativo. Si ambos criterios se cumplen, la condición de salto está exenta de fallos y puede ejecutarse [4].
- Técnicas basadas en hardware. Implican modificaciones o ampliaciones del hardware que permiten prevenir o corregir fallos transitorios.
  - En la mayoría de los casos son soluciones basadas en sistemas redundantes: múltiples procesadores, coprocesadores u otros módulos especializados. Se conocen como *watchdog processor* (procesador vigía).
    - *Watchdog processor* activo: Otro procesador ejecuta un programa concurrentemente con el microprocesador observado. Esto introduce un nuevo microprocesador en el sistema, incrementando mucho el área y la complejidad, sin embargo garantiza total redundancia por lo que la tasa de detección de errores es muy alta. Las arquitecturas con tres o más procesadores y un circuito votador permiten, además de detectar los errores, corregirlos.

- *Watchdog processor* pasivo: utiliza la misma técnica de los bloques libres de salto (BBs) descrita en la sección de software, pero analizando las firmas de manera externa. Es una solución de menor tamaño que el *watchdog processor* activo, pero aun así requiere almacenar gran cantidad de información. Además requiere modificar el código para introducir las firmas.

❖ En ambos casos, el *watchdog processor* se debe conectar al bus entre la memoria y el procesador, siendo un procedimiento delicado, y en ocasiones imposible debido a la arquitectura de cada microprocesador.

Mediante técnicas hardware es posible detectar y corregir errores tanto en los datos como en el flujo de ejecución, al acceder a los siguientes parámetros del microprocesador:

- Valor del PC. Obtiene la dirección apuntada por el contador de programa así como por el puntero de la pila (*stack pointer*). El valor del PC es muy útil para realizar comprobaciones del flujo de ejecución del programa, mediante técnicas como la predicción de PC [6].
- Valor del opcode. Contiene el código de la instrucción ejecutada, permitiendo conocer tanto el tipo de instrucción como sus atributos. Con esta información se puede comprobar:
  - Si los resultados de las operaciones realizadas son correctos con aplicaciones tanto en control de datos (resultados aritméticos) como de flujo (resoluciones de condiciones de salto).
  - Si el código de operación no se ha visto alterado durante el pipeline [7].
- Resultados de la instrucción. Recopila información relacionada con los registros de estado del microprocesador.
- Datos de programa. Permite acceder directamente al contenido de los registros de datos para leerlos o modificarlos.

- Técnicas híbridas. Combinan las características más favorables de cada una de las anteriores [6], [7]. Para discernir la idoneidad de cada una, se deben analizar sus ventajas e inconvenientes:
  - Técnicas software: pueden utilizarse directamente en COTS al no requerir modificaciones de hardware, pero penalizan el tiempo de ejecución, siendo más pesadas las técnicas de control de flujo con firmas que la redundancia de datos.
  - Técnicas hardware: son muy rápidas pero pueden penalizar el tamaño del circuito conforme aumenta su capacidad de detección. Además suelen requerir modificaciones internas en la arquitectura del microprocesador, lo que no permite su uso con COTS, incrementando el coste de fabricación y reduciendo el mercado de aplicación.

2.3 Interfaz de traza y observación no intrusiva

Recientemente se han propuesto técnicas hardware que emplean la interfaz de traza de los microprocesadores como canal de observación [8]. La interfaz de traza es un recurso que incluyen la gran mayoría de procesadores modernos y sirve para dar soporte al proceso de desarrollo y depuración de las aplicaciones. Sin embargo, una vez finalizado este proceso, esta interfaz ya no tiene utilidad, de modo que se pueden reutilizar para realizar monitorización en línea sin suponer un incremento de coste. Su interés radica en que, por definición, otorga acceso al microprocesador desde el exterior sin modificar su hardware ni afectar al tiempo de ejecución del software.

El empleo de la interfaz de traza permite acceder a los mismos recursos que un *watchdog processor*, lo que significa que se pueden aplicar las mismas técnicas de detección y corrección de errores que las descritas en el apartado anterior, pero modificando sustancialmente el modo de acceso a la información.

Es importante destacar que el proceso de diseño de un microprocesador es una tarea que requiere de un gran esfuerzo por parte de un equipo humano y técnico e involucra grandes inversiones económicas de prototipado y depuración, por lo que las soluciones basadas en la modificación de las arquitecturas preexistentes deben evitarse [8].

Para capturar la información procedente de la interfaz de traza se usan módulos IP (*Intellectual Property*) diseñados para ser capaces de seguir el flujo de datos y de ejecución [5], [6], [8] . Estos módulos se pueden integrar en el proceso de fabricación sin alterar el diseño del microprocesador utilizado, logrando su observación no intrusiva, y garantizando la compatibilidad con COTS, evitando añadir costes al ya de por sí costoso proceso de diseño de un microprocesador.

En resumen, el empleo de la interfaz de traza mejora las capacidades de las técnicas hardware ya que no requiere realizar modificaciones en el hardware, permitiendo el uso eficiente de los recursos ya existentes en el microprocesador de manera no intrusiva, y constituyendo una sólida alternativa a los *watchdog processors*, gracias a sus mayores posibilidades de ampliación y menores costes asociados. De esta manera se pueden utilizar técnicas hardware en sistemas cuyas arquitecturas no se pueden modificar, flexibilizando las opciones de diseño de la tolerancia a fallos al unirse a las técnicas software existentes para confeccionar técnicas híbridas no intrusivas. Actualmente ya se están desarrollando trabajos con la interfaz de traza en procesadores como el PicoBlaze [6], el LEON3 [7] o el MiniMIPS [5].

En este caso, las técnicas de observación no intrusiva se aplicarán en un procesador menos estudiado, como es el ARM Cortex-A9, como parte de una línea de investigación del Grupo de Diseño Microelectrónico y Aplicaciones del Departamento de Tecnología Electrónica de la Universidad Carlos III de Madrid. Se persigue diseñar un módulo observador capaz de proveer información suficiente para robustecerlo frente a fallos transitorios que provoquen una alteración en el flujo de ejecución del programa.

2.4 Sistemas empotrados

En términos estrictos, se define como sistema empotrado (*embedded system* en inglés) a *"un sistema computacional optimizado para llevar a cabo una, o un número determinado de aplicaciones dedicadas"* [9, p. 173]. Sin embargo, se puede extender su uso a los sistemas que pueden dar lugar a un sistema empotrado.

Un rasgo característico de los sistemas empotrados es la implementación en un único circuito integrado de un microprocesador (capacidad computacional) y una serie de recursos adicionales que optimizan su operación para determinadas funciones. En el presente caso, se puede considerar un sistema empotrado a un microprocesador unido a un módulo observador diseñado para realizar las funciones de detección y corrección de errores transitorios.

El empleo de sistemas empotrados está justificado tanto en términos económicos como tecnológicos:

- Los sistemas en un único chip, SoC (*System on Chip*) minimizan los costes de desarrollo al reducir todos los conceptos a un único producto que diseñar, probar y fabricar.
- De la misma manera, los SoC maximizan las prestaciones al permitir elevadas frecuencias de funcionamiento gracias a la reducción de la longitud de las interconexiones al mismo tiempo que se garantiza una alta calidad en las mismas.

2.4.1 Sistemas empotrados programables

Los sistemas en chip programable o SOPC (*System On Programmable Chip*) por sus siglas en inglés son dispositivos capaces de implementar físicamente cualquier sistema empotrado mediante programación:

- Programación software con lenguajes de programación como ensamblador o C. Se definen secuencias de operaciones y subrutinas que ejecuta el microprocesador.
- Programación mediante lenguajes de descripción hardware, o HDL (*Hardware Description Language*). A través de un fichero de texto permiten describir el comportamiento que debe tener un circuito que posteriormente se sintetiza. De ello se desprende que utilizan tecnología de lógica programable o FPGA (*Field-Programmable Gate Array*).

Son un buen entorno de desarrollo ya que permiten validar el funcionamiento de todos los tipos de técnicas de endurecimiento frente a fallos transitorios ya sea introduciendo los errores de forma virtual [4], [5], [7] , aprovechando las posibilidades que otorga la FPGA, o sometiendo al sistema resultante a ensayos reales de radiación [10].

Para su denominación como sistema empotrado, ya se ha puntualizado que es un requisito necesario la incorporación de un microprocesador. Se distinguen dos escenarios:

- *Soft-core*. La FPGA no implementa a priori ningún microprocesador, sino que este se programa en su interior, así como los recursos adicionales mediante lenguajes de descripción hardware. Permiten una gran flexibilidad de implementación, pudiendo activar y desactivar recursos para optimizar el tamaño del circuito resultante. Además, algunos de ellos dan la posibilidad de conectarse indiscriminadamente a sus recursos internos o externos. Sin embargo, los procesadores distribuidos de esta manera son escasos:
  - La mayor parte de microprocesadores *soft-core* disponibles son desarrollados por las mismas empresas que fabrican las FPGAs, para su uso en conjunto. Algunos ejemplos de este caso son el MicroBlaze o el PicoBlaze [6] de Xilinx o el Nios II de Altera.
  - Adicionalmente, existen otros microprocesadores en versión *soft-core* para su desarrollados por terceros, como el LEON3 [7] utilizado en aplicaciones aeroespaciales, o el MiniMIPS [4].
- *Hard-core*. El circuito integrado que contiene la FPGA, está compartido también por uno o varios procesadores preexistentes sobre el silicio. La gran ventaja de esta configuración es que permite integrar microprocesadores más complejos en un área mucho menor; por contra, el acceso a recursos internos y la escalabilidad de los *soft-core* no están disponibles. Un ejemplo es la familia ZYNQ [11] de Xilinx.

No existe a nivel funcional ninguna diferencia entre el resultado de una u otra solución pudiendo realizar el diseño del módulo observador indistintamente en cada una, tan solo atendiendo en cada caso a las restricciones de diseño ya mencionadas.

Por tanto, el empleo de una u otra solución dependerá en gran medida de las necesidades del diseño así como de la disponibilidad de recursos para cada caso.

2.5 Alternativas de diseño

El planteamiento del estado de la técnica en cuanto a tolerancia a fallos para sistemas basados en microprocesadores genera distintas vías para lograr el objetivo del endurecimiento del sistema final frente a fallos transitorios. Sintetizando las ideas principales podemos distinguir dos variables:

Por una parte distingue entre la plataforma sobre la que se aplican las técnicas:

- Hardware: permiten circuitos resultantes más rápidos, y la tasa de detección aumenta a la vez que su tamaño. Requieren realizar modificaciones en el circuito original del microprocesador, salvo que se utilice la interfaz de traza como canal.
- Software: no requieren modificar el hardware, ni conocer su arquitectura. La tasa de detección aumenta a la vez que se reduce el tiempo de ejecución. Sobre todo cuando el objetivo es robustecer el control de flujo de programa.

❖ La combinación de las características más ventajosas de cada una configura las técnicas híbridas. Generalmente se utilizan técnicas software para endurecer los datos, lo que no penaliza en exceso el tiempo de ejecución en conjunto con técnicas hardware para endurecer el control de flujo, por ejemplo mediante un predictor de PC [6].

Por otra sobre las características que se quieren proteger:

- Datos del programa: los errores pueden producir resultados y comportamientos erróneos en el sistema.
- Flujo de ejecución del programa: los errores pueden producir comportamientos erróneos en el sistema que lo pueden conducir a un bucle infinito sin retorno.

❖ La necesidad de robustecer los datos dependerá lo crítica que sea la aplicación desarrollada, cabiendo la posibilidad de que todas, o al menos algunas de ellas no necesiten endurecimiento. Sin embargo, el control de flujo de ejecución es necesario en la totalidad de los casos, ya que un fallo en él llevaría al procesador a un estado del que no podría salir por sí mismo.

Y por último, la plataforma utilizada para implementarlo.

- Montar el circuito sobre una placa directamente con COTS: requiere de un gran trabajo de selección de los componentes (microprocesador, FPGA y otros auxiliares), así como para el montaje de la placa, siendo extraordinariamente delicados los temas de retardos y compatibilidad de entradas y salidas.
- Utilizar un sistema empotrado: incrementa la velocidad de funcionamiento, la calidad de las conexiones y la fiabilidad total del conjunto.

Los objetivos del trabajo marcan que el módulo observador diseñado será una solución hardware para endurecer el flujo de ejecución.

2.6 Requisitos del sistema

Se debe configurar un sistema formado por dos componentes imprescindibles:

- Por un lado, un microprocesador Cortex-A9 a observar. Es una condición necesaria que disponga de interfaz de traza, o que se le pueda añadir una de manera externa.
- Por otro lado, lógica programable suficiente para implementar un módulo observador de pequeño tamaño.
- Adicionalmente, en caso de que ambos componentes se suministren por separado será necesario añadir los dispositivos necesarios para su correcta interconexión.

Existe en el mercado una amplia gama de FPGAs de distintos fabricantes. Las principales características que diferencian unas de otras son:

- Por una parte la capacidad, es decir, la cantidad de celdas que contiene, las cuales determinarán el tamaño del circuito más grande que son capaces de implementar.
- Por otra, lado la velocidad a la que la información puede viajar por su interior, que limitará la frecuencia máxima a la que puede funcionar cada implementación.
- Adicionalmente, existen otros parámetros a tener en cuenta como la tecnología utilizada en su fabricación.

La tecnología de FPGAs se encuentra en un alto grado de desarrollo, y en la actualidad encontramos fácilmente modelos en fabricación que se adaptan a nuestras necesidades. Sin embargo, la arquitectura Cortex-A9 no está disponible en versión *soft-core*, circunstancia que condiciona la elección de la plataforma a utilizar, ya que, como se ha razonado en el apartado anterior, en el caso de no poder utilizar un sistema empotrado, la complejidad de la tarea aumentaría considerablemente.

Por suerte, Xilinx ha lanzado recientemente una nueva familia de SOPC llamada ZYNQ [12], que incorpora junto a una porción de lógica programable, uno o más procesadores ARM en implementación *Hard-Core*. Se trata de una solución mucho más sencilla de implementar, al estar solo es necesario encontrar un sistema de traza preinstalado, o una manera de añadirlo utilizando la lógica programable disponible.

Existen diferentes modelos y versiones de ZYNQ, destacando las denominadas ZYNQ Ultrascale, que poseen mayores prestaciones. Sin embargo dado su elevado precio, no es justificable su uso para realizar un estudio inicial de las posibilidades del sistema, por lo que el foco del trabajo se limitará al uso de la familia ZYNQ-7000 [11] [13], de prestaciones más reducidas, pero suficientes.

Así, el problema se abordará diseñando un módulo observador externo no intrusivo sobre un SoC de la familia ZYNQ-7000 de Xilinx. Este módulo deberá ser capaz de observar la posición del PC a través de la interfaz de traza para utilizarlo en técnicas de endurecimiento hardware o híbridas.

Capítulo 3
Sistema empleado

3.1 Introducción

En el capítulo anterior se ha tomado la decisión de utilizar la familia ZYNQ-7000 de Xilinx para el desarrollo del trabajo. Se trata de un abordaje ambicioso para resolver el problema ya que destaca la poca documentación disponible sobre el uso de la interfaz de traza en esta plataforma, así como la escasez de desarrollos realizados sobre ella.

Una vez elegida la plataforma, es necesario seleccionar un sistema de desarrollo sobre el que implementarlo. Existen en el mercado diferentes placas que montan una ZYNQ-7000, y entre ellas destaca la ZYBO [14], de Digilent, destaca por ser la solución más económica sin penalizar en exceso las prestaciones, que para el objetivo de este trabajo son más que suficientes.

La elección de una placa determinada condiciona las especificaciones del sistema completo sobre el que se va a realizar el diseño, ya que monta un chip específico. Caracterizar estas prestaciones [15] es imprescindible para conocer los detalles constitutivos y funcionales de la plataforma en la que se basará todo el desarrollo del trabajo. Así pues, se incidirá en las características detalladas tanto de la lógica programable como del microprocesador, integrados en un SoC de la familia ZYNQ-7000 de Xilinx.

Se realizará especial énfasis sobre la ubicación y características de la interfaz de traza CoreSight incorporada en el microprocesador, identificando cada uno de sus componentes y localizándolo en el contexto del sistema a través de la documentación disponible por parte de las tres empresas involucradas:

- ARM: Creador del procesador Cortex-A9 y de la interfaz de traza CoreSight.
- Xilinx: Fabricante del chip que integra el microprocesador junto a la lógica programable (FPGA).
- Digilent: Fabricante de la placa ZYBO.

El análisis de la documentación permite discernir de entre las distintas unidades que componen el ecosistema de traza y depuración CoreSight, cuáles son los más apropiados para obtener la información que necesitará el módulo observador durante su funcionamiento.

Por último, se detallarán las principales características de las herramientas informáticas específicas necesarias para trabajar con los elementos citados anteriormente y que se detallan en los siguientes apartados.

3.2 Placa utilizada

3.2.1 Familia ZYNQ de Xilinx

ZYNQ [12] es el nombre que da Xilinx a su recientemente lanzada familia de SoC programables o SOPC. Se caracterizan principalmente por incluir dentro del mismo circuito integrado, no solamente una FPGA, sino además otros bloques *hard-core* como un microprocesador Cortex-A9 de doble núcleo, un Cortex-A53 de cuatro núcleos, o incluso una GPU (*Graphics Processing Unit*), dependiendo del modelo.

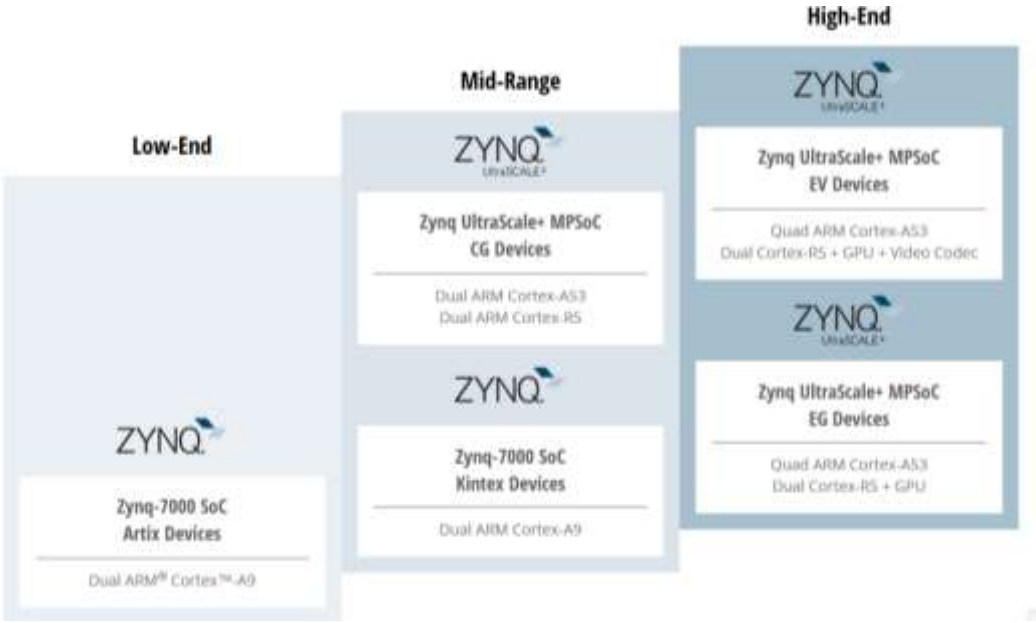


Figura 3-1 Productos de la familia ZYNQ [12]

Esta integración supone un incremento de las posibilidades que ofrecerían los distintos componentes por separado. Xilinx denomina a esta gama *"All Programmable"* (todo programable), al otorgar al usuario el control total sobre su funcionamiento. No solamente es posible programar el comportamiento de un microprocesador ARM mediante código fuente, o describir en HDL la implementación de un circuito digital sobre una FPGA, sino que además, es posible establecer internamente las interconexiones realizadas entre ellos. Ello aumenta notablemente la velocidad de transmisión de datos entre ambas partes, que sería impensable de utilizarse un chip distinto para cada una. Además, al incluir todos los sistemas en el mismo circuito integrado logramos un de menor área, y por tanto de menor precio.

La gran complejidad que implica un SoC de estas características obliga a que para diseñar tanto el hardware como el software que posteriormente se implementarán sobre la ZYNQ, sea preciso el uso herramientas informáticas especializadas. Una descripción de las mismas se encuentra en el apartado 3.4 Entorno de desarrollo utilizado.

3.2.2 ZYBO

Durante el desarrollo del trabajo se ha utilizado la placa ZYBO [14] [15], nombre procedente de las palabras ZYnq BOard, distribuida comercialmente por la empresa Digilent, cuyo aspecto se corresponde con el de la siguiente figura.

