eman ta zabal zazu
UPV - EHU
Euskal Herriko Unibertsitatea
Universidad del País Vasco

Ingeniarien Goi Eskola
Escuela Superior de Ingenieros
Bilbao

Tesis Doctoral

Reconfiguración dinámica de sistemas modulares multi-procesador en dispositivos SoPC

Armando Astarloa
Ingeniero Industrial

Mayo de 2005

Director: Dr. Aitzol Zuloaga Izaguirre

A Mirentxu y a Mónica

Agradecimientos

Hace unos años, cuando afronté el reto de hacer esta tesis, me dijeron que era un trabajo personal. Si bien ahora puedo afirmar que, efectivamente, es una labor *básicamente* personal, también he de reconocer que, sin la ayuda y los apoyos recibidos, esta tesis se habría convertido en una empresa imposible. Debo agradecer:

A mi Director de tesis, Aitzol. Realmente, tu labor en este trabajo comenzó hace muchos años, cuando yo era alumno interno del Departamento de Electrónica y Telecomunicaciones. Entonces, gracias a tu pasión por la ingeniería y tu actitud de ayuda sin condiciones, sembraste un ingeniero entusiasta de su trabajo y dispuesto a llegar siempre, un poco más lejos. Gracias por la confianza que has puesto en mí.

A los compañeros del grupo de investigación. Tras pasar unos años en la empresa privada, cuando tuve la oportunidad de volver al Departamento, ya como profesor e investigador, no lo dudé. Sabía que me incorporaba a trabajar con excelentes profesionales, pero lo que es más importante para mí, con magníficas personas. Por ello, a mi vuelta, me encontré de nuevo con Aitzol. Con Jose Luis, que con tu dedicación y experiencia, siempre me has prestado toda la ayuda necesaria y más. Con Jaime, dispuesto a la reflexión y al debate. Y con Unai:

A Unai. Debo agradecerte especialmente la ayuda que me has prestado estos años. Las correcciones exhaustivas de esta tesis, los acertados análisis técnicos sobre mis propuestas y ensayos o la aportación de tu experiencia en las publicaciones; que aún siendo todos estos aspectos inestimables, no son más que un apéndice comparado con tu amistad. Tu visión crítica y humana de la vida y de la ingeniería, junto con tu capacidad de dar ánimos en los momentos más difíciles, me han ayudado y me siguen ayudando a afrontar los distintos retos.

A Txus y a Jagoba, mis dos compañeros de despacho en la actualidad. A vosotros debo agradeceros, además de una inestimable ayuda en aspectos técnicos e informáticos, ser unos excelentes compañeros del día a día, que es al fin y al cabo, lo que cuenta.

A todos los alumnos que me han ayudado en el laboratorio. En especial a Sergio, Mikel, Jaime y Eneko. Espero haberos inculcado la ilusión por la investigación y por la ingeniería.

A mis amigos, que sabéis apreciar las aventuras en las que me embarco (como esta tesis), en especial a Amaya, Eva, Iñigo, Javi, Julio, Mikel y Txaro.

A mi familia y en especial, a mi madre. Hemos avanzado estos años contra viento y marea, sin tu fortaleza y tu cariño, nada de esto hubiera sido posible.

A Mónica. Gracias por tu apoyo incondicional; por aceptar que no siempre elija el camino más fácil; por tus consejos; por tu familia; por decidir recorrer el camino conmigo; por tu alegría; en definitiva, por ser como eres.

Gracias a todos.

Abstract

Nowadays, the transistor density of electronic devices allows the integration of complete digital systems on a single integrated circuit. In order to reduce the development time and to face successfully these type of designs, they are made usually using cores. These cores, due to their complexity, often include processors inside. So, in these cases, they are called multi-processor systems.

This level of integration has also been spread to FPGA reconfigurable devices, which have been widely used because of their flexibility. In spite of that, the most common use of the reconfiguration capability is limited to the development stage of the design, to facilitate the debug process and, in some cases, to perform latter upgrades of the digital system.

However, the latest FPGAs allow to modify part of their configuration while the rest of the configured circuit remains running. This ability, called dynamic partial reconfiguration, has an additional interest in the designs that integrate digital systems on a single device. For these situations, the computations made into the chip can also set what context changes must be performed to the modules and apply them. These are the so called auto-reconfigurable systems.

Auto-reconfiguration is a complex operation. To perform it safely in a multi-processor core based system, the FPGA must both technologically admit the reconfiguration and contain a control system to manage the whole process.

This thesis proposes an auto-reconfiguration control system for multi-processor core based systems. First, a general proposal for the control system is presented. It is valid to be integrated into systems that use the most common standard specifications for System-on-a-Chip design. This generalization is described based on s generic multi-processor reconfigurable model, that is used to define the specifications of the different elements that build the control infrastructure. To help the designer to study the viability of the auto-reconfiguration in a design, the infrastructure area and time cost have been modelled and parameterized.

The proposed theoretic system is validated using a specific reconfigurable technology. For this purpose, all the elements specified in the control infrastructure are implemented, and some additional tools are developed to manage multi-processor and multi-context designs. The implemented infrastructure has been applied to three platforms that have been designed to experiment the auto-reconfiguration with the proposed approach. A custom prototype, that admits an exhaustive control of the FPGA configuration sequence, has been built to perform these experiments.

Laburpena

Gaur egun, gailu elektronikoetan onargarria den dentsitatea hain da handia, ezen zirkuitu integratu bakarrean sistema digital osoen integrazioa posiblea den, *System-on-Chip* edo SoC deiturikoak. Garapenaren denbora gutxitzeko eta mota honetako diseinuen arrakasta lortzeko, moduluz edo *core*-z osatuta daude. Modulu hauek, konplexuak izanik, prozesadore batean edo gehiagotan eragina izan ohi dute. Beraz, kasu hauetan, sistema multiprozesadoreak dira.

Integrazio-maila gailu birkonfiguragarrietara (FPGAk nagusiki) ere hedatu egin da. Aukera hau erabilienetariko bat da eskaintzen duen malgutasunagatik. Hala ere, birkonfigurazioraako gaitasuna prototipo faseetara mugatzen da diseinuaren arazketa erraztearren eta, batzuetan, beraren gaurkotzeak egiteko.

Dena den, FPGA berrienek beren konfigurazioaren zati bat moldatzea onartzen dute, zirkuituaren gainontzekoa exekutazen dabilen bitartean. Gaitasun honek, birkonfigurazio partzial dinamikoa deiturikoak, zirkuitu integratu bakarrean eratzen diren moduluz osatutako sistema digitalen diseinuan interes berezia du. Eta, kasu hauetan, txipean egindako prozesaketak moduluen programa edota zirkuituetarako kontestuaren aldaketak erabaki eta aplika ditzake. Mota honetako diseinuei sistema autobirkonfiguragarri izena eman zaie, hain zuzen ere.

Autobirkonfigurazioa egitea konplexua da. *Core*-z oinarritutako sistema multiprozesadore batean ondo egiteko bi gauza beharrezkoak dira: FPGAk birkonfigurazio partzial dinamikoa maila teknologikoan onartzea eta diseinua bera kontrolatzeko sitema bat izatea.

Tesi honek *core*-etan oinarritutako sistema multiprozesadore autobirkonfigurazioa kontrolatzeko sistema proposatzen du. Hasteko, kontrol-sistemaren planteamendu orokorra egiten da. Honek, txip batean sistema ohikoenen diseinua burutzeko, espezifikazio estandarrak erabiltzen dituzten diseinuetan txertatu ahal izateko balio behar du. Orokortasun hau prozesadore anizkuna den eredu birkonfiguragarrian gauzatzen da eta kontrolaren azpiegituran beharrezkoak diren elementuen ezaugarriak definitzen dira. Diseinu batean, autobirkonfiguarzioaren egokitasuna aztertzeko, azpiegiturak behar dituen baliabideen eredu matematikoa egiten da eta denbora tarteen parametroak neurtzen dira. Aurkeztutako sistema teorikoa teknologia birkonfiguragarri jakin bat erabiliz frogatzen da. Horretarako, birkonfigurazioaren kontrolak duen azpiegituran zehaztutako elementu guztiak burutzen dira eta, horrez gain, multikontestu eta multiprozesadorea duten diseinuak baimentzen dituzten tresna osagarriak sortzen dira.

Azpiegitura hau diseinatutako hiru plataformatan aplikatzen da. Hirurak autobirkonfigurazioa gauzatzeko proposaturiko kontrol sistema probatzeko diseinatuta daude. Saiakera hauek egiteko FPGA-ren konfigurazioa guztiz kontrolatzea onartzen duen prototipo berezia egin behar izan da.

Resumen

En la actualidad, la densidad de transistores admisible en los dispositivos electrónicos es tal que ya es posible la integración de sistemas digitales completos en un único circuito integrado, son los denominados *System-on-Chips* o SoCs. Con el fin de reducir el tiempo de desarrollo y poder afrontar con éxito este tipo de diseños, habitualmente se componen los mismos a base de módulos o *cores*. Estos módulos, dada su complejidad, incluyen frecuentemente uno o varios procesadores, por lo que, en estos casos, se dispone de sistemas multi-procesador.

Este nivel de integración también se ha extendido a los dispositivos reconfigurables FPGA, siendo esta alternativa una de las más utilizadas dada la flexibilidad que ofrece. Sin embargo, el uso más común de su capacidad de reconfiguración se limita a las fases de prototipado para facilitar la depuración de diseño y, en algunos casos, para realizar posteriores actualizaciones del mismo.

Sin embargo, las FPGAs más recientes admiten que se modifique parte de su configuración mientras el resto del circuito configurado sigue funcionando. Esta capacidad, denominada reconfiguración parcial dinámica, tiene particular interés en los diseños de sistemas digitales mediante módulos en un único circuito integrado. En estos casos, el procesamiento realizado en el *chip*, también puede determinar cambios del contexto para circuitos o programas de los módulos y aplicarlos. Concretamente, el término utilizado para denominar este tipo de diseños es el de sistemas auto-reconfigurables.

La operación de auto-reconfiguración es compleja. Para realizarla con garantías en un sistema multi-procesador basado en *cores*, además de que la FPGA admita la reconfiguración parcial dinámica a nivel tecnológico, es necesario un sistema de control en el propio diseño.

Esta tesis propone un sistema de control de la auto-reconfiguración para sistemas multi-procesador basados en *cores*. Inicialmente, se realiza un planteamiento generalizado del sistema de control, válido para ser incorporado a diseños que utilicen las especificaciones estándar para diseño de sistemas en un *chip* más habituales. Esta generalización se plasma en un modelo reconfigurable multi-procesador, definiéndose a partir del mismo las características de los elementos requeridos en la infraestructura de control. Para facilitar el análisis de la idoneidad de la auto-reconfiguración en un determinado diseño, se realiza un modelado de los recursos necesarios por la infraestructura y se parametrizan los tiempos involucrados en el proceso de reconfiguración.

El sistema teórico presentado se valida utilizando una tecnología reconfigurable concreta. Para ello, se implementan todos los elementos especificados en la infraestructura de control de la reconfiguración y se desarrollan herramientas adicionales que permiten diseños multi-procesador y multi-contexto. Esta infraestructura se aplica sobre tres plataformas diseñadas expresamente para experimentar la auto-reconfiguración con el sistema de control propuesto. Estos ensayos han requerido la construcción de un prototipo especial que admita un control total de los procesos de configuración de la FPGA.

Índice general

Abstract v

Laburpena vii

Resumen ix

Tabla de Acrónimos xxi

I. Introducción, estado del arte y objetivos 1

1. Introducción 3
1.1. Contexto de la tesis . . . 3
1.2. Introducción al tema de la tesis y objetivo general . . . 7
1.3. Estructura de la tesis . . . 8

2. Sistemas Reconfigurables 11
2.1. Introducción . . . 11
2.2. Plataformas reconfigurables experimentales . . . 17
2.3. Dispositivos comerciales reconfigurables dinámicamente . . . 24
2.4. Resumen de los Sistemas Reconfigurables . . . 28

3. Sistemas Auto-Reconfigurables basados en cores sobre FPGAs 31
3.1. Definición formal de auto-reconfiguración . . . 31
3.2. Reconfiguración parcial dinámica aplicada a circuitos digitales . . . 34
3.3. Simulación de la reconfiguración parcial dinámica de circuitos . . . 38
3.4. Reconfiguración parcial dinámica de IP-Cores . . . 41
3.5. Reconfiguración parcial dinámica aplicada a plataformas FPGA basadas en IP-Cores . . . 45

4. Cores Mixtos 63
4.1. Integración de pequeños procesadores en cores . . . 63
4.2. Campo de aplicación de los Cores Mixtos . . . 65

5. Análisis de las alternativas para el control de la auto-reconfiguración parcial dinámica 69

6. Objetivos **73**
6.1. Objetivo general 74
6.2. Objetivos parciales 74

II. Sistema Tornado **77**

7. Introducción y justificación **79**

8. Modelo reconfigurable multi-procesador **83**

9. Infraestructura de control de reconfiguración **89**
9.1. Protocolos de control 89
9.2. Controlador de reconfiguración: TBC 97
9.3. Unidad procesadora Tornado nano-Processor: TnP 99

10. Modelado de Tornado **103**
10.1. Caracterización temporal 103
10.2. Modelado de recursos lógicos 107

11. Reconfiguración intra-task **113**
11.1. TnP-Cores 113
11.2. Flujo de diseño para reconfiguración intra-task 116

12. Reconfiguración inter-task **121**
12.1. Tornado Bus-Macro 121
12.2. Flujo de diseño para reconfiguración inter-task 123

III. Validación y Evaluación del sistema **125**

13. Plan de pruebas y tecnología **127**
13.1. Plan de pruebas 127
13.2. Reconfiguración parcial dinámica en dispositivos Virtex 127

14. Herramientas hardware **133**
14.1. Especificación estándar para interconexión de IP-Cores Wishbone 133
14.2. Unidad procesadora TnP para Virtex y Wishbone 138
14.3. Controlador de reconfiguración TBC para Virtex y Wishbone 143
14.4. Módulos parametrizables auxiliares 148

15. Herramientas software **149**
15.1. Sistema de desarrollo ISE 149
15.2. Ensamblador para procesadores TnP 150
15.3. Scripts de automatización 153

16. Plataforma de verificación básica: Video-Multimaster **155**
16.1. Descripción general . . . 155
16.2. Diagrama de bloques . . . 155
16.3. Módulos IP . . . 156
16.4. Resultados experimentales . . . 159

17. Plataforma de verificación avanzada I: AMT **169**
17.1. Descripción general . . . 169
17.2. Diagrama de bloques . . . 169
17.3. Módulos IP . . . 170
17.4. Resultados experimentales . . . 175

18. Plataforma de verificación avanzada II: IP-Cores Bajo Demanda **183**
18.1. Descripción general . . . 183
18.2. Diagrama de bloques . . . 184
18.3. Tornado Bus-Macro . . . 186
18.4. Módulos IP dinámicos . . . 191
18.5. Módulos IP estáticos . . . 193
18.6. Resultados experimentales . . . 195

IV. Conclusiones y trabajo futuro **197**

19. Conclusiones **199**
19.1. Principales aportaciones . . . 200
19.2. Publicaciones generadas a partir de esta Tesis . . . 204

20. Trabajo futuro **207**

Índice de figuras

1.1. Diseño de SoC basado en bloques. 5

2.1. Sistemas de computación según el grado de flexibilidad. 11
2.2. Modelo genérico de una FPGA. 13
2.3. Modelos de configuración. 15
2.4. Modelos de reconfiguración dinámica. 16
2.5. Plataforma DECPeRLe-1 18
2.6. Diagrama de bloques del sistema Garp. 19
2.7. Arquitectura simplificada del sistema Chimaera. 20
2.8. Procesador *Dynamic Instruction Set Computer* (DISC). 21
2.9. Maia CSoC. Procesador reconfigurable heterogéneo. 22
2.10. *Dynamically Reconfigurable Architecture for Mobile communication systems* (DReAM) CSoC. 23
2.11. CSoC académico/comercial compuesto por un *core* Leon y *cores* PAC XPP™ . 24
2.12. *Configurable Logic Block* de Virtex (2 *Slices*). 25
2.13. Estructura de la celda de los dispositivos de la serie AT6000 de Atmel. 28

3.1. Gestor de reconfiguración genérico de Shirazi et al. 33
3.2. Modelo de componente reconfigurable basado en una red estática. 33
3.3. Circuito digital para la aplicación de *offset* y control de ganancia a una señal digital [1]. 35
3.4. Entorno JAVA para soporte de la reconfiguración dinámica. 36
3.5. Dispositivos virtuales integrados en el diseño para permitir la simulación de lógica reconfigurable dinámicamente. Técnica *Dynamic Circuit Switching (DCS)*. 41
3.6. Modelo de los *Programmable Multi-function Cores* (PMCs). 42
3.7. Aplicación del flujo de diseño mixto de Dyer a *cores* reconfigurables dinámicamente. 44
3.8. Antenas Gasket para la interconexión de DHPs a la zona estática del sistema. 45
3.9. Arquitectura de la plataforma *Flexible* URISC. 46
3.10. Sistema auto-reconfigurable para la comparación de patrones de datos. 47
3.11. Tipos de buses en la especificación CoreConnect™ de IBM. 50
3.12. Tipos de buses en la especificación AMBA de ARM. 51
3.13. Conexión de *Cores* mediante un único bus Wishbone. 52
3.14. Topologías de interconexión de bus soportadas por especificación Wishbone. . 53
3.15. Módulos hardware de la plataforma SRP. 54
3.16. Niveles software para la plataforma SRP. 55

3.17. Arquitectura del *framework* para auto-reconfiguración integrado en una FPGA propuesto por Fong et al. 56
3.18. Arquitectura *Multi-controller*. 57
3.19. Implementación mediante reconfiguración parcial dinámica de la arquitectura *Multi-controller*. 57
3.20. Bloques principales de modelo de reconfiguración parcial dinámica con gestión de prioridades. 58
3.21. *Bus-Macro* para la NoC propuesta en el modelo de Ullmann et al. 59
3.22. Infraestructura general del sistema R8NR (modelo FiPRe). 60
3.23. Entorno *Run-Time* para Sistema Operativo para hardware reconfigurable de H. Walder y M. Platzner. 61

4.1. Arquitectura de *core* con `nano-procesador` embebido (`Core Mixto`). 65
4.2. *Core* `FFR16`. 66

8.1. Arquitectura generalizada de un SoPC basado en *cores*. 83
8.2. Arquitectura generalizada de un *Configurable*-SoPC compatible `Tornado`. . . 84
8.3. Modelo `Tornado`: arquitectura generalizada de un CSoPC, compatible `Tornado`, con la posibilidad de integrar `Cores Mixtos`. 87

9.1. Flujo de petición de reconfiguraciones. Diferentes fuentes de solicitud de reconfiguraciones para los `T-Cores`. 90
9.2. Palabra de solicitud de reconfiguración `CRW`. 91
9.3. Interfaces `Tornado`. 93
9.4. Interfaces `Tornado` incluidas en los `T-Cores` y en el controlador de reconfiguración `TBC` 96
9.5. Secuencia del *handshake* para la aplicación de la reconfiguración. 96
9.6. Arquitectura del controlador básico de reconfiguración definido en `Tornado`. . 98
9.7. Interfaz `RM IF`. 100
9.8. Interfaces `Tornado` de los `TnPs`. 101

10.1. Diagrama de tiempos de la escritura y aplicación de una palabra de petición de reconfiguración `CRW`. 104
10.2. Pasos de la secuencia del *handshake* de configuración. 105
10.3. Parámetros temporales de la secuencia del *handshake* de configuración. 106

11.1. Modificación de los `Cores Mixtos`. 114
11.2. Diagrama de flujo de diseño con reconfiguración *intra-task*. 117

12.1. Máquina de estados implementada en la `Tornado Bus-Macro`. 122
12.2. Diagrama de flujo de diseño con reconfiguración *inter-task*. 123

13.1. Distribución de los recursos lógicos en el dispositivo más pequeño de la familia Spartan-II. 128
13.2. Distribución de los diferentes tipos de *frames* en un dispositivo Virtex. 129
13.3. Distribución de las *frame columns* en la memoria de configuración. 130
13.4. Distribución de los *frames* en los *frame columns*. 130

13.5. Conexión entre módulos reconfigurables para aplicar reconfiguración *inter-task* en dispositivos Virtex de Xilinx 131

14.1. Conexión de *Cores* mediante un único bus Wishbone. 135
14.2. Ciclo de lectura simple en Wishbone. 136
14.3. Unidad procesadora *Picoblaze*. 139
14.4. Arquitectura del `TBC` para dispositivos Virtex y compatible Wishbone. 143
14.5. Prototipo desarrollado para la experimentación con reconfiguración parcial dinámica. 145
14.6. Subsistema de carga de *bitstreams* parciales y totales integrado en el prototipo. 146

15.1. Flujo de diseño ISE. 149
15.2. Diagrama de flujo de ficheros del ensamblador `TRA` para un `TnP`. 152
15.3. Generación automática de los ficheros VHDL para la inicialización de los bloques de RAM y del controlador de reconfiguración. 154

16.1. Diagrama de bloques de la plataforma de verificación *Video-Multimaster*. . . 156
16.2. `TnP-Master Video` *core*. 157
16.3. Sistema de visualización de la actividad en los `TnP-Master Video` mediante un monitor seccionado. 158
16.4. `WB-VGA` *core*. 159
16.5. Evolución del error en las estimaciones de recursos lógicos en función de la complejidad de la plataforma. 163
16.6. Parámetros temporales normalizados. 164
16.7. Carga de un *bistream* parcial controlada por el `TBC`. 165
16.8. Distribución de los BRAM *frames* en un dispositivo Virtex con dos columnas de BRAMs. 166

17.1. Plataforma `AMT` desarrollada para la evaluación del sistema `Tornado`. 170
17.2. Partición interna del `TnP-Cores DDS-TXi`. 171
17.3. Comparativa de la utilización de recursos lógicos entre plataformas `AMT` de un único canal con `Tornado` y sin `Tornado`. 177
17.4. Evolución de la frecuencia máxima de funcionamiento para el conjunto de plataformas `AMT` de un canal. 178
17.5. Comparativa de la utilización de recursos lógicos entre plataformas `AMT` MCSoPC y MSoPC de distinto número de canales y tres tipos de modulación. 179
17.6. Plataformas `AMT` MCSoPC y MSoPC con distintos número de canales y tres tipos de modulaciones. 181

18.1. Plataforma experimental `IP-Cores bajo demanda`. 185
18.2. `TBM` para Spartan-II y Wishbone. 187
18.3. Lógica de control integrada en la `TBM` 189
18.4. Circuito interno de la `TBM` 190
18.5. Arquitectura del `WB_ADC_dynamic_core`. 191
18.6. Módulo dinámico `WB_ADC_dynamic_core` emplazado y rutado. 192
18.7. Módulo dinámico `WB_mult_dynamic_core` emplazado y rutado. 193
18.8. Módulos estáticos agrupados, emplazados y rutados. 194

18.9. Arquitectura del `TnP-UART`. 194

Índice de tablas

0.1. Tabla de Acrónimos xxi

2.1. Resumen de los Sistemas Reconfigurables presentados. 29

3.1. Especificaciones para interconexión de *cores* y sus licencias de utilización. . . 49

4.1. Coste de implementación de un sistema de computación ATA+FAT para distintas alternativas SoPC. 67

5.1. Comparación entre las alternativas actuales para la aplicación de Reconfiguración Parcial Dinámica a plataformas FPGA comerciales basadas en IP-Cores. 71

7.1. Términos utilizados en el sistema `Tornado`. 82

8.1. Características generales de los controladores de reconfiguración de las diferentes propuestas. 85

13.1. Plan de Pruebas. 128

14.1. Juego de instrucciones del `nano-procesador` *soft* `TnP`. 140
14.2. Resultados de implementación de un `nano-procesador TnP`. 142
14.3. Parámetros de modelado del `TnP`. 143
14.4. Resultados de implementación del controlador de reconfiguración `TBC`. . . . 147
14.5. Parámetros de modelado del `TBC`. 147

16.1. Resultados de implementación del `core TnP-Master Video` en un dispositivo XC2S150. 158
16.2. Estimaciones del coste en recursos área de `Tornado` para distintas variantes de la plataforma `Video-Multimaster`. 160
16.3. Resultados de implementar las variantes de la plataforma `Video-Multimaster` sin infraestructura `Tornado`. 161
16.4. Resultados de implementar las variantes de la plataforma `Video-Multimaster` con la infraestructura de `Tornado`. 161
16.5. Recursos lógicos reales y estimados de la infraestructura `Tornado`. 162
16.6. Valores normalizados de los parámetros temporales para la plataforma `Video Multi-master` 164
16.7. Tiempos no normalizados para las plataformas `Video-Multimaster`. 165

17.1. Resultados de implementación del `TnP-Core DDS-TXi` sobre una FPGA Spartan-II X2S150-6 172
17.2. Resultados de implementación del `TnP-Core RSSI-LD` en una FPGA Spartan-II X2S150 173
17.3. Valores normalizados de los parámetros temporales para la plataforma `AMT` . 178

18.1. Valores normalizados de los parámetros temporales para la plataforma `IP-Cores bajo demanda` 196

Tabla de Acrónimos

Tabla 0.1.: Tabla de Acrónimos

Acrónimo	Término
ADC	Analog to Digital Converter
AHB	Advanced High speed Bus
ALU	Arithmetic Logic Unit
AMT	Adaptive Multi-Transmitter
APB	Advanced Peripheral Bus
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
ASP	Advanced general purpose System Bus
ASSP	Application-Specific Standar Product
BRAM	BlockRAM
CCCU	Configuration Control Cache Unit
CLB	Configurable Logic Block
CM	Core Mixto
CMOS	Complementary Metal-Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRW	Configuration Request Word
CSoC	Configurable-Systems-on-Chips
CSoPC	Configurable-System-On-Programable-Chip
DAC	Digital to Analog Converter
DCM	Digital Clock Management
DCR	Device Control Register
DCS	Dynamic Circuit Switching
DDS	Direct Digital Synthesis
DHP	Dynamic Hardware Plugin
DISC	Dynamic Instruction Set Computer
DLL	Delay-Locked Loop
DReAM	Dynamically Reconfigurable Architecture for Mobile communication systems
DRIP	Dynamically Reconfigurable IP
ECU	Execution Control Unit
FFR16	First File Reader FAT16

FIFO	First In - First Out
FIP	Flexible Instruction Processor
FPGA	Field Programmable Gate Array
FPSLIC	Field Programmable System Level Integrated Circuit
FPU	FAT Processor Unit
FPX	Field Programmable Port eXtender
FSK	Frequency Shift Keying
GF	Galois Field
GPL	General Public License
HAU	Host ATAPI Unit
HDL	Hardware Description Languaje
IDE	Integrated Device Electronic
IOB	Input-Output-Block
IP-Core	Intellectual Proprietary Core
ISA	Industry Standard Architecture
LB	Logic Block
LHM	Linear Hardware Model
LUT	Look-Up Table
MAC	Media Access Controller
MCSoPC	Multiprocessor-Configurable-System-On-Programable-Chip
MIPS	Million Instructions Per Second
MPEG-2	Motion Picture Expert Group -2
NoC	Network-on-Chip
OCIDEC	Open Core IDE Controller
OOK	On-Off Keying
OPB	On-chip Peripheral Bus
OSCI	Open SystemC Initiative
PA	Processing Array
PAE	Processing Array Element
PAL	Programmable Array Logic
PCI	Peripheral Component Interconnect
PCR	Program Counter Reset
PIO	Peripheral Input-Output
PLB	Processor Local Bus
PMC	Programmable Multifunction Core
PSK	Phase Shift Keying
RA	Reconfigurable Array
RAM	Random Access Memory
RCD	Reconfiguration Condition Detector
RFU	Reconfigurable Functional Unit
RISC	Reduced Instruction Set Computer

RM IF	Reconfiguration Media InterFace
ROM	Read Only Memory
RPU	Reconfigurable Processing Unit
RSSI	Radio Signal Strength Indicator
RTL	Register-Transfer-Level
RTP	Run-Time Parametrizable
RTR	Run-Time-Reconfiguration
SCM	Schedule Control Modules
SDR	Software Defined Radio
SoC	System-on-Chip
SoPC	System-on-Programmable-Chip
SPL	Swappable Logic Units
SRP	Self-Reconfiguring Platform
SPR	Stack Pointer Reset
SRAM	Static Random Access Memory
SRGA	Self-Reconfigurable Gate Array Architecture
SRI	System Reconfiguration Information
TBC	Tornado Basic Controller
TCB	Task Communication Bus
T-Core	Tornado compatible Core
TnP	Tornado nano-Processor
TnP-Core	Tornado nano-Processor Core
TBM	Tornado Bus-Macro
TIF (M)	Tornado Interface Master
TIF (S)	Tornado Interface Slave
TRA	Tornado Reconfigware Assembler
TSR	Task Status Registers
UAL	Unidad Aritmético Lógica
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Languaje
XPART	Xilinx PArtial Reconfiguration Toolkit

Parte I.

Introducción, estado del arte y objetivos

1. Introducción

1.1. Contexto de la tesis

El trabajo de investigación que ha dado como resultado esta tesis doctoral se ha desarrollado en el *Grupo de Investigación en Electrónica Aplicada* del Departamento de Electrónica y Telecomunicaciones de la Universidad del País Vasco/Euskal Herriko Unibertsitatea.

Este grupo está compuesto en la actualidad por once investigadores, centrando sus actividades de investigación en las siguientes líneas:

- *Circuitos de control y potencia para convertidores de energía*: Diseño y desarrollo de los sistemas electrónicos necesarios para el control de equipos relacionados con las energías renovables, fundamentalmente aerogeneradores, en el proceso de generación, transformación y almacenamiento de energía.

- *Circuitos digitales de comunicaciones:* Investigación y desarrollo de los sistemas de comunicaciones desde un punto de vista electrónico, tanto de sistemas cableados como inalámbricos. Dentro de ella, se distinguen dos sublíneas principales. Por un lado, el diseño de circuitos digitales necesarios para la comunicación entre diversos dispositivos conectados al mismo bus, así como el modelado y la simulación de la red de comunicaciones utilizada. Por otro lado, el estudio de redes de sensores inalámbricas compuestas por elementos de bajo coste con capacidades de comunicación, de manera que, conjuntamente, son capaces de realizar una tarea de manera distribuida.

- *Circuitos digitales para procesamiento de imágenes:* Diseño de nuevas arquitecturas de procesamiento de imágenes en tiempo real utilizando dispositivos lógicos programables, normalmente como paso previo para procesamientos posteriores.

- *Circuitos reconfigurables y sistemas en un único chip (System-on-Chip -SoC-):* Utilización de dispositivos lógicos programables de nueva generación y de alta capacidad, tipo *Field Programmable Gate Arrays* (FPGAs), para integrar un sistema digital en un único circuito y hacer uso de la capacidad de reconfiguración de estos dispositivos. Se está trabajando en diseño orientado a síntesis, diseño orientado a verificación, arquitecturas de interconexión de módulos y circuitos de alta frecuencia.

El trabajo de investigación presentado en esta tesis, se enmarca en esta última línea de investigación, *Circuitos reconfigurables y sistemas en un único chip.* En los diseños electrónicos realizados por este grupo de investigación a lo largo de su historia los circuitos reconfigurables han estado siempre presentes.

Básicamente durante los años 80 y principio de los noventa, los circuitos reconfigurables más utilizados fueron PALs y CPLDs. Éstos son dispositivos reconfigurables de baja capacidad. La función que debe realizar el circuito se diseña y modifica en el ordenador

del diseñador. La configuración resultante se descarga en el propio laboratorio con unos medios sencillos. Este mecanismo supuso una clara ventaja para el diseño de prototipos y pequeñas series respecto a la utilización de componentes discretos (circuitos integrados digitales estándar).

La utilidad de estos dispositivos de baja capacidad no se centró únicamente en el prototipado, sino que también se extendió a diseños industriales, básicamente a aplicaciones de *glue logic*[1]. La utilización de éstos permitía la agrupación de componentes discretos con la consecuente reducción de área. Además, dotaban de una cierta flexibilidad para corrección de errores y actualización del sistema sin necesidad de cambio en el circuito impreso, compensando de esta forma el sobrecoste de estos dispositivos respecto a la utilización de circuitos integrados discretos.

A principio de los años 90 hacen su entrada en el mercado de los dispositivos reconfigurables las FPGAs con tecnología SRAM. Éstas disponían de una capacidad lógica mayor que las CPLDs y PALs. En los prototipos del grupo ya se utilizaban FPGAs encargadas de realizar procesamientos más complejos que un simple *glue logic*, básicamente, pre-procesamiento para comunicaciones digitales y algoritmos para visión artificial.

Sin embargo, la evolución de las FPGAs no acaba ahí. Al igual que ocurre con los circuitos de aplicación específica (*Application Specific Integrated Circuit* -ASIC-), la capacidad de integración en los mismos se incrementa de forma sorprendente año a año. La conocida Ley de Moore [2] que preveía que el número de transistores integrables en una misma área de silicio se duplicaría cada 18 meses se está cumpliendo.

Esta tendencia, que ha permitido que en la actualidad se integren sistemas digitales completos en ASICs surgiendo los *System-on-Chips* (SoCs), también afecta a los dispositivos reconfigurables, en concreto a las FPGAs. En la actualidad se dispone de FPGAs con capacidades de integración de millones de *puertas lógicas equivalentes* [3]. Esta situación permite, al igual que con los ASICs, integrar un sistema digitale completo en un único dispositivo FPGA, definiéndose este caso como *System-on-Programmable-Chip* (SoPC).

Frente a los SoCs basados en tecnología ASIC, los SoPCs tienen una desventaja en cuanto a precio y a eficiencia puesto que no se tratan de circuitos diseñados específicamente para una determinada aplicación. Sin embargo, en muchos casos compensa esta situación con su flexibilidad y accesibilidad para los grupos de investigación y para las pequeñas y medianas empresas.

La necesidad de gestionar la complejidad generada por este nivel de integración en los SoCs y SoPCs ha incorporado nuevas metodologías basadas en el diseño a partir de módulos hardware y módulos software. Sobre la base de este concepto, se define un SoC de la siguiente forma: "Un sistema *System-on-a-Chip* (SoC) está compuesto por un conjunto de módulos y subsistemas (hardware y software) que se interconectan de forma apropiada para cumplir las funciones requeridas en una determinada aplicación" [4]. R. Rajsuman también define el concepto de SoC como "un circuito integrado, diseñado mediante interconexión de múltiples bloques VLSI independientes, que ofrece toda la funcionalidad de una aplicación" [5]. Esta definición hace hincapié en los bloques previamente diseñados que agrupan funcionalidades complejas. J. Borel define el mismo término como "la agrupación en un único *chip* de las capacidades de procesamiento, control, almacenamiento, comunicación, sensóri-

[1] Con el término de *glue logic* se denomina al conjunto de circuitos auxiliares necesarios en un sistema digital para adaptarlo a una configuración concreta: decodificación y banqueo de memoria, interrupciones, etc.).

ca y de actuación" [6]. Los dispositivos actuales no permiten la integración de todas estas capacidades, pero sí de una gran parte de ellas.

El diseño de un SoC se basa en la reutilización de componentes prediseñados, denominados IP-Cores, *cores* o módulos, reutilizando incluso arquitecturas predefinidas. Aún así, la complejidad del diseño de SoCs es mayor respecto a la del diseño convencional de sistemas distribuidos en circuitos impresos y en distintos circuitos integrados, pero aporta numerosas mejoras como son [4]:

- Reducción del coste del producto.
- Incremento del rendimiento del sistema. Al integrar los buses en un dispositivos se disminuyen las líneas de interconexión existentes en el circuito impreso y además se reduce considerablemente la longitud de las mismas lo que permite velocidades mucho mayores de comunicación.
- Descenso significativo del consumo.
- Reducción global de la superficie de silicio.

El diseño de un sistema SoC implicar integrar en un único dispositivo un sistema electrónico completo mediante diversos bloques complejos (figura 1.1). Estos bloques son tanto bloques hardware (microprocesadores, procesadores digitales de señal, buses *on-chip*, periféricos, etc.) como bloques software. Los elementos software pueden abarcar desde sistemas operativos en tiempo real a programas específicos para la aplicación.

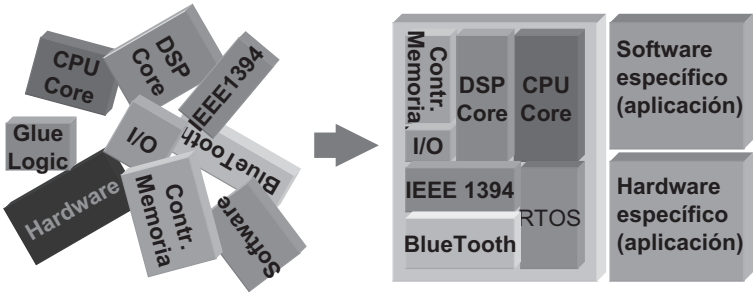


Figura 1.1.: Diseño de SoC basado en bloques.

En cuanto a los *cores* hardware, se diferencian tres tipos en función de su flexibilidad para la implementación:

- **Cores *soft*:** Estos módulos están descritos en lenguajes de descripción hardware (VHDL o Verilog) a nivel de transferencia entre registros (*Register-Transfer-Level* - RTL-). El código representa entidades hardware sintetizables.

  Este tipo de *cores* son flexibles y relativamente independientes de la tecnología, pudiéndose sintetizar para distintos dispositivos, incluso para ASICs o lógica programable.

- **Cores *firm*:** La descripción de los mismos se realiza mediante información a nivel de puertas y biestables (*netlist*) optimizada en tamaño y rendimiento para un dispositivo o familias de dispositivos en concreto. Por tanto la asignación de recursos del dispositivo para realizar esas tareas ya está realizada antes de su utilización por parte del diseñador, restando únicamente la fase de interconexión que se realizará en la implementación.

  Son menos flexibles pero permiten la protección de la Propiedad Intelectual y son más eficientes en general.

- **Cores *hard*:** En el caso del diseño de ASICs este tipo de bloques son *cores* preparados para ser integrados en el silicio del dispositivo.

  Disponen además de la información de los recursos necesarios a bajo nivel que hay que implementar en el silicio, de la información del rutado exacto. Están optimizados en aspectos tales como el rendimiento, área o consumo.

  En el caso de los *cores hard* para dispositivos lógicos programables, son *cores* ya integrados en el silicio del dispositivo.

Los *cores soft* son los que admiten mayor grado de flexibilidad mientras que los *cores hard* son los que resuelven las necesidades más complejas (procesadores avanzados, multiplicadores de alto rendimiento, etc.) y se encuentran mejor optimizados.

El diseño de un SoC agrupa y mezcla las disciplinas del diseño hardware, diseño software y telecomunicaciones. Este nuevo concepto de diseño crea un un entorno convergente que abre unas dimensiones innovadoras de productos, mercados, competidores, tecnologías y servicios [7]. Los participantes en este entorno son los fabricantes de IP-Cores y de circuitos integrados; los diseñadores y verificadores de sistemas; los desarrolladores de sistemas operativos y de aplicaciones; así como las empresas dedicadas a las herramientas de desarrollo que permitan esta evolución.

Para que estos *cores* sean realmente reutilizables, deben presentar interfaces comunes que permitan una rápida y sencilla integración. Con este propósito se están estableciendo consorcios e iniciativas. Un ejemplo significativo es el del consorcio *Virtual Socket Interface Aliance (VSIA)* [8] cuyo objetivo es el de normalizar el desarrollo de *cores*. Esta evolución en la forma de diseñar los sistemas electrónicos ofrece numerosos grados de libertad al diseñador permitiendo no sólo la elección de los *cores*, sino también la selección de la topología de bus y de la arquitectura del sistema.

En la práctica no son manejables tantos grados de libertad. Con el fin de guiar al diseñador en la tarea de seleccionar una arquitectura para su sistema se ha evolucionado hacia un modo de diseño con *cores* más restringido denominado *diseño basado en plataformas* [9].

En este caso, el fabricante de los dispositivos propone al diseñador el uso de una especificación concreta para la interconexión de los *cores* y unas determinadas topologías de bus. Generalmente, estas propuestas están condicionadas por el microprocesador que integre el fabricante en los dispositivos. Los fabricantes de lógica programable proponen sus plataformas para desarrollar sistemas integrados en un *chip* empleando sus FPGAs de alta capacidad.

Estas plataformas SoPC están resultando un rotundo éxito dentro del área de los SoCs [9]. En la actualidad, sólo un pequeño grupo de grandes multinacionales se aventura a diseñar

SoC sobre ASICs. En contraste, con los SoPCs basados en plataformas han puesto al alcance de toda la comunidad diseñadora de sistemas electrónicos la posibilidad de realizar diseños integrando sus desarrollos específicos y reutilizando los ya realizados por otras fuentes en un único dispositivo.

Dada la rápida evolución de estos sistemas y la revolución producida en las metodologías de diseño, se ha considerado necesario en nuestro grupo de investigación definir la línea de investigación en que se enmarca esta tesis (*Circuitos reconfigurables y sistemas en un único chip*). Mediante esta línea de investigación, además de mantener un seguimiento de los avances en esta tecnología, se realizan investigaciones en aspectos innovadores de estos sistemas, sirviendo de ejemplo el trabajo de investigación presentado en esta tesis.

1.2. Introducción al tema de la tesis y objetivo general

Habitualmente cuando se utiliza el término de Computación Reconfigurable se tiende a pensar en sistemas basados en FPGAs. Realmente es necesario especificar con mayor detalle qué sistemas se pueden caracterizar con ese término.

Cuando se realiza un prototipo utilizando un dispositivo reconfigurable por sus facilidades para la depuración, y finalmente en la fase de producción se sustituye por un ASIC, no es adecuado hablar de Computación Reconfigurable.

En el caso de que un diseño incluya una FPGA tal que se puedan realizar actualizaciones del diseño o corregir errores detectados una vez que la placa esté en el mercado, se está aprovechando una característica intrínseca de los dispositivos reconfigurables y, el término que describiría adecuadamente esta utilización sería el de Computación Configurable [10].

La Computación Reconfigurable implica un paso más en el aprovechamiento de las propias características de los dispositivos reconfigurables. En concreto, la manipulación de la lógica configurada en el dispositivo durante el funcionamiento del sistema. Tal y como se presenta en el estado del arte de esta tesis, hay numerosas variantes para poder realizar esa operación. En todas éstas, el diseñador dispone de más hardware para ser configurado en el dispositivo que el que realmente éste admite simultáneamente.

Aunque el término de Computación Reconfigurable se acuñó a finales de los 60 por investigadores de la Universidad de California, realmente la investigación en esta área se ha incrementado notablemente a finales de los años noventa y en la presente década. El nivel de integración de los dispositivos reconfigurables actuales más la posibilidad de modificar parcial y dinámicamente su configuración, característica disponible en muchos de ellos, han acelerado las investigaciones en este campo tanto a nivel teórico como en posibles aplicaciones.

Se ha concluido el apartado anterior destacando cómo los diseños SoPC basados en plataformas son en la actualidad una alternativa extensamente utilizada para el diseño de sistemas embebidos en un único dispositivo. De nuevo, en estos sistemas, se podrá hablar de Computación Reconfigurable cuando realmente se esté haciendo uso de la capacidad de reconfiguración dinámica. Esto supone un nuevo reto para la investigación en esta área: *El poder controlar la aplicación de la reconfiguración parcial dinámica sobre un sistema compuesto por cores de distinta naturaleza teniendo en cuenta que deben mantener su funcionamiento mientras se modifica una única sección.*

A pesar de que estas plataformas son muy recientes, como se pondrá de manifiesto en el

estado del arte, ya hay diversos grupos de investigación que están proponiendo soluciones para controlar la reconfiguración parcial dinámica sobre este tipo de sistemas.

El autor, con el trabajo de investigación descrito en esta tesis, quiere dar un paso más en este campo. La heterogeneidad y complejidad de los *cores* actuales implican la integración de secciones de control dentro de éstos. La forma más flexible de resolver esas operaciones de control es mediante la utilización de microprocesadores. De hecho, muchos *cores* integran pequeños microprocesadores para trabajar de forma conjunta con secciones dedicadas de hardware para procesamientos más intensivos (también parte de ese mismo *core*). Por tanto, en un SoPC puede ser habitual disponer de diversos procesadores distribuidos en los diferentes *cores* del sistema. El autor ha utilizado en esta tesis el término `nano-procesadores` para referirse a los procesadores de pequeño tamaño utilizados para este propósito, debido a sus peculiares características detalladas en posteriores secciones.

El objetivo general de esta tesis es proponer un sistema de control de reconfiguración parcial dinámica sobre sistemas SoPC basados en *cores* que permita una modificación segura y eficiente de los mismos. Esta reconfiguración incluye tanto secciones hardware como secciones software para el caso en el que los *cores* dispongan de procesadores embebidos.

1.3. Estructura de la tesis

Esta tesis se ha dividido en cuatro partes y cada una de ella en seis capítulos, salvo la cuarta con dos capítulos.

En la parte I se presenta el estado del arte sobre los diferentes sistemas para controlar la reconfiguración parcial dinámica en sistemas basados en *cores* sobre FPGAs de grano fino. Se analizan sus características, identificando las carencias existentes para poder soportar diseños multi-procesador dinámicamente reconfigurables. A partir de este análisis se establecen los objetivos del sistema de control de la reconfiguración parcial dinámica presentado en esta tesis.

Previamente a la presentación de los sistemas de control de la reconfiguración parcial dinámica de forma concreta para ese tipo de sistemas, se completa el estado del arte con cuestiones relacionadas con la Computación Reconfigurable consideradas imprescindibles para ubicar adecuadamente la propuesta realizada en esta tesis. Estas cuestiones incluyen una presentación de los dispositivos reconfigurables más representativos en el capítulo 2.

Concretando el área relacionada con el trabajo de esta tesis, en el capítulo 3 se detallan los aspectos relacionados con la auto-reconfiguración parcial dinámica sobre FPGAs de grano fino. Inicialmente se definen conceptos teóricos relacionados con la reconfiguración parcial dinámica resumiéndose a continuación las cuestiones relativas a la reconfiguración parcial dinámica aplicada a circuitos digitales. Este capítulo finaliza presentando las investigaciones realizadas estos últimos años sobre la aplicación de la reconfiguración parcial dinámica a *cores* y a plataformas basadas en *cores* con lo que se concreta específicamente el campo de aplicación de la propuesta realizada.

El siguiente capítulo, el número 4, presenta una arquitectura denominada `Core Mixto` para el diseño de *cores* que integra procesamiento software y procesamiento hardware. Mediante los `Cores Mixtos` se componen las plataformas SoPC multi-procesador que se desean ampliar a sistemas configurables dinámicamente.

Para identificar soluciones que permitan esa ampliación, en el capítulo 5, se contrastan las

alternativas para el control de la reconfiguración parcial dinámica sobre sistemas basados en *cores* en FPGAs de grano fino presentadas en el capítulo 3.

Un elemento clave en esta comparación es la identificación de los tipos de reconfiguración parcial dinámica que soporta cada propuesta, incluyendo en estos tipos la reconfiguración hardware/software de los `Cores Mixtos`. A partir de este análisis se identifican las carencias existentes para poder controlar la reconfiguración parcial dinámica sobre este tipo de sistemas. Esto permite definir en el capítulo 6 los objetivos que debe tener la propuesta realizada en esta tesis de forma que cubra adecuadamente las carencias detectadas.

En la parte II se desarrolla la propuesta innovadora de control de la reconfiguración dinámica, denominada sistema `Tornado`. Tras el capítulo 7 de introducción y justificación, se realiza en el capítulo 8 un planteamiento general de todo el sistema. Éste incluye la definición de un modelo multi-procesador reconfigurable mediante el sistema propuesto. El capítulo 9 aborda la infraestructura necesaria que se debe añadir a un diseño SoPC para convertirlo en un sistema que admita la auto-reconfiguración controlada mediante `Tornado`. En el capítulo 10, se detallan los parámetros que se han definido para caracterizar los procesos involucrados en `Tornado` y las ecuaciones que permiten estimar el coste de la infraestructura añadida. Tras este planteamiento teórico, en el capítulo 11 se desarrolla el control planteado por `Tornado` en la reconfiguración parcial dinámica de `Cores Mixtos`. Para finalizar esta parte, en el capítulo 12 se propone la utilización del sistema `Tornado` para controlar la utilización de la reconfiguración parcial dinámica para cambiar *cores* completos en un sistema basado en IP-Cores.

Los resultados de la validación experimental de la propuesta planteada, así como la aplicación de `Tornado` a una tecnología concreta se han plasmado en la parte III de esta tesis. En los capítulos 13, 14 y 15 se presentan el plan de pruebas y las características de la tecnología utilizada, las herramientas hardware y las herramientas software empleadas, respectivamente. Los capítulos 16, 17 y 18 recogen tres plataformas, tres diseños diferentes, en los que se ha aplicado `Tornado` y que sirven para validar experimentalmente la propuesta.

Finalmente, en la parte IV se resumen las aportaciones de esta tesis y conclusiones (capítulo 19) incluyendo una lista de publicaciones derivadas de este trabajo. Se completa esta tesis con la descripción de las líneas de trabajo futuro en el capítulo 20.

2. Sistemas Reconfigurables

2.1. Introducción

Los métodos convencionales para realizar computación son dos. El primero de ellos consiste en el procesamiento hardware utilizando circuitos cableados de forma fija, bien mediante la integración en un circuito de aplicación específica (*Application Specific Integrated Circuit* -ASIC-) o bien conectando componentes individuales en una placa [11]. El segundo método, denominado procesamiento software, se basa en el uso de microprocesadores que ejecutan un conjunto de instrucciones para realizar la computación.

El primer sistema se caracteriza por su rapidez y eficiencia para la aplicación concreta para la que ha sido diseñado, pero el circuito no puede ser alterado después de la fabricación, lo que le resta flexibilidad. Usando un microprocesador se incrementa la flexibilidad del sistema para poder cambiar la funcionalidad empleando otro software, pero se reduce la eficiencia debido a las secuencias necesarias de lectura, decodificación y ejecución de instrucciones.

Los dispositivos reconfigurables vienen a cubrir el espacio existente entre estos dos métodos, de forma que se disponga de la eficiencia del procesamiento hardware y de un alto grado de flexibilidad [12]. La adaptabilidad de las arquitecturas reconfigurables permite explotar el paralelismo existente en muchas aplicaciones de forma que se realice computación específica. En la figura 2.1 se muestra cómo los sistemas configurables, basados en dispositivos reconfigurables, se sitúan en una zona intermedia en la relación flexibilidad-especialización. No son tan flexibles como un procesador de propósito general ni tan específicos, ni tan óptimos, como un ASIC. Sin embargo se benefician de las características positivas de ambos.

Las diferencias más importantes entre la lógica reconfigurable y el procesamiento convencional se pueden resumir en los siguientes aspectos según K. Bondalapati y V. K. Prasanna [13]:

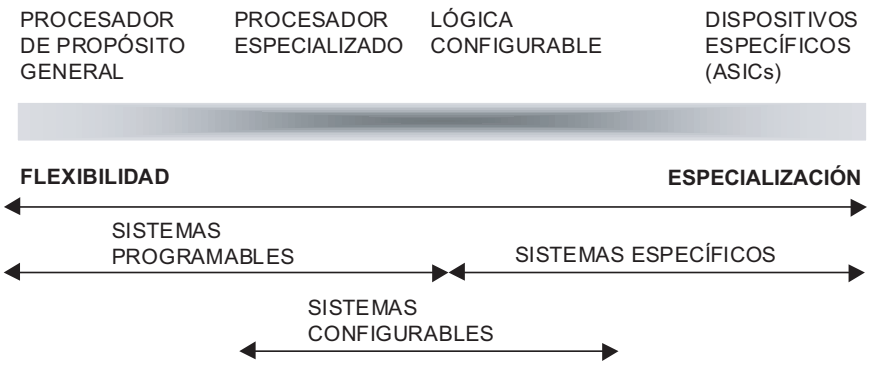


Figura 2.1.: Sistemas de computación según el grado de flexibilidad.

- **Computación espacial:** El procesamiento de los datos se realiza distribuyendo las computaciones de forma espacial, en contraste con el procesamiento secuencial.
- **Ruta de datos (*datapath*) configurable:** Empleando un mecanismo de configuración es posible cambiar la funcionalidad de las unidades de computación y de la red de interconexión.
- **Control distribuido:** Las unidades de computación procesan datos de forma local en vez de estar gobernados por una única instrucción.
- **Recursos distribuidos:** Los elementos requeridos para la computación se encuentran distribuidos por todo el dispositivo, en contraste con una única localización.

Otras características importantes de estos sistemas destacadas por A. DeHon [14] y R. Tessier [15] son la capacidad de adaptación, la posibilidad de configuración en tiempo de ejecución y la especialización. Beneficiándose de estas características específicas se han desarrollado sistemas reconfigurables eficientes para aplicaciones como la programación genética [16], detección de patrones de texto [17, 18], criptografía [19, 20, 21], compresión de datos [22] o procesamiento de imágenes [23, 24] entre otras.

En este capítulo se realizará un repaso de varias arquitecturas reconfigurables que han dado soporte al desarrollo de la disciplina de la Computación Reconfigurable. A lo largo del mismo se presenta la evolución sufrida desde los primeros dispositivos FPGAs hasta las plataformas híbridas que integran microprocesadores de propósito general y ASICs en el mismo *chip* entre otros elementos. De esta forma la distribución de la computación puede ser repartida entre los distintos componentes del sistema. También se ha incluido una sección donde se presentan los distintos modos de configuración de los dispositivos reconfigurables, siendo estas distintas modalidades uno de los factores más característicos de los mismos.

2.1.1. Tecnología reconfigurable: FPGAs

Las FPGAs consisten en una matriz de bloques lógicos *(Logic Blocks -LBs-)* y una red de interconexión. La funcionalidad de los LBs y las conexiones de la red de interconexión pueden modificarse mediante la descarga de los bits de configuración en el hardware [13]. La configuración del dispositivo se realiza empleando dispositivos *anti-fuse* [25] o bits de memoria SRAM que controlan la configuración de los transistores [26]. El primer modo de configuración tiene menor capacidad de reprogramación mientras que la configuración mediante elementos de memoria SRAM es más versátil admitiendo incluso reconfiguración dinámica y/o parcial.

La figura 2.2 muestra la estructura interna simplificada de una FPGA. A modo ilustrativo se ha elegido para la representación una distribución de tipo isla empleada en varias familias de Xilinx. Existen otras arquitecturas de interconexión como la basada en filas [27], *sea-of-gates* [28], jerárquica o estructuras en una única dimensión como las empleadas en Garp [19], Chimaera [29] o NAPA [30].

Los LBs interconectados mediante esa red contienen típicamente un circuito combinacional programable *Look-Up Table* (LUT), un biestable, lógica adicional y las celdas de memoria SRAM requeridas para la configuración de todos los elementos. Las tareas de entrada-salida se realizan en la periferia del dispositivo, bien mediante LBs o disponiendo

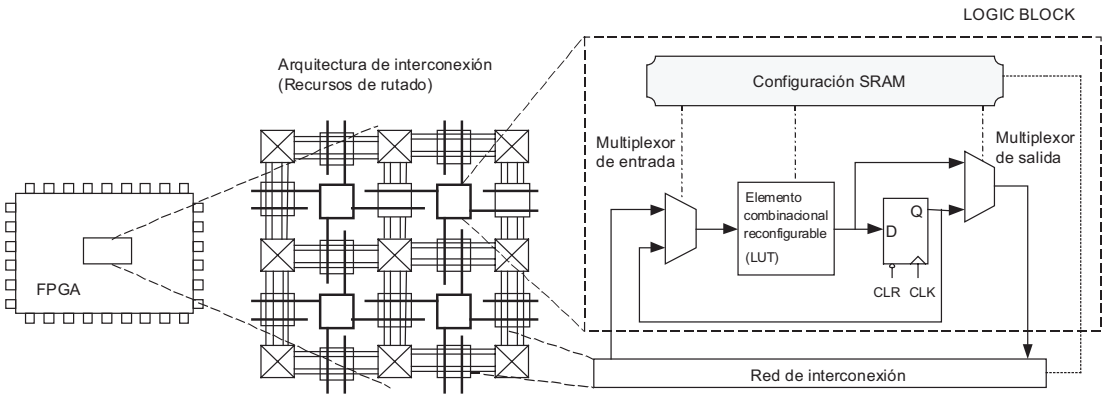


Figura 2.2.: Modelo genérico de una FPGA.

de bloques específicos denominados *Input-Output-Blocks (IOBs)*. Actualmente se integran habitualmente otros elementos como son los bloques de memoria RAM dedicada [31], multiplicadores e incluso microprocesadores.

Una de las clasificaciones más habituales de las FPGAs se realiza atendiendo a su granularidad. La granularidad de la lógica reconfigurable se define como el tamaño de la menor unidad funcional que es tratada por las herramientas de emplazamiento y rutado [13]. Las FPGAs de grano fino disponen de elementos funcionales de pequeño tamaño, lo que las dota de una mayor flexibilidad. Sin embargo, sufren de retardos elevados cuando se componen circuitos complejos. De forma típica son unidades funcionales de 2 a 4 entradas. Las FPGAs con unidades funcionales grandes se denominan de grano grueso, existiendo arquitecturas como la Chameleon [32] con elementos aritméticos de 32 bits.

Con la integración de múltiples elementos arquitecturales (entendidos como procesadores, memoria e interfaces para periféricos) en FPGAs que disponen de una sección de lógica programable por el usuario surge el concepto de *Arquitecturas Híbridas* [13].

El avance tecnológico que ha permitido la integración de sistemas en un único dispositivo ha ido acompañado de diversa terminología, todavía no normalizada, para designar a estos sistemas. A continuación se detallan algunos de estos términos:

- **System-on-Chip (SoC):** Circuito integrado formado por diversos módulos VLSI con distinta funcionalidad que interconectados entre sí ofrecen toda o casi toda la funcionalidad específica para una aplicación.
- **System-on-Programable-Chip (SoPC):** Se aplica este término específicamente cuando el dispositivo utilizado para realizar el sistema en un *chip* es reconfigurable. En los SoPC no se utiliza la capacidad de reconfiguración dinámica que puedan disponer estos integrados, sino únicamente las facilidades que ofrecen estos dispositivos en la fase de desarrollo y posteriores actualizaciones del sistema.
- **Configurable-System-On-Programable-Chip (CSoPC):** Mediante este término se definen los sistemas SoPC en los que se hace uso de la capacidad de reconfiguración de los mismos para aplicaciones de Computación Reconfigurable. Pueden incluirse bajo la denominación CSoPC tanto los sistemas que admiten diferente configuración

estática según ciertas condicionantes, como los que utilizan la reconfiguración parcial dinámica para modificar en tiempo de ejecución una sección hardware.

- **Multiprocessor-Configurable-System-On-Programable-Chip (MCSoPC):** Se aplica esta definición a los sistemas CSoPC que incluyen varios procesadores que ejecutan software, funcionando de forma simultánea.

En los apartados 2.2 y 2.3 se presentarán brevemente algunas de las arquitecturas reconfigurables, tanto académicas como comerciales, más relevantes resultado de las investigaciones realizadas en los últimos años.

2.1.2. Tipos de configuración

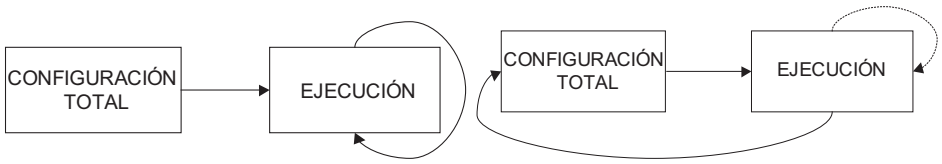
De forma general un dispositivo reconfigurable se configura cargando en el mismo una secuencia de bits denominada *bitstream*. El modo de carga varía según la interfaz que éste disponga. Las interfaces de configuración son de tipo serie o paralelo. El tiempo de configuración es directamente proporcional al tamaño del *bitstream*. Las FPGAs de grano fino tienen, en general, tiempos de configuración mayores que las de grano grueso debido al mayor tamaño de sus *bitstreams*. Esto es debido a que tienen muchos elementos para ser configurados lo que implica ficheros de configuración grandes.

Las FPGAs tradicionalmente se han utilizado para realizar una determinada función en un único contexto, realizándose una configuración de todo el dispositivo. En el caso de que se desee reconfigurar en tiempo de funcionamiento, la reconfiguración de todo el dispositivo es un proceso lento y limitado. La lógica que se va a reconfigurar debe parar la computación y continuar tras la nueva configuración. La penalización impuesta por el tiempo de reconfiguración es importante [33], haciendo en muchas aplicaciones inviable la aplicación de la reconfiguración. A continuación se presentan brevemente los modelos de reconfiguración más representativos:

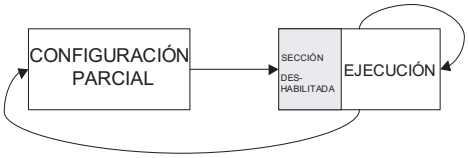
- **Reconfiguración estática:** Implica parar el sistema y reiniciarlo con una nueva configuración. Su utilidad se centra en los procesos de diseño (depuración) y en la actualización de sistemas. Cada aplicación dispone de una configuración que se carga una vez tras el encendido. La mayoría de los sistemas realizados en la actualidad con lógica reconfigurable disponen de este tipo de reconfiguración, también denominada reconfiguración en tiempo de compilación (en el proceso de diseño). La figura 2.3(a) sintetiza este modo de operación en el que tras la configuración comienza la ejecución de la lógica configurada sin posibilidad de una nueva carga.

- **Reconfiguración dinámica:** Con el objetivo de obtener un equilibrio entre velocidad de ejecución y área de silicio surge el modo de configuración dinámico mediante el cual se modifica la lógica configurable (incluyendo el rutado) en tiempo de ejecución. La reconfiguración dinámica se basa en el concepto de *Hardware Virtual* [34] de forma similar a la memoria virtual. Utilizando la capacidad de reprogramación del dispositivo se cambian las configuraciones según se requieren distintas computaciones, reduciendo de esta manera el área de circuito necesaria.

  La figura 2.3(b) representa el modelo de reconfiguración dinámico. De la computación realizada por la lógica configurada (ejecución) se obtiene información que sirve para



(a) ***Modelo estático reconfiguración.*** **(b)** ***Modelo dinámico de reconfiguración.***



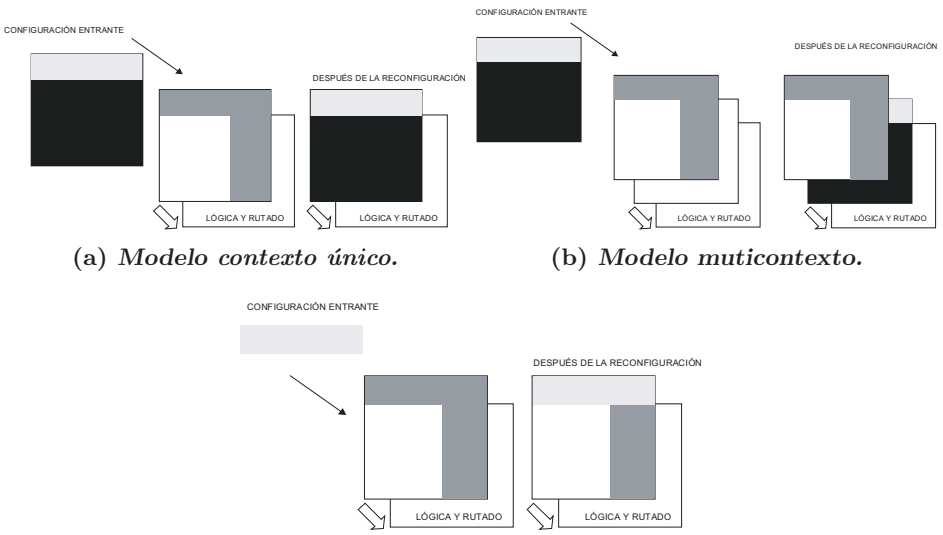
(c) ***Modelo parcial y dinámico de reconfiguración.***

Figura 2.3.: Modelos de configuración.

determinar la nueva configuración. En el caso de que el sistema lo permita, se podría realizar la aplicación de la nueva configuración mientras se mantiene la ejecución. En la práctica, la solución más habitual es la de mantener deshabilitada la sección que se va a reconfigurar mientras continúa la ejecución en la otra sección del dispositivo. Este modo de reconfiguración parcial dinámico se representa en la figura 2.3(c).

Por tanto, la reconfiguración dinámica tiene diversas variantes según el modo en el que ésta se aplique. Las tres más representativas son:

- **Reconfiguración contexto único:** Corresponde con el modo de configuración de las FPGAs que únicamente soportan un acceso secuencial a la memoria de configuración. En el caso de realizar una reconfiguración dinámica con estos dispositivos se sufren unas penalizaciones temporales importantes debido a que cada intercambio de funcionalidad requiere una reprogramación completa de los mismos. El modelo de este modo se ha representado en las figuras 2.3(b) y 2.4(a). La configuración entrante sustituye completamente a la que estaba aplicada sobre la lógica configurable.
- **Reconfiguración multi-contexto:** Los dispositivos que soportan este tipo de reconfiguración tienen varios bits de memoria de configuración para cada bit de los elementos configurables [35, 36]. En la figura 2.4(b) se representa un modelo de estos dispositivos donde los bits de memoria pueden considerarse como múltiples planos de información de configuración. Cada plano debe configurarse totalmente, de igual forma que los de contexto único. Sin embargo, el cambio entre contextos se realiza de forma muy rápida, admitiéndose además la carga de una nueva configuración en un plano no activo mientras otro lo está.
- **Reconfiguración parcial:** Uno de los avances tecnológicos más importantes en el área de la reconfiguración consiste en la capacidad de algunos dispositivos



(a) ***Modelo contexto único.*** **(b)** ***Modelo muticontexto.***

(c) ***Modelo parcialmente reconfigurable.***

Figura 2.4.: Modelos de reconfiguración dinámica.

para admitir la modificación de parte de la configuración mientras el resto del hardware sigue realizando la computación de forma ininterrumpida.

La figuras 2.3(c) y 2.4(c) muestran este modelo de reconfiguración. En este caso el plano de configuración funciona como una memoria RAM. De este modo se pueden emplear las direcciones para especificar una determinada localización que de desea reconfigurar. La reconfiguración parcial dinámica también permite que se carguen configuraciones diferentes en áreas no usadas del dispositivo con el fin de reducir la latencia en el cambio de contexto, tal y como se propone en el trabajo de K. Danne [37].

En los apartados 2.2 y 2.3 se presentarán varias plataformas reconfigurables que soportan la reconfiguración parcial como Chimaera [29], PipeRench [38], NAPA [30], Xilinx 6200 [39] y Virtex [40].

Una variante de la reconfiguración parcial realizada con el fin de reducir la penalización por el tiempo necesario para la carga de los *bitstreams* parciales es la *reconfiguración pipeline*. En este caso, la reconfiguración ocurre en incrementos de etapas de *pipeline* [41]. Este sistema está orientado a computaciones de estilo *datapath*, donde se emplean más etapas *pipeline* que las que caben simultáneamente en el hardware. El caso más avanzado sería una situación donde comenzaría la computación de cada etapa tras el instante de ser programada. En este caso la configuración de cada etapa se situaría un paso a la cabeza del flujo de datos [11].

Para aliviar la problemática de la penalización en tiempo impuesta por el proceso de reconfiguración, se han investigado diversas tácticas. El *prefetching* de configuraciones [42] es una de ellas. El objetivo en este caso es cargar la configuración en el dispositivo con anti-

cipación a que se requiera ésta. El carácter especulativo de esta técnica fija la complejidad de la misma en determinar con suficiente antelación cuál va a ser la siguiente configuración requerida. También la compresión de la configuración se ha estudiado como una de las alternativas para la reducción del tiempo de configuración [43].

Otra alternativa abordada con el mismo fin es la del uso de *caché* de configuraciones en el dispositivo [44]. Al retener configuraciones en el integrado, se reduce la cantidad de información de configuración transferida. Al igual que en las *cachés* de los procesadores de propósito general, se aplican los conceptos de localidad temporal y espacial con el fin de decidir qué configuraciones se mantienen y cuáles se eliminan cuando se produce la reconfiguración. Decisiones incorrectas pueden producir el efecto contrario al deseado, aumentando la penalización temporal producida por la reconfiguración.

2.2. Plataformas reconfigurables experimentales

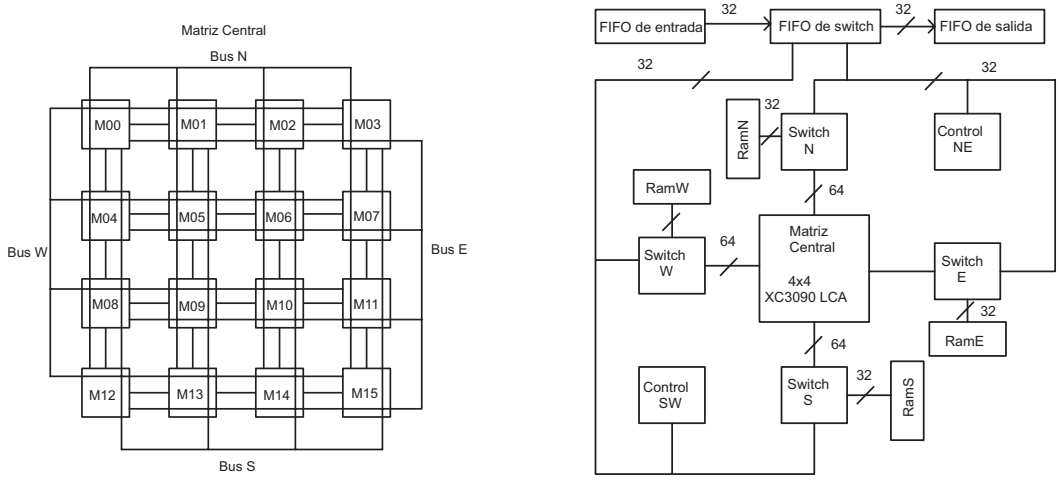
A continuación se presentan una serie de plataformas reconfigurables relevantes resultado de las investigaciones realizadas en los últimos años. Algunas son sistemas híbridos que conjugan secciones reconfigurables interconectadas con procesadores de propósito general, mientras que otros sistemas son procesadores reconfigurables experimentales.

2.2.1. Aceleradores Hardware

Una referencia obligada en esta área es la de los sistemas Splash y Splash2 [45]. Desarrollados a mediados de los años 90, se trataban placas para inserción en el bus de expansión de estaciones de trabajo para realizar computaciones de alto rendimiento en problemas con procesamiento a nivel de bit. La placa Splash contenía 32 FPGAs 3090 de Xilinx conectadas mediante una matriz lineal de 32 bits de anchura. La conexión al *host* se realizaba con dos buses, uno para la transferencia de datos y otro para las configuraciones. El sistema incluía herramientas para *mapear* los algoritmos en los dispositivos reconfigurables y rutinas en C para poder acceder a la placa desde programas.

Splash2 mejoraba a su antecesora en aspectos como la velocidad de entrada y salida (10 veces mayor) y en la capacidad de computación al incluir dispositivos de la familia XC4000 de Xilinx. Éstos incluían lógica rápida para el acarreo soportando mayor velocidad de funcionamiento y un número de bloques de lógica reconfigurable (*Configurable Logic Blocks* -CLBs-) superior. Además, permitía operaciones de *broadcast* sobre otras placas Splash2 conectadas al bus, disponiendo de mayor capacidad de memoria.