Figura 3-2 Placa ZYBO [14]

Es una placa que destaca principalmente por ser la más económica de cuantas implementan una ZYNQ. Está equipada con gran variedad de puertos.

- Gigabit Ethernet.
- HDMI de doble sentido (fuente y receptor).
- VGA fuente de 16 bits por píxel.
- Ranura para MicroSD con soporte para ejecutar Linux [9].
- USB 2.0 OTG (*On The Go*) maestro y esclavo.
- Conectores de audio y micrófono.
- 4 interruptores, 6 pulsadores y 5 LEDs incorporados.
- 6 conectores PMOD (*Peripheral MODules*) de expansión de 8 señales cada uno.
- USB de programación.

Es posible alimentarla a través del conector USB de programación, aunque para aplicaciones que requieran una demanda de potencia moderada se recomienda utilizar el adaptador de corriente que suministra. Adicionalmente incorpora las conexiones necesarias para una batería si se va a utilizar para desarrollar una aplicación móvil.

3.3 Características del sistema

La placa ZYBO incorpora un integrado de la familia ZYNQ-7000, en concreto el Z-7010 que es el más bajo de la gama [13] [11]. No obstante, comparte la gran mayoría de las especificaciones con sus hermanos mayores, siendo las principales diferencias:

- El tamaño de la lógica programable.
- La frecuencia máxima del reloj del procesador.

Dado que el módulo observador diseñado debe cumplir el objetivo de ser una solución de tamaño reducido, las prestaciones aportadas por el Z-7010 deberían ser más que suficientes.

Todos los ZYNQ-7000 comparten la misma arquitectura, que puede verse en la siguiente imagen.

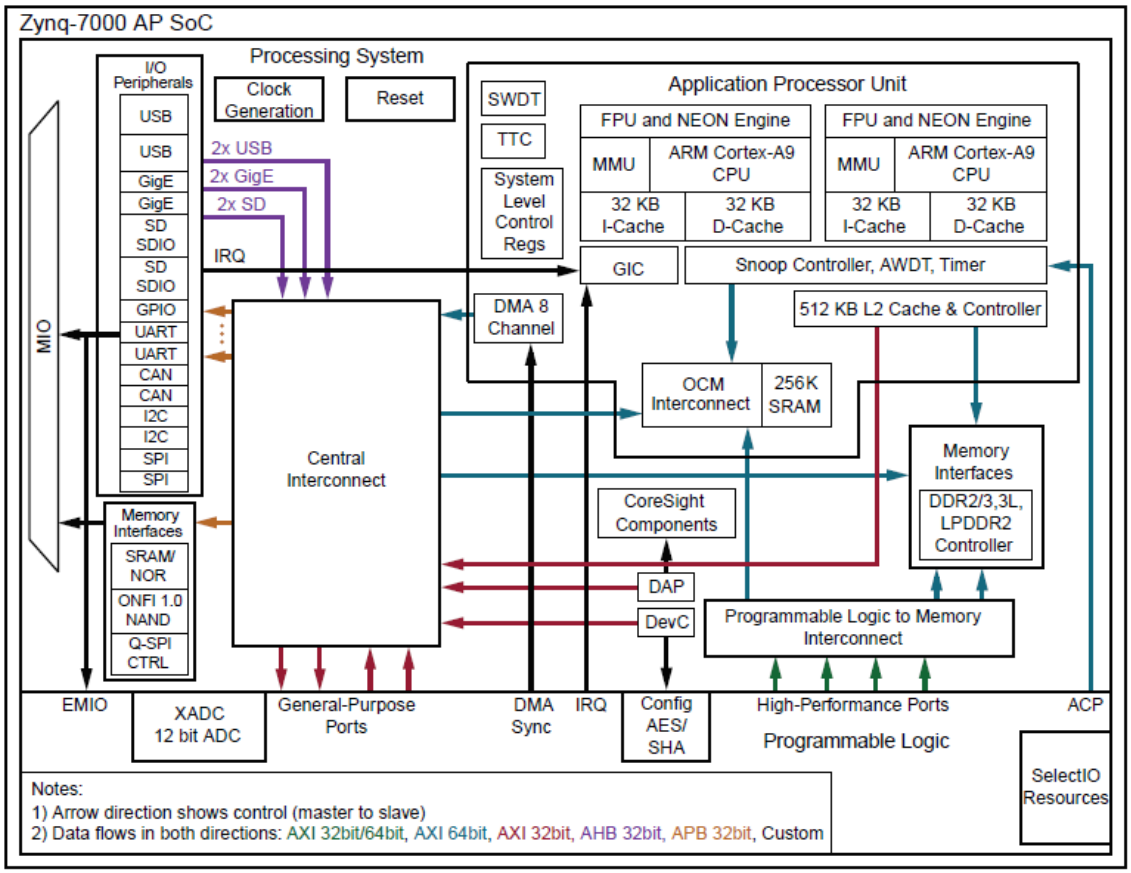


Figura 3-3 Vista general de componentes de una ZYNQ-7000 [16, p. 27]

Se pueden distinguir dos grandes grupos de componentes, los pertenecientes a la lógica programable y los pertenecientes al microprocesador. A continuación se detallan las especificaciones de cada uno.

3.3.1 Lógica programable

En la Figura 3-3 se sitúa en la parte inferior, marcada como *Programmable Logic* (PL). Como ya se ha comentado, el tamaño de la lógica programable de una ZYNQ-7000 está estrictamente relacionado con el modelo exacto, en este caso se trata del Z-7010, que cuenta con las siguientes características:

- Lógica programable equivalente a una FPGA Artix-7.
  - 4400 *logic slices* (divisiones lógicas) con 8 biestables y 4 LUTs de 6 entradas.
  - 240 kB en bloques de RAM.
  - Dos células de gestión de reloj, cada una con un PLL (*Phase Locked Loop*).
  - Conversor analógico digital incorporado (XADC).
- El chip cuenta con 400 pines, de los cuales, 100 pueden ser utilizados por la PL.
- Soporta frecuencias de reloj internas de más de 450 MHz.

3.3.2 Microprocesador

En la Figura 3-3 se sitúa en la parte superior, marcado como *Processing System* (PS). En torno a un procesador de doble núcleo Cortex-A9 MPCore [17] a 650MHz se agrupan, entre otros:

- Una unidad SIMD (*Single Instruction, Multiple Data*) NEON para cada núcleo. Permite realizar múltiples operaciones matemáticas, a la vez. Por ejemplo, es posible realizar cuatro sumas, cada una con distintos sumandos, al mismo tiempo.
- Una FPU (*Floating-Point Unit*) para cada núcleo. La unidad de coma flotante, es un subsistema especializado en realizar cálculos matemáticos con decimales.
- 8 canales DMA(*Direct Memory Acces*), El acceso directo a memoria es la tecnología que permite a los periféricos del microprocesador hacer uso de la memoria de datos de éste de manera independiente, mientras el núcleo se encarga de otra tarea distinta. Esta técnica permite maximizar la velocidad de operaciones repetitivas, eximiendo al núcleo de su atención.
- Puertos de comunicaciones externas preinstalados:
  - 2x UART (Universal Asynchronous Receiver-*Transmitter*).
  - 2x CAN (*Controller Area Network*).
  - 2x I2C (*Inter-Integrated Circuit*).
  - 2x SPI (*Serial Peripheral Interface*).
  - 4x GPIO (*General-Purpose Input / Output*) de 32 bits.
  - 2x USB 2.0 OTG.

  - 2x Gigabit Ethernet.
  - 2x SDIO (*Secure Digital Input Output*), para tarjetas SD.
- Interfaces de comunicación con la lógica programable:
  - 2x AXI (*Advanced eXtensible Bus*) Maestro de 32 bits.
  - 2x AXI Esclavo de 32 bits.
  - 16 fuentes de interrupción.

Debido a la arquitectura de la familia ZYNQ, el asunto de las interconexiones es precisamente interesante, ya que se desdobla en las dos posibilidades diferentes:

- Interconexiones internas del PS. No se ven afectadas por la implementación, el microprocesador embebido en la ZYNQ conserva la misma arquitectura que si estuviera implementado en un chip para él solo, esto es un COTS.
- Interconexiones del PS con el exterior. El microprocesador está integrado junto a la lógica programable, lo que significa que su contorno no está directamente conectado con los pines físicos de la ZYNQ: salvo las señales imprescindibles (programación, reloj, reset, etc…), el resto de las conexiones están "flotando", a la espera de que se configure una ruta para cada una.
  - Conexiones del PS con el exterior: son limitadas, ya que necesitan ser rutadas por un pin físico dedicado de la ZYNQ, habiendo solamente 54 pines disponibles. Se las denomina MIO (*Multiplexed Input / Output*).
  - Conexiones del PS con la PL: son ilimitadas, el número máximo será de tantas como puertos disponga el microprocesador, más de 3000 conexiones que garantizan la total integración del hardware diseñado por el usuario con el procesador. Se las denomina EMIO (*Extended Multiplexed Input / Output*). Dado que la lógica programable tiene pines externos reservados por lo que también existe la posibilidad de rutar las conexiones del PS hacia el exterior a través de la PL, logrando sacar un número de señales externas superior a 54.

3.3.3 Interfaz de traza

El subsistema sobre el que se enfoca este trabajo recibe el nombre de CoreSight [18]. Este nombre agrupa un amplio conjunto de tecnologías y arquitecturas que ARM ha desarrollado para permitir la depuración (*Debug*) y la traza (*Trace*) de diseños SoC.

La tecnología CoreSight provee los recursos necesarios para la depuración y la traza de sistemas completos de gran complejidad, con varios procesadores y altas frecuencias de reloj, gracias a su gran ancho de banda.

Los conceptos de depuración y traza hacen referencia tanto a características implementadas en el diseño como a las técnicas empleadas en su utilización, sin embargo su propósito es muy distinto:

- Depuración: su finalidad es observar y/o modificar el estado de una o varias partes del diseño.
  - Se realiza de forma invasiva, esto es, interfiriendo en el funcionamiento autónomo normal del sistema a través de un puerto dedicado.
    - Posibilidad de leer y modificar valores de los registros.
    - Posibilidad de congelar la ejecución inmediatamente al detectar un error, lo que facilita su corrección al capturar la instantánea del momento en que se ha producido.
- Traza: permite recolectar continuamente información de la ejecución para su análisis.
  - Se realiza de forma no invasiva, utilizando únicamente la información que el propio sistema genera para tal fin.
    - Existen subsistemas destinados específicamente para ello.
    - La información puede ser utilizada a nivel interno, o transmitirse al exterior del sistema mediante un puerto.

Se puede concluir que la diferencia más significativa entre ambos conceptos es en lo que se refiere al marco temporal, tal y como sintetiza la siguiente tabla.

Tabla 3-1 Comparativa entre depuración y traza

	Depuración	**Traza**
Perspectiva	Invasiva	No invasiva
Flujo de información	Entrada y salida	Solo salida
Observa	Estado del sistema	Ejecución
Modifica	Registros y ejecución	Nada
Acceso a información	Ilimitado	Solo la proporcionada
Marco temporal	Esporádico. La modificación de registros así como la captura de instantáneas de ejecución sólo suceden de forma puntual.	Continuo: La información se genera de manera continua, durante todo el tiempo que dure la ejecución.
Ejemplos CoreSight	*Debug Access Port*	*Embedded Trace Macrocell* *Program Trace Macrocell*

Una vez analizadas las dos posibilidades que ofrece CoreSight, es inmediato concluir que dado su carácter de observación continuada, la que mejor se adapta a los objetivos de este proyecto es la traza.

La información de traza tiene su origen en el núcleo del microprocesador, por lo que es preciso conducirla hasta su exterior donde sea posible recopilarla de manera no intrusiva. Se puede abstraer el flujo de la información de traza mediante el siguiente diagrama.

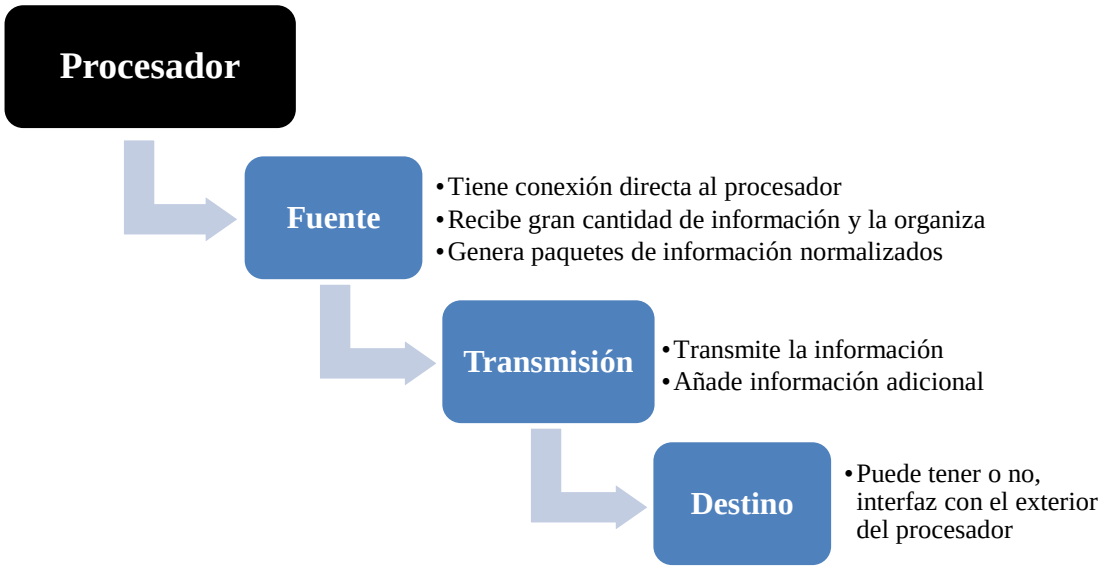


Figura 3-4 Flujo genérico de la información de traza

ARM diseña todos sus microprocesadores para ser compatibles con CoreSight. Ello supone una gran ventaja competitiva gracias a dos puntos clave:

- Es estándar. Todos los componentes tienen registros, interfaz y buses comunes, lo que facilita el desarrollo de herramientas de depuración con independencia de cuál sea el modelo de microprocesador utilizado.
- Es modular. CoreSight agrupa una gran variedad de componentes [19] con distintas prestaciones lo que facilita a los fabricantes adaptar las características de cada sistema a sus necesidades.
    - Además, gracias a la documentación disponible por parte de ARM, dichos fabricantes pueden diseñar nuevos componentes compatibles con CoreSight [20] para utilizar en sus implementaciones o realizar a la carta la configuración de los componentes que mejor se adapte a sus necesidades a partir de una librería [21].

En el caso de las FPGAs ZYNQ-7000, Xilinx incluye un componente adicional de su creación. Se trata del *Fabric Trace Monitor* (FTM) [16], que permite incorporar a los flujos de depuración y traza información relacionada con la lógica programable. En la siguiente figura se puede comprobar cómo esta nueva fuente se integra a la perfección con el resto de componentes, además de dar una visión global de la complejidad del ecosistema CoreSight instalado, tanto para depuración como para traza.

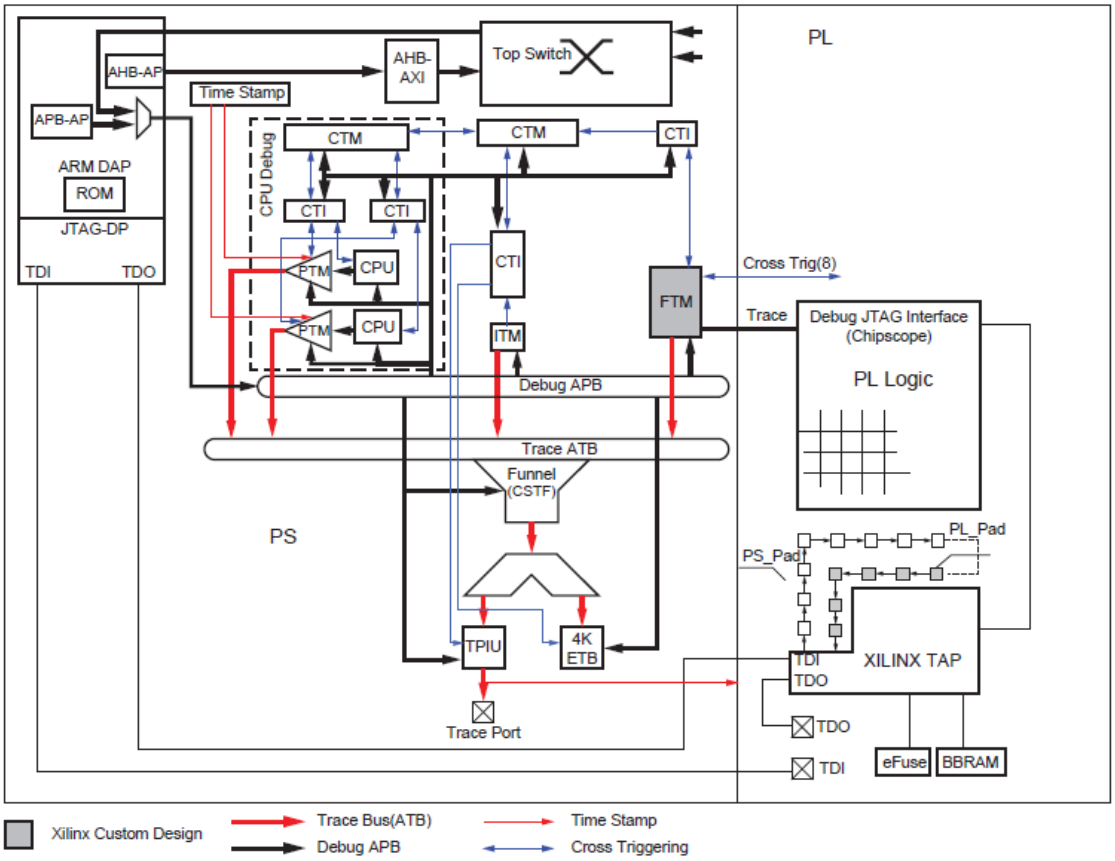


Figura 3-5 Sistema CoreSight completo a bordo de la ZYNQ [16, p. 714]

De la figura anterior se puede aislar la parte asociada al flujo de información de la traza, que se puede simplificar y reordenar tal y como puede verse a continuación.

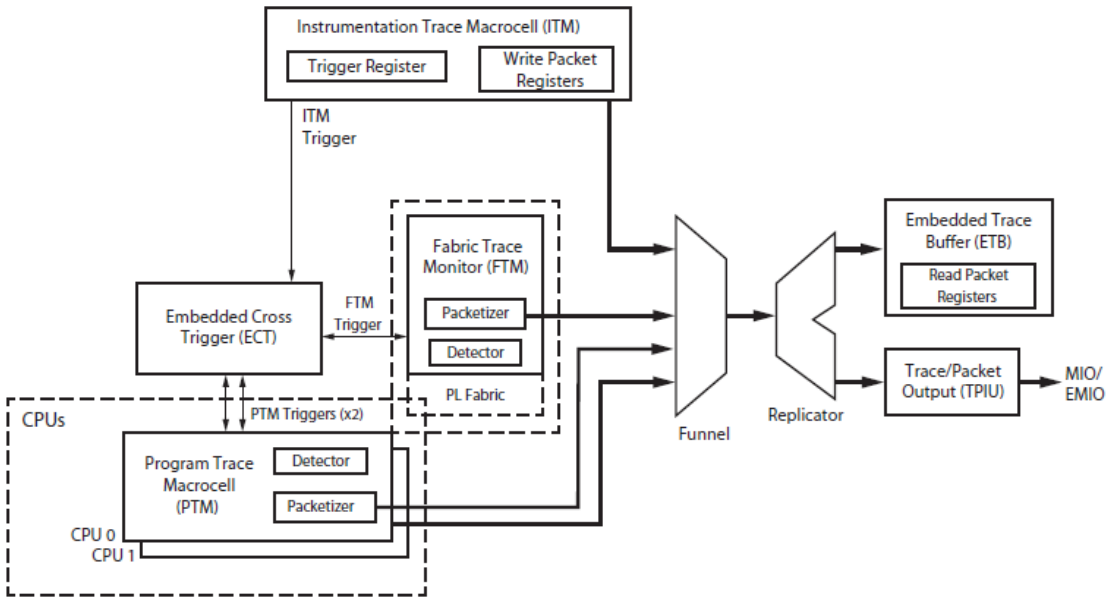


Figura 3-6 Subsistema de traza a bordo de la ZYNQ [16, p. 722]

Es posible distinguir los distintos elementos clasificándolos con los mismos criterios que en la Figura 3-4 de la siguiente forma:

- Fuentes (trace source)
  - *Program Trace Macrocell* (PTM). Genera la traza del flujo de ejecución del procesador. Reduce la cantidad de información necesaria para reconstruir la ejecución del programa a partir de puntos clave, llamados *waypoints*.
  - *Instrumentation Trace Macrocell* (ITM). Es el bloque que permite al software producir información de traza de los valores de registros o variables, gracias a los 32 registros StimPort implementados. La escritura en cualquiera de ellos desencadena paquetes traza que contienen la misma información que se escribió, además del número de registro y otros datos.
  - *Fabric Trace Monitor* (FTM). Recibe la información de traza de la PL para darle el formato de paquetes compatibles con el flujo de traza generado por la PTM o la ITM de forma que los eventos de todas ellas puedan ser trazados simultáneamente [16, p. 661].
- Transmisiones (trace link)
  - *Funnel*. Su función es recaudar la información de traza procedente de diferentes orígenes y canalizarla por una única salida. Permite activar independientemente las distintas entradas y asignarles diferentes prioridades.
  - *Replicator*. Duplica la información procedente del *Funnel* para introducirla de manera idéntica y sincronizada en la ETB y la TPIU (que se describen a continuación). No necesita configuración, y su funcionamiento es transparente [19, p. 154].
    - Cabe mencionar que las interconexiones entre las distintas entidades se realizan mediante el bus ATB (AMBA Trace Bus), indicado en la Figura 3-6 mediante las flechas más gruesas.
- Destinos (trace sink)
  - *Trace Port Interface Unit* (TPIU). Es el bloque encargado de transmitir la información de traza fuera de los límites del PS, ya sea a la PL vía EMIO o fuera del chip vía MIO. Permite configurar distintos anchos de puerto.
  - *Embedded Trace Buffer* (ETB). Se trata de una unidad de almacenamiento para la información de traza, con un tamaño de 4kB. Garantiza captura en tiempo real y a máxima velocidad para no perder ningún paquete de información.

❖ Adicionalmente, existe una entidad denominada *Embedded Cross Trigger* (ECT) [16, pp. 724 - 728], que permite a los distintos componentes de CoreSight interactuar unos con otros de manera que determinados estados en cada uno de ellos puedan desencadenar eventos en los demás.

Con esta información, ya es posible seleccionar los elementos precisos para confeccionar el flujo de traza desde el procesador hasta la PL, marcados en la siguiente tabla.

Tabla 3-2 Selección de los componentes de traza

PTM	**Sí**	**Produce la información de traza requerida en los objetivos**
ITM	No	No es ámbito de este proyecto trazar el valor de las variables
FTM	No	No es ámbito de este proyecto trazar la información de la PL
Funnel	**Sí**	**Conduce la información hacia el exterior del PS**
Replicator	**Sí**	**Conduce la información hacia el exterior del PS**
TPIU	**Sí**	**Es un puerto accesible desde el exterior del PS**
ETB	No	La información no es accesible desde el exterior
ECT	No	No es necesario para una aplicación sencilla como esta

Quedando el siguiente camino a configurar:

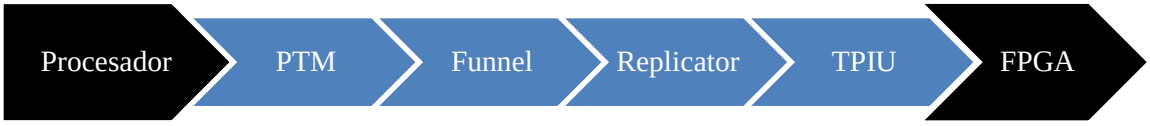


Figura 3-7 Ruta definitiva de la información de traza

3.4 Entorno de desarrollo utilizado

3.4.1 Xilinx Vivado

Vivado [22] es la herramienta más potente de desarrollo de hardware de Xilinx. Se ha empleado, en su versión 2015.3, para diseñar el sistema empotrado. Una vez creado un proyecto, proporciona las herramientas necesarias para:

- Describir el sistema a implementar mediante diagramas de bloques.
  - Utilizar bloques prediseñados por Xilinx (IP).
  - Crear nuevos bloques a partir de lenguajes de descripción hardware (HDL).
  - Crear nuevos bloques a partir de síntesis de alto nivel (HLS).
- Realizar simulaciones comportamental, funcional y temporal.
- Realizar los procesos de síntesis e implementación de un diseño.
  - Caracterizar un diseño en área y velocidad.
  - Obtener el *bitstream* del hardware, esto es obtener un fichero que describe el hardware que se va a implementar sobre la ZYNQ.
  - Implementar el hardware en una FPGA con dicho *bitstream.*
- Depurar las señales deseadas [23] en el interior de la FPGA para poder visualizar internamente las formas de onda.

Se trata de la interfaz más reciente lanzada por Xilinx con este propósito, y aglutina todas las aplicaciones que anteriormente se instalaban por separado, simplificando el proceso de diseño para aumentar la productividad del usuario. Cabe mencionar que el diseño hardware no se limita solamente a los sistemas empotrados (que incorporan microprocesador), sino a cualquier tipo de sistema combinacional o secuencial.

Es una herramienta profesional de gran implantación en la industria actual con la que se pueden desarrollar desde los ejercicios más sencillos a los sistemas más complejos. El gran interés que despierta su aprendizaje justifica su empleo para la ejecución de este Trabajo Fin de Grado.

Está dotado de los menús habituales un sistema operativo, que dan acceso a las diferentes opciones. Pero además, cuenta con una innovadora interfaz modular e interactiva, compuesta por distintas ventanas estratégicamente colocadas que se actualizan con las interacciones del usuario facilitando el proceso de diseño y contribuyendo a maximizar la productividad del desarrollador.

En la parte izquierda se ubica el *Flow Navigator*, que agrupa los accesos directos a las diferentes fases del diseño, en la parte inferior se sitúa una zona reservada para recoger los resultados de los distintos procesos automáticos de desarrollo y mostrar su estado actual, y finalmente la ventana del diseño, que además de una representación gráfica del mismo cuenta con un listado de los distintos componentes que contiene.

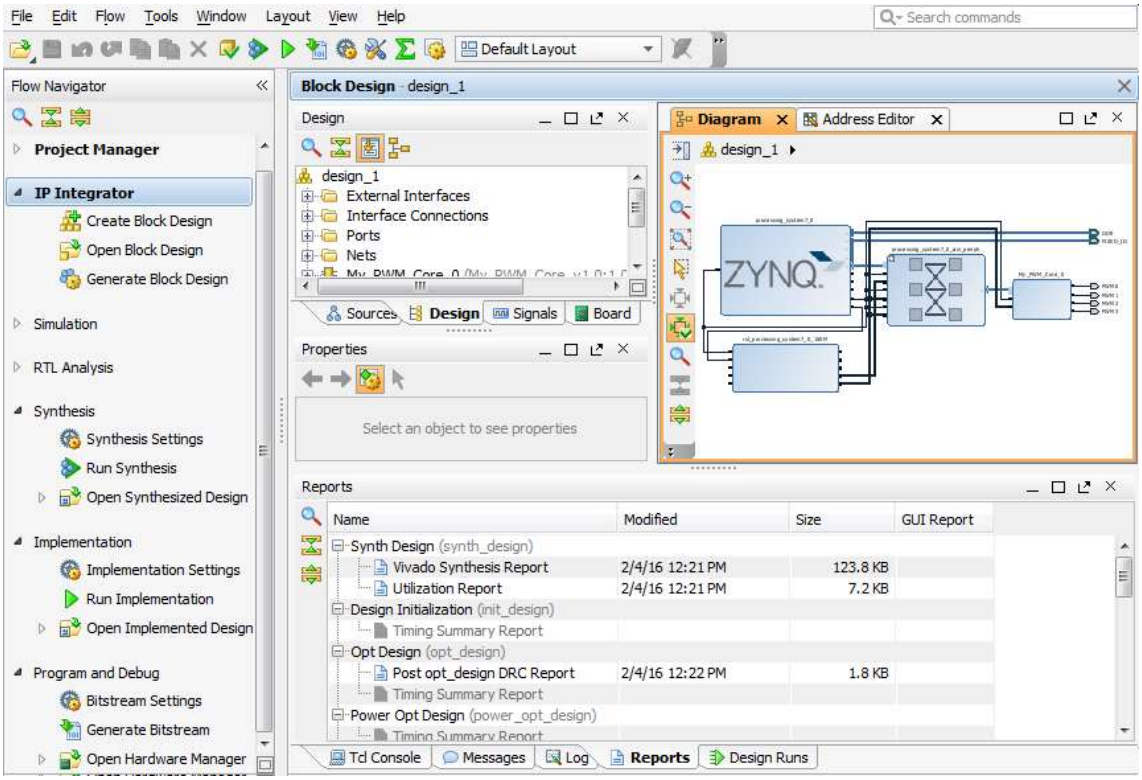


Figura 3-8 Interfaz principal de la aplicación Xilinx Vivado

3.4.2 Xilinx SDK

El SDK, o *Software Development Kit* [24], es la utilidad para el desarrollo del software suministrada por Xilinx. Se trata de un entorno basado en Eclipse capaz de crear el código que ejecutará cada microprocesador del diseño. Se trata de una aplicación distinta de Vivado, ya que no es indispensable para el desarrollo de hardware (únicamente cuando se requiere programar un microprocesador). A partir de un bitstream generado por Vivado, la interfaz permite, entre otras cosas:

- Iniciar un proyecto software en diferentes lenguajes (C, C++ o Java)
- Importar, crear, editar y compilar código fuente
- Programar y depurar el código compilado sobre la placa

De igual manera que Vivado, sus posibilidades van desde las más básicas con programas sencillos en un solo procesador hasta la complejidad de un sistema operativo operando en múltiples núcleos. Las dos aplicaciones juntas, constituyen la solución profesional completa de Xilinx para el desarrollo de sistemas empotrados de cualquier envergadura.

Capítulo 4
Sistema implementado

4.1 Introducción

Tras haber presentado las herramientas con las que se realizará el proyecto, se procede con el detalle de su desarrollo.

Conviene resaltar el hecho de que este Trabajo Fin de Grado aborda la investigación de un terreno poco explorado. Se considera así ya que no ha sido posible la localización de información relacionada con un desarrollo similar en la literatura ni en internet. Por tanto, la información se ha obtenido directamente del análisis de la documentación de los fabricantes, teniendo que emprender el diseño desde cero. Se puede dividir el proceso de desarrollo en dos partes.

En primer lugar se realiza la configuración definitiva de la interfaz de traza. Para ello es preciso sintetizar la información elaborada en el capítulo anterior, que permite conocer los elementos clave que participarán en el diseño final del sistema, para profundizar en las características principales de cada uno y comprender cómo las configuraciones aplicadas pueden modificar su funcionamiento.

A partir de un estudio de las distintas configuraciones posibles, unido a un conocimiento pormenorizado de los elementos de traza seleccionados, se puede configurar, para cada uno, el comportamiento más concordante con el objetivo del trabajo. Esta configuración es especialmente importante ya que no se dispone de ninguna referencia como punto de partida, y condiciona todo el desarrollo posterior.

En segundo lugar, se documenta el proceso íntegro de desarrollo del módulo observador, teniendo en cuenta las características de la información proporcionada por la interfaz de traza, que dependen intrínsecamente de la configuración elegida para ella.

Una caracterización precisa de la información que puede obtenerse a través de dicha interfaz permite asentar las especificaciones básicas que ha de cumplir el módulo observador diseñado. El análisis de estas especificaciones permite desarrollar poco a poco las funcionalidades que debe incorporar el módulo observador hasta alcanzar un diseño completo.

Por último, se incluye un resumen de las características más importantes del módulo resultante, tales como área o frecuencia máxima de reloj.

4.2 Configuración de la interfaz de traza

4.2.1 Procesador

El *Processing System* (PS) de la ZYNQ está compuesto por un procesador ARM Córtex-A9 MPCore [17] de doble núcleo, por lo que tenemos la posibilidad de ejecutar código en cualquiera ellos, denominados CPU0 y CPU1. De las dos CPU disponibles, el código a trazar se ejecutará en la CPU0.

La arquitectura [25] del núcleo Cortex-A9 incorpora, como todas las desarrolladas por ARM, una interfaz [16] [17] con el sistema CoreSight, tanto para traza como para depuración, tal y como se observa en la siguiente figura.

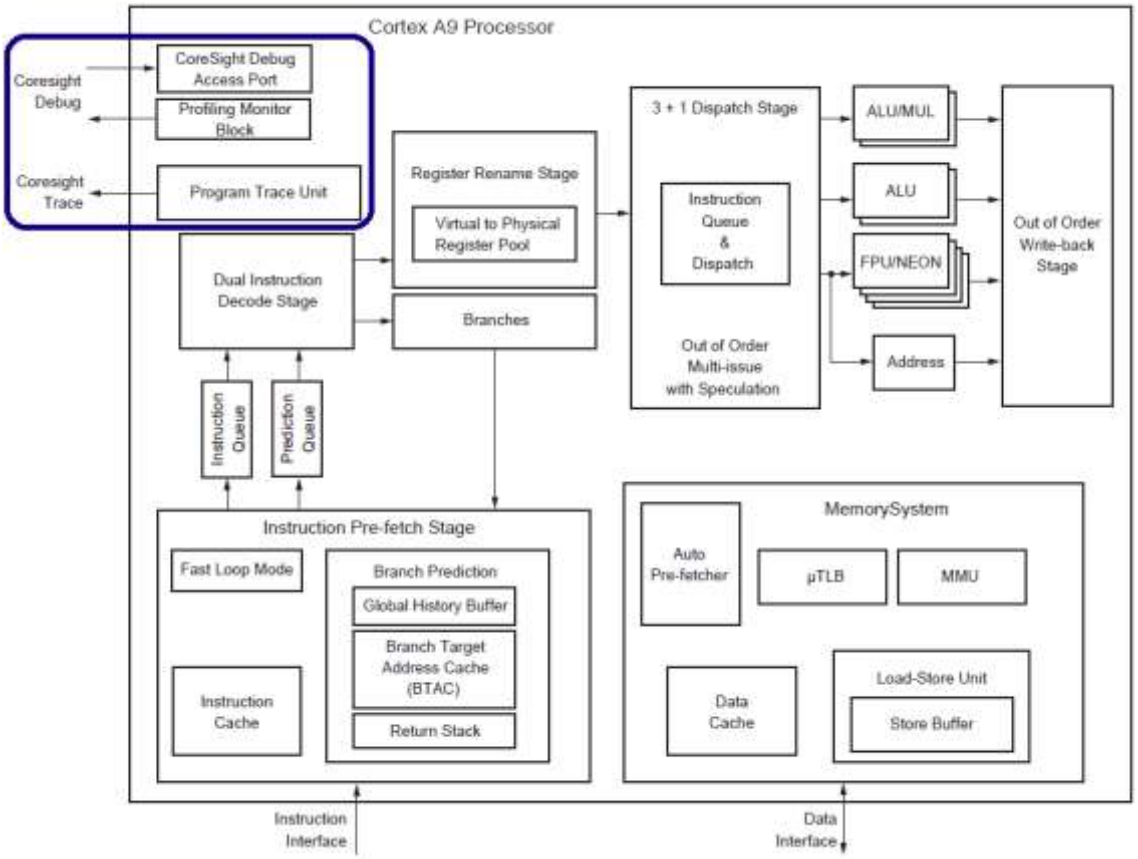


Figura 4-1 Arquitectura del núcleo Cortex-A9 [16, p. 66]

La interfaz, resaltada con un recuadro azulado, está activa por defecto, por lo que no requiere configuración.

4.2.2 Program Trace Macrocell (PTM)

La ZYNQ implementa dos unidades PTM, una para la traza de cada procesador ARM Cortex-A9 (PTM0 para la CPU0, y PTM1 para la CPU1). Se trata concretamente de la PTM-A9 [26] y está diseñada bajo la especificación ARM *Program Flow Trace* (PFT) *Architecture* [27]. Permite seguir el flujo de ejecución del procesador y proporciona una gran cantidad de recursos para maximizar la versatilidad a la hora de adaptarse a los diferentes requisitos de cada sistema.

La especificación ARM PFT *Architecture* establece una filosofía de uso eficiente de los recursos disponibles, minimizando la cantidad de información emitida mediante técnicas de compresión: utiliza el mínimo número de Bytes necesarios para comunicar cada actualización. De esta manera se logra un elevado ancho de banda sin comprometer la calidad de la información proporcionada.

La siguiente figura muestra una descripción funcional de la PTM.

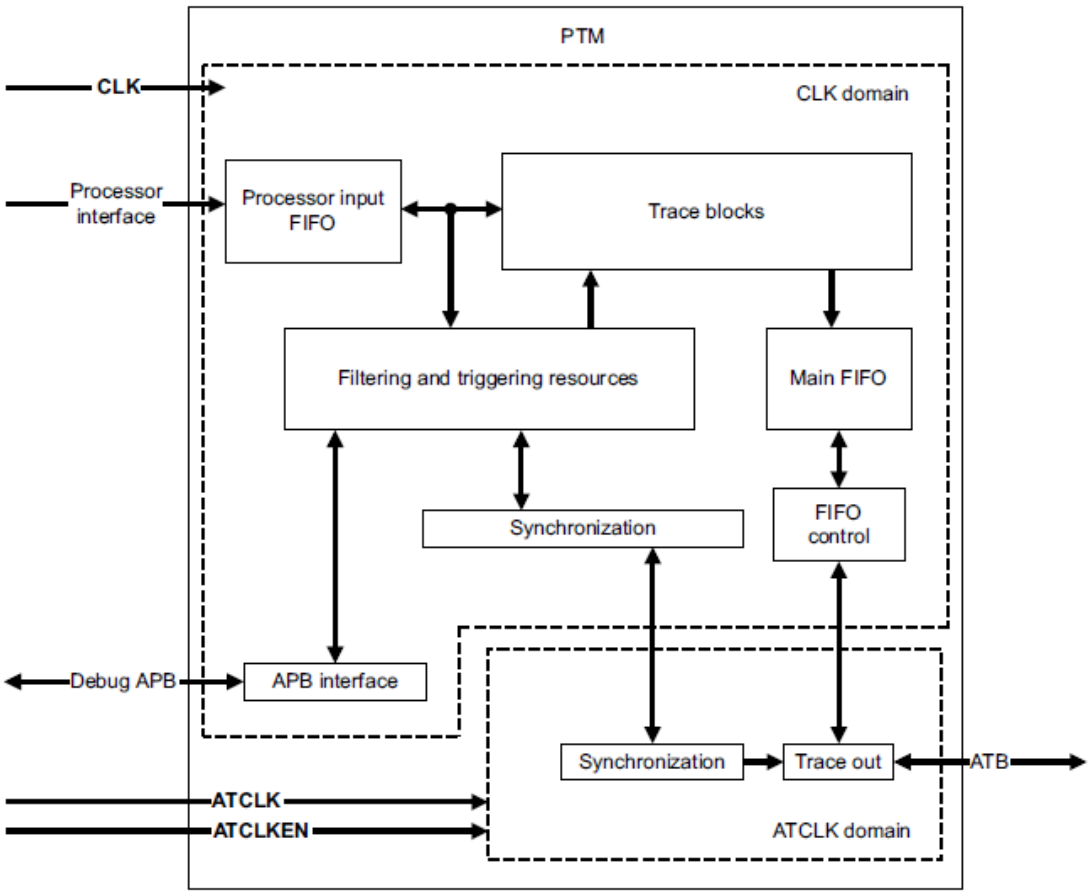


Figura 4-2 Diagrama funcional de la PTM-A9 [26, p. 14]

Del diagrama se desprende lo siguiente:

- La PTM está conectada de forma directa a la interfaz de traza del procesador, por lo que recibe toda la información que éste genera.
- La PTM funciona a la misma velocidad que el procesador, sin embargo, el flujo de traza no tiene por qué funcionar a la misma velocidad. Por tanto, es necesario que en su interior se produzca la separación entre los dominios de reloj del procesador (CLK) y del flujo de traza (ATCLK), permitiendo la independencia de las velocidades de funcionamiento de cada uno, en conjunción con las dos pilas FIFO:
  - La FIFO del procesador (*Processor Input* FIFO) almacena la información suministrada por el procesador hasta que éste confirma que está completa y es válida.
  - La FIFO principal (*Main* FIFO) sirve para acomodar las velocidades de ambos dominios de reloj. Minimiza el impacto que una ráfaga repentina de información pueda provocar en los componentes que funcionan a menor velocidad, evitando la pérdida de información.
- La PTM cuenta con los bloques (*Trace Blocks*) necesarios para generar la información de traza a partir de los datos recibidos del procesador. Esta información deberá estar adaptada al formato ARM PFT *Architecture*.
- La PTM incluye los recursos (*Filtering and triggering resources*) necesarios para condicionar su funcionamiento a las circunstancias de la ejecución. Entre otros, dispone de los siguientes:
  - *Single Address Comparator* (SAC). Detectan cuándo la dirección de memoria de programa apuntada por el PC coincide con la registrada en su interior.
    - *Address Range Comparator* (ARC). Los SAC se pueden configurar por parejas para detectar cuándo la dirección de programa apuntada por el PC se encuentra en el intervalo delimitado por las direcciones registradas en ellos.
  - *Context ID comparator*. Se activan cuando el contexto de ejecución del procesador coincide con el valor que tienen registrado.
    - Son recursos independientes, pero también se pueden asociar tanto a los SAC, como a los ARC para supeditar su activación a las dos condiciones.
  - *Counter*. Están provistos varios contadores configurables para desencadenar acciones con un período fijo de tiempo, así como contabilizar las ocurrencias de un determinado evento, o los ciclos de reloj transcurridos entre dos de ellos.
  - *Sequencer*. También incorpora una pequeña máquina de tres estados capaz de detectar la sucesión de determinados eventos gracias a la programación de sus condiciones de transición [27] [19].