Dentro de esta área de sistemas aceleradores hardware programables conectados a una estación de trabajo cabe destacar la plataforma DECPeRLe-1 [46]. La matriz central estaba compuesta por 16 FPGAs XC3090 de Xilinx conectadas a 4 buses de 64 bits. Estos buses se representan en la figura 2.5(a) como los buses Norte (N), Sur (S), Este (E) y Oeste (W) de la matriz. Éstos, a su vez, se conectan con otras XC3090, en este caso utilizadas como *data switches*. En la figura 2.5(b) se observa cómo cada uno de estos *switches* se conecta a un módulo de memoria SRAM de 256K x 32. Un quinto *switch* denominado *FIFO switch* conecta el sistema DECPeRLe-1 con el *host* a través de una ranura de expansión TurboChannel^{TM} .



(a) ***Matriz Central.*** **(b)** ***Arquitectura del sistema.***

Figura 2.5.: Plataforma DECPeRLe-1

El DECPeRLe-1 se programaba empleando C++ como lenguaje de descripción hardware. Para ello se proveía una biblioteca C++ con primitivas de bajo nivel para la descripción de los circuitos. Mediante funciones C y declaraciones de clases C++ se podían construir diseños jerárquicos *mapeando* el diseño en la arquitectura de la placa empleando un formato de fichero de diseño estándar.

2.2.2. Coprocesadores Reconfigurables

La plataforma Garp [19, 47] desarrollada en la Universidad de Berkeley se engloba en el conjunto de los sistemas Coprocesadores Reconfigurables, ya que dispone de acceso a la memoria del procesador central. La matriz reconfigurable puede ser reconfigurada de forma parcial gracias a su distribución en columnas.

En esta arquitectura, la FPGA funciona como una unidad computacional esclava localizada en el mismo integrado que el procesador. La figura 2.6 muestra el diagrama de bloques del sistema a alto nivel.

Este sistema está diseñado para trabajar en un entorno similar al de un procesador de propósito general. Éste incluye programas estructurados, bibliotecas, cambios de contexto, memoria virtual y múltiples usuarios. La mayor parte de la ejecución del programa la realiza el programa principal. Sin embargo, para ciertas subrutinas o lazos, el programa es capaz de pasar a utilizar la matriz reconfigurable. El control se realiza desde el programa principal ejecutándose en el procesador. De esta forma se obtienen mejoras sustanciales en la velocidad de procesamiento. La plataforma Garp se compone de entidades denominadas *blocks*. Por cada columna, uno de los *blocks* es de control mientras que el resto son *blocks* de lógica.

Garp en comparación con las FPGAs convencionales requiere más lógica para operaciones como sumas y desplazamientos. Cada fila de Garp se puede aproximar a una Unidad Arimético Lógica (UAL) convencional. Sin embargo, al encontrarse la sección reconfigurable incluida en el mismo integrado que el procesador, se pueden obtener importantes beneficios

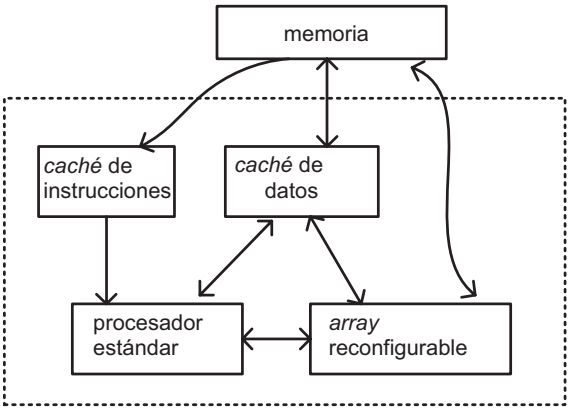


Figura 2.6.: Diagrama de bloques del sistema Garp.

en velocidad debido a la rápida reconfiguración del dispositivo.

La arquitectura de OneChip [48, 49] también incluye una lógica estática compuesta por un *core* de un procesador MIPS con lógica reconfigurable. Entre las aplicaciones adecuadas para esta plataforma se incluyen sistemas de control que requieren interfaces específicas y sistemas aceleradores por hardware para computaciones intensivas. Un ejemplo de este último tipo de computaciones es el cálculo de la transformada discreta del coseno, habiéndose comprobado una mejora en velocidad con OneChip de unas 50 veces respecto al procesamiento únicamente mediante software.

El sistema Chimaera [29] integra, al igual que Garp y OneChip, una unidad reconfigurable en el propio integrado donde se encuentra el procesador. Con esta estrategia se quiere resolver la falta de eficiencia en el sistema de comunicación de los sistemas de computación específicos formados por procesadores y unidades reconfigurables.

El sistema integra la sección reconfigurable de procesamiento hardware (*Reconfigurable Functional Unit* -RFU-) en el propio *pipeline* del procesador *superescalar*. Chimaera trata la sección reconfigurable como una *caché* de instrucciones RFU implementadas en hardware, modificado mediante tecnología de reconfiguración parcial dinámica. Por tanto, dado el carácter especulativo de la *caché*, se aplican técnicas de *prefetching* y algoritmos de *catching* para reducir el número de reconfiguraciones erróneas. El modelo de procesamiento hardware en paralelo de grano fino que las operaciones RFU ofrecen es más potente que las actuales extensiones de instrucciones para multimedia [29].

La figura 2.7 muestra la arquitectura general del sistema. En la sección reconfigurable (*Reconfigurable Array* -RA-) se ejecutan las operaciones. La Unidad de Control de Ejecución (*Execution Control Unit* -ECU-) decodifica la instrucción y guía la ejecución. Esta unidad está comunicada con la lógica del procesador para coordinar la ejecución de las instrucciones RFU. La sección Control de Configuración y Unidad de Caché (*Configuration Control and Cache Unit* -CCCU-) realiza la gestión de la sección reconfigurable RA como *caché* cargando las configuraciones parciales necesarias. Los registros denominados en la figura *ocultos* (en el sistema Chimaera se identifican como *Shadow File Registers* -SFR-) son una copia de un subconjunto de los registros del procesador. Éstos sirven de entrada de datos para la RA.

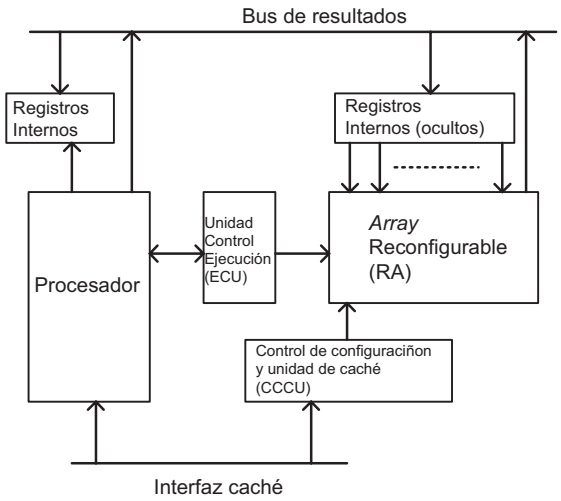


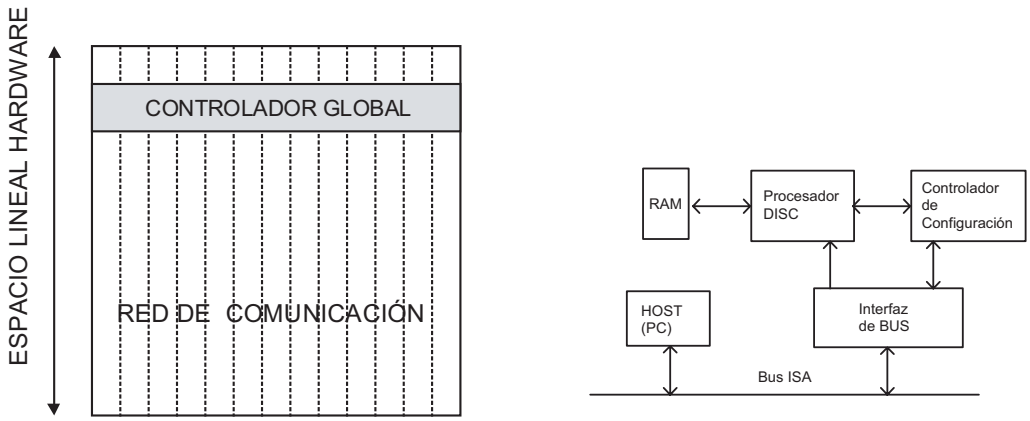
Figura 2.7.: Arquitectura simplificada del sistema Chimaera.

La FPGA implementada en la sección reconfigurable dispone de una arquitectura más simple que la de las FPGAs de propósito general, estando optimizada para operar de forma rápida conjuntamente con el procesador. Está compuesta por una serie de bloques lógicos programables distribuidos en filas. Cada una contiene el número de bloques lógicos necesarios para procesar el tipo de dato que requiera mayor número de bits. Cada bloque puede configurarse como una LUT de 4 entradas, dos de 3 entradas o una LUT de 3 entradas y lógica para computación de la llevada.

Chimaera se ha implementado en un circuito integrado que ha sido utilizado para comparar su rendimiento frente a un procesador MIPS funcionando a frecuencia de 150 MHz [29]. Para evaluar su rendimiento, los autores han analizado aplicaciones de criptografía usando el algoritmo DES obteniéndose un factor de mejora de 7.8 respecto al procesamiento con el MIPS. También se ha evaluado el sistema para aplicaciones de codificación de vídeo MPEG-2 obteniendo un factor de mejora de 1.5.

El procesador *Dynamic Instruction Set Computer* (DISC) [50, 51] constituye uno de los avances más destacados en el área de la Computación Reconfigurable durante los años 90. Esta arquitectura plantea el uso de recursos hardware relocalizables sobre una FPGA, hoy obsoleta, de National Semiconductor llamada CLAy31 junto con una memoria RAM externa. Mediante reconfiguración parcial es posible cargar y descargar instrucciones fuera y dentro de la FPGA según sea demandado por el programa que se está ejecutando. Los módulos se distribuyen en la FPGA de forma lineal siguiendo el modelo denominado *Linear Hardware Model* (LHM) [50] mediante el cual la rejilla bidimensional de la FPGA se distribuye en filas de anchura igual a la de toda la FPGA (véase figura 2.8(a)). El procesador se completa con un controlador derivado de un microprocesador de 8 bits integrado en una sección de la FPGA.

El propototipo desarrollado para la evaluación este procesador se realizó mediante una tarjeta para inserción en el bus de expansión *Industry Standard Architecture* (ISA). La figu-



(a) ***Modelo LHM.***

(b) ***Arquitectura de la tarjeta ISA para la evaluación del sistema DISC.***

Figura 2.8.: Procesador *Dynamic Instruction Set Computer* (DISC).

ra 2.8(b) representa los bloques de esta plataforma experimental. Ésta incluía dos FPGAs habiéndose implementado el controlador y la sección dinámicamente reconfigurable en cada una de las FPGAs, respectivamente. El rendimiento del DISC se ha comparado para aplicaciones de visión (filtros) [50] con un procesador de propósito general obteniéndose mejoras en velocidad de procesamiento de más de 23 veces la velocidad de referencia del procesador.

P.C. French y R.W. Taylor ya planteaban en 1993 la idea de diseñar procesadores con una sección fija mínima optimizando la posibilidad de reconfiguración de las FPGAs [52]. En esta línea otra propuesta de procesadores con instrucciones variables para FPGAs se recoge en los trabajos de S.P. Seng, W. Luk y P.Y.K. Cheung [53, 54] donde se presenta el concepto de Procesadores de Instrucciones Flexibles (*Flexible Instruction Processors* -FIPs-). El FIP es un esqueleto de procesador y un conjunto de parámetros que permiten obtener diferentes implementaciones del procesador según la aplicación. En [54] se propone además la aplicación de la reconfiguración en tiempo real de los mismos para modificar sus características.

2.2.3. Configurable-Systems-on-Chips: CSoCs

La integración de secciones reconfigurables dentro de los SoCs también es un área de investigación intensiva relacionada con la Computación Reconfigurable, habiendo evolucionado en esta década hacia los sistemas configurables integrados en un *chip* (*Configurable-Systems-on-Chip* -CSoCs-). Aunque se basan en tecnología ASIC, con la consiguiente eficiencia de la misma, también se consigue reducir costes y tiempos de diseño al integrarse secciones reconfigurables.

El integrado FIPSOC [55] es uno de los dispositivos pioneros en el área de los CSoC. Integra un *core* de microcontrolador 8051 junto con una área reconfigurable FPGA de grano grueso y módulos analógicos. Cabe destacar su posibilidad de admitir reconfiguración

dinámica de la sección reconfigurable en un único ciclo de escritura del controlador de reconfiguración, que es el propio microcontrolador embebido. Esta velocidad en la reconfiguración se consigue disponiendo de dos contextos de memoria de configuración a los cuales puede acceder el microprocesador: uno activo y otro en modificación.

El procesador Pleiades [56] es otro ejemplo representativo de estos sistemas. Combina un microprocesador con un conjunto de unidades computacionales programables con diferentes granularidades, denominadas satélites, en un único circuito integrado. La conexión entre los módulos se realiza mediante una red de interconexión. La configuración y los procesos de control son tareas del microprocesador, mientras que las tareas de computación regulares e intensivas se realizan en los satélites.

El CSoC Maia [57] (figura 2.9), también desarrollado por este grupo de investigación, integra en el mismo *chip* un procesador ARM8 con 21 satélites procesadores. Estos satélites son dos unidades especializadas en el procesamiento de comunicaciones a alta velocidad (*Media Access Controller* -MACs-), dos ALUs, ocho generadores de direcciones, ocho memorias embebidas y un *core* FPGA de grano fino y de bajo consumo. En este caso el CSoC es configurable en dos aspectos, la interconexión con los satélites y la FPGA. La secuencia de aplicación de la configuración puede variar de forma dinámica estando controlada por un *Kernel* de procesamiento en tiempo real que se ejecuta en el microprocesador [58]. Este sistema está optimizado para sistemas de terminales móviles de telefonía.

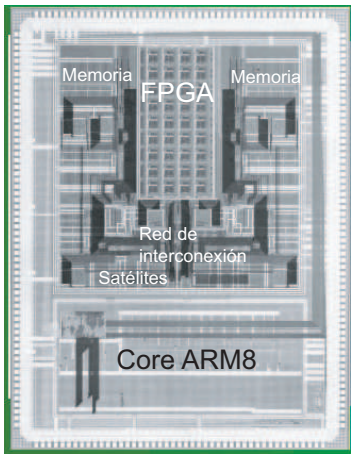
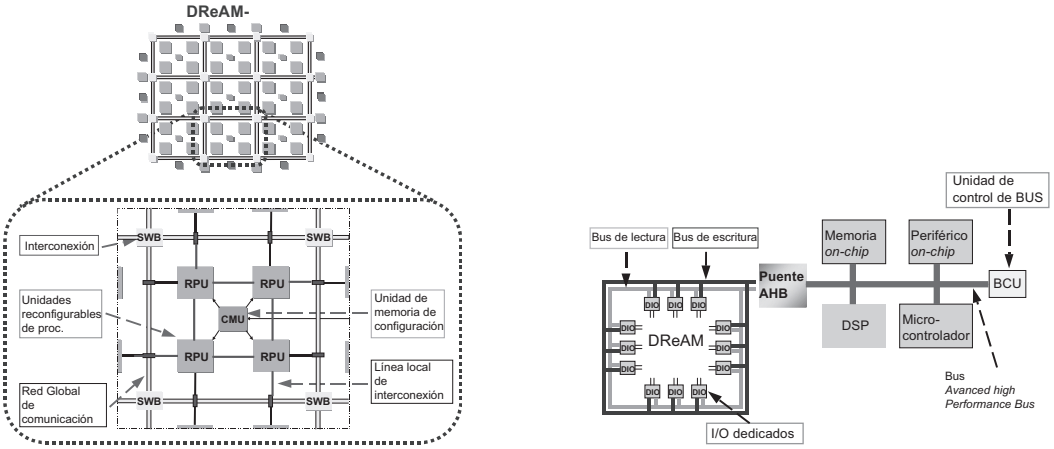


Figura 2.9.: Maia CSoC. Procesador reconfigurable heterogéneo.

Otro ejemplo relevante de CSoC, también orientado a aplicaciones de telefonía de tercera generación (3G), es el CSoC *Dynamically Reconfigurable Architecture for Mobile communication systems* (DReAM) [59, 60] desarrollado por la Universidad de Darmstadt. En este caso, se trata de un sistema dinámica y parcialmente configurable formado por módulos de grano grueso denominados *Reconfigurable Processing Units* (RPUs). Estos elementos se interconectan a través de una red local y global de comunicación. La principal función de estos RPUs es la manipulación de datos para las secciones de procesamiento digital de señal.

La figura 2.10(b) detalla la integración del módulo reconfigurable DReAM en un CSoC mediante un *bridge* que conecta a éste con el bus *on-chip*, bus AMBA [61] en este caso.

A modo de ejemplo J. Becker presenta en [59] la especificación de un receptor RAKE implementado en tecnología de 0.35μm CMOS para esta plataforma.



(a) *Módulo reconfigurable DReAM.* **(b) *Módulo reconfigurable DReAM integrado en un CSoC.***

Figura 2.10.: *Dynamically Reconfigurable Architecture for Mobile communication systems* (DReAM) CSoC.

La arquitectura XPP$^{\mathrm{TM}}$[62, 63] consiste, al igual que los dos casos anteriores o la arquitectura *polimórfica* TRIPS [64], en una matriz reconfigurable diseñada para ser integrada en CSoCs. Es una estructura parcial y dinámicamente reconfigurable de grano grueso capaz de soportar distintos tipos de paralelismos: *pipeline*, procesamiento *dataflow*, ejecución de tareas en paralelo, etc. Está orientada a aplicaciones de procesamiento digital de señal, multimedia y similares.

Los elementos básicos adaptativos de la estructura reconfigurable se denominan *Processing Array Elements* (PAEs), los cuales incluyen, entre otros elementos, una Unidad Aritmético Lógica reconfigurable donde se realizan las operaciones. Los PAEs se agrupan en *Processing Arrays* (PAs). Las configuraciones se determinan a partir de los nodos del gráfico del flujo de datos correspondiente al algoritmo que se desea implementar. Durante la computación el gráfico permanece estático. En esta situación no se modifican las conexiones ni los operadores, por lo que, a diferencia del proceso secuencial en un procesador donde se requiere un proceso de decodificación de instrucciones, se obtiene un rendimiento óptimo por ciclo de reloj [65]. Las configuraciones se aplican de forma secuencial compensando el sobrecoste del tiempo de reconfiguración mediante la ejecución de múltiples operaciones en paralelo, disponiendo además, de un sistema de configuración distribuido mediante varios controladores de configuración.

A partir de esta estructura ha surgido una nueva familia de *cores* y procesadores autoreconfigurables que se están aplicando en distintos proyectos. Desde el prototipo XPU128-ES realizado por *PACT Corporation* [66] hasta la integración de una estructura XPP$^{\mathrm{TM}}$junto con un procesador Leon [67] compatible con SPARC y bus AMBA [61] para la composición de un CSoC en tecnología de 0.18μm CMOS [68, 69] (figura 2.11).

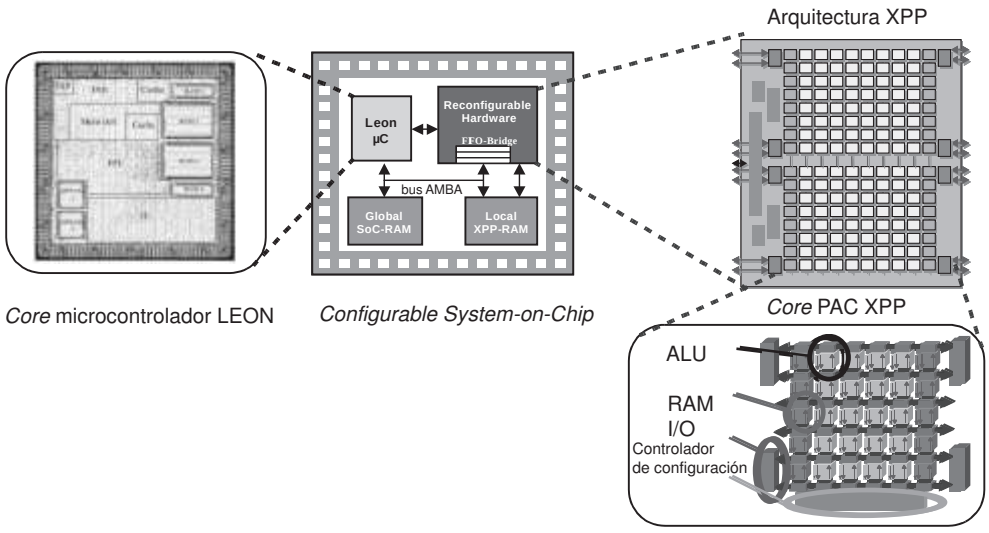


Figura 2.11.: CSoC académico/comercial compuesto por un *core* Leon y *cores* PAC XPP^{TM} .

Entre otras propuestas en la misma línea que XPP^{TM}, pero empleando distintos modelos de ejecución, destacan la de R. Enzler presentada en [70] o la arquitectura *Self-Reconfigurable Gate Array Architecture* (SRGA) de R. Sidhu [71]. Esta última arquitectura soporta un cambio de contexto en un único ciclo. También destacan otras propuestas experimentales como el integrado reconfigurable Colt [72] desarrollado por la Universidad de Virginia, el cual emplea un sistema de reconfiguración (*Wormhole*) para sus unidades de computación distribuidas que incluye en el *bitstream* tanto la información configuración como los datos de la computación.

2.3. Dispositivos comerciales reconfigurables dinámicamente

Como resultado de estas investigaciones están surgiendo dispositivos comerciales como el XPP/PACT ofertado por *PACT Corporation* [66] realizado a partir de la arquitectura XPP^{TM}. Incluso ya hay disponibles de forma comercial FPGAs con capacidad multicontexto, como la serie CS2000 RCP de Chameleon Inc. [32]. Este dispositivo está dotado de dos planos separados de configuración. Mientras que uno de ellos define la configuración de la lógica reconfigurable, el otro queda en espera de ser cargado con la siguiente configuración. Este dispositivo integra además un procesador dentro del mismo dispositivo, conformando un CSoC de grano grueso.

De igual forma, la compañía Triscend [73] fabrica un dispositivo denominado específicamente *Configurable System-on-Chip*, el cual combina en un mismo circuito integrado, un procesador, un controlador de acceso directo a memoria, memoria interna, controladores de memoria externa, lógica programable por el usuario, periféricos y un bus SoC específico [74].

Frente a estas estructuras específicas de grano grueso, potentes, pero actualmente no muy extendidas, se sitúan las FPGAs comerciales de grano fino. Éstas son ampliamente

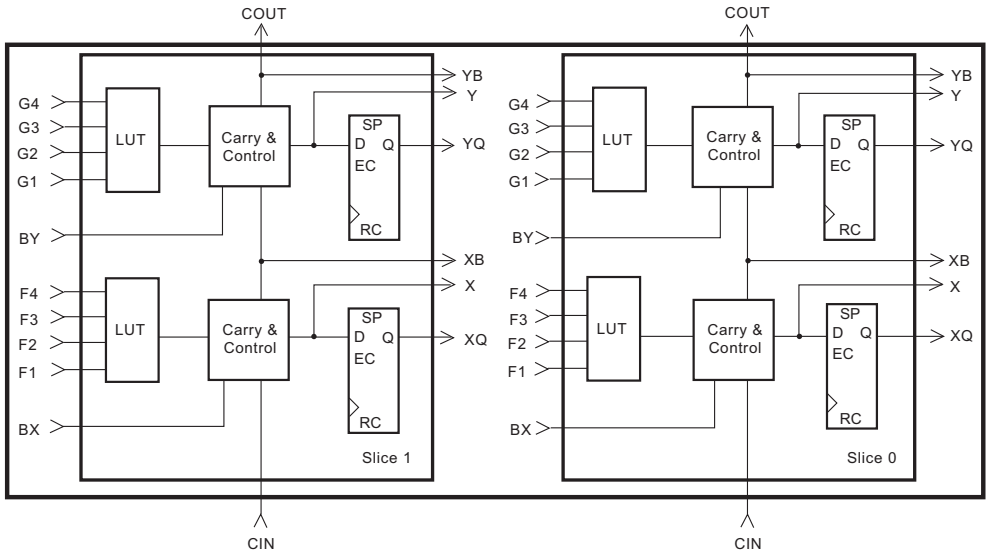


Figura 2.12.: *Configurable Logic Block* de Virtex (2 *Slices*).

utilizadas. Principalmente no por su capacidad de reconfiguración dinámica y/o parcial (actualmente soportada por un conjunto reducido de familias), sino por su facilidad para realizar desarrollos rápidos y posteriores modificaciones.

2.3.1. FPGAs de grano fino reconfigurables dinámicamente

En la actualidad las FPGAs de grano fino reconfigurables dinámicamente disponibles en el mercado son familias de dispositivos de los fabricantes Xilinx y Atmel. Las FPGAs de ambas familias disponen de una memoria de configuración de tipo SRAM. Los dispositivos de las familias de Xilinx que se presentan a continuación son los más utilizados en el área de investigación de la Computación Reconfigurable con FPGAs comerciales de grano fino. Sin embargo, el fabricante Atmel dispone también de FPGAs que admiten la reconfiguración parcial dinámica.

Los circuitos integrados de las familias Spartan-II, Spartan-IIE, Virtex, Virtex-E y Virtex-II de Xilinx son reconfigurables parcial y dinámicamente. Puesto que disponen de una estructura interna muy similar (las familias Spartan-II [75] y Spartan-IIE son las versiones de bajo coste de Virtex) se resumen a continuación las características más relevantes de la arquitectura interna y de configuración de los dispositivos Virtex [40].

Los elementos configurables más importantes son los bloques lógicos configurables (*Configurable Logic Blocks* -CLBs-) y los bloques de entrada-salida (*Input/Output Blocks* -IOBs-).

El elemento básico del CLB de Virtex es la celda lógica (*Logic Cell* -LC-). En la figura 2.12 se representa la estructura interna de una LC. Incluye un circuito programable LUT de 4 entradas generador de funciones, la lógica de acarreo y un elemento de almacenamiento que puede usarse para guardar (o sincronizar con un reloj global) el resultado de la LUT de 4 entradas. Cada CLB de Virtex dispone de cuatro LCs, distribuidas en dos *slices*.

Destaca en estas familias de Xilinx la versatilidad de las LUTs. Una LUT, además de poder

funcionar como generador de funciones combinacionales de 4 entradas, puede emplearse como memoria RAM síncrona de 16 x 1 bit (o de diferentes tamaños combinando varias LUTs) o como registro de desplazamiento de 16 bits [76].

La comunicación entre los *pines* del encapsulado y los CLBs se realiza a través de los IOBs. La interconexión entre los diferentes CLBs se realiza mediante una matriz global de rutado. La configuración se realiza mediante unos *switches* situados en la intersección entre las rutas horizontales y verticales. Además de las conexiones globales, la FPGA dispone de numerosas rutas locales para unir los CLBs con la matriz global de rutado y también para realizar conexiones rápidas (cortas) entre ellos.

Además de los CLBs y los IOBs, completan el dispositivo los siguientes elementos:

- **Bloques de memoria RAM dedicados** de 4096 bits [31]. Se utilizan como una alternativa eficiente a la memoria distribuida construida empleando LUTs ya que no emplean recursos de propósito general de la FPGA. La distribución de estos bloques se realiza en columnas, cubriendo toda la altura del dispositivo. También dispone de recursos de rutado dedicados para conectar éstas con las CLBs. El contenido de la RAM se puede fijar mediante la memoria SRAM de configuración, incluyéndose en el *bitstream* inicial o en posteriores *bitstreams* parciales. Esta característica permite el uso de estos bloques de RAM como memorias ROM.

- ***Delay-Locked Loops* (DLLs)** [77]. Se emplean para compensar el retardo por la distribución del reloj y para realizar el control de los relojes globales (según los modelos, distintas posibilidades de multiplicación, división, etc.).

- ***Buffers* triestado**. Se utilizan para conectar CLBs a las rutas horizontales globales con la posibilidad de utilizar el valor lógico 'Z'(alta impedancia), obteniéndose buenos resultados en términos de ahorro en recursos y en velocidad máxima de funcionamiento [78]. En los dispositivos de Xilinx se denominan *Tbufs*.

La configuración de todos estos recursos, lógica y rutado, se realiza mediante valores almacenados en celdas de memoria SRAM. Cambiando estos valores durante el funcionamiento del dispositivo se puede cambiar su funcionalidad. Esto es, mediante reconfiguración dinámica.

Estos dispositivos admiten diversos sistemas para cargar el *bitstream* en la memoria SRAM de configuración [79]. Hay tres modos principales, admitiéndose en cada uno de ellos la opción de mantener el estado de los *pines* de salida en *pull-up* o en alta impedancia durante el estado de reconfiguración:

- **Configuración serie.** Mediante este sistema de configuración la FPGA es capaz de leer el *bitstream* almacenado en una PROM serie exterior. Se distinguen dos variantes: configuración serie *Master*, en la que la FPGA genera el reloj sincronizador de la carga, y configuración serie *Slave*, donde el reloj lo genera otro dispositivo externo. Este último modo permite que la carga se realice desde un microprocesador externo o en modo *daisy-chain*, mediante el que se pueden configurar varios dispositivos conectados en cadena.

- **Configuración paralelo.** Estas familias de Xilinx disponen de un potente sistema de configuración a través de un puerto paralelo de 8 bits denominado *SelectMap*

[79, 80, 81]. La memoria de configuración está dividida en *frames*. La carga de información en estos *frames* se realiza introduciendo a través del SelectMap *frames* de datos concatenados en el *bitstream*. Aunque la interfaz es de 8 bits, la configuración se realiza mediante paquetes de 32 bits.

Internamente el SelectMap dispone de varios registros internos: el registro de comandos, el de control y el de dirección del *frame* entre otros. Los paquetes de configuración de 32 bits pueden incluir información para estos registros. De esta forma se controlan operaciones avanzadas como la lectura de la memoria de configuración [82] o la reconfiguración parcial dinámica.

- **Configuración *Boundary-Scan* (JTAG).** La configuración a través de la interfaz *Boundary-Scan* se realiza mediante el protocolo serie JTAG [83]. La interfaz está activa desde la inicialización del dispositivo y está accesible en todo momento. Habitualmente es el sistema utilizado en la fase de desarrollo y depuración.

Los detalles relativos a la configuración parcial dinámica con estos dispositivos se ampliarán en el apartado 13.2 de esta tesis, ya que los ensayos se han realizado con dispositivos de estas familias.

En el caso de Atmel, se dispone de las familias AT40K, AT40KLV [84] y AT6000 [84]. Los dispositivos de las dos primeras familias son FPGAs con memoria de configuración SRAM y compatibles con el bus PCI. Disponen de memoria SRAM distribuida y dedicada de alta velocidad con múltiples posibilidades de funcionamiento (síncrona, asíncrona, doble puerto, etc.). Las celdas incluyen multiplicadores dedicados y conexiones directas horizontales, verticales y diagonales entre ellas. Pueden implementar cualquier par de funciones booleanas de tres entradas o una de cuatro.

Las celdas de los dispositivos de la familia AT6000 son pequeñas y simples. Son simétricas con cuatro lados funcionalmente iguales (figura 2.13) y pueden usarse como elementos de rutado local. Las celdas están unidas formando una matriz de un lado al otro del dispositivo interrumpida únicamente por los repetidores de bus situados cada 8 celdas.

El dispositivo de mayor tamaño de estas familias alcanza las 50.000 puertas y 2.304 biestables. Si se requieren tamaños mayores, la familia AT6000 alcanza las 30.000 puertas y los 6.400 biestables, adecuándose a aplicaciones de coprocesadores reconfigurables.

Desde el punto de vista de la reconfiguración parcial dinámica hay que destacar que las familias AT40K, AT40KLV y FPSLIC (sección 3.5.3) de Atmel pueden implementar una *caché* lógica para las configuraciones totales o parciales, de forma que se reduzca el tiempo de reconfiguración.

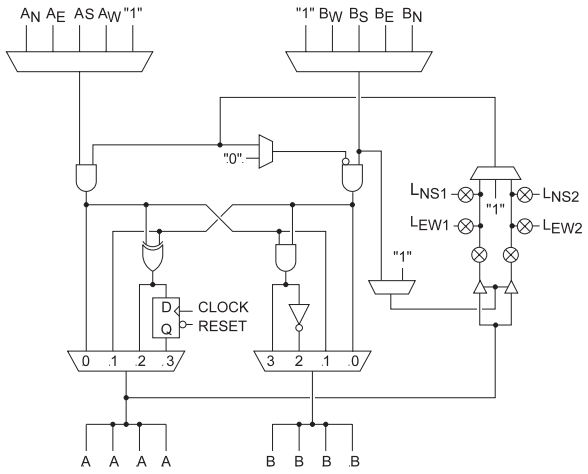


Figura 2.13.: Estructura de la celda de los dispositivos de la serie AT6000 de Atmel.

2.4. Resumen de los Sistemas Reconfigurables

En la tabla 2.1 se resumen las características más relevantes de los sistemas reconfigurables presentados en los apartados anteriores.

Se han distribuido en cuatro grupos de sistemas, aunque debido a la heterogeneidad de las diferentes propuestas no se puede hacer una separación muy estricta. Una característica diferenciadora es el tamaño de las unidades reconfigurables y su complejidad. Atendiendo a este aspecto se califica el apartado de *granularidad* como sistemas reconfigurables de grano fino y de grano grueso (sección 2.1).

En cuanto al campo *Tipo de reconfiguración*, se detalla si el sistema admite configuración dinámica, y en su caso alguna característica relevante de la misma (por ejemplo, si es parcial).

La información de la columna *Grado de acoplamiento con el host* indica el modo de conexión del sistema reconfigurable con el procesador. En el caso de que se trate de tarjetas acoplables a un bus de expansión estándar de un ordenador, el grado de acoplamiento se indentifica con el término de *remoto*. Si el procesador se sitúa en la misma placa compartiendo buses con el sistema reconfigurable, el término utilizado es *local*. Si el procesador está embebido en el dispositivo reconfigurable, se utiliza el término *Incluye el procesador*. Los sistemas en los que se indica *Es el procesador*, significa que el propio sistema reconfigurable es un microprocesador. Con la indicación *Puede incluir el procesador*, se identifica a los sistemas reconfigurables que tienen distintas versiones de dispositivos. Unas con procesadores *hard* embebidos en el propio circuito y otras con capacidad de admitir procesadores *soft* utilizando los recursos lógicos generales.

En la columna *Dominio de las Aplicaciones* se presentan el campo de aplicación o las aplicaciones más representativas para cada uno de los sistemas.

Tabla 2.1.: Resumen de los Sistemas Reconfigurables presentados.

Grupo	Denominación	Granularidad	Tipo de reconfiguración	Grado de acoplamiento con el *Host*	Dominio de las Aplicaciones
Aceleradores Hardware	**Splash&Splash2**	Grano fino	Estática con múltiples contextos	Remoto	Computaciones complejas a nivel de bit
	DECPeRLe-1	Grano fino	Estática con contexto único	Remoto	Computaciones complejas a nivel de bit
Coprocesadores Reconfigurables	**GARP**	Grano fino	Estática con múltiples contextos	Local	Procesamiento de imágenes a nivel de bit, criptografía (DES)
	OneChip	Grano fino	Estática con contexto único	Local	Acelerador de aplicaciones y controladores embebidos
	CHIMAERA	Grano fino	Estática con contexto único	Local	Computaciones a nivel de bit, criptografía (DES)
	DISC	Grano fino	Dinámica con contexto único	Es el procesador	Propósito general
	FIP	Grano fino	Estática o Dinámica según implementación	Es el procesador	Propósito general. Criptografía (AES)
CSoCs	**FIPSOC**	Grano grueso	Dinámica con dos contextos	Incluye el procesador	Microcontrolador de propósito general con módulos analógicos
	Pleiades/Maia	Diversa granularidad	Dinámica con contexto único	Incluye el procesador	Computaciones multimedia de alta velocidad y bajo consumo./ Procesador banda base *Wireless*
	DReAM	Grano grueso	Parcial dinámica	Local (Arquitectura integrable en CSoCs)	Aplicaciones telefonía 3G
	XPP	Grano grueso	Parcial dinámica	Local (Arquitectura integrable en CSoCs)	Aplicaciones de procesamiento digital de señal y multimedia
	Colt	Grano grueso	"Wormhole" (datos + configuración)	Local (Arquitectura integrable en CSoCs)	Aplicaciones de procesamiento digital de señal y multimedia
Dispositivos comerciales reconfigurables dinámicamente	**CS2000 RCP**	Grano grueso	Dinámica con dos contextos	Incluye el procesador	Comunicaciones (wireless, voz sobre IP, etc.)
	Triscend	Grano grueso	Dinámica con contexto único	Incluye el procesador	Comunicaciones y multimedia
FPGAs comerciales de grano fino reconfigurables dinámicamente	**Xilinx**	Grano fino	Parcial Dinámica	Puede incluir el procesador	Propósito general
	Atmel	Grano fino	Parcial Dinámica	Puede incluir el procesador	Propósito general

3. Sistemas Auto-Reconfigurables basados en cores sobre FPGAs

3.1. Definición formal de auto-reconfiguración

El concepto de reconfiguración dinámica, más general que el de auto-reconfiguración, se aplica a los dispositivos capaces de mantener su funcionamiento sin interrupción mientras varias partes de la matriz lógica están siendo configuradas [85]. Esta lógica se denomina lógica reconfigurable dinámicamente, diferenciándose de la lógica reconfigurable con la que se define a los dispositivos reconfigurables que no soportan un cambio de configuración en tiempo de funcionamiento.

P. Lysaght diferencia en [86] entre reconfiguración dinámica *intra-task* y reconfiguración dinámica *inter-task*. El primer tipo se refiere a tareas que únicamente se pueden realizar mediante reconfiguración dinámica, esto es, circuitos diseñados para uso de forma exclusiva con lógica reconfigurable dinámicamente. Por tanto, para modificar la tarea únicamente habrá que alterar una sección de ese circuito manteniendo la estructura del mismo.

La reconfiguración dinámica *inter-task* por contra, se aplica a los circuitos diseñados para realizar tareas que pueden funcionar en lógica no reconfigurable dinámicamente. Por tanto, para cambiar la tarea será necesario cambiar mediante reconfiguración el circuito completo. De esta forma, se pretende obtener beneficio de la lógica reconfigurable dinámicamente analizando qué tareas no requieren un funcionamiento simultáneo e intercambiándolas en el tiempo.

R. Sidhu y V.K. Prasanna definen en [18] el concepto de auto-reconfiguración como la capacidad de un dispositivo para generar en tiempo de funcionamiento los bits de configuración y usarlos para modificar su propia configuración. Sin embargo, habitualmente se utiliza el término de auto-reconfiguración de una manera más amplia. En la actualidad hay muy pocos sistemas capaces de generar un nuevo *bitstream* en tiempo de ejecución, si bien es más habitual que sean capaces de determinar internamente qué nuevos *bitstreams* parciales previamente generados, se deben cargar dinámicamente.

En un dispositivo auto-reconfigurable se distinguen dos secciones: Una, sobre la que se aplica la reconfiguración parcial dinámica, cuyos recursos lógicos están configurados para realizar las operaciones de la aplicación. En esta sección se realiza la *computación*; y otra, sección estática, con los recursos lógicos necesarios configurados para determinar la siguiente configuración aplicable. El conjunto de operaciones requeridas para obtener esa información es la *metacomputación*.

En función de esta separación se establece un criterio general para determinar la viabilidad de la auto-reconfiguración: *Se obtendrá una mejora de la eficiencia con la aplicación de la auto-reconfiguración cuando la reducción del coste de computación sea superior al coste de la metacomputación* [87]. Se resaltan como casos óptimos para aplicar este proceso aquellos en los que la modificación de la lógica de computación se basa en datos de entrada que son

conocidos de forma posterior a que comience la ejecución. En este caso la reconfiguración se denomina *dato-dependiente*.

En cuanto a los requisitos de la lógica auto-reconfigurable caben destacar los siguientes aspectos [87]:

- La *computación* deberá ser diseñada de tal forma que pueda beneficiarse de la auto-reconfiguración. Para ello la *metacomputación* deberá ser suficientemente flexible como para poder cubrir los requisitos de la modificación de la lógica.

- Debe proveer un sistema que permita desconectar de sus salidas la lógica que esté siendo modificada, o en su defecto, que pueda realizar todo el cambio de configuración en un único ciclo de reloj.

- La *metacomputación* deberá se ser suficientemente flexible como para poder ser utilizada en distintas aplicaciones de manera sencilla.

La auto-reconfiguración requiere una serie de elementos que hagan posible su aplicación y den soporte a la *metacomputación*. En este sentido, Shirazi, Luk y Cheung del Imperial College de Londres proponen en [88] una infraestructura teórica para la gestión de la reconfiguración dinámica en tiempo de ejecución en diseños reconfigurables. El gestor de reconfiguración representado en la figura 3.1 está formado por tres módulos: un monitor, un cargador y un elemento para el almacenamiento de la configuración. El monitor dispone de la información sobre el estado de configuración. Cuando se requiere un cambio de configuración, porque se ha recibido una solicitud, bien desde la aplicación o bien desde la FPGA, el monitor indicará al cargador que instale un nuevo circuito en una localización determinada de la FPGA. Se propone que el elemento almacén de configuraciones disponga de un directorio de configuraciones de los circuitos almacenados en forma de pares dirección-dato. De esta manera, el dato indicará la configuración para una celda de la FPGA y la dirección la posición dentro de la FPGA. A partir de este gestor genérico se pueden definir controladores específicos para una determinada aplicación (más optimizados) o bien de carácter más general (más complejos y menos eficientes). Una característica que puede hacer variar sustancialmente la complejidad del gestor es la operación del monitor que se podrá simplificar en casos donde se conozca en tiempo de compilación tanto la duración como la secuencia de las reconfiguraciones.

El flujo de diseño de Shirazi, Luk y Cheung para la generación de los controladores de reconfiguración está asistido por un conjunto de herramientas [89] desarrolladas por su grupo de investigación y soportado por un modelo de componente reconfigurable. El modelo propuesto por este grupo en [90] es un ejemplo representativo de abstracción teórica dentro del área de lógica reconfigurable. En la figura 3.2 se representa el modelo estático del componente dinámico. Éste podrá tener la funcionalidad A o B. La descripción se realiza mediante una red conectando A y B a través de dos bloques de control (*RC_DMux* y *RC_Mux*). La señal C, dependiente de la configuración, conectará los puertos reales X e Y a los bloques A o B. Este modelo no es dependiente de la tecnología puesto que los bloques *RC_DMux* y *RC_Mux* se podrán sintetizar en multiplexores y demultiplexores tanto reales como virtuales que simulen los mecanismos de reconfiguración.

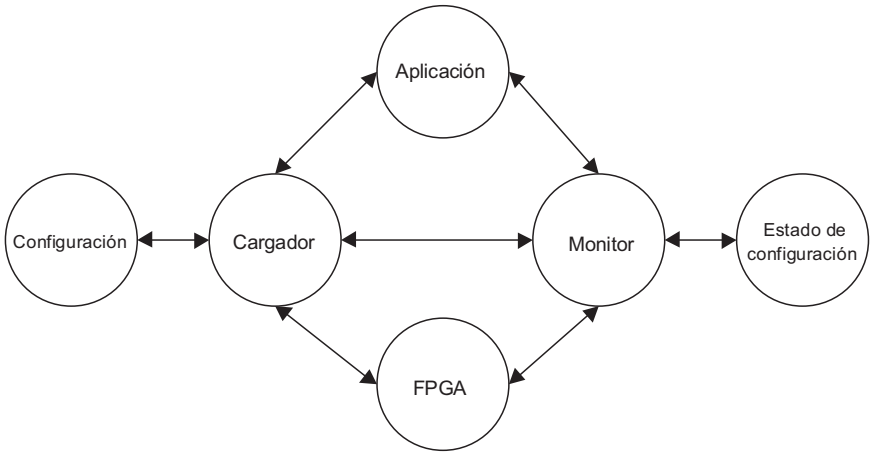


Figura 3.1.: Gestor de reconfiguración genérico de Shirazi et al.

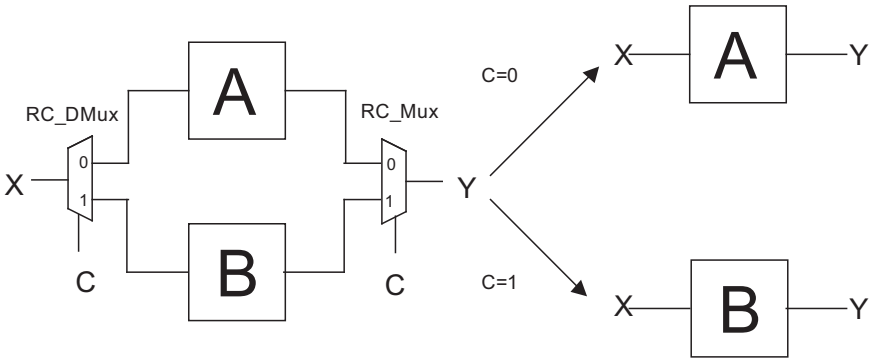


Figura 3.2.: Modelo de componente reconfigurable basado en una red estática.

3.2. Reconfiguración parcial dinámica aplicada a circuitos digitales

En esta sección se presenta el estado del arte relativo a la modificación dinámica de circuitos digitales específicamente diseñados para ser modificados mediante reconfiguración dinámica (reconfiguración *intra-task*) implementados en FPGAs de grano fino. Se abordan las metodologías y flujos de diseño más relevantes así como aplicaciones descritas en la bibliografía.

Para conocer la evolución de la modificación dinámica de circuitos implementados sobre lógica reconfigurable, que en la actualidad ya se aplica a la reconfiguración dinámica de *cores*, se considera relevante presentar los resultados más destacados de la investigación realizada en los últimos años sobre la modificación dinámica de circuitos. De forma especialmente intensiva en los años finales de la década de los 90 se ha trabajado en metodologías basadas en entornos desarrollados en lenguaje de programación Java. A continuación se presenta, entre otros, el sistema basado en Java (*Jbits*) [91] que Xilinx a proporcionado a los investigadores, con distintas evoluciones a lo largo de estos años. Con este sistema se han realizado gran parte de la investigación relacionada con la reconfiguración parcial dinámica aplicada a circuitos digitales en FPGAs de grano fino.

3.2.1. Modificación dinámica de circuitos mediante entornos JAVA

Uno de los elementos más extendidos en el flujo de diseño convencional con FPGAs es el uso de los circuitos pre-construidos agrupados en librerías. Inicialmente se trataba de circuitos básicos como los componentes de la familia 74XX. A medida que ha aumentado la capacidad de las FPGAs y por tanto la de los módulos que se pueden integrar, se ha evolucionado hacia módulos parametrizables [92, 93]. El diseñador introduce ciertos parámetros que permiten generar un circuito con las características deseadas, por tanto, optimizando el diseño en área y/o velocidad. Un ejemplo habitual es un contador binario, para el que se pueden definir parámetros como el número de bits, tipo de carga admisible, etc. Esta flexibilidad se limita a la fase de diseño del circuito (fase de compilación).

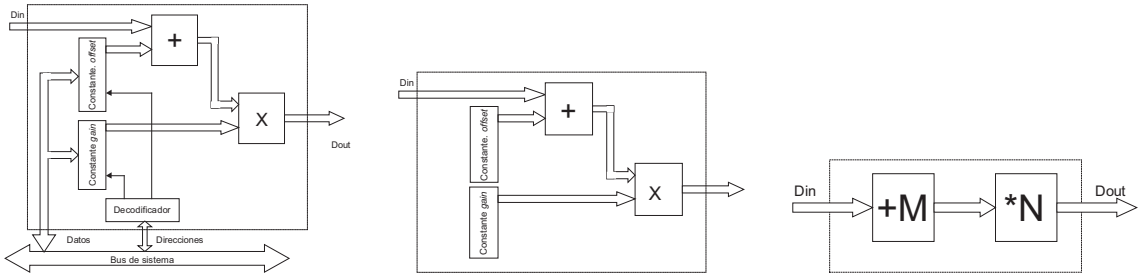
Sin embargo, con la lógica reconfigurable dinámicamente, surge un nuevo grado de libertad en el diseño: es posible modificar la lógica y el rutado de las FPGAs en tiempo real de forma que se puedan variar esos circuitos, dando lugar a los *Run-Time Parametrizable cores* (RTPs) [94].

S. A. Guccione y D. Levi en [1] ya resaltaban las ventajas de la aplicación de la reconfiguración parcial dinámica justificando su utilización en un circuito para aplicación de un *offset* (suma) y control de ganancia a una señal digital (multiplicación). Este circuito de tipo *datapath* corresponde con el conjunto de circuitos abordables de forma viable mediante un flujo de diseño basado únicamente en *Jbits*, ya que se pueden plantear diseños específicos para que se beneficien de una reconfiguración parcial dinámica *intra-task*. Con ello se consigue, por tanto, con la posibilidad de modificar la *computación* únicamente reconfigurando una pequeña sección del circuito.

En la figura 3.3 se representan tres implementaciones del circuito. En el caso de la implementación para ASIC (figura 3.3(a)) se busca un diseño muy flexible, con registros para el cambio de las constantes de multiplicación y suma (con toda la lógica adicional que esto supone: líneas de dirección, comparadores, etc.).

Si se usa un dispositivo reconfigurable (figura 3.3(b)), se puede modificar el valor de las constantes durante el diseño reduciendo la lógica y la complejidad del mismo. Sin embargo, para poder cubrir toda su funcionalidad, se debe poder modificar el valor de las constantes en tiempo real. Si el dispositivo reconfigurable admite reconfiguración dinámica, es posible la modificación del valor de esos parámetros, eliminándose toda la lógica adicional necesaria que se requería en la implementación ASIC para acceder a los registros en el mapa de memoria del sistema (aunque se deberá tener en cuenta el coste generado por la lógica para la *metacomputación* que surgiría en ese caso).

La figura 3.3(c) muestra una implementación todavía más reducida y simple que se puede abordar mediante reconfiguración dinámica. En vez de usar un multiplicador y un sumador de propósito general y registros para las constantes, se propone el uso de aritmética de coeficientes constantes. Las unidades se construirían en tiempo de ejecución dependiendo de la variable que se desease usar.



(a) *Implementación para ASIC.*

(b) *Implementación para FPGA reconfigurable dinámicamente.*

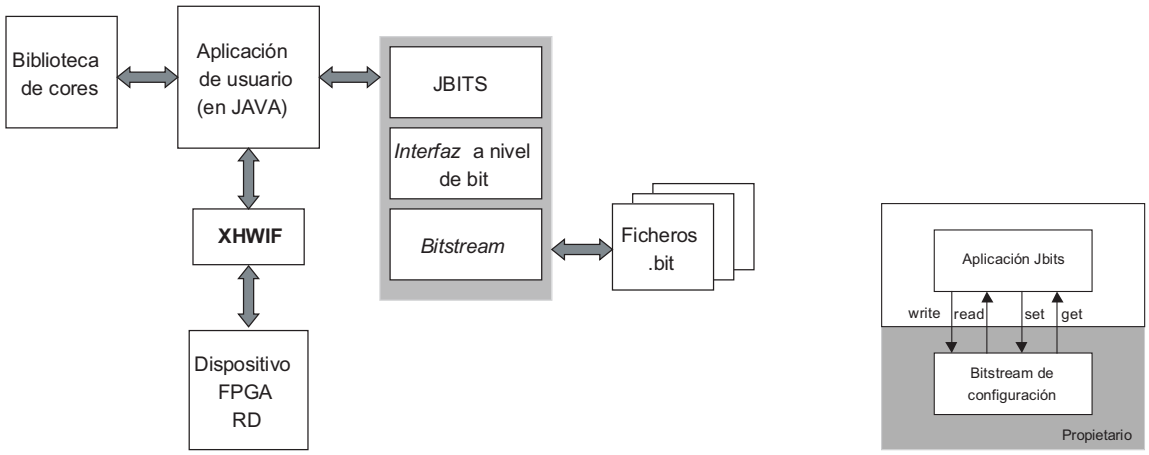
(c) *Modificación del circuito para Aritmética de Coeficientes Constantes.*

Figura 3.3.: Circuito digital para la aplicación de *offset* y control de ganancia a una señal digital [1].

El entorno Java *Jbits* da soporte a la reconfiguración en tiempo de funcionamiento (*Run-Time-Reconfiguration* -RTR-) a varios dispositivos reconfigurables dinámicamente de Xilinx (figura 3.4). De esta forma se pueden abordar diseños con *cores* parametrizables en tiempo de funcionamiento (los *Run-Time Parametrizable Cores* previamente definidos). Mediante una interfaz específica (*Application Program Interface* -API-), *Jbits* posibilita el acceso a los datos de configuración *(bitstream)* del dispositivo. Esta facilidad hace posible la modificación en tiempo de funcionamiento tanto de la lógica como del rutado. Actualmente *Jbits* soporta la familia Virtex de Xilinx, siendo una evolución del sistema JERC6K [95] desarrollado para los ya obsoletos dispositivos XC6200 [39] y posteriormente ampliado a la familia XC4000.

Jbits usa un modelo del dispositivo reconfigurable basado en una matriz de CLBs. Los circuitos en general y los *Run-Time Parametrizable Cores* en particular ocupan una zona definida en dos dimensiones de esa matriz. *Jbits* es capaz de usar esas secciones de forma dinámica como objetos Java. Puede replicarlos, implementarlos condicionalmente, etc. Los detalles del sistema *XBI*, nombre de desarrollo interno de *Jbits*, se exponen en [96].

Desde *Jbits* es abordable la construcción e instanciación de *Run-Time Parametrizable Cores*. Para cada conjunto de CLBs se asigna un grupo de elementos de rutado para realizar las conexiones. Pero la labor de interconectar esos *cores* de forma dinámica no estaba resuelta



(a) ***Sistema de reconfiguración parcial dinámica basado en el sistema Jbits.*** **(b)** ***El API de Jbits.***

Figura 3.4.: Entorno JAVA para soporte de la reconfiguración dinámica.

en las primeras versiones de *JBits*. De hecho, el conexionado es uno de los elementos más complejos que se abordan en el área de la reconfiguración parcial dinámica. Las herramientas que proveen los fabricantes de dispositivos reconfigurables para emplazamiento y rutado ofrecen mecanismos para especificar la localización de los elementos lógicos de la matriz reconfigurable. Sin embargo, no proveen de métodos para guiar el rutado de forma que una determinada ruta que conecte una sección fija con una sección dinámica siempre utilice la misma línea en cualquiera de los módulos configurables en esa sección.

Para solucionar esa problemática, investigaciones como las de Breber y Donlin [97] proponen modelos de conexión para unidades reconfigurables orientadas a distintos tipos de computación (paralelo y secuencial con procesador) o, de forma más general, *cores* para rutado como los presentados por Guccione en [94]. Estos *cores* especiales denominados *Sitcher cores* son *Run-Time Parametrizable Cores* sin lógica, únicamente con recursos de rutado. De esta forma se abstrae el conexionado real del dispositivo. Soluciones similares se plantean en años posteriores, como las *Bus-Macro*[1] de Xilinx [98], para interconectar *cores* dinámicamente. Actualmente el *Jbits SDK* [91] incluye una herramienta de rutado automático capaz de conectar y desconectar conexiones de forma dinámica, aunque con limitaciones como la imposibilidad de utilizar líneas largas.

El sistema *Jbits* inicialmente ha estado restringido por la falta de mecanismos para el diseño de diferentes tipos de circuitos, especialmente circuitos de control, máquinas de estado, etc. [99]. El flujo de diseño original mediante *Jbits* se ha ido modificando con el fin de integrar diseños de circuitos desarrollados mediante herramientas de síntesis convencionales basadas en lenguajes de descripción hardware. De esta forma se ampliaban los tipos de circuitos que se podían diseñar con soporte para reconfiguración dinámica. Roxby y Guccione presentaban en [100] una herramienta para *Jbits* denominada *JbitsDIFF* capaz de extraer de *bitstreams* de configuración *cores* desarrollados mediante otro flujo de diseño

[1] Aunque este término originalmente es de la terminología de Xilinx, cada vez con mayor frecuencia se emplea para denominar en general a los circuitos con esta función.

(por ejemplo, en VHDL). Tras la extracción, estas secciones de *bitstream* son convertidas en *cores Jbits*, pudiéndose utilizar posteriormente para configurar dinámicamente el dispositivo reconfigurable. La metodología propuesta se aplica, a modo de ejemplo, a un circuito para multiplicación de una variable por una constante (*Constant Coefficient Multiplier*) que había sido descrito originariamente en VHDL [101].

Cabe destacar también otros circuitos reconfigurables dinámicamente sobre los dispositivos XC6200 mediante *Jbits* propuestos en otros trabajos, como el multiplicador en base $GF(2^k)$ (*Galois Field* [102]) presentado por R. Payne en [103]. En esta aplicación se profundiza sobre la problemática del cumplimiento de tiempos en los circuitos generados dinámicamente. P. Athanas y A. Abbott se centran en aplicaciones de procesamiento de imágenes en [104] y A. Dandalis y V.K. Prasana en algoritmos de procesamiento de señal en [105].

El flujo de diseño de *Jbits* también se ha visto modificado por la incorporación de otras herramientas, como el programa *JRTR* [106], que dan soporte a la capacidad de reconfiguración parcial dinámica de los dispositivos Virtex.

Uno de los mayores inconvenientes de la reconfiguración dinámica es el tiempo necesario para la reconfiguración que hace inviable en muchas aplicaciones la utilización de la misma [33, 44]. Por ello, la modificación parcial dinámica supone uno de los avances más prometedores en el área de la Computación Reconfigurable.

3.2.2. Modificación dinámica de circuitos mediante la manipulación directa del bitstream

La ventaja más destacable de las metodologías que se basan en la manipulación directa del *bitstream* es que éste puede provenir de herramientas estándar para diseño y síntesis de FPGAs. Estas herramientas están más preparadas para optimizar los circuitos y son preferidas en general por los diseñadores ya que el diseño inicialmente puede partir de descripciones realizadas en lenguajes HDL, siendo ésta la forma de trabajo convencional. La base de estas metodologías consiste en la extracción de la sección del circuito dinámico del *bitstream* global de configuración. Con la parte extraída se compone un *bitstream* parcial.

Los ejemplos más significativos de estas metodologías son los dos flujos de diseño para reconfiguración parcial dinámica propuestos por Xilinx en [98]: *Small Bit Manipulations* y *Module Based*. Se basan en la utilización de las herramientas que provee el propio fabricante para realizar diseños convencionales: sintetizador, programas para emplazamiento y rutado y herramientas adicionales, todas ellas incluidas en el entorno de diseño de Xilinx.

El flujo de diseño denominado *Small Bit Manipulations* está orientado a la reconfiguración *intra-task* de circuitos. En estas situaciones, tal y como se ha presentado anteriormente, únicamente se requiere cambiar una pequeña sección del hardware para modificar la funcionalidad, ya que el circuito está específicamente diseñado para ser modificado mediante reconfiguración. Por ello, en este flujo de diseño se propone modificar directamente en el fichero que agrupa la información de emplazamiento y rutado los recursos lógicos (CLBs, bloques de memoria RAM dedicados, etc.) que varían entre los distintos contextos. Esta modificación se realiza con la ayuda del programa *FPGA Editor* que muestra todos los elementos configurables (lógica y rutas) de la FPGA de forma gráfica. Para cada contexto se obtiene un fichero modificado con pequeñas variaciones respecto al original. El programa

que genera los *bitstreams* de configuración es capaz de generar *bitstreams* parciales únicamente con los cambios que se deben aplicar para configurar los diferentes contextos en base a las diferencias entre los ficheros modificados y el original.

Cuando las modificaciones que se deben realizar mediante la reconfiguración parcial dinámica son importantes, por ejemplo, en un caso de reconfiguración *inter-task*, Xilinx propone el segundo flujo de diseño denominado *Module Based*. En este caso, aunque las herramientas son las mismas, la forma de obtener los *bitstreams* es muy diferente. Es necesario realizar el emplazamiento y rutado de cada sección (dinámica y estáticas) de la FPGA por separado y obtener los correspondientes *bitstreams* parciales, además de un *bitstream* total para la carga inicial.

Para mantener la integridad de las conexiones entre la sección estática y las dinámicas (y la de todos los módulos dinámicos que puedan ser cargados en una sección dinámica) se requiere cumplir unas pautas muy estrictas en el flujo de diseño, que incluyen la utilización de elementos pre-rutados denominados *Bus-Macro*, cuidar la distribución de los *pines* de salida y tener en cuenta las limitaciones impuestas a las señales globales, entre otras.

3.3. Simulación de la reconfiguración parcial dinámica de circuitos

3.3.1. Simulación de circuitos reconfigurables y nuevas metodologías de diseño

La simulación de los circuitos electrónicos actualmente es una de las etapas más importantes en el diseño de los mismos. La complejidad de los sistemas analógicos y digitales, junto con los reducidos tiempos de diseño, imponen la necesidad de realizar simulaciones exhaustivas de éstos antes de ser probados sobre prototipos. Incluso el diseño con dispositivos reconfigurables que permiten probar distintas soluciones con el mismo dispositivo, requiere simulaciones que posibiliten la comprobación de la correcta funcionalidad del sistema y el cumplimiento de tiempos.

Las herramientas CAD para diseño con FPGAs integran simuladores de lenguaje de descripción hardware (VHDL y Verilog principalmente) que permiten realizar simulaciones empleando modelos con diferente nivel de detalle [107]. Estos modelos están asociados con una fase, más o menos avanzada, del diseño. Normalmente suelen ser cuatro: *Behavioral* VHDL *model*, *Post-Translate* VHDL model, *Post-Map* VHDL model y *Post-Place & Route* VHDL model. A medida que se pasa de la primera (*Behavioral* o comportamental) a la última (*Post-Place & Route*), la descripción del circuito es más compleja y los tiempos de simulación mayores pero, a cambio, los resultados de simulación van incluyendo más información temporal y son más precisos.

Respecto a la simulación de diseños digitales estáticos, también hay que destacar cómo en los últimos años han proliferado las alternativas basadas en metodologías descendentes basadas en la utilización de modelos con distintos niveles de abstracción. Estos modelos son descripciones con un alto nivel de abstracción que parten generalmente desde una especificación ejecutable, pasando por modelos funcionales no sintetizables, hasta detallar el

circuito a nivel de transferencia entre registros (*Register-Transfer-Level* -RTL-). El complejo diseño de SoCs requiere, además de la evaluación de funcionalidades, estudiar cómo realizar la distintas particiones de tareas en elementos hardware y software, y su evaluación mediante co-simulación [108, 109]. Un nivel de descripción tan detallado como el RTL [110] para las entidades hardware dificulta y ralentiza en gran medida las simulaciones de ese tipo de sistemas. Por ello, estas metodologías parten de modelos con descripciones funcionales (simples), que permiten rápidas simulaciones que facilitan la definición de la arquitectura y las particiones entre hardware y software. Algunas de estas metodologías emplean lenguajes de descripción hardware como la propuesta Polis [111, 112] basada en VHDL. Otras, utilizan lenguajes de programación, como el SpecC [113] basado en un *super-set* del ANSI-C con extensiones para descripción hardware, o el SystemC basado en C++.

SystemC destaca como uno de los sistemas con más éxito para diseño con metodología descendente. Parte de este éxito se puede justificar por su desarrollo coordinado por el grupo *Open SystemC Initiative (OSCI) Language Working Group* [114, 115] que ha agrupado en una librería y un *Kernel* el SystemC, por lo que son compilables por herramientas genéricas de C++. Este grupo está patrocinado por un consorcio de numerosas empresas (más de 50) que incluyen a Cadence, Synopsis, Texas Instruments, Motorola, Nec, Frontier, Coware, etc. Aunque SystemC se utiliza en trabajos de investigación y experimentación como los reflejados en [116, 117, 118, 119], ya es una realidad en el diseño de sistemas comerciales. Además de un sistema de desarrollo libre, hay numerosas herramientas comerciales que soportan *SystemC* (SystemC de Synopsys Inc., ModelSim de Mentor Graphics, etc.).

3.3.2. Simulación de circuitos parcial y dinámicamente reconfigurables

En el apartado anterior se ha puesto de manifiesto la importancia de la simulación en el diseño electrónico actual. De igual forma, en el diseño de sistemas con circuitos reconfigurables dinámica y parcialmente la simulación tiene una gran importancia. Sin embargo, las características inherentes a los circuitos reconfigurables dinámicamente (por ejemplo, la posibilidad de permutación de los mismos), no están soportadas por los sistemas de simulación para circuitos convencionales. Básicamente, porque los lenguajes de descripción hardware no disponen de construcciones adecuadas para este tipo de diseños.

La Computación Reconfigurable es una disciplina emergente, por lo que muchas de las herramientas que la soportan se encuentran en fase de desarrollo. Esta situación se pone de manifiesto especialmente en el apartado de la simulación. Aunque a continuación se presentará el estado del arte en esta materia, es significativo que la nota de aplicación más reciente [98] del fabricante Xilinx relativa a flujos de diseño para sistemas reconfigurables parcial y dinámicamente proponga, como método de simulación, la evaluación estática de las diferentes combinaciones que se pueden componer en la FPGA entre la sección estática y cada uno de los módulos que se pueden cargar de forma dinámica. Es decir, que realmente el proceso de reconfiguración no se puede simular con las herramientas integradas en el entorno de diseño que provee el fabricante.

Las técnicas propuestas para la simulación de lógica reconfigurable dinámicamente utilizan tanto metodologías descendentes *(top-down)* como ascendentes *(bottom-up)*. Como propuestas ascendentes destacan las de Kwiat [120] y Faura [121]. Proponen la simulación de la reconfiguración dinámica mediante la aplicación de la configuración a un modelo deta-

llado en VHDL de la FPGA (lógica, rutas y memoria de configuración). Aunque la propuesta permitiera unas simulaciones muy detalladas, según el análisis realizado en [122] es de difícil aplicación debido a su falta de generalidad y complejidad, ya que es necesario disponer de los *bitstreams* de configuración antes de la simulación.

Una propuesta descendente, cuya técnica de *clock-morphing* ha sido utilizada posteriormente por otros investigadores [123], es la realizada por Vasilko y Cabanis en [124]. Mediante la modificación del *package* de VHDL `std_logic_1164` se introduce un nuevo estado `'V'` posible para una determinada señal. Este nuevo valor representa el estado inactivo. Cuando se desactiva un circuito reconfigurable dinámicamente, se introduce el valor `'V'` en su línea de reloj propagándose a todas las señales del circuito reconfigurable que se desactiva (sistema válido únicamente para diseños síncronos). De esta forma se elimina el circuito del conjunto activo. Los autores resaltan dos problemas detectados con este sistema: el primero es debido al propio retardo generado por la propagación de `'V'` y el segundo está relacionado con la dificultad de conservar la propagación de `'V'` dentro de los límites del circuito que se reconfigura parcialmente.

También el modelo de Luk [90] basado en multiplexores y demultiplexores presentado en el apartado 3.1 se plantea como una alternativa para la simulación. Sin embargo, debido a su alto nivel de abstracción, no permite estudiar el intervalo de reconfiguración, por lo que es una técnica muy limitada para la simulación.

La investigación más intensiva en los últimos años sobre la simulación de lógica reconfigurable dinámicamente se sitúa en la Universidad de Strathclyde (UK). En 1996 P. Lysaght y J. Stockwood [125] presentaban la técnica *Dynamic Circuit Switching* (DCS) completada por D. Robinson [122] con la aplicación del VHDL. Esta técnica introduce en el diseño original elementos, denominados *Virtual Components*, que permiten la simulación de circuitos dinámicamente reconfigurables con herramientas de simulación convencionales.

El conjunto de *Virtual Components* está compuesto por detectores de condición de la configuración (*Reconfiguration Condition Detectors* -RCDs-), módulos controladores (*Schedule Control Modules* -SCMs-), registros de estado de las tareas (*Task Status Registers* -TSR-) [126] y conmutadores de tareas *(Isolation Switches)*. En la figura 3.5 se representan estos elementos. Las tareas (circuitos) A y D son estáticas mientras que las tareas B y C pueden intercambiarse dinámicamente. Los RCDs detectan la señal que indica el cambio de tareas. Estas peticiones pasan a los controladores SCMs que en conjunción con los registros que indican el estado de las mismas (TSRs) controlan los conmutadores de tareas (*Isolation Switches*). Estos conmutadores de tareas son la base de la técnica *Dynamic Circuit Switching* (DCS) [123]. Cuando una tarea se encuentra desactivada, los *Isolation Switches* que conectan el módulo de esa tarea al sistema ponen su salida en un estado de alta impedancia `'Z'`, que es el valor más débil del *package* `std_logic`. En este *package* se definen en VHDL los estados posibles de las señales. Cuando se simula el estado de reconfiguración, la salida de los conmutadores de tareas del módulo que se está reconfigurando pasan a tener el valor `'X'`, que es la representación del estado *desconocido* en el *package* `std_logic`. En esta situación, los circuitos desactivados, al disponer en sus conmutadores de tareas el valor más débil del *package* `std_logic` (`'Z'`), no afectan a la propagación del valor `'X'` o a los valores fijados por los circuitos que se mantienen activos. Mediante este sistema de simulación se puede comprobar hasta donde afecta en el diseño la indeterminación producida por el proceso de

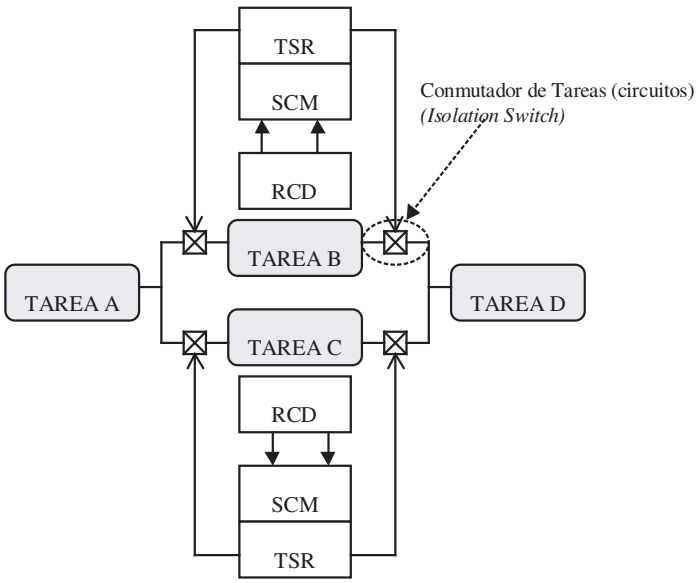


Figura 3.5.: Dispositivos virtuales integrados en el diseño para permitir la simulación de lógica reconfigurable dinámicamente. Técnica *Dynamic Circuit Switching (DCS)*.

reconfiguración parcial de una zona del circuito.

Los parámetros de los que dispone cada circuito reconfigurable son básicamente: la condición que causa su activación y desactivación más los tiempos requeridos para realizar la carga en la FPGA y la desactivación segura previa a la carga de una tarea (circuito) distinta.

En el año 2002, este grupo publicó la extensión de la técnica DCS [127] a los dispositivos Virtex reconfigurables parcial y dinámicamente de Xilinx. En ese trabajo se presenta una herramienta avanzada denominada DCSTech, integrada dentro del propio flujo de diseño de Xilinx, que permite la simulación una vez generados los *bitstreams* finales.

3.4. Reconfiguración parcial dinámica de IP-Cores

Mediante los apartados anteriores 3.2.1 y 3.2.2 se han presentado los trabajos relativos a técnicas de modificación y reconfiguración dinámica de circuitos digitales. Sin embargo, la integración que permite la actual tecnología ha dirigido la ingeniería de sistemas digitales hacia un diseño basado en módulos o *cores* [128] reutilizables (bien comprados a una compañía externa o propios) sobre un único integrado. Este tipo de circuitos se denominan comúnmente *Intellectual Proprietary Cores* (IP-Cores). Este concepto, surgido inicialmente en el entorno del diseño de ASICs, se ha extendido a las FPGAs cuando éstas han incrementado su capacidad de recursos lógicos y de rutado hasta tal punto que admiten la integración de sistemas digitales en un único dispositivo. Estos módulos complejos pueden integrar no solamente un único tipo de circuito, sino que frecuentemente están compuestos por una sección de control y otra de computación, incluyendo incluso procesadores [129].