La Figura 4-3 muestra el funcionamiento simplificado de algunos de los recursos provistos en la PTM:

- Los SAC 1 y 2 están configurados con sendas direcciones. Uno de ellos está asociado al comparador de ID 1, mientras que el otro no.
  - Esta discrepancia entre ambos provoca que el ARC correspondiente no deba utilizarse, ya que su comportamiento podría ser impredecible tal y como muestra la nota inferior de la figura.
- Los SAC 3 y 4 están configurados y asociados al comparador de ID 2, lo que sí permite el uso del ARC 2.
- Los comparadores de ID, por su parte, pueden generar señales de activación independientemente de la activación de los demás recursos.
- El contador está configurado para desencadenar un evento periódico cada vez que llegue a cero.
- El resto de recursos representados están configurados y producen sus respectivas señales asociadas.

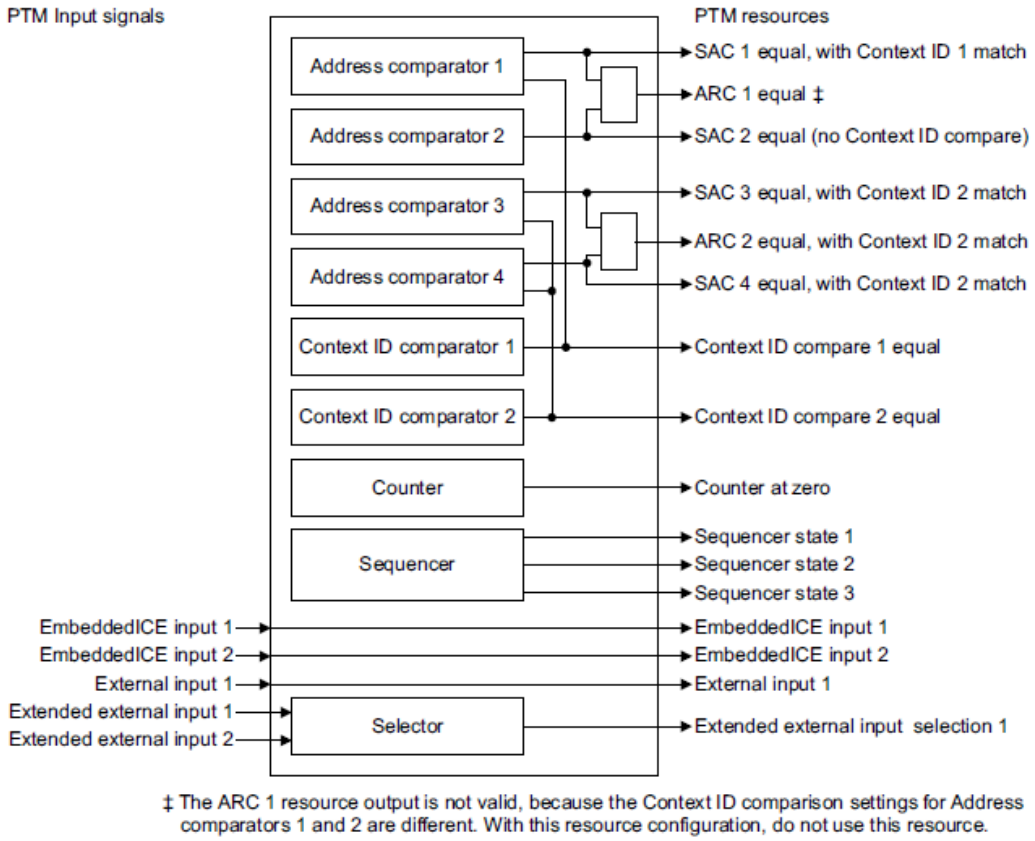


Figura 4-3 Ejemplo de configuración de los recursos de la PTM [27, p. 51]

Estos, y otros recursos, están gobernados por registros de configuración, mapeados en memoria y accesibles a través del bus APB, tal y como puede verse en la Figura 4-2.

Configuración utilizada

En este proyecto ha sido necesario configurar los siguientes recursos:

- SAC 1 y 2 contienen las direcciones de inicio y fin de la zona de interés a trazar.
  - ARC 1 está configurado para habilitar la traza en este intervalo.
- Los comparadores de ID han sido deshabilitados.
- Se ha deshabilitado el contador de ciclos de ejecución.
- Se ha configurado el contador de sincronización a 1024 ciclos de período.
- El evento que desencadena el *Trigger* de la PTM se ha configurado para que no salte nunca.
- Se ha activado la generación forzada de paquetes de dirección de salto (*Branch Broadcasting* = 1). Esta característica tiene como objetivo facilitar la traza del código cuando no se tiene una copia del mismo.
  - En condiciones normales, la PTM únicamente emite estos paquetes cuando no se pueden inferir a partir del código ejecutado. Esto minimiza la cantidad de información necesaria para los sistemas de traza más potentes (usualmente basados en un computador). Sin embargo, el observador que se pretende diseñar no va a conocer este código, tal y como consta en los objetivos del trabajo, por lo que es preciso activar esta opción.

La totalidad de los recursos de configuración disponibles puede consultarse en [19] y [27].

4.2.3 Funnel

Los flujos de traza generados por ambas PTM, así como por las demás fuentes de traza llegan hasta el *Funnel* [19] de forma independiente, a través de sus múltiples entradas. Esta entidad será la encargada de hacerlos converger en un único canal para enviarlos hacia la salida, mediante dos tareas complementarias:

- Habilitar o deshabilitar las distintas entradas.
- Establecer una prioridad entre ellas.

La siguiente figura muestra los bloques que componen esta entidad.

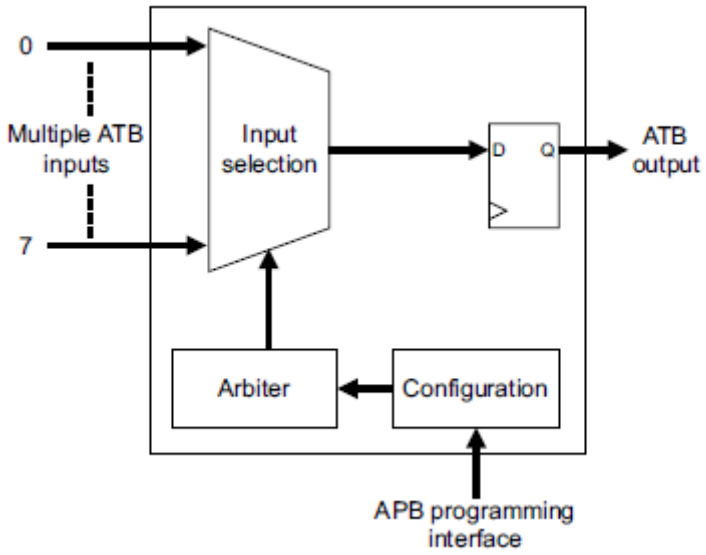


Figura 4-4 Diagrama funcional del Funnel [19, p. 160]

Entre las distintas entradas que recibe el multiplexor, el árbitro selecciona la más prioritaria de las que están habilitadas. Ambos criterios quedan establecidos a través de sus registros de configuración que están mapeados en memoria y son accesibles a través de la interfaz APB.

Configuración utilizada

Se deben realizar los siguientes pasos:

- Habilitar la entrada del *Funnel* correspondiente a la PTM0 del procesador.
- Dotar la entrada de la PTM0 de un código de prioridad válido. Éste puede ser cualquiera ya que es la única entrada habilitada, y no debe competir por el recurso con ninguna otra.

4.2.4 Replicator

La función de este componente [19] es dar la posibilidad de que dos destinos de traza operen con la misma información, tal y como muestra la siguiente figura.

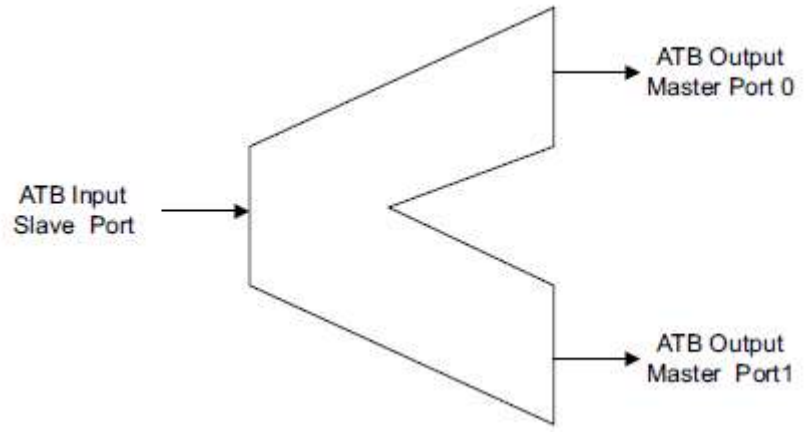


Figura 4-5 Diagrama funcional de Replicator [19, p. 154]

Su sencillez provoca que se pueda obviar su presencia, funcionando de forma totalmente transparente al diseño, y sin requerir ningún registro de configuración.

4.2.5 Trace Port Interface Unit (TPIU)

Una vez configurados los dispositivos anteriores, tenemos la información de traza a las puertas de la PL. El último paso es configurar el puerto de salida [19], representado en la siguiente figura.

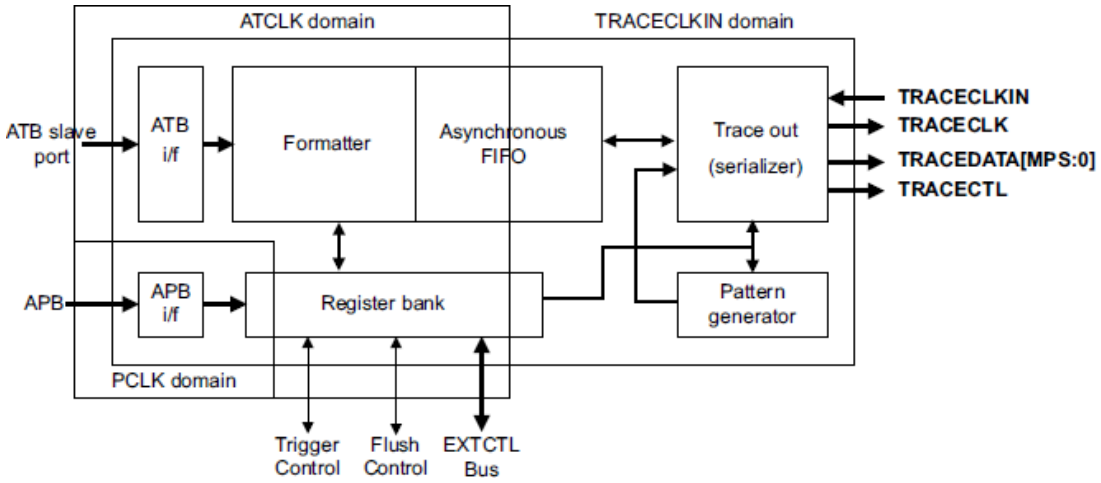


Figura 4-6 Diagrama funcional de la TPIU [19, p. 186]

En este caso, también coexisten dos dominios de reloj, el que gobierna el flujo de traza (ATCLK) y el que marca el ritmo de salida de los paquetes a través del puerto (TRACECLKIN). Su división está marcada por la presencia de una pila FIFO, que al igual que en el caso de la PTM acomoda ambas velocidades absorbiendo el efecto de las ráfagas repentinas y evitando que se pierda información.

- En el dominio ATCLK se encuentra:
  - La interfaz con el bus ATB por donde llega la información de traza generada por la PTM y tras haber pasado por el *Funnel* y el *Replicator*.
  - El *Formatter*, un bloque configurable capaz de redistribuir la información recibida, para adaptarla a un formato determinado. Esto es imprescindible cuando hay varias fuentes habilitadas en el *Funnel*, ya que es la única manera de distinguir su origen.
- Y en el dominio TRACECLKIN:
  - Un generador de patrones predefinidos, cuya función es la de revisar el correcto funcionamiento del puerto de salida, así como comprobar que la máxima velocidad a la que puede comunicarse un dispositivo conectado.
  - El *Serializer* coloca la información de la FIFO en el puerto de salida con cada flanco de subida de TRACECLKIN. Recordemos que el puerto puede funcionar con ancho configurable, por lo que la labor de este bloque garantiza que la información se divida correctamente.
- Asimismo, es destacable la presencia de una interfaz con el bus ATB que da acceso a todos los registros de configuración, mapeados en memoria y permite establecer el funcionamiento de cada uno de los bloques de esta entidad.

Este puerto permite la comunicación con un receptor ubicado en la PL vía EMIO, o en otro chip vía MIO, a través de las siguientes señales:

- TRACECLKIN: Señal de reloj que establece la velocidad de transmisión. Al ser una señal de entrada, es el módulo receptor quien la impone, permitiendo adaptarla a sus características particulares.
- TRACECLK: Señal de reloj que sincroniza el intercambio de información. El puerto emite los paquetes con la velocidad de TRACECLKIN en cada flanco de TRACECLK.
- TRACEDATA: Bus a través del cual se emite la información de traza.
- TRACECTL: Señal que indica la validez de los datos, así como la presencia de algún error en el puerto.

Existe información disponible sobre la TPIU tanto por parte de ARM como de Xilinx acerca de las distintas configuraciones válidas.

En primer lugar se deben tener en cuenta las recomendaciones facilitadas por ARM, dado que es la empresa que ha diseñado el componente.

Tabla 4-1 Configuraciones recomendadas de la TPIU, ARM [19, p. 209]

TRACECLK present	TRACECTL present	TRACEDATA width	Total pin count	Comment
Yes	Yes	32 bits [31:0]	34	Largest implementation
Yes	No	9 bits [8:0]	10	Extra data pin available in comparison to the typical ETM implementation.
Yes	Yes	8 bits [7:0]	10	Typical ETM-compatible TPA implementation.
Yes	Yes	2 bits [1:0]	4	Smallest implementation with typical TPAs.
Yes	No	1 bit [0]	2	Smallest implementation with a protocol-aware TPA

El puerto admite configuraciones diversas, siendo especialmente relevante el número de pines utilizado por cada una: desde un único pin hasta 34. Asimismo destaca la posibilidad de eliminar el pin de TRACECTL, todo ello con la intención de minimizar al máximo los recursos necesarios.

En segundo lugar es preciso conocer las posibilidades que ofrece la ZYNQ, dadas por Xilinx, que es la empresa que ha fabricado el chip.

Tabla 4-2 Posibles configuraciones de la TPIU en la ZYNQ, Xilinx [16, p. 727]

Active interface	EMIO	MIO
Clock source to operate the TPIU	EMIOTRACECLK	PS clock controller
Output clock present?	No	Yes
Clock edge(s) to sample trace data and control	Rising	Rising and falling
Supported data widths	1, 2, 4, 8, 16, 32	1, 2, 4, 8, 16
Application Note	Since PL supplies the clock to TPIU, PL can use the same clock to sample.	External device should delay the trace clock output by approximately half clock period, and use the delayed clock to sample.

La circunstancia más notable es que la configuración de 32 bits únicamente está disponible vía EMIO.

Una vez establecidas las distintas opciones es posible seleccionar la más apropiada para cumplir con los objetivos.

Configuración utilizada

Se deben establecer dos parámetros generales:

- El comportamiento del *Formatter*. En este caso al contar solamente con una fuente de traza, no es preciso utilizarlo. De esta manera se obtendrá la información tal cual ha sido generada por la PTM, y ello facilitará la comprensión del funcionamiento global de la traza. Permanecerá desactivado.
- El ancho del puerto de salida. A la vista de la especificación ARM PFT *Architecture*, el ancho de puerto más adecuado si no se utiliza *Formatter* es de 8 bits.

Por último, constatar que dado que el módulo observador se va a diseñar en la PL, se conectará a la TPIU vía EMIO.

4.3 Diseño del módulo observador

4.3.1 Caracterización de la información de traza

La especificación ARM PFT *Architecture* [27] establece un sistema de comunicación dividida en paquetes. Toda la información se divide en palabras de 1 Byte, el mismo ancho que se ha configurado para la TPIU. La siguiente tabla muestra un resumen de los distintos paquetes posibles.

Tabla 4-3 Paquetes de la especificación ARM PFT Architecture

Tipo de paquete	Encabezado	Número de Bytes*
A-sync	b0000 0000 = 0x00	5 o más
I-sync	b0000 1000 = 0x08	6
Atom	b1XXX XXX0	1
Branch address	bXXXX XXX1	De 1 a 5
Waypoint update	b0111 0010 = 0x72	De 2 a 7
Trigger	b0000 1100 = 0x0C	1
Context ID	b0110 1110 = 0x6E	De 2 a 5
*VMID***	b0011 1100 = 0x3C	2
*Timestamp***	b01000X10	De 2 a 8
Exception return	b0111 0110 = 0x76	1
Ignore	b0110 0110 = 0x66	1

*Opciones de *Cycle-accurate tracing* y *Context* ID desactivadas en la configuración de la PTM.
**Paquete no soportado por la PTM implementada en la ZYNQ.

Cada paquete se puede identificar por su encabezado, esto es, la primera palabra de cada paquete. Muchos de ellos incorporan a continuación más Bytes que completan la información proporcionada, y en varios de los casos, esta cantidad de palabras que siguen a la primera es variable, ya que la PTM utiliza solamente el número mínimo de Bytes necesarios para aportar la información. Como consecuencia, reconocer el último Byte de cada paquete no es trivial, la especificación establece en él un código distintivo para permitir su reconocimiento.

Un paquete identifica un suceso reconocido por la PTM:

- ***A-sync***, sincronización de alineamiento. Emite una gran cantidad de ceros para permitir al receptor cerciorarse de que el puerto no se ha desalineado.
- ***I-sync***, sincronización de flujo de instrucción. Este paquete contiene la dirección completa de la ubicación PC en el momento exacto en que se genera. Se emite en los siguientes casos:
  - Al activarse la traza para servir como punto de partida.
  - Cuando se produce un *overflow* en la FIFO principal de la PTM, perdiendo información de traza.
  - Periódicamente, cuando el contador de sincronización pasa por cero.
  - Al desactivarse la traza, para indicar la última instrucción trazada.
- ***Atom***. Indica, utilizando la menor cantidad de bits posibles, el resultado de las últimas instrucciones de salto condicional. Se trata de un paquete de un solo byte, que es el encabezado, con la configuración b1xxx xxx0:
  - El MSB (bit7) siempre vale '1' y el LSB (bit0) siempre vale '0'.
  - Los bits marcados en x contienen los *"Atoms"*, tantos como condiciones de salto se hayan evaluado desde el último paquete de *Atom*.
  - Los *Atoms* indican con un '1' o un '0' si la condición de salto tuvo resultado positivo o negativo
  - El byte puede contener entre uno y cinco *Atoms*.
  - Los *Atoms* se alinean a la derecha, siendo el bit 1 el más reciente. El primer bit x no ocupado por un *Atom* será un '1'.
  - Así pues, el ejemplo b1001 0110 contiene tres *Atoms*.
- ***Branch address***, dirección de salto. Indica un cambio en el flujo del programa. Contiene la información necesaria para conocer la dirección del PC de destino del salto, a partir de la última dirección conocida. Es decir, emitirá tantos bytes como sean necesarios para reconstruir la dirección pero no más; lo que concuerda con el criterio de minimizar la información de traza necesaria de cara a maximizar el ancho de banda.
- ***Waypoint update***, actualización de punto de interés. Este paquete se genera tras un salto no previsible a partir del código del programa, como pueda ser una excepción. Contiene la posición del PC en el momento de la excepción.
- ***Trigger***, indica que se ha producido el evento que desencadena el *Trigger*, configurado en la PTM.
- ***Context ID***, indica un cambio en el contexto del procesador.

- ***VMID***. La versión de PTM implementada en la ZYNQ no es compatible con este paquete.
- ***Timestamp***. Contiene un código de tiempo común a todos los componentes de la ZYNQ, muy útil para comprobar el orden en que ocurren distintos sucesos. Lamentablemente la PTM implementada en la ZYNQ no cuenta con esta característica.
- ***Exception return***. Indica que el flujo de programa ha finalizado la rutina de interrupción y que ha vuelto al punto donde estaba en el momento en que se produjo.
- ***Ignore***. Indica que el paquete no tiene validez y debe ser ignorado.

La posibilidad de que aparezca cada uno de ellos depende en gran medida de la naturaleza del código ejecutado por el procesador, así como de la configuración de la PTM. Esta situación se resume en la siguiente tabla.

Tabla 4-4 Previsión de aparición de los distintos paquetes de traza

Tipo de paquete	Esperable	Razón
A-sync	**Sí**	**Obligatorio**
I-sync	**Sí**	**Obligatorio**
Atom	**Sí**	**Todo código tiene condiciones de salto**
Branch address	**Sí**	***Branch Broadcasting* está activado**
Waypoint update	No	No se han configurado excepciones
Trigger	No	Está configurado para no saltar nunca
Context ID	No	Está deshabilitado
VMID	No	La versión de PTM no es compatible
Timestamp	No	Característica no incluida en la PTM
Exception return	No	No se han configurado excepciones
Ignore	**Sí**	**Se generan de forma espontánea**

El módulo observador diseñado debe ser capaz de identificar el tipo de paquete recibido y proveer los medios necesarios para extraer la información que contiene.

4.3.2 Solución propuesta

Para comenzar el desarrollo del módulo observador se debe tener en cuenta la interfaz a la cual se va a conectar, siendo necesaria una descripción precisa de su funcionamiento. Como muestra la Tabla 4-2, rutar el puerto hacia la EMIO tiene ciertas particularidades:

- La señal de reloj de salida TRACECLK no está presente
- La información de traza se actualiza con el flanco de subida de TRACECLKIN
- La nota de aplicación recomienda utilizar la misma señal TRACECLKIN para capturar esta información.

Además, se dispone del siguiente cronograma para caracterizar el comportamiento temporal de estas señales.

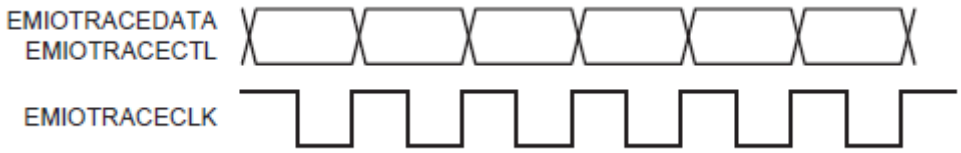


Figura 4-7 Cronograma de las señales de la TPIU por EMIO [16, p. 728]

Extrayendo que las señales TRACEDATA y TRACECTL se actualizan en el flanco de subida de TRACECLKIN y son estables en el flanco de bajada.

Para conectarse a estas señales de la TPIU, hay que tener en cuenta su dirección:

- Entradas: TRACECLKIN. No es trivial generar una señal de reloj en el interior de una FPGA. Para solucionar este problema, la lógica programable (PL) de la ZYNQ incluye dos celdas dedicadas de gestión de reloj, cada una con un PLL asociado. Se empleará la primera de ellas (FCLK_CLK0) para generar esta señal, dejando la segunda (FCLK_CLK1) sin utilizar.
- Salidas: TRACECTL, TRACEDATA. Puesto que no se conoce su comportamiento dinámico, para poder trabajar correctamente con ellas, el módulo observador debe ser capaz de muestrearlas y registrarlas en el momento en que sean estables (flanco de bajada de TRACECLKIN).

La configuración anterior se puede representar con el siguiente diagrama de bloques.

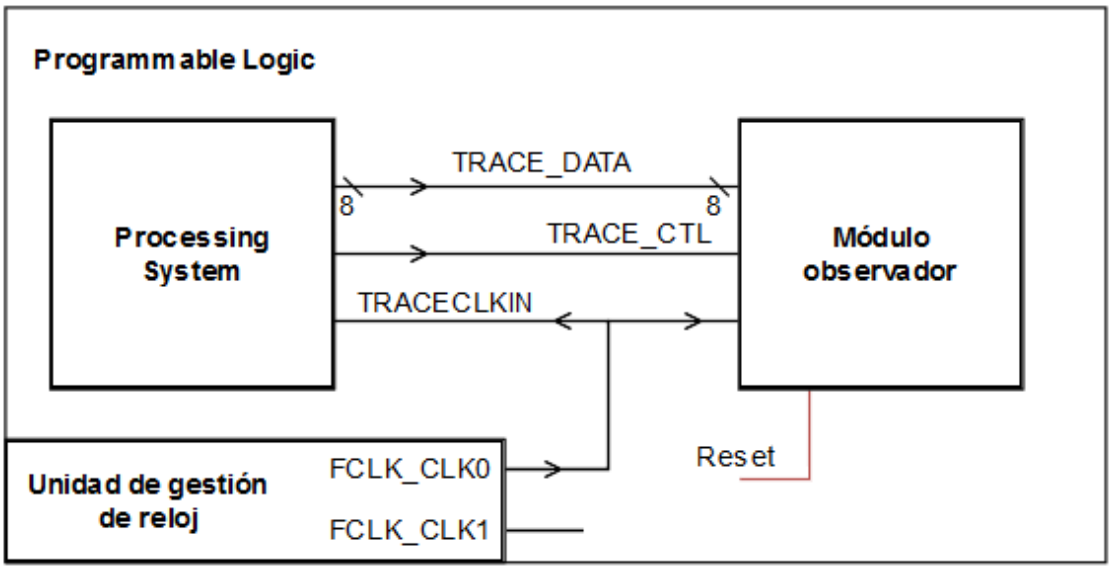


Figura 4-8 Diagrama de bloques conceptual

Está formado por el PS de la ZYNQ (ZYNQ7 *Processing System*), el observador (Checker_v1_0) y la célula de gestión de reloj de la PL.

La señal de reloj TRACECLKIN es generada en por la célula de gestión de reloj FCLK_CLK0 y entra tanto en la TPIU como en el observador. Las señales TRACE_CTL Y TRACE_DATA están conectadas directamente desde el PS hasta el observador. Adicionalmente, se ha provisto de una señal de Reset ya que todo sistema secuencial necesita uno.

Una vez establecidas las conexiones entre el módulo observador y el PS, el siguiente paso es establecer las funciones que debe ser capaz de realizar. Para ello es necesario recopilar las características del sistema que se conocen hasta el momento:

- Los paquetes de información de traza generados por la PTM cumplen la especificación ARM PFT *Architecture*, que establece que se compongan de palabras de 8 bits (1 Byte). Cada paquete tiene un encabezado único (ver Tabla 4-3.
- La TPIU está configurada con el mismo tamaño de palabra que la PTM; emite un nuevo Byte a través de la señal TRACEDATA con cada flanco de subida de TRACECLKIN (ver Figura 4-7).

- La señal TRACECTL indica la validez de los datos. Se dispone de la siguiente tabla para ilustrar su funcionamiento.

Tabla 4-5 Uso de la señal TRACECTL [19, p. 211]

TRACECTL	TRACEDATA [1]	TRACEDATA [0]	Trigger Yes/No	Capture Yes/No	Description
0	x	x	No	Yes	Normal Trace Data
1	0	0	Yes	Yes	Trigger Packet[a]
1	1	0	Yes	No	Trigger
1	x	1	No	No	TraceDisable

a. The trigger packet encoding is required for the current ETMv3 protocol that uses a special encoding for triggers that always occur on the lower bits of **TRACEDATA**.

  La aparición de *Triggers* está deshabilitada, por lo que las posibilidades se reducen a dos:

  - Información normal de traza: TRACECTL = '0'.
  - Traza inactiva: TRACECTL = '1'.

  Para entender el comportamiento de esta señal se debe tener en cuenta que el flujo de información de traza no tiene por qué ser continuado, ya que los paquetes solamente se generan cuando ocurre un evento en la PTM, siendo posible que durante distintos intervalos de tiempo, no haya ninguna información que enviar. Es en este momento cuando la señal TRACECTL tiene el valor '1', lo que equivale a una señal de Enable que distingue cuándo la traza está inactiva y cuándo la información proporcionada es válida.

- Las señales TRACEDATA y TRACECTL son estables en el flanco de bajada de TRACECLKIN.
- La especificación ARM PFT *Architecture* permite que los paquetes estén compuestos por un número variable de Bytes, y establece para cada tipo de paquete un método de reconocer su final.
  - Es importante notar que para un funcionamiento eficaz del observador, no solamente será suficiente identificar los paquetes por su encabezado sino además distinguir el final de cada uno para esperar a continuación el siguiente encabezado. Por tanto no basta con atender solamente a los paquetes que se espera obtener, marcados en la Tabla 4-4, sino a todos ellos para conseguir delimitar cada uno. Deben estar previstos la mayor cantidad de casos posibles.

Se puede dividir el diseño del módulo en dos fases.

En primer lugar, es fácil advertir que, independientemente de la finalidad del módulo observador diseñado, éste debe incorporar los recursos necesarios para reconocer y delimitar los paquetes recibidos. Dado que la identificación de un nuevo paquete depende inexorablemente de reconocer la finalización del anterior, la implementación más lógica es en la representada en la siguiente figura.

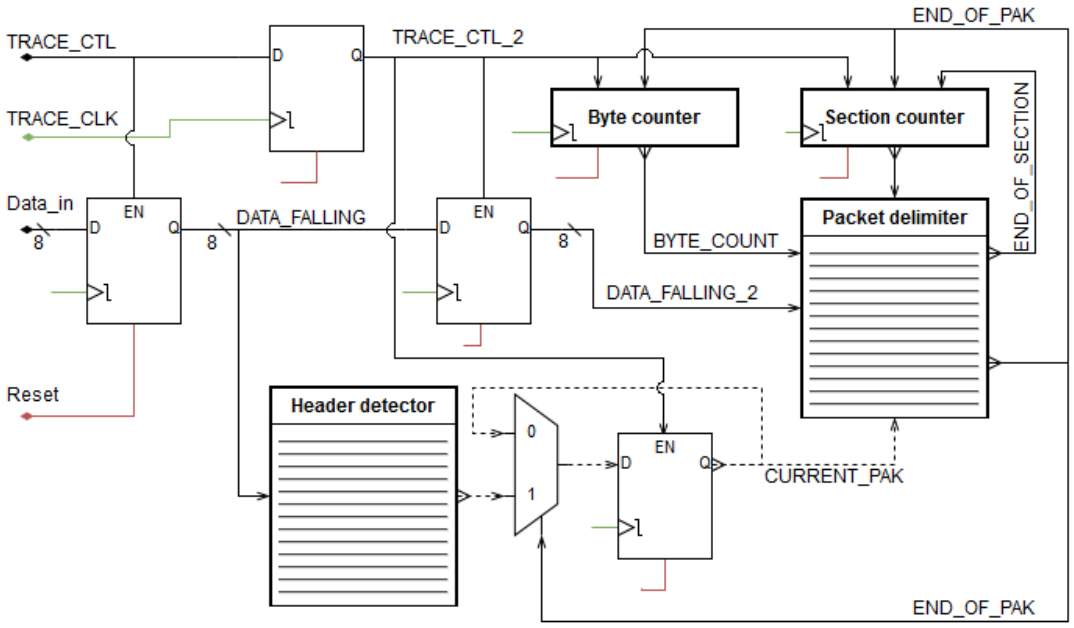


Figura 4-9 Delimitador de paquetes del módulo observador

Mediante el diagrama de bloques simplificado de la figura anterior resulta sencillo entender el funcionamiento del sistema delimitador de paquetes diseñado para el módulo observador:

- La señal TRACEDATA se muestrea en flanco de bajada en el registro DATA_FALLING. En el siguiente flanco pasa al segundo registro del pipeline, DATA_FALLING_2.
  - Puesto que la señal TRACE_CTL es la señal de Enable del primer registro, es necesario registrarla para disponer de ella en el segundo.
- La configuración en pipeline simplifica la tarea de detección tanto de inicio como de fin de paquete:
  - El detector de encabezamientos comprueba constantemente si la información registrada en DATA_FALLING se corresponde con algún encabezamiento conocido. Emite una señal con el resultado de la comparación.
  - Si el delimitador de paquetes detecta que DATA_FALLING_2 contiene el final de un paquete, entonces DATA_FALLING va a contener el encabezamiento del siguiente. Mediante la señal END_OF_PAK habilita el multiplexor para actualizar la información del paquete actual en el registro CURRENT_PAK con el nuevo encabezamiento detectado.

- La estructura interna de los paquetes está dividida en secciones. Es necesario distinguirlas para extraer correctamente la información que contienen. Para ello se han dispuesto sendos contadores de Byte y de sección que indican de forma precisa el lugar que ocupa cada palabra dentro del paquete.
- El circuito es síncrono. Tanto los biestables como los contadores son activos por el flanco de bajada del mismo reloj TRACECLKIN.

En segundo lugar, una vez el circuito es capaz de conocer en todo momento tanto el paquete en que se encuentra como la posición que ocupa cada Byte en su interior, se puede añadir la lógica necesaria para realizar el seguimiento del PC, como muestra la siguiente figura.

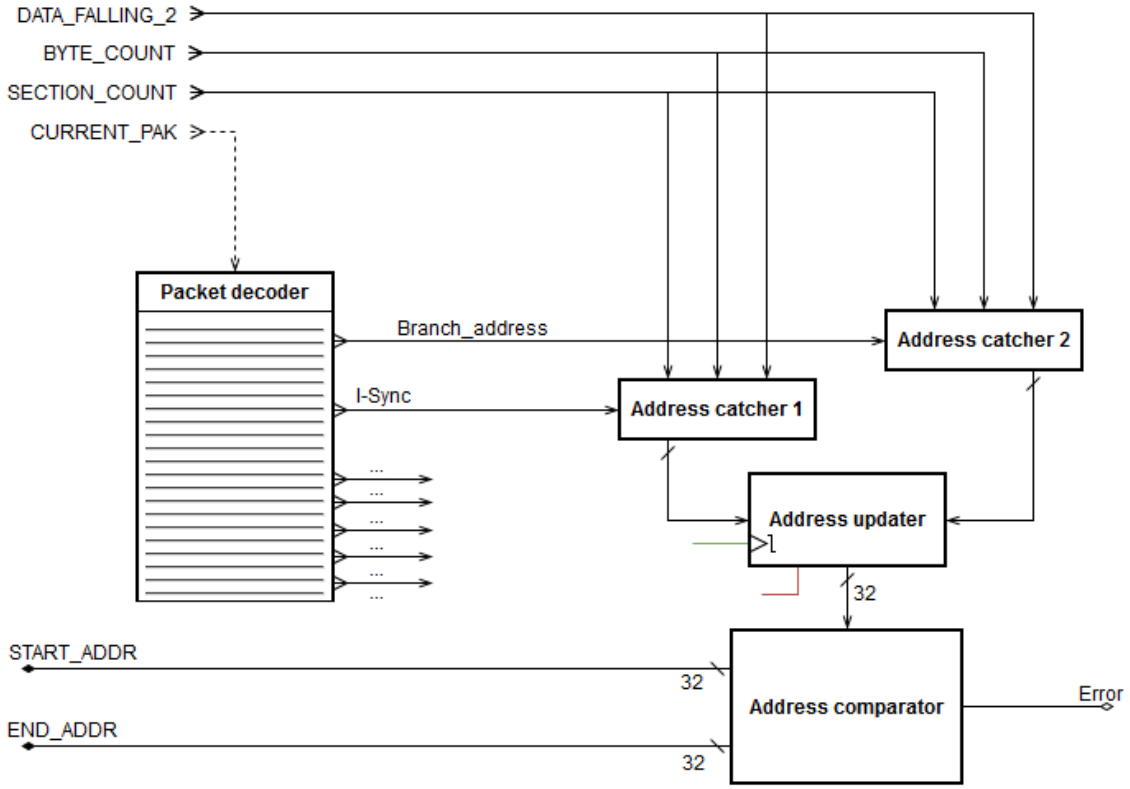


Figura 4-10 Seguidor de PC del módulo observador

De entre todos los paquetes disponibles en la especificación ARM PFT *Architecture* únicamente los que corresponden a *Branch_address* y a *I-Sync* contienen información de la posición del PC.

- El decodificador de paquete es capaz de habilitar o deshabilitar los componentes conectados a él, de manera que solamente se encontrarán activos los que se encuentren conectados a la salida correspondiente con el paquete en curso.
  - Es preciso constatar el hecho de que dado que el circuito delimitador es capaz de distinguir todos los tipos de paquetes, es posible conectar al decodificador de paquetes una cantidad ilimitada de componentes que enriquezcan el funcionamiento del observador, tal como se razona en las líneas futuras.

- Independientemente de que tanto el paquete *Branch Address* como el *I-Sync* contengan información sobre la posición del PC, cada uno la codifica de manera diferente, por lo que es necesario disponer dos captores diferentes.
- Una vez que el captor ha extraído la información sobre la dirección la envía al actualizador, que recompone la dirección actual del PC a partir de los datos proporcionados por los captores y la última posición conocida.
- Por último, está dispuesto un comparador para establecer si la dirección registrada en el actualizador se encuentra en el rango permitido, establecido por las señales START_ADDR y END_ADDR.

A estas alturas, el diseño del módulo observador está casi terminado, sin embargo aún falta resolver una última necesidad: en la Figura 4-10 aparecen dos nuevas entradas al módulo observador, que no estaban previstas en el diagrama inicial de la Figura 4-8. Se trata de las señales START_ADDR y END_ADDR, ambas de 32 bits, que delimitan el intervalo de direcciones permitidas para el PC.

Más allá de estas señales, esta necesidad deja patente una carencia importante del módulo observador hasta el momento: la falta de registros de configuración que permitan al usuario modificar su funcionamiento a través del software.

Para satisfacer ambas carencias y facilitar ampliaciones posteriores se va a incluir una conexión al bus AXI del microprocesador. Vivado incluye un asistente de creación de periféricos compatibles con el bus AXI que proporciona una plantilla en la que únicamente hay que incluir los componentes diseñados. Dado que ya se dispone de un módulo observador operativo, no tiene sentido volver a describirlo dentro de la plantilla del bus AXI, sino que utilizaremos una instanciación de componente permitida por el lenguaje VHDL.

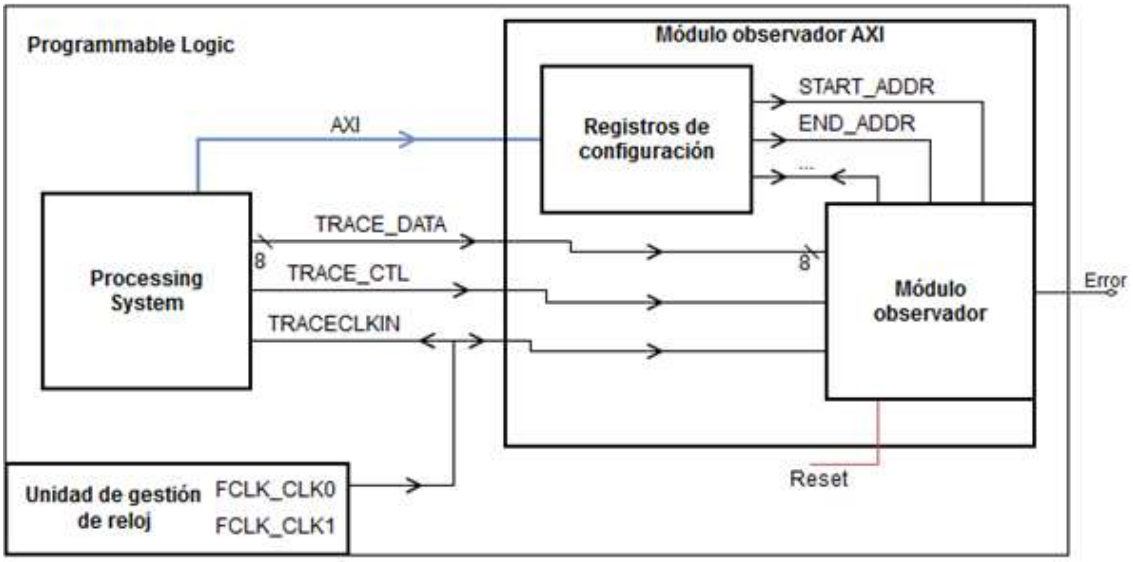


Figura 4-11 Diagrama de bloques definitivo

Se puede entender una instanciación de componente como su inserción en una entidad de nivel superior, equivalente a crear una "carcasa" que contenga las nuevas características (como la conexión AXI y los registros de configuración), instalar en su interior el componente antiguo y realizar las conexiones necesarias para su funcionamiento. De esta manera se incrementan las capacidades del módulo observador sin apenas alterar su estructura interna, únicamente añadiendo las entradas que faltasen.