Los *cores* desarrollados para ASICs disponen de registros internos de configuración para poder variar su funcionalidad, de igual forma que los circuitos básicos para implementa-

ción sobre ASICs (véase sección 3.2). De esta forma se dota de flexibilidad a los módulos, pudiendo cubrir, por tanto, diversas aplicaciones.

Si bien para el diseño de circuitos integrados específicos esta característica es claramente beneficiosa y no supone un sobrecoste destacable, cuando se implementan estos *cores* en FPGAs surge un doble nivel de programación. Se considera redundante el hecho de que la FPGA pueda modificar el diseño interno vía reconfiguración y que los *cores* dispongan de varios modos de funcionamiento seleccionables mediante registros (por tanto, con secciones del circuito no activas).

Si se tiene en cuenta que el costo estimado de la implementación de un IP-Core en una FPGA en términos de área de silicio y de consumo es de 10 a 20 veces superior [130] respecto a la implementación del mismo en ASIC, buscar soluciones a esta redundancia supone una mejora importante en la eficiencia de los diseños implementados con FPGAs.

En el apartado 3.2.1 se ha presentado la mejora en cantidad de recursos lógicos requeridos y en velocidad obtenida con la utilización de los *Run-Time Parametrizable cores* [94], en sustitución de las descripciones para ASIC, en los dispositivos con capacidad de reconfiguración dinámica.

Basándose en el mismo concepto, MacBeth y Lysaght, investigadores de la Universidad de Strathclycle, han abordado la reconfiguración aplicada a IP-Cores. Para ello han identificado un conjunto de *cores* susceptibles de ser beneficiarios de la reconfiguración dinámica. Denominan a esta clase de circuitos *Programmable Multifunction Cores* (PMCs) [131]. La figura 3.6 representa un modelo genérico para este tipo de *cores*.

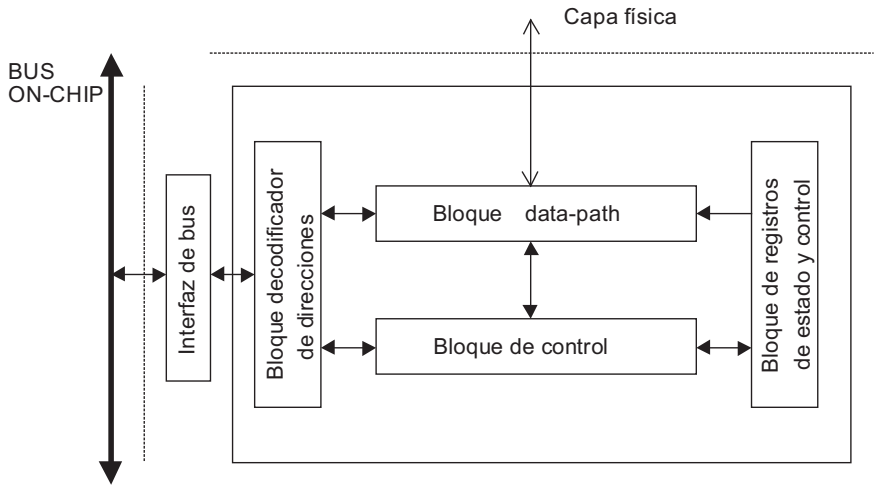


Figura 3.6.: Modelo de los *Programmable Multi-function Cores* (PMCs).

Los *Programmable Multifunction Cores* disponen de secciones de computación y control además de una interfaz adecuada para la conexión al bus *on-chip* del SoPC. Se incluyen en esta categoría UARTs, controladores PCI, controladores de vídeo o controladores USB. Este conjunto de circuitos sufren la redundancia anteriormente planteada. Mediante sus registros de control realmente se selecciona un circuito particular dentro de un conjunto definido para ese *Programmable Multifunction Core*.

Aunque estas implementaciones, tal y como se ha indicado previamente, son adecuadas

para ASICs debido a la flexibilidad que aportan, se quiere evitar la duplicidad de reconfiguración cuando se aplican a FPGAs mediante la reconfiguración dinámica. En [132] MacBeth propone una metodología de diseño para los *Programmable Multifunction Cores* en VHDL. En esta metodología se parte identificando los registros de control del *Programmable Multifunction Core*, creándose un tipo enumerado para cada modo de funcionamiento. Para cada modo se realiza una síntesis, cableando los registros de control a una configuración determinada. De esta forma, la herramienta de síntesis es capaz de optimizar la implementación mediante la propagación de constantes en el diseño [133]. Del conjunto de ficheros de configuración realizados con las funciones requeridas por el diseñador se localiza el del circuito que requiere mayor área, denominándose *worst-case*. Éste servirá para determinar los límites requeridos para la implementación de todos los casos.

El flujo de diseño propuesto en esta metodología se concluye generando ficheros de configuración o *bitstreams* parciales. En estos ficheros se incluye únicamente los cambios en la configuración relativos a lo que está operativo en el dispositivo. De esta forma, se dispone de un conjunto de *cores* reconfigurables dinámicamente relocalizables en la sección de área asignada al *worst-case*.

Los autores validan esta metodología con la aplicación de ésta a una UART, obteniendo reducciones en área en el entorno del 20 %. En [131] dotan a esta metodología de una sencilla formulación matemática que permite comparar la utilización de recursos y la máxima velocidad de funcionamiento para distintas implementaciones de estos *cores* reconfigurables dinámicamente, denominados *Dynamically Reconfigurable IP cores* (DRIP) [133].

En este apartado relativo a los *cores* reconfigurables dinámicamente y como puente a la siguiente sección centrada en la reconfiguración parcial dinámica en plataformas, cabe destacar los trabajos de Dyer, Pless y Platzner [99, 134] del Instituto Federal Suizo de Tecnología. Usando de forma mixta las técnicas actualmente disponibles para la modificación parcial de circuitos (apartados 3.2.1 y 3.2.2) se define un flujo de diseño verificado mediante una aplicación real sobre dispositivos Virtex [134].

Esta aplicación consiste en un sistema de decodificación de audio realizado mediante una FPGA reconfigurable dinámicamente. Se distinguen en esta FPGA una sección fija y otra sección modificable de forma dinámica. En la sección estática se localiza el microprocesador Leon [67] (*core soft*). Este microprocesador recibe la secuencia (*stream*) de audio a través de la red local, realizándose la tarea de decodificación del *stream* en la sección dinámica mediante un *core* coprocesador. Este módulo es reconfigurado dinámicamente según el formato de audio empleado.

El flujo de diseño propuesto en esta aplicación es mixto. Por un lado, se beneficia del nivel de abstracción utilizado en *Jbits* (CLBs, rutas, etc.). Este nivel es mayor comparado con el disponible en los sistemas de manipulación directa de los bits de un *bitstream*. Además *Jbits* permite una manipulación de los *bitstreams* más comprensible [134]. Y por otro lado, evita las restricciones de un flujo de diseño únicamente basado en *Jbits*, que son básicamente unas capacidades de síntesis, emplazamiento y rutado limitadas. Para ello utilizan también el flujo de diseño basado en la manipulación directa de *bitstreams* que se basa en herramientas convencionales HDL (sección 3.2.2).

El la figura 3.7 se resume este flujo de diseño mixto aplicado por Dyer [99]. Los *bitstreams* totales (figuras 3.7(a) y 3.7(b)) se crean mediante las herramientas convencionales. Utilizando la herramienta *Jbitscopy* se extrae el *core* coprocesador de 3.7(b) (figura 3.7(d))

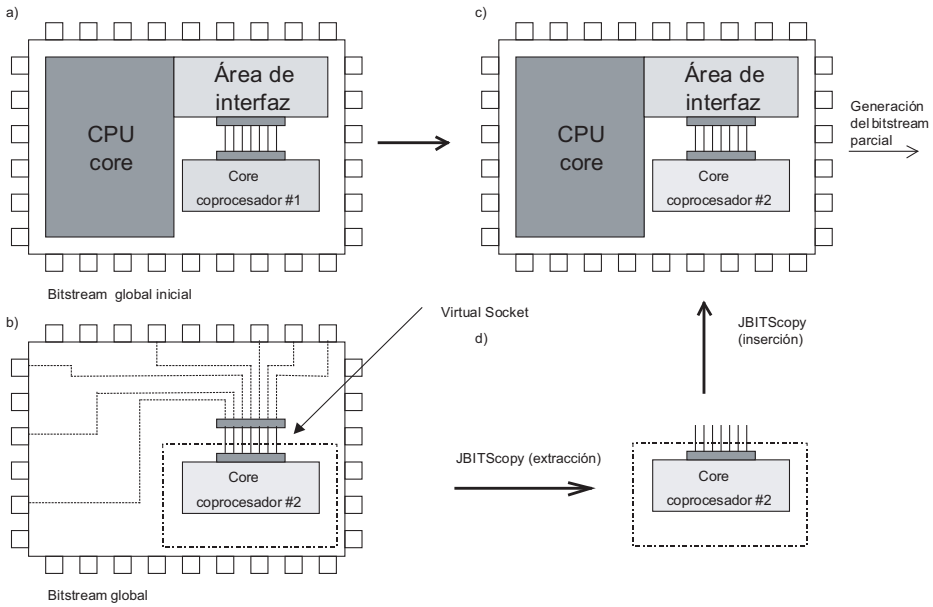


Figura 3.7.: Aplicación del flujo de diseño mixto de Dyer a *cores* reconfigurables dinámicamente.

integrándose en el *bitstream* completo original (figura 3.7(c)). A partir de esta configuración, *Jbits* puede generar un *bitstream* parcial aplicable dinámicamente con el área del coprocesador. Puesto que actualmente no es posible restringir los recursos de rutado en los dispositivos de Xilinx, la conexiones entre las secciones estáticas y dinámicas se resuelven mediante interfaces localizadas en posiciones fijas *(Virtual Sockets)*. Para el caso de rutas que se deban mantener en la zona dinámica, se propone el uso de *Bus-Macros* denominadas *feed-through*. Cada una de estas *Bus-Macro* consiste en dos CLBs sin operación lógica definida, pero la herramienta de emplazamiento y rutado permite establecer una posición específica para la misma en la matriz lógica de la FPGA. De esta forma se dispone de un cierto control del rutado, pudiéndolo guiar a secciones menos problemáticas.

De forma paralela a estas investigaciones del Instituto Suizo de Tecnología, cabe destacar los avances del estudio en el área de la reconfiguración dinámica de módulos hardware para comunicaciones realizados por el Departamento de Ciencias de Computación de la Universidad de Washington. Los módulos experimentales desarrollados se han aplicado a la plataforma *Field Programmable Port eXtender* (FPX) [135, 136] orientada al diseño rápido de prototipos de *routers* [137] y *firewalls* [138].

Estos módulos son *cores* reconfigurables dinámicamente y se denominan *Dynamic Hardware Plugins* (DHPs) [139]. Su aplicación es la del procesamiento de paquetes para Internet de alta velocidad. Las investigaciones realizadas incluyen el desarrollo de herramientas y metodología para soportar la reconfiguración parcial dinámica de los DHPs sobre la plataforma FPX, de forma que se puedan variar la funciones de procesamiento de paquetes a través de nueva información que se reciba a través de la red [135].

Al igual que en la metodología de Dyer, la configuración de la zona de la matriz lógica que contiene el DHP se extrae del *bitstream* global mediante una herramienta específica.

La herramienta que permite realizar esta operación es un programa para ordenadores PC denominado PARBIT [140]. Éste requiere como parámetros de entrada las coordenadas del bloque que se quiere extraer del *bitstream* origen y las coordenadas destino donde se desea incorporar dinámicamente. PARBIT genera un *bitstream* parcial con ese bloque. Se incluyen además los bits de configuración necesarios para completar las columnas donde se sitúa el bloque configurable, puesto que la tecnología de Virtex impone la configuración en unidades verticales (columnas) completas.

Para solucionar la problemática del rutado, de forma similar a los *Virtual Interfaces* referenciados anteriormente, se propone el uso de unos puntos fijos de interconexión para conectar los DHPs a la infraestructura lógica de la FPGA denominados Gaskets [141]. Los Gaskets tienen la misma localización para cualquier DHP. De esta forma, se mantiene el mismo rutado para la interconexión con la sección estática del diseño cuando se realiza el intercambio. Éstas rutas fijas se denominan antenas Gasket. La figura 3.8 representa la zona de rutado libre definida dentro del área del DHP y una zona de exclusión de rutado para ese módulo fuera de los bordes definidos por las antenas Gasket.

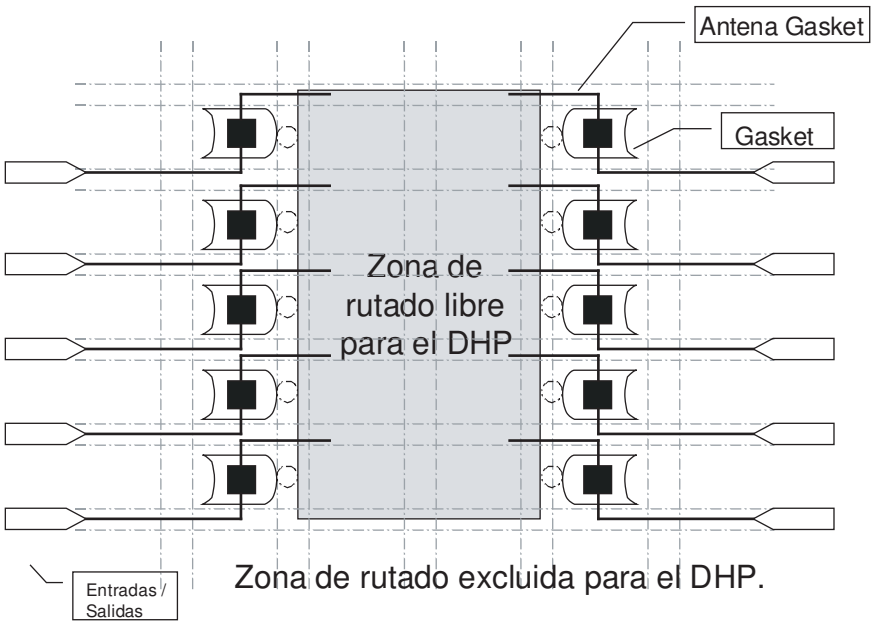


Figura 3.8.: Antenas Gasket para la interconexión de DHPs a la zona estática del sistema.

3.5. Reconfiguración parcial dinámica aplicada a plataformas FPGA basadas en IP-Cores

3.5.1. Las primeras propuestas

La aplicación de reconfiguración parcial dinámica a sistemas basados en *cores* en la década de los 90 se realizaba principalmente utilizando los dispositivos reconfigurables parcial y dinámicamente XC6200 [39] de Xilinx.

Empleando estos dispositivos A. Donlin presenta en [142] la arquitectura Flexible URISC. Se propone el uso del procesador Ultimate-RISC [143] como controlador de un sistema co-procesador FPGA dinámicamente reconfigurable. El URISC es una arquitectura mínima de procesamiento con una instrucción básica: *Mover de memoria a memoria*. La computación se realiza mediante el movimiento de los datos entre los módulos distribuidos en el bus del sistema. Los módulos de procesamiento se denominan *Swappable Logic Units* (SLU) y sus registros están *mapeados* en la memoria del *core* URISC. En esta arquitectura se propone una reconfiguración *inter-task* de los módulos *Swappable Logic Units* de forma que se puedan cargar distintos procesamientos. En el mapa de memoria del sistema se incluye, además de la memoria de datos y de programa, la memoria de configuración de la FPGA. Esta última característica dota a esta arquitectura de la capacidad de auto-reconfiguración.

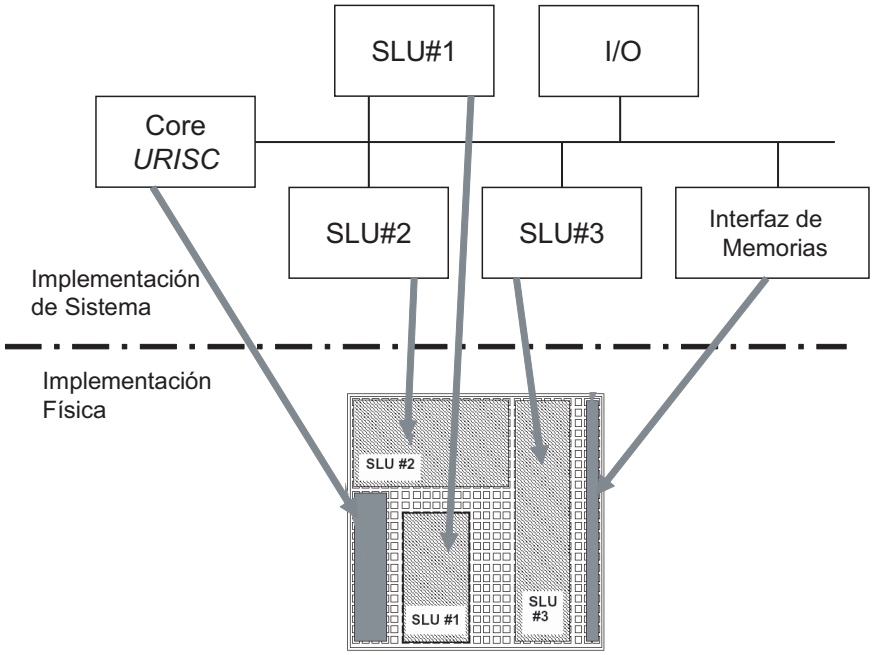


Figura 3.9.: Arquitectura de la plataforma *Flexible* URISC.

McGregor y Lysaght proponen en [85] un sistema de auto-reconfiguración (*Self controlling DRL system*) implementado, al igual que la plataforma anterior, sobre los dispositivos ya obsoletos XC6200 de Xilinx. Aunque propiamente el sistema propuesto no está generalizado para diseños basados en *cores*, ofrece uno de los primeros ejemplos de una aplicación auto-reconfigurable sobre lógica reconfigurable dinámicamente con un sistema modularizado.

Con la aplicación de la reconfiguración *intra-task* en este caso se pretende obtener mejoras tanto en velocidad como en área para aplicaciones de comparación de patrones. Para ello, el diseño de los circuitos objeto de reconfiguración se basa en el concepto de *data folding* [144]. El *data folding* consiste en la implementación de circuitos mediante coeficientes fijos, aplicable, por ejemplo, a circuitos tipo comparadores de patrones o unidades de multiplicación. De esta forma se obtienen mejoras sustanciales en cuanto a velocidad y área en comparación a implementaciones con circuitos equivalentes que soporten coeficientes variables.

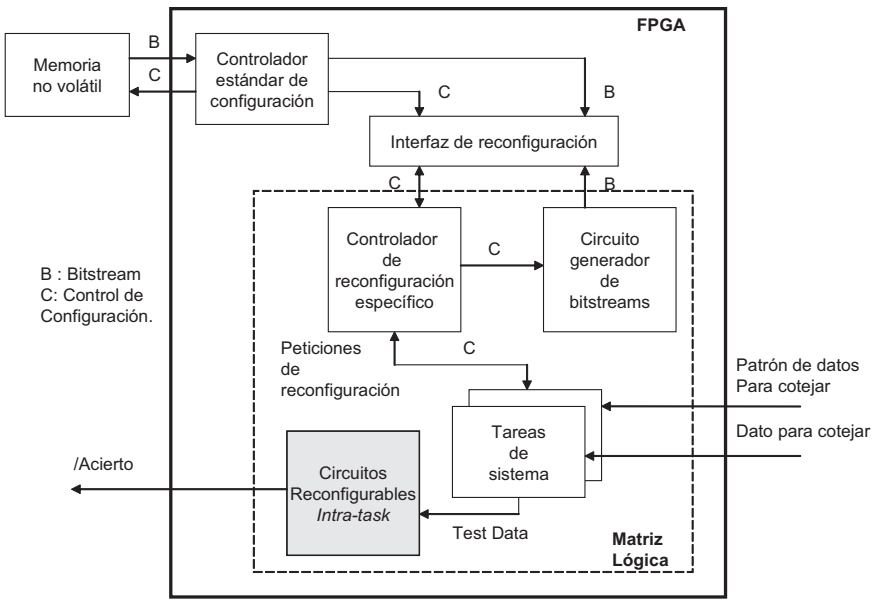


Figura 3.10.: Sistema auto-reconfigurable para la comparación de patrones de datos.

La figura 3.10 muestra la arquitectura del sistema propuesto integrado en una FPGA reconfigurable dinámicamente. En la inicialización, el controlador estándar de configuración carga el *bitstream* inicial en la FPGA. El control de la configuración se transfiere al *controlador de reconfiguración específico* y comienza la ejecución del sistema. Se realizan comparaciones entre el patrón y los datos de entrada, indicándose el resultado mediante la señal */Acierto*. Cuando el patrón de comparación varía, se inicia un nuevo proceso de reconfiguración: *el controlador de reconfiguración específico*, conociendo el nuevo patrón, indica al *circuito generador de bitstreams* mediante un identificador la configuración deseada. El nuevo *bitstream* parcial se carga a través de la interfaz de reconfiguración continuando el proceso de comparación con el nuevo patrón integrado en la lógica (circuitos reconfigurables sobre los que se aplica una reconfiguración *intra-task*).

3.5.2. Plataformas FPGA comerciales de grano fino

G. Martin destaca en el capitulo *"The History of the SoC Revolution"* del libro *"Winning the SoC Revolution: Experiences in Real Design"* [9] el éxito obtenido por las plataformas FPGA comerciales en el diseño de SoCs basados en *cores*. En la actualidad se están desarrollando y aplicando flujos de diseño específicos [98] para la aplicación de la reconfiguración parcial dinámica a *cores* con interesantes y prometedores resultados prácticos [37, 145, 146]. De las diferentes plataformas FPGA comerciales de grano fino destacan las alternativas ofertadas por los fabricantes más importantes de circuitos reconfigurables: Xilinx, Altera y Atmel.

Xilinx propone el uso de la especificación CoreConnect^{TM} *Open Peripheral Bus* [147] de IBM para la interconexión de los *cores* y bus *on-chip*. Los microprocesadores de propósito general que se integran en las plataformas de Xilinx son: el microprocesador *soft* Microblaze

y el PowerPC, ambos de 32 bits. El Microblaze es un *core* parametrizable de tipo *soft*, requiriendo por tanto recursos generales de la FPGA para su implementación. En cambio el PowerPC se trata de un *core hard* ya embebido en el propio silicio del dispositivo de forma previa.

La alternativa ofertada por Altera soporta la especificación para interconexión de *cores* AMBA de ARM [61] y la *Avalon* propia de Altera [148]. Los microprocesadores integrados en este caso también son de 32 bits, disponiendo de un *core soft* denominado NIOS y un *core hard* de un procesador ARM (ARM992T) embebido en los dispositivos *Excalibur*.

También cabe destacar el producto *Field Programmable System Level Integrated Circuits* (FPSLIC) de Atmel [149]. En esta familia de dispositivos se combina un microcontrolador RISC de 8 bits con memoria y una sección de lógica programable de hasta 4000 puertas en un único integrado.

Un elemento básico del diseño con estas plataformas y en general en diseño SoC [110] es la especificación de interconexión de *cores* utilizada. En el siguiente apartado se realiza una breve presentación de las tres especificaciones más utilizadas en plataformas FPGAs de grano fino.

Buses y especificaciones para interconexión de cores en SoCs

Los buses normalizados más conocidos (VME, PCI, ISA, etc.) están específicamente diseñados para funcionar como bus del sistema conectando componentes discretos sobre circuitos impresos o cables. Las restricciones propias en estos sistemas, como son la longitud de las pistas y el número de pines disponibles, no son tales en un dispositivo SoC. Mediante la tecnología SoC se pueden obtener importantes ventajas de la posibilidad de aumentar considerablemente la velocidad del bus del sistema junto con una gran disminución del área del circuito impreso. Sin embargo, en este tipo de diseños integrados en un *chip*, el rendimiento del mismo es altamente dependiente de la arquitectura elegida y del sistema de arbitraje del bus.

Actualmente hay definidas numerosas especificaciones para buses (interconexión de IP-Cores) para diseños SoC. Diversos estudios comparativos [150] concluyen que la arquitectura elegida para un determinado sistema está determinada por el tipo de aplicación. Sin embargo, desde el punto de vista global de diseño de un SoC, se deben tener en cuenta parámetros muy importantes como la *reusabilidad* de *cores*, el *time-to-market* del producto o el pago de derechos por el uso de una determinada especificación. La elección de una determinada arquitectura implica también adoptar una especificación en el diseño de las interfaces de los IP-Cores. Tras realizar diseños aplicando una especificación se dispondrá de un conjunto de *cores* dispuestos para ser utilizados de nuevo fácilmente.

A continuación se resumen en la tabla 3.1 [151] las especificaciones más representativas utilizadas para interconexión de *cores* en sistemas SoC. En esta tabla se reflejan, además de la empresa o consorcio patrocinador de la misma, las condiciones de uso (licencias). Una comparativa más detallada de estas especificaciones puede encontrarse en el trabajo de U. Bidarte [152].

Tal y como se ha indicado anteriormente, la elección de una de estas topologías tiene diversas implicaciones en todas las fases del diseño de SoC, tanto a nivel de sistema (selección

Tabla 3.1.: Especificaciones para interconexión de *cores* y sus licencias de utilización.

Nombre del Bus	**Organización**	**Información sobre patentes**
AMBA	ARM Limited - Cambridge (UK)	Propietario. *Copyright* por ARM. Disponible aceptando la licencia.
ATLANTIC	Altera Corporation - San Jose, CA (US)	Propietario. *Copyright* por ALTERA. Disponible *on-line*.
AVALON	Altera Corporation - San Jose, CA (US)	Propietario. *Copyright* por ALTERA. Disponible *on-line*. La licencia restringe su uso únicamente a dispositivos de Altera.
CHAIN	University of Manchester - Manchester (UK)	No hay evidencia de que este registrado.
CoreConnect™	IBM - Armonk, NY (US)	Propietario. *Copyright* por IBM. Disponible sin pago de tasas.
CoreFrame	Palmchip Corporation - San Jose, CA (US)	Propietario. *Copyright* por PalmChip. Disponible bajo aceptación de licencia.
FISPbus	Mentor Graphics - Wilsonville, OR (US)	Propietario. Especificación no disponible *on-line*.
IPBus	Integrated Device Technology - Santa Clara CA (US)	Propietario. Licencia no disponible.
IP Interface	Motorola, Inc. - Schaumburg, IL (US)	Propietario. Restricciones de uso en la licencia.
MARBLE	University of Manchester - Manchester (UK)	No hay evidencia de que este registrado.
OCP-IP	Open Microprocessor Systems Initiative (OMI)	En la información no se clarifica si es un estándar *abierto* o está sujeto a requerimientos de licencias.
SiliconBackplane	Sonics, Inc. - Mountain View, CA (US)	Propietario. Disponible bajo aceptación de licencia.
VSIA On-Chip Bus	Virtual Socket Interface Alliance (VSIA)	Propietario. La especificación sólo está disponible para los miembros de VSIA bajo aceptación de la licencia.
Wishbone	OpenCores	Especificación abierta. Especificación disponible *on-line* sin restricciones de uso.

de la topología de bus, arbitraje, etc.) como a nivel RTL o incluso del software. Por tanto se trata de un elemento clave en la la metodología de diseño de SoCs.

Con el fin de presentar el concepto de especificación estándar a continuación se comparan tres arquitecturas con idéntico fin: conectar IP-Cores. Esta comparación se basa en el trabajo de R. Usselmann realizado para seleccionar una especificación para la comunidad OpenCores dedicada al desarrollo de hardware libre `http://www.opencores.org` y publicado en [153].

CoreConnect™

En la figura 3.11 se muestra la topología de esta arquitectura con los distintos elementos constructivos: *Processor Local Bus* (PLB), *On-Chip Peripheral Bus* (OPB) y *Device Control Register* (DCR).

El PLB se trata de un bus de alta eficiencia, que se emplea como bus local por los procesadores embebidos en el SoC. Admite hasta dos transferencias por ciclo (solapando la lectura y escritura). Tiene una anchura de bus de 32 o 64 bits de datos y 32 bits de direcciones. Dispone de líneas separadas de lectura y escritura permitiendo *bursts* de información de hasta 64 bytes. Admite arbitraje de dispositivos maestros y modos especiales de acceso directo a memoria. Para disminuir la latencia está provisto de sistemas como el *pipeline* de direcciones entre otros.

El OPB dispone de características similares al anterior pero además permite algunas optimizaciones en consumo (*bus parking*) y control dinámico del tamaño efectivo del bus. Su funcionalidad está orientada a la interconexión de periféricos. El DCR se trata de un bus muy especializado para transferencia de datos entre registros de propósito general de la CPU y los registros de configuración de los dispositivos conectados al bus. De esta forma se pretende aliviar de tráfico el bus local y aumentar su ancho de banda. Dispone de 10 bits de bus de direcciones y 32 bits de datos.

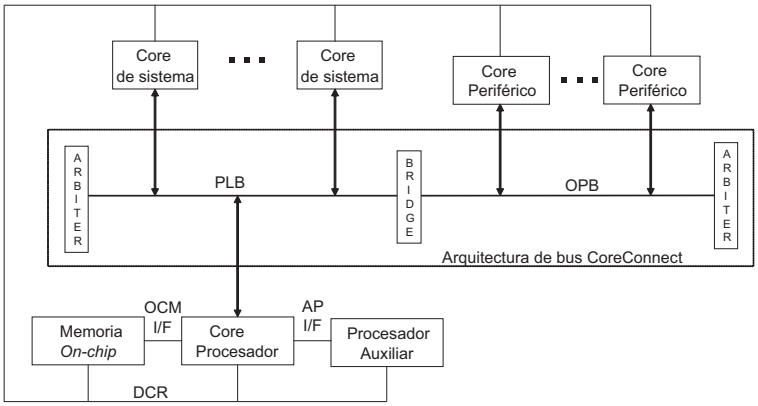


Figura 3.11.: Tipos de buses en la especificación CoreConnect™ de IBM.

Coreconnect™ es una arquitectura muy potente, pero compleja. Para aplicaciones SoC relativamente sencillas puede resultar sobredimensionada e innecesariamente sofisticada [153].

AMBA

Es la especificación definida por la compañía ARM. Estructuralmente se asemeja a la CoreConnect™. Inicialmente se definió con tres tipos de buses: *Advanced High Speed Bus*

(AHB), *Advanced Peripheral Bus* (APB) y *Advanced general purpose System Bus* (ASB). En la figura 3.12 se muestra un ejemplo de conexión de diversos *cores* empleando esta especificación. El bus de alto rendimiento que se corresponde con el bus local del microprocesador puede ser del tipo AHB mientras que los periféricos se pueden conectar a un bus de tipo APB a través de un *bridge*. El ASP tiene características similares al APB.

En [153] se critica negativamente este bridge y se pone en duda la utilidad del bus ASB, indicando qué latencias producidas por periféricos de alto rendimiento se ven propagadas hacia el bus AHB a través del ASB. De hecho, este último bus ha sido retirado de la especificación AMBA 2.0 [61].

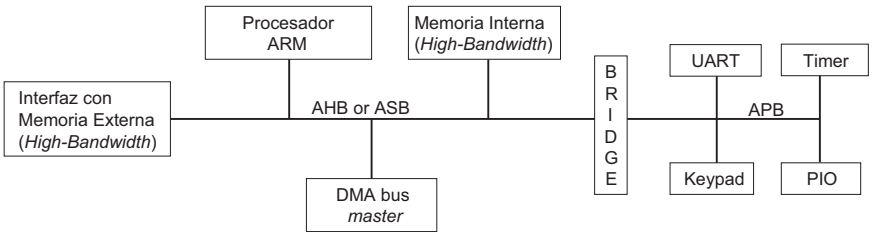


Figura 3.12.: Tipos de buses en la especificación AMBA de ARM.

El propósito principal del bus AHB es la interconexión de *cores* de dispositivos de alto rendimiento tales como microprocesadores, controladores de acceso a memoria y procesadores digitales de señal. Sus características técnicas mas relevantes se resumen en:

- Multi-maestro con control de arbitraje.
- Acceso en un único ciclo por parte del maestro.
- Anchura de bus de datos de 32 a 1024 bits.
- Sistema de protección de acceso integrado, con distinción de nodos privilegiados y no privilegiados.
- Admitidas transferencias *burst* y *pipeline*.
- *Bursts* limitados a 16 transferencias.
- Transferencias en modo byte, *half-word* y *word*.

El bus APB se utiliza para la interconexión de periféricos. Está especificado con el fin de disminuir el consumo y facilitar su uso. Se define un único *master* y únicamente dispone de 4 señales de control. Dispone de un espacio de memoria de 32 bits y buses separados de lectura y escritura.

Wishbone

Se trata de una especificación de interconexión de *Cores* inicialmente desarrollada por la empresa *Silicore Corporation*. Actualmente se encuentra bajo el control de la comunidad de hardware libre *Opencores*.

La revisión 3 de Wishbone está disponible en la página web `http://www.opencores.org` [154]. Es una especificación muy versátil, agrupando diversos niveles de obligatoriedad en los distintos apartados. De mayor a menor obligatoriedad se definen respectivamente: *Rules, Suggestions, Permissions y Observations.*

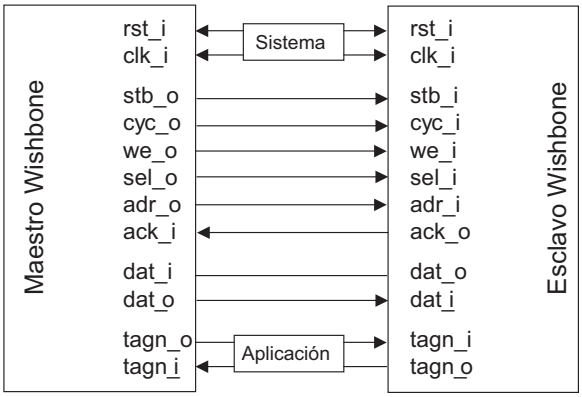


Figura 3.13.: Conexión de *Cores* mediante un único bus Wishbone.

En la figura 3.13 se representa la simplicidad de una arquitectura Wishbone para una conexión punto a punto entre dos *cores*. Esta especificación no define distintos buses para bloques de alto rendimiento y para los de menor requerimiento.

Sin embargo, las numerosas variantes que se pueden realizar en la interconexión de los distintos bloques y los distintos ciclos de acceso que se admiten, permiten diseñar un determinado bus Wishbone adaptado a cada aplicación.

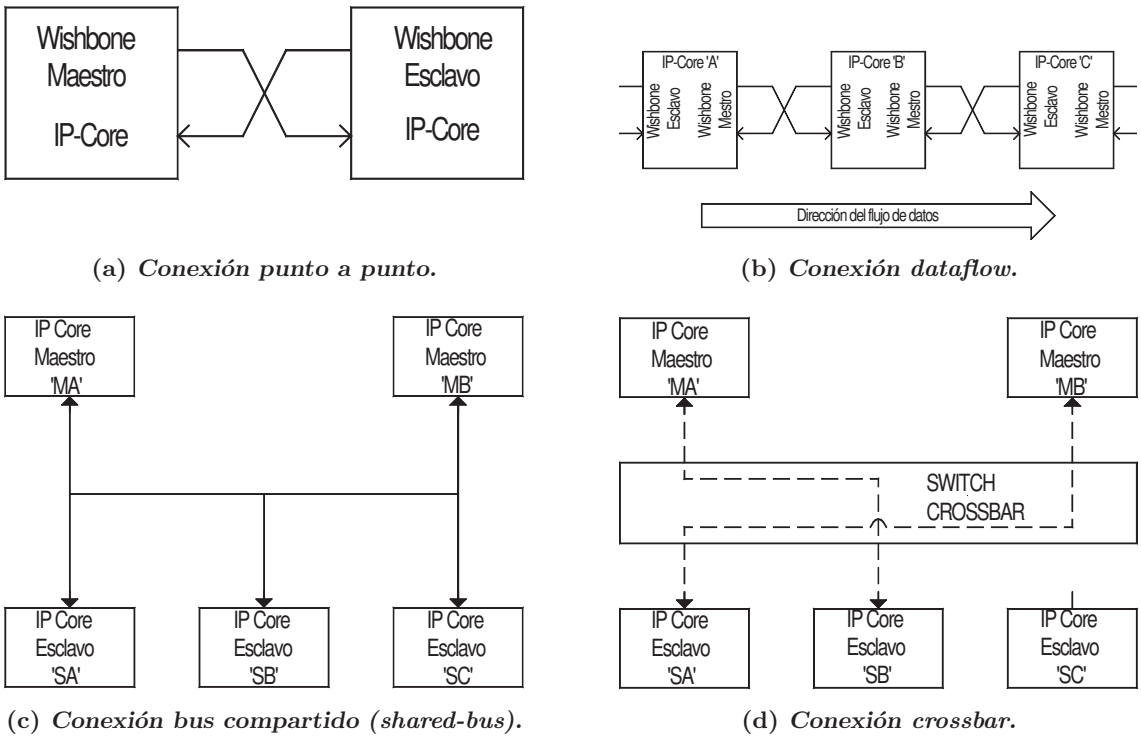
En las figuras 3.14(a), 3.14(b), 3.14(c) y 3.14(d) se muestran las topologías de interconexión admisibles en la especificación Wishbone [154]. Según el modo óptimo de flujo de datos, la complejidad del sistema y el *throughput* necesario, se puede optar por una topología punto a punto, *dataflow*, de bus compartido o de tipo *crossbar*.

Mediante estas topologías se pueden optimizar las arquitecturas tanto para aplicaciones embebidas de tamaño reducido como para las de complejidad elevada y con requerimientos de baja latencia en el bus (por ejemplo, desarrollando una matriz para interconexión *crossbar*).

La especificación soporta también arquitecturas multi-maestro, disponiendo de 64 bits de espacio de memoria direccionable y una anchura de bus de datos mínima de 8 bits escalable a mayores tamaños según las necesidades.

Wishbone especifica un protocolo *handshake* para el control de las transferencias en el bus entre los *cores*. Usa una topología maestro-esclavo y aunque la especificación soporta transferencias asíncronas en un único ciclo de reloj, para diseños con FPGAs se utiliza únicamente la versión síncrona del protocolo.

El sistema de arbitraje está definido por el propio usuario. También se reservan un grupo de señales en el bus, denominadas TAGs, para definición por parte del usuario según la aplicación.



(a) ***Conexión punto a punto.*** **(b)** ***Conexión dataflow.***

(c) ***Conexión bus compartido (shared-bus).*** **(d)** ***Conexión crossbar.***

Figura 3.14.: Topologías de interconexión de bus soportadas por especificación Wishbone.

3.5.3. Sistemas para control de la reconfiguración parcial dinámica en plataformas FPGA comerciales de grano fino

Los diseños realizados con estas plataformas e implementados sobre dispositivos reconfigurables se benefician de la capacidad de configuración de los mismos básicamente en la etapa de diseño y en posteriores actualizaciones. Sin embargo, la aplicación de la tecnología de auto-reconfiguración parcial dinámica que ofrecen algunos de estos nuevos dispositivos se encuentra todavía en una fase embrionaria respecto a la aplicación en diseños comerciales.

En los últimos años los fabricantes de dispositivos reconfigurables han incluido notables mejoras respecto a las capacidades de reconfiguración parcial y dinámica de éstos, siendo las posibilidades cada vez más prometedoras en este campo. Destacan la integración de sistemas de acceso a los puertos de configuración desde el interior de los propios dispositivos, como el módulo ICAP de Xilinx [155] para dispositivos Virtex-II o el control integrado de la reconfiguración dinámica del FPSLIC de Atmel [156].

Para poder hacer uso de estas posibilidades en los diseños actuales basados en *cores* se requieren metodologías, entornos y herramientas que lo faciliten. En este campo concreto, cabe destacar el trabajo de B. Blodget [157, 158] sobre el desarrollo de la plataforma de auto-reconfiguración *Self-Reconfiguring Platform* (SRP). SRP es una plataforma para dispositivos Virtex-II [159] y Virtex-II Pro [160] que permite controlar de forma dinámica la reconfiguración de la FPGA bajo el control del microprocesador embebido en el CSoPC. SRP está compuesta por una sección hardware y otra sección software.

El subsistema hardware está formado por el componente ICAP de acceso interno al puer-

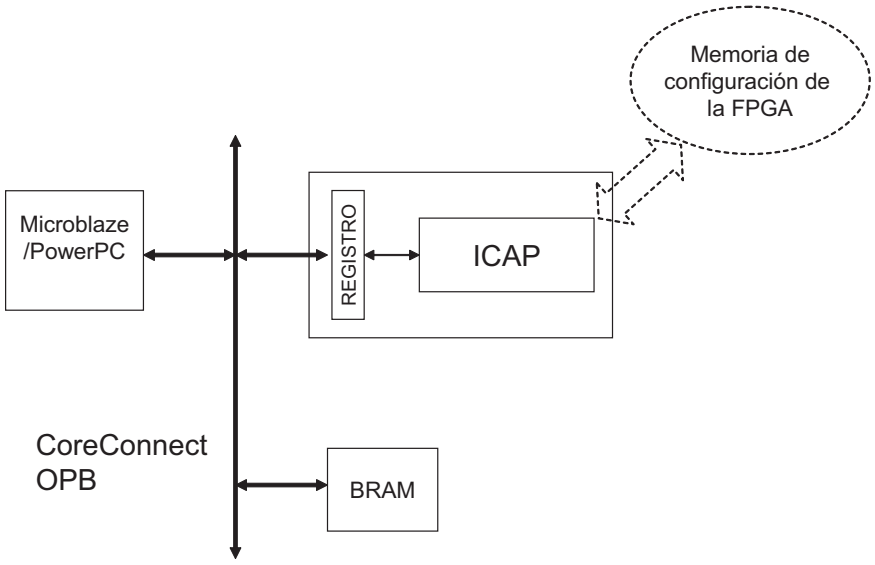


Figura 3.15.: Módulos hardware de la plataforma SRP.

to de configuración SelectMAP, la lógica de control, una pequeña *caché* de configuración y el procesador embebido. En este caso, la *metacomputación* se realiza mediante el procesador incluido en el mismo circuito integrado que la *lógica configurada*, variante de auto-reconfiguración denominada *reconfiguración en tiempo de ejecución*. La lógica de control de las operaciones de lectura y escritura sobre el ICAP se ejecutan mediante un *driver* software de bajo nivel. El bloque de memoria RAM dedicada BlockRAM [31] (BRAM) de la figura 3.15 es una *caché* de datos de configuración. El procesamiento se realiza mediante los microprocesadores embebidos los cuales acceden a la *caché* y al ICAP a través del bus *on-chip*.

El subsistema software está diseñado en capas, de forma que se puedan diferenciar niveles dependientes del hardware y no dependientes del mismo. La figura 3.16 muestra esta distribución obtenida mediante el API desarrollado (*Xilinx Partial Reconfiguration Toolkit* -XPART-) que provee de métodos para transferir información entre la memoria de configuración y la *caché* y accesos a ésta. En la misma figura se representa también un apartado denominado *Sistema Externo de Emulación* que permite la emulación del acceso a ICAP en una estación de trabajo o PC. El conjunto de herramientas XPART define métodos para realizar modificación dinámica de recursos. XPART usa una API (ICAP API) para acceder a la interfaz de configuración interna ICAP. Esta escritura de datos de configuración a través del ICAP es interna al circuito integrado, por lo que habilita la posibilidad de desarrollar sistemas auto-reconfigurables totalmente integrados.

Aunque este avanzado sistema de aplicación de auto-reconfiguración aún se encuentra en fase de desarrollo, está tratándose de aplicar a diseños industriales. Un ejemplo representativo es el *router* reconfigurable dinámicamente para redes ópticas presentado por S. Young en [161].

Fong, Harper y Athanas también proponen en [145] un *framework* para la aplicación de

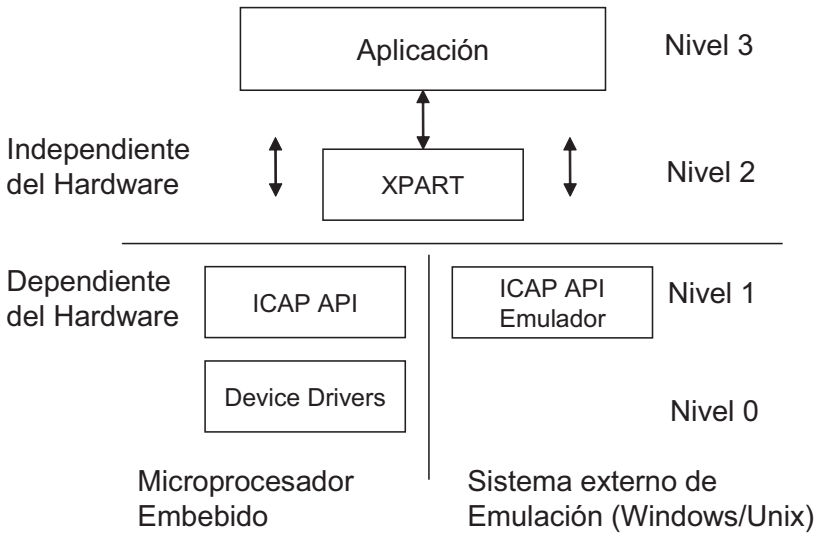


Figura 3.16.: Niveles software para la plataforma SRP.

la auto-reconfiguración a diseños implementados en FPGAs Virtex-II the Xilinx utilizando la interfaz ICAP. La figura 3.17 muestra la arquitectura generalizada de su propuesta.

A través del canal de comunicaciones, denominado *Configuration Data Medium*, se reciben los datos de configuración de la FPGA. La interfaz de comunicaciones se completa con lógica implementada en la propia FPGA (*Interfaz de comunicaciones* en la figura). Esta propuesta de plataforma de auto-reconfiguración incluye un módulo gestor de seguridad en la lógica integrada en la FPGA. De esta forma se pueden añadir elementos de protección sobre el hardware IP descrito en la configuración que se recibe a través del canal de comunicaciones.

Siguiendo el flujo de los datos de configuración, tras el módulo de seguridad se verifica la integridad de los datos, pasando finalmente a una interfaz de configuración interna. Este último módulo carga la configuración parcial dinámica sobre la *aplicación* de usuario, siendo esta la única zona que se modifica. El prototipo se ha realizado sobre una FPGA Virtex-II, empleando el flujo de diseño propuesto por Xilinx en [98]. El prototipo incluye soporte para una interfaz de comunicación serie con un *host* externo y un módulo criptográfico *Blowfish*.

Los resultados obtenidos a partir de ese trabajo demuestran la viabilidad de la tecnología de reconfiguración parcial dinámica para la modificación *inter-task* (sección 3.1), ofreciendo unos tiempos de modificación para un 13 % del área de FPGA del orden de 5 segundos. Este tiempo incluye la recepción de los datos de configuración a través de un canal lento de comunicación serie, procesamiento de seguridad, verificación y aplicación de los mismos.

También se ha considerado interesante destacar en este apartado la aplicación de la reconfiguración parcial dinámica a controladores lineales industriales (e.j. controladores mecatrónicos) publicada por K. Danne en [37]. En este trabajo se plantea un diseño realizado sobre lógica parcial y dinámicamente reconfigurable (dispositivos Virtex de Xilinx) aplicando una arquitectura denominada *Multi-controller*.

Esta arquitectura, representada en la figura 3.18, está compuesta por un conjunto de *módulos controladores* (*Controller Modules* -CM-) optimizados cada uno de ellos para un

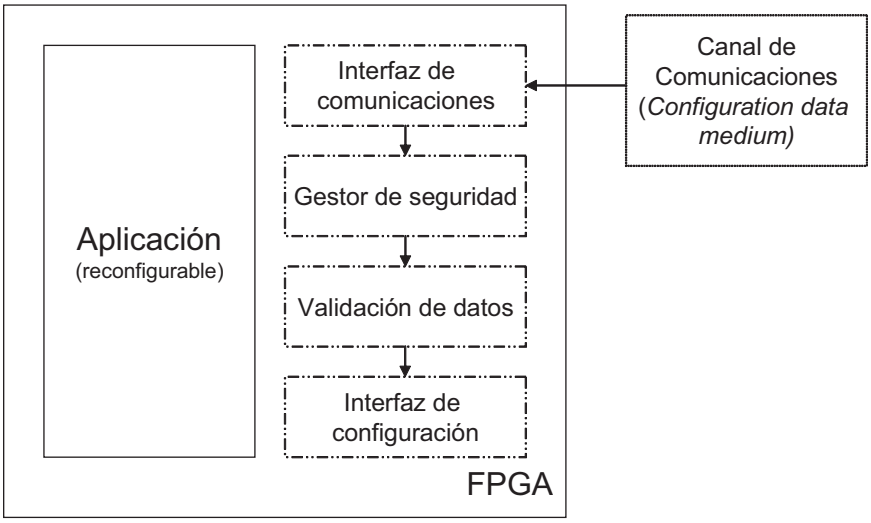


Figura 3.17.: Arquitectura del *framework* para auto-reconfiguración integrado en una FPGA propuesto por Fong et al.

régimen de operación de la planta. El *supervisor* es capaz de cambiar el módulo de control activo. Para reducir el coste en área derivado de la implementación de esta arquitectura mediante la instanciación de cada componente dedicado para cada CM en paralelo, se propone la utilización de la reconfiguración parcial dinámica.

La representación realizada en la figura 3.19 corresponde con este diseño. Los CMs se configuran en unas áreas restringidas en la FPGA para las secciones dinámicas. Estas áreas se denominan *slots*. El *supervisor* está situado en la sección estática. Si éste determina que se debe cambiar de controlador, lo indicará al *controlador de configuración*. El controlador cargará el nuevo CM en el *slot* que no esté siendo utilizado, indicando el final de la configuración al *supervisor*. Mediante el módulo multiplexor (MUX) se cambiará de *slot* aplicando el nuevo controlador a la planta. Mediante este sistema se consigue reducir sustancialmente el tiempo de cambio entre controladores.

Aunque la aplicación presentada está circunscrita a un campo concreto (controladores lineales de plantas), este trabajo plantea un enfoque práctico del uso de la reconfiguración parcial dinámica en FPGAs de grano fino con la tecnología disponible en la actualidad y con un flujo de diseño modular extensible a otros sistemas basados en *cores*.

Más recientemente, M. Ullmann et al. de la Universidad de Karlsruhe han presentado un sistema que también explota las posibilidades de la reconfiguración parcial dinámica de los dispositivos Virtex de Xilinx aplicada a la electrónica de automoción [162]. Utilizan un software que en tiempo de ejecución controla la reconfiguración y el intercambio de mensajes. Además, incluyen elementos de prioridad dinámica como una primera aproximación hacia soluciones para toma de decisiones adaptativas relativas a la reconfiguración.

La reconfiguración *inter-task* propuesta parte de la premisa de que las distintas funciones de los *cores* del sistema electrónico del automóvil no necesitan estar simultáneamente activas, siendo posible definir unos subconjuntos que se carguen en el dispositivo reconfigurable aplicando una estrategia de multiplexado en el tiempo bajo demanda [163].

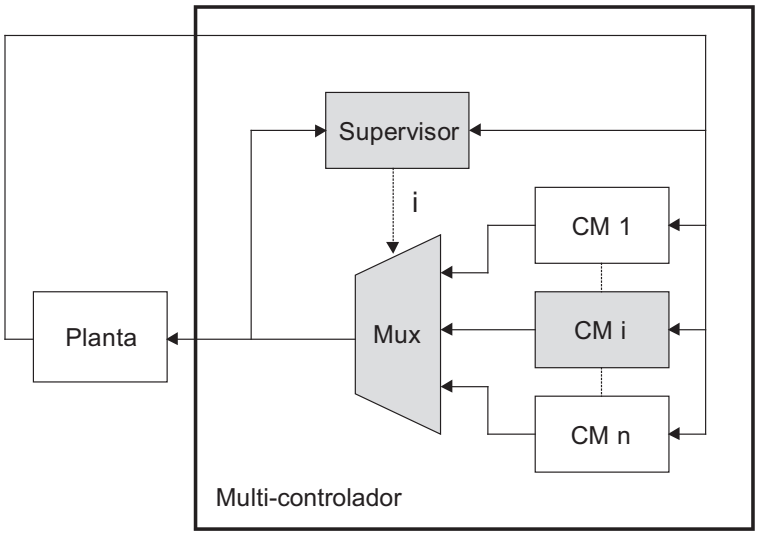


Figura 3.18.: Arquitectura *Multi-controller*.

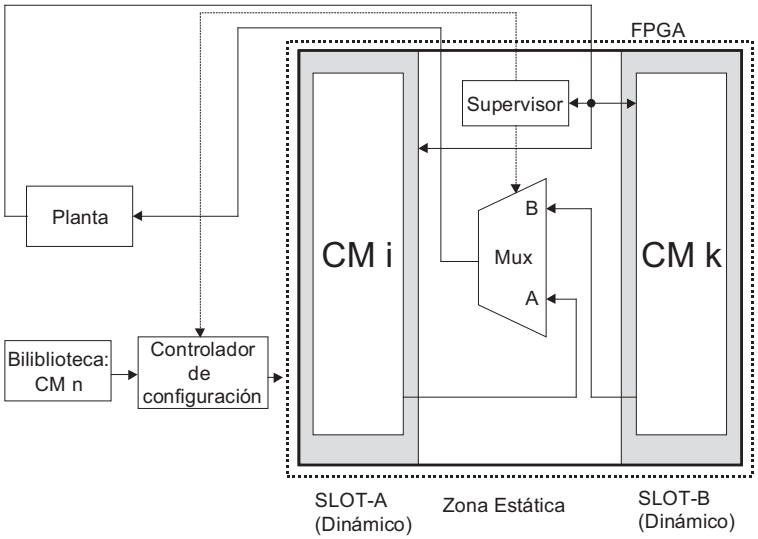


Figura 3.19.: Implementación mediante reconfiguración parcial dinámica de la arquitectura *Multi-controller*.

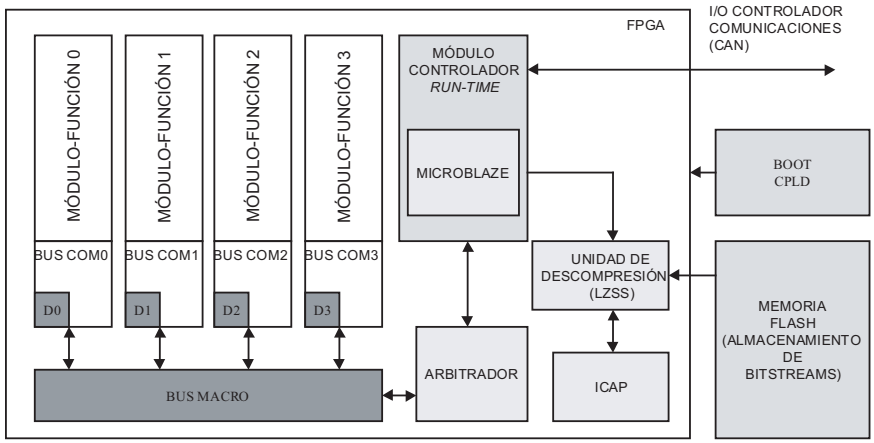


Figura 3.20.: Bloques principales de modelo de reconfiguración parcial dinámica con gestión de prioridades.

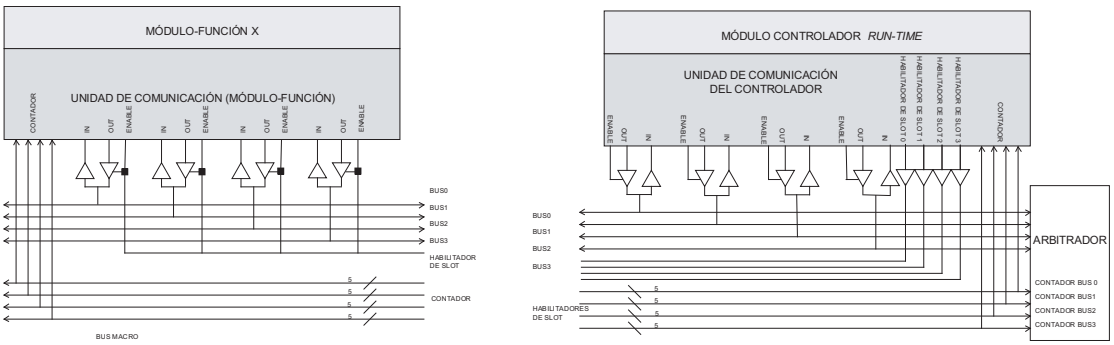
La figura 3.20 muestra el modelo del sistema propuesto. La FPGA reconfigurable parcial y dinámicamente incluye un controlador de reconfiguración implementado, al igual que en la propuesta de Blodget [157], mediante el microprocesador *soft* de propósito general Microblaze. Éste se comunica a través de un periférico CAN con los distintos sensores y actuadores. Los mensajes, tras ser procesados y decodificados por el microcontrolador, se envían al arbitrador que controla las líneas del bus *on-chip*. A este bus se conectan los módulos-función reconfigurables. En la figura 3.20 se han representado cuatro módulos función, que se pueden reconfigurar parcialmente para cargar distintas funcionalidades. De igual forma, los mensajes generados por éstos se reciben en el arbitrador y se envían a los dispositivos externos a través del microprocesador Microblaze.

Una aportación destacable de esta propuesta es la capacidad del sistema de guardar el estado y diversos parámetros de una función cuando es sustituida por otra. El mecanismo de salvaguarda es administrado por el controlador, enviándose los datos de cambios de contexto a través del bus bajo el control del arbitrador.

Puesto que el controlador puede recibir mensajes dirigidos a módulos-función no activos, el software de control de reconfiguración en tiempo de funcionamiento que se ejecuta en el controlador gestiona un *buffer* de mensajes, almacenándolos temporalmente hasta que el módulo destino esté activo.

Para la aplicación de automoción planteada se introduce una métrica de prioridad que permite determinar adecuadamente qué función es la más importante de las que se demandan que sean ejecutadas. Para ello, se definen parámetros estáticos que dependen de la importancia de las funciones y parámetros dinámicos función del estado de la comunicación (mensajes almacenados, tiempo máximo de espera para cada función, etc.).

La comunicación entre los distintos módulos-función reconfigurables se realiza mediante un sistema *Network-on-Chip* (NoC) basado en comunicación serie [164]. Los sistemas de interconexión NoC [165, 166] se basan en redes de conmutación de paquetes, similares a las



(a) ***Interfaz Bus-Macro para un módulo-función genérico.***

(b) ***Interfaz Bus-Macro para el controlador.***

Figura 3.21.: *Bus-Macro* para la NoC propuesta en el modelo de Ullmann et al.

empleadas en las redes de ordenadores. Los distintos elementos que intervienen en la red (en el caso de las NoCs, los *cores* que componen el sistema) se conectarán a los elementos de conmutación de paquetes (*switches*) mediante una interfaz de red. Las topologías de interconexión de los diferentes módulos mediante NoCs son muy variadas, empleándose mallas, hipercubos de varias dimensiones, redes en mariposa, estructuras en árbol de diferentes tipos, etc. [167].

En esta propuesta de Ullman et al. para sistemas digitales en automoción, los módulos reconfigurables acceden a las líneas de comunicación de la NoC a través de las *Bus-Macros* de Xilinx. Estas *Bus-Macros*, presentadas anteriormente en los apartados 3.2.2 y 3.2.1, son módulos pre-rutados con *buffers* triestado (componentes *Tbuf* de Xilinx) que permiten emplazar las conexiones en posiciones concretas de la matriz lógica de la FPGA. Son, por tanto, necesarias para la aplicación de la reconfiguración parcial dinámica en la tecnología Virtex de Xilinx [98]. En esta aplicación, las *Bus-Macros* están diseñadas de tal manera que componen *buffers* bidireccionales que permiten desconectar del bus el módulo-función que se esté reconfigurando (figura 3.21(a)). El arbitrador de bus integra un contador de cinco bits para cada línea de bus que divide el tiempo de acceso en *slots* de tiempo. Las unidades de comunicación de los módulos-función reconfigurable y del controlador disponen de una determinada cantidad de *slots* de tiempo que se corresponden con el valor de cuenta para el acceso a una o más líneas de bus durante un determinado periodo del contador [164]. Cada módulo tiene asignados una lista de números que representan el número de *slot* para el que el módulo puede acceder al bus.

Otra propuesta, realizada para el control de la reconfiguración parcial dinámica aplicada a diseños basados en *cores* en FPGAs de grano fino, es la denominada *Fixed core Processor connected to Reconfigurable Coprocesor* (FiPRe) [168, 169]. Incluye la implementación de coprocesadores reconfigurables. Estos coprocesadores se encargan de acelerar tareas específicas, siendo posible su inserción o sustitución en tiempo de funcionamiento del sistema. Además, para facilitar el diseño de software, incluyen el *core* de un microprocesador que dispone de un conjunto de instrucciones fijo. Este microprocesador, denominado R8R está derivado del R8 [170] al cual se le han añadido cinco instrucciones para controlar los

coprocesadores reconfigurables.

El modelo multi-procesador FiPRe también se ha implementado sobre la tecnología parcial y dinámicamente reconfigurable Virtex-II [159] de Xilinx. Además de utilizar las herramientas de diseño convencionales que oferta este fabricante, emplean herramientas auxiliares específicamente diseñadas por el grupo de investigación [171], como ocurre en la mayoría de los sistemas presentados.

En la figura 3.22 se presenta la infraestructura genérica de un sistema R8NR con N unidades coprocesadoras, realizado según el modelo FiPRe. La sección no reconfigurable, identificada en la figura como *región fija*, está compuesta por el procesador de propósito general R8R junto con un *Controlador de Configuración* implementado en hardware. Este controlador es un módulo esclavo del *host* externo para las operaciones de transferencia de configuraciones desde éste y del R8R, el cual lo utiliza para acceder a la interfaz de configuración interna ICAP [155]. El procesador R8R es capaz de comunicarse con el bus del sistema de la sección fija y con la sección reconfigurable. Para ello de nuevo emplea las *Bus-Macro* de Xilinx.

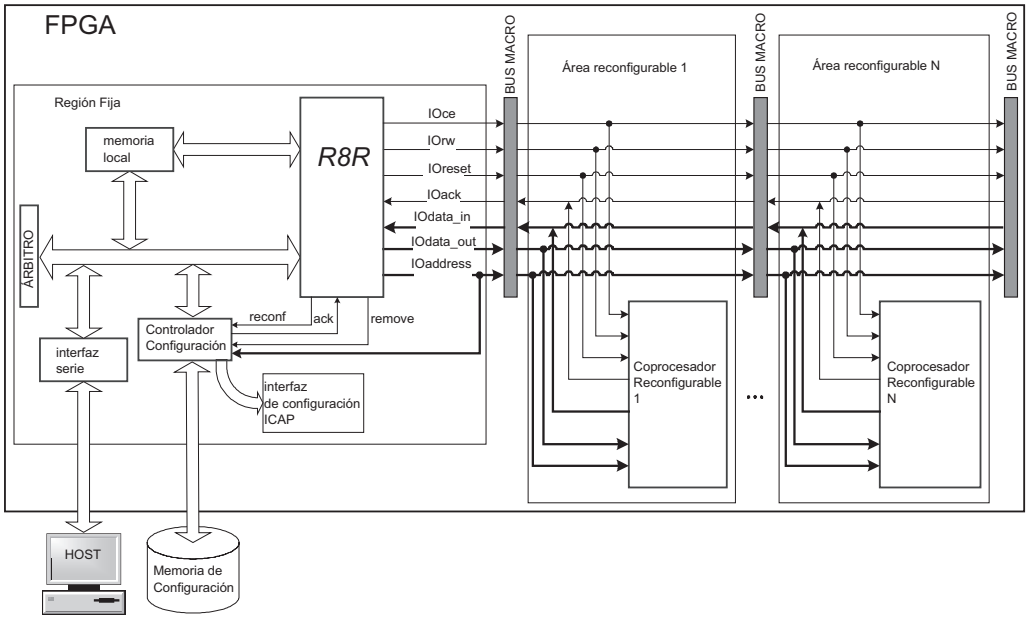


Figura 3.22.: Infraestructura general del sistema R8NR (modelo FiPRe).

Para finalizar este apartado, cabe destacar las aportaciones más recientes en esta área relativas a *sistemas operativos para hardware reconfigurable* [172], surgidas en gran medida por la disponibilidad de nuevas FPGAs de alta densidad.

En esta línea, H. Walder y M. Platzner presentan en [173, 174] una infraestructura de bus (*Task Communication Bus* -TCB-), señales y lógica para que un *sistema operativo de hardware reconfigurable* pueda gestionar múltiples tareas hardware en una FPGA Virtex-II de Xilinx. El sistema operativo se ejecuta en un procesador embebido en la propia FPGA y dispone de elementos similares a los utilizados en los sistemas operativos de tiempo real para sincronización como son FIFOs, semáforos, temporizadores, buzones de mensajes, etc.

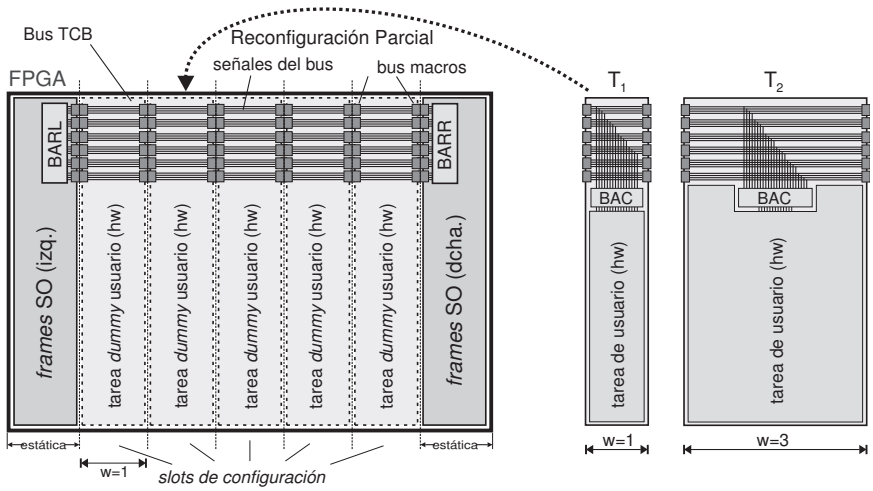


Figura 3.23.: Entorno *Run-Time* para Sistema Operativo para hardware reconfigurable de H. Walder y M. Platzner.

La figura 3.23 representa la infraestructura propuesta, orientada a cumplir con requisitos tecnológicos impuestos por la tecnología Virtex de Xilinx para soportar la reconfiguración parcial dinámica: reconfiguración en una dimensión (por columnas) [175] y uso de *Bus-Macro* para interconectar las rutas que unen secciones reconfigurables.

A la izquierda y a la derecha se sitúan las secciones estáticas, con el procesador y el sistema operativo. En el centro, conectados a través de un bus con topología de bus compartido, se sitúan unos *slots* (áreas delimitadas para los circuitos que se configuran dinámicamente) donde se cargarán las distintas tareas hardware. Una tarea hardware puede ocupar más de un *slot* reconfigurable. Cada *slot* incluye una interfaz de control de la reconfiguración, que permite determinar el estado del mismo, e interfaces para el intercambio de datos.

También en la misma línea de sistemas operativos con soporte para tareas hardware hay que destacar el modelo *Alter-Ego* de A. Segard y F. Verdier [176]. Proponen dotar tanto a las tareas software como a las hardware (IPs) con la misma interfaz. Mediante la abstracción de todos las módulos a tareas software se facilita la gestión de las mismas mediante un sistema operativo de tiempo real. Aunque la reconfiguración parcial dinámica todavía no ha sido implementada usando este modelo, esta propuesta conjuga elementos claves en la evolución de los sistemas CSoPC, como son el uso de interfaces normalizados y sistemas operativos de tiempo real ejecutándose en procesadores embebidos en el mismo dispositivo.

4. Cores Mixtos

4.1. Integración de pequeños procesadores en cores

La construcción de las secciones de control de los sistemas digitales mediante microprocesadores es una práctica ampliamente extendida dada su flexibilidad y su optimización de recursos. Además de microprocesadores y microcontroladores de propósito general, los cuales siguen distintas evoluciones según el mercado al que van dirigidos, también se utilizan extensamente núcleos o *cores* de circuitos digitales de procesamiento secuencial para gobernar las secciones de control de circuitos integrados de aplicación específica (ASICs y ASSPs) [177]. En concreto procesadores con un conjunto reducido de instrucciones (*Reduced Instruction Set Computer* -RISC-).

Dada la alta densidad de recursos lógicos y de rutado de las FPGAs actuales, es viable la integración de pequeños procesadores en los propios *cores* utilizados para componer diseños SoPC. Esta posibilidad es una solución muy versátil para diversos diseños. Además, la naturaleza *soft* de estos procesadores cuando se implementan sobre dispositivos programables, añade un grado de libertad en el diseño en lo referente a la optimización de los recursos necesarios para los procesadores. Máquinas de estado complejas o lazos de control pueden realizarse de forma eficiente empleando este tipo de dispositivos embebidos. Actualmente tienen mucha aceptación procesadores parametrizables descritos en lenguajes de descripción hardware (*Hardware Description Languaje* -HDL-) y de libre distribución como el PicoBlaze de Ken Chapman [178, 179] o el sistema CPUGEN [180], además de muchos otros de libre distribución disponibles a través de Internet [181].

Las características generales de este tipo de procesadores orientados a la integración en *cores soft* son:

- **Disponen de un tamaño muy reducido.** Lo que permite la optimización de los recursos del dispositivo reconfigurable. Para obtener una optimización efectiva se parte de un núcleo procesador básico desprovisto de puertos y periféricos. De esta forma en la lógica adicional de cada módulo se incluyen únicamente los puertos o elementos adicionales que se requieran para la funcionalidad específica. Por este motivo, el autor se referirá a este tipo de procesadores como `nano-procesadores`.

- **Proveen un modelo detallado simulable en HDL** para poder ser integrado en el entorno de co-simulación. En muchos casos al ser procesadores de complejidad relativamente baja, no es necesario trabajar con modelos de alto nivel de abstracción. Es viable (en cuanto a tiempos de simulación) realizar co-diseño y co-simulación con modelos sintetizables [182, 183, 184].

- **Son sintetizables para diversos dispositivos reconfigurables.** Esta premisa, en muchos casos está restringida a un grupo de familias de dispositivos de un fabricante. Al tratar de optimizar al máximo el uso de recursos puede que la descripción HDL

utilice entidades propias de una determinada tecnología. En estos casos, se sacrifica la portabilidad. Un ejemplo de esta situación es el microprocesador *PicoBlaze* que únicamente es sintetizable para algunas familias del fabricante de dispositivos reconfigurables Xilinx.

- **Permiten la composición de sistemas SoPC multi-procesador**. Mediante la integración de múltiples *cores* con este tipo de procesadores integrados se pueden componer sistemas heterogéneos multi-procesador [177, 185], disponiendo de unas capacidades de procesamiento superiores a las obtenidas con potentes procesadores de propósito general.

Aunque estos procesadores pueden emplearse para aplicaciones de procesamiento de datos, por ejemplo en aplicaciones de criptografía [186, 187], son más adecuados para cubrir secciones de control en el *core*. El factor dominante de diseño para estos procesadores es su tamaño. Pero incluso con estas restricciones pueden ofrecer una potencia de procesamiento elevada (varios MIPS). La velocidad de ejecución dependerá tanto de la arquitectura interna del microprocesador como de las características del dispositivo reconfigurable sobre el que se va a implementar.