De este modo es posible interaccionar con el observador para modificar su comportamiento escribiendo en los registros que se mapean en memoria gracias a las características del bus AXI, haciendo que su acceso sea tan sencillo como cualquier otro registro del microprocesador.

Una vez realizadas en VHDL la descripción completa del módulo observador, así como su ampliación para comunicarse por AXI, el último paso necesario es utilizar el asistente de Vivado para la creación de bloques IP. Se trata de un proceso automático en el que solamente se requiere que el usuario compruebe que todos los parámetros del aspecto del bloque han quedado bien ubicados y valide el resultado final, aunque de manera opcional es posible realizar comprobaciones más exhaustivas así como modificar algunas configuraciones. De esta manera es posible obtener un bloque que se puede instanciar en un diseño de Vivado y que integra todas las funcionalidades descritas anteriormente:

- Módulo observador, cuyo funcionamiento se puede dividir en dos partes:
  - Delimitador de paquetes.
  - Seguidor de PC.
- Interfaz con el bus AXI y registros de configuración.

4.3.3 Resultados de síntesis

Una vez terminado el proceso de diseño del módulo observador, y habiendo obtenido el bloque IP asociado, el último paso para completar el diseño del hardware es integrar el sistema empotrado. Mediante Vivado, es posible describir el hardware del sistema con un diagrama de bloques, utilizando como referencia la Figura 4-11.

Será necesario crear un nuevo proyecto, en el que integrar los distintos bloques:

- Módulo observador con interfaz AXI.
- El microprocesador (*Processing System*, PS) de la ZYNQ.
- La lógica de gestión de reloj.

Para su correcta interconexión, Vivado analiza de forma automática el diseño y genera bloques adicionales tales como un gestor de Reset del microprocesador o un puente de interconexiones de Bus AXI. También genera las conexiones adicionales necesarias.

El diagrama de bloques de Vivado resultante se muestra en la siguiente figura.

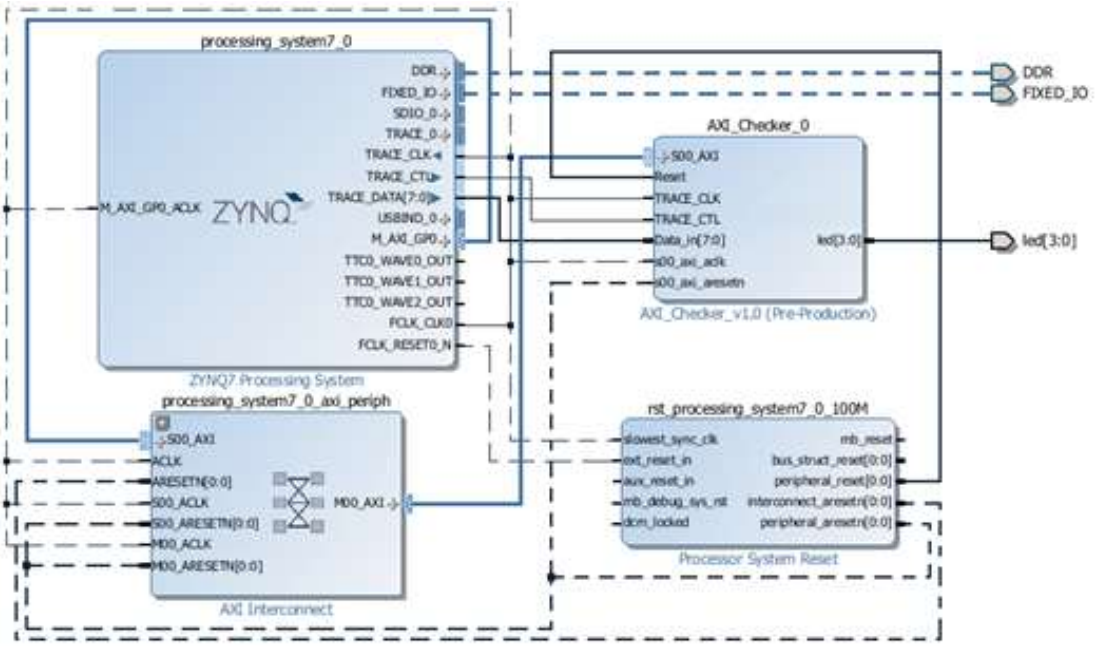


Figura 4-12 Diagrama de bloques de Vivado del sistema completo

Como puede apreciarse, la cantidad de interconexiones es muy superior a las que mostraba la Figura 4-11. Con el fin de facilitar su correcta visualización se han discontinuado las líneas que no se corresponden estrictamente con ella. Es destacable el empleo de los LEDs instalados a bordo de la placa para indicar la detección de un error. Su uso puede ampliarse utilizando diferentes códigos para cada tipo de error detectado.

Para poder programar la placa ZYBO con el sistema descrito mediante la figura anterior, se deben seguir los pasos del flujo de diseño, representados en la siguiente figura.

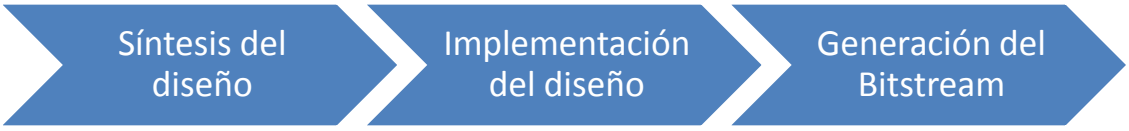


Figura 4-13 Flujo del proceso de diseño en Vivado

Vivado genera distintos informes sobre cada uno de los procesos que permiten comprobar si los resultados fueron satisfactorios o si se produjo algún error.

Además, si el resultado de todo proceso ha sido positivo, Vivado genera estadísticas sobre el diseño final, denominados resultado en área y velocidad.

4.3.3.a) Resultados en área

Hacen referencia a la cantidad de recursos de la lógica programable empleados por el módulo observador. Se adjuntan sendas capturas con los resultados.

Tabla 4-6 Uso de los recursos por el módulo observador

Resource	Utilization	Available	Utilization %
LUT	558	17600	3.17
LUTRAM	64	6000	1.07
FF	831	35200	2.36
IO	8	100	8.00
BUFG	1	32	3.12

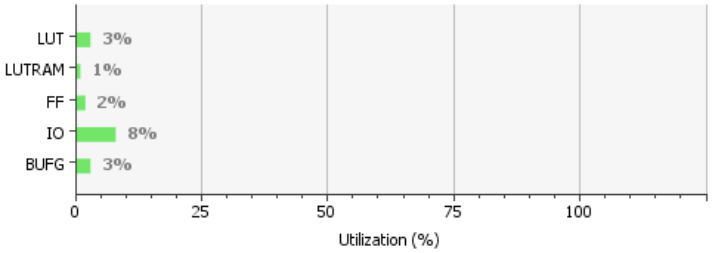


Figura 4-14 Gráfica de uso de recursos del módulo observador

Tal y como arrojan las estadísticas anteriores, el diseño del módulo observador ha cumplido su objetivo de tener un tamaño reducido.

4.3.3.b) Resultados en velocidad, para 150MHz

Hacen referencia a la frecuencia máxima de funcionamiento del circuito. Esta característica se puede conocer mediante el cálculo del tiempo que tarda en recorrer su camino la señal más lenta, teniendo en cuenta el retardo que produce cada elemento que atraviesa, así como los efectos de carga de otros componentes y los tiempos de subida y bajada. La siguiente captura muestra estos resultados, para una frecuencia de 150MHz.

Tabla 4-7 Resultados en velocidad del módulo observador, a 150MHz

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,016 ns	Worst Hold Slack (WHS):	0,057 ns	Worst Pulse Width Slack (WPWS):	2,353 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1899	Total Number of Endpoints:	1899	Total Number of Endpoints:	902

All user specified timing constraints are met.

Para comprobar la frecuencia máxima de funcionamiento del diseño, se utiliza una técnica iterativa. Se implementa el circuito mediante Vivado, y cada vez se establece como objetivo una frecuencia superior, de manera que la herramienta tenga que optimizar la posición de los elementos e interconexiones en la FPGA hasta que ya no sea capaz de hacerlo más. Se puede comprobar que esto se cumple para hasta 150MHz, cuando el margen de tiempo de la señal más lenta es de solamente 16ps. Con este resultado se constata que el módulo observador diseñado cumple también el objetivo de ser rápido. Además, existen técnicas de optimización capaces de aumentar la frecuencia de funcionamiento de los diseños, de modo que si en el futuro esto fuera necesario, podrían aplicarse.

Capítulo 5
Resultados experimentales

5.1 Introducción

Una vez obtenido el sistema definitivo, es necesario probar su correcto funcionamiento desde el punto de vista de las especificaciones de partida. Para ello es necesario diseñar distintos experimentos que permitan asegurar o descartar la validez del módulo diseñado para la utilidad que se pretende. Serán necesarios dos experimentos distintos:

El primero servirá de comprobación de que la interfaz de traza ha sido configurada correctamente y se encuentra activa. Para ello utilizaremos técnicas de depuración hardware capaces de acceder a los valores de las señales internas del circuito y así comparar el comportamiento de la traza cuando se encuentra activa y cuando no.

El segundo, validará el módulo observador. Una vez se tenga la certeza de que la interfaz de traza está activa y proporciona información concorde a la esperada, se procederá en dos fases:

1. Simulación comportamental del módulo diseñado utilizando para ello información real de la interfaz de traza. Con esta prueba se podrá revisar completamente el funcionamiento interno del módulo, que permitirá detectar la mayoría de los errores de diseño otorgando un primer punto de control. En caso de que la simulación arroje un veredicto satisfactorio, los resultados obtenidos se compararán con los de la siguiente fase.
2. Prueba de campo. El sistema completo se implementa sobre la ZYNQ y el funcionamiento del módulo observador se depura en hardware, obteniendo los valores reales de sus señales internas. En caso de que estos valores se correspondan con los obtenidos en la simulación, se podrá inferir que la simulación representa fielmente el funcionamiento real del módulo y que éste es correcto.

Para realizar ambos los experimentos, se utilizará un diseño implementado para funcionar a 100MHz.

En cada prueba, el microprocesador del diseño debe ejecutar un código conocido que servirá de test para las capacidades del módulo diseñado. Este código de prueba debe contener las instrucciones que el módulo está diseñado para reconocer, tales son salto condicional e incondicional.

5.2 Software empleado

La siguiente figura representa el software utilizado mediante un diagrama de flujo. Aunque el módulo observador diseñado podría trazar cualquier código, se ha utilizado un programa muy sencillo que facilita la verificación del funcionamiento del módulo observador, así como la comprensión de su funcionamiento. Contiene siete bloques de instrucciones libres de saltos (BBs), en azul; instrucciones de decisión (salto condicional), en verde; así como un bucle infinito (salto incondicional), en rojo.

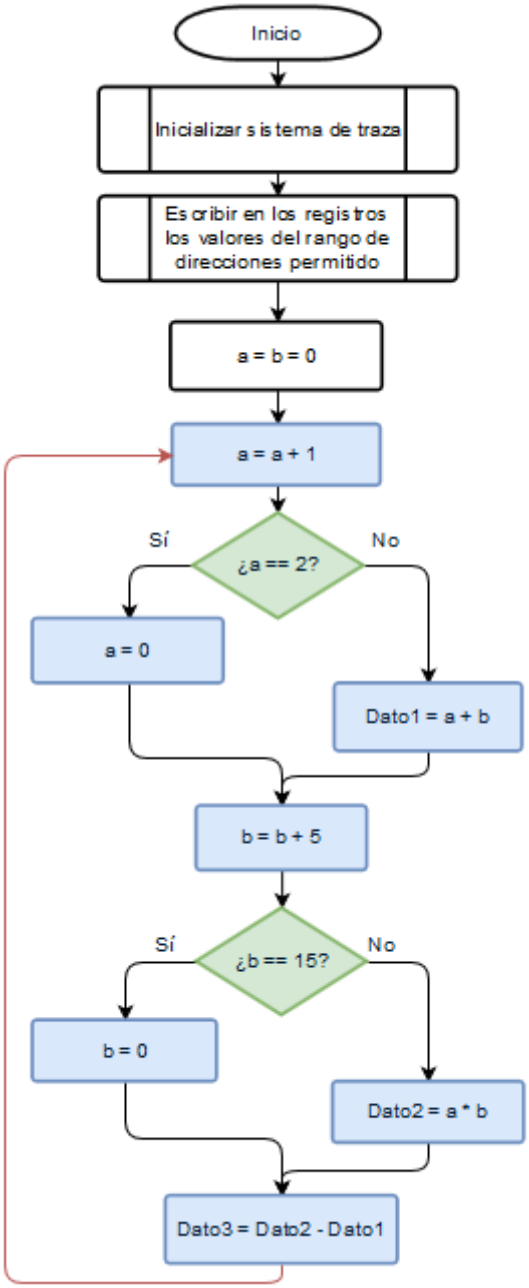


Figura 5-1 Diagrama de flujo del software de prueba

5.3 Experimentos realizados

5.3.1 Depuración de las señales internas

5.3.1.a) Objetivos

La selección de los parámetros de configuración de la interfaz de traza se ha basado en la información procedente de la documentación, sin embargo es necesario comprobar su correcto funcionamiento para la configuración elegida, así como cerciorarse de que su comportamiento es el esperado. Para ello se utiliza la característica de depuración hardware incluida en Vivado.

5.3.1.b) Desarrollo

El procedimiento [23] para obtener las formas de onda es el siguiente:

1. Sintetizar el diseño de la Figura 4-12.
2. Seleccionar las señales que se desean depurar de la *netlist* generada. Éstas son las señales de la TPIU: TRACE_CLK, TRACE_CTL y TRACE_DATA.
3. Utilizar el asistente de configuración de Vivado, que añade de forma automática los dispositivos necesarios para realizar la depuración.
4. Continuar el proceso de implementación y generación del bitstream.
5. Programar la FPGA con el bitstream.
6. Abrir el Hardware Manager de Vivado, y activar la depuración para obtener las formas de onda.

Esta metodología se aplicará para dos casos: traza activa y traza inactiva.

5.3.1.c) Resultados

En primer lugar se elimina la parte del código que configura la traza, con la intención de observar el comportamiento de la interfaz de traza cuando se encuentra inactiva. Puede verse en la siguiente figura.

Figura 5-2 Captura de las señales de traza inactiva

Se obtienen los resultados esperados: TRACE_CTL fija en ‘1’ y TRACE_DATA con valor “01”, en consonancia con la Tabla 4-5, que indica traza inactiva.

A continuación, se restaura el código ejecutado para recuperar la parte que configura la interfaz de traza. Se espera que el comportamiento del sistema cambie en comparación con el experimento anterior. Los resultados se muestran en la siguiente figura.

Figura 5-3 Captura de las señales de traza activa

Efectivamente, el comportamiento del puerto ha cambiado: la señal TRACE_CTL alterna su valor, como también lo hace TRACE_DATA. Se extrae la siguiente información:

- La interfaz de traza no permanece activa durante todo el tiempo de ejecución, sino que únicamente se activa cuando tiene información disponible que emitir.
  - Cuando hay información disponible, TRACE_CTL vale '0' y la información sale por TRACE_DATA.
  - Cuando no la hay, el comportamiento es idéntico a la Figura 5-2, traza inactiva.
- Los valores de TRACE_DATA y TRACE_CTL se actualizan en el flanco de subida de TRACE_CLK y son estables en el flanco de bajada, confirmando la información de la Figura 4-7 Cronograma de las señales de la TPIU por EMIO Figura 4-7.
- Los valores que toma la señal TRACE_DATA se pueden corresponder, con ayuda de la Tabla 4-3, con los encabezados de algunos de los paquetes de la ARM PFT *Architecture*:
  - 0x86 = b1000 0110, esto es un *Atom*.
  - 0x08 = b0000 1000, esto es el encabezado de un *I-Sync*.

Así pues, se concluye que la interfaz de traza se encuentra correctamente configurada, así como que tiene el comportamiento inicialmente esperado. Este resultado permite continuar hacia el siguiente experimento.

5.3.2 Comportamiento del módulo diseñado

5.3.2.a) Objetivos

Una vez comprobadas las condiciones en las que va a operar el módulo observador, el siguiente paso es someterlo a una prueba de funcionamiento en la ubicación para la que ha sido diseñado: conectado al puerto de la TPIU. Para ello lo más inmediato es implementar el diseño de la Figura 4-12 en la FPGA y depurar las señales internas.

Sin embargo, por sí sola, esta prueba no va a arrojar toda la luz posible sobre el funcionamiento interno del observador, ya que durante el proceso de implementación del circuito en la FPGA las señales utilizadas para su diseño se optimizan y llegan a perder su significado. Para realizar estas comprobaciones es mucho más apropiado el uso de un simulador.

Las herramientas de simulación son muy útiles en todos los niveles de desarrollo de un sistema electrónico. Existen distintos tipos que dependen del diseño realizado así como de la etapa de desarrollo en la que se encuentre. En este caso se va a utilizar la herramienta de simulación comportamental de hardware Vivado Simulator. Con la ayuda de un fichero de banco de pruebas, es posible obtener el comportamiento de cualquier diseño hardware descrito en VHDL a partir del valor de dado de las entradas. A partir de este valor, el simulador completará el funcionamiento del sistema, infiriendo el valor de las distintas señales a través de relaciones causales.

En sus inicios, el lenguaje VHDL se utilizaba únicamente para realizar simulaciones, y fue posteriormente cuando se desarrollaron los software de síntesis e implementación. Los algoritmos que los gobiernan están diseñados para generar un circuito digital equivalente al comportamiento del código sometido a una simulación. Resulta por tanto suficiente la obtención de datos de simulación para inferir el funcionamiento real de un diseño.

Se realizará el experimento en dos fases: en primer lugar una simulación del comportamiento del sistema facilitará la comprensión de su funcionamiento para posteriormente contrastar los resultados de la simulación con los datos reales obtenidos por depuración del hardware.

5.3.2.b) Desarrollo

Para realizar la simulación es preciso contar con el valor de las entradas en todo momento. De la Figura 4-11 se extraen las diferentes entradas al módulo observador:

- Bus AXI
- TRACE_CTL
- TRACE_DATA
- TRACE_CLK
- Reset

Sin embargo, ARM, como inventora del bus AXI, no permite incluirlo en simulaciones a no ser que se abone una licencia de modelo funcional.

En este momento queda patente la utilidad de realizar un diseño jerárquico del módulo observador: simular la entidad de nivel superior, con el bus AXI incorporado requiere una costosa licencia, pero es posible simular la entidad inmediatamente inferior sin necesidad de ella. En la Figura 4-11 también se detallan las entradas de esta entidad:

- START_ADDR
- END_ADDR
- TRACE_CTL
- TRACE_DATA
- TRACE_CLK
- Reset

En este caso sí es posible realizar la simulación al no haber en el diseño ningún componente de propiedad intelectual protegido.