La flexibilidad de los dispositivos reconfigurables para realizar la distribución de funciones entre procesamiento hardware y procesamiento software [110, 111, 112, 188] puede extenderse a la partición a nivel del interior de los *cores* con `nano-procesadores` embebidos. Las funciones del *core* que requieran un procesamiento hardware intensivo se pueden resolver mediante circuitos descritos en HDL integrados en el mismo, mientras que el procesamiento software lo realizan el o los `nano-procesadores` integrados en el *core*. Éstos pueden tener distintos grados de conexión con las entidades hardware internas. Además, la propia naturaleza *soft* de estos procesadores admite la parametrización de los mismos de diversas maneras, como por ejemplo, la anchura de los buses, el número de puertos [180] o incluso la adición de instrucciones específicas [189]. De esta forma se obtienen descripciones hardware optimizadas.

Tal y como se ha presentado en secciones anteriores, desde el punto de vista de los sistemas basados en *cores* se considera esencial el uso de una especificación estándar para la interconexión de IP-Cores [150, 151, 153]. También, de forma más específica, los sistemas multi-procesador integrados en un *chip* en general se articulan siguiendo una determinada especificación estándar que facilite las tareas de integración y diseño [190, 191]. Estas especificaciones, entre otros elementos, definen el bus de interconexión y las interfaces asociadas. En el conjunto de circuitos hardware integrados en el *core* se encuentran también los correspondientes a estas interfaces.

El autor denominará `Cores Mixtos` a los *cores* que incluyan un `nano-procesador` embebido (*core soft*) más circuitos adicionales encargados del procesamiento hardware. Además, estarán dotados de una interfaz estándar para la conexión de los mismos al bus *on-chip*.

En la figura 4.1 se representan los bloques que definen la arquitectura de `Core Mixto`: un `nano-procesador` con el programa embebido en bloques de memoria dedicada, un hardware para procesamiento específico y una interfaz para la conexión con el bus *on-chip* estándar. Los buses de datos e instrucciones del `nano-procesador` son locales al `Core Mixto`, lo que facilita la integración de múltiples `Cores Mixtos` en un mismo sistema. Éstos funcionan de manera concurrente pudiendo intercambiar información a través del bus *on-chip*.

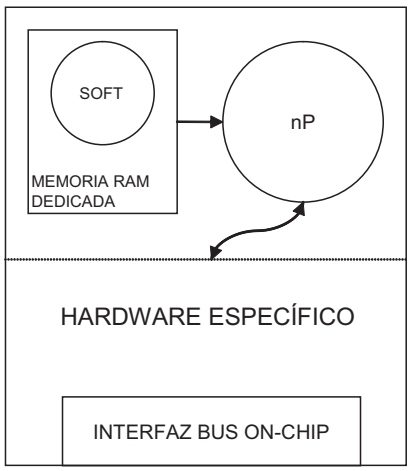


Figura 4.1.: Arquitectura de *core* con `nano-procesador` embebido (`Core Mixto`).

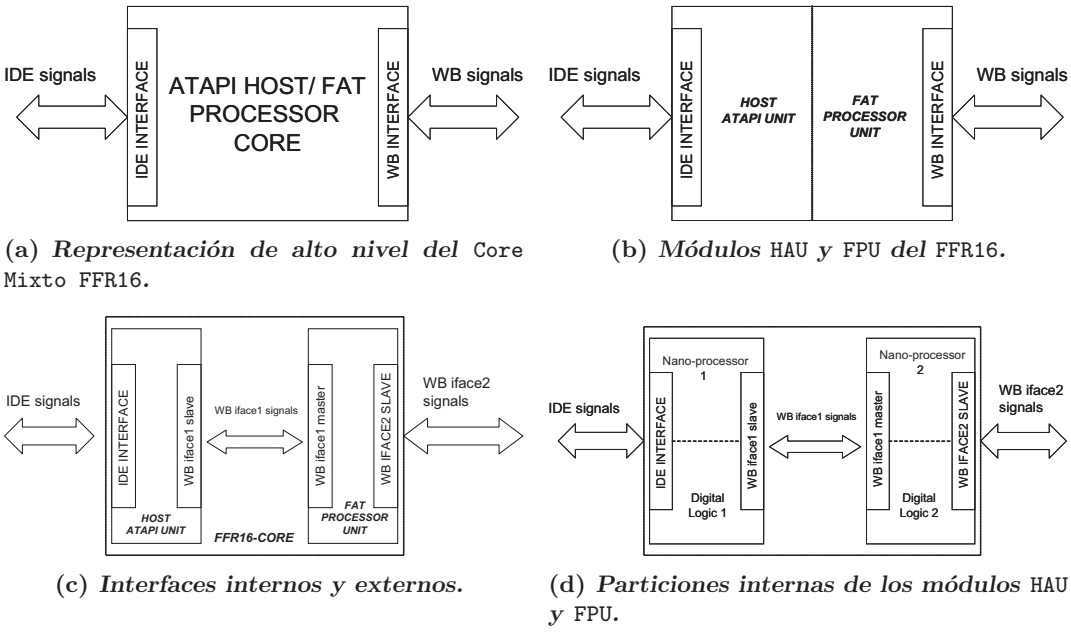
4.2. Campo de aplicación de los Cores Mixtos

Los `Cores Mixtos` son muy adecuados para diversas aplicaciones. Su flexibilidad permite desarrollos rápidos y actualizables. En los nuevos diseños se están empleando como alternativa para aplicar complejas secuencias de inicialización sobre *chipsets* LAN o vídeo. Pero al incluir procesamiento hardware y software, su utilidad no queda reducida únicamente a tareas de control. También se emplean en *appliances*[1] TCP-IP para procesamiento en paralelo de tramas de comunicación [192, 135], o como interfaces inteligentes para la comunicación dispositivos de almacenamiento masivo [193]. Las disciplinas emergentes como la Radio Definida por Software (*Software-Defined-Radio* -SDR-) [194] o la *mecatrónica* constituyen unos sectores para los que estas arquitecturas ofrecen mayores ventajas. Es habitual encontrar *cores* a nivel industrial con esta topología, como el controlador PID de MEET Electronics Ltd. [195].

Un ejemplo ilustrativo de la aplicación de las arquitecturas de `Core Mixto` a aplicaciones complejas es el *core First File Reader FAT16* (`FFR16`) diseñado por el autor de esta tesis y publicado en [196]. Este *core* ha sido desarrollado con el fin de experimentar la viabilidad del co-diseño y co-simulación [182] con modelos HDL detallados de los `Cores Mixtos`, además de evaluar los beneficios aportados. Actualmente el código de este *core* se encuentra disponible de forma libre en Internet [197].

Se trata de un *core soft* descrito en VHDL con dos `nano-procesadores` embebidos, hardware adicional e interfaces estándar. A través de la interfaz estándar puede ser integrado en sistemas SoPC con otros *cores* de diferente funcionalidad capaces de incluir otros procesadores. La función del *core* `FFR16` es la de leer de forma autónoma el primer fichero que se encuentre almacenado en el directorio raíz de una unidad de almacenamiento masivo IDE y con el formato de Microsoft FAT16 [198]. Internamente, el `FFR16` está estructurado en dos módulos reutilizables de forma separada: el *Host ATAPI Unit* (`HAU`) y el *FAT Processor Unit* (`FPU`) (véase figura 4.2).

[1] Módulos hardware-software especializados en el procesamiento de comunicaciones en redes telemáticas.



(a) ***Representación de alto nivel del*** `Core Mixto FFR16`**.**

(b) ***Módulos*** `HAU` ***y*** `FPU` ***del*** `FFR16`**.**

(c) ***Interfaces internos y externos.***

(d) ***Particiones internas de los módulos*** `HAU` ***y*** `FPU`**.**

Figura 4.2.: *Core* `FFR16`.

El módulo *Host ATAPI Unit* lee los sectores *Logic Block Address* (LBA) solicitados a través de su interfaz estándar. El programa del mismo controla las señales IDE y el `nano-procesador` embebido controla todo el protocolo ATA/ATAPI [199].

El módulo *FAT Processor Unit* rastrea la unidad lógica FAT buscando el primer fichero almacenado en la misma. Esta tarea no es trivial debido a la complejidad de los formatos FAT, ya que se requiere realizar computaciones de 32 bits y emplear información dispersa por distintas zonas (*Master Boot Record*, cadena de *clusters*, FAT, etc.).

Como alternativa al uso de `Cores Mixtos`, otros sistemas embebidos incluyen de potentes procesadores de propósito general que pueden realizar el procesamiento de la FAT y del protocolo ATA como una tarea más, siendo la carga de computación aceptable. Para esta solución convencional hay disponibles *cores* para realizar la interfaz de estos micros con el dispositivo IDE, tanto de origen académico para el modo ATA/ATAPI-5 [200] y el modo PIO [201], como de orientación comercial [202, 203].

La aplicación de este `Core Mixto` se centra en sistemas donde la existencia de un módulo independiente que realice todo el procesamiento y que finalmente entregue los datos al bus *on-chip*, suponga mejoras tanto en aprovechamiento de área, como en rendimiento y simplificación del diseño. Esta situación es muy común en los actuales diseños SoPC. Dentro de este tipo de sistemas encontramos aplicaciones embebidas con requerimientos de alto rendimiento y procesamiento paralelo, donde el tráfico en el bus *on-chip* debe ser optimizado: grabadoras DVD autónomas basadas en SoC [204], impresoras con soporte de *Compact Flash* para almacenar la información que se desea imprimir, etc.

El uso de un *core* como el `FFR16` en estas aplicaciones simplifica la arquitectura facilitando el cambio del elemento origen de datos (comunicaciones) [205] dada su naturaleza modular e interfaces estándar incorporados. Pero además, la cuidada optimización en área de los

Tabla 4.1.: Coste de implementación de un sistema de computación ATA+FAT para distintas alternativas SoPC.

Implementación	IDE Interface Core [2]	Microprocesador Soft-core	Subsistema ATA+FAT completo
Leon Sparc[3]	305 slices[4]	4.170 slices	4.475 slices
Microblaze[5]	228 slices	685 slices	913 slices
FFR16	258 slices	0 slices	258 slices

`nano-procesadores` y del hardware adicional permite obtener sustanciales ahorros en área de silicio. En la publicación [196], el autor desarrolla la comparativa resumida en la tabla 4.1. En este trabajo comparamos los resultados de implementación sobre dispositivos FPGA de dos soluciones convencionales SoPC (con procesadores de propósito general y *cores* interfaz IDE no inteligentes) y el `Core Mixto FFR16`. Para las tres soluciones se considera que se consigue un *throughput* de datos similar al emplearse el modo PIO IDE, centrándose la comparativa en los recursos necesarios para componer el sistema de computación ATA+FAT global.

La opción que usa el procesador Microblaze [208] de propósito general de Xilinx requiere más de 3.5 veces el número de *slices* de FPGA (Virtex) que el `FFR16`. Es más, si se pretende comparar con la opción SoPC convencional realizada en torno al procesador Leon Sparc, el ahorro es más evidente (la solución del `FFR16` sería más de 16 veces menor).

Este ejemplo ilustra cómo la aplicación de estas arquitecturas autónomas, además de habilitar el procesamiento paralelo y permitir desahogar los buses internos (especialmente para los sistemas con topologías de bus compartido), resuelven aplicaciones complejas a pesar de la simplicidad estructural de los `nano-procesadores` embebidos.

[2] El *core* OCIDEC es una interfaz IDE disponible de forma libre y pública en [206]. Dispone de tres versiones para implementar diferentes tipos de accesos. La implementación seleccionada para este análisis ha sido la más simple (OCIDEC-1). Soporta modo básico de acceso IDE (PIO) de igual forma que el modo que soporta el `HAU`. El *core* original OCIDEC dispone de una interfaz estándar Wishbone [154]. Para la conexión con los procesadores Leon Sparc y Microblaze se han añadido los *bridges* con los buses AMBA [61] y CoreConnect™ [147] respectivamente [207].

[3] La CPU *soft* LEON implementa un procesador de 32-bit con un juego de instrucciones acorde con el estándar Sparc V8. Está especialmente diseñado para su uso en aplicaciones embebidas y está disponible bajo licencia GNU LGPL en [67]. La interfaz con el bus del sistema es compatible con la especificación AMBA.

[4] El Microblaze es un procesador RISC de 32-bits. La interfaz con el bus del sistema es CoreConnect™ compatible.

[5] Una *slice* de Virtex está compuesta por dos LUTs de 4 entradas, dos elementos de almacenamiento (biestables) y lógica adicional (acarreo, control, etc.). El dato en *slices* es una aproximación puesto que no se especifica el grado de aprovechamiento de las mismas.

5. Análisis de las alternativas actuales para el control de la auto-reconfiguración parcial dinámica en sistemas CSoPC

En la tabla 2.1 del apartado 2.4 se presenta un resumen de los sistemas reconfigurables donde se plasma una muestra representativa de las diferentes propuestas realizadas en esta área. Muchos de estos sistemas son arquitecturas académicas y experimentales, mientras que otros no soportan la reconfiguración parcial dinámica.

El estado del arte presentado en las secciones anteriores muestra una clara evolución de los sistemas digitales hacia diseños basados en *cores* [9, 209], empleándose de forma generalizada las FPGAs comerciales de grano fino para realizar la implementación física de los mismos frente a diseños basados en tecnología ASIC.

La aplicación de la reconfiguración parcial dinámica a los diseños aporta un nuevo grado de libertad que puede implicar una nueva revolución en la forma de entender los sistemas electrónicos. Especialmente si se considera la posibilidad de que la *metacomputación* se realice por el propio sistema integrado en la FPGA, modo de reconfiguración denominado auto-reconfiguración.

Por ello, la investigación sobre sistemas que controlen la aplicación de la reconfiguración parcial dinámica en diseños basados en *cores* sobre FPGAs comerciales de grano fino ha copado gran parte de las aportaciones realizadas en esta área de conocimiento durante los últimos años. En el apartado 3.5.3 se han presentado las características de seis propuestas diferentes en esta área concreta. Todas estas propuestas, salvo la de K. Danne que se conecta a un *host* externo mediante un bus de expansión PCI, son sistemas que implementan la auto-reconfiguración. En las cinco restantes, las computaciones relacionadas con la reconfiguración se realizan en el propio dispositivo donde se integra el sistema.

A continuación se comparan, utilizando la tabla 5.1, para cada una de estas cinco alternativas, siete aspectos que se han considerado críticos para el control de la reconfiguración parcial dinámica y de la auto-reconfiguración:

- **Especificación de bus:** Especificación estándar con la que es compatible la propuesta de control.
- **Controlador de reconfiguración:** Tipo y modelo de procesador interno utilizado para realizar las tareas de *metacomputación*.
- **Soporte para reconfiguración *intra-task*:** En este campo se indica si en el correspondiente sistema se especifica un modo de control específico para que se pueda aplicar de forma segura reconfiguración de tipo *intra-task* (modificación parcial de un IP-Core).

- **Soporte para reconfiguración *inter-task*:** En esta columna se indica para cada alternativa si provee de algún método para controlar la aplicación de la reconfiguración de tipo *inter-task* (cambio completo de IP-Cores).

- **Soporte para reconfiguración software de `Cores Mixtos`:** Se indica con un *Sí* en esta columna las propuestas que dispongan de un mecanismo adecuado para poder cambiar mediante reconfiguración el programa de los `nano-procesadores` embebidos en IP-Cores.

- ***Handshake* de reconfiguración:** Se califica si se provee un mecanismo de *handshake* para controlar la reconfiguración entre el controlador de reconfiguración y el módulo que esté siendo objeto de la misma.

- **Desconexión del *core* durante reconfiguración:** Se indica para cada propuesta si incluye una infraestructura adecuada para desconectar del bus *on-chip* interno de la FPGA el *core* que esté siendo configurado.

En la primera fila se resumen las características de la propuesta experimental promocionada por Xilinx, *Self-Reconfiguring Platform* [157, 158] (SRP). Es compatible con la especificación de bus CoreConnect^{TM} y soporta la reconfiguración de los recursos de las CLBs, no admitiendo el cambio dinámico de otros elementos como los bloques de memoria RAM. La posibilidad de reconfiguración con una granularidad tan pequeña habilita los modos de reconfiguración *intra-task* e *inter-task*.

Aunque la plataforma SRP provee un subsistema software potente y bien estructurado, a nivel hardware propone una topología en la que se sobrecarga el bus compartido del sistema con gran cantidad de accesos. Esto es debido principalmente a que el controlador accede a través del bus tanto a la interfaz de configuración interna como a la memoria caché de configuración.

SRP no define un sistema de señalización que permita aplicar la reconfiguración dinámica a *cores* del CSoPC de forma independiente y segura, no soportando, por tanto, una reconfiguración *intra-task* de *cores* con `nano-procesadores` embebidos. También debe destacarse como una de las desventajas de la plataforma SRP, la necesidad de un procesador de propósito general de 32 bits. En muchas aplicaciones no se justifica el uso de un procesador de estas características y en especial en el caso del microprocesador *soft* Microblaze utilizado, ya que requiere gran cantidad de recursos generales de la FPGA para su implementación.

En la segunda fila de la tabla, se califica en los siete apartados la propuesta de R.J. Fong [145]. En la misma línea que SRP, esta alternativa implica un coste elevado de recursos necesarios para componer la infraestructura para el control de la reconfiguración. Este sobrecoste es debido principalmente al controlador de reconfiguración, el cual requiere del orden del 64 % de los recursos lógicos de un dispositivo Virtex (VC2V1000). Esta reducida optimización hace difícilmente justificable la mejora obtenida por la aplicación de la auto-reconfiguración parcial dinámica en un sistema en el que se emplee esta propuesta. Especialmente si se tiene en cuenta que la *metacomputación* que se realiza se limita a la aplicación de la configuración.

En la infraestructura planteada no se define un sistema de control de la reconfiguración que permita la modificación selectiva de zonas de lógica, elemento básico para poder trasladar esta tecnología a sistemas basados en *cores*. Únicamente soporta la reconfiguración

Tabla 5.1.: Comparación entre las alternativas actuales para la aplicación de Reconfiguración Parcial Dinámica a plataformas FPGA comerciales basadas en IP-Cores.

Denominación	**Especificación bus**	**Controlador reconf.**	**Soporte para reconf. *intra-task***	**Soporte para reconf. *inter-task***	**Soporte para reconf. software** `Cores Mixtos`	***Handshake* para reconf.**	**Desconexión del *core* durante la reconf.**
Blodget et al. (SRP) [157]	CoreConnect™	Procesador propósito general (MicroBlaze/-PowerPC)	Sí	Sí	No	No	No
Fong et al. [145]	No	Procesador específico	Sí	No	No	No	No
Ullmann et al. [162]	Específico (NoC)	Procesador propósito general (MicroBlaze)	No	Sí	No	Sí	Sí
Möller et al. (FiPRE)[168]	Específico	Procesador propósito general (R8R) (modificado)	No	Sí	Sí	Sí	Sí
Walder et al. [172]	Específico	Procesador propósito general (MicroBlaze)	No	Sí	No	Sí	Sí

basándose en los elementos *Tbufs* tecnológicamente dependientes de Xilinx [98]. Por lo tanto, tampoco es posible afrontar mediante esta propuesta un cambio selectivo de hardware y software de los `nano-procesadores` embebidos en `Cores Mixtos`.

Tanto la propuesta de M. Ullmann [162] (fila 3) como la FiPRe de L. Möller [168] (fila 4), aportaciones en esta área presentadas en el año 2004, proponen unos mecanismos para abordar la reconfiguración parcial dinámica muy elaborados y potentes. Sin embargo, presentan unas topologías de bus específicamente diseñadas para sistemas reconfigurables dinámicamente (en la tabla se completa el campo de *Especificación bus* con el término *Específico*). Esto supone que los diseñadores de SoPC deben adoptar esa especificaciones concretas en sus diseños, tomándolas como base para la definición arquitectural de los sistemas, lo que implica cambiar sus metodologías y sistemas ya desarrollados. Realmente no aportan una extensión para que una especificación de interconexión de *cores* como CoreConnect^{TM} o Wishbone, utilizadas en el diseño SoC, admitan la reconfiguración parcial dinámica. Este planteamiento puede limitar su aplicabilidad real ya que reducen la capacidad de reutilización de elementos previamente diseñados.

El modelo de M. Ullmann dispone de un sistema de control de reconfiguración de los módulos reconfigurables lo que se ha indicado en el campo *Handshake para reconf.* con un *Si*. Esta característica permite realizar reconfiguraciones *inter-task* seguras, destacando además el uso de un sistema de comunicación NoC que permite que el bus siga funcionando, por lo que admite transferencias entre los distintos módulos mientras se produce la reconfiguración parcial.

Sin embargo, al obligar a realizar las reconfiguraciones mediante *slots* completos, este sistema no se puede beneficiar de la rápida reconfiguración aplicable en el caso de reconfiguración *intra-task* que implica *bitstreams* de tamaño muy reducido. Y en el caso de que esa reconfiguración *intra-task* se quiera aplicar a módulos con `nano-procesadores` embebidos, el sistema tampoco dispone de mecanismos para controlar el estado de esos procesadores durante el proceso de reconfiguración.

Estas últimas limitaciones aparecen también en la propuesta FiPRe. El cambio de unidades coprocesadoras se soporta mediante reconfiguración *inter-task* controlada. Además, hay que destacar que con la topología de bus propuesta, las *Bus-Macro* definen zonas estancas para los distintos módulos pero empleando líneas compartidas de bus. Esto supone que durante el proceso de reconfiguración el bus no sea accesible.

En la última fila de la tabla 5.1 se identifican las posibilidades del modelo reconfigurable de H. Walder. La morfología de esta propuesta, detallada junto con las anteriores en el apartado 3.5.3, implica al igual que en modelo reconfigurable de FiPRe zonas estancas para los módulos reconfigurables que bloquean el tráfico a través del bus en el caso de que estén siendo reconfiguradas. Estos módulos reconfigurables (*slots*) se cambian dinámicamente de forma completa, lo que implica que este sistema sólo admite reconfiguración *inter-task*. Por tanto, la reconfiguración del software de los `nano-procesadores` embebidos en los `Cores Mixtos`, al estar basada en la reconfiguración *intra-task*, tampoco está soportada por esta propuesta.

6. Objetivos

En el estado del arte presentado se realiza un recorrido por distintas alternativas existentes que hacen uso de lógica reconfigurable. Se estudian desde las propuestas que combinan secciones reconfigurables con secciones fijas en integrados ASIC y plataformas específicamente diseñadas para la experimentación, y se llega hasta las alternativas que dan soporte a las nuevas posibilidades tecnológicas de las FPGAs de grano fino, como la reconfiguración parcial dinámica y la auto-reconfiguración.

La exposición realizada sobre la investigación en las alternativas de lógica reconfigurable constata la evolución del diseño de sistemas electrónicos hacia diseños integrados en un único dispositivo [9, 110, 128, 210] basados en el uso de módulos anteriormente realizados o adquiridos a empresas especializadas, denominados IP-Cores. Para los casos en los que estos diseños se realizan empleando lógica reconfigurable (SoPCs), las investigaciones más recientes, resumidas en el apartado 3.5.3, proponen la aplicación de la reconfiguración parcial dinámica a los mismos.

Sin embargo, el análisis de las alternativas existentes para el control de la reconfiguración parcial dinámica desde los propios sistemas integrados en los dispositivos, sistemas auto-reconfigurables, pone de manifiesto ciertas carencias que dificultan la aplicación de la misma a los diseños realizados en base a *cores*:

- Los diseños basados en *cores* emplean especificaciones estándar para la interconexión de *cores* para facilitar la *reusabilidad* y la integración. Únicamente la propuesta SRP [157] utiliza una interfaz estándar, pero no para los *cores* reconfigurables, sino para el control de la reconfiguración.

- La reconfiguración *intra-task* en los casos que es aplicable es mucho más rápida y menos intrusiva en el diseño que la *inter-task*, ya que se mantiene la estructura del circuito que se está reconfigurando y se requieren unos *bitstreams* de configuración pequeños.

  La mayor parte de las propuestas actuales están orientadas a la reconfiguración *inter-task*. Esto implica *bitstreams* parciales largos y, por tanto, tiempos de reconfiguración grandes. En muchos casos, tras un análisis de las penalizaciones temporales y de consumo producidas por este proceso de reconfiguración [211], se concluye que es poco viable la aplicación del mismo.

- Las propuesta analizadas no contemplan en ningún caso la posibilidad de una reconfiguración *intra-task hardware/software* de *cores* que integren `nano-procesadores` junto con hardware adicional (`Cores Mixtos`). Tal y como se ha argumentado en el capítulo 4, este tipo de *cores* dotan a los diseños SoPC de una gran flexibilidad y potencia, disponiéndose de forma efectiva de sistemas multi-procesador en integrados accesibles a cualquier equipo de desarrollo, como son las FPGAs comerciales de grano fino.

- Todas las plataformas proponen una *metacomputación* centralizada en un procesador integrado en el propio dispositivo para disponer de sistemas reconfigurables. Esta centralización obliga a utilizar procesadores complejos que requieren gran cantidad de recursos lógicos.

6.1. Objetivo general

Teniendo en cuenta los argumentos anteriores, el objetivo principal de esta tesis es:

Definir una infraestructura de señales, protocolos y lógica que permita la aplicación segura y eficiente de la auto-reconfiguración parcial dinámica en los diseños Multi-procesador CSoPC (MCSoPC) realizados sobre FPGAs comerciales de grano fino.

Partiendo de una arquitectura SoPC genérica, se debe definir un modelo multi-procesador capaz de soportar la auto-reconfiguración parcial dinámica de forma controlada según unos mecanismos definidos en esta tesis.

A este modelo inicial se le añaden los elementos necesarios para el control de los distintos procesos involucrados en una auto-reconfiguración parcial dinámica en un diseño con *cores* que puedan integrar `nano-procesadores` funcionando en paralelo.

La aplicación de la reconfiguración parcial dinámica a los diseños SoPC se encuentra en una fase embrionaria. Los flujos de diseño de los distintos fabricantes de lógica reconfigurable que soportan este tipo de reconfiguración requieren ser modificados y completados con nuevas aplicaciones para poder afrontar sistemas multi-procesador. Por tanto, para poder aplicar el modelo y el sistema de control planteado de forma teórica, se debe desarrollar un entorno para diseño de MCSoPCs. Este entorno incluye diferentes elementos heterogéneos, como son un compilador, módulos de lógica específicos, *scripts* de automatización del diseño, *Bus-Macros* especiales, etc. Debido a esta heterogeneidad, el término utilizado es *framework*.

Para realizar la validación de la propuesta se han diseñado varias plataformas con FPGAs de grano fino de forma que se pueda analizar la infraestructura de control con distintos tipos de reconfiguraciones parciales (*intra-task* e *inter-task*).

El sistema en su conjunto, compuesto por el modelo, el sistema de control y las herramientas asociadas, se ha denominado `Tornado`.

6.2. Objetivos parciales

Para cumplir el objetivo general propuesto en esta tesis se determinan una serie de objetivos parciales. Los distintos elementos definidos y desarrollados en cumplimiento de estos objetivos parciales completarán el sistema `Tornado`:

- **Especificación de una señalización para el control de la reconfiguración parcial dinámica de sistemas multi-procesador**. Para poder utilizar la reconfiguración parcial dinámica en los diseños SoPC implementados en FPGAs no es suficiente con que a nivel tecnológico el dispositivo la admita. Además, se requieren mecanismos de control de la reconfiguración parcial a nivel de la aplicación [87]. De esta forma se podrán modificar secciones hardware y software de los *cores* de forma segura.

- **Definición de un conjunto de fórmulas que permitan modelar la infraestructura de control de la auto-reconfiguración**. Se deben establecer un conjunto de fórmulas que permitan conocer de forma anticipada a la implementación a bajo nivel de un determinado diseño, el coste en recursos lógicos de la infraestructura `Tornado` y los retardos involucrados en el proceso de auto-reconfiguración. De esta manera se dispondrá de una herramienta de análisis útil para las fases de diseño a nivel de sistema. La validez de las estimaciones realizadas con está fórmulas se deberá contrastar con resultados obtenidos de ensayos de implementación y rutado de un conjunto de diseños.

- **Especificación del controlador de reconfiguración compatible `Tornado`.** Una de las bases en la propuesta del sistema `Tornado` es la distribución de la *metacomputación* entre distintos *cores* del sistema, con el fin de reducir la complejidad de la infraestructura añadida para el control y aplicación de la reconfiguración parcial dinámica.

  Con esta orientación se debe definir un controlador de reconfiguración adecuado, desarrollándose en la fase de validación una versión compatible con la especificación estándar de bus Wishbone [154] y parametrizable de forma automática mediante el *framework* `Tornado`.

- **Especificación y desarrollo de un `nano-procesador` compatible con `Tornado`.** La aplicación de la reconfiguración parcial dinámica a `Cores Mixtos` requiere de un `nano-procesador` *soft* con soporte para las señales de control de reconfiguración definidas en `Tornado` y con instrucciones específicas para el control del mismo durante el proceso de reconfiguración.

- **Desarrollo de un ensamblador multi-contexto para el `nano-procesador` compatible Tornado.** Con el fin de facilitar la integración de varios `Cores Mixtos` multicontexto en un sistema CSoPC se debe crear una herramienta para realizar el ensamblado del código para los `nano-procesadores`. Mediante directivas incluidas en el código se podrá separar el código asociado a cada contexto, generando tanto el código HDL para la integración del mismo en las memorias embebidas de la FPGA, como para la generación automática de un controlador de reconfiguración optimizado.

- **Definición de los flujos de diseño para reconfiguración intra-task e inter-task.** El *framework* `Tornado` y la infraestructura de elementos necesarios para aplicar el sistema de control deben integrarse junto con las herramientas de diseño de los fabricantes de dispositivos reconfigurables. Para ello se establecerán unos diagramas de flujo generales diferenciando los dos tipos de reconfiguración indicados.

- **Especificación y construcción de plataformas de verificación.** Con el fin de validar la propuesta `Tornado`, se deben desarrollar plataformas experimentales sobre prototipos con FPGAs reconfigurables dinámicamente. Con estos diseños se realizarán implementaciones de `Tornado` optimizadas para una determinada tecnología y para una especificación de bus *on-chip* estándar concretas.

Parte II.

Sistema Tornado

7. Introducción y justificación

G. Martin en el capítulo *"The History of the SoC Revolution"* [9] resalta el éxito actual y la importancia futura de los diseños realizados con *cores* sobre plataformas reconfigurables basadas en FPGAs.

Algunas de las nuevas FPGAs con las que se están implementando estos diseños admiten a nivel tecnológico la reconfiguración parcial dinámica. Sin embargo, para disponer de sistemas basados en *cores* que se beneficien de este modo de reconfiguración no es suficiente con que la FPGA lo admita.

Se requiere de un sistema que controle la reconfiguración parcial en el propio diseño, a nivel de aplicación. No es viable realizar la reconfiguración de una zona del circuito si no se puede desconectar ésta del resto del sistema, congelar los procesadores internos o actuar sobre los *pines* asignados al circuito que se configura.

Se debe considerar, además, que es de interés creciente la posibilidad de que sean los propios diseños configurables basados en *cores* los que decidan las reconfiguraciones aplicables sobre ellos mismos. De esta forma, el control de la reconfiguración queda integrado también en la propia FPGA. Este modo de operación se denomina auto-reconfiguración.

Con el objeto de disponer de una solución global que resuelva estos aspectos, se presenta en esta tesis el sistema *Tornado*. Tal y como se ha establecido en los objetivos de la misma, mediante este trabajo se pretende definir un sistema de control de la auto-reconfiguración dinámica y parcial para sistemas multi-procesadores implementados sobre dispositivos reconfigurables comerciales (FPGAs). El término empleado para definir concretamente a estos sistemas que admiten su modificación parcial y dinámica, es el de MCSoPC (*Multiprocessor-Configurable-System-on-a-Programmable-Chip*).

En los siguientes capítulos se realiza un planteamiento teórico y generalizado del sistema `Tornado`, definiéndose los siguientes elementos:

- **Modelo reconfigurable multi-procesador CSoPC compatible `Tornado`:** Se especifican en el mismo las características que deben tener los sistemas basados en *cores* realizados con FPGAs parcial y dinámicamente reconfigurables, para que admitan el control de la reconfiguración mediante el sistema `Tornado`. Estas especificaciones abarcan aspectos tales como la distribución de la *metacomputación* o las características que deben tener los `nano-procesadores` que puedan ser reconfigurados dinámicamente.

- **Infraestructura de control `Tornado`:** Se definen en esta infraestructura los elementos necesarios (controlador de reconfiguración, señales de control, interfaces de reconfiguración, etc.) para poder realizar la reconfiguración controlada en un sistema basado en el modelo anterior.

- **Modelado de la infraestructura `Tornado`:** Se caracteriza el coste de la infraestructura `Tornado`, tanto en recursos lógicos como en tiempo, mediante ecuaciones y parámetros obtenidos a partir de datos empíricos. Mediante esta caracterización se

desea disponer de unas fórmulas que permitan estimar el coste de hacer compatible con `Tornado` un determinado SoPC, antes incluso de que se desarrolle a bajo nivel.

Este planteamiento teórico se lleva a la práctica en el apartado de esta tesis correspondiente a la validación del sistema `Tornado`. Para analizar la viabilidad de este sistema se realiza la integración del mismo en tres plataformas. Para poder afrontar esta tarea, se han desarrollado los conceptos teóricos sobre una tecnología concreta, lo que ha supuesto la creación de diversas herramientas software y una infraestructura específica para una tecnología. El conjunto de herramientas desarrolladas para asistir al diseño se denominan *framework* `Tornado`.

En el cápítulo 5 de la parte I de esta tesis, se ha realizado un análisis de las alternativas actuales para el control de la auto-reconfiguración parcial dinámica en sistemas CSoPC. Se han comparado cinco propuestas cuyo objetivo es integrar la reconfiguración parcial dinámica en sistemas basados en *cores*, contrastándose sobre siete aspectos considerados claves en aplicaciones reales, como son la disponibilidad de un *handshake* para la desconexión del módulo que esté siendo reconfigurado o la posibilidad de cambiar el contexto software de un `nano-procesador` mediante reconfiguración.

A partir de este análisis que pone de manifiesto las carencias existentes en las propuestas referenciadas, se establecen las bases del sistema `Tornado`. De está forma, se busca definir una alternativa lo más adecuada posible a la realidad actual del diseño de SoPC y que contemple los aspectos demandados en esa comparativa. Estas bases son:

- Los diseños basados en *cores* emplean especificaciones estándar para la interconexión de *cores* de forma que se facilite la reutilización y la integración de los mismos. `Tornado` estará orientado hacia arquitecturas basadas en especificaciones estándar para interconexión de IP-Cores.

- La reconfiguración *intra-task*, en los casos que sea aplicable, es mucho más rápida y menos intrusiva en el diseño que la reconfiguración *inter-task*. El sistema `Tornado` deberá incluir los mecanismos necesarios para dar soporte a la reconfiguración *intra-task*.

- Las propuestas analizadas no contemplan en ningún caso la posibilidad de una reconfiguración *intra-task hardware/software* de *cores* que integren `nano-procesadores` junto con hardware adicional (`Cores Mixtos`). Será un elemento básico en `Tornado` dar soporte tanto teórico (señalización, modelo, etc.) como práctico (ensamblador, *scripts*, etc.) a la reconfiguración *intra-task hardware/software* de los `Cores Mixtos`.

- La evolución hacia plataformas capaces de gestionar su propia reconfiguración, integrando en el mismo dispositivo tanto la *metacomputación* como el acceso interno al sistema de reconfiguración (sección 3.5), haría poco viable una propuesta que no contemplase la posibilidad de auto-reconfiguración. `Tornado` se estructurará de forma que admita en el propio integrado la *metacomputación*, planteando además una descentralización de la misma, de forma que se reduzca la penalización en tiempo y área generada por la incorporación de una infraestructura adicional [212].

Un sistema con `nano-procesadores` trabajando en paralelo, integrados en distintos módulos, y a los cuales se puede aplicar en tiempo de funcionamiento y de manera independiente

cambios de hardware y software, podría visualizarse como una *tormenta* donde los distintos torbellinos integrantes de un tornado, con ejes de rotación independientes que giran en torno al centro del tornado, van actuando sobre los distintos `Cores Mixtos`. Este símil ha inspirado el nombre de `Tornado`.

Un sistema generalizado e innovador como el propuesto requiere la definición de numerosos términos que designen a los nuevos elementos. En la tabla 7.1 se resumen los términos específicos que se presentan en los siguientes capítulos y los que, aún no siendo propios de `Tornado`, tienen especial relevancia.

Para facilitar la identificación de los términos que específicamente se definen en este trabajo, se utiliza el tipo de letra `courier` a lo largo de este documento. Además, todos los acrónimos empleados para identificar los elementos que integran la infraestructura `Tornado` comienzan con la letra `T`.

Tabla 7.1.: Términos utilizados en el sistema Tornado.

Acrónimo	Denominación	¿Específico de Tornado?	Descripción
IF	InterFace	No	Una interfaz hardware de un *core* es el circuito que permite la comunicación de éste con otros módulos.
IP-Core	*Intelectual Property Core*	No	Módulo hardware pre-diseñado integrable en diseños SoC. Se emplean diversos términos para denominarlos como: *core*, módulo IP, IP, etc.
-	Sistema Tornado	Sí	La propuesta (modelo, infraestructura y modelado) para controlar la reconfiguración parcial dinámica en sistemas CSoPC multi-procesador presentada en esta tesis.
-	Modelo Tornado	Sí	Caracterización de un sistema SoPC para que pueda ser ampliado a CSoPC mediante Tornado.
-	Infraestructura Tornado	Sí	La componen los elementos definidos en el modelo Tornado para la reconfiguración (controlador, interfaces, señales, protocolos, etc.).
-	Modelado Tornado	Sí	Parámetros y ecuaciones que permiten estimar el impacto de la aplicación de Tornado en un SoPC.
-	*Framework* Tornado	Sí	Conjunto de herramientas desarrolladas para utilizar Tornado con un determinada tecnología.
-	Nano-procesador	No	*Core soft* de una pequeña unidad procesadora.
-	Core Mixto	No	*Core* que integra al menos un nano-procesador, hardware adicional y una interfaz estándar.
TnP	Tornado nano-Processor	Sí	Nano-procesador capaz de admitir un cambio de contexto controlado por Tornado
T-Core	Tornado compatible Core	Sí	IP-Core con los elementos necesarios para admitir una reconfiguración parcial controlada por Tornado.
TnP-Core	Tornado nano-Processor Core	Sí	Core Mixto compatible con Tornado (el nano-procesador que integra es un TnP).
TIF (S)	Tornado InterFace (Slave)	Sí	Interfaz incluida en los módulos susceptibles de ser reconfigurados dinámicamente para compatibilizarlos con Tornado (T-Cores).
TIF (M)	Tornado InterFace (Master)	Sí	Interfaz del controlador TBC para aplicar el sistema de control Tornado.
TBC	Tornado Basic Controller	Sí	Versión básica del controlador de reconfiguración definido en la infraestructura Tornado.
TBM	Tornado Bus-Macro	Sí	Circuito pre-emplazado y pre-rutado para *envolver* IP-Cores compatibilizándolos a una determinada especificación.
RM IF	Reconfiguration Media InterFace	Sí	Interfaz del controlador de reconfiguración con el sistema de carga de la FPGA.
SRI	System Reconfiguration Information	Sí	Software del controlador de reconfiguración TBC.
CRW	Configuration Request Word	Sí	Palabra de petición de reconfiguración.

8. Modelo reconfigurable multi-procesador

En este capítulo se presenta el modelo generalizado de un sistema basado en *cores*, implementado en una FPGA reconfigurable dinámicamente y que controle la auto-reconfiguración mediante el sistema `Tornado`.

Para obtener este modelo se parte de un modelo de diseño SoPC basado en *cores* genéricos. A esta estructura original se añade la infraestructura `Tornado`, detallándose a continuación cada uno de los elementos que la componen y el protocolo definido para el control de la reconfiguración.

La figura 8.1 muestra el diagrama de bloques de una arquitectura generalizada de un diseño SoPC basado en *cores*. Este sistema no tiene ningún mecanismo para controlar la reconfiguración parcial dinámica, no es *configurable* aunque se implemente en una FPGA. Esta compuesto por un número z de *cores* genéricos (IP-Cores), los cuales pueden ser microprocesadores, DSPs, periféricos, u otros.

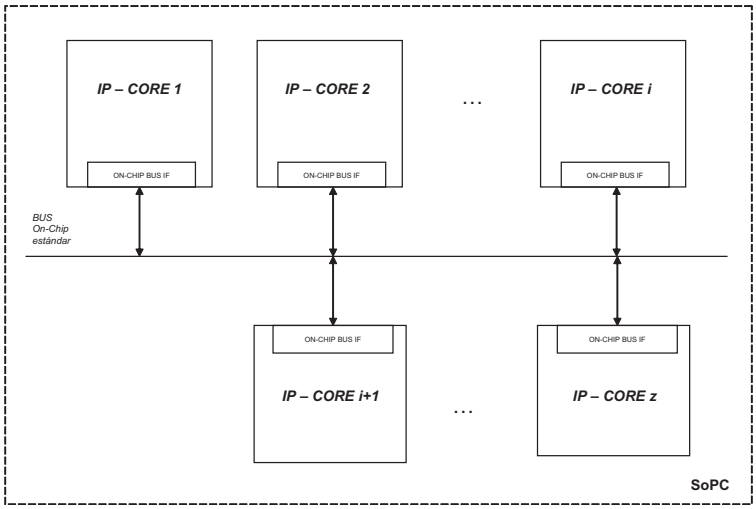


Figura 8.1.: Arquitectura generalizada de un SoPC basado en *cores*.

Todos ellos disponen de una interfaz estándar para la conexión con el bus. La topología de interconexión entre módulos está restringida únicamente por la especificación para interconexión de IP-Cores que se haya elegido. En el estado del arte se han presentan varios ejemplos de estas especificaciones, pudiéndose constatar cómo hay especificaciones que admiten varias topologías de bus (bus compartido, punto a punto o *crossbar* entre otras) [150, 151, 152, 153], además de admitir varios buses con distintas topologías en un mismo SoPC [205].

En este caso la topología de interconexión seleccionada para la representación del modelo

es la de bus compartido, aunque como se expone posteriormente, el modelo de reconfiguración obtenido se puede aplicar a las otras topologías que la especificación utilizada permita.

Aunque la FPGA sobre la que funcione el diseño admita tecnológicamente una reconfiguración parcial dinámica, se requieren mecanismos desde la aplicación que controlen el estado de los *cores* que se estén reconfigurando (desconexión del bus, control de los microprocesadores embebidos, etc.). Por ello se define como *cores* `Tornado Compatible` (`T-Cores`), a los módulos con capacidad de ser controlados en el proceso de reconfiguración, *intra-task* o *inter-task*, mediante `Tornado`. Estos `T-Cores` disponen de una interfaz denominada `Interfaz Tornado (Tornado IF)`[1], a través de la cual se realiza el control del estado del *core* durante el proceso de reconfiguración.

En la figura 8.2 se representa la arquitectura generalizada de un sistema CSoPC con un número n de `T-Cores`, que admiten la reconfiguración parcial dinámica, y un número z de *IP-Cores* no compatibles con `Tornado` (*convencionales*).

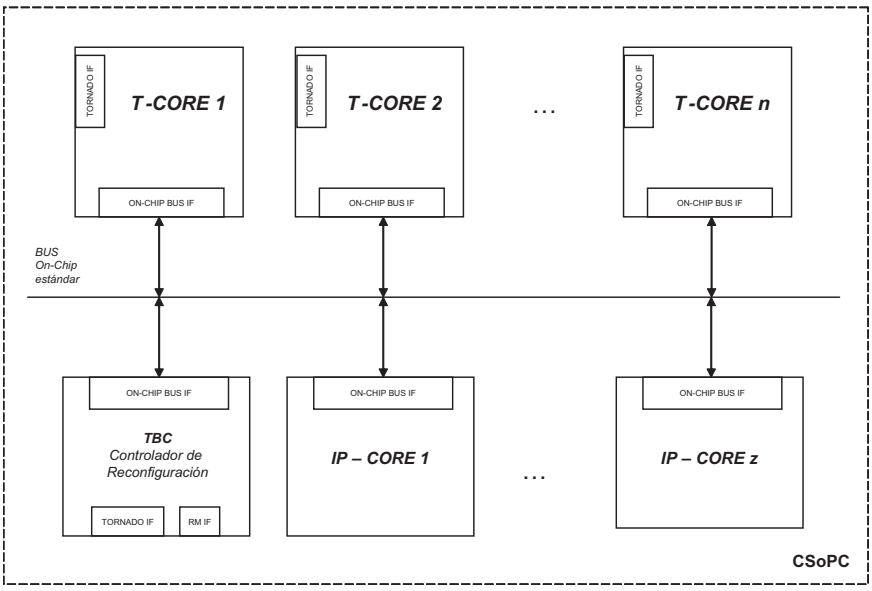


Figura 8.2.: Arquitectura generalizada de un *Configurable*-SoPC compatible `Tornado`.

Los *cores convencionales* tienen una interfaz para la conexión al bus *on-chip* normalizado que se esté empleando, pero no tienen los mecanismos específicos para soportar la reconfiguración. El hecho de que se plantee un modelo en el que puedan coexistir tanto *cores* compatibles con `Tornado` como los que no lo son, plantea diversas ventajas:

- Posibilidad de utilizar los *cores* disponibles para la especificación normalizada que se esté empleando.
- Facilidad para la integración de `Tornado` en diseños ya realizados.
- Generalidad en cuanto al tipo de *core*. Los *cores* permiten definir la sección estática del diseño implementado en la FPGA. Al tratarse de una sección fija, estos *cores* podrán

[1] En el apartado 9.1.2, se presentan las dos variantes de esta interfaz, la interfaz `Tornado` esclava (`TIF (S)`) y la maestra `TIF (M)`).

Tabla 8.1.: Características generales de los controladores de reconfiguración de las diferentes propuestas.

Denominación	Tipo de controlador de reconfiguración	Nombre del controlador	Tamaño aproximado
Blodget (SRP) (I)[157]	Procesador de propósito general	MicroBlaze	685 *slices Virtex*[2][213]
Blodget (SRP)(II)[157]	Procesador de propósito general	PowerPC	*Core hard* integrado[3]
Fong [145]	Procesador específico	Controlador de reconfiguración	3.276 *slices Virtex*
Danne [37]	*Host* externo	-	-
Ullmann [162]	Procesador de propósito general	MicroBlaze	685 *slices Virtex*[213]
Möller (FiPRE) [168]	Procesador de propósito general modificado	R8NR	1.237 *slices Virtex*[4]
Walder [172]	Procesador de propósito general	MicroBlaze	685 *slices Virtex* [213]

ser tanto *cores soft* como *cores hard*, incluyéndose en este último grupo de *cores* a los procesadores integrados en el silicio de la FPGA.

- Modularidad. Un *core* conectado al bus a través de su interfaz estándar puede disponer de otra interfaz mediante la cuál se conecte a otro bus estándar (incluso con otra topología), donde se podrán situar otros *cores* [205].

En el apartado 3.1 se presenta el concepto de *metacomputación*. En este término se engloban las acciones necesarias para determinar la configuración que se debe aplicar en cada situación a cada sección parcialmente reconfigurable, incluyendo además en algunas propuestas la composición dinámica del *bitstream* parcial [18].

En la mayoría de las alternativas presentadas en el estado del arte, la *metacomputación* está centralizada en un único microprocesador, denominado controlador de reconfiguración. En la tabla 8.1 se presentan las características más relevantes de los controladores empleados por las plataformas para auto-reconfiguración analizadas en el capítulo 5.

La plataformas de Blodget, Walder y Ullmann utilizan microprocesadores de propósito general para realizar el control y el procesamiento de la reconfiguración. La ventaja que supone esta opción es que la *metacomputación* se puede considerar como una tarea más que realiza el procesador, pudiendo intercambiar fácilmente información con las tareas software propias de la aplicación.

Sin embargo, en el caso de que se utilice un procesador de propósito general con la única función de controlar la reconfiguración, la cantidad de recursos de la propia FPGA requeridos para la implementación del mismo, hacen poco rentable la aplicación de la auto-reconfiguración parcial dinámica al SoPC.

[2]Una *slice* de Virtex está compuesta por dos LUTs de 4 entradas, dos elementos de almacenamiento (biestables) y lógica adicional (acarreo, contol, etc.). El dato en *slices* es una aproximación, puesto que no se especifica el grado de aprovechamiento de las mismas.

[3]El *core* procesador PowerPC 405D5 es una implementación *hard* en 0.13 micras del *core* PowerPC 405D4 de IBM utilizado en la biblioteca de *cores* IBM Blue Logic™ .

[4]Los recursos lógicos requeridos para la implementación son 2.473 LUTs y 1.761 biestables. Para unificar las unidades en la comparación se supondrá que se realiza un aprovechamiento máximo de las *slices*, con lo que como mínimo se requerirían 1.237 *slices*.

En las propuestas de Fong y Möller se utilizan procesadores más especializados, pero la cantidad de recursos necesarios es todavía mayor que en los casos anteriores. La razón de esta necesidad de recursos se debe a la complejidad de los controladores.

El planteamiento de disponer de un único controlador de reconfiguración complejo donde se centralice la *metacomputación*, no resulta adecuado para los sistemas construidos en base a *cores* en general y con `Cores Mixtos` en particular (sistemas multi-procesador). La funcionalidades del sistema están distribuidas entre los *cores* que lo integran y, por tanto, las decisiones de aplicar de la auto-reconfiguración se determinan a partir de los procesamientos realizados por estos módulos distribuidos.

Un ejemplo de esta situación se presenta en la plataforma `Adaptive Multi-Transmitter` (`AMT`) [212] descrita en el apartado de validación del sistema propuesto. La plataforma está compuesta por varios `Cores Mixtos` transmisores de radio que generan directamente la señal de alta frecuencia (*Direct Digital Synthesis* -DDS-), los cuales pueden cambiar su modulación mediante reconfiguración parcial dinámica. La orden de reconfiguración enviada al controlador de reconfiguración, el cual está integrado en la propia FPGA, puede provenir de cambios en la calidad del canal y por órdenes recibidas de un procesador de comandos de comunicaciones. Los cambios de calidad del canal o el procesamiento de comandos provenientes de un *host* externo, se realizan en *cores* diferentes de la propia plataforma.

Por tanto, esos módulos o bien determinan los contextos aplicables a las secciones reconfigurables o bien sirven como conexión con otras fuentes de comando. Esto implica la distribución de la *metacomputación* entre las distintas unidades de procesamiento integradas en esos módulos.

Por ello, plantear un complejo controlador de reconfiguración donde se centralicen todas las operaciones de *metacomputación* o asociadas a ella, se contradice con la filosofía del diseño basado en *cores*. Especialmente cuando se integran `nano-procesadores`, disponiéndose en este caso de un sistema multi-procesador donde se pueden distribuir las tareas de *metacomputación* con gran flexibilidad.

En la figura 8.2 se representa el diseño SoPC original de la figura 8.1 adaptado para poder admitir auto-reconfiguración. Además de los `T-Cores` y *cores* no compatibles con `Tornado`, se incluye un *core* controlador de reconfiguración conectado al bus *on-chip*. Este controlador se denomina `Tornado Basic Controller` (`TBC`). Las pautas seguidas para definir sus características y operatividad se basan en la argumentación anterior, buscando la mayor simplicidad posible en el mismo.

La función de este controlador, que se describe en detalle en el apartado 9.2, es la de aplicar las peticiones de reconfiguración parcial dinámica que reciba desde el bus *on-chip* estándar. Estas peticiones de reconfiguración son escritas por otros módulos del sistema (*metacomputación* distribuida), simplificándose de esta forma el controlador de reconfiguración.

Este planteamiento, muy adecuado al flujo de datos existentes en arquitecturas SoPC con transferencia de datos masiva entre distintos *cores* [214], permite definir un modelo reconfigurable muy flexible. En él, el *controlador de reconfiguración* es un *core* conectado al bus *on-chip* mediante una interfaz estándar adecuada a la especificación para la interconexión de *cores* que se esté empleando.

Uno de los objetivos de esta tesis es la definición de una infraestructura de reconfiguración que permita la modificación hardware/software de sistemas multi-procesador basados en

Cores Mixtos. Por ello, en el grupo de los T-Cores habrá también *cores* que integren nano-procesadores.

Para que estos nano-procesadores admitan el cambio de contexto software mediante reconfiguración controlada por Tornado, deberán tener unas características específicas (detalladas en el apartado 9.3, dedicado a estos elementos). Estas unidades de procesamiento reconfigurables se denominan Tornado nano-Processors (TnPs). El conjunto de T-Cores que integran procesadores TnPs, utilizando una arquitectura de Core Mixto, se identifican como TnP-Cores.

Teniendo en cuenta esta situación, se detalla en la figura 8.3 el CSoPC genérico anteriormente presentado en la figura 8.2. En este caso, se considera la posibilidad de que haya nano-procesadores distribuidos por el sistema, integrados en TnP-Cores. En la figura 8.3 se representa el modelo incluyendo en este caso los T-Cores con dos secciones diferenciadas: una sección con todos sus elementos representados con línea discontinua, la que corresponde al TnP embebido y a su programa integrado en memoria RAM dedicada; y otra sección, identificada como *hardware específico*, la que realiza el procesamiento hardware.

Por tanto, el concepto de T-Cores engloba tanto a módulos únicamente hardware como a módulos con sección de procesamiento software. En los T-Cores que únicamente dispongan de sección hardware, los elementos punteados no están implementados.

Como se puede observar en la figura 8.3, en este caso la interfaz Tornado IF se sitúa entre la sección del TnP y la del *hardware específico*. La razón por la que afecta a las dos áreas es que ambos elementos pueden requerir un control durante el proceso de reconfiguración.

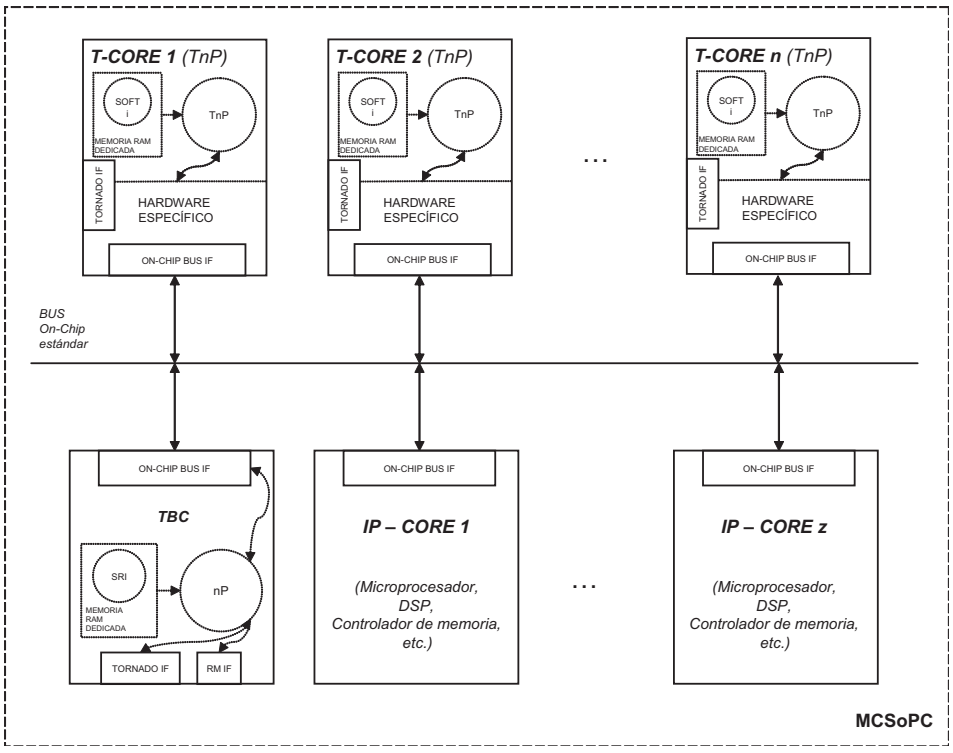


Figura 8.3.: Modelo Tornado: arquitectura generalizada de un CSoPC, compatible Tornado, con la posibilidad de integrar Cores Mixtos.

En esta figura también se detallan los bloques que integran el controlador de reconfiguración TBC. Tiene una arquitectura de Core Mixto, con el fin de obtener un alto grado de flexibilidad junto con una implementación eficiente en términos de área y velocidad.

Este modelo está generalizado para un número n de T-Cores y un número φ_i de posibles configuraciones parciales (contextos) para cada uno de ellos. Las configuraciones parciales pueden implicar un cambio completo de un T-Core, es decir, una reconfiguración *inter-task*; o bien un cambio del software o del hardware del circuito de un Core Mixto, por tanto, una reconfiguración *intra-task*.

Puesto que las reconfiguraciones son parciales, aplicadas de forma independiente a cada T-Core, el número total de configuraciones (Γ) que el sistema Tornado deberá gestionar, queda establecido mediante la ecuación 8.1. El número de variantes para un diseño concreto (Ω), teniendo en cuenta las distintas combinaciones de T-Cores con sus contextos, se expresa mediante la ecuación 8.2.

$$\Gamma = \sum_{i=1}^{n} \varphi_i \tag{8.1}$$

$$\Omega = \prod_{i=1}^{n} \varphi_i \tag{8.2}$$

Una vez definido el modelo Tornado, se detallan a continuación los elementos añadidos al SoPC genérico original para hacerlo compatible, los cuales componen la infraestructura Tornado.

9. Infraestructura de control de reconfiguración

Mediante el modelo `Tornado` se especifica la topología de un sistema MCSoPC auto-reconfigurable. En este capítulo se detallan los elementos que permiten el control de la auto-reconfiguración, los cuales componen la infraestructura `Tornado`. Estos elementos son:

- **Protocolos de control**: Señalizaciones que especifican el modo en el que se realizan las peticiones de reconfiguración y cómo se controlan los *cores* y los procesadores durante el proceso de reconfiguración.
- **Interfaces de control**: Secciones hardware de los distintos módulos involucrados en el proceso de reconfiguración encargadas de aplicar los protocolos de control establecidos.
- **Controlador de reconfiguración** (`TBC`): Módulo incluido en el diseño configurado en la FPGA encargado de recibir y aplicar las peticiones de reconfiguración.
- **Unidad procesadora Tornado nano-Processor** (`TnP`): `Nano-procesador` con capacidad de cambio de contexto mediante reconfiguración.
- **Tornado Bus-Macro** (`TBM`): Circuito pre-emplazado y pre-rutado para adaptar *cores* al sistema `Tornado` y permitir reconfiguraciones *inter-task*.

9.1. Protocolos de control

En el modelo MCSoPC `Tornado` propuesto, se distinguen dos acciones relacionadas con la auto-reconfiguración. La primera es la solicitud de reconfiguración al controlador `TBC` a través del bus *on-chip* por parte de los distintos *cores* donde se realiza la selección del contexto aplicable a cada `T-Core`; es decir, la *metacomputación*.

La segunda acción engloba la aplicación de los cambios de contexto solicitados, la reconfiguración. El control se realiza mediante un protocolo *handshake* gestionado por las interfaces `Tornado`, tal que permita el control de los `T-Cores` en general y de los `TnP-Cores` en particular, así como una aplicación segura de las reconfiguraciones.

9.1.1. Solicitud de cambio de contexto

La *metacomputación* en el modelo `Tornado` está distribuida entre diferentes *cores* del sistema. Por lo tanto, es necesario definir el método que se va a emplear para que el controlador de reconfiguración reciba la información sobre los cambios de contexto que se deben aplicar.

El mecanismo propuesto es el siguiente: *El controlador `TBC` recibe las peticiones de reconfiguración codificadas a través del bus on-chip*. De esta forma, cualquier *core* o elemento con capacidad de escribir en el bus *on-chip*, puede solicitar una reconfiguración de un `T-Core`.

En la figura 9.1 se representa el flujo posible de peticiones de reconfiguración. Estas parten tanto de *cores* como de `T-Cores`, siempre que estos dispongan de una interfaz maestra del bus *on-chip*, siendo escritas en el *core* controlador de reconfiguración `TBC`.

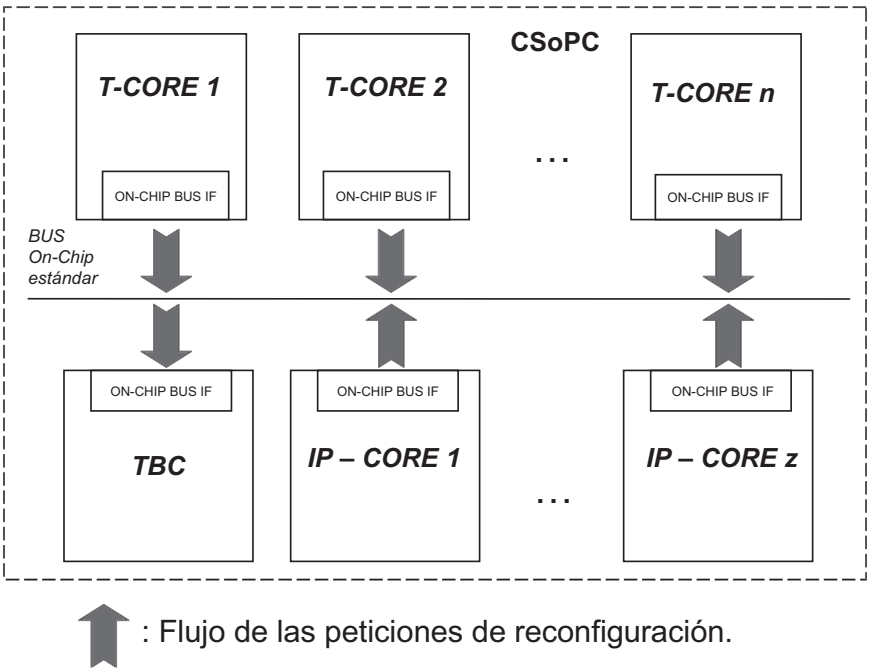


Figura 9.1.: Flujo de petición de reconfiguraciones. Diferentes fuentes de solicitud de reconfiguraciones para los `T-Cores`.

Para solicitar una determinada reconfiguración, el *core* que solicita un cambio de contexto escribe a través de su interfaz con el bus *on-chip* una palabra al controlador `TBC`. Esta palabra se denomina `Configuration Request Word` (`CRW`). La escritura de la palabra en el bus *on-chip* se realiza según indique la especificación estándar que siga el bus. La mayoría de las especificaciones para buses *on-chip* utilizadas en FPGAs utilizan un *handshake* síncrono entre la interfaz maestra que controla el ciclo de acceso al bus (en lectura o escritura) y la interfaz esclava del otro *core*. El formato de la palabra `CRW` se representa en la figura 9.2. Está integrada por dos campos: el primero de ellos indica el número del `T-Core` objetivo de la solicitud y el segundo el número del contexto que se desea aplicar al mismo.

Tanto el número del `T-Cores` como el contexto aplicable están codificados en binario. De forma generalizada se define la composición de la palabra `CRW` para un sistema compuesto por un número n de `T-Cores` y φ_i contextos para cada `T-Core`. Con el fin de normalizar el controlador de reconfiguración, se realiza la simplificación de asignar a cada `T-Core` el mismo número de contextos. Este número es el correspondiente al número de contextos del `T-Core` con más variantes, y se representa con la letra m. El tamaño, en número de bits, de la palabra de configuración, ecuación 9.1, y los tamaños de los distintos campos quedan cuantificados como:

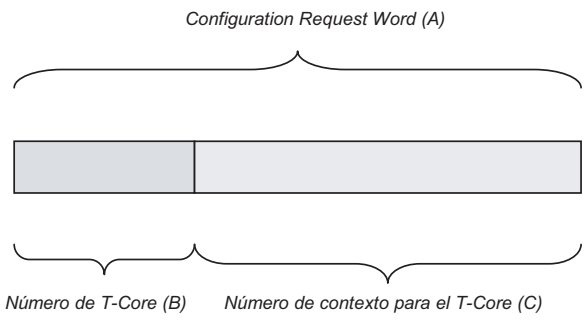


Figura 9.2.: Palabra de solicitud de reconfiguración `CRW`.

- A: Tamaño en bits de la palabra `CRW`.
- B: Tamaño en bits del campo indicador del número de `T-Cores`.
- C: Tamaño en bits del campo indicador del número de contexto.

$$\begin{aligned} \forall i \in [\mathbf{1}, \mathbf{n}] : \qquad m &= \text{máx}\,(\varphi_i) \\ B &= \left\lceil \frac{\ln n}{\ln 2} \right\rceil \\ C &= \left\lceil \frac{\ln m}{\ln 2} \right\rceil \\ A &= B + C \end{aligned} \tag{9.1}$$

Puesto que las solicitudes de cambio de contexto pueden provenir desde `T-Cores` o desde *cores*, se habilita la posibilidad de que *cores* de procesadores de propósito general tanto de tipo *hard* [215] o de tipo *soft* [208, 216] embebidos en la propia plataforma SoPC realicen tareas de *metacomputación*. También se contempla el caso en el que un `T-Core` solicite al controlador de reconfiguración su propio cambio de contexto, escribiendo esta petición a través del bus *on-chip*.

9.1.2. Interfaces y protocolo para el control de la reconfiguración

La reconfiguración parcial dinámica implica el cambio de una sección del circuito reconfigurable mientras que el resto del diseño sigue funcionando. Para poder realizar esta operación de forma viable se deben cumplir un conjunto de condiciones.

A nivel tecnológico, cada familia de dispositivos tiene unas características especificas relativas a la aplicación de la reconfiguración parcial dinámica. En el apartado de esta tesis correspondiente a la validación del sistema propuesto, se analizarán las restricciones impuestas en un tecnología concreta.

En el modelo de reconfiguración presentado, se distribuye la *metacomputación* entre el controlador `TBC` integrado en el sistema y otros *cores* capaces de escribir en el bus *on-chip* una palabra `CRW`. Por tanto, éste realmente se puede considerar como un modelo de autoreconfiguración, ya que los elementos que determinan los cambios se encuentran integrados

en el dispositivo. Las condiciones más generales que debe cumplir un sistema de autoreconfiguración han sido identificadas por R. Sidhu et al. en [87] y se presentan en el estado del arte de esta tesis (sección 3.1).

Además de estas condiciones generales, existen una serie de condicionantes aplicables al modelo propuesto en `Tornado`. Éstas definen las características que debe tener el sistema de control:

- Que el `T-Core` que se esté siendo reconfigurado no genere valores erróneos o indeterminados en sus salidas durante ese proceso. Tanto en las salidas de la interfaz con el bus *on-chip* como en las salidas propias del *core* (según la aplicación).
- Que el bus *on-chip* siga accesible para los *cores* que no estén siendo reconfigurados.
- Que en el caso de que se esté aplicando una reconfiguración *intra-task* de un `TnP-Core`, esto es, una modificación selectiva de parte del hardware o cambio del software, se controle adecuadamente el estado del `nano-procesador`. El control de este estado también puede incluir la retención de datos que pueden ser requeridos en el nuevo contexto.
- Que cada `T-Core` tenga la capacidad de rechazar temporalmente una reconfiguración que quiera aplicar el controlador de reconfiguración sobre él. Con esta condición se asegura que el proceso de cambio de contexto no se ejecute en instantes en los que el `T-Core` esté realizando tareas que no admitan interrupción, retrasando la reconfiguración hasta que ésta sea habilitada.

El sistema de control `Tornado` se articula mediante una infraestructura integrada en el propio diseño. Parte de esta infraestructura la componen las interfaces `Tornado`, donde se agrupan las señales específicas de este sistema de control. Se definen dos tipos de interfaces, la interfaz maestra (`TIF (M)`) y la esclava (`TIF (S)`).

El controlador de reconfiguración `TBC` tiene una interfaz `Tornado` maestra (figura 9.3(a)) y los `T-Cores` disponen de interfaces `Tornado` esclavas[1] (figura 9.3(b)).

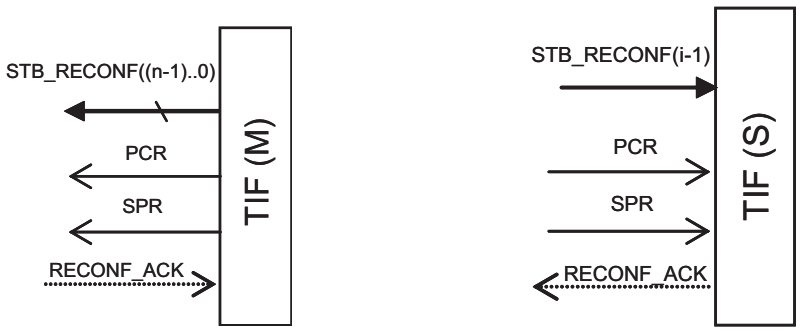
La definición del conjunto de señales que se agrupan en las interfaces y del protocolo *handshake* que se establece entre ellas, se justifican a continuación detallando las funcionalidades que se requieren del sistema de control.

Gestionar las peticiones de reconfiguración

Uno de los condicionantes expuestos obliga a que los `T-Cores` dispongan de mecanismos para rechazar de forma temporal el cambio de contexto que le esté solicitando el controlador de reconfiguración. Por ello, la operación de gestión de las solicitudes por parte del controlador de configuración debe establecerse de forma que tenga en cuenta esta particularidad. El software del controlador de reconfiguración `TBC` será el encargado de esta gestión.

Una de las premisas anteriormente definidas, impone que el bus *on-chip* siga accesible durante el proceso de reconfiguración parcial. Las peticiones de reconfiguración llegan al controlador `TBC` a través de ese bus. Puesto que el `TBC` puede encontrarse aplicando una

[1] Las interfaces `Tornado` esclavas se podrán incorporar a otros módulos susceptibles de ser controlados en el proceso de reconfiguración y que no sean específicamente `T-Cores`, por ejemplo *Bus-Macros* especiales con una interfaz `Tornado`.



(a)
Interfaz `Tornado` ***maestra.***

(b)
Interfaz `Tornado` ***esclava.***

Figura 9.3.: Interfaces `Tornado`.

petición de reconfiguración recibida anteriormente, no podrá ejecutar de forma inmediata una nueva petición de reconfiguración.

Por tanto, se debe dotar al `TBC` de un mecanismo de almacenamiento para estas solicitudes. Mediante una pila de memoria interna al controlador, donde se almacenen las peticiones de reconfiguración que se van recibiendo, se podrá establecer un sistema de gestión que atienda las peticiones almacenadas en la pila de forma ordenada. La gestión de la pila la realizará el software del controlador de reconfiguración.

En el caso de que se produzcan rechazos temporales por parte de algún `T-Core`, se pasará a atender la siguiente petición almacenada en la pila, reintentando posteriormente el cambio de contexto rechazado.

Al recibirse una petición de reconfiguración, se almacena inmediatamente en la pila de peticiones liberando el bus *on-chip*. Con el bus *on-chip* no bloqueado, se continúa la operativa normal de transferencia de datos entre los distintos *cores* mientras se estén aplicando reconfiguraciones parciales dinámicas.

Puesto que la arquitectura del controlador `TBC` es de `Core Mixto`, la política de gestión definida en el *Kernel* la aplica el software del `nano-procesador` embebido en éste. Esta flexibilidad permitirá que se utilicen distintas políticas de prioridad para la gestión de la pila de peticiones de configuración.

En resumen, la recepción de las peticiones de reconfiguración se realiza empleando únicamente el bus *on-chip* del CSoPC, no influyendo ni en las interfaces `Tornado` ni en el protocolo que se deba establecer entre éstas.

Habilitación/deshabilitación de la reconfiguración

Se propone que cada `T-Core` sea capaz de rechazar temporalmente una reconfiguración que quiera aplicar el controlador de reconfiguración sobre ese módulo. Por tanto, el `T-Core` dispondrá de mecanismos para habilitar o deshabilitar temporalmente su capacidad de admitir reconfiguraciones.

En el caso de que el `T-Core` sea un `TnP-Core`, por tanto, con al menos un `TnP` embebido, el control de la habilitación/deshabilitación se deberá poder realizar desde el propio programa. De esta forma, se podrán declarar rutinas o estados de alta prioridad no interrumpibles

(atención a interrupciones, acceso a periféricos, etc.) e indicar al controlador de reconfiguraciones que se requiere un tiempo adicional para realizar un cambio de contexto.

En resumen, la interfaz `Tornado` maestra debe disponer de una señal que indique a la esclava que se quiere aplicar un cambio de contexto. En el caso de que esté habilitada la reconfiguración, la interfaz esclava responderá admitiendo la misma mediante la activación de otra señal.

Control del contador de programa y del puntero de pila de los TnPs

Uno de los condicionantes anteriormente planteados establece que el sistema `Tornado` sea capaz de controlar el estado del `TnP` cuando se aplica sobre el `TnP-Core` donde está incluido el mismo, una reconfiguración *intra-task*. Teniendo en cuenta que esa reconfiguración *intra-task* puede estar orientada a modificar el programa del `nano-procesador`, alterando el contenido de la memoria dedicada de la FPGA mediante el cambio de la memoria de configuración, se propone el control de los registros específicos críticos del procesador y la posibilidad de bloquear el funcionamiento del mismo.

Para cada `TnP-Core` se podrá controlar el valor del contador de programa y del puntero de pila del `nano-procesador TnP` embebido durante y tras la aplicación de la reconfiguración parcial dinámica. Además, al *congelar* el funcionamiento del mismo, se podrá mantener el contenido de ciertos registros cuyos valores se utilicen en el nuevo contexto.

Puesto que se está proponiendo un sistema de control de reconfiguración flexible que puede utilizarse para diferentes aplicaciones, se plantean dos posibles modos de actuación para el contador de programa y para el puntero de pila en el proceso de reconfiguración. Estos dos modos de operación se deberán poder aplicar de forma independiente a los dos registros específicos. Así es posible definir un comportamiento diferente para cada uno de ellos. Cada modo implica para los dos registros:

- **Que se mantengan sus valores invariables.** De esta manera, cuando termine el proceso de cambio de contexto software, el procesador pasará a ejecutar la instrucción situada en la posición de memoria que indicaba el contador de programa antes de que se iniciase el proceso de reconfiguración. Para el caso del puntero de pila, si no se altera su valor, se mantendrán los anidamientos de subrutinas que se hayan realizado antes de iniciarse la reconfiguración.

  Esta operativa es aplicable a diseños donde únicamente se modifiquen en el programa inicial estructuras situadas en alguna posición concreta de la memoria de instrucciones, por ejemplo, tablas.

- **Que se inicialicen sus valores tras la reconfiguración.** Tras el cambio de contexto, se pasará a ejecutar la primera instrucción del nuevo código desde el programa principal.

  Esta situación, la más común, cubre las aplicaciones en las que se cambia completamente el programa del `TnP`, por ejemplo, para modificar el procesamiento de protocolos de comunicaciones [135].

En resumen, la interfaz `Tornado` maestra debe disponer de dos señales independientes para indicar a la interfaz esclava la acción que se debe aplicar sobre estos registros. Se realizará la acción indicada por éstas cuando se admita la aplicación de la reconfiguración por parte del módulo que tiene la interfaz esclava.

Control del estado de otros elementos hardware durante la reconfiguración

La utilización de las señales y del protocolo definidos en base a los dos puntos anteriores, se puede ampliar para definir el estado de ciertos elementos hardware que deban ser controlados durante la aplicación de la reconfiguración parcial dinámica según el diseño que se trate. Estos elementos pueden ser *Bus-Macros*, registros cuyo contenido debe conservarse entre contextos, etc.

En resumen, las señales de las interfaces `Tornado` pueden ser utilizadas para controlar elementos críticos del CSoPC durante el proceso de reconfiguración.

En concreto, las señales que integran las interfaces `Tornado` son (figura 9.4):

- `STB_RECONF`: Esta señal se genera en la interfaz maestra del controlador de reconfiguración `TBC` para indicar a un determinado `T-Core` que hay una petición de cambio de contexto sobre él. Cada interfaz `Tornado` esclava dispone de una única señal de entrada `STB_RECONF`.

- *Program Counter Reset* (`PCR`) y *Stack Pointer Reset* (`SPR`): Estas señales controlan el estado del `TnP` que esté siendo configurado en el caso de que el *core* objeto de la reconfiguración sea un `TnP-Core`. Son señales de salida para la interfaz `Tornado` maestra.

- `RECONF_ACK`: Mediante esta señal, el `T-Core` seleccionado mediante `STB_RECONF`, permite al `TBC` la aplicación del cambio de contexto solicitado cuando sea activada.

La distribución de las interfaces entre los módulos del modelo reconfigurable `Tornado` se representa en la figura 9.4. Los `T-Cores` disponen de interfaces esclavas y el controlador de reconfiguración `TBC` de una interfaz maestra.

Para poder aplicar las funcionalidades anteriormente establecidas (habilitación/deshabilitación de la reconfiguración, control de los `TnP` y de hardware crítico, etc.), se define un protocolo *handshake* hardware entre las interfaces `Tornado`. A continuación se desarrollan los pasos de esta secuencia de control representada en la figura 9.5:

- **Paso 1**: El controlador de reconfiguración `TBC` activa la señal `STB_RECONF` del `T-Core` número **i** sobre el que require realizar un cambio de contexto. Hay una señal `STB_RECONF` por cada `T-Core` que se conecta con el controlador `TBC`. De esta forma, se selecciona únicamente el `T-Core` objeto de reconfiguración.

  Simultáneamente a la activación de la señal `STB_RECONF`, el `TBC` pone el estado que corresponda para las señales `PCR` y `SPR` según el `T-Core` que esté siendo direccionado. Estas señales definen, tras la aplicación de la reconfiguración, el estado de los registros específicos de los `TnP` que pudiera tener embebidos el `T-Core`.

- **Paso 2**: El `T-Core`, a través de su interfaz `Tornado` esclava, activa la señal `RECONF_ACK` si en presencia de la señal `STB_RECONF` activa, tiene habilitada la reconfiguración.

  A partir de ese instante, la interfaz `Tornado` esclava utiliza la señal `STB_RECONF` para activar internamente los mecanismos que en el `T-Core` deben controlarse durante la reconfiguración:

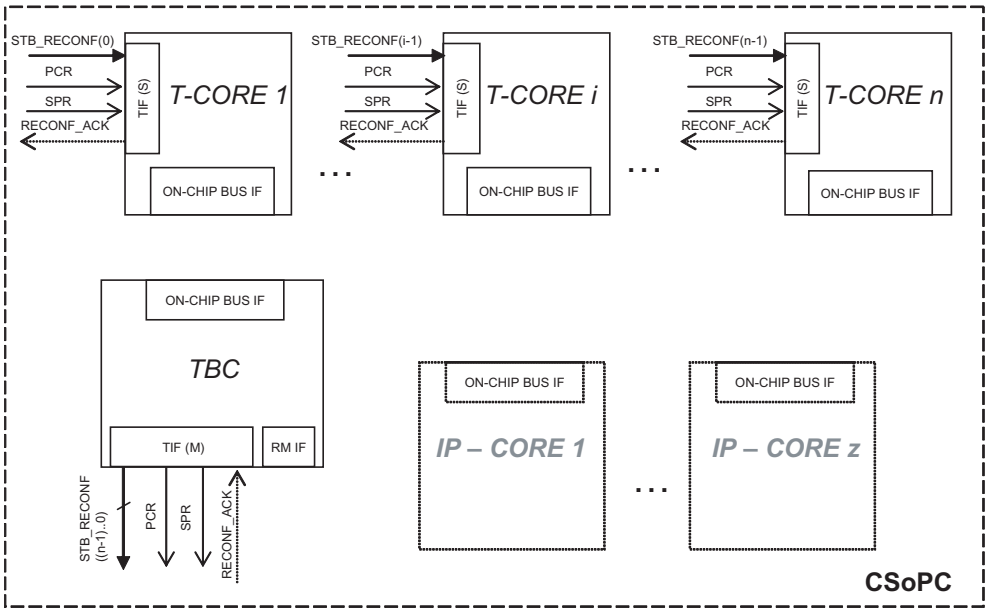


Figura 9.4.: Interfaces `Tornado` incluidas en los `T-Cores` y en el controlador de reconfiguración `TBC` .

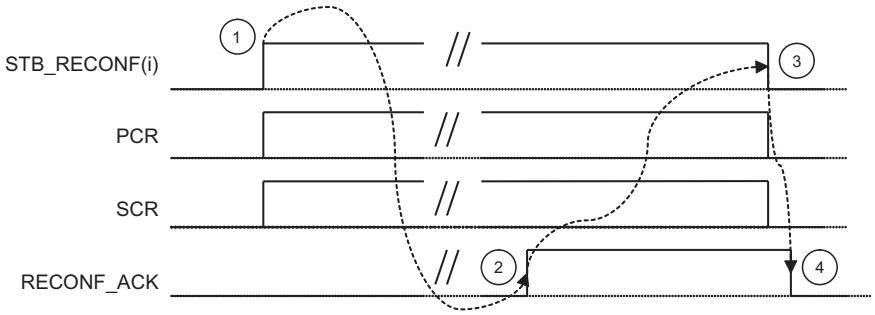


Figura 9.5.: Secuencia del *handshake* para la aplicación de la reconfiguración.

- Si el `T-Core` tiene embebido un `TnP` (o varios), éste se mantiene congelado. Esto implica que el `TnP-Core` objeto de la misma tiene todos sus módulos parados. Ni se ejecuta software, ni se atiende a las interfaces, a excepción de la interfaz `Tornado` esclava.
- Se desconectan[2] las salidas del *core* del bus *on-chip* de forma que no afecten a

[2]Si el dispositivo que se está empleando admite estados de alta impedancia mediante *buffers* triestado, el método empleado para realizar las *desconexiones* será el control de éstos, imponiendo el estado de alta impedancia durante el proceso de reconfiguración.

las transiciones que en él se puedan seguir manteniendo.

- Se fijan los valores adecuados para una situación de transición en los *pines* de salida que sean controlados por el *core* que se está reconfigurando.
- Si el diseño lo requiere, la señal `STB_RECONF` puede indicar a partir de la activación de la señal `RECONF_ACK`, en un registro de semáforos, el número de `T-Core` que esté siendo reconfigurado. Cuando se desactive la señal `STB_RECONF`, significará que ha finalizado el proceso de cambio de contexto, situación que se reflejará en ese sistema de semáforos.

También a partir del **paso 2**, la interfaz `Tornado` esclava valida las señales `SPR` y `PCR`. Si están a nivel alto, el contador de programa y el puntero de pila son inicializados tras la operación de cambio de contexto.

Tras el **paso 2**, el controlador de reconfiguración inicia la carga del *bitstream* parcial. Durante el tiempo requerido para realizar la carga completa del mismo, el controlador de reconfiguración mantiene la señal `STB_RECONF` activa. Cuando el controlador de reconfiguración ponga a nivel bajo la señal `STB_RECONF`, la señal `RECONF_ACK` será desactivada por la interfaz `Tornado` esclava.

Si el `T-Core` no responde al controlador con la activación de la señal `RECONF_ACK`, esto indica que el `T-Core` no admite reconfiguraciones temporalmente. Entonces, el controlador de reconfiguración pasará a aplicar otras peticiones que tenga pendientes en la pila o bien reintentará ésta, hasta que tenga habilitada la admisión de reconfiguraciones.

- **Paso 3**: Cuando el controlador de reconfiguración `TBC` finaliza la carga de la configuración parcial, desactiva la señal `STB_RECONF` de su interfaz maestra.

- **Paso 4**: Tras la desactivación de la señal `STB_RECONF`, un ciclo de reloj más tarde, puesto que se opera de forma síncrona, la interfaz esclava desactiva la señal `RECONF_ACK`.

  Cuando `RECONF_ACK` pasa a nivel lógico bajo, todos los elementos del `T-Core` comienzan a funcionar y las salidas que se habían desconectado se vuelven a activar. Si el `T-Core` incluye `TnPs`, éstos inician la ejecución del programa según las condiciones que se habían establecido mediante las señales `SPR` y `PCR`.

En el apartado 10.1 se presentan en detalle los parámetros temporales de esta secuencia de señales. Estos parámetros, junto con los que se definen para el proceso de escritura de la palabra `CRW` en el controlador de reconfiguración, permitirán caracterizar los retardos producidos en el MCSoPC por la utilización de la reconfiguración parcial dinámica empleando el sistema `Tornado`.

9.2. Controlador de reconfiguración: TBC

El modelo `Tornado` presentado en el capitulo anterior incluye un controlador de reconfiguración en el CSoPC integrado en la infraestructura de control de la auto-reconfiguración. La versión básica de este controlador se denomina `TBC`.

En este modelo se plantea una distribución de la *metacomputación* entre diferentes módulos del sistema. Por ello, el controlador únicamente se debe encargar de gestionar las peticiones de reconfiguración que le lleguen a través del bus *on-chip* y controlar la aplicación de las mismas. Para la definición de las características del `TBC`, se ha partido de unos objetivos generales orientados a obtener un controlador simple y flexible:

- Obtener un controlador capaz de admitir un número variable de `T-Cores` y contextos.
- Dotar al controlador de una arquitectura flexible que permita una configuración sencilla y automatizada.
- Poder realizar distintas implementaciones del controlador sobre distintas tecnologías, por lo que deberá ser descrito en lenguaje HDL. De esta forma, se quiere disponer de versiones del controlador optimizadas para cada tecnología y que puedan adaptar su método de carga de los *bitstreams* de acurdo con el método de configuración que tenga el dispositivo.
- Disponer en el controlador de una sección de comunicación con el bus *on-chip* estándar, definida a través de una interfaz. De esta manera se podrá modificar esta interfaz para cumplir con diversas especificaciones, siendo fácilmente integrable en SoPCs diseñados sobre la base de diferentes plataformas.

Un *core* con estas características se adapta adecuadamente a la arquitectura definida para los `Cores Mixtos` (capítulo 4). En la figura 9.6 se muestra la arquitectura interna del controlador `TBC` definido con la arquitectura de `Core Mixto`. Los elementos que componen el controlador son:

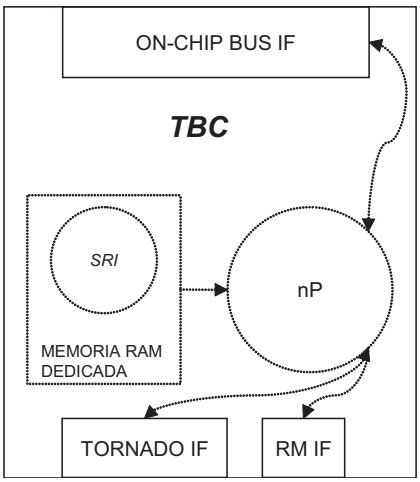


Figura 9.6.: Arquitectura del controlador básico de reconfiguración definido en `Tornado`.

- `Nano-procesador` **embebido**. Es un pequeño procesador *core soft* de tipo RISC encargado de realizar las tareas de control.

- ***System Reconfiguration Information*** (`SRI`). Es el software del `nano-procesador`. Se sitúa en la memoria RAM dedicada de la FPGA y está compuesto por dos secciones: una común para los diferentes diseños en los que se use el `TBC` sobre una misma tecnología, denominada *Kernel*; y otra parametrizable para cada diseño CSoPC donde se incluye información sobre los distintos `T-Cores`.

  El *Kernel* gestiona la pila de peticiones de reconfiguración y controla la aplicación de las mismas. La sección parametrizable del software del controlador tiene información específica sobre la aplicación de la reconfiguración a cada `T-Core`. Por ejemplo, si se trata de un `TnP-Core`, dispondrá de información relativa a la gestión de los registros específicos del `nano-procesador` durante el proceso de reconfiguración.

  La parametrización del software del controlador se realizará automáticamente en la fase de ensamblado del código de los distintos `Cores Mixtos` del diseño, mediante un ensamblador propio del *framework* `Tornado`[3].

- **Interfaz del bus *on-chip***. Es la interfaz del `TBC` que recibe las peticiones de reconfiguración y permite la conexión del mismo al bus *on-chip*.

- **Interfaz con el sub-sistema de carga y almacenamiento de *bitstreams* (*Reconfiguration Media Interface* (`RM IF`)**. Es la interfaz mediante la que el controlador de reconfiguración `TBC` se comunica con el sub-sistema de carga y almacenamiento de los ficheros de configuración parciales (*bitstreams*). Este sistema es dependiente del dispositivo de lógica reconfigurable que se use.

  Los sub-sistemas de carga y almacenamiento pueden ser; desde sistemas básicos realizados mediante la combinación de una CPLD externa junto a memorias FLASH [217]; sistemas específicos del fabricante para almacenamiento masivo de *bitstreams* como el sistema ACE [218]; hasta mecanismos de reconfiguración completamente integrados en el dispositivo como el ICAP de Xilinx [155].

  A través de la interfaz `RM IF`, el controlador selecciona el *bitstream* parcial que corresponde con la configuración requerida, iniciando la carga del mismo a través del puerto de configuración de la FPGA [75, 80]. Esta operación se representa en la figura 9.7.

9.3. Unidad procesadora Tornado nano-Processor: TnP

En el modelo reconfigurable `Tornado` se considera la posibilidad de que se empleen `Cores Mixtos`. Si estos *cores* admiten la reconfiguración controlada mediante el sistema propuesto, se denominan `TnP-Cores` puesto que integran `nano-procesadores` compatibles con `Tornado` (TnP).

Para poder controlar el estado de los `TnPs` durante el proceso de reconfiguración, éstos deben disponer de una interfaz `Tornado` esclava. Esta interfaz actúa sobre el circuito interno del `nano-procesador` de forma que se puedan realizar las siguientes acciones:

[3] El *framework* `Tornado` es un conjunto de herramientas que asisten a las herramientas del fabricante en el diseño de sistemas con `Tornado` para una determinada tecnología. En la parte de esta tesis se presentará un *framework* completo desarrollado para una tecnología concreta.

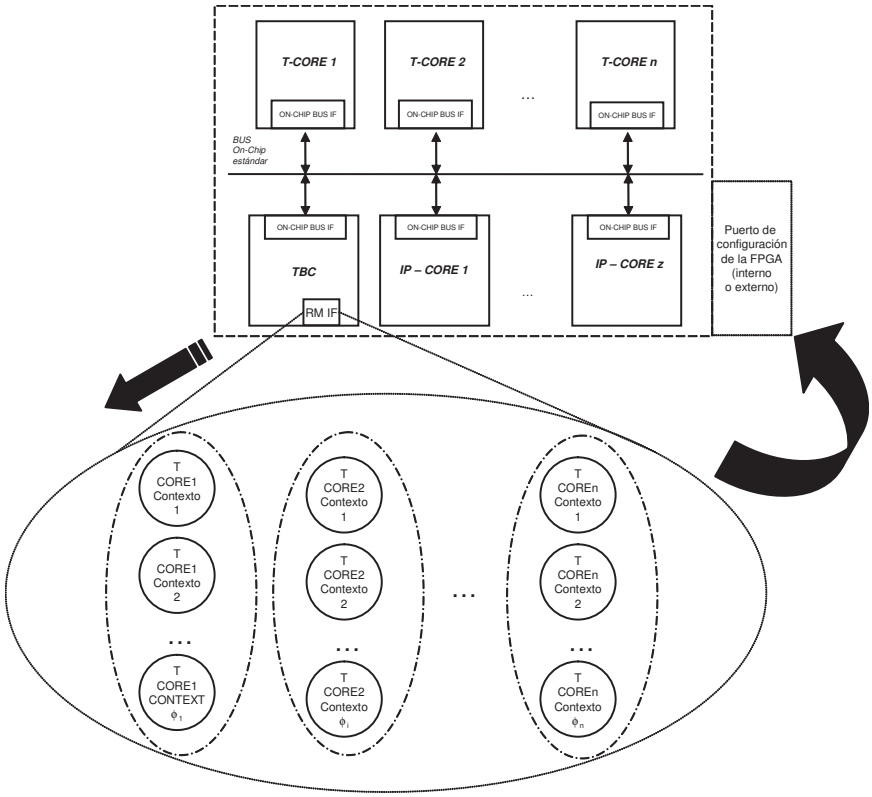


Figura 9.7.: Interfaz `RM IF`.

- **Habilitación y deshabilitación de la reconfiguración desde el software**. Con esta acción un `TnP` puede rechazar temporalmente una reconfiguración que se quiera aplicar sobre él.
- **Control del contador de programa y del puntero de pila**. Con esta acción se controlan estos dos registros específicos durante el proceso de reconfiguración *intra-task*. Se consideran las siguientes variantes en el control, teniendo en cuenta que se pueden aplicar independientemente a ambos:
  - Que se mantengan sus valores invariables.
  - Que se inicialicen sus valores tras la reconfiguración.

Para disponer de un `nano-procesador TnP` que cumpla estas características deberán añadirse tanto elementos software como hardware al `nano-procesador` original (no compatible con `Tornado`):

Modificaciones software

Se agrega al `TnP` una nueva instrucción para habilitar y deshabilitar temporalmente la aceptación de reconfiguraciones realizadas desde el controlador de reconfiguración `TBC`. Se definen los mnemónicos para las dos posibilidades como:

- `RENCONF ENABLE`: El `TnP` admite reconfiguraciones.
- `RENCONF DISABLE`: Los intentos de aplicar reconfiguraciones sobre el `TnP` serán rechazados hasta que no se habilite de nuevo la admisión de éstas.

Modificaciones hardware

Se incluye en el `TnP` una interfaz `Tornado` esclava para comunicarse con el controlador de reconfiguración `TBC` utilizando las señales y protocolos definidos en `Tornado`.

En la figura 9.8 se representan los `TnPs` con las interfaces `Tornado` esclavas conectados a la interfaz `Tornado` maestra del `TBC`. Las señales mostradas son parte de la infraestructura `Tornado`, no habiéndose incluido el bus *on-chip* a fin de simplificar la representación.

Además de esta interfaz, el hardware del `nano-procesador` original se debe modificar para que admita la nueva instrucción añadida y para que la interfaz `Tornado` tenga control sobre el mismo.

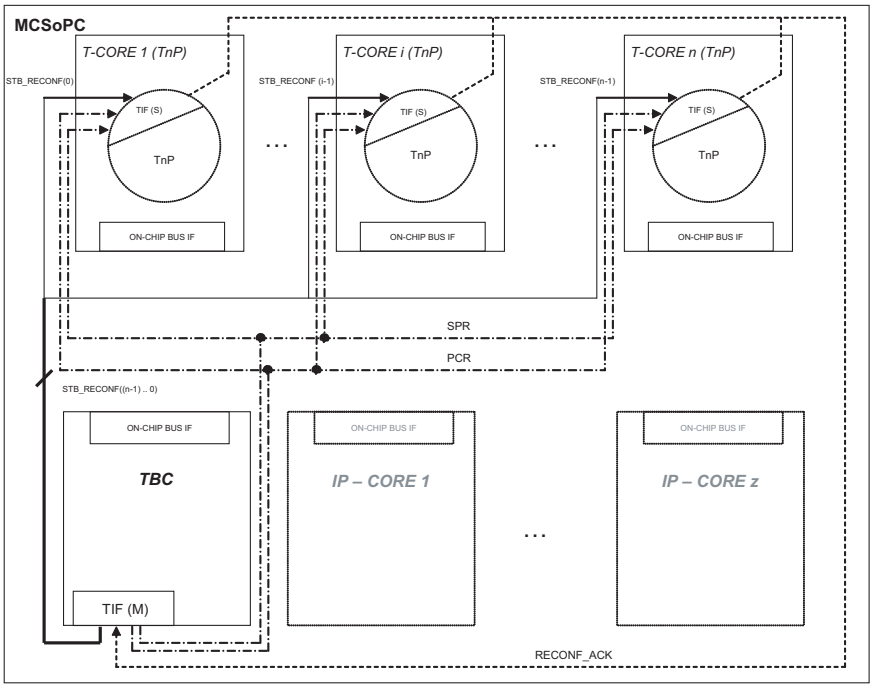


Figura 9.8.: Interfaces `Tornado` de los `TnPs`.

En función de estas directivas, las modificaciones hardware que se deberán aplicar para que un `nano-procesador` se convierta en un `TnP` son:

- Que disponga de acceso a las señales internas de inicialización del contador de programa y del puntero de pila, para poder controlar ambos según el valor de las señales `SPR` y `PCR`.
- Que incluya un mecanismo para que la interfaz `Tornado IF` controle el estado de la habilitación de reconfiguraciones definido mediante la instrucción añadida con ese propósito.

- Que pueda parar la ejecución del programa en curso (*congelar* el procesador) durante el procedimiento de cambio de contexto mediante reconfiguración. Esta sección de la interfaz deberá combinar, además de las señales `Tornado`, el estado de habilitación o deshabilitación de la reconfiguración definido internamente mediante la instrucción software.

En la sección de validación del sistema propuesto, se aplican estas directivas para el diseño de un `TnP` optimizado para una tecnología concreta. Ese `TnP` se utiliza en las tres plataformas de evaluación presentadas en esta tesis.

10. Modelado de Tornado

El tiempo de desarrollo de los sistemas SoPC se ha reducido drásticamente. La competencia actual en el mercado de los productos que utilizan SoPCs requiere los nuevos diseños en un breve espacio de tiempo.

Si estos sistemas, además, incluyen la auto-reconfiguración parcial dinámica, debe proveerse algún sistema que permita evaluar antes de la introducción de la infraestructura necesaria para su soporte, si ésta es viable. Por otro lado, se necesita tener caracterizados los procesos que producen retardos en el sistema debidos a la auto-reconfiguración. Esta identificación se realiza mediante parámetros que permiten cuantificar las penalizaciones introducidas en el sistema.

En este capítulo se introducen dos elementos que facilitan estas tareas:

- Un modelado de los recursos lógicos necesarios por los elementos de la infraestructura `Tornado`.
- Una caracterización temporal de los procesos involucrados en la auto-reconfiguración parcial dinámica controlada con el sistema `Tornado`.

10.1. Caracterización temporal

En esta sección se detallan los parámetros temporales que caracterizan los protocolos involucrados en el sistema de control `Tornado`. Mediante estos parámetros se podrá modelar y cuantificar la secuencia temporal de aplicación de la reconfiguración parcial dinámica sobre los sistemas MCSoPC.

Se distinguen dos acciones: la escritura de la palabra de configuración `CRW` y el protocolo *handshake* de configuración entre las interfaces `Tornado` maestra y esclava.

10.1.1. Parámetros relacionados con la palabra de configuración

Para caracterizar el proceso de petición de reconfiguración, es necesario establecer el tiempo transcurrido entre la escritura de la palabra `CRW` por un *core* maestro al controlador de reconfiguración `TBC` y, la activación de la señal `STB_RECONF` por este último. A partir de ese instante comienza la secuencia de *handshake* entre el controlador de reconfiguración `TBC` y el `T-Core` objeto del cambio.

Este primer tiempo, el necesario para realizar la escritura de la palabra `CRW`, no es constante. Los factores que hacen que este tiempo varíe incluso en un mismo diseño, son múltiples. Éstos se han englobado en dos grupos: los que dependen del bus *on-chip* y los relacionados con la pila de petición de reconfiguraciones del controlador.

Tráfico en el bus on-chip.

Parte del tiempo requerido para esta secuencia es el que transcurre desde que la interfaz *on-chip* maestra que escribe la palabra `CRW` e inicia el ciclo de escritura sobre el bus, hasta que el `TBC` la lee. En un SoPC no se puede asegurar que este tiempo sea igual para todos los accesos.

Por ejemplo, si se emplea una topología de bus compartido entre varios maestros, el *core* que escribe la petición `CRW` deberá esperar a que el arbitrador del bus le ceda el control del mismo. Lo cual depende del número de maestros que desean acceder simultáneamente al bus y de la política de prioridades establecida en el arbitrador.

La operación de escritura de la palabra `CRW` desde el maestro hacia el controlador `TBC`, se ha representado como una operación síncrona de acceso al bus *on-chip* (figura 10.1). La interfaz maestra que escribe la `CRW` activa la señal de *strobe* para indicar que desea realizar una escritura sobre el esclavo. Cuando la escritura sobre el `TBC` se haga efectiva, éste responderá activando durante un pulso del reloj del bus *on-chip* la señal *acknowledge*. La interfaz maestra desactiva entonces la señal de *strobe* en respuesta al *acknowledge* desde la interfaz esclava del `TBC`, terminando de esta forma el ciclo de escritura en el bus.

Este tipo de señalizaciones son ampliamente utilizadas por las especificaciones estándares de buses *on-chip*, habiéndose englobado también en esta operación la gestión del arbitrador para las topologías de bus en las que sea aplicable.

Este tiempo se define como "el tiempo necesario para la escritura de la palabra de configuración `CRW`" (*CRW Write Time* -**Twcrw**-).

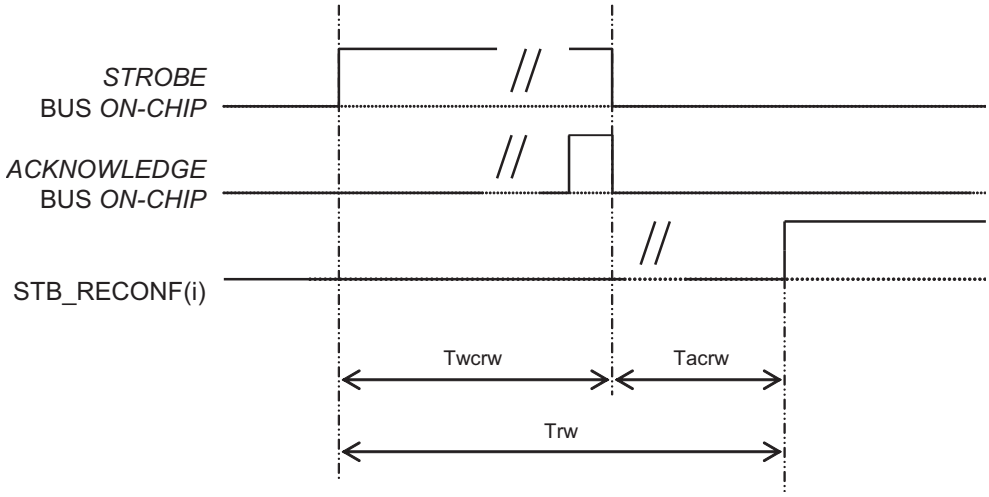


Figura 10.1.: Diagrama de tiempos de la escritura y aplicación de una palabra de petición de reconfiguración `CRW`.

Número de peticiones de reconfiguración pendientes en la pila del controlador de reconfiguración `TBC`

Cuando se escribe una `CRW` en el `TBC`, se coloca ésta en la última posición[1] de la pila de peticiones de reconfiguración. Por tanto, según la posición de la pila que se esté procesando y el número de peticiones de reconfiguración pendientes, diferirán sustancialmente los tiempos que discurre desde la lectura de la palabra de configuración por parte del `TBC` hasta que aplica ésta. Este tiempo se define como "el tiempo transcurrido desde la lectura de la palabra `CRW` del bus *on-chip* por parte del `TBC`, hasta que se ejecuta[2]" (*CRW Application Time* -**Tacrw**-).

En resumen, el tiempo transcurrido entre la escritura de la palabra de configuración y la ejecución de la misma (inicio de la secuencia de *handshake*) se establece como:

- **Tcrw** : "Tiempo asociado a la palabra `CRW`" (*CRW Time*).

$$\mathbf{Tcrw} = \mathbf{Twcrw} + \mathbf{Tacrw} \tag{10.1}$$

La medida de este parámetro, al igual que la del resto de los parámetros de modelado temporal, se representa con la unidad de ciclos de reloj del bus *on-chip*.

10.1.2. Parámetros del protocolo de configuración

El protocolo *handshake* de configuración es el segundo definido en la infraestructura del sistema de control `Tornado`. En la figura 10.2 se resume la secuencia[3] descrita anteriormente en el apartado 9.1.2. A continuación se definen los parámetros que caracterizan este proceso

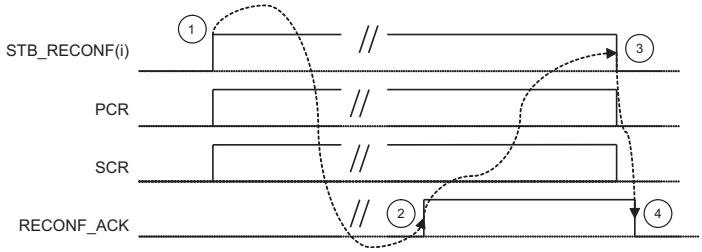


Figura 10.2.: Pasos de la secuencia del *handshake* de configuración.

[1] Esta forma de gestionar las palabras `CRW` por parte del controlador de reconfiguración no es estricta. En la versión básica `TBC` se propone que el *Kernel* del mismo tenga esa secuencia de funcionamiento, quedando el sistema abierto a incluir otras políticas de gestión mediante la modificación del programa del `TBC`.

[2] Se entiende como *ejecución* de una partición de reconfiguración como el *intento* de aplicación de la misma sobre un determinado `T-Core`, ya que esta puede ser rechazada.

[3] Resumen de la secuencia: En **1** el controlador de reconfiguración activa las señales `STB_RECONF(i), SPR y PCR`. En **2**, cuando el `T-Core` que está siendo direccionado admite el requerimiento de reconfiguración, éste activa la señal `RECONF_ACK`. Ésta se mantiene activa mientras `STB_RECONF(i)` lo esté (el tiempo que dura la carga del *bitstream* parcial). En **3** termina la carga, el controlador desactiva `STB_RECONF(i)` y finalmente en **4** la interfaz `Tornado` esclava desactiva la señal `RECONF_ACK`.

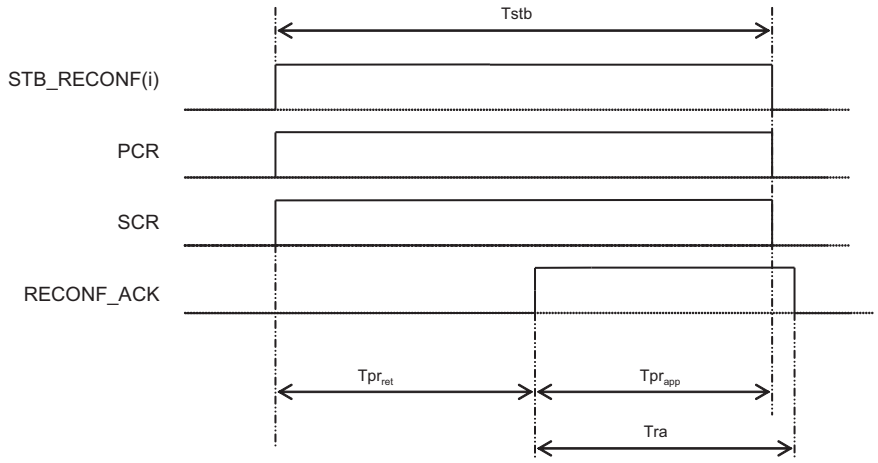


Figura 10.3.: Parámetros temporales de la secuencia del *handshake* de configuración.

(figura 10.3.):

- **"Tiempo de aceptación de la reconfiguración parcial"** (*Partial Reconfiguration Retry Time* -$\mathbf{Tpr_{ret}}$-): Tiempo desde que el controlador de reconfiguración activa la señal `STB_RECONF(i)`, hasta que el `T-Core` que va a ser reconfigurado activa la señal `RECONF_ACK` indicando que se puede aplicar la misma.

  Tal y como se ha definido el proceso de *handshake* en el apartado 9.1.2, el `T-Core` objeto de reconfiguración puede rechazar el intento de reconfiguración temporalmente, con lo que este tiempo se incrementaría.

- **"Tiempo de carga de la reconfiguración parcial"** (*Partial Reconfiguration Application Time* -$\mathbf{Tpr_{app}}$-): Tiempo durante el que se aplica la reconfiguración de forma efectiva. El controlador de reconfiguración mantiene la señal `STB_RECONF(i)` de forma que la interfaz de reconfiguración del `T-Core` siga activando la señal `RECONF_ACK`.

- **"Tiempo de activación de la señal `RECONF_ACK`"** (*Reconfiguration Acknowledge Time* -**Tra**-): Tiempo durante el que el `T-Core` mantiene la señal `RECONF_ACK` activa. Este tiempo es un ciclo de reloj más largo que el Tpr_{app}, ya que la señal `RECONF_ACK` es desactivada por el `T-Core` de forma síncrona tras detectarse que la señal `STB_RECONF(i)` deja de estar activa.

- **"Tiempo de activación de la señal `STB_RECONF`"** (*Strobe Time* -**Tstb**-): Tiempo total durante el que está activa la señal `STB_RECONF(i)` en una secuencia de cambio de contexto. Este tiempo no es fijo. Depende del estado del `T-Core` que va a ser reconfigurado. Esta variabilidad proviene del parámetro $\mathbf{Tpr_{ret}}$ previamente definido.

  Tal y como se representa en la figura 10.3, este parámetro está compuesto por la suma

de los siguientes tiempos:

$$\mathbf{Tstb} = \mathbf{Tpr_{ret}} + \mathbf{Tpr_{app}} \tag{10.2}$$

10.1.3. Normalización de los parámetros temporales variables

Aunque los parámetros temporales asociados a la palabra `CRW` (**Tcrw**) y tiempo de activación de la señal `STB_RECONF` (**Tstb**) dependen de varios factores según se ha justificado anteriormente, con el fin de facilitar el modelado y la realización de comparativas se definen un conjunto de parámetros normalizados. Mediante esta normalización, se dispone de datos comparables obtenidos bajo las mismas condiciones.

En el parámetro **Tstb**, la variabilidad proviene del tiempo $\mathbf{Tpr_{ret}}$. Este parámetro es dependiente de los reintentos necesarios antes de realizar la reconfiguración, debidos al rechazo del `T-Core` objeto de la misma. La normalización realizada implica *que no hay rechazo al primer intento de aplicación de la reconfiguración*. Los parámetros así normalizados se definen como $\mathbf{Tpr_{ret0}}$ y **Tstb0**.

En el caso del parámetro **Tcrw**, la variabilidad proviene de dos operaciones. Por un lado, de la escritura de la palabra de configuración `CRW` en bus *on-chip* (**Twcrw**) y, por otro, del número de reconfiguraciones pendientes en la pila y de cómo se resuelva cada una de ellas (situaciones caracterizadas por el parámetro **Tacrw**).

La normalización en este caso se establece como:

- **Twcrw0**: **Twcrw** normalizado a la escritura de la palabra `CRW` en un bus compartido multi-maestro, por parte del primer maestro que accede al mismo. El bus está libre cuando se realiza esa operación.
- **Tacrw0**: **Tacrw** normalizado a una situación en la que la pila de petición de reconfiguraciones del controlador esté vacía cuando se recibe la petición de reconfiguración `CRW`.

Por tanto, los parámetros normalizados **Tstb0** y **Tcrw0** quedan definidos según las ecuaciones 10.3 y 10.4:

$$\mathbf{Tstb0} = \mathbf{Tpr_{ret0}} + \mathbf{Tpr_{app}} \tag{10.3}$$

$$\mathbf{Tcrw0} = \mathbf{Twcrw0} + \mathbf{Tacrw0} \tag{10.4}$$

10.2. Modelado de recursos lógicos

Para que un SoPC implementado en una FPGA que tecnológicamente admita la reconfiguración parcial dinámica pueda realmente hacer uso de la misma, requiere que en el diseño, se añada una infraestructura de control. Un ejemplo de esta infraestructura es la presentada en esta tesis.

La infraestructura añadida para poder realizar el control definido por el sistema `Tornado`, se implementa utilizando recursos lógicos generales[4] y bloques de memoria RAM dedicados

[4]Se consideran recursos lógicos generales de una FPGA de grano fino básicamente a las *Look Up Tables (LUTs)* y los biestables. Al tener en cuenta únicamente estos recursos, se está realizando una simplificación ya que, según la tecnología empleada, las celdas básicas de la FPGA pueden incluir otros elementos como una lógica rápida para la llevada, multiplexores, etc.

de la FPGA. Desde el punto de vista de área de silicio necesaria para un determinado diseño SoPC, será viable la adición de esta infraestructura al SoPC (para convertirlo en un CSoPC) cuando los recursos requeridos por ésta sean inferiores a los ahorrados como resultado de hacer uso de la auto-reconfiguración parcial dinámica; o bien, si el uso de la misma es la única solución posible para esa aplicación.

Teniendo en cuenta la complejidad de este tipo de diseños, mediante los parámetros definidos en esta sección, se articula un modelo aproximado de la infraestructura que permita estimar, para cada implementación de `Tornado`, y de forma previa al diseño HDL, los recursos requeridos para la misma. La cantidad de recursos adicionales necesarios dependen de los siguientes factores:

- **Complejidad del controlador:** El sistema `Tornado` se ha planteado en los apartados anteriores de forma general. La complejidad del controlador puede variar según el número de módulos reconfigurables que se desea que admita, el sistema de configuración que emplee la FPGA, o la política de gestión de prioridades.
- **Número de `T-Cores`:** Cada `T-Core` dispone de una interfaz `Tornado` esclava para el control de la reconfiguración. Por lo tanto, parte de los recursos necesarios son dependientes del número de módulos.
- **Número de `TnP-Cores`:** Los `TnP-Cores` incluyen un `nano-procesador` modificado (`TnP`) que admite cambios de contexto mediante el sistema `Tornado`. Esta diferencia de recursos entre el `nano-procesador` no compatible con `Tornado` y el `nano-procesador` modificado para que sea compatible, también se debe tener en cuenta para el modelado. En estos casos las interfaces `Tornado` están incluidas en los `TnPs`.
- **Opciones de síntesis, emplazamiento y rutado:** Las distintas opciones de las herramientas utilizadas para las fases de síntesis, emplazamiento y rutado permiten especificar optimizaciones en área y velocidad. En general, las optimizaciones en velocidad implican la necesidad de más recursos lógicos para acortar caminos con retardos elevados.

  La utilización de los parámetros de modelado que se definirán a continuación, implica coherencia con las opciones empleadas en esas fases de implementación, de forma que los datos sean comparables.

  También hay que resaltar que los recursos requeridos por una infraestructura de este tipo pueden variar según las restricciones impuestas a las herramientas de emplazamiento y rutado en cada diseño. Por tanto, es posible que estas herramientas utilicen más recursos que los estimados por el modelo o incluso menos.

En función de estos factores se definen a continuación los parámetros que caracterizan a la infraestructura `Tornado`.

Sobrecoste estimado por recursos lógicos debidos al controlador

El parámetro denominado "Sobrecoste estimado por recursos lógicos debidos al controlador" (*Estimate Controller Overhead* -$\mathbf{CTR_{over\_est}}$-) caracteriza los recursos lógicos estimados para la implementación del controlador de reconfiguración[5].

[5]La versión básica especificada de controlador de reconfiguración es el `TBC`.

- $\mathbf{CTR_{lut}}$: Número de LUTs requeridas para la implementación del controlador. Básicamente estos recursos representan la lógica combinacional del mismo.
- $\mathbf{CTR_{ff}}$: Número de biestables utilizados para la implementación del controlador.
- $\mathbf{CTR_{4Kram}}$: Número de bloques de memoria dedicada de 4 Kbits[6] de tamaño. Según el dispositivo empleado, el tamaño de los bloques de memoria puede variar. En estos casos se normaliza al número de bloques de 4K requeridos.

Puesto que se desea obtener un modelo genérico que permita contrastar soluciones para distintas tecnologías, no se ha utilizado el concepto de *puertas lógicas equivalentes* que permite unificar las unidades de los distintos recursos. Esta decisión se basa en que las fórmulas para el cálculo de las mismas difieren de un fabricante a otro [3], por lo que las unidades *puertas lógicas equivalentes* no son comparables. Por este motivo, se mantienen separados en las distintas ecuaciones propuestas la cantidad de cada tipo de recurso lógico necesario.

Sin embargo, en el caso de que trabaje únicamente con una tecnología, se pueden unificar las distintas unidades de recursos lógicos (LUTs, biestables y bloques de memoria) a la unidad definida como *puertas lógicas equivalentes*. Para ello, se deberá usar el sistema de conversión propuesto por el propio fabricante. A modo de ejemplo, en [219] se detalla el proceso de conversión a *puertas lógicas equivalentes* de los recursos lógicos de los dispositivos de la familia XC4000 de Xilinx, muy similares a los ampliamente extendidos Virtex de ese mismo fabricante.

En conjunto, la estimación de recursos lógicos del controlador de reconfiguración se agrupan según la ecuación 10.5:

$$\mathbf{CTR_{over\_est}} = \mathbf{CTR_{lut}} + \mathbf{CTR_{ff}} + \mathbf{CTR_{4Kram}} \tag{10.5}$$

Sobrecoste estimado por recursos lógicos debidos a los nano-procesadores TnPs

El parámetro "Sobrecoste estimado por recursos lógicos debidos al nano-procesador TnP" (*Estimate TnP Overhead* -$\mathbf{TnP_{over\_est}}$-) permite estimar el sobrecoste teórico en LUTs y biestables de cada unidad `TnP` utilizada en el diseño. El número total de unidades se obtendrá de sumar todas las embebidas en los `TnP-Cores` del sistema. De forma general, un `TnP-Core` puede incluir más de un procesador `TnP`.

Al tratarse de una modificación del `nano-procesador` utilizado en los `Cores Mixtos`, parte de la infraestructura `Tornado` serán los recursos necesarios para esa adaptación. Estos recursos se estiman según la ecuación 10.6.

- $\mathbf{TnP_{orig\_lut}}$: Número de LUTs del `nano-procesador` original.
- $\mathbf{TnP_{orig\_ff}}$: Número de biestables del `nano-procesador` original.
- $\mathbf{TnP_{mod\_lut}}$: Número de LUTs del `nano-procesador` modificado `TnP`.

[6] 4Kbits es la unidad más habitual para los bloques de memoria RAM dedicados en las FPGAs de complejidad media.

- $\mathbf{TnP_{mod\_ff}}$: Número de biestables del `nano-procesador` modificado `TnP`.

$$\mathbf{TnP_{over\_est}} = (\mathbf{TnP_{mod\_lut}} - \mathbf{TnP_{orig\_lut}}) + (\mathbf{TnP_{mod\_ff}} - \mathbf{TnP_{orig\_ff}}) \quad (10.6)$$

En este caso, no aparece el término de los bloques de memoria RAM dedicada, ya que el número necesario para un `nano-procesador` no depende de que éste sea compatible con `Tornado`.

Sobrecoste por recursos lógicos debidos a las Tornado Bus-Macro

El parámetro temporal "Sobrecoste por recursos lógicos debidos a una `Tornado Bus-Macro`" (*Tornado Bus-Macro Overhead* -$\mathbf{MACRO_{over}}$-) permite estimar, en los casos en los que se utiliza la reconfiguración parcial *inter-task*, los recursos necesarios debidos a los elementos adicionales requeridos para solventar la problemática de la interconexión entre las secciones fijas y los módulos reconfigurables. En la infraestructura `Tornado`, a éstos se denominan elementos `Tornado Bus-Macros` (`TBMs`).

En el apartado 3.2.1 se exponen las alternativas propuestas por distintos autores para este tipo de elementos que posibilitan el guiado del rutado, aunque todas ellas están orientadas a una determinada tecnología.

En el caso de `Tornado`, aunque se realiza un planteamiento generalizado de los mismos, también hay que tener en cuenta la tecnología sobre la que se implementa el diseño. Se va a suponer que las `Tornado Bus-Macros` se pueden construir con LUTs ($\mathbf{MACRO_{lut}}$), biestables ($\mathbf{MACRO_{ff}}$) y *buffers* triestado ($\mathbf{MACRO_{tribuf}}$). Una implementación en base a estos elementos, puede resultar inviable para determinados dispositivos si, por ejemplo, no disponen de *buffers* triestado. En estos casos, se deberán construir las `Tornado BMs` con los elementos disponibles en el dispositivo, de tal forma, que cumplan los protocolos de control establecidos en el sistema `Tornado`.

El coste que supone cada `TBM` se cuantifica según la ecuación 10.7. Para el caso de las `Tornado BMs`, no es una estimación teórica, sino un dato definitivo, ya que son elementos pre-emplazados y pre-rutados. Los recursos necesarios son conocidos de forma previa a las fase de síntesis, emplazamiento y rutado del diseño completo.

$$\mathbf{MACRO_{over}} = \mathbf{MACRO_{lut}} + \mathbf{MACRO_{ff}} + \mathbf{MACRO_{tribuf}} \quad (10.7)$$

Sobrecoste estimado de recursos lógicos -área de silicio- por la adición de la infraestructura Tornado

Con el término $\mathbf{Area\_over_{Tornado\_est}}$ ("Sobrecoste estimado de recursos lógicos -área de silicio- por la adición de la infraestructura `Tornado`" -*Estimate Tornado Area Overhead*-) se agrupa el sobrecoste total estimado de la infraestructura `Tornado` para un diseño de las siguientes características.

- n : Número de `T-Cores`.
- x_i : Número de `TnP` para cada i `TnP-Core`. En el caso de que se trate de un `TC-Core`, sin `nano-procesadores` embebidos, se tomará el valor $x_i = 1$. Con esta aproximación, se contabilizan los recursos lógicos necesarios por la interfaz `Tornado` incluida en ese *core*.

- j : Número de `TBMs`.

$$\mathbf{Area\_over_{Tornado\_est}} = \mathbf{CTR_{over\_teor}} + \sum_{i=1}^{n} \mathbf{TnP_{over\_teor}} \cdot x_i + \mathbf{MACRO_{over}} \cdot j \quad (10.8)$$

El valor para cada uno de los parámetros de estimación de recursos se obtiene empíricamente realizando síntesis e implementaciones independientes de los distintos elementos que forman la infraestructura `Tornado`. En el capítulo de validación del sistema se obtienen estos parámetros para una tecnología concreta.

Aunque son parámetros cuyo valor se obtiene experimentalmente, se añade el término de *estimado* ya que, tal y como se ha indicado anteriormente, las herramientas de síntesis, emplazamiento y rutado pueden hacer variar el número de recursos lógicos requeridos cuando se integra la infraestructura en el diseño final. Esta variación depende de la complejidad del diseño, de la frecuencia de trabajo que se desea obtener (y se impone a las herramientas como objetivo) y de las diferentes opciones indicadas a las mismas.

11. Reconfiguración intra-task

Uno de los objetivos del sistema Tornado es la definición de un sistema de control de la auto-reconfiguración que admita la reconfiguración *intra-task* de *cores*. En concreto, se desea poder aplicar la reconfiguración a *cores* con arquitectura de Core Mixto. Por tanto, con nano-procesadores embebidos, cuyo programa pueda ser alterado mediante reconfiguración.

11.1. TnP-Cores

En la definición de los elementos integrantes de la infraestructura Tornado, se presenta el nano-procesador compatible con Tornado, denominado TnP. Los Cores Mixtos susceptibles de ser reconfigurados con el sistema propuesto, deberán integrar nano-procesadores TnPs, en cuyo caso se denominarán TnP-Cores.

La utilización de la reconfiguración parcial dinámica en los TnP-Cores pretende beneficiarse de dos características inherentes a los mismos:

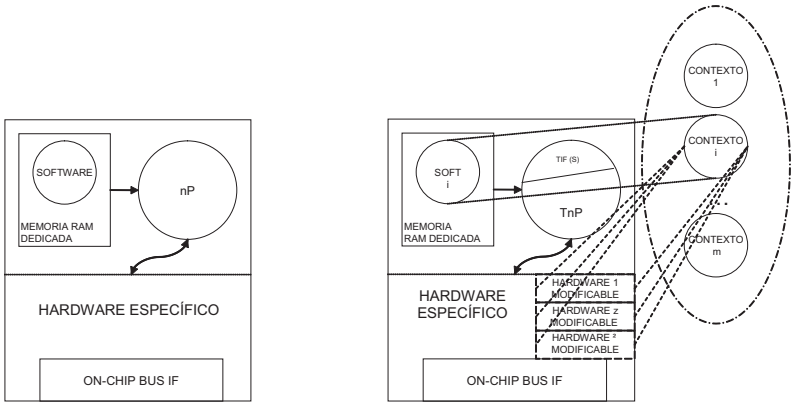
- La ubicación del circuito del hardware dedicado en el mismo entorno que el TnP-Core, lo que facilita la reconfiguración *intra-task*.
- La localización del software del procesador embebido en bloques de memoria RAM dedicados de la FPGA. Estos bloques son elementos hardware cuyo contenido puede ser alterado mediante reconfiguración.

En TnP integra una interfaz Tornado esclava (TIF (S)) para poder controlar el estado del nano-procesador durante el tiempo en el que esté siendo reconfigurado el *core*. De esta manera, se consigue evitar situaciones indeseadas en el proceso de reconfiguración, como son la contención de señales o la pérdida del control del programa, que harían inviable el cambio dinámico del contexto software.

También, a través de esta interfaz, el nano-procesador puede rechazar temporalmente un intento de reconfiguración. La habilitación y deshabilitación de la reconfiguración se realiza mediante una instrucción específica. Por tanto, se pueden establecer secciones de código o rutinas críticas que no puedan ser interrumpidas por una reconfiguración (sección 9.3).

La figura 11.1(b) sintetiza la mejora propuesta: al Core Mixto representado en la figura 11.1(a) se le reemplaza el nano-procesador integrado por un TnP. El TnP dispone de una interfaz Tornado esclava que además de influir sobre el propio nano-procesador, puede actuar sobre otros elementos del *core* que deban ser controlados durante el proceso de reconfiguración (señales de salida, *pines*, etc.). Es decir, las funciones definidas para una interfaz Tornado esclava.

Cuando se aplica un cambio de contexto sobre un TnP-Core utilizando reconfiguración *intra-task*, el *bitstream* parcial que se carga puede contener nueva información para la RAM dedicada (cambio de software) y cambios locales para el circuito del TnP-Core que realiza



(a) `Core Mixto`. **(b)** `TnP-Core`.

Figura 11.1.: Modificación de los `Cores Mixtos`.

procesamiento hardware. Esta última posibilidad se representa en la figura 11.1(b) como secciones de *hardware modificable*.

Para cada `TnP-Core`, se consideran varios contextos, implicando cada uno de ellos un *bitstream* de configuración parcial diferente. Cada contexto puede incluir un cambio del software y un circuito diferente para cada sección hardware modificable.

Los `T-Cores` (*cores* compatibles con `Tornado` pero sin `nano-procesadores` embebidos) se consideran, en cuanto a la reconfiguración *intra-task*, como un caso particular de los anteriores en los que únicamente se hace una modificación de la sección hardware.

11.1.1. Reconfiguración software

El tamaño del programa de los `nano-procesadores` está fuertemente restringido por el tamaño de los bloques de memoria RAM dedicados de las FPGAs. Esta restricción es todavía más evidente en los diseños en los que no se usan bloques de memoria dedicados, sino que se sitúa el programa en recursos de propósito general de la FPGA (generalmente LUTs).

La propia simplicidad de la arquitectura de los `nano-procesadores`, ofrece rangos direccionables reducidos para la memoria de programa. Se trata de programas compactos y optimizados, programados directamente en lenguaje ensamblador. Son, en la mayoría de los casos, máquinas de estados complejas y programables orientadas a operaciones de control. Estas secciones de control están íntimamente ligadas a la sección de procesamiento hardware del propio *core* (*hardware específico*) [129, 178, 192, 197].

El sistema `Tornado` soporta el cambio de este software mediante la reconfiguración parcial dinámica de las memorias dedicadas de las FPGAs, alterando directamente su contenido en la memoria de configuración SRAM de la FPGA. De esta forma, se emplean los propios recursos tecnológicos que el dispositivo contiene para realizar la reconfiguración (memoria de configuración y lógica asociada), sin necesidad de utilizar recursos generales de la FPGA.

Otras alternativas propuestas para el cambio dinámico del contenido de los bloques de memoria RAM dedicada, se basan en el método convencional de escribir en éstas a través de un bus de datos y un bus de direcciones. Es decir, tener direccionadas en escritura estas

memorias en el mapa de memoria del sistema. Este método se utiliza en el caso del módulo reconfigurable FPX KCPSM [192]. Para poder escribir en ellas utilizando este método, por ejemplo desde el bus *on-chip*, habría que utilizar otro puerto de la memoria y conectarlo a la interfaz *on-chip* que disponga el *core*, o incluir otra nueva.

En estos casos, se están empleando recursos generales de rutado del dispositivo. Si el número de `TnP-Cores` es elevado, los recursos de rutado necesarios supondrían un porcentaje importante de los mismos. Por ello, una solución de este tipo implica una mayor complejidad para el rutado del diseño, repercutiendo en una disminución de la máxima frecuencia de funcionamiento. Además, al no emplearse la infraestructura tecnológica que, de por si, dispone el dispositivo, se produce una disminución en la eficiencia del aprovechamiento de todos los recursos del mismo.

También se plantean soluciones para el cambio del contenido de memorias dedicadas empleando sistemas específicos de configuración como el JTAG. K. Chaplin en [220] propone usar los recursos JTAG para la reconfiguración de memorias RAM embebidas. Aunque estos recursos JTAG son dedicados, la conexión de éstos con los bloques de RAM se realiza empleando recursos de rutado generales, lo que implica que surjan los inconvenientes anteriormente mencionados.

Las aplicaciones más beneficiadas por disponer de distintos contextos software reconfigurables dinámicamente, coinciden con las citadas como campo de aplicación de los `Cores Mixtos` en el apartado 4.2. Cabe destacar los sistemas de procesamiento de protocolos de comunicación [135], criptografía [186, 189] o controladores mecatrónicos [195], entre otros.

Sirva para ilustrar el cambio de contexto software mediante reconfiguración parcial dinámica en los `Cores Mixtos` la extensión obtenida del *core* multi-procesador `FFR16` presentado en el apartado 4.2. Al módulo *Fat Processor Unit* de este *core*, se añaden tres contextos adicionales al original. Este primer contexto procesa autónomamente una unidad formateada según el formato FAT16, para ofrecer de forma autónoma al bus *on-chip* los datos del primer fichero almacenado. Los tres contextos añadidos están especializados en las siguientes tareas respectivamente: exploración y determinación del tipo de FAT, procesamiento de unidades formateadas según el estándar de Linux y carga autónoma de programas en la RAM tras el arranque (*bootloader*). Los detalles de está ampliación mediante reconfiguración *intra-task* están publicados en [129].

11.1.2. Reconfiguración hardware

La reconfiguración parcial *hardware intra-task* aplicada a los `T-Cores`, `TnP-Cores` incluidos, implica la modificación de zonas localizadas del circuito del *core*.

En el estado del arte se han presentado FPGAs comerciales de grano fino capaces de admitir este modo de reconfiguración. Y, aunque se trata de una herramienta relativamente nueva, y considerada experimental por muchos diseñadores, comienza a ser integrada de forma satisfactoria en diseños comerciales; por ejemplo, en la aplicación presentada en [146] para la modificación dinámica de los canales de comunicación de alta velocidad.

Aunque la utilización de la reconfiguración parcial hardware sobre *cores* es aún más novedosa, ya se disponen de estudios sobre los beneficios de su aplicación. En el apartado 3.2 se ha resumido el estado del arte relativo a reconfiguración *intra-task*. Algunas de las ventajas más destacables son:

- Optimización del *core*, obteniéndose como resultado módulos más pequeños y rápidos.
- Reducción de la complejidad del circuito (eliminación de registros y hardware para el control de distintos modos de funcionamiento).
- Potencial reducción del consumo de los circuitos [131].

El objetivo de la reconfiguración hardware sobre los `TnP-Cores` se centra en la modificación de los circuitos para procesamiento hardware integrados en el mismo. Esta alteración dinámica puede ser extendida incluso a módulos hardware fuertemente ligados con el `nano-procesador` embebido. Sus funciones puede ampliarse con alguna función específica, por ejemplo, con módulos para criptografía añadidos directamente a la entidad de proceso, tal y como se propone en [189].

En la parte III, dedicada a la validación del sistema, se detallan varios ejemplos de utilización de modificación parcial *hardware intra-task* sobre `Cores Mixtos` en sistemas MCSoPC. Uno de estos ejemplos es un `TnP-Core` capaz de generar una señal de alta frecuencia modulada mediante síntesis digital directa. Para cambiar el tipo de modulación, se aplica reconfiguración *intra-task*, modificándose el software y parámetros del acumulador de fase y del circuito Cordic implementado en la sección de *hardware dedicado*.

Aunque el sistema de reconfiguración hardware/software `Tornado` se plantea de un modo general, se deberán tener en cuenta las características propias de cada tecnología reconfigurable sobre la cual se desea implementar el MCSoPC. Esta circunstancia hace que puedan surgir restricciones en la aplicación real del sistema, según el dispositivo utilizado. Estas limitaciones se traducen generalmente en la obligación de mantener unas determinadas estructuras espaciales en la distribución del circuito objeto de la reconfiguración, volumen del mismo o restricciones de rutado.

11.2. Flujo de diseño para reconfiguración intra-task

El flujo de diseño de SoPCs propuesto por las empresas vendedoras de FPGAs permite obtener los ficheros de configuración de los dispositivos (*bitstreams* completos y parciales) usando exclusivamente sus herramientas.

Los *bitstreams* completos tienen la información de configuración de todos los elementos de la FPGA. Se incluye en estos ficheros la configuración de los recursos lógicos, del rutado y de los parámetros específicos del dispositivo. Para la reconfiguración *intra-task*, los *bitstreams* parciales tienen información de configuración de las secciones de la FPGA que varían respecto a un diseño global original tomado como referencia a la hora de la generación de los mismos.

Para poder aplicar el sistema `Tornado` a diseños MCSoPC, además de utilizar las herramientas propias del fabricante, se requieren unos elementos hardware específicos (controlador, interfaces, `TnP`, etc.) para esa tecnología, agrupados bajo el término infraestructura `Tornado`, y unas herramientas que faciliten el diseño. Este conjunto de herramientas definen el *framework* `Tornado`.

Aunque un *framework* es dependiente de la tecnología utilizada, a continuación se establece un diagrama de flujo general para el diseño de sistemas MCSoPC con `TnP-Cores` sobre los que se aplica únicamente reconfiguración *intra-task*. Por tanto, el *framework* `Tornado`

para una determinada tecnología, deberá completar los aspectos de este diagrama que las herramientas del fabricante del dispositivo no cubran.

En la figura 11.2 se muestra el diagrama propuesto. El MCSoPC objeto de diseño tiene n `TnP-Cores` y m contextos para cada `TnP-Core`.

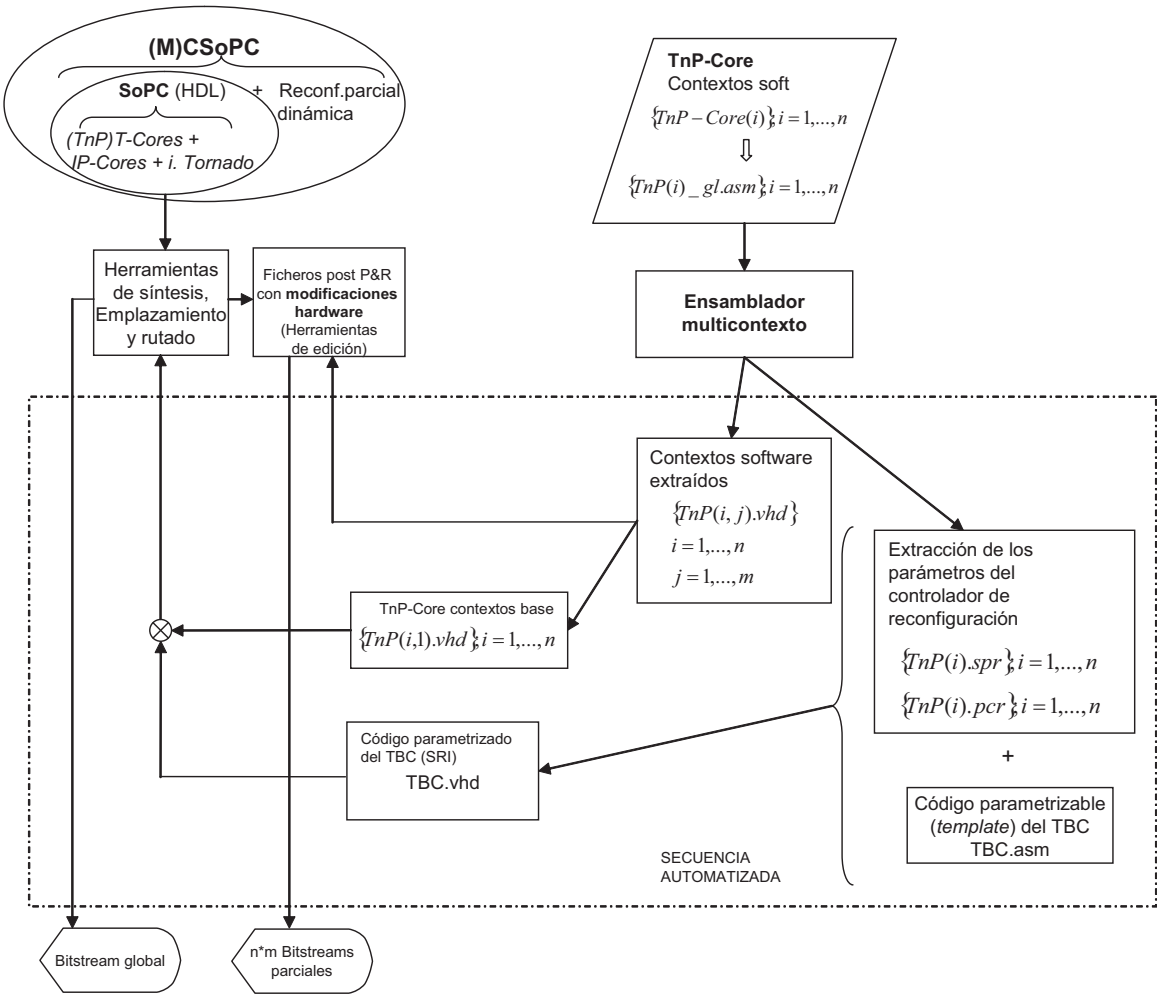


Figura 11.2.: Diagrama de flujo de diseño con reconfiguración *intra-task*.

El diseño tiene dos puntos de partida:

- **Diseño del SoPC:** Mediante las herramientas del fabricante se diseña el SoPC base. El SoPC incluye IP-Cores, `T-Cores` (en el caso de incluir `nano-procesadores` serán `TnP-Cores`), otros elementos adicionales que requiera el sistema y la infraestructura `Tornado` correspondiente a esta tecnología. Al aplicar la reconfiguración parcial dinámica sobre el SoPC se obtendrá un CSoPC (multi-procesador en su caso).

- **Escritura de los códigos fuente de los `TnPs`:** El código fuente de cada `TnP-Core`, con todos sus posibles contextos, se escribe en un fichero de texto (extensión asm).

Por tanto, el número de ficheros de código globales será n, con la siguiente secuencia de nombres:

$$\{\ TnP(i)\_gl.asm\ \};\quad i = 1, \cdots, n \tag{11.1}$$

Cada fichero global de código fuente se ensambla mediante un programa ensamblador específico del *framework* `Tornado`. Para cada fichero se realizan las siguientes acciones:

- **Separar cada contexto** software. Las secciones de código para cada contexto están separadas mediante directivas.
- **Ensamblar cada contexto** software para ser ejecutado por el `TnP`.
- **Formatear el código máquina en ficheros VHDL** (extensión vhd) que describan la inicialización de los bloques de memoria RAM dedicada de la FPGA. En cada fichero VHDL, se describe la inicialización de un contexto para un `TnP-Core`. Por tanto, la secuencia de ficheros VHDL obtenida será:

  $$\{\ TnP(i,j).vhd\ \};\quad i = 1, \cdots, n;\quad j = 1, \cdots, m \tag{11.2}$$

  De esta secuencia de ficheros, se seleccionan todos los del contexto 1 y se integran en la herramienta de diseño del fabricante para generar el SoPC base. Este conjunto de ficheros se identifica como:

  $$\{\ TnP(i,1).vhd\ \};\quad i = 1, \cdots, n \tag{11.3}$$

- **Extraer del código fuente la configuración de las señales `SPR` y `PCR`** para que el `TBC` las active adecuadamente según el `TnP-Core` correspondiente. Esta configuración, que se indica mediante directivas en el código, puede ser distinta para cada `TnP-Core`.

  La información extraída, junto con una plantilla (*template*) pre-diseñada de código para el controlador de reconfiguración `TBC`, permite generar automáticamente un código fuente específico para el `TBC` del diseño concreto que se está realizando. Tras ser ensamblado, se obtiene un fichero VHDL, el TBC.vhd, utilizado para inicializar el contenido del bloque de memoria correspondiente. Este fichero VHDL, se integra en la herramienta del fabricante para generar el diseño SoPC base.

Con todos los elementos necesarios para generar un primer diseño, el SoPC base, se genera mediante las herramientas del fabricante el *bitstream* global de configuración. Este *bitstream* se utiliza para la primera configuración del dispositivo.

También se obtiene un fichero que incluye la información detallada de rutado y configuración de la FPGA (fichero *post Place & Route*). La inserción de las modificaciones hardware y las diferentes configuraciones de los bloques de memoria RAM dedicados se realizan en las localizaciones correspondientes. De esta forma, se generan $\mathbf{n} \cdot \mathbf{m}$ ficheros *post Place & Route* parciales. A partir de estos ficheros se obtienen el mismo número de *bitstreams* parciales contrastando el fichero *post Place & Route* completo original y cada uno de los modificados. Es decir, un *bitstream* parcial por cada contexto y `TnP-Core`.

La palabra de petición de reconfiguración `CRW`, al estar compuesta por un número de `T-Core` y el contexto solicitado para el mismo, es un código único. El `TBC`, que conoce el número de *bitstreams* parciales y la colocación de los mismos en el sistema de almacenamiento, es capaz de identificar de forma unívoca el código del `CRW` con uno de los *bitstream* parciales e iniciar la carga del mismo.

El *framework* `Tornado` debe proveer de las herramientas auxiliares necesarias (como el ensamblador multi-contexto) y facilitar el diseño, automatizando las distintas fases. En la figura 11.2, los procesos agrupados en un rectángulo de línea discontinua pueden ser automatizados mediante *scripts* del *framework* de tal forma que se simplifiquen las tareas al diseñador.

12. Reconfiguración inter-task

La reconfiguración *inter-task* implica el cambio completo de un *core* por otro que puede tener una función completamente distinta. Para poder realizar esta operación, en un sistema donde el resto de los *cores* sigan funcionando y la actividad en el bus *on-chip* se mantenga, se requiere que la conexión al bus del *core* que se reconfigura se mantenga invariable para los diferentes contextos del mismo. Además, las señales de salida de ese *core* hacia el bus *on-chip* y hacia *pines* del dispositivo deben tomar un valor de alta impedancia o, en general, un valor conocido. Los circuitos que realizan esa conexión se denominan *Bus-Macros*.

Las *Bus-Macros*, además, sirven como elementos para guiar el rutado entre las secciones fijas y reconfigurables en algunas tecnologías que no permiten restringir las rutas por caminos específicos.

El sistema de control `Tornado`, cuya finalidad inicial es la de controlar la reconfiguración *intra-task* de `TnP-Cores` y `T-Cores`, se puede utilizar para controlar la reconfiguración *inter-task* actuando sobre *Bus-Macros* especiales que admitan el protocolo de reconfiguración de *Tornado*.