El valor de cada entrada debe ser establecido a partir de evidencias fiables de cara a obtener resultados representativos:

- Reset. Cualquier señal de reset se utiliza para inicializar los registros a un valor conocido. Esta señal se activará por nivel alto al comienzo de la simulación y quedará desactivada el resto del tiempo.
- START_ADDR y END_ADDR. Estos registros son escritos desde el software a través del bus AXI, y una vez escritos permanecen constantes. Los mismos valores que escribe el software usaremos en la simulación.
- TRACE_CLK. Será una señal de reloj idéntica a la obtenida en el experimento del apartado 5.3.1: 100MHz con un 50% del tiempo en nivel alto y el otro 50% en nivel bajo.
- TRACE_CTL y TRACE_DATA. Sobre estas señales hay información disponible de varias fuentes:
  - Con base en la Figura 4-7 conocemos que se actualizan en flanco de bajada y son estables en flanco de subida de TRACE_CLK.
  - Con base en el experimento 5.3.1, así como en la Tabla 4-5, el comportamiento de estas señales puede resumirse en la siguiente tabla.

Tabla 5-1 Comportamiento de las señales TRACE_DATA y TRACE_CTL

TRACE_CTL		TRACE_DATA
‘1’	Traza inactiva	0x01
‘0’	Traza activa	0xXX

Por tanto, será condición necesaria y suficiente establecer para cada señal valores que concuerden con dicho comportamiento.

- TRACE_CTL. Se dispondrán valores al azar para esta señal que se alternarán en los flancos de subida de TRACE_CLK.

▪ TRACE_DATA: Para obtener los valores reales de traza del código se utilizará el ETB, que como ya se explicó anteriormente en 3.3.3 se trata de un componente de CoreSight capaz de almacenar la información de traza en una memoria de 4kB. Una vez activado, es posible acceder a su contenido mediante acceso a su registro de lectura, mapeado en memoria. La siguiente tabla contiene un extracto de la información obtenida del ETB.

Tabla 5-2 Información de traza extraída del ETB

1	2	3	4	5	6
0x00000000	0x08800000	0x00100A88	0x0BCB6B21	0x71860AC5	0x0AC50BCB
7	8	9	10	11	12
0x0BD3866B	0xCB6B0AC5	0x860AC50B	0xC50BCB71	0x0BCB6B0A	0x71860AC5
13	14	15	16	17	…
0x0AC50BCB	0xC50BCB6B	0xCB71860A	0x6B0AC50B	0x0AC50BCB	…

Los valores obtenidos del ETB son de 32 bits, de forma que se dividirán en palabras de 8 bits y se leerán consecutivamente para colocarse en la señal TRACE_DATA en cada flanco de subida de TRACE_CLK siempre que el valor de TRACE_CTL sea '0'; de lo contrario se hará un alto en la lectura para insertar el valor "0x01", tal y como indica la Tabla 5-1, y continuar leyendo una vez TRACE_CTL haya vuelto al valor '0'.

5.3.2.c) Resultados

Mediante un banco de pruebas es posible comprobar el funcionamiento del módulo observador utilizando los valores de cada señal detallados en el apartado anterior. La Figura 5-4 contiene una captura de la simulación resultante; en ella aparecen la mayoría de las señales de la Figura 4-9 y la Figura 4-10. A continuación se muestra un resumen del significado de cada una de ellas:

- Reset. Señal de inicialización de los registros del circuito.
- TRACE_CLK, TRACE_CTL y TRACE_DATA reciben las señales directamente de la TPIU. TRACE_DATA se muestrea en DATA_FALLING.
- TRACE_CTL_2 y TRACE_DATA_2. Registran el valor de sus antecesoras para la segunda etapa del pipeline.
- CURRENT_PAK y NEXT_PAK son las señales de funcionamiento de la máquina de estados que permite conocer el tipo de paquete en curso.
- BYTE_COUNT y SECTION_COUNT distinguen en el interior de cada paquete el número de cada Byte y la sección en la que se encuentra para decodificarlo.
- LAST_COMPLETE_ADDR es la última posición conocida del PC.
- Error. Se activa cuando la posición del PC ha salido de los límites de control.

La siguiente figura contiene un extracto de la simulación realizada.

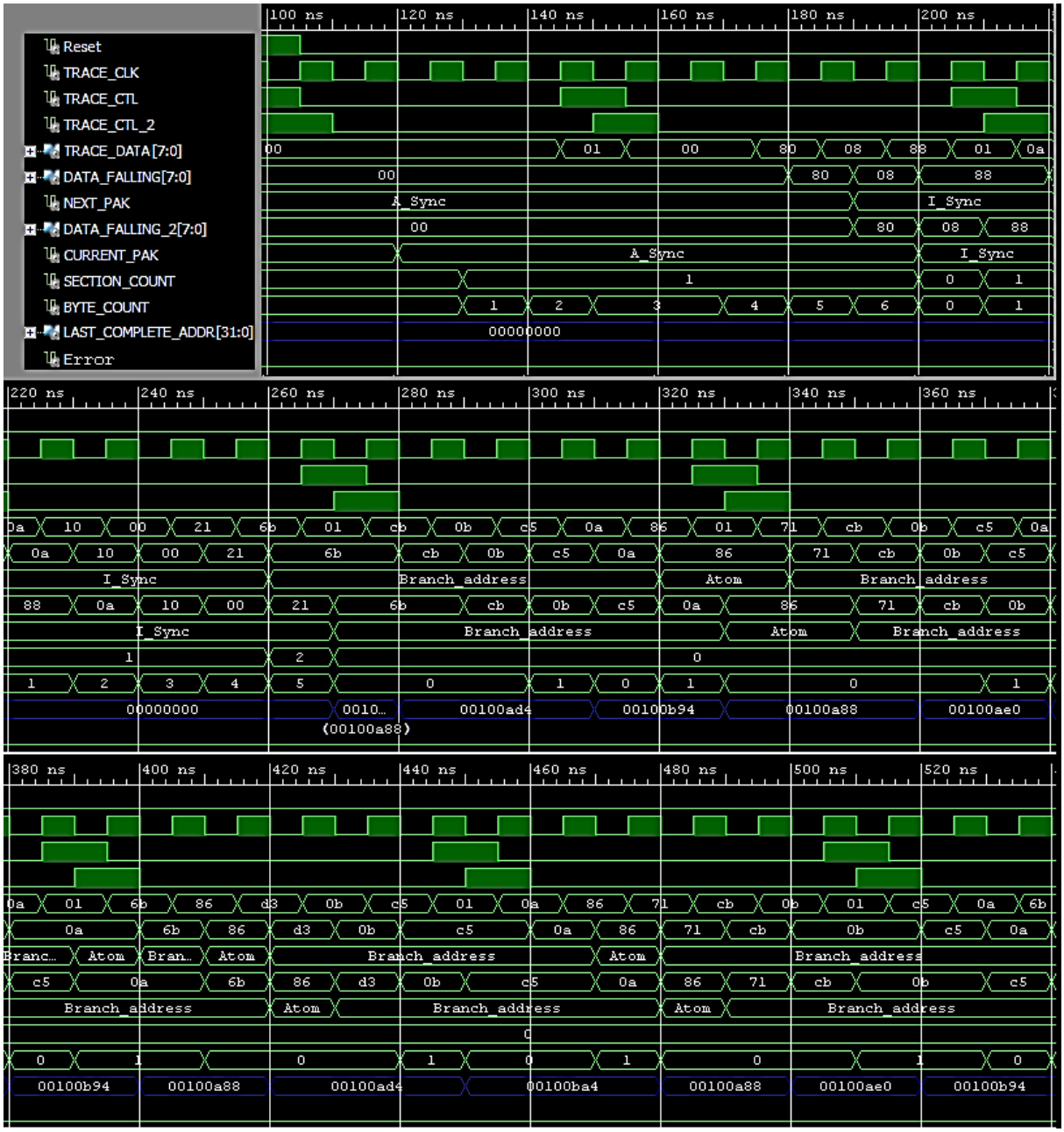


Figura 5-4 Simulación del módulo observador

En ella se puede ver el comportamiento del observador, distinguiendo fácilmente

- Las transiciones entre distintos tipos de paquete, y la cuenta de Bytes y secciones.
- La correcta gestión de los valores de TRACE_CTL.
- Las sucesivas direcciones conocidas del PC, marcadas en color azul.
    - Es preciso recordar, que la PTM genera información sobre la posición del PC cada vez que se produce un salto en la ejecución mediante un paquete *Branch Address*, así como periódicamente mediante un paquete *I-Sync*.

La siguiente tabla resume la progresión de posiciones del PC, distinguiendo con colores el tipo de paquete que genera cada nueva dirección.

Tabla 5-3 Posiciones del PC obtenidas por simulación

1 (I-Sync)	2 (Br. Addr)	3 (Br. Addr)	4 (Br. Addr)	5 (Br. Addr)
0x00100A88	0x00100AD4	0x00100B94	0x00100A88	0x00100AE0
6 (Br. Addr)	7 (Br. Addr)	8 (Br. Addr)	9 (Br. Addr)	10 (Br. Addr)
0x00100B94	0x00100A88	0x00100AD4	0x00100BA4	0x00100A88
11 (Br. Addr)	12 (Br. Addr)	...		
0x00100AE0	0x00100B94	...		

Con esta información se completa la primera fase del experimento, ahora se debe implementar el diseño sobre la placa de desarrollo y realizar depuración hardware sobre él. Para ello se siguen los mismos pasos que en 5.3.1, obteniendo el siguiente resultado.

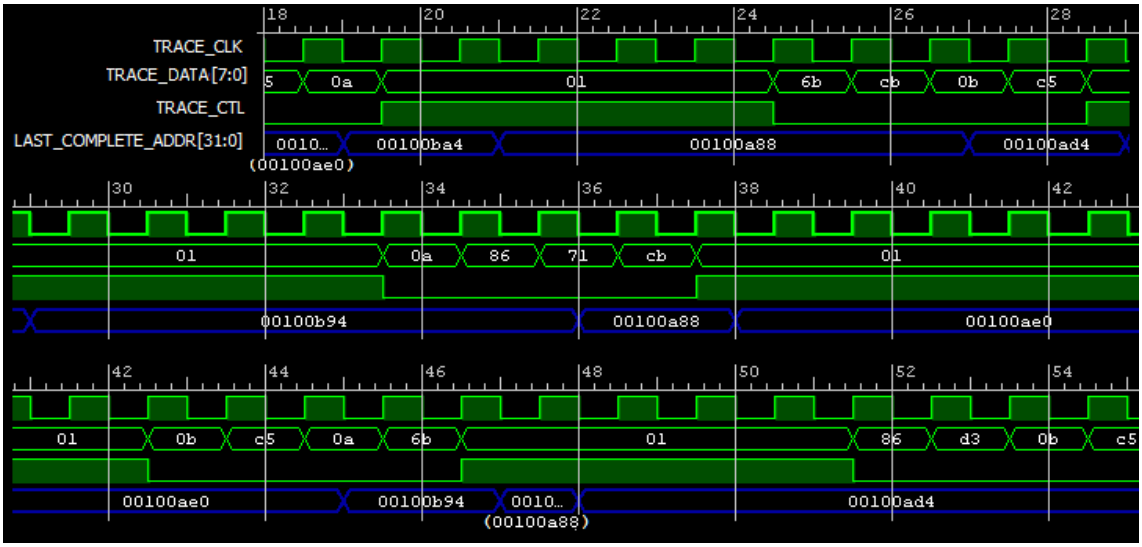


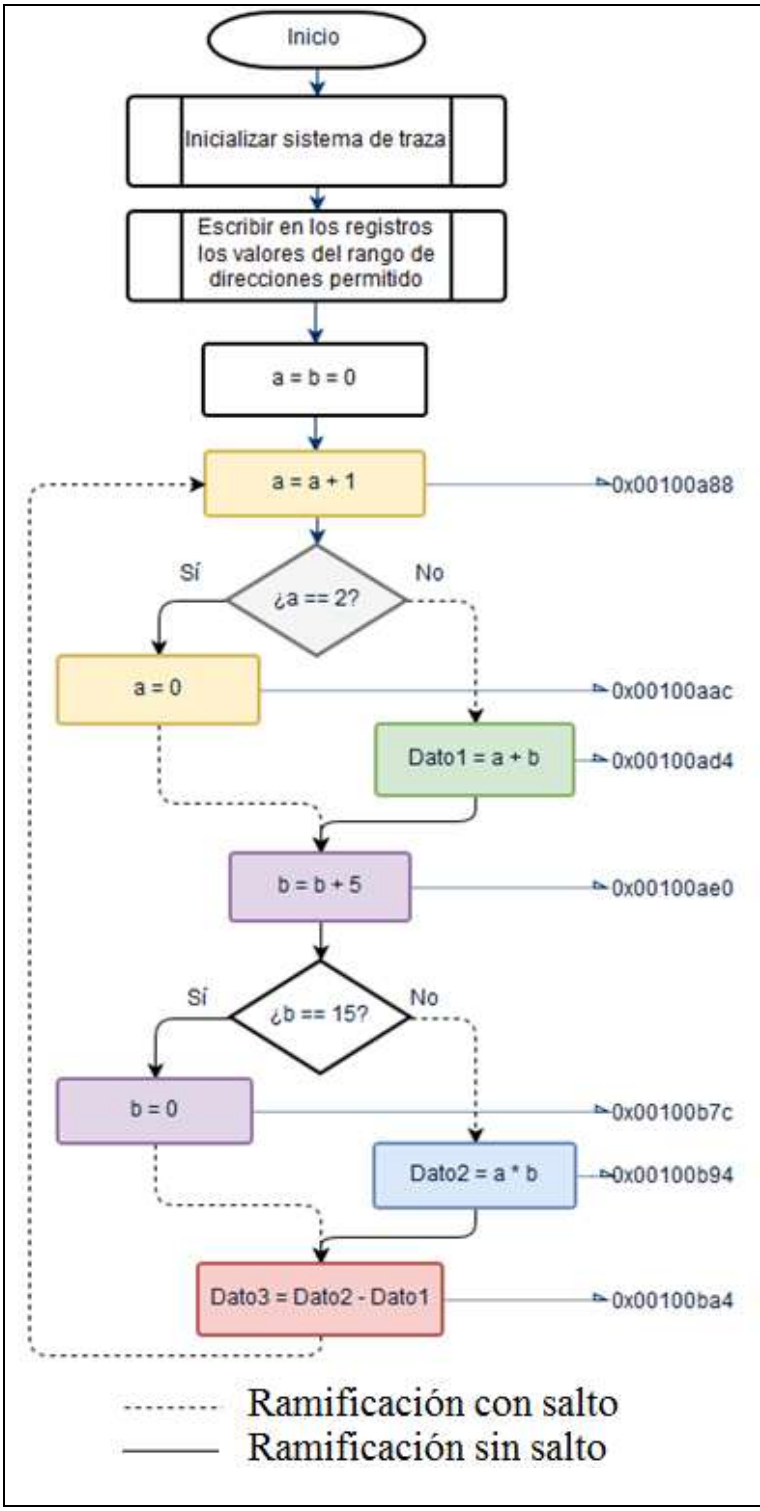
Figura 5-5 Comportamiento real del módulo observador.

A diferencia de la simulación de la Figura 5-4, los datos de esta captura pertenecen al funcionamiento real del sistema, es decir, son las señales que efectivamente circulan por el interior del circuito.

Puede comprobarse que las distintas posiciones del PC se corresponden con el comportamiento de la simulación anterior resumido en la Tabla 5-3, donde aparecen las distintas direcciones del PC previamente conocidas. Si bien la correlación obtenida en esta figura no se corresponde de manera exacta con la obtenida en la simulación esto se debe únicamente a que se han capturado momentos diferentes de la ejecución.

Este fenómeno se ilustra en la Tabla 5-4 de la página siguiente, en la que se utiliza la información de la Tabla 5-3 para reconstruir la ejecución de varias iteraciones del programa utilizando información del código desensamblado para conocer las posiciones de inicio de cada uno de los BBs. Se puede observar que, dependiendo de los valores de las variables a y b, la combinación de saltos en cada iteración es distinta.

Tabla 5-4 Correlación de la información de traza con la ejecución del programa



a	b		
0	0	a=0, b=0	
0	0	0x00100a88	1
1	0	a = 0 + 1 = 1	
1	0	¿a == 2? NO	
1	0	0x00100ad4	2
1	0	Dato1 = 1	
1	5	b = 0 + 5 = 5	
1	5	¿b == 15? NO	
1	5	0x00100b94	3
1	5	Dato2 = 5	
1	5	Dato3 = 4	
1	5	0x00100a88	4
2	5	a = 1 + 1 = 2	
2	5	¿a == 2? SI	
0	5	a = 0	
0	5	0x00100ae0	5
0	10	b = 5 + 5 = 10	
0	10	¿b == 15? NO	
0	10	0x00100b94	6
0	10	Dato2 = 0	
0	10	Dato3 = -1	
0	10	0x00100a88	7
1	10	a = 0 + 1 = 1	
1	10	¿a == 2? NO	
1	10	0x00100ad4	8
1	10	Dato1 = 11	
1	15	b = 10 + 5 = 15	
1	15	¿b == 15? SI	
1	0	b = 0	
1	0	0x00100b94	9
1	0	Dato3 = -11	
1	0	0x00100a88	10
2	0	a = 1 + 1 = 2	
2	0	¿a == 2? SI	
0	0	a = 0	
0	0	0x00100ae0	11
0	5	b = 0 + 5 = 5	
0	5	¿b == 15? NO	
0	5	0x00100b94	12
0	5	…	

Cabe mencionar que no todas las direcciones de inicio de un BB aparecen en la traza, ya que las direcciones 0x00100AAC y 0x00100B7C no son destino de ningún salto, y por tanto ningún paquete *Branch Address* las incorpora.

Se comprueba que el flujo del programa se puede seguir mediante la información obtenida en los experimentos, por lo tanto el módulo observador cumple su función.

5.4 Conclusiones

A través de los experimentos se ha conseguido verificar satisfactoriamente la consecución de todos los objetivos, así como un desempeño adecuado de todas las prestaciones implementadas en el módulo diseñado. Para todo ello se ha diseñado un software de prueba que contiene únicamente los elementos necesarios para validar el funcionamiento del sistema.

A continuación se resumen los resultados clave del proceso experimental:

- La interfaz de traza se encuentra operativa.
- La configuración utilizada para la interfaz de traza es correcta, ya que proporciona la información deseada.
- La información real de traza ha sido capturada y utilizada para simular el funcionamiento del módulo observador en las condiciones más próximas a la realidad posibles, y obteniendo un resultado positivo de esta rigurosa simulación.
- El resultado de la simulación ha sido contrastado con los valores reales de las señales del sistema implementado sobre la ZYNQ con idéntico resultado, lo que prueba que la simulación representa el funcionamiento real del sistema, validando el sistema completo.

El conjunto de los resultados confirma que el desarrollo realizado ha sido un éxito.

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

El desarrollo de este Trabajo Fin de Frado ha significado un acercamiento sin precedentes a la labor investigadora, al mismo tiempo que ha permitido al autor profundizar en el conocimiento de nuevos conceptos y disciplinas en el campo de la Tecnología Electrónica mediante la realización de un proyecto ambicioso de interés científico.

Durante su elaboración, el autor ha tenido la oportunidad de iniciarse en el aprendizaje de nuevas herramientas, como el software informático utilizado durante el proceso de desarrollo, así como nuevas metodologías de trabajo, como las relativas a sistemas empotrados. De igual manera se han abordado conceptos y tecnologías propias de ámbitos especializados de la electrónica, como la tolerancia a fallos en circuitos digitales, y su intensificación en el estudio de la interfaz de traza.

La documentación ha sido imprescindible en todas las fases, por una parte para comprender el estado actual de la técnica de tolerancia a fallos, permitiendo ubicar el diseño realizado dentro de una línea de investigación actual. Por otra, constituyendo una guía para conocer los detalles del funcionamiento de cada uno de los subsistemas utilizados durante el trabajo, imprescindible para lograr configurar su comportamiento.

Todo ello, unido a una sólida base de conocimientos técnicos adquirida durante la realización del Grado en Ingeniería Electrónica Industrial y Automática, ha dotado al alumno de la capacidad suficiente para cumplir con los objetivos planteados.

Un módulo observador con las características obtenidas puede sentar un punto de partida para futuras investigaciones y proyectos mucho más ambiciosos, que combinando las técnicas de endurecimiento existentes con las propias de la tecnología de ARM pueden suponer grandes mejoras en los productos basados en arquitecturas ARM complejas, fuertemente integradas en el mercado de la electrónica de consumo.

Los resultados muestran que el observador diseñado es válido para trazar todos los paquetes de la especificación ARM PFT *Architecture*, lo que le otorga un gran potencial de cara a futuros desarrollos, como se describe más adelante.