12.1. Tornado Bus-Macro

Para poder realizar reconfiguraciones *inter-task* controladas mediante el sistema `Tornado`, se necesita incluir un elemento adicional a la infraestructura de control de la reconfiguración. Este elemento es una *Bus-Macro*, circuito que permite conectar un *core* al bus *on-chip*, teniendo en cuenta que durante el cambio de contexto del mismo, se reemplaza toda la configuración de la FPGA referente a ese *core* por otra diferente. Por tanto, esa *Bus-Macro* compatible con el sistema de control propuesto, denominada `Tornado` *Bus-Macro* (`TBM`), debe:

- Desconectar las señales de salida del *core* durante el estado de reconfiguración del mismo.
- Atender a las señales de reconfiguración. Por tanto, debe incluir una interfaz `Tornado` esclava.
- Incluir un mecanismo mediante el cual el *core* que se va reconfigurar pueda indicar si está habilitada o no la reconfiguración.

Como se indica, la interfaz `Tornado` esclava debe encontrarse en el propio circuito *Bus-Macro*, que no se reconfigura. De esta forma, durante el proceso de reconfiguración del *core*, no se pierde el control de la misma. Puesto que la `TBM` dispone de una interfaz `Tornado` esclava, está actuando realmente como un *wrapper*[1] [110, 190, 221], envolviendo al *core* al

[1]Este término se emplea para describir a los circuitos utilizados para dotar un módulo de una interfaz que previamente no disponía. De esta forma, se hace compatible a ese módulo con una determinada especificación sin necesidad de modificar el código HDL del mismo.

que se conecta, de forma que se hace compatible con Tornado. Esta característica, permite utilizar IP-Cores no compatibles con Tornado en procesos de reconfiguración *inter-task*.

De forma general, para cumplir con los requisitos descritos, los elementos básicos que componen una TBM son:

- Recursos para desconectar o fijar a un estado conocido las señales de salida del *core*. Si la tecnología dispone de *buffers* triestado, éstos son una solución adecuada, ya que permiten una implementación simple del circuito, al poder asignar a las salidas los mismos el estado de alta impedancia durante la reconfiguración.

- Un biestable que pueda ser escrito por el *core* para indicar el estado de admisión de la reconfiguración.

- Una máquina de estados que implemente el *handshake* Tornado e interaccione con el dato almacenado en el biestable anteriormente descrito. Esta máquina de estados se debe realizar con la menor cantidad de recursos lógicos posibles, de forma que el circuito resultante sea sencillo y fácilmente integrable.

  En la figura 12.1 se representa el diagrama de estados de la máquina que se debe implementar. Las entradas son: la señal STB_RECONF(i) de Tornado y la señal ALLOW_CORE que proviene del biestable anterior. La señal RECONF_ACK es la única señal de salida.

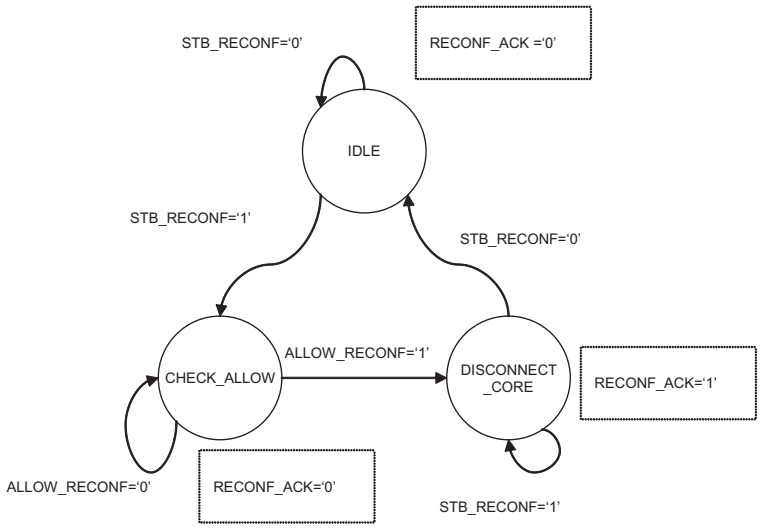


Figura 12.1.: Máquina de estados implementada en la Tornado Bus-Macro.

El circuito TBM es una unidad realizada con los recursos disponibles en la tecnología utilizada. Se integran en el diseño como un elemento ya sintetizado y con los recursos necesarios asignados y rutados. De esta forma, se puede restringir el área del *core* a una zona delimitada de la FPGA. El *bitstream* que se carga a petición del TBC para la reconfiguración de ese IP-Core, únicamente contiene información de configuración de esa área concreta.

12.2. Flujo de diseño para reconfiguración inter-task

En el flujo de diseño de CSoPCs con reconfiguración *inter-task* controlada por `Tornado`, se requiere introducir pocas modificaciones en el flujo de diseño que defina el fabricante del dispositivo reconfigurable.

En la figura 12.2 se muestran los cambios introducidos: en el diseño del sistema se debe añadir la infraestructura `Tornado` (controlador, interfaces, etc.) incluyendo las `Tornado Bus-Macros` necesarias para los *cores* que se quieran reconfigurar dinámicamente.

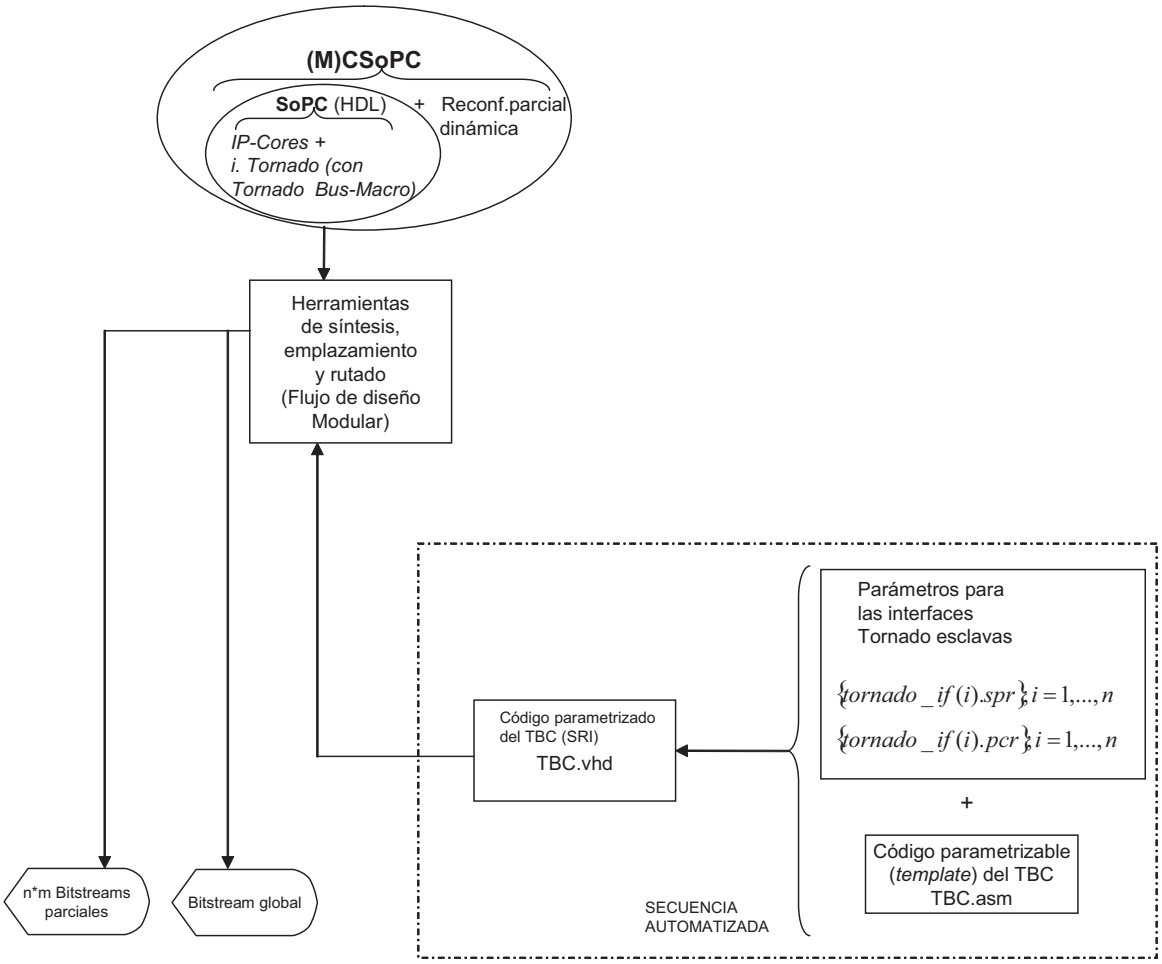


Figura 12.2.: Diagrama de flujo de diseño con reconfiguración *inter-task*.

También se incluye en el diseño, el código fuente del programa del controlador de reconfiguración `TBC`. En este caso, la información para la parametrización del mismo, proviene del comportamiento que se desee especificar para cada interfaz `Tornado` esclava disponible en el sistema.

Para finalizar, cabe destacar que el sistema `Tornado`, tal y como se ha especificado, permite disponer de ambos tipos de reconfiguración simultáneamente en un mismo diseño. Sin embargo, esta posibilidad puede resultar poco práctica debido a la complejidad resultante.

Parte III.

Validación y Evaluación del sistema

13. Plan de pruebas y tecnología

En los capítulos anteriores se ha presentado el sistema `Tornado` de forma teórica. Es un sistema aplicable a distintas tecnologías y suficientemente flexible como para poder ser adaptado a distintas necesidades y arquitecturas. Sin embargo, para poder analizar la viabilidad del mismo es necesario aplicarlo a una tecnología en concreto y realizar una serie de diseños que permitan experimentar la propuesta planteada.

La tecnología elegida para la experimentación es la del fabricante Xilinx. Los dispositivos de las familias FPGA Virtex [40] y su versión de bajo coste Spartan-II[1] [75], admiten la reconfiguración parcial dinámica, siendo además dispositivos comerciales ampliamente utilizados para diseños con lógica reconfigurable convencionales (no dinámicamente reconfigurables).

Para realizar los ensayos se ha definido un plan de pruebas con el objetivo de validar los diferentes elementos definidos en `Tornado`.

13.1. Plan de pruebas

El plan de pruebas está orientado a verificar la aplicabilidad de `Tornado` con la tecnología actualmente disponible de forma comercial.

Se plantean tres plataformas de pruebas. Para cada plataforma se realizan varios diseños e implementaciones diferentes, con el fin de probar distintos tipos de reconfiguración. Con la primera plataforma (`Video-Multimaster`) se experimenta la reconfiguración *intra-task* software. Ésta se amplia en la segunda plataforma (`Adaptive Multi-Transmitter -AMT-`) a una reconfiguración *intra-task hardware/software*. Finalmente, con la tercera plataforma (`IP-Cores Bajo Demanda`), se evalúa la ampliación del sistema de control propuesto a una reconfiguración *inter-task*. El plan de pruebas propuesto se recoge en la tabla 13.1.

13.2. Reconfiguración parcial dinámica en dispositivos Virtex

En los últimos años, la familia de dispositivos FPGA Virtex ha sido a alternativas más utilizada para experimentación con lógica dinámica y parcialmente reconfigurable basada en FPGAs comerciales de grano fino.

La estructura interna de estas FPGAs se describe en el apartado 2.3 de esta tesis. En la figura 13.1 se representa la distribución de los recursos lógicos programables en un dispositivo de la familia Spartan-II. Se diferencian cuatro tipos de recursos: los bloques lógicos configurables (*Configurable Logic Blocks* -CLBs-) cuyos elementos están divididos en dos *slices*; los bloques de entrada-salida (*Input/Output Blocks* -IOBs-); los bloques de memoria RAM dedicados (BRAMs); y los dispositivos para la gestión del reloj. Éstos, en concreto, para

[1] En el resto del documento se empleará el término Virtex para referirse a todas las familias de dispositivos similares.

Tabla 13.1.: Plan de Pruebas.

Nombre de la plataforma	**Reconf. *intra-task* software**	**Reconf. *intra-task hardware/ software***	**Reconf. *inter-task***	**Sistema de evaluación**	**Objetivo**
`Video-Multimaster`	Sí	No	No	Simulación/-Prototipo	Evaluación de los tiempos de Reconfiguración. Validación del modelado para la estimación de los recursos de área
`Adaptive Multi-Transmitter (AMT)`	Sí	Sí	No	Simulación	Evaluación de las mejoras en área/tiempo por la aplicación de la auto-reconfiguración parcial dinámica
`IP-cores Bajo Demanda`	No	No	Sí	Simulación/-Prototipo	Evaluación reconfiguración inter-task (`Tornado Bus-Macro`)

la familia Spartan-II son los bucles enganchados en retardo (*Delay-Locked Loops* -DLLs-) y para la Virtex los gestores digitales de reloj (*Digital Clock Managers* -DCMs-).

Los recursos lógicos en la FPGA están distribuidos de forma regular formando una matriz cuyas posiciones se identifican en la nomenclatura de Xilinx con una "R" indicando la fila (*row*) y una "C" para identificando a la columna (*column*).

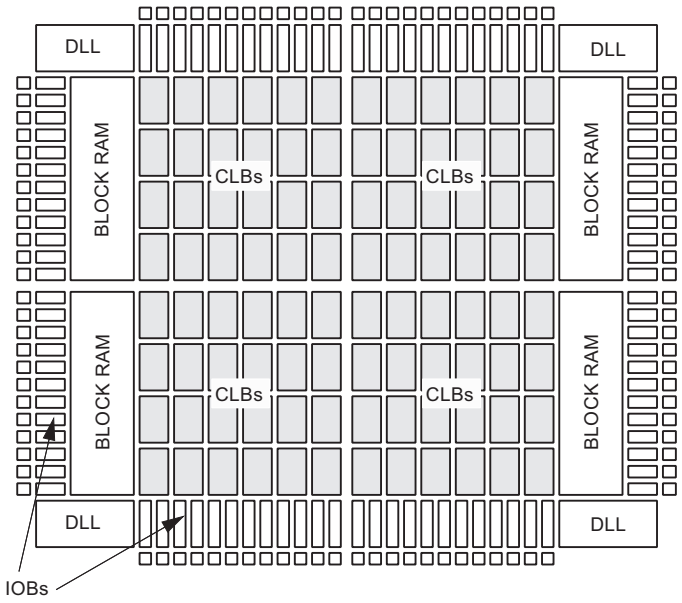


Figura 13.1.: Distribución de los recursos lógicos en el dispositivo más pequeño de la familia Spartan-II.

La configuración de estos elementos y del rutado se escribe en la memoria de configuración SRAM de la FPGA. La mínima sección del dispositivo que se puede cambiar en una única operación de reconfiguración parcial está definida por un *frame*. Este término, propio de Xilinx, especifica una unidad de la memoria SRAM de configuración de la FPGA. Un *frame*

tiene un bit de anchura y se extiende desde la fila superior de la matriz hasta la fila inferior.

La figura 13.2 muestra la distribución de dos tipos de *frames* en la memoria de configuración de la FPGA. Se distinguen tres zonas: a la izquierda y a la derecha, BRAM0 y BRAM1 incluyen los *frames* para la configuración de los datos de los bloques de RAM dedicados (BRAM *frames*) y en el centro de la matriz se sitúan los *frames* de configuración de los CLBs (CLB *frames*).

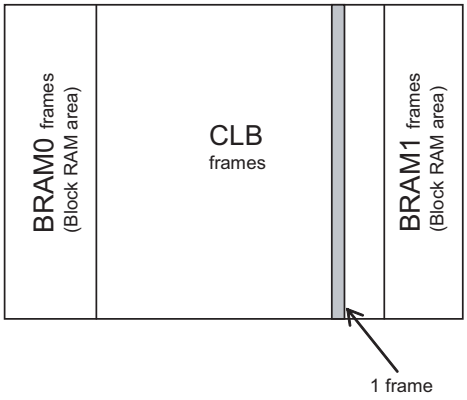


Figura 13.2.: Distribución de los diferentes tipos de *frames* en un dispositivo Virtex.

Los CLB *frames* contienen todos los datos de configuración para todos los elementos programables de la FPGA incluyendo los valores de las LUTs, lógica auxiliar, IOBs, elementos de control de los BRAMs y las interconexiones. De esta forma, se puede acceder a la configuración de estos recursos mediante una operación de escritura sobre el puerto de configuración de la FPGA, sin interrupción en el funcionamiento del circuito[2].

Además, como el contenido inicial de los biestables de los CLBs no se especifica en el *bitstream* de configuración, y por tanto, tampoco en el *frame*, los datos almacenados en los mismos no se ven modificados en el proceso de reconfiguración parcial.

En cuanto a los bloques de memoria RAM dedicada, se debe tener en cuenta que al escribir los BRAM *frames* en la memoria de configuración se está alterando su contenido. Por ello, si la aplicación accede a los mismos durante este periodo transitorio puede leer valores no válidos. Esta situación debe ser tenida en cuenta por el sistema de control de la reconfiguración que se esté utilizando en el circuito.

Los *frames* están agrupados en unidades superiores denominadas *frame columns* [80, 81]. El área de la memoria de configuración de los CLB *frames* tiene cuatro categorías de *frame columns*: una columna central de 8 *frames*, columnas de 48 *frames* para la configuración de los CLBs, dos columnas de 27 frames que especifican el conexionado de los BRAMs y dos columnas para la configuración de los IOBs (54 frames por columna). En la figura 13.3 se muestra la distribución y numeración de las *frame columns* en la memoria de configuración, siguiendo una distribución de *ping-pong*[3].

[2] En el caso de que se estén utilizando LUTs como memoria distribuida, si que se interfiere en su contenido en el proceso de reconfiguración, ya que en el *frame* se especifica el contenido inicial de la misma [82].

[3] A partir de la columna central, la parte derecha de la memoria de configuración está ocupada por los *frame columns* con direcciones impares y la parte izquierda con los de direcciones pares.

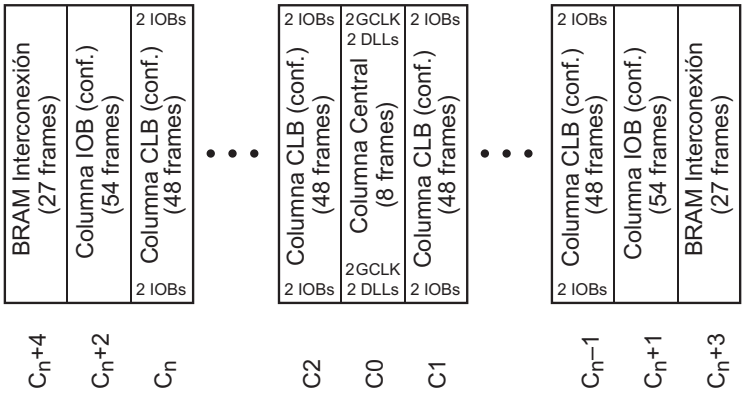


Figura 13.3.: Distribución de las *frame columns* en la memoria de configuración.

Dentro de cada *frame column*, los *frames* se numeran secuencialmente desde el *frame* que se encuentre más cercano al centro hasta el más alejado, tal y como se muestra en la figura 13.4.

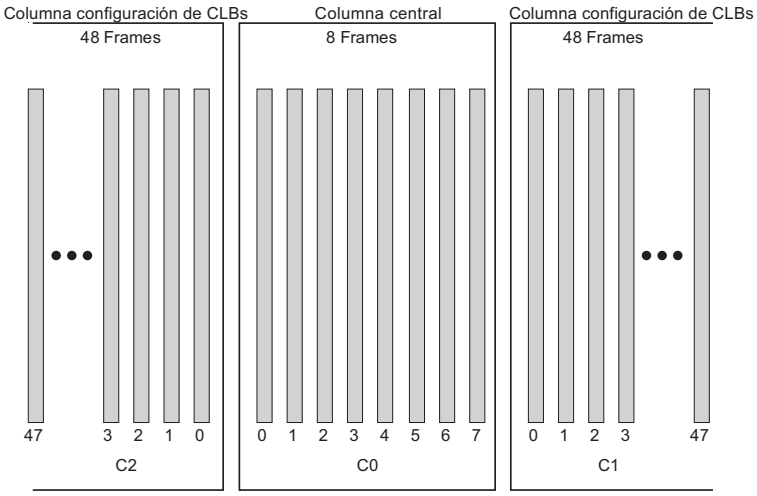


Figura 13.4.: Distribución de los *frames* en los *frame columns*.

La dirección de un *frame* en la memoria de configuración está compuesta por dos números: el *major address*, que es el número de *column frame*; y el *minor address*, que es el número del *frame* en esa columna. Por tanto, los *bitstream* parciales de configuración incluyen las direcciones y los contenidos de los *frames* que se deben modificar, además de comandos específicos para el puerto de configuración.

Pero para poder aplicar la reconfiguración parcial dinámica sobre estos dispositivos, se deben tener en cuenta las siguientes restricciones generales [98]:

- La altura de una sección reconfigurable deberá ser la de toda la matriz de recursos lógicos.
- La anchura ocupada por un módulo reconfigurable deber ser múltiplo de 4 *slices* (2 CLBs). Por ejemplo, un módulo reconfigurable podría comenzar en coordenadas X=0, 4, 8, 10, etc. con anchuras de w=4, 8, 12, etc. *slices*.
- Toda la lógica que se defina en la anchura ocupada por ese módulo reconfigurable pertenece a ese módulo. Esto incluye tanto a los elementos que forman parte del CLB como a los *buffers* triestado (*Tbufs*), IOBs, multiplicadores y rutado.
- La lógica de reloj (*buffers* globales, DCM o DLL, etc.) tiene *frames* separados de los utilizados para configurar los CLBs y BRAMs.
- La configuración de los IOBs situados en el lateral derecho y en el izquierdo de la matriz reconfigurable se realiza mediante los *CLB frames* correspondientes a las columnas extremas.
- Los límites definidos para un determinado módulo reconfigurable no pueden ser cambiados dinámicamente, siendo fija la posición y el área que ocupa.
- La comunicación entre distintos módulos reconfigurables se debe realizar utilizando unos elementos fijos que definan rutas únicas de configuración. Estas herramientas se denominan *Bus-Macros* y se encuentran pre-rutadas antes de realizar la fase de emplazamiento y rutado de todo el diseño de la FPGA (figura 13.5).

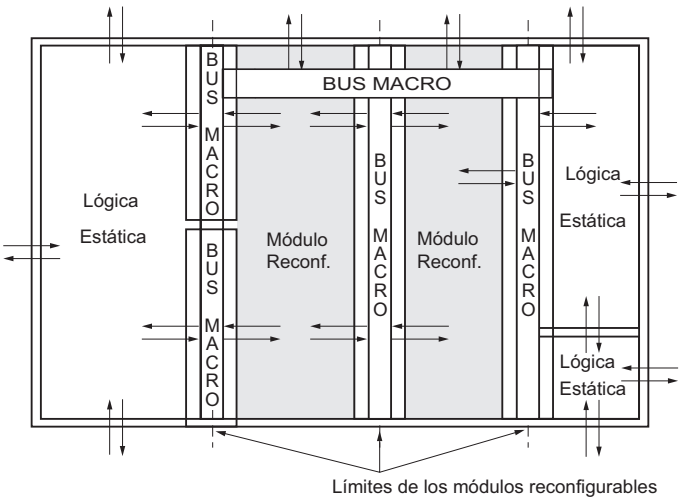


Figura 13.5.: Conexión entre módulos reconfigurables para aplicar reconfiguración *inter-task* en dispositivos Virtex de Xilinx .

Esta restricción se debe tener en cuenta cuando el tipo de reconfiguración parcial dinámica que se pretende aplicar es de tipo *inter-task*, lo que obliga al cambio completo del módulo reconfigurable.

- Las señales globales de *reset* y *set* no pueden utilizarse de manera local sobre un módulo reconfigurable, lo que implica que sea necesario disponer de señales de inicialización independiente para cada módulo.

14. Herramientas hardware

A continuación se presenta el conjunto de elementos hardware empleados en las plataformas de validación. Se incluyen: la especificación estándar utilizada, la infraestructura `Tornado` para la tecnología Virtex y el prototipo desarrollado.

14.1. Especificación estándar para interconexión de IP-Cores Wishbone

En el apartado 3.5.2 del estado del arte se presentan brevemente varias de las especificaciones para interconexión de IP-cores más utilizadas.

`Tornado` define un sistema de control de la auto-reconfiguración parcial dinámica de forma independiente a la especificación elegida. Sin embargo, la validación de `Tornado` requiere el diseño de una infraestructura y unas plataformas para las que se requiere utilizar una especificación que homogenice las interfaces y facilite la reutilización de los distintos módulos.

Los `IP-Cores`, `T-Cores` y elementos propios de la infraestructura `Tornado` como el controlador de reconfiguración deben ser también compatibles con la especificación seleccionada. La elección de la misma ha estado condicionada por la selección de especificación que realizó el Grupo de Investigación APERT para el diseño de sus *cores*.

El disponer de una infraestructura `Tornado` compatible con la especificación utilizada en los distintos diseños permite reutilizar *cores* previamente diseñados en las plataformas de validación y, además, actualizar diseños SoPC a sistemas que se beneficien de la auto-reconfiguración parcial dinámica.

La especificación seleccionada por APERT fue Wishbone, detallándose en el trabajo de U. Bidarte [152] los criterios utilizados para realizar esta selección. A continuación, se resumen los argumentos más importantes que sirvieron para la elección de esta especificación:

- **Especificación libre y sin licencias:** La tabla 3.1 del apartado 3.5.2 representa la diferente casuística de licencias para distintas especificaciones. Como se puede observar en ésta, la especificación Wishbone es una especificación abierta y libre de licencias. La iniciativa Wishbone fue puesta en marcha por la compañía Silicore en junio de 1999 [154]. En enero de 2001 se retiraron las licencias quedando en el dominio público a través de la asociación Opencores en su página web `http://www.opencores.org`.

  Esta asociación no tiene ánimo de lucro y su principal objetivo es el de crear un entorno de diseño, intercambio y reutilización de *cores* para diseño digital, es decir, se puede equiparar al desarrollo software bajo la licencia *General Public License* (GPL). El objetivo de esta asociación se concreta en las siguientes tareas:

  - Desarrollo de arquitecturas estándar para interconexión de *cores* en SoC.

- Especificación de metodologías y herramientas para desarrollo de *cores* y plataformas de uso libre.
- Desarrollo de *cores* y plataformas de uso libre.
- Creación de un centro de documentación de *cores* y plataformas.

Tras un análisis de las especificaciones existentes [151], los usuarios de este grupo eligieron Wishbone para el desarrollo de sus *cores*, facilitando de esta forma la reusabilidad, integración e intercambio de sus diseños.

El hecho de que sea una especificación libre permite disponer del código fuente de distintas implementaciones de interfaces y buses, además de poder diseñar nuevas versiones sobre las que se dispone un control y conocimiento a bajo nivel. Además, la elección de esta especificación habilita la utilización de un gran número de *cores* disponibles en la página web de *Opencores* con licencia GPL.

- **Flexibilidad.** Un elemento técnico destacable de esta especificación es su flexibilidad. La diversidad de topologías de bus que soporta (bus compartido, punto a punto, *dataflow* y *crossbar*) permite una aplicación de la especificación óptima para cada diseño.

  También a nivel de señalización Wishbone es muy flexible. Únicamente se definen un pequeño subconjunto de señales obligatorias, pudiéndose emplear el resto de señales definidas en la especificación cuando sean necesarias para cumplir los requisitos de los diseños (arbitrajes complejos, modos de acceso al bus avanzados, gestión de errores, etc.).

- **Independencia de la tecnología.** La especificación Wishbone admite el diseño de buses tanto con *buffers* triestado como con multiplexores. Además, el *handshake* definido para la transferencia de datos puede emplearse de forma asíncrona para la transferencia de una unidad de información en un único ciclo de reloj (diseño con ASICs) o de forma síncrona (diseño con FPGAs).

El documento de especificación Wishbone está disponible de manera gratuita en [154]. La última versión es la B.3, del 7 de septiembre de 2002. A continuación se resumen los aspectos principales de dicha especificación.

El intercambio de datos se realiza siempre entre un elemento maestro y otro esclavo y es iniciado por el primero. Se emplea un protocolo síncrono del tipo *handshake* en base a las señales mostradas en la figura 14.1. Las terminadas en "_i" son señales de entrada y las terminadas en "_o" son señales de salida. Todas las señales de control son activas a nivel alto.

Las señales comunes a módulos maestros y esclavos son las siguientes:

- `clk_i`: Reloj global. Todas las señales (también *rst_i*) son síncronas con los flancos de subida de esta señal de reloj.
- `rst_i`: Inicializa todos los *cores* conectados al bus.

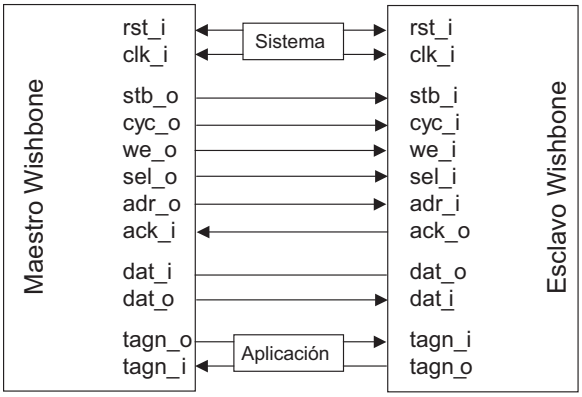


Figura 14.1.: Conexión de *Cores* mediante un único bus Wishbone.

- `tagn_i`, `tagn_o`: Reset global. Son señales de entrada y de salida, respectivamente, definidas por el diseñador en función de las necesidades de la aplicación. Pueden emplearse tantas como sean necesarias, tanto en bloques maestros como esclavos.

A continuación se describen las señales de control diferenciadas en los *cores* maestros y esclavos. Son las empleadas para el control de las transferencias tipo *handshake*. Se muestran en parejas, la primera asociada al *core* maestro y la segunda asociada al *core* esclavo:

- `stb_o`, `stb_i`: Señal de requerimiento de acceso al bus. El maestro activa esta señal para indicar el comienzo de un ciclo de bus.
- `we_o`, `we_i`: Señal de escritura. El maestro pone esta señal a nivel alto para indicar que el ciclo es de escritura y pone la señal a nivel bajo para indicar que el ciclo es de lectura.
- `adr_o`, `adr_i`: Bus de direcciones activado por el maestro. Los bits más significativos se emplean para seleccionar el *core*. De la parte menos significativa, no empleada para la activación de *cores*, cada *core* únicamente recibe los bits necesarios para su direccionamiento interno.
- `sel_o`, `sel_i`: Selección de la parte del bus de datos válida en la transferencia. Se emplea cuando el tamaño del bus y el de la palabra transferida en un acceso no coinciden. En ese caso el maestro indica mediante la señal `sel_o` en que parte del bus de datos se encuentra la información válida. Es una señal opcional.
- `cyc_o`, `cyc_i`: Petición del control del bus. El maestro mantiene esta señal activada mientras dure un ciclo de bus. Es de utilidad en las transferencias de bloque o transferencias múltiples, en las que un ciclo de bus se corresponde con varias transferencias de datos. El maestro activa la señal `cyc_o` durante la primera transferencia y permanece activada hasta la última transferencia. En caso de emplear ciclos unitarios esta señal es opcional, ya que coincide con la señal `stb_o`.
- `ack_i`, `ack_o`: Validación de la transferencia. El esclavo activa esta señal para indicar la terminación de un ciclo de bus normal.

- `err_i`, `err_o`: Error en la transferencia. El esclavo activa esta señal para indicar la terminación de un ciclo de bus con error. La causa de este error y la respuesta generada por el maestro han de ser definidas por el diseñador. Es una señal opcional.
- `rty_i`, `rty_o`: Petición de repetición. El esclavo activa esta señal para indicar que no está disponible para aceptar o enviar el dato, por lo que el ciclo de acceso ha de ser intentado posteriormente. El diseñador ha de especificar cuándo y cómo se repetirá el intento de acceso. Es una señal opcional.

Para el bus de datos, en caso de implementarse mediante multiplexores, están previstas dos señales en cada *core*: `dat_i`, bus de datos de entrada, y `dat_o`, bus de datos de salida. En caso de implementarse mediante *buffers* triestado no es necesaria la diferenciación entre el *array* de entrada y el de salida, por lo que se emplea un único bus de datos denominado `dat`.

En el caso de las señales que contienen más de un bit (direcciones, selección y datos) el estándar indica que el tamaño del bus queda abierto y ha de ser especificado por el diseñador.

Se definen tres tipos de accesos de lectura y escritura: simple o unitario, de bloque o múltiple y de lectura, y finalmente, modificación y escritura de manera indivisible. En los primeros, cada acceso se corresponde con la lectura o escritura de un único dato. En los segundos, se realiza un acceso múltiple, es decir, se transfiere un conjunto de datos o bloque. Los terceros se emplean para operaciones indivisibles de tipo semáforo. A modo de ejemplo la figura 14.2 muestra un ciclo de lectura simple. El protocolo de bus se explica a continuación:

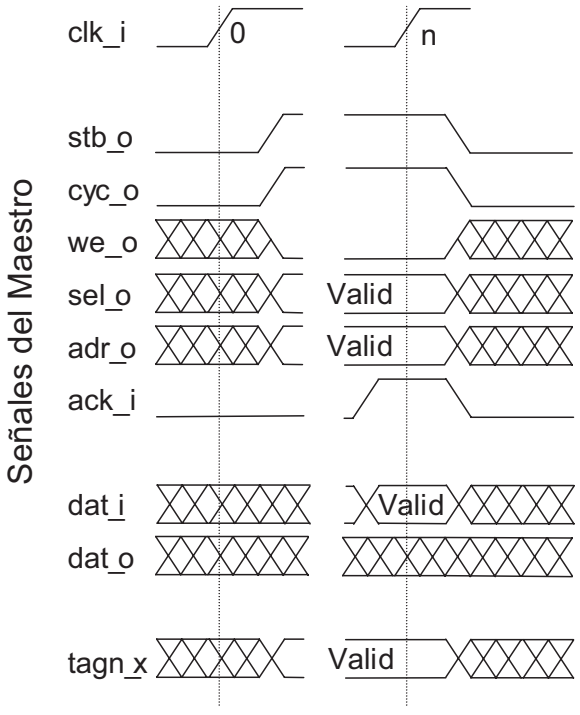


Figura 14.2.: Ciclo de lectura simple en Wishbone.

- En el flanco activo del ciclo 0 de `clk_i` se producen los siguientes eventos:
  - El maestro presenta `adr_o`, `sel_o` y `tagn_o`.
  - El maestro pone `we_o` a nivel bajo para indicar ciclo de lectura.
  - El maestro activa `stb_o` y `cyc_o` para iniciar el acceso.

- Un tiempo de establecimiento (*setup*) antes del flanco activo del ciclo `n` se producen los eventos descritos a continuación (`n` puede ser 1 si se realiza un acceso por ciclo). En este caso la activación de las señales `ack_o`, `err_o` y `rty_o` se realiza únicamente mediante lógica combinacional, sin registros, para poder estar estables para el flanco activo del ciclo 1. El esclavo puede retrasar la activación de estas señales tantos ciclos como requiera. De esta forma, el protocolo basado en el reconocimiento por parte del esclavo de la petición realizada por el maestro, permite ajustar la velocidad de transferencia.
  - El esclavo detecta todas las señales del maestro, realiza el procesamiento necesario y presenta las señales `dat_o` y `tagn_o`.
  - El esclavo responde activando `ack_o` para validar los datos. Si hubiera error o hubiera que posponer el acceso activaría `err_o` y `rty_o`, respectivamente.
  - El maestro detecta las señales `tagn_i` y `ack_i` y se prepara para registrar los datos.

- En el flanco activo del ciclo `n` de `clk_i`.
  - El maestro registra los datos en `dat_i` y las señales dependientes de la aplicación en `tagn_i`.
  - El maestro desactiva `stb_o` y `cyc_o` para indicar la finalización del acceso.

Si la anchura de los operandos manejados por el sistema es mayor que la del bus de datos, el estándar Wishbone permite organizar los datos en modo *Big Endian* o en modo *Little Endian*. En el primer modo de ordenación la parte más significativa de un operando se almacena en la dirección más baja, mientras que en el segundo modo se almacena el la dirección más alta.

El estándar Wishbone permite cuatro tipos básicos de topologías para la interconexión de los *cores*. A continuación se explican brevemente:

1. **Punto a punto.** Es la topología más sencilla. Permite conectar un maestro con un esclavo.

2. **Flujo de datos o *pipeline*.** Se emplea cuando el procesamiento de datos se realiza de manera secuencial. Cada módulo dispone de una interfaz de tipo maestro y otra de tipo esclavo. Permite incrementar la velocidad de ejecución porque todos los *cores* trabajan en paralelo. Para poder emplear esta topología, el algoritmo de procesamiento se ha de poder resolver de forma secuencial y cada uno de los *cores* ha de tardar el mismo tiempo en ejecutar su parte.

3. **Bus compartido.** Permite conectar varios maestros con varios esclavos. Sólo se puede realizar una transferencia en cada instante, por lo que, en caso de haber más de un maestro, se requiere un bloque que realice las funciones de árbitro del bus. Este árbitro establece cuando gana cada maestro el control del bus compartido. El diseñador ha de establecer los criterios de arbitraje. Su mayor ventaja reside en que los recursos necesarios para su implementación suelen ser reducidos en comparación con otras posibilidades. Su mayor inconveniente consiste en que sólo se puede ejecutar una transferencia en cada instante, por lo que, en caso de existir más de un maestro, resulta necesario esperar hasta ganar el control del bus.

4. **Matriz de interconexión.** Se emplea cuando se requiere realizar más de una transferencia de forma concurrente. Permite la ejecución simultánea de tantas transferencias como *cores* maestros existan, siempre que no accedan a un mismo *core* esclavo. En este caso se requiere un árbitro para indicar cuando un maestro gana el acceso a un determinado esclavo. Por lo tanto, se establecen varios canales de comunicación simultáneos. Esta topología consigue tasas de transferencia mayores que la de bus compartido, pero se requieren muchos más recursos para implementarla.

Estas topologías de interconexión de *cores* se pueden combinar para formar estructuras jerárquicas con varios buses. El estándar no impone ninguna limitación para estas asociaciones, por lo que permite topologías muy flexibles.

14.2. Unidad procesadora TnP para Virtex y Wishbone

En el apartado 9.3 de esta tesis, dedicado al planteamiento teórico de `Tornado`, se definen las características generales que debe tener un `nano-procesador` embebido en `Cores Mixtos`, para ser compatible con el sistema `Tornado`.

En base a esos requerimientos se ha desarrollado un `nano-procesador TnP` optimizado para la tecnología Virtex de Xilinx. Se ha utilizado como base el procesador *soft* Picoblaze. Éste se ha presentado en el apartado 4.1 como un procesador candidato óptimo para ser integrado en *Cores Mixtos* dadas sus características. Está descrito en VHDL, y se ha diseñado para ser utilizado en FPGAs de grano fino de Xilinx. En la figura 14.3 se representa la unidad procesadora junto con el bloque de memoria RAM dedicada donde se almacena el programa del mismo. Sus características más representativas se resumen a continuación [179]:

- Procesador de 8 bits.
- 16 registros internos de propósito general.
- Una entrada de interrupción para eventos hardware externos.
- Operaciones de suma y resta aritméticas.
- Capacidad de direccionamiento de hasta 256 bytes en escritura y 256 bytes en lectura para la memoria de datos y 256 instrucciones en la memoria de programa.
- Arquitectura Von-Newman con bus dedicado para la memoria de programa.

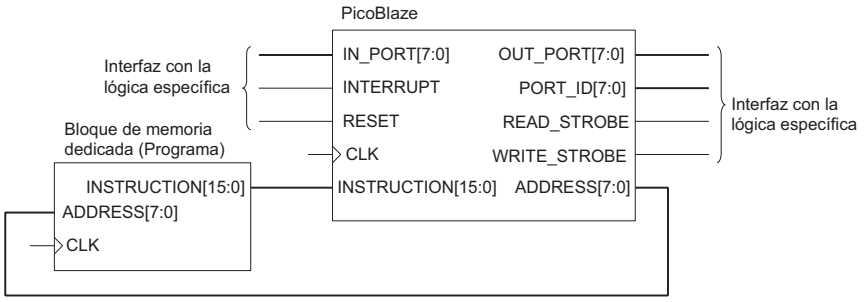


Figura 14.3.: Unidad procesadora *Picoblaze*.

- Profundidad del *Stack* de 15 niveles.

El código VHDL de este procesador se ha modificado en cuatro aspectos para convertirlo en el `nano-procesador TnP`:

- **Ampliación del juego de instrucciones** para incluir las relativas al control de la reconfiguración (habilitación y deshabilitación temporal de la misma). Los mnemónicos de estas instrucciones se establecen como:

  - `RENCONF ENABLE`: El `TnP` admite reconfiguraciones.
  - `RENCONF DISABLE`: Los intentos de aplicar reconfiguraciones sobre el `TnP` son rechazados hasta que no se habilite de nuevo la admisión de estas.

  En la tabla 25 se refleja el juego de instrucciones del `TnP` tras la ampliación del mismo. Los símbolos utilizados para representar los argumentos de las instrucciones representan:

  - `cc:` Valor constante de 8 bits.
  - `sX (ó sY):` Cualquiera de los 16 registros de propósito general internos al procesador. Como se puede observar en el juegos de instrucciones cualquiera de ellos puede funcionar como acumulador.
  - `aa:` Dirección del mapa de memoria de instrucciones. Todas las instrucciones son de anchura 16 bits, la misma que la del mapa de direcciones. La versión desarrollada para los dispositivos Spartan-II dispone un rango direccionable sin banqueo para 256 instrucciones.
  - `pp:` Dirección del mapa de memoria de datos. Los datos son de 8 bits de anchura y es el diseñador el que compone la zona de datos según la aplicación. Pueden conectarse otros bloques BRAM para disponer de espacio de memoria RAM, o bien otros periféricos descritos en VHDL a los que el procesador tenga acceso directo mediante su espacio de memoria.

En los `Cores Mixtos`, la sección de *hardware dedicado* y la interfaz con el bus *on-chip* están situadas en esta zona de datos[1] de tal forma que el `TnP` pueda interactuar con ellos de forma rápida, eficiente e independiente del resto del MCSoPC. Esta organización permite un funcionamiento efectivo de varios procesadores en paralelo.

Tabla 14.1.: Juego de instrucciones del `nano-procesador` *soft* `TnP`.

Grupo	Instrucción	Descripción
Lógicas		
	LOAD sX, cc	Carga de constante en registro
	AND sX, cc	AND entre constante y registro
	OR sX, cc	OR entre constante y registro
	XOR sX, cc	XOR entre constante y registro
	LOAD sX, sY	Movimiento de datos entre registros
	AND sX, sY	AND entre registros
	OR sX, sY	OR entre registros
	XOR sX, sY	XOR entre registros
Aritméticas		
	ADD sX, cc	Suma de constante y registro
	ADDCY sX, cc	Suma con llevada de constante y registro
	SUB sX, cc	Resta de constante y registro
	SUBCY sX, cc	Resta con llevada de constante y registro
	ADD sX, sY	Suma entre registros
	ADDCY sX, sY	Suma con llevada entre registros
	SUB sX, sY	Resta entre registros
	SUBCY sX, sY	Resta con llevada entre registros
Desplazamiento y rotación		
	SR0 sX	Despl. dcha. con inserción de 0
	SR1 sX	Despl. dcha. con inserción de 1
	SRX sX	Despl. dcha. manteniendo bit menos significativo
	SRA sX	Despl. dcha. usando bit de llevada
	RR sX	Rotación hacia dcha.
	SL0 sX	Despl. izda. con inserción de 0
	SL1 sX	Despl. izda. con inserción de 1
	SLX sX	Despl. izda. manteniendo bit menos significativo
	SLA sX	Despl. izda. manteniendo bit menos significativo
	RL sX	Rotación hacia izda.
Entrada y salida		
	INPUT sX, pp	Lectura bus datos posición pp
	INPUT sX, (sY)	Lectura bus datos posición (sY)
	OUTPUT sX, pp	Escritura del dato de sX en posición pp

[1] Por tanto, todos estos elementos deben estar direccionados en el espacio de memoria del propio `nano-procesador`. Este espacio es local para cada `Core Mixto`, no interfiriendo en ningún caso con el bus *on-chip* general del sistema.

	OUTPUT sX, (sY)	Escritura del dato de sX en posición (sY)
Control de programa		
	JUMP aa	Salto no condicional
	JUMP Z, aa	Salto si cero
	JUMP NZ, aa	Salto si no cero
	JUMP C, aa	Salto si llevada
	JUMP NC, aa	Salto si no llevada
	CALL aa	Llamada a subrutina no condicional
	CALL Z, aa	Llamada a subrutina si cero
	CALL NZ, aa	Llamada a subrutina no cero
	CALL C, aa	Llamada a subrutina si llevada
	CALL NC, aa	Llamada a subrutina si no llevada
	RETURN	Retorno no condicional
	RETURN Z	Retorno si cero
	RETURN NZ	Retorno si no cero
	RETURN C	Retorno si llevada
	RETURN NC	Retorno si no llevada
	INTERRUPT ENABLE	Habilitación de interrupción
	INTERRUPT DISABLE	Deshabilitación de interrupción
	RETURNI ENABLE	Retorno con interrupciones habilitadas
	RETURNI DISABLE	Retorno con interrupciones deshabilitadas
	RECONF ENABLE	Habilitación de reconfiguración
	RECONF DISABLE	Deshabilitación de reconfiguración

- **Inclusión de la interfaz** `TIF (S)`, para conectar el juego de señales (`STB_RECONF`, `RECONF_ACK`, `PCR` y `SPR`) definido en `Tornado`.
- **Modificación global del diseño del procesador con el fin de poder congelar su estado** y parar su funcionamiento cuando se está aplicando la reconfiguración.

  A partir del momento en el que el `nano-procesador` permite al controlador de reconfiguración la aplicación de la reconfiguración parcial activando la señal `RECONF_ACK`, se bloquea la actividad de éste hasta que la señal `STB_RECONF` sea desactivada.
- **Modificación del contador de programa y del puntero de pila** para poder ser inicializados tras la aceptación de la reconfiguración en función del valor que el controlador de reconfiguración haya definido para cada `TnP-Core`.

La descripción de este `TnP` se ha realizado completamente en código VHDL, lo que facilita su portabilidad a distintos diseños y a diferentes dispositivos. El aspecto mencionado de portabilidad entre dispositivos está restringido en este caso a varias familias de dispositivos. Esto es debido a que en el código VHDL se realizan instanciaciones de elementos específicos de la tecnología de Xilinx. En cierta manera, se sacrifica la portabilidad, pero se mejora el resultado del sintetizador de VHDL guiándolo a estructuras optimizadas para los dispositivos en cuestión.

Aún así, los resultados de la implementación de este procesador, tanto en lo referente a los recursos en área requeridos como a la velocidad máxima de funcionamiento obtenida, pueden

diferir para distintos diseños. Los factores básicos por los que se producen estas diferencias son: el dispositivo utilizado, el porcentaje del mismo ocupado por todo el diseño, y las opciones de síntesis, emplazamiento y rutado establecidas en la herramienta de desarrollo.

En la tabla 14.2 se muestran los resultados de implementación del procesador `TnP` para una FPGA Spartan-II XC2S150. En la segunda columna se indica la cantidad de recursos lógicos requeridos para este `nano-procesador`, mientras que en la tercera columna se indican los necesarios para la implementación del `nano-procesador` original (no compatible con `Tornado`). Para cada tipo de recurso, además de la cantidad requerida, se indica el porcentaje del mismo utilizado en una FPGA Spartan-II XC2S150. Ambos resultados se han obtenido empleando las mismas opciones de síntesis, emplazamiento y rutado. Estos datos únicamente representan la implementación de una única unidad procesadora, sin software asociado (requiere un BRAM adicional) ni hardware específico direccionado en la memoria de datos, ya que éste depende de la aplicación.

Tabla 14.2.: Resultados de implementación de un `nano-procesador TnP`.

Tipo de Recurso	**Cantidad necesaria para el** `TnP`	**Cantidad necesaria para el** `nano-procesador` **original**
LUTs de 4 entradas	139 (4 %)	131 (3 %)
Biestables	65 (1 %)	61 (1 %)
Slices Virtex	82 (4 %)	74 (4 %)
Bloques de RAM 4K dedicados	0 (0 %)	0 (0 %)
Vel. máx. de funcionamiento	127 MHz	128 MHz

Aunque en apartados posteriores se presentan datos de implementación más realistas al integrarse los `TnPs` en *cores* y en sistemas complejos, estos resultados permiten cuantificar los parámetros para el modelado de la infraestructura `Tornado` definidos en el apartado 10.2 y referidos a la unidad procesadora `TnP`.

Sustituyendo los datos de implementación en la ecuación 14.1 se obtiene el término $\mathbf{TnP_{over\_est}}$, el cual engloba el sobrecoste estimado para cada procesador `TnP` incluído en el diseño (tabla 14.3).

$$\begin{aligned}\mathbf{TnP_{over\_est}} = (\mathbf{TnP_{mod\_lut}} - \mathbf{TnP_{orig\_lut}}) + (\mathbf{TnP_{mod\_ff}} - \mathbf{TnP_{orig\_ff}}) = \\ = (139 - 131)\ lut + (65 - 61)\ ff = 8\ lut + 4\ ff\end{aligned} \tag{14.1}$$

Tabla 14.3.: Parámetros de modelado del TnP.

Parámetro	Descripción	Valor
$\mathbf{TnP_{orig\_lut}}$	Número de LUTs del nano-procesador original	131 lut
$\mathbf{TnP_{orig\_ff}}$	Número de biestables del nano-procesador original	61 ff
$\mathbf{TnP_{mod\_lut}}$	Número de LUTs del nano-procesador modificado TnP	139 lut
$\mathbf{TnP_{mod\_ff}}$	Número de biestables del nano-procesador modificado TnP	65 ff
$\mathbf{TnP_{over\_est}}$	Sobrecoste estimado en LUTs y biestables por TnP	8 lut + 4 ff

14.3. Controlador de reconfiguración TBC para Virtex y Wishbone

Las peticiones de reconfiguración por parte de los *cores* maestros se realizan escribiendo una palabra de configuración (CRW) al controlador de reconfiguración TBC. En el apartado 9.2 se han definido las características generales y la arquitectura de este controlador propuesto en la infraestructura Tornado.

Para las distintas plataformas de validación se ha desarrollado un único controlador TBC específico para el sistema de carga de configuración paralelo de Xilinx y compatible con la especificación Wishbone. En la figura 14.4 se representa su arquitectura interna. En este caso, el bloque de memoria RAM dedicada se identifica como 4K BRAM, término propio de Xilinx. Las distintas interfaces de esta implementación quedan definidas de la siguiente forma:

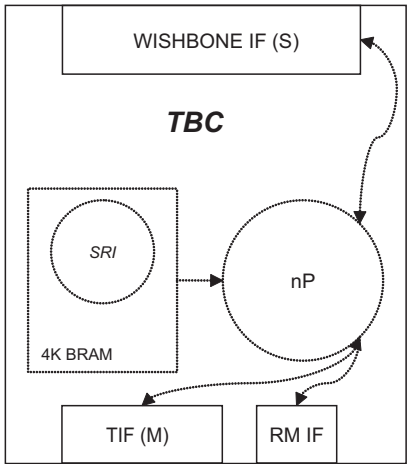


Figura 14.4.: Arquitectura del TBC para dispositivos Virtex y compatible Wishbone.

- **Interfaz del bus *on-chip*:** Interfaz esclava compatible con la especificación Wishbone para un bus de datos de 16 bits. Tiene las señales obligatorias definidas en la norma: rst_i, clk_i, ack_o, adr_i, dat_i_o, stb_i y we_i.
- **Interfaz Tornado (TIF (M)):** Mediante esta interfaz el TBC gestiona el *handshake* de

control de la auto-reconfiguración con los T-Cores. Es una interfaz maestra ya que inicia el ciclo de control. Las señales incluidas en ésta son:

- STB_RECONF(15:0). Al tratarse de una interfaz maestra, estas son señales de salida. Son un total de 16, lo que admite un control de hasta un máximo de 16 T-Cores.
- SPR. Señal de salida para controlar el puntero de pila.
- PCR. Señal de salida para controlar el contador de programa.
- RECONF_ACK. Señal de entrada compartida para todos los T-Cores.
- LOCK(15:0). Estas señales permiten que el TBC congele el funcionamiento de algún T-Core aunque no sea el que específicamente este soportando la reconfiguración. Pese a que esta señal no está incluida en la especificación de Tornado, se ha añadido a efectos de facilitar experimentación. La restricción existente en los dispositivos de Xilinx que obliga a realizar la reconfiguración completa por columnas, obliga a extender el control durante el proceso de cambio de contexto a todos los T-Cores situados en una misma columna.

- **Interfaz Reconfiguration Media (RM IF):** La interfaz RM IF está diseñada para controlar el puerto de configuración paralelo SelectMap [80] incluido en las FPGAs de Xilinx.

  Este sistema de carga paralelo es el utilizado en el prototipo desarrollado para implementar las plataformas de evaluación presentadas en esta tesis y para verificar otros módulos como el FFR16 presentado anteriormente. En la figura 14.5 se muestra uno de los prototipos. La FPGA incluida es una Spartan-II X2S150-5PQ150, siendo éste el elemento de computación principal. El resto de componentes son circuitos auxiliares. Éstos dotan al prototipo de múltiples canales de comunicación, un sistema de almacenamiento masivo de datos y un flexible sub-sistema de carga y almacenamiento de *bitstreams* específicamente diseñado para experimentar con reconfiguración parcial dinámica. Los principales elementos de este prototipo son:

  - **Canales de comunicación:** El prototipo está dotado de las siguientes interfaces físicas:
    - Un canal de comunicación serie de alta velocidad con interfaces RS-232 y RS-422.
    - Un canal de comunicación paralelo compatible *Enhanced Parallel Port* (EPP).
    - Dos canales IDE. Al estar gestionados por la FPGA, pueden utilizarse como esclavos o como maestros.
    - Un canal USB gestionado por un integrado específico para este tipo de comunicación.
    - Un canal de comunicación serie de baja velocidad para operaciones de *telecarga* y depuración.
    - Una interfaz especialmente protegida para control de sistemas de potencia (motores, cabezales térmicos, etc.).

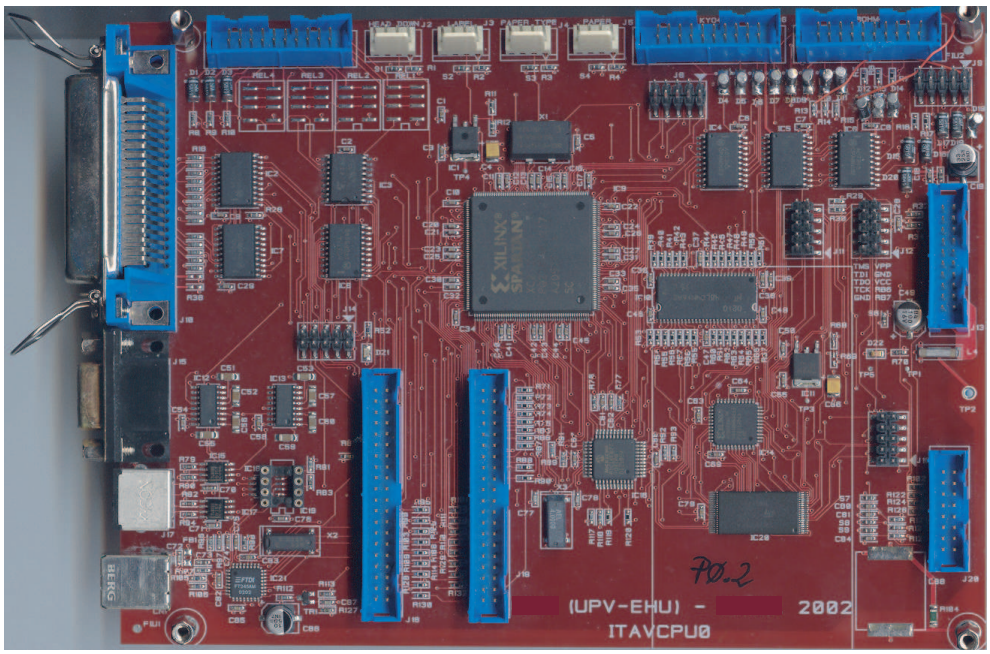


Figura 14.5.: Prototipo desarrollado para la experimentación con reconfiguración parcial dinámica.

- **Interfaz VGA:** Se ha incluido una interfaz VGA simple. Las señales se deben generar en la FPGA y son adaptadas al monitor mediante un conjunto de resistencias.
- **Memoria RAM de 64 Mbits:** El prototipo incorpora una memoria RAM dinámica síncrona, cuyo control se realiza de forma íntegra en la FPGA y cuyo propósito es el de almacenar los datos procesados en la misma.
- **Subsistema de carga y almacenamiento:** En la figura 14.6 se muestra el subsistema de carga de *bitstreams* totales y parciales que dispone el prototipo utilizado para las plataformas de evaluación. Los *bitstreams* se almacenan en una memoria FLASH paralelo de 8 bits. Las direcciones para la lectura de los datos en FLASH y su escritura en la FPGA se generan mediante una CPLD auxiliar. Además, esta CPLD dispone de otro modo de funcionamiento que actúa en el proceso de grabación de los ficheros de configuración en la FLASH. Estas configuraciones se reciben desde un ordenador externo a través del canal de comunicación serie de baja velocidad gestionado por un microcontrolador de 8 bits incluido en la placa.

  La interfaz `RM IF` dispone de un conjunto de señales que controlan la CPLD para indicar a ésta qué *bitstream* debe cargar en cada momento, según la reconfiguración que deba aplicar. El controlador `TBC` conoce por tanto, la ubicación en la FLASH de cada *bitstream* requerido por cada palabra de petición de reconfiguración `CRW`. En concreto, las señales incluidas en la interfaz `RM IF` son:

  - `PIN_MODE_CPLD`. Selección del modo de funcionamiento de la CPLD.

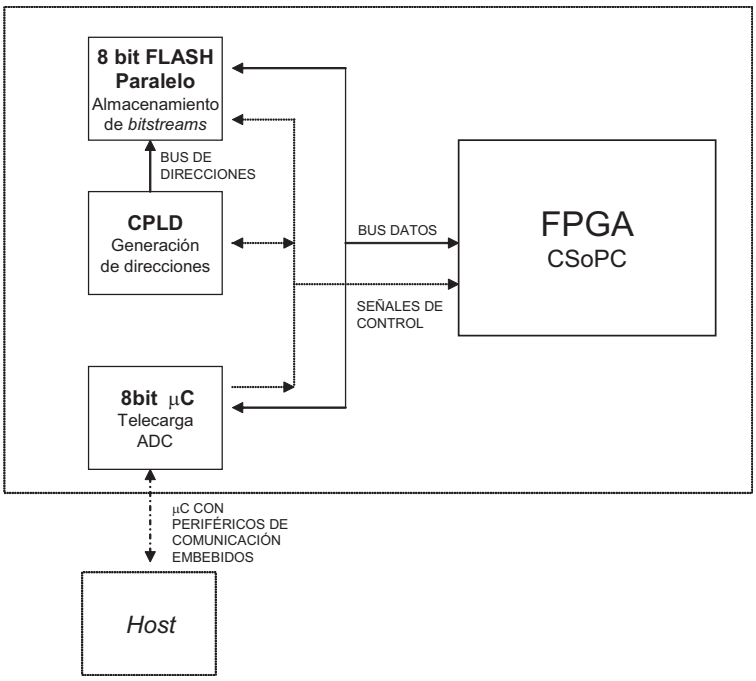


Figura 14.6.: Subsistema de carga de *bitstreams* parciales y totales integrado en el prototipo.

- `PIN_NCSCPLD`. Habilitación de la CPLD.
- `PIN_NCSFLASH`. Habilitación de la memoria FLASH de almacenamiento de *bitstreams*.
- `PIN_NRD`. Señal de lectura de la memoria FLASH.
- `PIN_NWR`. Señal de escritura de la memoria FLASH.

El `TBC` se ha descrito íntegramente en VHDL. Se ha diseñado el mismo con una arquitectura de `Core Mixto`, tal y como se establece en el planteamiento general de su estructura interna. El procesador integrado es el `TnP`, presentado en el apartado anterior, aunque en este caso la interfaz `TIF (S)` del procesador no se utiliza, dejándose en manos del sintetizador la tarea de eliminar su lógica asociada en el proceso de optimización.

El código para el `nano-procesador` del `TBC` se genera específicamente para cada aplicación de forma automática mediante los *scripts* de automatización. Éstos utilizan la información extraída por el programa ensamblador específico de `Tornado` a partir de los distintos códigos fuente de los `TnP-Cores`. De igual forma, el código final VHDL del `TBC` se obtiene automáticamente tal que puede ser integrado en el flujo de diseño de Xilinx.

La tabla 14.4 muestra los resultados de una implementación aislada de un controlador `TBC`. Como se puede observar en estos datos, la filosofía aplicada al diseño de este *core* (arquitectura de `Core Mixto`, *metacomputación distribuida*, etc.) permite obtener un controlador realmente pequeño, sencillo y rápido. Estos datos permiten completar el modelado para la estimación de recursos necesarios por la infraestructura `Tornado`. Para ello, se seleccionan los parámetros que modelan el coste en recursos lógicos del `TBC` (tabla 14.5). Aplicando la ecuación 14.2 se agrupan estos parámetros en un único término denominado $\mathbf{CTR_{over\_est}}$.

Tabla 14.4.: Resultados de implementación del controlador de reconfiguración TBC.

Tipo de Recurso	Cantidad necesaria	Porcentaje de recursos[2]
Puertas Lógicas Equivalentes	20.668	-
Slices Virtex	119	6 %
LUTs de 4 entradas	189	5 %
Biestables	1	2 %
Bloques de RAM 4K dedicados	1	8 %
Vel. máx. de funcionamiento	74 MHz	-

Tabla 14.5.: Parámetros de modelado del TBC.

Parámetro	Descripción	Valor
$\mathbf{CTR_{lut}}$	Número de LUTs utilizadas para la implementación del TBC	189
$\mathbf{CTR_{ff}}$	Número de biestables utilizados para la implementación del TBC	119
$\mathbf{CTR_{4Kram}}$	Número de bloques de memoria dedicada de 4 Kbits	1
$\mathbf{CTR_{over\_est}}$	Sobrecoste estimado de recursos lógicos al agregar el controlador TBC	189 lut + 119 ff + 1 bram

$$\mathbf{CTR_{over\_est}} = \mathbf{CTR_{lut}} + \mathbf{CTR_{ff}} + \mathbf{CTR_{4Kram}} = 189\ lut + 119\ ff + 1\ bram \tag{14.2}$$

Una vez conocido este término, ya se dipone de los módulos TnP y TBC caracterizados mediante sus parámetros de modelado para la infraestructura Tornado en esta tecnología. En la ecuación 14.3 se sustituyen éstos para obtener el valor del parámetro global $\mathbf{Area\_over_{Tornado\_est}}$ ("Sobrecoste estimado de recursos lógicos -área de silicio- por la adición de la infraestructura Tornado" -*Estimate Tornado Area Overhead*-), donde n y j representan respectivamente el número de TnP-Cores y de TBMs utilizadas en el diseño, y x_i el número de TnP para cada i TnP-Core.

$$\begin{aligned}\mathbf{Area\_over_{Tornado\_est}} &= \mathbf{CTR_{over\_teor}} + \mathbf{TnP_{over\_teor}} \cdot n + \mathbf{MACRO_{over}} \cdot j = \\ &= (189\ lut + 119\ ff + 1\ bram) + \sum_{i=1}^{n}(8\ lut + 4\ ff) \cdot x_i + \mathbf{MACRO_{over}} \cdot j \end{aligned} \tag{14.3}$$

[2]Esta cifra representa para cada tipo de recurso lógico el porcentaje de utilización de los disponibles en una FPGA Spartan-II X2S150-6.

14.4. Módulos parametrizables auxiliares

Para la construcción de las plataformas utilizadas para la validación de `Tornado`, además de los diferentes *cores* que las componen, son necesarios elementos auxiliares para completar el sistema digital.

Los elementos auxiliares más relevantes son el decodificador de direcciones y el arbitrador de bus. Ambos se han diseñado incluyendo elementos parametrizables (número de maestros, rango de direcciones, etc.). De esta forma son fácilmente reutilizables para distintos diseños. Con este fin, se han utilizado descripciones basadas en *genéricos* VHDL. Además, estos módulos son compatibles con la especificación Wishbone. Al utilizar las señales de esta norma se facilita aún más la integración en diferentes sistemas.

El decodificador de direcciones es un simple diseño combinacional que en presencia de una señal de *strobe* (`stb_i`) activada por una interfaz de bus maestra, genera una señal de *strobe* específica para un *core* con interfaz de bus esclava. El *strobe* generado es función de la dirección indicada en el bus de direcciones por el maestro.

El arbitrador de bus desarrollado dispone de una arquitectura simple para realizar un arbitraje de tipo *round-robin* (con igual prioridad) entre un número genérico de *cores* maestros conectados a un bus compartido Wishbone. El funcionamiento de este arbitrador se resume en la siguiente secuencia:

1. El valor decodificado de un contador binario se compara bit a bit con las entradas de petición de acceso al bus de los distintos *cores* maestros conectados al sistema (señales `cyc_i` de la interfaz Wishbone).

2. El valor decodificado es el asignado al maestro autorizado para poder controlar el bus en esa secuencia de la rueda.

3. En el caso de que se produzca acierto entre ese valor y la entrada `cyc_i` activada para el maestro autorizado, se da permiso a este para que tome el control del bus y realice las transferencias necesarias.

4. Únicamente cuando finalicen éstas se volverá a incrementar el valor del contador continuando el ciclo de cesión del testigo a los siguientes maestros.

15. Herramientas software

Se presentan a continuación las herramientas software utilizadas. Las propias del fabricante se agrupan en el sistema de desarrollo *Integrated Software Environment* (ISE). Las específicamente desarrolladas en esta tesis, se presentan en el apartado dedicado al ensamblador para el `nano-procesador TnP`.

15.1. Sistema de desarrollo ISE

Cada fabricante de dispositivos reconfigurables pone a disposición del diseñador un entorno de desarrollo específico. Habitualmente son paquetes integrados software a los cuales se les puede incorporar herramientas desarrolladas por otras compañías, tales como sintetizadores de lenguajes HDL, simuladores, etc.

Las plataformas y diseños presentados en esta tesis se han desarrollado utilizando el entorno de diseño integrado ISE de Xilinx junto con las herramientas específicas definidas en el *framework* `Tornado`. En la figura 15.1 se representa el diagrama de flujo de diseño simplificado del entorno de desarrollo ISE, similar a otros sistemas de desarrollo.

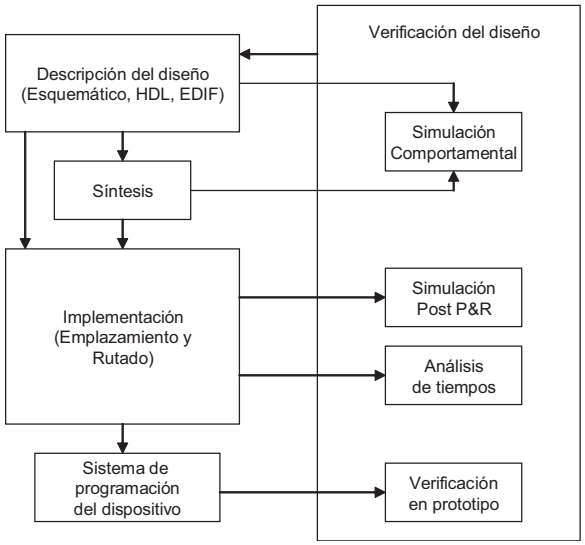


Figura 15.1.: Flujo de diseño ISE.

Se parte de la descripción del diseño, realizada generalmente utilizando descripciones en HDL de los distintos módulos. El sintetizador elegido (puede ser el suministrado por Xilinx u otro) convierte las descripciones HDL a una lista de recursos lógicos básicos interconectados (*netlist).*

Mediante un simulador HDL, que en el entorno ISE por defecto es el Modelsim, se pueden realizar simulaciones funcionales previas a las fases de emplazamiento y rutado del diseño en el dispositivo, e incluso antes de la síntesis.

Las fases de emplazamiento y rutado se ejecutan mediante herramientas específicas del fabricante y permiten obtener ficheros con información detallada de los recursos del dispositivo asignados, interconexión y retardos estimados para las distintas señales. Con esta información, en el caso de que el diseño lo requiera, se pueden realizar simulaciones detalladas utilizando de nuevo el simulador HDL pero, en este caso, utilizando modelos más complejos.

A partir de los ficheros que integran la información de emplazamiento y rutado, otra herramienta específica del fabricante genera el *bitstream* de configuración del dispositivo. Éste se puede descargar en la FPGA y verificarlo mediante sistemas de depuración (analizador embebido) también integrados en el entorno de desarrollo ISE.

15.2. Ensamblador para procesadores TnP

15.2.1. Descripción general

La unidad procesadora `TnP` diseñada para la integración en `Cores Mixtos` compatibles con el sistema `Tornado` tiene unas características especiales derivadas de su capacidad multi-contexto. Esto es, un código por cada contexto de cada `TnP` que compone el sistema. Por ello, ha sido necesario desarrollar un programa ensamblador (`Tornado Reconfigware`[1] `Assembler -TRA-`) que facilite al diseñador la gestión de los diversos códigos fuente del sistema y definir el comportamiento de cada `TnP` tras su reconfiguración.

Este programa ensamblador, además de las funciones habituales de una herramienta de este tipo, como es la de conversión a código máquina de todas las instrucciones, dispone de las siguientes particularidades:

- **Soporte para las instrucciones de habilitación/deshabilitación temporal de la reconfiguración** para cada `Core Mixto` compatible `Tornado` (`TnP-Core`). Además de estas instrucciones específicas, el ensamblador `TRA` reconoce el juego de instrucciones del `TnP`.

- **Ficheros de salida del código ensamblado en VHDL.** El código ejecutable de los `Cores Mixtos` se sitúa en los bloques de memoria RAM dedicada de las FPGAs. Puesto que se emplea el mecanismo de configuración de las FPGAs (mediante *bitstreams*) para la carga del contenido de estas memorias, es necesario integrar los códigos máquina de los distintos `TnPs` en el flujo de diseño HDL del fabricante. En este caso, en el flujo de diseño del entorno ISE.

  Esta integración se realiza escribiendo el contenido ensamblado de los programas en ficheros VHDL que definen la inicialización de los bloques de memoria dedicados. Estos ficheros son interpretados en el entorno de desarrollo ISE. Además, este entorno incluye también las herramientas para generar los *bitstreams* parciales para la aplicación de la reconfiguración parcial dinámica. En éstos se integran los códigos de los programas

[1] El término *reconfigware* se aplica a la descripción de los componentes reconfigurables dinámicamente. Se describen más detalles sobre sobre este término en las publicaciones de J.M. Cardoso [222].

para los distintos contextos además de los cambios locales de hardware realizados en el circuito.

- **Integración de todos los contextos software para cada `TnP` en un único fichero**. Esta agrupación facilita el diseño de sistemas multi-procesador con reconfiguración dinámica, ya que se dispone de una visión global de todos los códigos fuente para cada `TnP`. De esta forma, se puede escribir y modificar el código para cada procesador de una manera más sencilla, reduciéndose de manera considerable el tiempo de diseño y verificación.
- **Directivas para la gestión de múltiples códigos fuente en un mismo fichero**. La integración de códigos en un mismo fichero requiere ciertas directivas para indicar al programa ensamblador qué sección de código corresponde a cada contexto. Estas directivas se definen como:

  - `START j`: Indica que comienza una zona de código fuente correspondiente al contexto `j`.
  - `STOP j`: Indica que finaliza una zona de código fuente correspondiente al contexto `j`.