Sin embargo, el desarrollo del trabajo no ha estado exento de dificultades:

- La primera y mayor, la carencia de documentación sobre desarrollos similares, lo que ha obligado a iniciar la investigación desde la raíz: los manuales de producto de los distintos subsistemas, y a partir de ahí asentar los conceptos básicos en los que se apoya el módulo diseñado. Es evidente, que de haber existido otros desarrollos, este trabajo tendría que haber tenido otro planteamiento, probablemente más ambicioso, al partir desde un punto fiable, también más alto.
- En segundo lugar, la abrumadora cantidad de información presente en los manuales, y sobre todo, su dispersión. La presencia de tres fabricantes distintos involucrados en la placa de desarrollo utilizada ha requerido de un extenso análisis de la documentación proporcionada por cada uno para discernir en cada caso la información aplicable a este trabajo, de entre miles de páginas.
- En tercer lugar, la novedad de los sistemas utilizados durante el trabajo, tanto las herramientas informáticas de diseño como los sistemas hardware eran totalmente desconocidos hasta el comienzo del trabajo, así como la disciplina de tolerancia a fallos en microprocesadores y todos los conceptos asociados con ella.
- Por último, la dificultad para testar el diseño realizado, al requerir licencias de pago por parte de ARM para realizar comprobaciones en simulación ha sido necesario buscar una alternativa viable, como la depuración hardware.

Con todo, el desafío que supone se ha visto encarado a través de la combinación de dos variables; por una parte, la satisfacción de profundizar en el conocimiento de un ámbito de interés para el alumno, como es la electrónica digital; y por otra, el aliento que ha supuesto ver cómo, poco a poco, el trabajo realizado ha ido dando resultados a pesar de las dificultades.

El éxito del resultado final obtenido, no muestra si no el fruto del esfuerzo del alumno, siendo capaz de abordar problemas nuevos a partir de los conocimientos adquiridos en el Grado, así como de incorporar nuevos conocimientos técnicos relacionados con el ámbito de la electrónica, cumpliendo con los resultados del aprendizaje esperados del Trabajo Fin de Grado.

6.2 Líneas futuras

El desarrollo de este Trabajo Fin de Grado sienta la base para realizar nuevos diseños y experimentos que enriquezcan la arquitectura del observador propuesto, con el objetivo de cuantificar su nivel de tolerancia a fallos así como la posibilidad de incrementar este nivel mediante la adición de nuevas características. Varias de las siguientes propuestas de líneas futuras se desarrollarán próximamente en el marco de las investigaciones del Grupo de Diseño Microelectrónico y Aplicaciones de la Universidad Carlos III de Madrid.

6.2.1 Línea futura: ampliar el módulo observador

El componente principal del módulo observador es un decodificador de paquetes, capaz de discernir todos los tipos recogidos en la especificación ARM PFT *Architecture*, sin embargo actualmente esta característica está infrautilizada. El módulo diseñado en este trabajo únicamente hace uso de dos tipos de paquete, lo que limita sus posibilidades de detección de errores al no tener en cuenta, por ejemplo, paquetes relacionados con interrupciones.

Añadir progresivamente soporte al resto de tipos de paquetes aumentaría significativamente las prestaciones del módulo con un esfuerzo pequeño en comparación con el realizado durante este trabajo.

6.2.2 Línea futura: perfeccionar el diseño

El módulo observador diseñado, se ha desarrollado siguiendo en todo momento parámetros teóricos tanto de especificaciones como de objetivos, sin embargo no ha sido puesto a prueba en una aplicación software real trabajando en un entorno cambiante con sensores y actuadores.

Es posible que el desarrollo actual no contemple todas las prestaciones necesarias para realizar correctamente su labor en el ámbito de tolerancia a fallos. Investigar en esta dirección puede desembocar en la identificación de nuevas características cruciales en su diseño que de otra manera quedan inexploradas.

6.2.3 Línea futura: validación del diseño

Pese a que se ha comprobado experimentalmente que el módulo resultante del diseño funciona tal y como se planteó inicialmente, no se han realizado con él pruebas para determinar su validez en cuanto a capacidad de endurecimiento del sistema.

Una de las labores fundamentales de validación de circuitos tolerantes a fallos es realizar pruebas controladas de detección de errores mediante técnicas de inyección o simulación. Conociendo los errores que se introducen en el sistema, es posible determinar la tasa de detección alcanzada, y los datos obtenidos de estos test pueden servir a su vez para perfeccionar el diseño del módulo observador propuesto.

6.2.4 Línea futura: experimentos de radiación

El objetivo último del diseño de nuevas técnicas de endurecimiento es proporcionar cobertura a fallos transitorios provocados por impactos de partículas energéticas. Una vez alcanzada una alta tasa de cobertura de fallos a través de experimentos de inyección se puede testar su validez en una situación real gracias a los sistemas empotrados programables (SOPC).

Existen infraestructuras diseñadas específicamente para someter a los circuitos a este tipo de radiaciones con el objetivo de probar que cumplen su finalidad última. El Grupo de Diseño Microelectrónico y Aplicaciones realiza anualmente experimentos de este tipo con sus nuevos desarrollos y es su intención llevar un prototipo derivado de este trabajo a un ensayo de radiación a finales de 2016.

6.2.5 Utilizar traza con varios orígenes

El grado alcanzado por el diseño se limita a una configuración de traza en la que se recibe la información generada por un único procesador. Sin embargo las características del sistema permiten implementaciones mucho más complejas, utilizando varias fuentes de traza.

Esta vía contempla el uso no solo de la traza procedente de ambos procesadores sino también de la ITM y el FTM para integrar en una única solución información de dos flujos de ejecución distintos, así como de determinados datos del programa y parámetros de depuración de la lógica programable, pudiendo dar lugar a nuevas técnicas de detección y corrección de errores inéditas hasta ahora.

6.2.6 Línea futura: integración en técnicas híbridas

La integración de los resultados de cualquiera de las líneas futuras anteriores con otras técnicas ya existentes permitiría confeccionar técnicas híbridas variadas, con capacidad de adaptarse a múltiples escenarios.

Por otra parte, las características únicas de la ZYNQ permiten pensar en desarrollos de módulos observadores de gran complejidad, pudiendo desembocar en el desarrollo de técnicas híbridas novedosas, dadas sus características únicas en cuanto a posibles fuentes de traza para el observador así como la disponibilidad de dos potentes microprocesadores.

Se puede lograr por ejemplo, el endurecimiento de una aplicación mediante redundancia total (dos microprocesadores iguales ejecutando códigos idénticos) e implementar el circuito votador en la PL, o combinando la información de los distintos orígenes, como un sistema operativo en tiempo real sobre un sistema empotrado dedicado a una aplicación de seguridad crítica.

Acrónimos

A

AMBA – Advanced Microcontroller Bus Architecture

APB – Advanced Peripheral Bus

ARC – Address Range Comparator

ATB – AMBA Trace Bus

AXI – Advanced eXtensible Bus

B

BB – Branch-free Block

C

CAN – Controller Area Network

CFC – Control-Flow Checking

CMOS – Complementary Metal-Oxide-Semiconductor

COTS – Commercial Off-The-Shelf

D

DMA – Direct Memory Access

E

ECT – Embedded Cross Trigger

EMIO – Extended Multiplexed Input / Output

ETB – Embedded Trace Buffer

F

FIFO – First In, First Out

FPGA – Field Programmable Gate Array

FPU – Floating-Point Unit

FTM – Fabric Trace Monitor

G

GPIO – General-Purpose Input / Output

GPU – Graphics Processing Unit

H

HDL – Hardware Description Language

HDMI – High-Definition Multimedia Interface

HLS – High Level Synthesis

I

I2C – Inter-Integrated Circuit

IP – Intellectual Property

ITM – Instrumentation Trace Macrocell

L

LED – Light-Emmiting Diode

LSB – Least Significant Bit

LUT – Look-Up Table

M

MBU – Multiple-Bit Upset

MCU – Multiple-Cell Upset

MIO – Multiplexed Input / Output

MSB – Most Significant Bit

O

OTG – On The Go

P

PC – Program Counter

PFT – Program Flow Trace

PL – Programmable Logic

PLL – Phase Locked Loop

PMOD – Peripheral MODules

PS – Processing System

PTM – Program Trace Macrocell

R

RAM – Random Access Memory

S

SAC – Single Address Comparator

SDIO – Secure Digital Input Output

SDK – Software Development Kit

SEFI – Single-Event Functional Interrupt

SEL – Single-Event Latch-up

SET – Single-Event Transient

SEU – Single-Event Upset

SIMD – Single Instruction, Multiple Data

SoC – System on Chip

SOPC – System On Programmable Chip

SPI – Serial Peripheral Interface

T

TPIU – Trace Port Interface Unit

U

UART – Universal Asynchronous Receiver-Transmitter

USB – Universal Serial Bus

V

VGA – Video Graphics Array

X

XADC – Xilinx Analog to Digital Converter

Bibliografía

[1] B. W. Johnson, Design and Analysis of Fault-Tolerant Digital Systems, Addison-Wesley, 1989.

[2] R. Velazco, P. Fouillat y R. Reis, Radiation effects on embedded systems, Springer Science & Business Media, 2007.

[3] M. Nicolaidis, Soft Errors in Modern Electronic Systems, Springer Science & Business Media, 2010.

[4] L. Parra, A. Lindoso y L. Entrena, Comparative of software-based hardening techniques for LEON 3 microprocessor, Leganés, España: IEEE, 2014.

[5] B. Du, M. S. Reorda, L. Sterpone, L. Parra, M. Portela-García, A. Lindoso y L. Entrena, «Online Test of Control Flow Errors: A New Debug Interface-Based Approach,» *IEEE Transactions on Computers*, vol. 65, nº 6, pp. 1846-1855, 2016.

[6] L. Parra, A. Lindoso, M. Portela, L. Entrena, F. Restrepo-Calle, S. Cuenca-Asensi y A. Martínez-Álvarez, «Efficient mitigation of data and control flow errors in microprocessors,» *IEEE Transactions on Nuclear Science*, vol. 61, nº 4, pp. 1590-1596, 2014.

[7] L. Parra, A. Lindoso, M. Portela-García, L. Entrena, B. Du, M. S. Reorda y L. Sterpone, «A new hybrid nonintrusive error-detection technique using dual control-flow monitoring,» *IEEE Transactions on Nuclear Science*, vol. 61, nº 6, pp. 3236-3243, 2014.

[8] L. Entrena, A. Lindoso, M. Portela-García, L. Parra, B. Du, M. S. Reorda y L. Sterpone, «Fault-Tolerance Techniques for Soft-Core Processors Using the Trace Interface,» de *FPGAs and Parallel Architectures for Aerospace Applications*, Springer International Publishing, 2016, pp. 293-306.

[9] L. H. Crockett, R. A. Elliot, M. A. Enderwithz y W. R. Stewart, «The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 all Programmable Soc,» 2014. [En línea].

Disponible en: http://www.zynqbook.com/. [Último acceso: Septiembre 2016].

[10] L. A. Tambara, P. Rech, E. Chielle, J. Tonfat y F. L. Kastensmidt, «Analyzing the Impact of Radiation-Induced Failures in Programmable SoCs,» *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, vol. 63, nº 4, pp. 2217-2224, 2016.

[11] Xilinx, Inc, «DS190 Zynq-7000 All Programmable SoC Overview,» v1.9, Enero 2016. [En línea].

Disponible en: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf. [Último acceso: Septiembre 2016].

[12] Xilinx, Inc, «All Programmable SoCs and MPSoCs,» web, [En línea].

Disponible en: http://www.xilinx.com/products/silicon-devices/soc.html. [Último acceso: Septiembre 2016].

[13] Xilinx, Inc, «XMP097 Zynq-7000 All Programmable SoCs Product Tables and Product Selection Guide,» 2015. [En línea].

Disponible en: http://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf. [Último acceso: Septiembre 2016].

[14] Digilent, Inc, «Zybo Resource Center,» web, [En línea].

Disponible en: https://reference.digilentinc.com/reference/programmable-logic/zybo. [Último acceso: Septiembre 2016].

[15] Digilent, Inc, «ZYBO™ FPGA Board Reference Manual,» Abril 2016. [En línea].

Disponible en: https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf.

[Último acceso: Septiembre 2016].

[16] Xilinx, Inc, «UG585 Zynq-7000 All Programmable SoC Technical Reference Manual,» v1.10, Febrero 2015. [En línea].

Disponible en: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. [Último acceso: Septiembre 2016].

[17] ARM, «Cortex-A9 MPCore Technical Reference Manual,» v r4p1, Enero 2016. [En línea].

Disponible en: http://infocenter.arm.com/help/topic/com.arm.doc.100486_0401_10_en/cortex_a9_mpcore_trm_100486_0401_10_en.pdf.

[Último acceso: Septiembre 2016].

[18] ARM, «CoreSight Technical Introduction White Paper,» Agosto 2013. [En línea]. Disponible en: http://infocenter.arm.com/help/topic/com.arm.doc.epm039795/coresight_technical_introduction_EPM_039795.pdf. [Último acceso: Septiembre 2016].

[19] ARM, «DDI0314H CoreSight™ Components Technical Reference Manual,» Julio 2009. [En línea]. Disponible en: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0314h/DDI0314H_coresight_components_trm.pdf. [Último acceso: Septiembre 2016].

[20] ARM, «CoreSight Creator,» web, [En línea].

Disponible en: https://www.arm.com/products/system-ip/ip-tooling/coresight-creator.php. [Último acceso: Septiembre 2016].

[21] ARM, «CoreSight SoC-400 Debug & Trace,» web, [En línea].

Disponible en: https://www.arm.com/products/system-ip/debug-trace/coresight-soc-400.php. [Último acceso: Septiembre 2016].

[22] Xilinx, Inc, «Vivado Design Suite,» web, [En línea].

Disponible en: http://www.xilinx.com/products/design-tools/vivado.html. [Último acceso: Septiembre 2016].

[23] Xilinx, Inc, «UG936 Vivado Design Suite Tutorial: Programming and Debugging,» v2015.3, Octubre 2015. [En línea].

Disponible en: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_3/ug936-vivado-tutorial-programming-debugging.pdf.

[Último acceso: Septiembre 2016].

[24] Xilinx, Inc, «Software Development Kit,» web, [En línea].

Disponible en: https://www.xilinx.com/products/design-tools/embedded-software/sdk.html. [Último acceso: Septiembre 2016].

[25] ARM, «DDI0406C ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition,» Mayo 2014. [En línea]. Requiere registro para acceder.

Disponible en: https://silver.arm.com/download/ARM_and_AMBA_Architecture/AR570-DA-70000-r0p0-00rel2/DDI0406C_C_arm_architecture_reference_manual.pdf. [Último acceso: Septiembre 2016].

[26] ARM, «DDI0401C CoreSight™ PTM-A9 Technical Reference Manual Rev. r1p0,» Julio 2011. [En línea].

Disponible en: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0401c/DDI0401C_coresight_ptm_a9_r1p0_trm.pdf. [Último acceso: Septiembre 2016].

[27] ARM, «IHI0035B CoreSight™ Program Flow Trace™ Architecture Specification,» Marzo 2011. [En línea].

Disponible en: http://infocenter.arm.com/help/topic/com.arm.doc.ihi0035b/IHI0035B_cs_pft_v1_1_architecture_spec.pdf. [Último acceso: Septiembre 2016].

Anexos

Anexo A: Planificación y presupuesto

En este capítulo se incluyen las descripciones de los medios materiales y temporales utilizados en el desarrollo del Trabajo Fin de Grado.

i. Descomposición en fases

El Trabajo Fin de Grado tiene una dotación de 12 créditos ECTS. Teniendo en cuenta que cada crédito ECTS equivale a entre 25 y 30 horas del trabajo del alumno, se extrae que la dedicación total ha de oscilar entre 300 y 360 horas.

En primer lugar se procede a distinguir dentro del proceso de realización del Trabajo Fin de Grado, las siguientes tareas, así como a su descripción y carga temporal asociada:

- Búsqueda de documentación. Recopilación de la documentación asociada con el trabajo, incluyendo su análisis para discernir su validez en relación a los objetivos del diseño y a las necesidades del trabajo. En total: 20 horas.
- Aprendizaje de la herramienta de diseño. Tiempo dedicado a la descarga, instalación y aprendizaje del uso de las distintas utilidades de las herramientas Xilinx Vivado y Xilinx SDK. Incluye el tiempo dedicado a la realización de tutoriales y otros ejercicios de asentamiento. En total 35 horas.
- Estudio del sistema. Se refiere al tiempo invertido en el estudio de los distintos manuales así como a las pruebas realizadas para comprobar progresivamente el funcionamiento en las distintas etapas del desarrollo. Es el tiempo dedicado a la labor investigadora. En total 135 horas.
- Configuración de la interfaz de traza. Aplicación de las conclusiones extraídas del estudio anterior al PS de la ZYNQ. Incluye la redacción del software de prueba, con la configuración de cada uno de los registros de los subsistemas de traza, así como el tiempo dedicado a las pruebas finales de operación. En total: 40 horas.
- Diseño del módulo observador. Especificación y descripción hardware del módulo observador, síntesis, implementación, simulaciones y pruebas de funcionamiento. En total: 40 horas.
- Elaboración de la memoria. Incluye la toma de notas durante el desarrollo del trabajo, así como la elaboración del documento final. En total: 60 horas
- Tutorías: Tiempo empleado en el intercambio de opiniones con la tutora, así como indicaciones, consejos, orientaciones y enseñanzas recibidas. En total: 30 horas.

El resultado final es de 360 horas de trabajo del alumno, más 30 horas de la tutora.

A continuación se muestra un resumen cronológico de las tareas descritas en el apartado anterior.

Tabla A-1 Diagrama de Gantt del desarrollo del proyecto

	TAREAS	MAYO				JUNIO				JULIO				SEPTIEMBRE			
		S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
1.	**Búsqueda de documentación**																
1. 1	Publicaciones sobre el estado de la técnica																
1. 2	Manuales de la placa ZYBO de DIGILENT																
1. 3	Manuales del SOPC ZYNQ de Xilinx																
2.	**Aprendizaje de la herramienta de diseño**																
2. 1	Tutoriales de Vivado																
2. 2	Tutoriales de SDK																
2. 3	Ejercicios de asentamiento																
3.	**Estudio del sistema**																
3. 1	Búsqueda de la interfaz de traza																
3. 2	Comprensión de la tecnología Coresight																
3. 3	Selección de los componentes necesarios																
4.	**Configuración de la interfaz de traza**																
4. 1	Análisis de los registros de configuración																
4. 2	Configuración del sistema																
4. 3	Pruebas y corrección de errores																
5.	**Diseño del módulo observador**																
5. 1	Módulo observador inicial																
5. 2	Adaptación al bus AXI																
5. 3	Pruebas y simulaciones																
6.	**Redacción de la memoria**																
6. 1	Apuntes sobre la evolución del trabajo																
6. 2	Compilación y redacción																
6. 3	Revisión																

Se debe advertir que la dedicación en horas no tiene por qué corresponderse al avance cronológico debido a una dedicación temporal que no es de tiempo completo.

ii. Presupuesto

Por último, se presenta el detalle de los gastos asociados a la realización del proyecto, mediante la siguiente tabla.

Tabla A-2 Presupuesto del proyecto

Código	Unidad	Descripción	Medición	Precio unitario	Precio total
1		**CAPÍTULO 1: MATERIALES**			
1 .01	ud.	**Placa ZYBO** Suministro de placa ZYBO marca DIGILENT. Incluye cable USB de programación y adaptador de corriente. Probado y funcionando.	1	189,00 €	189,00 €
1 .02	h.	**Ordenador portátil** Amortización de ordenador portátil HP 1000€ en 4 años, con 1700 horas laborables al año. Incluye adaptador de corriente. Probado y funcionando.	300	0,15 €	44,10 €
1 .03	ud.	**Licencia Xilinx Vivado WebPack** Suministro de licencia y archivo de instalación de Xilinx Vivado WebPack 2015.3. Incluye Xilinx SDK. Todo instalado y funcionando.	1	0,00 €	0,00 €
		SUBTOTAL CAPÍTULO 1: MATERIALES			**233,10 €**
2		**CAPÍTULO 2: HONORARIOS**			
2 .01	h	**Responsable de proyecto** Horas de trabajo del tutor	30	50,00 €	1.500,00 €
2 .02	h	**Ingeniero de proyecto** Horas de trabajo del alumno	300	20,00 €	6.000,00 €
		SUBTOTAL CAPÍTULO 2: HONORARIOS			**7.500,00 €**
		TOTAL TRABAJO FIN DE GRADO			**7.733,10 €**

A 26 de Septiembre de 2016
Manuel Peña Fernández