  El propio programa ensamblador se encarga de agrupar las distintas secciones de código definidas para un mismo contexto a lo largo de cada fichero de código . De esta forma, se facilita la escritura de código multi-contexto al poder situar códigos fuente similares (por ejemplo, tablas para generación de ondas) de forma conjunta aunque se apliquen en contextos diferentes.
- **Directivas para la gestión del estado de cada `TnP-Core` en el proceso de reconfiguración**. Es necesario definir para cada `TnP-Core` la gestión del contador de programa y del puntero de pila tras la aplicación de la reconfiguración parcial, pudiéndose determinar la inicialización o no inicialización de los mismos. El control se realiza mediante las señales `SPR` y `PCR` para el puntero de pila y el contador de programa respectivamente, las cuales son fijadas por el controlador de reconfiguración `TBC`.

  Las diferentes posibilidades de gestión de estos registros se plasman para cada `TnP` en su fichero correspondiente de código fuente mediante unas directivas específicas. Se debe fijar su valor una única vez por cada fichero. Estas directivas son:

  - `SPR [YES/NO]`: Indica si se debe realizar (`SPR YES`) o no (`SPR NO`) una inicialización del puntero de pila tras la aplicación de la reconfiguración parcial.
  - `PCR [YES/NO]`: Indica si se debe realizar (`PCR YES`) o no (`PCR NO`) una inicialización del contador de programa tras la aplicación de la reconfiguración parcial.
- **Generación automática del controlador de reconfiguración**. El ensamblador `TRA`, junto con los *scripts* de automatización genera de manera automática el código VHDL del controlador de reconfiguración para cada diseño concreto. Las directivas indicadas en el punto anterior determinan el comportamiento del controlador de reconfiguración cuando tenga que aplicar una reconfiguración parcial dinámica sobre cada `TnP-Core`.

15.2.2. Flujo de ficheros

El proceso de ensamblado parte de un único archivo de código fuente por cada `TnP-Core` embebido en el MCSoPC, escrito en lenguaje ensamblador, en el que se agrupan todos los programas de los diferentes contextos. La salida del proceso de ensamblado se resume en un conjunto de ficheros VHDL donde se describe la inicialización de los bloques de memoria RAM dedicados para cada contexto software.

Aunque estos ficheros de salida y el de código fuente de entrada son los más relevantes, hay una serie de archivos auxiliares que permiten al diseñador analizar el código y definen los parámetros necesarios para que los *scripts* de automatización generen el código VHDL del controlador de reconfiguración `TBC`.

En la figura 15.2 se muestran tanto los archivos de entrada como los generados por el ensamblador. En esta representación los archivos están ordenados por grupos en función de la utilidad de los mismos. Se resume a continuación su utilidad:

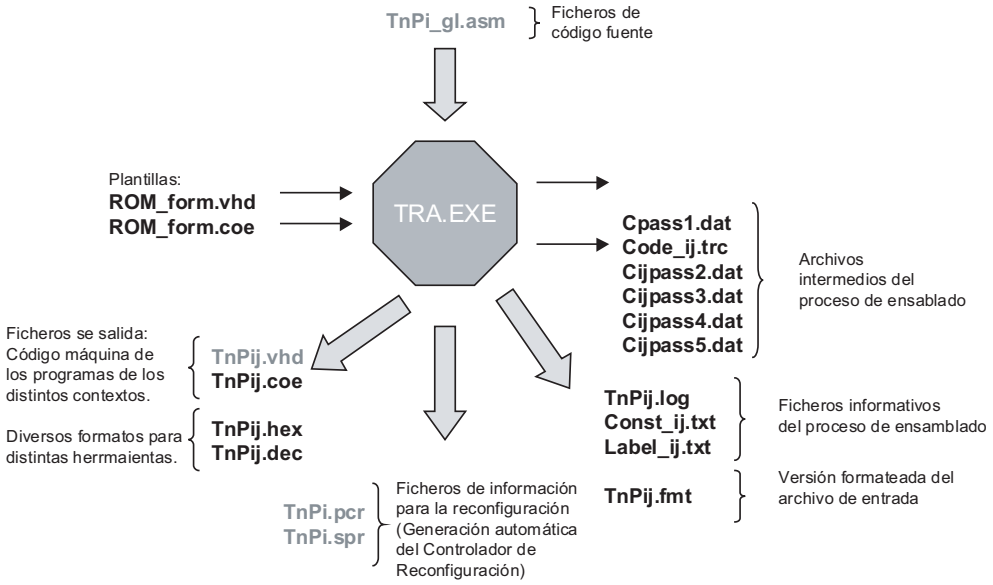


Figura 15.2.: Diagrama de flujo de ficheros del ensamblador `TRA` para un `TnP`.

- **Archivos de entrada:**

  - Ficheros de código fuente (`TnPi_gl.asm`)[2]. En este fichero se incluyen todos los contextos software aplicables al `nano-procesador` del `TnP-Core` número `i`.

    El formato es el de fichero de texto, incluyéndose las instrucciones con sus respectivos mnemónicos y argumentos además de las directivas necesarias de control y comentarios.

[2]La `i` se sustituye en cada caso por el número asignado de `T-Core`. La numeración se realiza teniendo en cuenta todos los `T-Cores` del diseño, no únicamente los `TnP-Cores`. Tendrán contenido de código fuente los `TnPi_gl.asm` que estén asignados a un `TnP-Core`, el resto sólo podrá incluir directivas de control.

- Plantilla para el código VHDL (`ROM_form.vhd`). Este archivo de texto se utiliza como referencia para componer cada uno de los archivos `.vhd` que genera el ensamblador. A partir de esta plantilla se obtienen los ficheros VHDL con los atributos que contienen los datos de inicialización de los bloques de memoria completados. `ROM_form.coe` es otra plantilla para generar los datos de inicialización en un formato que interpretan las herramientas específicas de Xilinx, como el *CoreGenerator* entre otras.

- **Archivos de salida:**

  - Ficheros de salida VHDL con los atributos de inicialización de los bloques de memoria para los diferentes contextos y `nano-procesadores` (`TnPij.vhd`[3]). La información codificada en estos atributos son los programas que tienen que ejecutar los procesadores `TnP` en lenguaje máquina.

    También se obtienen un conjunto de ficheros con la misma información pero con un formato específico para ser interpretados por la herramienta *CoreGenerator* (`TnPij.coe`).
  - Ficheros para la configuración automática del controlador de reconfiguración (`TnPi.pcr` y `TnPi.spr`). Se generan a partir de las directivas incluidas en el código fuente. Éstas especifican el modo de gestión de los registros de contador de programa y de puntero de pila. Para cada `T-Core` se dispone de una pareja de estos ficheros, los cuales son utilizados por los *scripts* de inicialización para generar automáticamente un controlador de reconfiguración `TBC` específico.

15.3. Scripts de automatización

Mediante los *scripts* de automatización y el programa ensamblador `TRA`, se obtienen todos los ficheros VHDL con el código máquina para los `nano-procesadores` del sistema. Además, se genera el código VHDL del programa del controlador de reconfiguración `TBC` configurado específicamente para cada diseño MCSoPC.

La figura 15.3 resume la operación conjunta del `TRA` y de los *scripts* de automatización para obtener los ficheros VHDL indicados. La secuencia seguida en este diagrama de flujo sigue los pasos establecidos en el diagrama de flujo general para reconfiguración *intra-task* presentado en el apartado 11.2. Mediante *scripts* adicionales se integran los ficheros de salida `.vhd` en el flujo de diseño de ISE y se automatiza la generación de los *bitstreams* parciales.

[3] `i` es el número de `T-Core` y `j` el número de contexto para ese *core*. En la versión del `TRA` desarrollada únicamente se soportan `TnP-Cores` con un `TnP`.

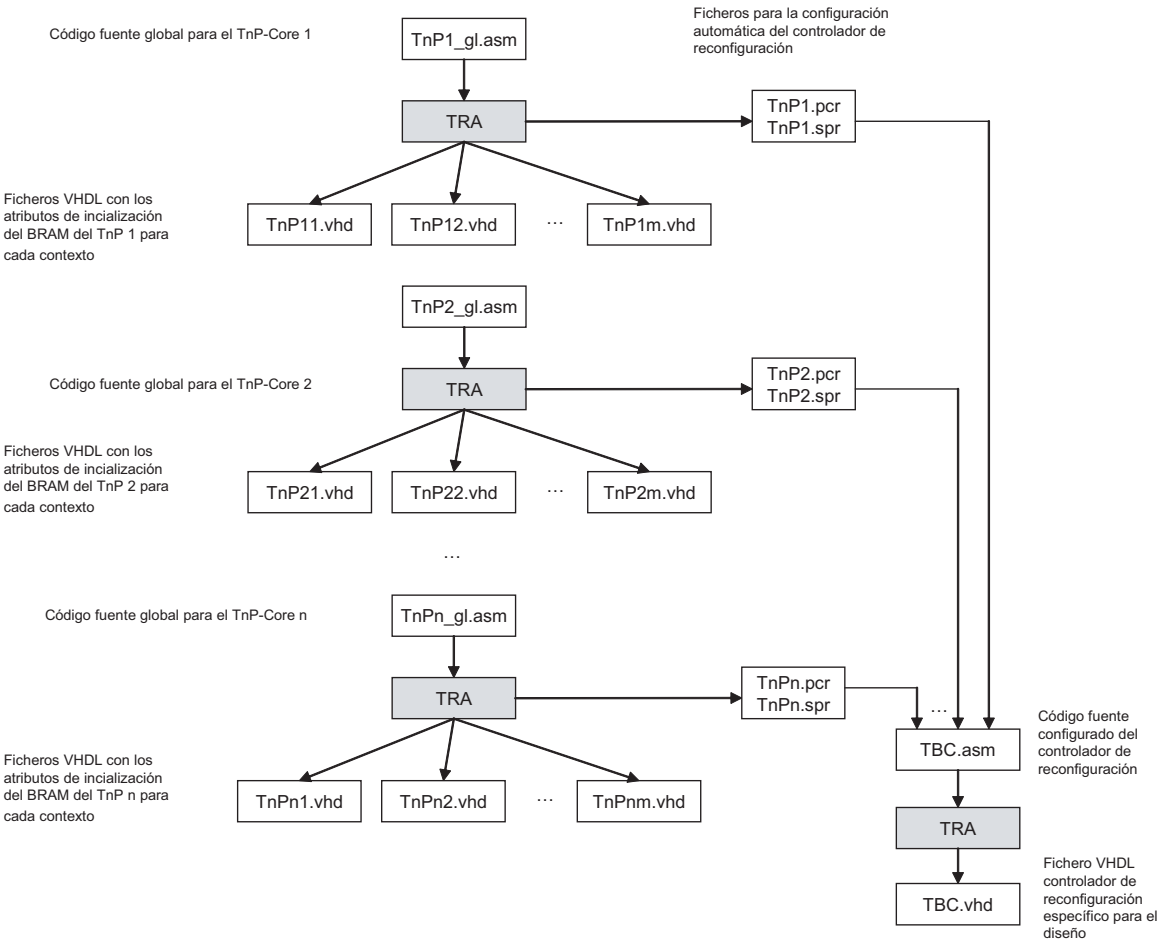


Figura 15.3.: Generación automática de los ficheros VHDL para la inicialización de los bloques de RAM y del controlador de reconfiguración.

16. Plataforma de verificación básica: Video-Multimaster

16.1. Descripción general

Uno de los objetivos definidos para el sistema Tornado es el de dar soporte a la reconfiguración *intra-task* de los Cores Mixtos. Este modo de reconfiguración sobre este tipo de *cores* incluye la posibilidad del cambio del contexto software de los nano-procesadores.

Mediante la plataforma Video-Multimaster se evalúa la utilización de Tornado para realizar este tipo de cambios de contexto sobre los dispositivos de la familia Virtex de Xilinx. Es un diseño orientado a validar y experimentar este sistema de control y su infraestructura asociada, habiéndose definido para este fin una arquitectura sencilla y escalable. El diseño se ha realizado en VHDL, empleando *genéricos* para poder ensayar plataformas con distintas composiciones de *cores*.

La plataforma Video-Multimaster es un sistema basado en *cores*. Dispone tanto de T-Cores como de IP-Cores. Los T-Cores tienen arquitectura de Core Mixto, perteneciendo por tanto, al grupo de los TnP-Cores. En concreto los desarrollados para esta plataforma se denominan TnP-Master Video.

Todos los *cores* se conectan mediante la especificación estándar Wishbone. La topología de bus empleada es la de bus compartido. Los TnP-Master Video son todos maestros del bus Wishbone en el sistema, conformando una estructura multi-maestro sobre el bus compartido que deberá ser gestionada por un arbitrador que controle el acceso al mismo. Los TnP-Master Video pueden escribir sobre un *core* periférico controlador de vídeo. Este periférico, denominado WB-VGA, sintetiza de forma directa las señales de vídeo para un monitor VGA y escribe en éste caracteres de texto almacenados en una memoria interna del propio *core*.

16.2. Diagrama de bloques

En la figura 16.1 se muestra el diagrama de bloques de una implementación de la plataforma con dos TnP-Master Video. Completan la plataforma el *core* periférico de vídeo WB-VGA, el controlador de reconfiguración TBC y el arbitrador de bus.

Cada uno de los TnP-Master Video tiene asociada una parte de la pantalla del monitor VGA para visualizar información. Esta información depende de los programas que se estén ejecutando en los TnPs. Éstos, mediante las instrucciones de acceso al bus *on-chip*, escriben en el periférico WB-VGA la información relativa al contexto en curso.

Las peticiones de reconfiguración parten de los mismos *cores* que van a ser reconfigurados. Éstos, utilizando las instrucciones de acceso al bus, escriben las palabras de petición de reconfiguración CRW en el controlador de reconfiguración TBC. En la figura 16.1 se han representado mediante flechas unidireccionales las posibles rutas de las CRW.

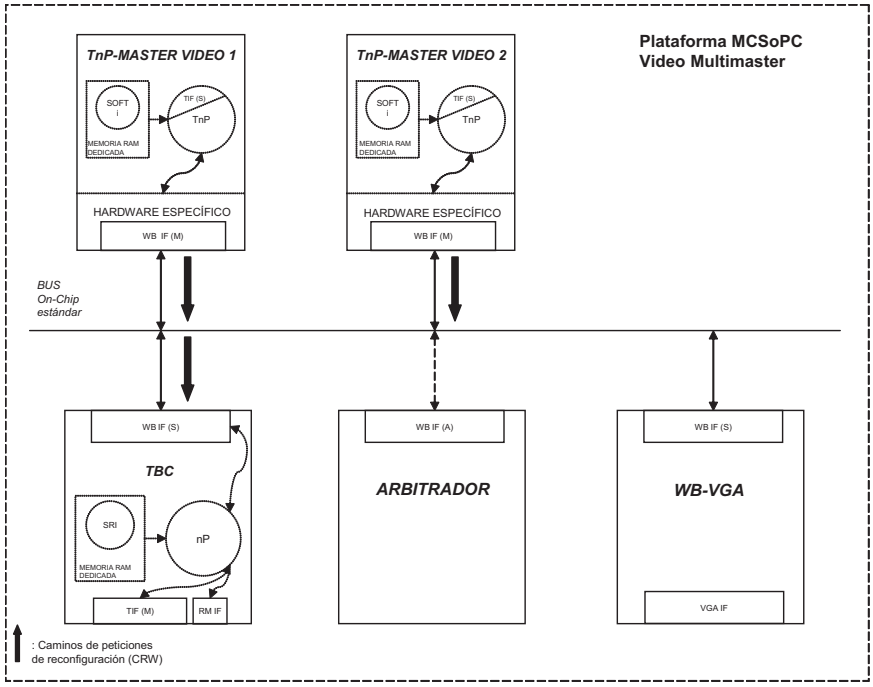


Figura 16.1.: Diagrama de bloques de la plataforma de verificación *Video-Multimaster*.

Cada `TnP-Master Video` muestra su número de contexto activo en la porción de pantalla asignada para él. De este modo se tienen *cores* maestro trabajando en paralelo y representando en pantalla información sobre su estado interno, información que es fácilmente interpretable. La reconfiguración del contexto software del *core* implica una modificación del bloque de memoria RAM que se utiliza para almacenar el código de programa. Por tanto, una vez que la reconfiguración se realiza satisfactoriamente, el `TnP-Master Video` ve modificado su programa, lo que implica el cambio del mensaje a visualizar en pantalla. De esta forma, el usuario puede verificar el cambio de contexto mediante observación de la información reflejada en la pantalla del monitor.

16.3. Módulos IP

16.3.1. Módulos maestro: TnP-Master Video

El `TnP-Master Video` es un `TnP-Core`, es decir, un `Core Mixto` compatible con el sistema `Tornado` y con un `TnP` integrado. En la plataforma `Video-Multimaster` se realiza una instanciación del mismo con un software ligeramente diferente para cada uno de ellos. Los cambios se centran en los mensajes que se muestran en el monitor VGA y en la frecuencia de escritura de las `CRW`. En la figura 16.2 se representa la arquitectura del *core*. Está compuesto por el `TnP`, un bloque de memoria RAM dedicada para almacenar el programa y dos interfaces. Además de la lógica necesaria para la interfaz `TIF (S)` de forma que disponga de compatibilidad con `Tornado`, el `TnP-Master Video` añade una lógica adicional para la conexión con el bus *on-chip*: la interfaz maestra Wishbone.

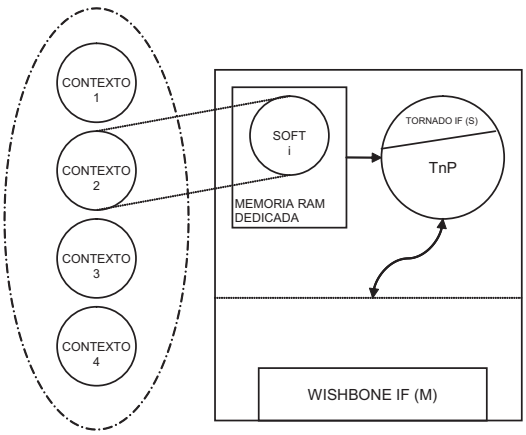


Figura 16.2.: `TnP-Master Video` *core*.

En esta figura también se representan los cuatro contextos que se pueden reconfigurar sobre la memoria de programa del `nano-procesador`. El programa deberá indicar al `TBC` cuál de esos contextos desea cargar pasando a la ejecución del nuevo contexto tras aplicarse la reconfiguración.

El funcionamiento del `TnP-Master Video` está gobernado por el programa almacenado en el bloque de memoria. Este software muestra, a través del `WB-VGA`, una serie de mensajes por la pantalla que identifican el contexto configurado. Por ejemplo, para una implementación de la plataforma con dos `TnP-Master Video`, se representa en la figura 16.3 la distribución resultante: cada uno de los *cores* escribe en una mitad del monitor. La ejecución del programa muestra en pantalla los mensajes que identifican el número de contexto, pasando posteriormente a un bucle de espera que muestra un mensaje intermitente por pantalla. Tras la finalización de ese bucle el programa procede a pedir una reconfiguración mediante la escritura de una palabra de configuración `CRW` al controlador `TBC`.

La tabla 16.1 muestra los resultados de implementación de un módulo `TnP-Master Video` para el dispositivo Spartan-II utilizado en el prototipo. La pequeña cantidad de recursos necesarios permite realizar implementaciones con múltiples *cores* de este tipo funcionando en paralelo. La principal limitación para incrementar el número de *cores* de este tipo es el número de bloques de memoria RAM disponibles. En la FPGA utilizada en el prototipo ese número es 12, pero se han sintetizado satisfactoriamente versiones de más de 50 módulos en dispositivos de mayor capacidad.

16.3.2. Controlador de reconfiguración: TBC

La función del `TBC` en el sistema de control de la reconfiguración `Tornado` es la de recibir, recopilar y aplicar las peticiones de reconfiguración provenientes de cualquier *core* con una interfaz maestra conectada al bus *on-chip*.

La implementación del `TBC` utilizada en esta plataforma es la presentada en el apartado 14.3. Es compatible con Wishbone y está parametrizada para el puerto de configuración paralelo SelectMap de los dispositivos Virtex.

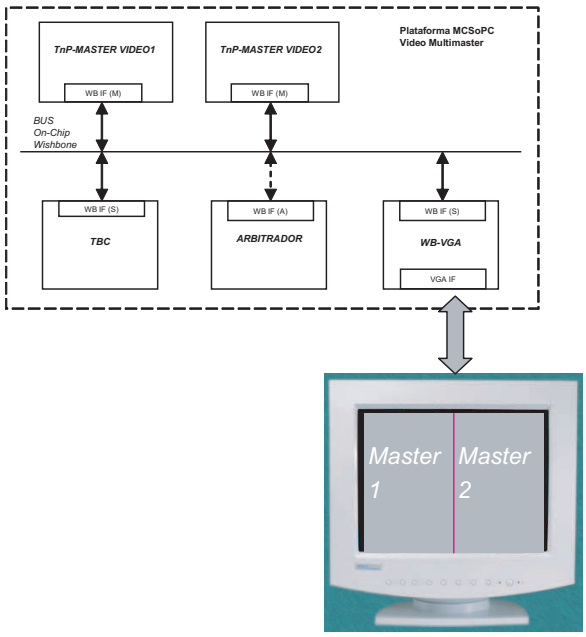


Figura 16.3.: Sistema de visualización de la actividad en los `TnP-Master Video` mediante un monitor seccionado.

Tabla 16.1.: Resultados de implementación del `core TnP-Master Video` en un dispositivo XC2S150.

Tipo de Recurso	**Cantidad requerida**	**Porcentaje de recursos**
Puertas Lógicas Equivalentes	20.362	-
Slices Virtex	100	5 %
LUTs de 4 entradas	168	4 %
Biestables	79	2 %
Bloques de RAM 4K dedicados	1	8 %
Vel. máx. de funcionamiento	70 MHz	-

16.3.3. Controlador de vídeo: WB-VGA

La figura 16.4 muestra la estructura interna del *core* `WB-VGA`. Está compuesto por un total de cuatro registros y dos bloques de memoria de doble puerto. Esta memoria de vídeo se emplaza en bloques de memoria RAM dedicados de la FPGA. Además de las memorias y registros, el *core* precisa de la lógica necesaria para implementar las dos interfaces de que dispone. Para los distintos *cores* maestros del sistema embebido, el `WB-VGA` se presenta como un *core* con 4 registros internos donde se ha de indicar: el carácter que se pretende visualizar en código ASCII de 7 bits; las coordenadas de la pantalla, fila y columna, donde se desea emplazar el carácter (para un tamaño de pantalla de 25 filas por 40 columnas); y el color con el que se desea mostrar el carácter en la pantalla. Éste debe especificarse dentro de un espectro cromático de 64 colores (todas las combinaciones posibles de las señales RGB de

2 bits cada una). Una memoria RAM del `WB-VGA`, utilizada como memoria ROM en este caso, contiene una tabla para la conversión del valor ASCII de 7 bits a su correspondiente combinación de *pixels* (16 de alto por 8 de ancho). La otra memoria que dispone el `WB-VGA` es una memoria RAM de doble puerto donde se almacena el contenido que se desea mostrar en pantalla. Por último, el `WB-VGA` incluye una lógica hardware para la interfaz Wishbone y para la interfaz VGA que es la encargada de generar las señales de sincronismo que necesita el monitor para una correcta visualización de los caracteres.

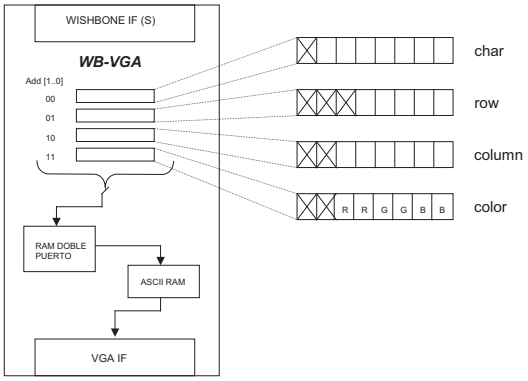


Figura 16.4.: `WB-VGA` *core*.

16.3.4. Módulos auxiliares

Los módulos auxiliares necesarios para componer esta plataforma son:

- Arbitrador de bus: Puesto que coexisten varias interfaces maestras de bus, tantas como módulos `TnP-Master Video`, se ha integrado el arbitrador genérico presentado en el apartado 14.4. El parámetro genérico para este arbitrador es el número de *cores* maestros que se conectan al bus, lo que permite generar automáticamente un arbitrador optimizado para las distintas versiones de la plataforma.

- Decodificador de direcciones: Se ha utilizado también el módulo presentado en el apartado 14.4. En este caso deberá discriminar los accesos a los dos *cores* esclavos: `WB-VGA` y `TBC`.

16.4. Resultados experimentales

En esta sección se presentan los ensayos realizados con diversas implementaciones de la plataforma `Video-Multimaster`. El primer grupo de ensayos está orientado a conocer la exactitud de las fórmulas de estimación de recursos de área requeridos por la infraestructura `Tornado`. El segundo grupo se centra en conocer la evolución de los parámetros de caracterización temporal definidos en el modelado. Estos tiempos permiten obtener una visión realista de los retardos producidos por la utilización de la auto-reconfiguración *intra-task* controlada por `Tornado` en esta tecnología.

16.4.1. Validación de las ecuaciones de modelado

Con el fin de validar el modelado presentado en el apartado 10.2 para la estimación de los recursos de área requeridos por la infraestructura `Tornado`, se realizan varias estimaciones del sobrecoste producido por la misma para distintas configuraciones de la plataforma `Video-Multimaster`. Las distintas configuraciones consisten en implementar 2, 4, 6, 8[1], 10, 12, 14 y 16 *cores* maestros `TnP-Master Video`.

Para obtener las estimaciones se parte de la ecuación general de sobrecoste de recursos de área (ecuación 10.8) en la cual se han sustituido los valores obtenidos para los elementos específicos de la infraestructura `Tornado`. Esta sustitución se realiza en el apartado 14.3, a partir de los datos obtenidos en las implementaciones independientes de los distintos módulos realizadas para la tecnología empleada (ecuación 14.3).

En esta plataforma, la reconfiguración utilizada es de tipo *intra-task* por lo que el término $\mathbf{MACRO_{over}}$, que agrupa los recursos necesarios para las *Bus-Macro* utilizadas, es cero ya que no se requieren ese tipo de módulos en la infraestructura. Por tanto, los parámetros estimados recogidos en la tabla 16.2 se obtienen a partir de la ecuación 16.1, siendo n el número de `TnP-Master Video` implementados. En esta plataforma $x_i = 1$ para todos los `TnP-Cores` ya que incluyen un solo `nano-procesador`.

$$\begin{aligned}\mathbf{Area\_over_{Tornado\_est}} = (189\ lut + 119\ ff + 1\ bram) + + \sum_{i=1}^{n}(8\ lut + 4\ ff)\cdot x_i + \\ + \mathbf{MACRO_{over}}\cdot j = \\ = (189\ lut + 119\ ff + 1\ bram) + (8\ lut + 4\ ff)\cdot n\end{aligned} \tag{16.1}$$

Tabla 16.2.: Estimaciones del coste en recursos área de `Tornado` para distintas variantes de la plataforma `Video-Multimaster`.

Area_over_Tornado_est / nº maestros	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
LUTs de 4 entradas	205	221	237	253	269	285	301	317
Biestables	127	135	143	151	159	167	175	183
Bloques de RAM 4K dedicados	1	1	1	1	1	1	1	1

Para conocer la exactitud de estas estimaciones se comparan con dos conjuntos de implementaciones reales. En el primer conjunto (tabla 16.3), se computan los recursos necesarios para las diferentes variantes de la plataforma sin `Tornado`, por tanto sin posibilidad de admitir auto-reconfiguración parcial dinámica. En el segundo conjunto (tabla 16.4), se recogen los resultados de implementación de las plataformas para el mismo número de *cores* `TnP-Master Video` que en el caso anterior, pero con la infraestructura `Tornado`.

En estas tablas, junto al número de recursos lógicos necesarios de cada tipo se indica el porcentaje de utilización de recursos lógicos disponibles en la FPGA XC2S150 empleada en el

[1]La FPGA utilizada en el prototipo tiene 12 bloques de memoria RAM dedicada, siendo este tipo de recurso el que limita el número de maestros a 8 para las plataformas que se pueden experimentar en el prototipo.

prototipo. Esta FPGA tiene 12 bloques de memoria RAM dedicada, siendo el tipo de recurso que limita el número de maestros. Por ello, los datos de porcentaje únicamente se representan para las plataformas de 2 hasta 8 maestros en el primer grupo de implementaciones y de 2 hasta 6 maestros para el segundo, ya que, con la capacidad del dispositivo utilizado en el prototipo, son las que realmente se pueden implementar.

Tabla 16.3.: Resultados de implementar las variantes de la plataforma `Video-Multimaster` sin infraestructura `Tornado`.

Recursos lógicos / nº maestros	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
Puertas Lógicas Equivalentes	107.522	148.118	188.360	229.304	269.907	310.499	351.220	391.069
Slices Virtex	290 (16 %)	493 (28 %)	696 (40 %)	905 (52 %)	1.114 (- %)	1.313 (- %)	1.518 (- %)	1.672 (- %)
LUTs de 4 entradas	444 (12 %)	773 (22 %)	1.103 (31 %)	1.429 (41 %)	1.757 (- %)	2.084 (- %)	2.428 (- %)	2.636 (- %)
Biestables	266 (7 %)	452 (12 %)	639 (18 %)	824 (23 %)	1.012 (- %)	1.199 (- %)	1.386 (- %)	1.572 (- %)
Bloques de RAM 4K dedicados	6 (50 %)	8 (66 %)	10 (83 %)	12 (100 %)	14 (- %)	16 (- %)	18 (- %)	20 (- %)

Tabla 16.4.: Resultados de implementar las variantes de la plataforma `Video-Multimaster` con la infraestructura de `Tornado`.

Recursos lógicos / nº maestros	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
Puertas Lógicas Equivalentes	128.234	169.044	209.871	250.740	291.535	323.403	372.518	413.484
Slices Virtex	423 (24 %)	640 (37 %)	860 (49 %)	1.085 (- %)	1.314 (- %)	1.535 (- %)	1.721 (- %)	1.726 (- %)
LUTs de 4 entradas	642 (18 %)	987 (28 %)	1.333 (38 %)	1.686 (- %)	2.029 (- %)	2.381 (- %)	2.711 (- %)	2.978 (- %)
Biestables	373 (11 %)	573 (17 %)	774 (23 %)	975 (- %)	1.185 (- %)	1.387 (- %)	1.586 (- %)	1.789 (- %)
Bloques de RAM 4K dedicados	7 (58 %)	9 (75 %)	11 (91 %)	13 (- %)	15 (- %)	17 (- %)	19 (- %)	21 (- %)

Utilizando estos dos conjuntos de implementaciones se obtienen los datos reales del coste de la infraestructura `Tornado`. Para ello se restan las cantidades de cada tipo de recurso lógico requerido para cada implementación en las dos versiones (con `Tornado` y sin él).

La comparación entre ambos grupos se resume en la tabla 16.5, en la que se especifica el error entre los datos reales del coste de la infraestructura `Tornado` frente a los obtenidos en la estimación realizada con las ecuaciones del modelado. El cálculo de los porcentajes de error para cata tipo de recurso lógico se ha realizado según el criterio establecido en la ecuación 16.2:

$$error = \frac{recursos\_estimados - recursos\_reales}{recursos\_reales} \tag{16.2}$$

En todos los casos el error en la estimación de los bloques de memoria RAM es cero. Esto se debe a que el BRAM necesario en la infraestructura `Tornado` para esta tecnología se instancia directamente en el código HDL del `TBC` (donde se incluye su software). Por tanto, en ningún proceso de optimización que realicen las herramientas de emplazamiento y rutado éste puede ser eliminado.

Sin embargo, los errores en las estimaciones de LUTs y biestables varían para las distintas implementaciones. En la gráfica 16.5 se ha representado la evolución de los mismos. Son más acusados para las plataformas de menor complejidad. Sin embargo, a partir de cuatro *cores* `TnP-Master Video` (situación a partir de la cual resulta rentable la utilización de la auto-reconfiguración, tal y como se analiza con la siguiente plataforma de validación) el error se mantiene en una franja del +/- 15 %. Cada plataforma es una situación distinta para las herramientas de emplazamiento y rutado. Tanto para aplicar simplificaciones y optimizaciones, como para añadir lógica adicional que permita el cumplimiento de las restricciones temporales impuestas. Por ello, el error es notablemente distinto para cada variante. Un error dentro de este rango es asumible para la fase de estudio de viabilidad de la plataforma con auto-reconfiguración, teniendo en cuenta que al utilizar las ecuaciones de modelado no es necesario tener el diseño detallado.

Tabla 16.5.: Recursos lógicos reales y estimados de la infraestructura `Tornado`.

Parámetros / nº maestros	**2**	**4**	**6**	**8**
Area_over$_{Tornado_est}$	205lut+127ff+1bram	221lut+135ff+1bram	237lut+143ff+1bram	253lut+151ff+1bram
Area_over$_{Tornado_real}$	298lut+107ff+1bram	214lut+121ff+1bram	230lut+135ff+1bram	257lut+151ff+1bram
Error estimación LUTs	-31 %	3,2 %	6,3 %	-1,5 %
Error estimación biestables	18,6 %	11,6 %	5,9 %	0 %
Error estimación bloques de memoria RAM	0 %	0 %	0 %	0 %
	10	**12**	**14**	**16**
Area_over$_{Tornado_est}$	269lut+159ff+1bram	285lut+167ff+1bram	301lut+175ff+1bram	317lut+183ff+1bram
Area_over$_{Tornado_real}$	272lut+173ff+1bram	297lut+188ff+1bram	283lut+200ff+1bram	342lut+217ff+1bram
Error estimación LUTs	-1,1 %	-4 %	6,3 %	-7,3 %
Error estimación biestables	-8 %	-11,1 %	12,5 %	-15,2 %
Error estimación bloques de memoria RAM	0 %	0 %	0 %	0 %

Estos resultados muestran que es posible estimar de forma previa a la realización de un diseño, el coste en recursos que supondrá que éste sea compatible con el sistema `Tornado`. Por tanto, las ecuaciones de modelización son una herramienta útil para la fase de análisis y estudio de la viabilidad del diseño.

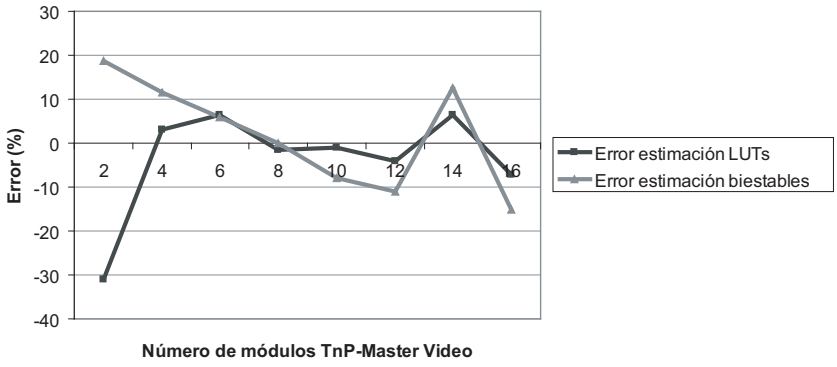


Figura 16.5.: Evolución del error en las estimaciones de recursos lógicos en función de la complejidad de la plataforma.

16.4.2. Caracterización temporal

Uno de los principales inconvenientes de la reconfiguración dinámica es la penalización temporal producida durante la aplicación de la misma. En las plataformas auto-reconfigurables, como la presentada, además de tenerse en cuenta el tiempo utilizado para realizar la reconfiguración parcial, se debe considerar el tiempo utilizado por el sistema de control de la auto-reconfiguración. Por ello, con el fin de cuantificar esa penalización temporal global, es necesario considerar no solamente el tiempo de carga del *bitstream*, sino también todos los parámetros temporales definidos en el apartado 10.1 que caracterizan todas las operaciones involucradas en la auto-reconfiguración.

Para conocer la magnitud de estos parámetros con la tecnología disponible y en el prototipo desarrollado, se han cuantificado estos parámetros para diferentes implementaciones de `Tornado`. Estas implementaciones varían en el número de `TnP-Video Masters`. Éstos, además de escribir en el periférico de vídeo mensajes según el contexto software que estén ejecutando, son los encargados de realizar las peticiones de reconfiguración al `TBC`. Para normalizar los ensayos, los `TnP-Video Master` realizan las peticiones de reconfiguración según las condiciones establecidas en el apartado 10.1.3.

El primer conjunto de tiempos, representados en la tabla 16.6, son los obtenidos para los parámetros temporales normalizados definidos en el apartado 10.1.3. En la figura 16.6 se indica la localización en la secuencia de tiempos de cada uno de estos parámetros. `STB` y `ACK` son las señales de la especificación Wishbone que controlan los ciclos de acceso al bus. Utilizando estos datos normalizados se caracterizan todas las versiones de la plataforma con auto-reconfiguración, ya que estos tiempos no dependen del número de *cores* del sistema.

Sumando los dos parámetros globales **Tcrw0**[2] y **Tstb0**[3] se obtiene un tiempo de 26.500 ciclos de reloj. Teniendo en cuenta que en prototipo se utiliza un reloj de 50 MHz, se necesita un total de 530 μs para realizar una reconfiguración software de un `TnP-Core`. Se debe tener

[2] "Tiempo asociado a la palabra `CRW`" (*CRW Time*), normalizado a la escritura de una palabra `CRW` en un bus compartido.

[3] "Tiempo de activación de la señal `STB_RECONF`" (*Strobe Time* -**Tstb**-) normalizado a un único intento de aplicación de reconfiguración.

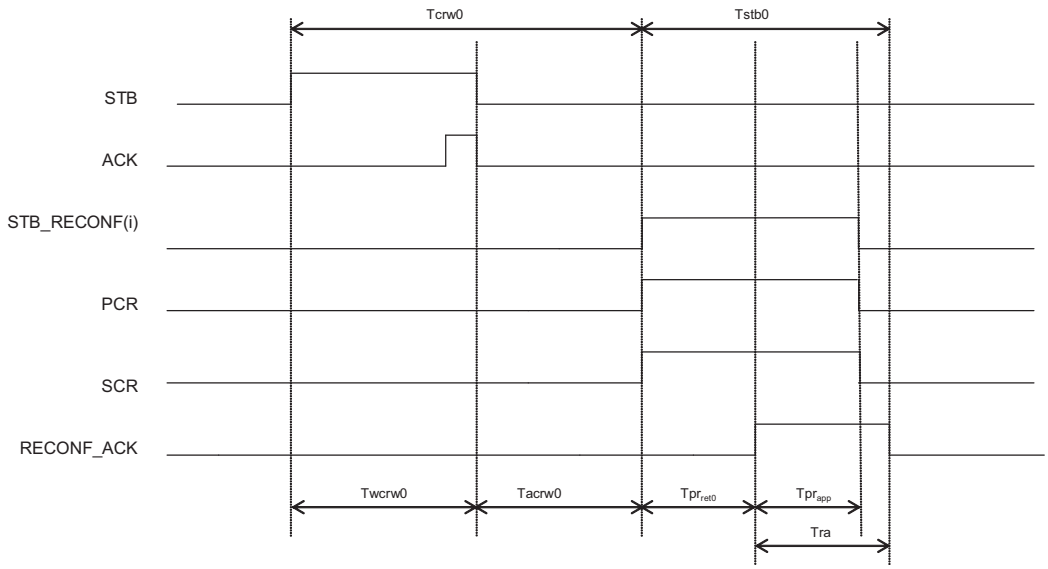


Figura 16.6.: Parámetros temporales normalizados.

en cuenta que la normalización realizada caracteriza el caso más favorable: el *core* que escribe la petición es atendido inmediatamente, no hay rechazos de la reconfiguración y la pila de peticiones de reconfiguración en el `TBC` está vacía. Aún asumiendo que según la aplicación concreta, estos tiempos van a ser mayores dependiendo del tráfico en el bus *on-chip* y de la frecuencia de las peticiones, son valores aceptables para muchas aplicaciones.

Tabla 16.6.: Valores normalizados de los parámetros temporales para la plataforma `Video Multi-master`

Parámetro	**Ciclos de reloj**	**Tiempo en el prototipo**[4]
Twcrw0	37	0,74 μs
Tacrw0	134	2,68 μs
Tcrw0	171	3,42 μs
$\mathbf{Tpr_{ret0}}$	1	20 ns
$\mathbf{Tpr_{app}}$	26.328	526,56 μs
Tstb0	26.329	526,58 μs

El parámetro que caracteriza el proceso de carga del *bitstream* parcial, $\mathbf{Tpr_{app}}$, es el de mayor peso con la normalización realizada. Tiene un valor constante para un determinado tamaño de *bitstream* y es proporcional a éste. La reconfiguración *intra-task* utilizada en esta plataforma es la clave para obtener *bitstreams* de tamaño reducido que permitan una rápida reconfiguración. El tamaño de estos *bitstreams* parciales es de 4.388 bytes, aproximadamente un 3.4 % del tamaño del *bitstream* global. El sistema de carga utilizado en el prototipo no

[4]La CPLD del sub-sistema de carga requiere de seis ciclos de reloj ($T_{clk} = 20ns$) para leer un byte del *bitstream* parcial y escribirlo en el puerto de configuración SelectMap.

funciona a la máxima velocidad que el puerto de configuración SelectMap admite en modo paralelo (50 MByte/s), por lo que este tiempo se puede reducir más con un sub-sistema de carga y almacenamiento de *bitstreams* mejorado.

En la tabla 16.7 se muestran los parámetros no normalizados para las versiones de la plataforma `Video-Multimaster` con 2, 4, 6 y 8 `TnP-Video Master`. Estos valores están capturados en los accesos de los últimos maestros en la secuencia de arbitraje. Es decir, que todos ellos solicitan simultáneamente una reconfiguración y se miden los tiempos del último en ser atendido. Éstos son los casos más desfavorables en cuanto al acceso al bus *on-chip*, puesto que todos ellos solicitan el acceso al bus de forma simultánea para escribir la `CRW` y además, hay un tráfico abundante en el bus generado por todos los `TnP-Video Master` escribiendo en el `WB-VGA` la información para mostrar en pantalla. El tiempo de activación de la señal `STB_RECONF`(**Tstb**) es constante puesto que para este ensayo se mantiene siempre habilitada la reconfiguración en los `TnP-Cores`.

Tabla 16.7.: Tiempos no normalizados para las plataformas `Video-Multimaster`.

Parámetro	**2 Maestros**	**4 Maestros**	**6 Maestros**	**8 Maestros**
Tcrw	1.447 clk	4.051 clk	6.735 clk	9.721 clk
Tstb	26.329 clk	26.329 clk	26.329 clk	26.329 clk

Ya que el número de bloques de memoria RAM de estas versiones no supera el máximo disponible en la FPGA del prototipo, estos tiempos se han podido medir en la propia placa. La figura 16.7 muestra una captura de pantalla del analizador lógico utilizado en estos ensayos. La secuencia representada incluye las señales de la interfaz `RM IF` del controlador `TBC` al iniciarse una carga de un *bitstream* parcial por orden de éste. El `TBC`, tras haber procesado la palabra de configuración, ha determinado que se debe cargar el *bitstream* parcial número '2' , lo cual indica a la CPLD del sub-sistema de carga y almacenamiento escribiendo ese número en el bus de datos. A partir de ese instante, el `TBC` junto con la CPLD realizan la carga en paralelo del *bitstream* parcial en la FPGA.

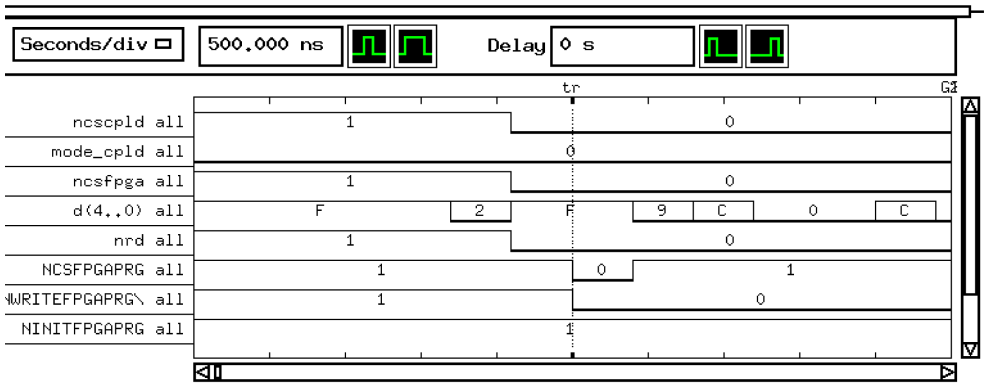


Figura 16.7.: Carga de un *bistream* parcial controlada por el `TBC`.

16.4.3. Problemática de la reconfiguración parcial por columnas para la reconfiguración intra-task software

En los ensayos realizados con el prototipo, al aplicar la reconfiguración *intra-task* software se ha experimentado un problema en la reconfiguración de los bloques BRAM adicional al de la posible lectura de una memoria cuando se está escribiendo mediante reconfiguración. Esta última situación se tiene en cuenta en el sistema de control `Tornado`, el cual bloquea el acceso de los `nano-procesadores` a estas memorias durante el proceso de reconfiguración. Sin embargo, el problema detectado es un ejemplo de las restricciones específicas que impone una determinada tecnología a la aplicación de `Tornado` con la misma.

El programa de los `nano-procesadores` de los `Cores Mixtos` y de los `TnP-cores`, se almacena en BRAMs. Estos bloques están distribuidos en columnas dentro del dispositivo. El número de columnas de BRAMs y el número de bloques por columna dependen del tamaño de la FPGA. Los *frames* que definen su configuración (BRAM *frames* -figura 16.8-), tienen en cuenta a todos los BRAMs de una misma columna. Esta característica implica que el contenido de varios BRAM se deba especificar con los mismos BRAM *frames*.

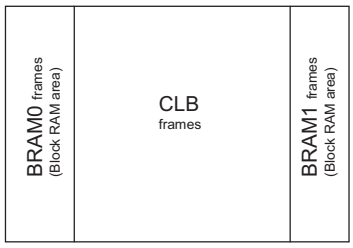


Figura 16.8.: Distribución de los BRAM *frames* en un dispositivo Virtex con dos columnas de BRAMs.

Puesto que la unidad mínima de configuración es el *frame*, una operación de escritura para realizar reconfiguración parcial reescribe todos los bits desde la parte superior hasta la inferior. Esta característica tecnológica hace que el cambio de contenido de un único bloque de RAM no sea una tarea trivial. La herramienta que genera los *bitstreams* parciales [223] en el flujo de diseño de Xilinx, incluye en los mismos únicamente las diferencias entre un fichero de *Place & Route* con el diseño completo original y otro fichero de *Place & Route* que incluye los cambios para la reconfiguración parcial [98]. Aunque se altere un único BRAM, el *bitstream* parcial obtenido para realizar la reconfiguración incluye los *frames* de la columna correspondiente a ese BRAM. En esta operación, al verse afectados todos los BRAM de la columna, si un BRAM de esa columna ha sido alterado previamente (después de la configuración inicial), el contenido sería reescrito de nuevo. Para poder realizar cambios selectivos de contexto de los distintos `nano-procesadores` distribuidos en el MCSoPC se proponen dos soluciones:

- **Generación dinámica de *bitstreams*:** Esta alternativa surge del uso de una característica tecnológica de estos dispositivos: la lectura de la configuración aplicada a través de los recursos de configuración o *readback* [80]. Mediante este mecanismo se habilita la opción de leer el contenido de los BRAM antes de crear el *bitstream* parcial, incluyendo en este caso únicamente los cambios en el BRAM correspondiente al

`nano-procesador` cuyo contexto se desea cambiar. Tecnológicamente los nuevos dispositivos Virtex-II [40] y Virtex-II Pro [160] proveen mecanismos para poder realizar todas estas operaciones de forma completamente interna al dispositivo (módulo ICAP [155]).

- ***Bitstreams* cruzados:** Esta opción pasaría por disponer de *bitstreams* con las combinaciones cruzadas de los distintos contextos software para los `nano-procesadores` que contengan el programa en la misma columna de BRAM. Esta alternativa aumenta de manera significativa el número de *bitstreams* parciales que se deben almacenar, pero supone una solución más sencilla que la anterior.

17. Plataforma de verificación avanzada I: AMT

17.1. Descripción general

La reconfiguración *intra-task software* de los *cores* con arquitectura de `Core Mixto` se ha experimentado en la plataforma simple presentada en el apartado anterior. El sistema `Tornado` también debe ser validado para reconfiguración *intra-task hardware/software*. Con este fin, se ha realizado un diseño que incluye múltiples `TnP-Cores` capaces de soportar reconfiguración hardware/software. Este diseño integra un conjunto de transmisores digitales con capacidad de síntesis digital directa de señal (*Direct Digital Synthesis* -DDS-) para la banda de HF. Estos módulos tienen una arquitectura de `Core Mixto` y transmiten la información digital enviada desde un ordenador externo (*host*).

La modulación generada por los `TnP-Cores` transmisores se modifica mediante reconfiguración parcial dinámica, alterándose con este objetivo, tanto el software como el hardware de los `TnP-Cores` transmisores. Dada esta capacidad de configuración, se ha denominado a esta plataforma `Adaptive Multi-Transmitter` (`AMT`). El cambio de modulación puede solicitarse bien desde un *host* externo o bien internamente mediante un sistema de evaluación de la calidad del enlace de radio.

De igual forma que en la plataforma de verificación previamente presentada, los elementos de automatización se han adecuado al flujo de diseño para reconfiguración parcial dinámica aplicando pequeñas modificaciones al *bitstream* tal y com se detalla en [98].

17.2. Diagrama de bloques

La arquitectura general de la plataforma `AMT` se representa en la figura 17.1. Para la interconexión de los *cores* se ha empleado de nuevo la especificación estándar Wishbone, integrando como unidades de procesamiento software los `TnPs`. El CSoPC multi-procesador está formado por `TnP-Cores` y por IP-Cores, conectados mediante un bus *on-chip* con topología de bus compartido.

Los módulos `DDS-TXi` son `Cores Mixtos` capaces de generar directamente una señal de HF modulada para transmitir información digital. La información la transmite un *host*, siendo recibida por estos módulos a través de la interfaz del bus *on-chip*. Estos *cores* disponen de diferentes configuraciones para poder generar distintas modulaciones, lo que requiere el cambio del software interno y una pequeña modificación hardware. El número de transmisiones simultáneas (en distintos canales) está limitado por el número de `DDS-TXis` implementados (un canal por cada *core*). Se han realizado varias versiones de la plataforma con distinto número de módulos transmisores.

El *core* `RSSI-LD` evalúa la calidad del enlace de radio y dispone de la capacidad de ordenar reconfiguraciones de los *cores* `DDS-TXi`. Para ello escribe la orden correspondiente

al controlador TBC. En este *core* se integra un TnP, admitiendo también cambios de contexto si la aplicación lo requiere.

Los *cores* WB-USB y WB-VGA son compatibles con la especificación Wishbone, pero no admiten reconfiguración parcial controlada por Tornado. Mediante el WB-USB se conecta la plataforma al *host* externo y el WB-VGA permite visualizar información en un monitor VGA.

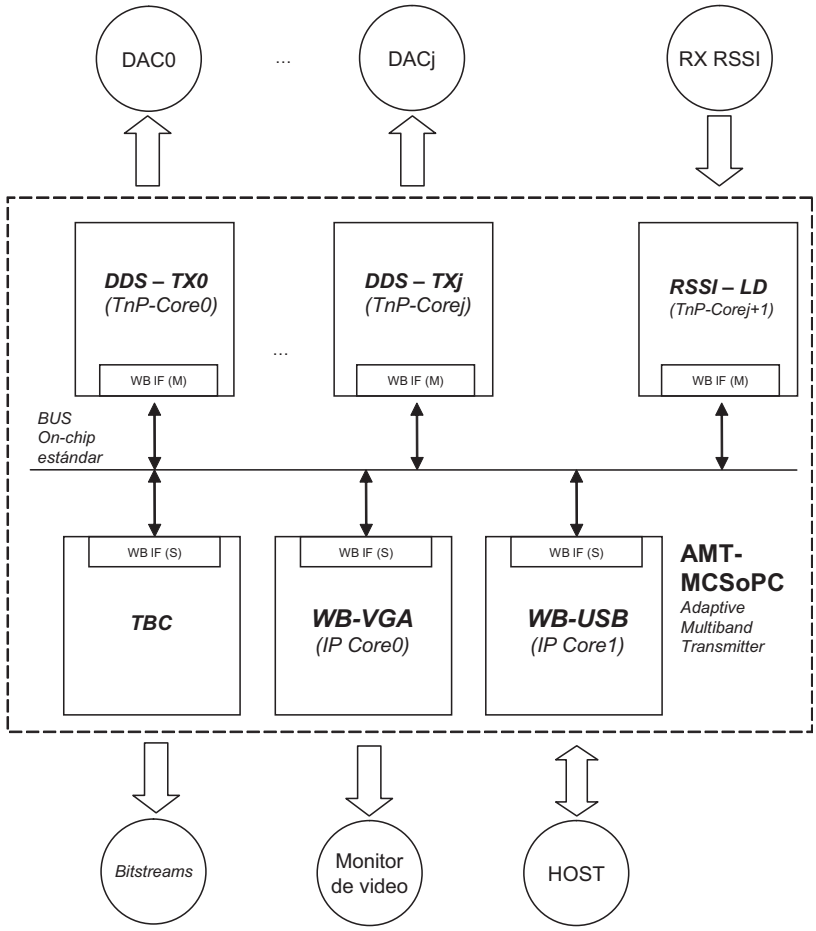


Figura 17.1.: Plataforma AMT desarrollada para la evaluación del sistema Tornado.

17.3. Módulos IP

Se resumen a continuación las características más relevantes de los módulos que componen el sistema.

17.3.1. DDS-TXi

Estos TnP-cores generan la señal modulada de alta frecuencia. La generación digital directa de las señales de RF [224, 225] requiere de la potencia de procesamiento hardware en la etapa de generación, de forma que se pueda disponer de múltiples transmisores

simultáneos en el mismo integrado. Los *cores* `DDS-TXi` admiten la reconfiguración parcial dinámica controlada mediante `Tornado`.

La estructura interna de `Core Mixto` de estos módulos se representa en la figura 17.2. Cada *core* incluye un `TnP` en la sección software. En la sección hardware se ha implementado el algoritmo Cordic [226] mediante un circuito en 15 etapas *pipeline* [227], incluyéndose además un acumulador de fase y una interfaz maestra Wishbone para la conexión del *cores* al bus *on-chip*. La precisión utilizada es de 16 bits.

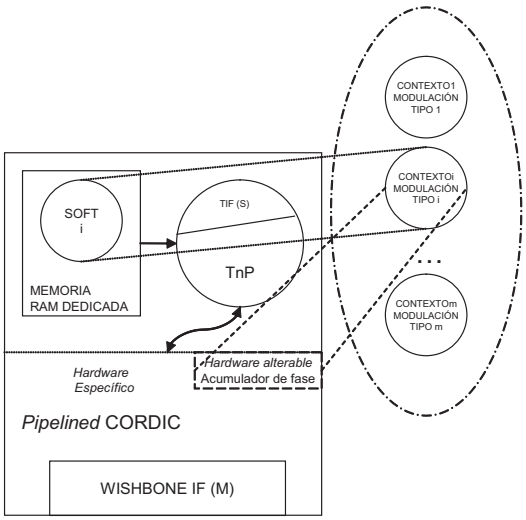


Figura 17.2.: Partición interna del `TnP-Cores DDS-TXi`.

El algoritmo Cordic permite obtener el valor del seno y del coseno a partir de un ángulo dado. Variando el ángulo de entrada se obtiene en estas salidas las modulaciones I y Q en el rango de alta frecuencia. Esta frecuencia se determina por medio del reloj global de alta velocidad utilizado para todo el diseño, multiplicado por dos mediante la DLL interna de la FPGA. Este reloj de frecuencia doble sincroniza la operación del Cordic.

El acumulador de fase establece el valor del ángulo de entrada para el Cordic. El `TnP` integrado en el `DDS-TXi` puede escribir en el acumulador de fase el módulo de incremento de la misma de modo sincronizado. Esta escritura se realiza en la rutina de atención a la interrupción externa. El evento que produce esta interrupción es una señal que se obtiene de un circuito divisor de reloj. Este circuito, integrado también en cada módulo `DDS-TXi` determina, por tanto, la velocidad de la transmisión de datos. Está basado en un contador sincronizado con el acumulador de fase cuyo módulo de 16 bits es accesible en escritura

Mediante este circuito y un acumulador de fase se generan la señales sinusoidales de alta frecuencia empleadas para la transmisión de datos digitales. De forma externa a la FPGA se requiere un convertidor Digital-Analógico por cada módulo transmisor.

Los tipos de modulación que soportan los `DDS-TXi` en esta plataforma son: FSK, OOK y PSK. Cada modo requiere un programa diferente para el `TnP` y cambios hardware localizados en el acumulador de fase previo al circuito Cordic. Con el fin de simplificar estos cambios y facilitar la experimentación, éstos se han realizado únicamente en cuatro *slices* donde se comparan los valores de los cuatro bits más significativos del valor del acumulador de

fase. Éstos valores cambian para cada tipo de modulación debido a las características de convergencia del Cordic. También se han generado otras versiones con cambios en los IOBs de salida mediante reconfiguración. Los tamaños de los distintos *bitstreams* generados se comparan en el apartado 17.4.1.

En el caso de que se necesiten realizar cambios de mayor entidad en el hardware mediante reconfiguración *intra-task*, se facilita[1] el diseño instanciando en el código HDL directamente los componentes necesarios de la propia FPGA (no dejando que el sintetizador los infiera automáticamente) y, mediante atributos de localización, fijar ese componente en una determinada posición. De esta forma, es más sencilla la identificación posterior de los recursos lógicos que hay que modificar en la herramienta de edición gráfica con la que se edita el diseño ya emplazado y rutado.

Los `TnP-Cores` transmisores se han simplificado al máximo para disponer de un funcionamiento robusto y para optimizar el área el diseño obteniendo. Esta optimización en área permite una simplificación del *core*, reduciéndose la complejidad de los procesos de emplazamiento y rutado, lo que redunda, a su vez, en un aumento de la velocidad máxima de funcionamiento del diseño para un dispositivo concreto y un número determinado de *cores*.

A través de la interfaz estándar Wishbone, cada *core* `DDS-TXi` recibe la información que debe transmitir enviada por el *host*. Puesto que ha sido dotado de una interfaz maestra, es el propio módulo el encargado de iniciar los ciclos de acceso al bus. Para ello detecta, mediante señales auxiliares, la disponibilidad de datos para ese módulo en los *cores* esclavos de comunicaciones (en este caso el *core* `WB-USB`).

Estos datos, además de la información que debe ser transmitida, pueden incluir comandos que interpreta el software del `DDS-TXi`. Uno de estos comandos permite indicar al propio `DDS-TXi` que sea él mismo el que solicite un cambio de modulación al `TBC`. De esta forma se consigue un control activo del tipo de modulación utilizada desde el *host*.

Tabla 17.1.: Resultados de implementación del `TnP-Core DDS-TXi` sobre una FPGA Spartan-II X2S150-6

Tipo de Recurso	**Ocupación (Optimización área)**	**Ocupación (Optimización velocidad)**
Puertas Lógicas Equivalentes	35.718	35.822
Slices Virtex	517 (29 %)	527 (30 %)
LUTs de 4 entradas	935 (27 %)	940 (27 %)
Biestables	843 (24 %)	843 (24 %)
Bloques de RAM 4K dedicados	1 (8 %)	1 (8 %)
Frecuencia máxima generable (DDS)	147 MHz	149 MHz

En la tabla 17.1 se muestran los resultados de implementación de un *core* `DDS-TXi` de manera independiente (sin el resto de *cores* que componen la plataforma). En este caso, además de aplicarse la arquitectura de `TnP-Core`, se ha utilizado una característica tecnológica de los dispositivos de Xilinx como es la posibilidad de disponer de un reloj global de frecuencia doble a la de entrada. La sección de procesamiento Cordic se ha conectado a este reloj con el fin de obtener una la señal generada directamente de frecuencia elevada. Los datos

[1]Esta recomendación la realiza el autor en base a los experimentos realizados con esta plataforma.

obtenidos de la frecuencia máxima de la señal generable mostrados en esta implementación son orientativos, ya que este *core* en la plataforma real está integrado con otros módulos, lo que determina unas frecuencias de funcionamiento más bajas.

17.3.2. RSSI-LD

Cada tipo de modulación que admiten los *cores* `DDS-TXi` tiene mejores características según la calidad del enlace de radio en el que se realiza la transmisión de datos. En esta plataforma, se plantea la posibilidad de que el MCSoPC de forma autónoma sea capaz de cambiar la modulación de los `DDS-TXi` según el estado del canal empleando la reconfiguración parcial dinámica.

El `TnP-Core RSSI-LD` (*Radio Signal Strength Indicator - Level Detector*) es el encargado de evaluar la calidad de la señal recibida a través del canal de RF. Para ello realiza un muestreo continuo del nivel de la señal recibida, transmitida desde una baliza remota, mediante la lectura de la información analógica que provee el receptor [228]. El software del `TnP` de este *core* tiene definidos unos rangos de la señal RSSI para cada tipo de modulación soportada por los `DDS-TXi`. La sección hardware incluye la parte digital de un convertidor Analógico-Digital *sigma-delta* (ADC) cuyo código HDL está basado en el diseño ADC-DAC propuesto por Xilinx en [229, 230].

Este módulo necesita únicamente un amplificador operacional externo para realizar la conversión ADC. En la tabla 17.2 se muestran los resultados de implementación de este *core* de forma independiente. De nuevo, la aplicación de la arquitectura de `Core Mixto` ofrece unos resultados equilibrados en términos de recursos necesarios y velocidad máxima de funcionamiento.

Tabla 17.2.: Resultados de implementación del `TnP-Core RSSI-LD` en una FPGA Spartan-II X2S150

Tipo de Recurso	**Ocupación (Optimización área)**	**Ocupación (Optimización velocidad)**
Puertas Lógicas Equivalentes	21.529	21.576
Slices Virtex	155 (8 %)	161 (9 %)
LUTs de 4 entradas	249 (7 %)	262 (7 %)
Biestables	148 (4 %)	148 (4 %)
Bloques de RAM 4K dedicados	1 (8 %) (%)	1 (8 %)
Frecuencia máxima de funcionamiento	73,5 MHz	74 MHz

17.3.3. WB-USB

En esta aplicación la información que hay que transmitir proviene de un *host* externo. En el prototipo utilizado la conexión con el *host* se realiza mediante USB, habiéndose dotado al prototipo de un integrado USB FT245UM [231]. La interfaz entre este dispositivo y el bus *on-chip* Wishbone se realiza mediante el *core* esclavo `WB-USB` perteneciente al catálogo de *cores* desarrollados en nuestro grupo de investigación (APERT) [232].

Este *core* tiene la lógica necesaria para determinar cuál de los módulos transmisores es el destinatario de la información recibida desde el *host*. Mediante una señal dedicada para cada *core* transmisor `DDS-TXi` (una señal TAG definida en la especificación Wishbone), el *core* interfaz USB-Wishbone indica al *core* maestro que hay datos pendientes para ser leídos.

17.3.4. Controlador de vídeo: WB-VGA

El *core* `WB-VGA` es un periférico Wishbone esclavo que sintetiza directamente las señales VGA permitiendo la conexión de un monitor VGA [233]. A través de este módulo, cualquiera de los *cores* maestros conectados al bus *on-chip* pueden mostrar información en forma de texto en el monitor. Por ejemplo, el tipo de modulación aplicada a cada transmisor u otros parámetros del sistema.

La arquitectura interna y la funcionalidad de este *core* se ha presentado en la plataforma de validación anterior (sección 16.3.3).

17.3.5. Controlador de reconfiguración: TBC

La función del `TBC` en el sistema `Tornado` es la de recibir, recopilar y aplicar la peticiones de reconfiguración (`CRW`) provenientes de cualquier *core* con una interfaz maestra conectada al bus *on-chip* (sección 9.6). En la plataforma `AMT`, las peticiones de reconfiguración pueden ser emitidas por dos tipos de fuentes:

- Los módulos transmisores `DDS-TXi` pueden escribir una palabra `CRW` en el controlador de reconfiguración `TBC` a través del bus *on-chip* para realizar una solicitud de auto-reconfiguración. En este caso, el módulo transmisor `DDS-TXi` habrá recibido previamente del *host*, a través del `WB-USB`, un comando para forzar el cambio a otra modulación.
- El *core* (o *cores* de forma general) `RSSI-LD` encargado de monitorizar la calidad del enlace puede escribir una `CRW` en el controlador `TBC` para modificar el tipo de modulación de los transmisores. Cuando los valores de cuantificación de la calidad del canal pasa a otro segmento definido para otro tipo de modulación, se escribe en el `TBC` el tipo de modulación correspondiente.

En esta plataforma se ha empleado de nuevo el controlador de reconfiguración optimizado para la tecnología Virtex y compatible con la especificación Wishbone cuyas características se han presentado en el apartado 14.3.

17.3.6. Módulos auxiliares

Los módulos auxiliares requeridos para componer esta plataforma son:

- `Arbitrador de bus`: Puesto que coexisten varios interfaces maestros de bus (tantos como módulos transmisores `DDS-TXi`) se ha integrado el arbitrador genérico presentado en el apartado 14.4. El parámetro genérico para este arbitraje es el número de *cores* maestros que se conectan en el bus, lo que permite generar automáticamente un arbitrador optimizado para las distintas versiones de la plataforma.

- `Decodificador de direcciones`: Se ha utilizado también el módulo presentado en el apartado 14.4. En este caso, deberá discriminar los accesos a los tres *cores* esclavos: `WB-USB`, `WB-VGA` y `TBC`.

17.4. Resultados experimentales

Mediante esta plataforma de validación se muestra un tipo de aplicaciones susceptibles de ser mejoradas mediante el uso de la auto-reconfiguración parcial dinámica.

Para la generación digital directa de señal se utilizan sistemas diferentes. En muchas aplicaciones incluso se opta por una solución completamente software, generalmente mediante el uso de procesadores digitales de señal. Sin embargo, la necesidad de disponer en un único integrado de múltiples transmisores funcionando en paralelo junto con otros *cores* específicos de la aplicación, reduce el abanico de soluciones que ofrezcan la concurrencia del procesamiento hardware y, por qué no, de forma conjunta la flexibilidad del software.

Descartando las soluciones basadas en ASICs, dado su enorme coste y su limitada flexibilidad (incluso considerando soluciones con secciones reconfigurables), destacan como dispositivos óptimos para este tipo de diseños del área de *Software-Defined-Radio* las FPGAs de grano fino [224, 225].

La comparativa de las diferentes propuestas realizada en capítulo 5 muestra cómo la reconfiguración hardware/software *intra-task* de `Cores Mixtos` no está soportada por ninguna de las alternativas actuales. Por lo tanto, no es posible realizar una comparativa directa con ningún otro sistema que permita el control de la auto-reconfiguración parcial tal y como se plantea en la plataforma `AMT`. Por ello, en esta sección se comparan dos grupos de diseños con los mismos requisitos (número de canales, número de modulaciones, etc.), pero englobando en uno las soluciones convencionales (sin auto-reconfiguración) y en el otro las soluciones con auto-reconfiguración controlada por `Tornado`.

Las plataformas convencionales necesitan tantos *cores* como modulaciones diferentes o canales. Por ello, se incluye un sistema de multiplexado para los los transmisores `DDS-TXi`. Estas plataformas no tienen la infraestructura `Tornado`. Todos los módulos necesarios para todas las modulaciones se cargan desde el principio en la FPGA.

Para cuantificar la mejora obtenida, se presentan a continuación dos tipos de análisis:

- **Idoneidad de la auto-reconfiguración en función del número de modulaciones diferentes para un canal.** Se comparan un conjunto de implementaciones de plataformas `AMT` con un único canal y distinto número de modulaciones (diseños con auto-reconfiguración y sin ella).

- **Idoneidad de la auto-reconfiguración en función del número de canales simultáneos.** Utilizando un dispositivo lógico Virtex de alta capacidad, se realizan implementaciones de múltiples transmisores funcionando en un mismo dispositivo lo que permite estudiar la evolución de los recursos lógicos y las frecuencias máximas de funcionamiento.

17.4.1. Idoneidad de la auto-reconfiguración en función del número de modulaciones diferentes para un canal.

Las gráficas 17.3(a), 17.3(b) y 17.3(c) muestran la cantidad de recursos lógicos necesarios de tipo combinacional (LUTs), de tipo secuencial (biestables) y de bloques de memoria RAM para los dos grupos de implementaciones realizadas de la plataforma `AMT`: convencionales (MSoPC) e implementaciones con auto-reconfiguración controlada por `Tornado` (MCSoPC). Todas ellas tienen un único canal activo y admiten hasta cinco tipos de modulaciones diferentes.

En el caso de las implementaciones con `Tornado` con un único canal activo, los recursos lógicos de la FPGA requeridos son constantes[2]. Las plataformas convencionales con un único canal y con una o dos modulaciones, son más simples que las adaptadas para admitir la auto-reconfiguración. En estas situaciones, no estaría justificado su uso aún con una infraestructura tan simple como la propuesta.

En el caso de tres modulaciones y un único canal, en cuanto a recursos lógicos, la situación está equilibrada. Sin embargo, hay que considerar que la auto-reconfiguración implica retardos en el funcionamiento del sistema (escritura de las palabras de configuración y carga de los *bitstreams*) y complica el sub-sistema de carga y almacenamiento de *bitstreams*, ya que éste requiere seguir activo durante el funcionamiento del sistema. Por ello, únicamente será atractiva esta solución cuando el ahorro obtenido sea considerable. En este caso, a partir cuatro o cinco tipos de modulación, se debe considerar la alternativa de la auto-reconfiguración.

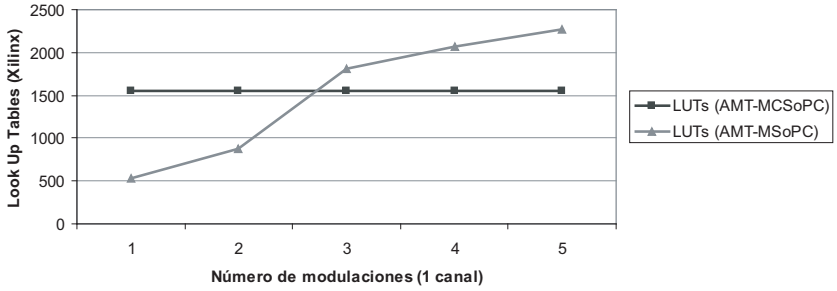
Uno de los mayores inconvenientes de las FPGAs, y en especial de las de grano fino, es la acumulación de retardos internos. Según aumenta la complejidad del diseño y, por tanto, los recursos lógicos necesarios, los caminos internos para las señales son más largos y están formados por más segmentos. En esta situación, a las herramientas de emplazamiento y rutado les es más difícil encontrar una solución que cumpla con los requisitos de tiempo establecidos. Para el caso de los diseños realizados según los preceptos del diseño síncrono (como son todos los presentados en la validación de esta tesis), estos requisitos se resumen generalmente en la especificación de la frecuencia de funcionamiento del reloj global del sistema. Por tanto, con un mismo dispositivo se obtienen diseños con una máxima frecuencia de funcionamiento diferente dependiendo de su complejidad.

La figura 17.4 representa las máximas frecuencias[3] para el reloj global de las plataformas `AMT` con un único canal. En este caso, las plataformas MCSoPC tienen la infraestructura `Tornado` incluida, lo que limita la frecuencia máxima de funcionamiento a 92 MHz. Las versiones MSoPC, sin `Tornado`, ofrecen velocidades de funcionamiento más elevadas en las versiones de hasta cuatro modulaciones.

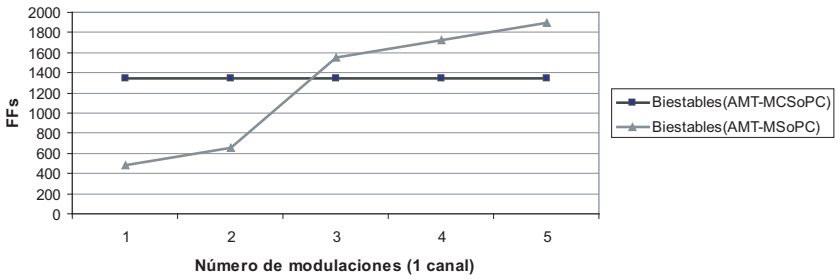
En cuanto a los retardos producidos por el proceso de auto-reconfiguración cabe destacar que, a excepción de los tiempos de carga de los *bitstream* parciales, que dependen del tamaño de éstos, son similares a los obtenidos en la plataforma `Video-Multimaster`. En la tabla

[2] Se debe indicar que se está realizando una simplificación al no tener en cuenta en la infraestructura `Tornado` el sub-sistema exterior de carga y almacenamiento de *bitstreams*. Éste puede variar en el caso de que se necesiten gestionar más *bitstreams* parciales, por ejemplo, requiriendo una FLASH mayor o más líneas de direcciones.

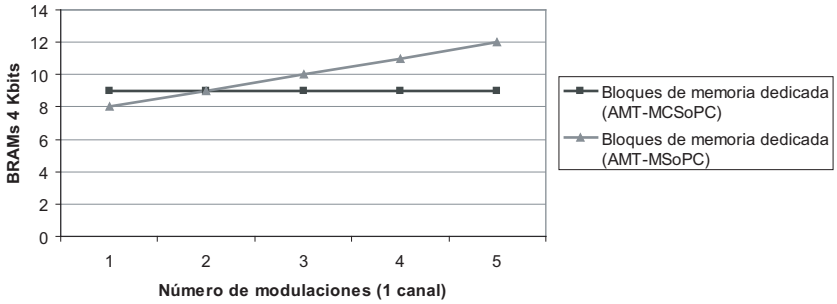
[3] El circuito Cordic de los `DDS-TXi` tiene un reloj de frecuencia doble que la del reloj global, por lo que los valores de las frecuencias máximas que se pueden sintetizar para los canales de RF se obtienen multiplicando los valores representados por dos.



(a) ***Look Up Tables de 4 entradas.***



(b) ***Biestables.***



(c) ***Bloques de memoria RAM dedicada.***

Figura 17.3.: Comparativa de la utilización de recursos lógicos entre plataformas `AMT` de un único canal con `Tornado` y sin `Tornado`.

17.3 se muestran los datos de los parámetros temporales normalizados para esta plataforma. Puesto que ésta tiene un bus de iguales características que la anterior y se utiliza la misma infraestructura `Tornado`, el tiempo normalizado de la palabra de configuración (**Tcrw0**) es el mismo. Sin embargo, el tiempo normalizado de activación de la señal `STB_RECONF` (**Tstb0**) cambia al verse modificado el tamaño del *bitstream* parcial. En este caso es algo mayor que en la situación de reconfiguración *intra-task* únicamente software, pero sigue siendo suficientemente reducido como para permitir una carga rápida del mismo.

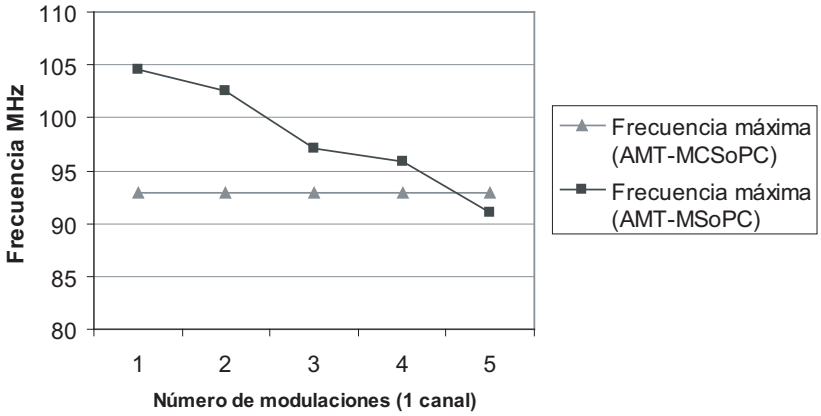


Figura 17.4.: Evolución de la frecuencia máxima de funcionamiento para el conjunto de plataformas `AMT` de un canal.

Tabla 17.3.: Valores normalizados de los parámetros temporales para la plataforma `AMT`

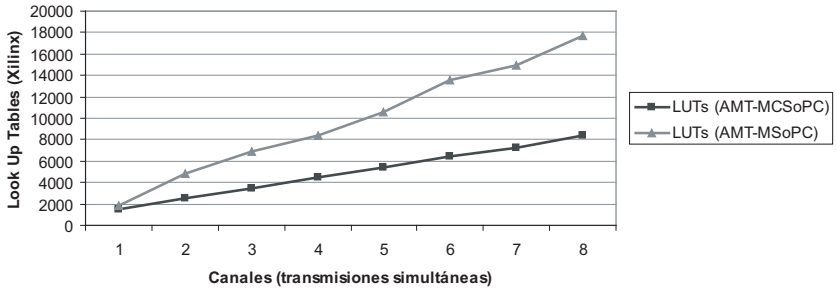
Parámetro	Ciclos de reloj	Tiempo en el prototipo
Twcrw0	37	0,74 μs
Tacrw0	134	2,68 μs
Tcrw0	171	3,42 μs
$\mathbf{Tpr_{ret0}}$	1	20 ns
$\mathbf{Tpr_{app}}$	39.492 clk	789,84 μs
Tstb0	39.493 clk	789,86 μs

17.4.2. Idoneidad de la auto-reconfiguración en función del número de canales simultáneos.

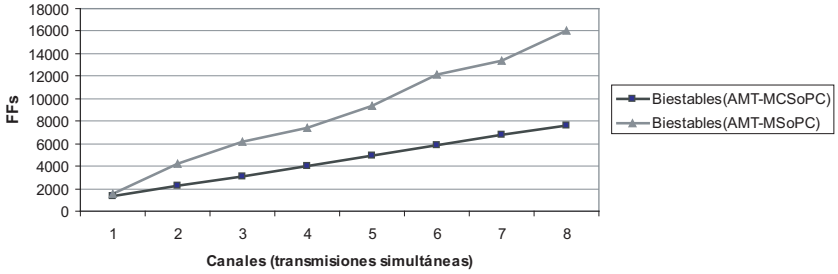
La comparación en este caso también se realiza entre dos grupos de implementaciones de la plataforma `AMT`, un grupo con versiones convencionales y otro con versiones con `Tornado`. Sin embargo, el número de modulaciones para cada canal se mantiene constante (tres tipos) variándose el número de transmisiones simultáneas. Las figuras 17.5(a), 17.5(b) y 17.5(c) muestran la evolución de los recursos lógicos LUTS, biestables y bloques de memoria RAM dedicada de 4 Kbits respectivamente para los dos grupos de implementaciones.

En el análisis anterior se ha verificado cómo, a partir de un canal con tres modulaciones diferentes, es competitiva, en cuanto a recursos lógicos, la opción de la plataformas con auto-reconfiguración. En estas gráficas se constata una evolución lineal de los bloques de memoria RAM y prácticamente lineal de los biestables y LUTs necesarios. Sin embargo, las pendientes de ambas soluciones son diferentes, acusándose una clara ventaja para las plataformas con auto-reconfiguración a medida que se incrementa el número de canales implementados.

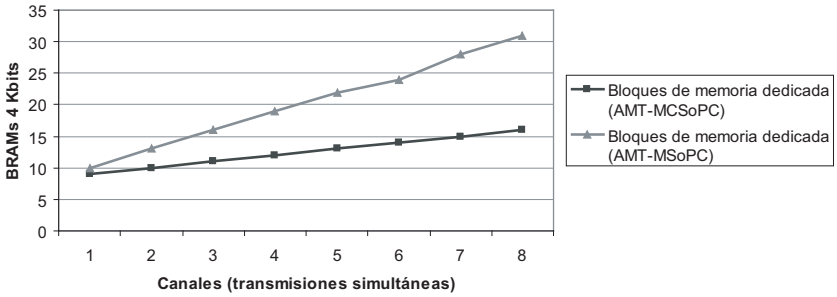
Los recursos combinacionales, representados por los recursos LUTs, en las plataformas



(a) ***Look Up Tables de 4 entradas.***



(b) ***Biestables.***



(c) ***Bloques de memoria RAM dedicada.***

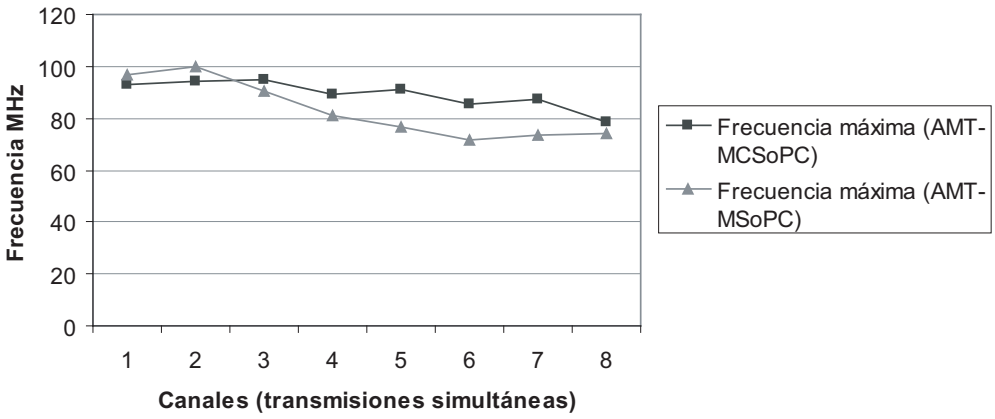
Figura 17.5.: Comparativa de la utilización de recursos lógicos entre plataformas `AMT` MCSoPC y MSoPC de distinto número de canales y tres tipos de modulación.

MCSoPC tienen también una progresión lineal. Pero en las versiones convencionales el crecimiento de número de LUTs requeridos tiene una pendiente más acusada, básicamente por el aumento no lineal de la complejidad de los multiplexores que controlan la selección de las señales de salida para el módulo activo con la modulación deseada. Destacar que el punto inicial de todas las gráficas coincide con la situación analizada en el apartado anterior de un único canal y tres modulaciones diferentes.

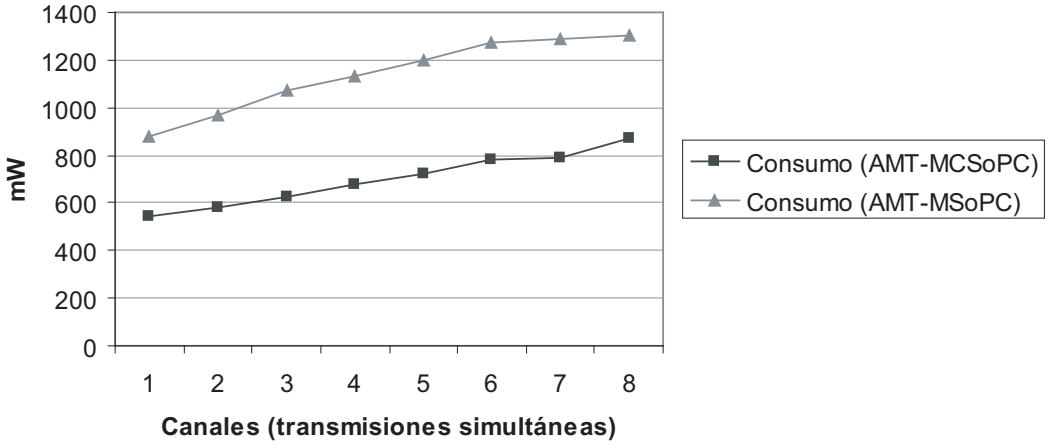
En la figura 17.6(a) se muestran las gráficas de la evolución de las frecuencias máximas

de funcionamiento para el reloj global de las diferentes plataformas. Para un número de canales mayor que cuatro, se obtienen mejores resultados con plataformas MCSoPC. Sin embargo, con porcentajes de ocupación del dispositivo altos, se produce una cierta saturación convergiendo hacia valores similares de frecuencia máxima.

A modo orientativo, se muestran en las gráficas de la figura 17.6(b) unas estimaciones de los consumos para las distintas plataformas. El motivo por el que se consideran *orientativas* se debe a que en ellas no se ha incluido el consumo de los procesos de reconfiguración y, además, se ha empleado la herramienta de estimación de consumos XPower que provee el fabricante. La frecuencia de funcionamiento fijada para los ensayos es de 50 MHz. Las versiones no reconfigurables ofrecen valores de consumo más elevados para todas las versiones de la plataforma al tener todos los circuitos cargados en la FPGA desde el principio. Aunque haya módulos que no se estén utilizando, consumen prácticamente lo mismo ya que integran secciones de control (procesadores) que continúan activas.



(a) ***Frecuencias máximas de funcionamiento.***



(b) ***Consumos en situación estática.***

Figura 17.6.: Plataformas `AMT` MCSoPC y MSoPC con distintos número de canales y tres tipos de modulaciones.

18. Plataforma de verificación avanzada II: IP-Cores Bajo Demanda

18.1. Descripción general

Tal y como se ha presentado en el estado del arte de esta tesis, el poder realizar reconfiguración *inter-task* gestionada desde el interior de los propios integrados es uno de los objetivos más perseguidos por las investigaciones en esta área.

Sin embargo, el nivel tecnológico actual de los dispositivos y en especial, el de las herramientas disponibles para el diseño con este modo de configuración es limitado. Los principales problemas que surgen al trabajar con este modo de reconfiguración son:

- **Tamaños de *bitstreams* parciales grandes.** Puesto que la reconfiguración *inter-task* implica la reconfiguración de áreas de la FPGA completas (cambio completo del *core*), los *bitstreams* parciales tienen la información de todos los recursos lógicos, rutado y pines asociados a esa área. Esto implica tamaños elevados, lo cual, unido a la utilización de sistemas de carga relativamente lentos, hace que los retardos introducidos por los procesos de carga de *bitstreams* parciales sean excesivamente largos. Esta situación repercute severamente en el funcionamiento del sistema.

- **Problemática con el rutado de las señales que unen zonas estáticas y dinámicas.** En el capítulo 12 se trata esta problemática. Se resume en la necesidad de disponer de un mecanismo que permita mantener invariables las rutas de las señales que unen una sección estática y una dinámica. De forma general, a estos elementos se denominan *Bus-Macro* y en particular a las compatibles con el sistema de control propuesto en esta tesis, `Tornado Bus-Macro` (`TBM`).

- **Necesidad de utilizar flujos de diseño específicos.** Dependiendo del fabricante se proponen flujos de diseño especiales para poder diseñar utilizando reconfiguración *inter-task*. Básicamente, en este caso, se necesita que los módulos estáticos y dinámicos se encuentren pre-emplazados y pre-rutados para ocupar una área delimitada de la FPGA. En el caso de utilizar el sistema de desarrollo ISE de Xilinx, como en este diseño, se debe seguir una metodología de diseño denominada *Module Based* [98].

El flujo de diseño definido en esta metodología es diferente al seguido para diseños no reconfigurables dinámicamente. Está orientado a facilitar que el diseñador pueda realizar un sistema que cumpla con todos los requisitos que la tecnología del fabricante impone para poder utilizar la reconfiguración *inter-task*. Estos condicionantes se detallan para los dispositivos Virtex en el apartado 13.2. Se resumen a continuación los pasos del flujo de diseño de la metodología *Module Based*:

- **Etapa 1: Descripción HDL del diseño y establecimiento de las restricciones.** Los diferentes módulos se describen en lenguaje VHDL incluyéndose un fichero de descripción de alto nivel (*Top*) para cada una de las posibles combinaciones de módulo estático y módulo dinámico. Estas descripciones no incluyen lógica, sino únicamente la instanciación de los módulos y de las *Bus-Macros*. Se debe crear un fichero de restricciones único para todos los ficheros *Top* donde se delimiten las áreas para los distintos módulos y se especifiquen los aspectos relativos a la lógica global (localización de los *pines*, reloj global, etc.). Las herramientas del fabricante utilizan estos ficheros *Top* para comprobar si se cumplen todos los requisitos del diseño modular.

- **Etapa 2: Síntesis.** Se realizan las síntesis de las descripciones VHDL:

  - De cada módulo dinámico por separado.
  - De todos los módulos no reconfigurables de forma conjunta (agrupados como un único módulo estático).
  - De cada fichero de descripción de alto nivel (*Top*).

- **Etapa 3: Implementación activa.** Mediante las herramientas del fabricante se realiza la asignación de recursos de la FPGA, emplazamiento y rutado para cada uno de los módulos (para el módulo estático y para cada posible módulo dinámico). Puesto que se tiene en cuenta en esta fase el fichero de restricciones previamente creado, los módulos emplazados y rutados quedan restringidos al área que se haya determinado en cada caso. Los *bitstreams* parciales para los módulos dinámicos se crean tras esta fase.

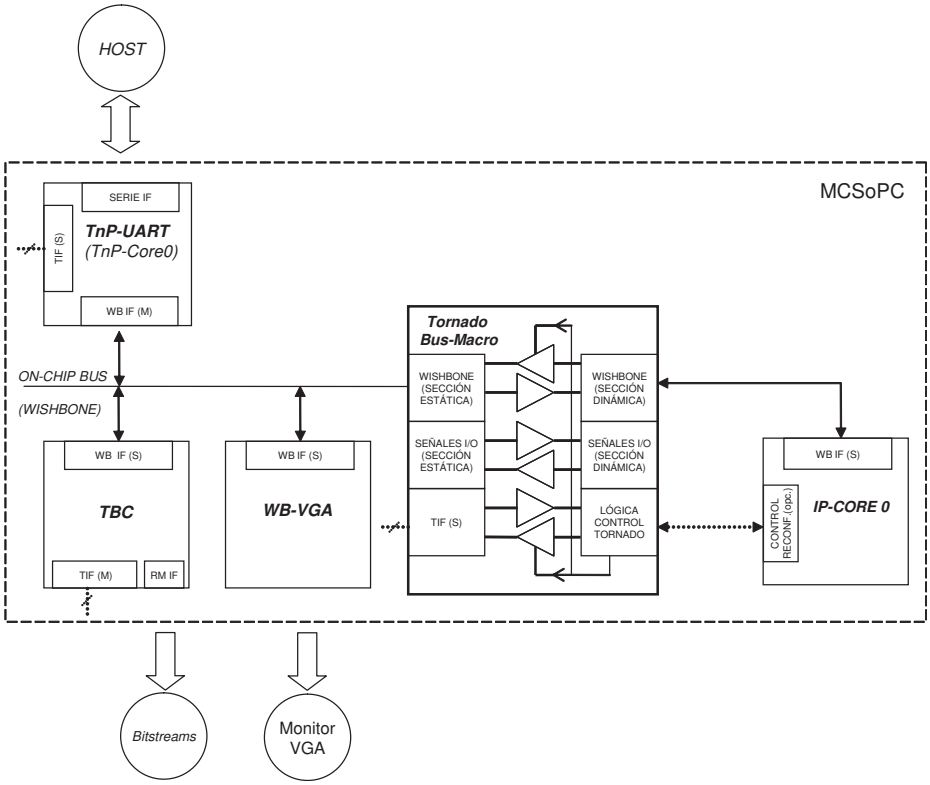
- **Etapa 4: Fase de ensamblado.** Combinando el módulo estático y cada posible módulo dinámico se obtienen los distintos diseños completos de la FPGA. Obligatoriamente se necesita al menos una de estas combinaciones de forma que se pueda crear un *bitstream* global para la carga de la FPGA tras el encendido, pero se recomienda componer todas las posibles combinaciones con los propósitos de simulación y verificación.

El diseño presentado en este apartado se realiza utilizando este flujo de diseño e incorpora la infraestructura `Tornado` para poder realizar el control de la auto-reconfiguración con cambio completo de *cores* mediante el sistema propuesto.

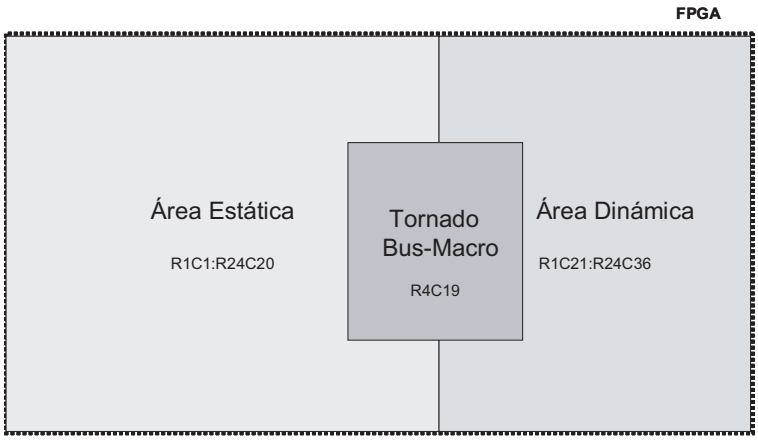
18.2. Diagrama de bloques

La plataforma `IP-Cores Bajo Demanda` es un sistema basado en *cores*, para cuyo diseño se ha utilizado la especificación Wishbone se ha integrado la infraestructura `Tornado`. En la figura 18.1(a) se muestran los bloques que componen esta plataforma. En ésta se distinguen dos zonas: un área estática, que comprende a todos los elementos situados a la izquierda de la `TBM` y un área dinámica localizada a la derecha. En la figura 18.1(b) se detallan las dos áreas definidas. Los recursos lógicos en la FPGA están distribuidos de forma regular formando una matriz cuyas posiciones se identifican en la nomenclatura de Xilinx con una "R" indicando la fila (*row*) y una "C" para referenciar la columna (*column*). Para la FPGA

utilizada (Spartan-II XC2S150), el número de filas es de 24 y el número de columnas de 36, iniciándose la numeración desde la esquina superior izquierda (R1:C1). El área estática incluye los recursos lógicos de todas las filas desde la columna 1 hasta la 24 (R1C1:R24C20) y la dinámica todos los restantes (de la columna 21 hasta la 36, R1C21:R24C36). La `TBM` se sitúa entre ambas áreas para realizar la conexión entre estas.



(a) *Diagrama de bloques.*



(b) *Grupos de área definidos en la FPGA.*

Figura 18.1.: Plataforma experimental `IP-Cores bajo demanda`.

Los módulos dinámicos (IP-Cores) se cargan en esta última área, asignada al `IP-Core0`. Este diseño admite únicamente un circuito dinámico por cada contexto. Éstos no tienen porque ser necesariamente `T-Cores`, ya que la interfaz `Tornado` esclava está integrada dentro de la `TBM`. El planteamiento teórico de este elemento (apartado 12.1) define a esta `TBM` como un *wrapper* para compatibilizar IP-Cores con `Tornado` para este tipo de reconfiguración.

Los *cores* dinámicos utilizados son dos: un `TnP-Core` similar al `RSSI-LD` empleado en la plataforma anterior y un IP-Core multiplicador que realiza multiplicaciones por hardware. A pesar de que son *cores* totalmente diferentes (tamaño, arquitectura, funcionalidad, etc.), al disponer ambos de una interfaz común para la conexión al bus *on-chip*, la tarea de intercambio de los mismos se va a posibilitar mediante la inserción de una `TBM` capaz de gestionar esas interfaces.

En el área estática se agrupan los circuitos que facilitan la experimentación con esta plataforma: el controlador de reconfiguración `TBC` y la `TBM` que componen la infraestructura `Tornado`; el IP-Core `WB-VGA` presentado en la plataformas anteriores y que permite mostrar texto en un monitor VGA; y el *core* `TnP-UART`, que como su nombre indica, integra un `nano-procesador`, es compatible con `Tornado` para reconfiguraciones *intra-task hardware/-software* e incluye un circuito UART para trasmisiones serie optimizado para la tecnología de Xilinx.

Utilizando el `TnP-UART` se pueden enviar palabras `CRW` desde un equipo externo (*host*) al `TBC` para que inicie la secuencia de reconfiguración. Además, este *core* permite verificar que la reconfiguración *inter-task* se ha realizado con éxito escribiendo y leyendo en las direcciones asignadas a los registros del IP-Core dinámico que se haya cargado en `IP-Core0`.

Se resumen a continuación las características más relevantes de los módulos que componen el sistema.

18.3. Tornado Bus-Macro

Se ha diseñado una `TBM` específica para la familia de dispositivos Spartan-II. Cumple con los requisitos especificados en la apartado 12.1 y permite envolver[1] IP-Cores compatibles Wishbone que dispongan de una interfaz con las siguientes características:

- Interfaz de bus esclava.
- Bus de datos de 16 bits.
- Un bit de direcciones para registros internos.

La figura 18.2 muestra los bloques internos de la *Bus-Macro* diseñada. La conexión interna entre los bloques de cada una de las dos áreas (estática y dinámica) se realiza utilizando *buffers* triestado denominados en la tecnología de Xilinx *Tbufs*.

Para cada señal se utilizan dos *buffers* triestado, uno en la parte estática y otro en la dinámica. La ruta entre estos dos *buffers* se elige de tal forma que sea fija y, además, que utilice una línea de conexión horizontal en la zona que limita las dos secciones. De esta forma, restringiendo el rutado para cada área a las dos zonas establecidas, los únicos nexos de unión son las señales horizonales pre-rutadas de la `TBM`. En todos los *cores* dinámicos

[1] Puesto que la `TBM` actúa como un *wrapper* para compatibilizar *cores*, se utiliza la traducción literal de *wrapp* para describir esta acción.

existen esas rutas que, al ser reconfiguradas con el mismo valor, no son alteradas. En el caso de que sean señales de entrada para el *core* dinámico siempre se mantienen activos los *buffers* aunque se esté realizando la reconfiguración puesto que éstas no afectan al funcionamiento del resto del sistema. Sin embargo, de las señales de salida del *core* dinámico, se controlan los *buffers* triestado que enlazan éstas con la sección estática, de forma que las desconecten (estado de alta impedancia) durante el proceso de reconfiguración.

Este control, gestionado según los protocolos definidos en `Tornado`, lo realiza la lógica integrada en la propia `TBM`. En la figura 18.2 se distinguen los siguientes bloques de la misma:

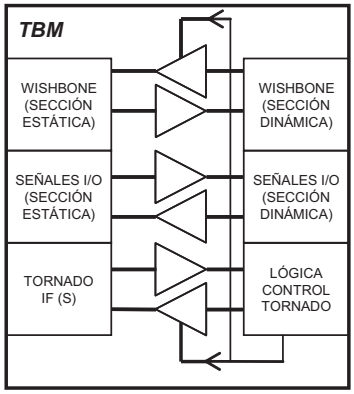


Figura 18.2.: `TBM` para Spartan-II y Wishbone.

- **Wishbone (sección estática y sección dinámica).** Al bloque de la sección estática se conectan las señales del bus *on-chip* Wishbone (situado en el área no reconfigurable). Desde estas conexiones hasta el bloque de la sección dinámica, se sitúan la rutas pre-establecidas y los *buffers* triestado. Al bloque de la sección dinámica se conecta la interfaz esclava Wishbone del *core* dinámico.
- **Señales I/O (sección estática y sección dinámica).** Los *bitstreams* parciales de cada área contienen también la configuración de los *pines* situados en la misma. Si el prototipo tiene ya fijados externamente (por rutado de la placa u otras razones) un *pin* controlado en la zona estática pero correspondiente a la zona dinámica, habrá que pasar esta señal también a través de la `TBM` de forma que no se pierda la conexión al cargar otro *core* dinámico distinto. Como esta situación también se puede producir a la inversa, se ha provisto a la `TBM` de señales en ambos sentidos para realizar esta función. En la `TBM` desarrollada se admiten cuatro señales en cada dirección.
- **Tornado IF (S).** Las señales que gestionan la auto-reconfiguración conectan al controlador de reconfiguración `TBC` a la `TBM` a través de esta interfaz. Desde el punto de vista del `TBC` se considera a la `TBM` como un `T-Core` más, teniendo una única señal `STB_RECONF` asignada. De esta forma, el `TBC` no varía[2] respecto al utilizado en las otras plataformas.

[2]El software del `TBC` sí deberá tener en cuenta la diferencia de tamaños de los *bitstreams*, ya que en este caso son mucho mayores que los obtenidos para la reconfiguración *intra-task*.

- **Lógica de control `Tornado`.** La gestión del protocolo de configuración entre el IP-Core dinámico y el `TBC` se realiza a través de esta lógica de control integrada en la `TBM`. Tal y como se ha definido en el apartado 12.1, esta sección debe estar compuesta por una máquina de estados y un elemento de almacenamiento (un biestable en esta implementación) que indique el permiso para reconfiguración, fijado por el *core* de la sección dinámica.

El método de diseño de estos circuitos pre-emplazados y pre-rutados es manual y se realiza empleando herramientas gráficas específicas. Por ello, este sistema limita la complejidad de estos circuitos y, los ejemplos que provee el fabricante son de una simplicidad tal, que no cubren los requisitos especificados para la `TBM`.

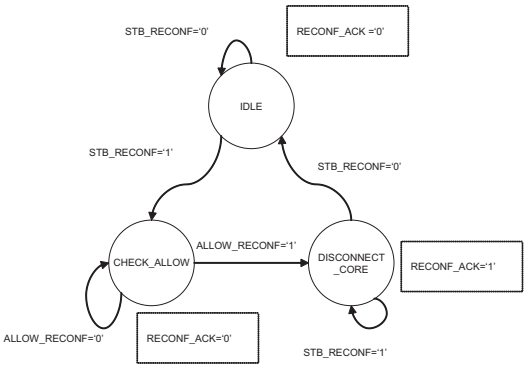
La `TBM` es un circuito pre-emplazado y pre-rutado. Sin embargo, es un diseño relativamente complejo. Permite la conexión de una interfaz de bus de 16 bits y, además, incluye lógica de control. La cantidad de componentes y rutas necesarias dificulta en exceso un diseño completamente manual. Por este motivo, el autor ha definido un nuevo flujo de diseño mixto para la creación de la `TBM`. A continuación se resumen las etapas que lo componen:

- **Fase HDL:** El circuito se describe en lenguaje VHDL. Para ello se instancian los *buffers* triestado y los elementos necesarios para la lógica de control integrada. Además, mediante el uso de los atributos que disponen estos elementos en la librería del fabricante, se especifica en el propio código la posición concreta en la FPGA para cada uno de ellos. Estas posiciones se eligen de forma que queden claramente separadas las secciones estáticas y dinámicas, facilitando a la herramienta de rutado la conexión de ambas mediante segmentos horizontales.

  La localización de la lógica de control también se debe especificar. Por ello,debe quedar restringida a unos pocos elementos para los que sea posible indicar una determinada posición mediante atributos VHDL. La lógica combinacional y los dos biestables de la máquina de estados han sido integrado en una única *slice*. Para ello se ha definido a bajo nivel el contenido exacto de las dos LUTs que realizan las operaciones combinacionales, indicando su contenido también mediante atributos en esta fase del diseño. El biestable utilizado para almacenar la información de la admisión de la reconfiguración se ha colocado en otra *slice* contigua. En la figura 18.3(b) se representa la tabla de verdad de la máquina de estados del control de la reconfiguración: `s0` y `s1` son los registros de estado; `init_lut0` e `init_lut1` los valores de los atributos de inicialización para las dos *LUTs* utilizadas; `allow_reconf` y `stb_reconf` las dos señales que junto con el valor de los registros de estado establecen el estado siguiente (figura 18.3(a)). El control de los *buffers* triestado se realiza mediante la señal `reconf_ack`, cuyo valor lógico coincide con el almacenado en el registro `s1`. En la figura 18.4(c) se detalla la localización de las *slices* que integran estos elementos.

  El código HDL de la `TBM` junto con todas las instanciaciones y atributos se sintetiza, emplaza y ruta automáticamente con las herramientas del fabricante. Esto implica que se debe especificar una FPGA concreta para esta fase, aunque el circuito generado sea válido para todos los dispositivos de la familia. En este caso, la FPGA seleccionada es de la familia Spartan-II.

- **Fase manual con el editor gráfico:** El diseño emplazado y rutado en la fase anterior se edita con la herramienta del fabricante que permite visualizar y modificar éste de



(a) ***Diagrama de burbujas de la máquina de estados.***

SALIDAS: RECONF_ACK <= S1
LÓGICA ESTADO SIGUIENTE:

S1(I3)	S0(I2)	ALLOW_RECONF(I1)	STB_RECONF(I0)	S1_next –INIT LUT1 (0-15)	S0_next-INIT LUT0 (0-15)
0	0	0	0	0 (0x0)	0 (0xA)
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0 (0xC)	1(0x3)
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0(0xA)	0(0x0)
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0(0x0)	0(0x0)
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

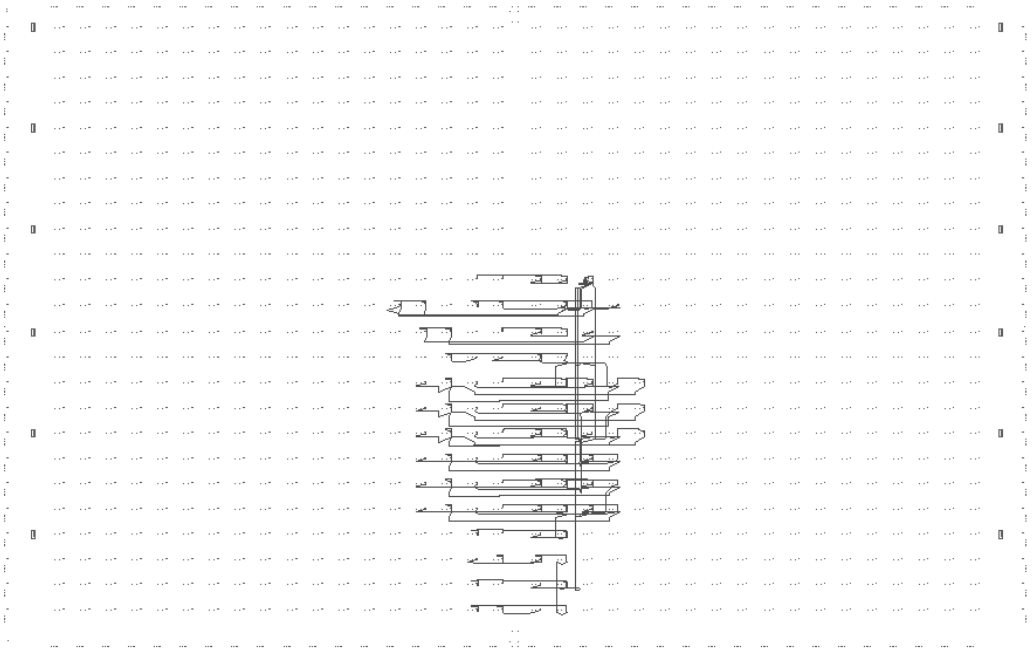
(b) ***Tabla de verdad de la máquina de estados y generación de los atributos para inicialización de las LUTs.***

Figura 18.3.: Lógica de control integrada en la TBM

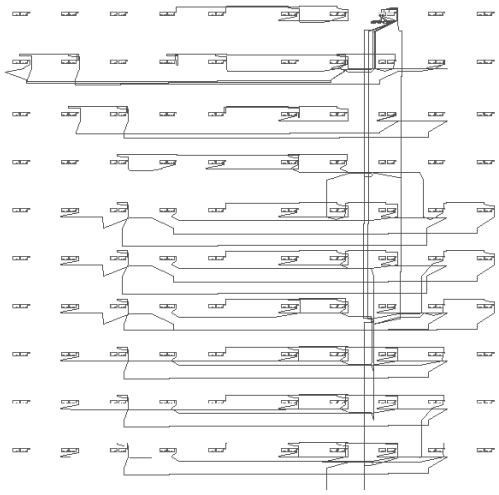
forma gráfica. La primera tarea consiste en convertir este circuito realizado para una determinada FPGA en una *Bus-Macro* que se pueda utilizar en otros diseños y con otras FPGAs de la misma familia. Para ello, se eliminan las rutas que en el proceso de rutado automático se hayan generado entre los *Tbufs* y los bloques de entrada y salida (IOBs) asignados a los *pines* de la FPGA. Esas rutas son las de las señales que en la plataforma dinámica conectan a la TBM con las secciones estáticas y dinámicas. Por tanto, deben eliminarse del circuito pre-diseñado de la TBM. A los puertos de los *Tbufs* y de las dos *slices* a las que estaban conectadas estas señales, se les añaden los *pines* externos de la TBM. Estos *pines* componen la entidad VHDL de la TBM. En los procesos de rutado de los diferentes módulos de la plataforma reconfigurable se conectan las señales que unen las dos secciones a estos *pines*. Por último, se selecciona uno de los componentes de la *Bus-Macro* como punto de referencia para ubicar la misma en la FPGA.

La figura 18.4(a) muestra el circuito de la TBM tras realizar estos pasos. En esta fase se han especificado un conjunto de localizaciones que sitúan este circuito en el centro de la FPGA. Exactamente la mitad de la FPGA marca la separación entre las dos áreas que se pueden definir mediante esta *Bus-Macro*. Cabe destacar, que en esta zona de separación únicamente hay rutas horizontales, como se puede comprobar en la figura 18.4(b) que muestra con más detalle la composición del circuito. Las figuras 18.4(c) y 18.4(d) representan ampliaciones de la zona donde se sitúan las dos *slices* de control y de un *buffer* triestado respectivamente.

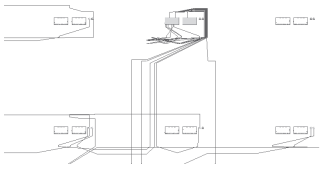
- **Instanciación en el código HDL:** La TBM generada se puede instanciar desde el código de un diseño descrito en VHDL de igual forma a como se realiza con cualquier otro componente, aunque realmente esta instanciación implica la utilización de un circuito ya rutado y con los recursos asignados. Además, mediante el componente de



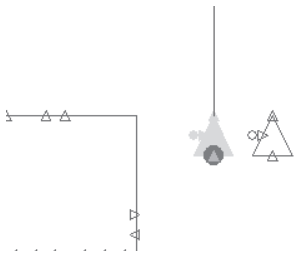
(a) ***Circuito tras la fase manual de edición.***



(b) ***Ampliación de la Bus-Macro.***



(c) ***Localización de las slices para la lógica de control.***



(d) ***Ampliación de un Tbuf con la salida rutada y un puerto externo de la Bus-Macro en la entrada.***

Figura 18.4.: Circuito interno de la `TBM`

referencia fijado en la etapa anterior, se especifica la posición para todo el circuito pre-diseñado, lo que permite variar su localización.

Utilizando este flujo de diseño, es factible la realización de un conjunto de `TBMs` válidas para interfaces Wishbone con distintas configuraciones en lo referente a la anchura del bus de datos y del bus de direcciones, más señales de control, etc.

18.4. Módulos IP dinámicos

Se presentan a continuación los dos IP-Cores que se cargan en el área dinámica, definiéndose dos contextos de funcionamiento para el sistema. En el diagrama de bloques de la plataforma se identifica de forma genérica al *core* activo en esta área como *IP-Core0*.

18.4.1. Core dinámico ADC con procesador embebido

El `WB_ADC_dynamic_core` es un `TnP-Core` que permite realizar conversiones analógico digitales. Al integrar un `nano-procesador`, se dispone de un método flexible para formatear los resultados. En la figura 18.5 se muestra la arquitectura de este *core*. Además de la interfaz Wishbone, la sección hardware integra el circuito de conversión analógico-digital sigma-delta, salvo un comparador que se sitúa en el exterior de la FPGA. Únicamente requiere tres señales para controlar la sección externa del convertidor [3].

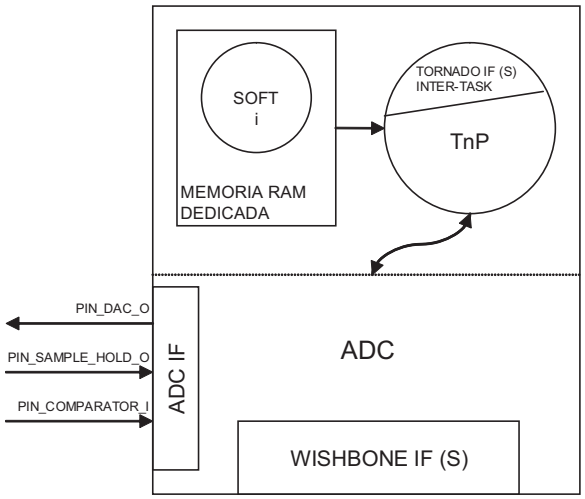


Figura 18.5.: Arquitectura del `WB_ADC_dynamic_core`.

Este *core* se carga dinámicamente en el área asignada al `IP-Core0`, lo que implica que se debe conectar a la `TBM`. Como la gestión del *handshake* de reconfiguración la realiza la propia `TBM`, la interfaz `TIF (S)` del `TnP` se ha modificado de forma que escriba la habilitación o deshabilitación de la reconfiguración en el biestable que para ese propósito tiene la `TBM`.

[3] Para más información sobre el funcionamiento de este circuito se referencia la nota de aplicación [230] de Xilinx, específica sobre la utilización de este tipo de convertidores en FPGAs.

En este caso no tiene sentido el control del estado del `nano-procesador`, puesto que se reconfigura el *core* completamente.

El flujo de diseño modular que se debe adoptar para poder realizar este tipo de diseños con reconfiguración *inter-task* obliga a que se realice el emplazamiento y rutado de cada módulo dinámico y de la sección estática por separado. En la figura 18.6 se muestra el resultado de este proceso para el módulo dinámico `WB_ADC_dynamic_core`. Cabe destacar, cómo queda restringido el emplazamiento y rutado al área establecida (de la columna 21 a la 36, la situada en el extremo derecho) quedando el circuito conectado a la `TBM`. Esta *Bus-Macro* se ha colocado en una posición superior y más desplazada a la derecha que en la empleada en el diseño original de la `TBM` (figura 18.4(a)).

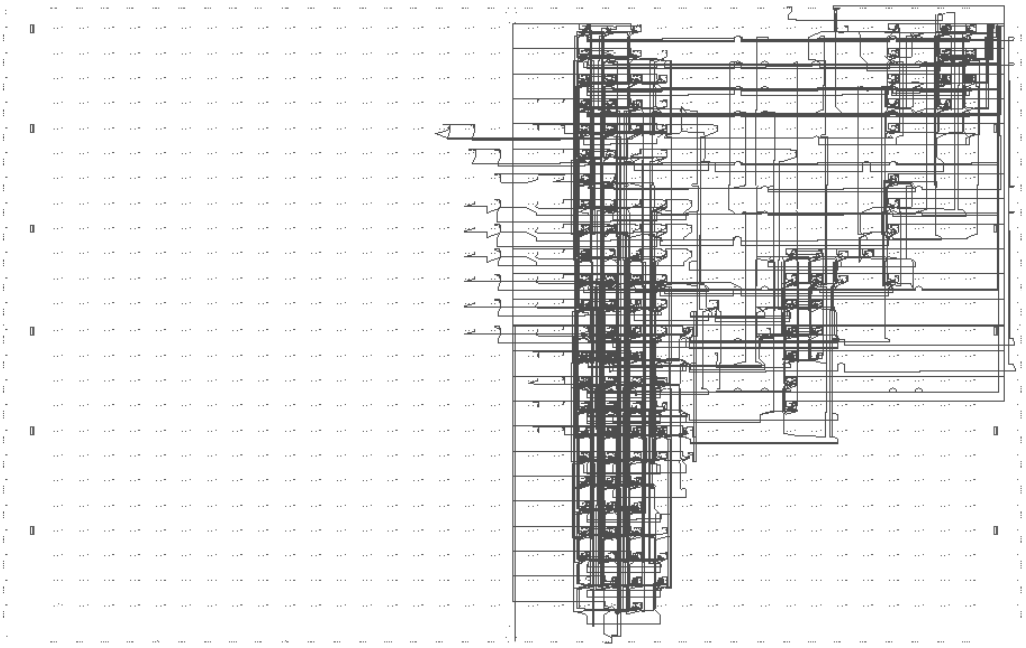


Figura 18.6.: Módulo dinámico `WB_ADC_dynamic_core` emplazado y rutado.

18.4.2. Core dinámico multiplicador por hardware

El módulo `WB_mult_dynamic_core` es un IP-Core cuya función es la de realizar multiplicaciones por hardware. La descripción del algoritmo en VHDL utilizada es original de V. V. Erohkhinde y de libre utilización (OpenCores [181]). A esta descripción se le ha añadido una interfaz Wishbone esclava para armonizar el *core* con la especificación utilizada y poderlo conectar a la `TBM`. El código del multiplicador dispone de *genéricos* VHDL para poder variar la anchura de los operandos y por tanto del circuito generado. Para esta implementación se ha creado un circuito para dos operandos de 16 bits y resultado de 32 bits.

Los recursos lógicos necesarios para implementar este *core* son mucho menores que los necesarios para el `WB_ADC_dynamic_core`. Sin embargo, el área asignada para el emplazamiento y rutado del mismo debe ser exactamente igual que la definida para éste, lo que implica

tamaños de *bistreams* iguales. En la figura 18.7 se muestran el `WB_mult_dynamic_core` y la `TBM` emplazados y rutados.

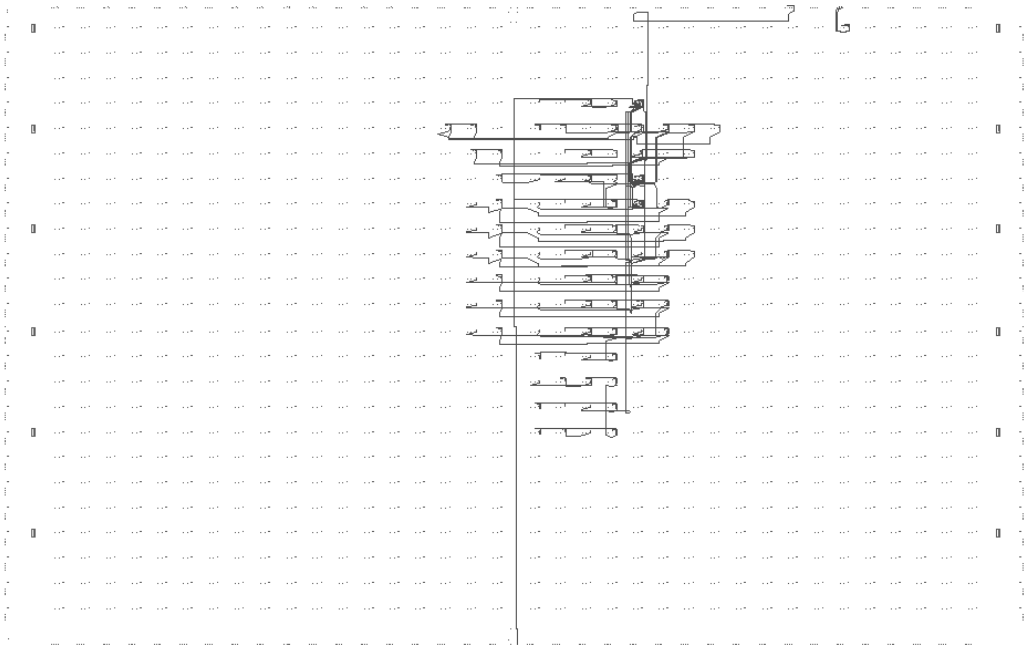


Figura 18.7.: Módulo dinámico `WB_mult_dynamic_core` emplazado y rutado.

18.5. Módulos IP estáticos

En esta plataforma experimental se utilizan módulos IP de diversa tipología en el área estática. Se incluyen *cores* que integran `nano-procesadores`, son compatibles con una especificación estándar y, en general, se utilizan en diseños SoPC comerciales. Además, el control de la auto-reconfiguración se realiza mediante `Tornado`. Estos módulos son: `TnP-UART`, `WB-VGA`, `TBC` y un decodificador de direcciones.

Todos los módulos situados en el área estática de la FPGA se sintetizan de forma conjunta. Como resultado, se obtiene un único módulo estático que se gestiona de forma similar a un módulo dinámico en lo referente a la conexión a la *Bus-Macro*, emplazamiento y rutado independiente, etc. El resultado de este procesamiento conjunto se presenta en la figura 18.8. Se puede observar cómo todos los recursos lógicos y de rutado quedan restringidos al área estática. El módulo estático en conjunto se conecta a los *Tbufs* de la parte izquierda de la `TBM`.

18.5.1. Controlador de comunicación serie inteligente: TnP-UART

El `TnP-UART` es un *core* con arquitectura de `Core Mixto` desarrollado por el autor con el propósito de disponer de acceso en lectura y escritura a cualquier módulo interno desde un equipo exterior (*host*). Forma parte del catálogo de IP-Cores utilizados en los diseños del

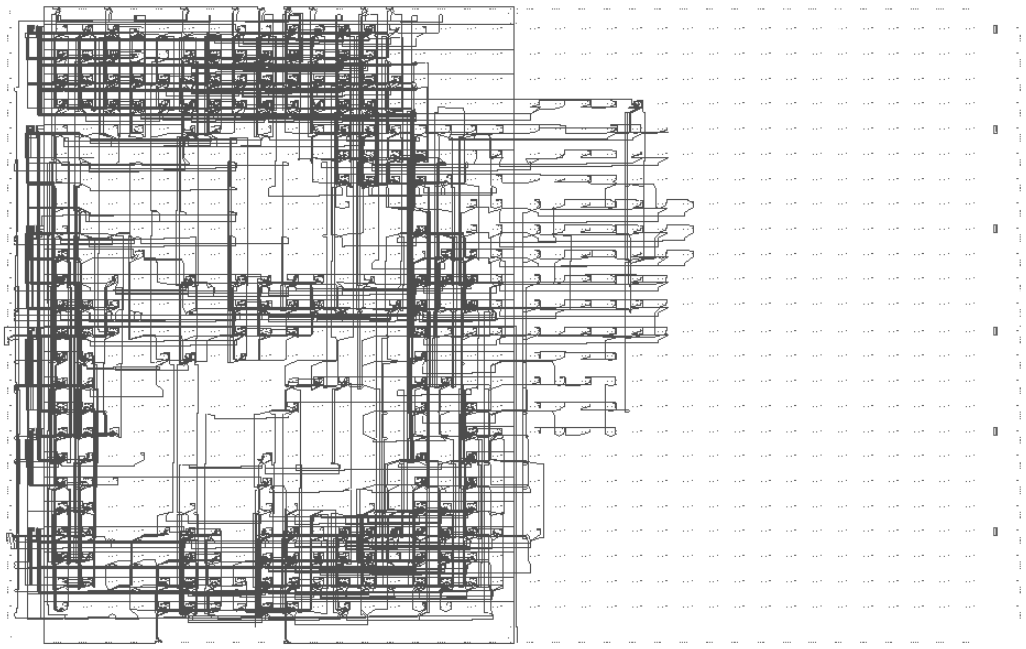


Figura 18.8.: Módulos estáticos agrupados, emplazados y rutados.

grupo de investigación dada su versatilidad para realizar tanto tareas de depuración como de control.

En la figura 18.9 se detalla la arquitectura interna del mismo. En la sección hardware se incluye una UART para comunicaciones de hasta 1 Mbit/s y en la sección de procesamiento software se incluye un `nano-procesador TnP`.

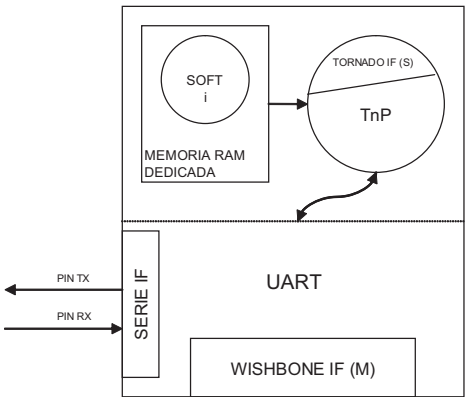


Figura 18.9.: Arquitectura del `TnP-UART`.

El programa del `TnP` gestiona los datos recibidos y transmitidos a través de la UART ofreciendo al *host* una serie de comandos de control. Éstos permiten indicar de forma remota las operaciones para el `TnP-UART` utilizando un programa terminal de comunicaciones. Para

ello, hay comandos que permiten especificar la dirección que se desea leer o escribir y, para este último caso, el dato que se vaya escribir.

El TnP incluye una interfaz Tornado esclava, que está conectada en la plataforma al TBC. De esta forma se podrán realizar en el futuro ensayos combinando la reconfiguración *inter-task* e *intra-task*.

18.5.2. Controlador de vídeo: WB-VGA

En esta plataforma se utiliza de nuevo el *core* WB-VGA presentado en el apartado 16.3.3, el cual permite mostrar en un monitor VGA caracteres de texto almacenados en su memoria interna. Esta información se puede escribir desde cualquier *core* maestro conectado al bus *on-chip*.

18.5.3. Controlador de reconfiguración: TBC

Esta implementación del controlador de reconfiguración TBC es la misma que la utilizada en el resto de plataformas. En este caso la escritura de las CRWs se realiza desde el TnP-UART, a petición del *host*.

18.6. Resultados experimentales

La creación de una *Bus-Macro* con las características expuestas ha supuesto la definición de un innovador flujo de diseño para las mismas, aunque siguen requiriendo tediosas labores manuales en diversas secciones.

Por otro lado, las herramientas del fabricante no son estables para diseños de este tipo. Tienen numerosos errores, que en muchos casos obligan a paralizar el desarrollo hasta que, en una nueva versión de las herramientas, han sido solucionados. Finalmente se ha completado el diseño de la plataforma y se ha simulado su comportamiento. A este respecto, cabe destacar, que hay un debate abierto muy activo en los foros sobre reconfiguración dinámica acerca de las características exactas que deben tener las *Bus-Macro* para esta tecnología. Por ello, uno de los trabajos futuros es la comprobación exhaustiva sobre el dispositivo de la TBM diseñada.

La implementación de la infraestructura Tornado utilizada, a excepción de la TBM, es la misma que en las plataformas anteriormente presentadas. En este caso, los valores obtenidos para los parámetros temporales normalizados se representan en la tabla 18.1. La topología del bus de la sección estática y las interfaces Wishbone son iguales. Por ello, los parámetros temporales normalizados de escritura de la palabra de configuración CRW (**Twcrw0**, **Tacrw0** y **Tcrw0**) no varían respecto a los obtenidos anteriormente.

Continuando con los parámetros temporales normalizados, en cuanto al tiempo de aceptación de la reconfiguración parcial ($\mathbf{Tpr_{ret0}}$), cabe destacar que en este caso el elemento que admite la reconfiguración en el *handshake* de control es la máquina de estados integrada en la TBM. Esta necesita dos ciclos de reloj para validar un intento de reconfiguración, por lo que el tiempo $\mathbf{Tpr_{ret0}}$ se duplica respecto al obtenido con reconfiguración *intra-task*.

La mayor diferencia se centra en el parámetro que cuantifica el tiempo de carga del *bitstream* ($\mathbf{Tpr_{app}}$). Este tiempo es proporcional al tamaño del *bitstream* parcial. Para la FPGA del prototipo, el tamaño de éste es de 38.804 bytes, lo que supone aproximadamente el

Tabla 18.1.: Valores normalizados de los parámetros temporales para la plataforma `IP-Cores bajo demanda`

Parámetro	Ciclos de reloj	Tiempo en el prototipo
Twcrw0	37	0,74 μs
Tacrw0	134	2,68 μs
Tcrw0	171	3,42 μs
$\mathbf{Tpr_{ret0}}$	2	40 ns
$\mathbf{Tpr_{app}}$	232.824	4,65 ms
Tstb0	232.826	4,65 ms

30 % del *bitstream* global. Con dispositivos mayores, se incrementa notablemente el número de bytes. Esto implica un problema adicional para la carga y almacenamiento de estos dispositivos.

Con la velocidad del sistema de carga de *bitstreams* del prototipo, se obtienen unos tiempos superiores a 4.6 ms. Estos retardos son sustancialmente mayores que los obtenidos con la reconfiguración *intra-task*, pudiendo ser inadmisibles para ciertas aplicaciones. Sin embargo, teniendo en cuenta que las auto-reconfiguraciones se realizan de una manera controlada y, que el resto del sistema sigue funcionando, este modo de reconfiguración abre nuevas posibilidades para que los diseñadores e investigadores desarrollen sistemas adaptativos y enfoquen problemas convencionales desde otro punto de vista.

Parte IV.

Conclusiones y trabajo futuro

19. Conclusiones

En el estado del arte de esta tesis se han presentado los diseños basados en *cores* utilizando dispositivos reconfigurables, los SoPC, como una de las opciones más extendidas para la integración de sistemas digitales en un único dispositivo.

Teniendo en cuenta que algunos de estos dispositivos reconfigurables admiten reconfiguración dinámica y parcial, las últimas investigaciones en Computación Reconfigurable proponen sistemas que permiten controlar la aplicación de la auto-reconfiguración parcial dinámica sobre diseños basados en IP-Cores. Estas propuestas, que dotan a los SoPC de *configurabilidad* (CSoPC), sufren de ciertas carencias como son la falta de soporte para la reconfiguración de *cores* con procesadores embebidos o su inadecuación a las especificaciones estandarizadas para interconexión de IP-Cores que se emplean de forma generalizada en el diseño de SoPCs.

El trabajo reflejado en esta tesis parte del objetivo general de definir y validar una solución global para controlar la auto-reconfiguración parcial dinámica en diseños SoPC para aplicaciones industriales y con *cores* con procesadores embebidos.

Para ello, se ha partido de un modelo reconfigurable basado en *cores* interconectados siguiendo una especificación estándar. A este modelo se le ha incorporado una infraestructura de control compuesta por un controlador de reconfiguración, señales, protocolos, procesadores e interfaces específicos.

Los criterios generales empleados para la definición de todos esos elementos han sido: crear una infraestructura sencilla en cuanto a los recursos lógicos necesarios para la misma y mínimamente intrusiva en el diseño, lo más general posible y orientada a las aplicaciones basadas en IP-Cores; distribuir la *metacomputación* entre diferentes *cores* del sistema; y soportar la reconfiguración *intra-task hardware/software* de *cores* con pequeños procesadores (`nano-procesadores`) embebidos.

Este sistema planteado de forma general y teórica, denominado `Tornado`, se ha validado sobre una tecnología concreta de dispositivos reconfigurables. Para ello se han diseñado tres plataformas basadas en IP-Cores, de distinta complejidad y orientadas cada una de ellas a experimentar aspectos concretos de `Tornado`. La infraestructura incluida en las tres plataformas es una implementación de los elementos definidos en `Tornado` para los dispositivos Virtex de Xilinx y compatibles con la especificación estándar Wishbone. En concreto, se incluye un controlador de reconfiguración, un `nano-procesador` compatible con `Tornado` optimizado para ser embebido en *cores* y un programa ensamblador para éste. De forma adicional, la última plataforma ha requerido de una *Bus-Macro* experimental compatible con `Tornado`.

Mediante estas plataformas, además de probar la validez del sistema de control, se ha podido cuantificar el coste de la utilización de la reconfiguración parcial dinámica con esta infraestructura. Los parámetros definidos para caracterizar esta infraestructura permiten contrastar tanto el coste en recursos lógicos como las penalizaciones temporales sufridas por el proceso de auto-reconfiguración.

Los ensayos se han realizado sobre un prototipo especialmente diseñado para experimentar tanto con la reconfiguración como con la auto-reconfiguración parcial dinámica. En la primera, se indican externamente los cambios de contexto aplicables y en la segunda se realiza la *metacomputación* en el propio dispositivo que se reconfigura. Para ello, se ha dotado al prototipo de un mecanismo que permite realizar un control total sobre el puerto de configuración de la FPGA y de un sistema versátil para el almacenamiento de los *bitstreams*.

Los datos obtenidos en las simulaciones y en los ensayos realizados con los prototipos permiten concluir que `Tornado` es un sistema viable que permite dotar a diseños complejos de un nuevo grado de flexibilidad, que es la auto-reconfiguración. Estos datos revelan que es especialmente viable y práctica la reconfiguración *intra-task* de *cores* con arquitectura de `Core Mixto`. Se trata de un modo de reconfiguración sencillo de aplicar y muy rápido debido al reducido tamaño de los *bitstreams* asociados.

Los resultados del coste en recursos lógicos de la infraestructura implementada para las plataformas revelan que se consigue realmente una solución sencilla y poco intrusiva, básicamente obtenida gracias la distribución de la *metacomputación* entre distintos *cores*. Al no centralizarse la *metacomputación* en un único complejo controlador de reconfiguración, se ha podido aplicar la arquitectura simple de `Core Mixto` al controlador de reconfiguración diseñado, obteniéndose excelentes resultados que destacan el grado de optimización obtenida.

El trabajo con el prototipo y con las herramientas de diseño del fabricante de lógica reconfigurable ha puesto de manifiesto las limitaciones actuales respecto a la reconfiguración parcial dinámica de la tecnología empleada: básicamente la obligatoriedad de definir las unidades reconfigurables en una única dimensión (columnas) y la imposibilidad de controlar el rutado de las secciones reconfigurables. Además, las opciones específicas de las herramientas de diseño para soportar la reconfiguración parcial dinámica han mostrado una falta de robustez notoria. Actualmente, los diseños parcialmente reconfigurables, son considerados por el fabricante como experimentales. Por ello, las herramientas se encuentran incompletas y sujetas a modificaciones importantes entre las diferentes versiones del software.

Estos experimentos permiten concluir que, si bien la reconfiguración *intra-task* es asequible y estable (añadiendo al flujo de diseño del fabricante herramientas propias), la utilización de la reconfiguración *inter-task* con esta tecnología fuerza a complicar en exceso el flujo de diseño sin garantías por parte del fabricante de que sus herramientas lo soporten.

19.1. Principales aportaciones

En esta sección se resumen las principales aportaciones de esta tesis. Están ordenadas siguiendo un criterio de generalidad. De la uno a la seis son las aportaciones más teóricas, mientras que la siete y las ocho son más específicas, realizadas para una tecnología concreta:

1. **Sistema de control de la auto-reconfiguración parcial dinámica.** Se especifica un sistema completo para controlar la auto-reconfiguración parcial dinámica en diseños implementados en dispositivos reconfigurables y basados en *cores*. Se contempla la casuística de que estos *cores* incluyan pequeños procesadores, pudiendo ser objeto de reconfiguración. Este sistema se denomina `Tornado`. En él se define un modelo reconfigurable y una infraestructura de control.

2. **Modelo reconfigurable y multi-procesador** basado en *cores* y con la topología habitual de los diseños SoPC. Mediante el modelo definido se establece cómo debe ser la infraestructura que se debe añadir a un diseño SoPC para convertirlo en un sistema auto-reconfigurable. Es un modelo flexible al admitir distintas especificaciones para interconexión de *cores* e incluso diversas topologías de bus.

3. **Protocolos de control de la auto-reconfiguración.** Cabe destacar la especificación de dos protocolos en la infraestructura de control. Uno utilizado para integrar las peticiones de auto-reconfiguración en el bus *on-chip* del sistema, lo cual facilita la distribución de la *metacomputación* entre diferentes *cores*. Y otro protocolo, un *handshake* síncrono, específicamente diseñado para poder controlar la aplicación de la reconfiguración sobre *cores* que integren pequeños procesadores.

4. **Infraestructura para el control de la auto-reconfiguración.** El modelo reconfigurable define los elementos y protocolos propuestos para controlar la auto-reconfiguración. En la infraestructura se detallan de forma generalizada las características que debe tener cada uno de los módulos, de forma que se puedan realizar versiones de la misma para distintas tecnologías. Por tanto, hay una infraestructura distinta para cada tecnología: una implementación del sistema `Tornado` específica.

5. **Modelado de la infraestructura de control.** Utilizando fórmulas que integran parámetros cuyos valores se obtienen experimentalmente, se ha modelado el coste en recursos lógicos de la infraestructura de control. Los resultados obtenidos al aplicar esa fórmulas son una estimación válida para cualquier diseño multiprocesador y auto-reconfigurable que se vaya a realizar empleando la tecnología para la que se hayan obtenido los valores de los parámetros experimentales. En la fórmula final, se tienen en cuenta tanto el número de procesadores reconfigurables embebidos como la cantidad de módulos auxiliares para la reconfiguración. El dato obtenido permite estudiar la viabilidad de la auto-reconfiguración en cuanto a área de silicio, comparándolo con otras alternativas que no usen la reconfiguración, que utilicen otro tipo de dispositivos, etc.

   Mediante otro conjunto de parámetros se han identificado y parametrizado las secuencias temporales involucradas en el proceso de auto-reconfiguración propuesto. De esta forma se puede disponer de datos contrastables y fácilmente analizables. También se ha normalizado un conjunto de estos parámetros de forma que permitan comparar los resultados obtenidos en distintas soluciones (topologías de bus, arquitecturas, tecnologías, etc.) bajo las mismas condiciones de operación.

6. **Flujos de diseño general para reconfiguración *intra-task* e *inter-task*.** Se han definido dos diagramas de flujo generales que integran en el flujo de diseño habitual para los fabricantes de dispositivos reconfigurables los elementos de la infraestructura `Tornado` necesarios. Esta integración se realiza asistida por el *framework* `Tornado` específico para la tecnología utilizada.

   La principal aportación del sistema propuesto se centra en el control de la reconfiguración *intra-task* sobre sistemas con *cores* que integran procesadores. Para estos casos el *framework* incluye un programa ensamblador multi-contexto y *scripts* de automa-

tización que permiten obtener los ficheros de configuración y de inicialización de los bloques de memoria para el software de los procesadores de forma automática.

Para la reconfiguración *inter-task*, el sistema propone unos módulos compatibles con los protocolos de control especificados para poder conectar *cores* no específicamente diseñados para admitir la reconfiguración parcial dinámica.

7. **Desarrollo de una infraestructura y un *framework* concreto para una tecnología.** Siguiendo las especificaciones establecidas en la sección teórica, se ha construido una infraestructura completamente operativa para la tecnología de los dispositivos de la familia Virtex de Xilinx. Esta infraestructura incluye un controlador de reconfiguración, un `nano-procesador` para integración en *cores*, interfaces maestra y esclava para el control de la auto-reconfiguración y un circuito especial para envolver IP-Cores adaptándolos al sistema `Tornado`.

   El *framework* desarrollado incluye un programa ensamblador multi-contexto para el `nano-procesador` y *scripts* de automatización para el entorno de diseño ISE.

8. **Validación del sistema propuesto con tres plataformas.** La propuesta teórica del sistema `Tornado` se ha experimentado sobre tres diseños que incorporan la infraestructura desarrollada y hacen uso del *framework* específico para Xilinx.

   La primera plataforma, `Video-Multimaster`, ha permitido verificar la reconfiguración software de los `nano-procesadores`. La validación de las ecuaciones de modelado se ha realizado estimando primero los recursos necesarios para la infraestructura en distintas implementaciones del diseño y luego contrastando estos datos con los obtenidos realmente. Se ha concluido que son estimaciones válidas y generales al verificarse el ensayo sobre distintas configuraciones.

   La segunda plataforma, `Adaptive Multi-Transmitter`, muestra un tipo de aplicación que se beneficia notoriamente del uso de la auto-reconfiguración parcial dinámica. Especialmente con el sistema propuesto en esta tesis, que distribuye la *metacomputación* entre diferentes *cores* del sistema. Esta situación se adapta adecuadamente a una plataforma de este tipo donde las decisiones de cambio de contexto se realizan en módulos que analizan la calidad de los canales de comunicación o en otros módulos de comunicaciones. Mediante datos empíricos se demuestra la adecuación del sistema y se contrastan estos datos con los de una solución convencional.

   En la tercera plataforma, denominada `IP-Cores bajo demanda`, se aplica el sistema `Tornado` a un sistema con reconfiguración *inter-task*. Para ello, se ha diseñado una `TBM` específica para la tecnología disponible en los prototipos y se han incorporado IP-Cores obtenidos de fuentes externas compatibilizándolos con `Tornado`.

19.1.1. Herramientas complementarias desarrolladas

La validación de `Tornado` ha requerido el desarrollo de una serie herramientas que, por un lado, facilitan la tarea del diseñador y, por otro, permiten demostrar la aplicabilidad del sistema propuesto a diseños reales. En cuanto a las herramientas hardware desarrolladas caben destacar:

- **Prototipo con Spartan-II y puerto SelectMAP de reconfiguración accesible.** La carga de *bitstreams* parciales de reconfiguración requiere un prototipo que esté específicamente diseñado para permitir el control de todas las señales del puerto paralelo de configuración incorporado a los dispositivos Spartan-II y Virtex de Xilinx. Además de poder mantener el control de esas señales durante el funcionamiento, es necesario dotar al mismo de un sistema de almacenamiento y de carga de los diferentes *bitstreams* de configuración.

  Para facilitar los experimentos, también se ha incluido un microprocesador exterior a la FPGA con la capacidad de controlar la carga de *bitstreams*. De esta forma, se han podido realizar ensayos con reconfiguración parcial dinámica sencillos previos a la auto-reconfiguración que realmente se realiza con `Tornado` en las plataformas propuestas.

  Con este fin se desarrolló un prototipo [205] teniendo en cuenta todas estas necesidades, de forma que permitiese realizar los ensayos de validación de esta tesis.

- **Módulos VHDL.** Además de los módulos integrantes de la infraestructura `Tornado` (el `TBC`, el `TnP`, las interfaces específicas y la `TBM`), ha sido necesario desarrollar numerosos módulos VHDL para las plataformas de validación, los cuales pueden ser utilizados en futuros diseños. Éstos son: el arbitrador de bus parametrizable y los *cores* `TnP-Master Video`, `WB-VGA`, `WB-USB`, `DDS-TXi`, `RSSI-LD`, `WB_ADC_dynamic_core`, `WB_mult_dynamic_core`, `FFR16` y `TnP-UART`.

En cuanto al software desarrollado, destacar que se han creado programas de muy distinta naturaleza. Desde herramientas para ordenadores personales, hasta numerosos programas para los `nano-procesadores` entre otros. Se resumen a continuación los elementos software desarrollados:

- ***Tornado Reconfigware Assembler* (TRA).** Software para ordenadores PC para ensamblar y generar automáticamente los códigos VHDL de los programas de los `Cores Mixtos` compatibles con `Tornado` (`TnP-Cores`).

- ***Scripts* de automatización.** En conjunción con el `TRA`, estos *scripts* integran los ficheros VHDL generados con los distintos códigos en el flujo de diseño de Xilinx, y obtienen automáticamente el código VHDL del controlador de reconfiguración `TBC` parametrizado para cada diseño.

- **Plataforma Virtual de verificación y depuración para `Cores Mixtos`.** Realizada en VHDL y sobre el entorno de simulación de Modelsim, esta plataforma ha permitido la verificación conjunta de la ejecución del software y del hardware. Ha constituido un elemento básico para diseñar todos los `Cores Mixtos` y `TnP-Cores` utilizados en esta tesis.

- **Software de `Cores Mixtos` y de `TnP-Cores`.** Ha sido necesario desarrollar el código fuente de todos los módulos que incluían `nano-procesadores`. Los *cores*, utilizados en las distintas plataformas, que incluyen este tipo de procesadores son: `TnP-Master Video`, `DDS-TXi`, `RSSI-LD`, `Wbs_ADC_dynamic_core`, `FFR16`, `TnP-UART` y el propio programa del controlador de reconfiguración `TBC`.

- **Software para un microprocesador externo a la FPGA en el prototipo de experimentación.** Tal y como se ha mencionado anteriormente, el prototipo dispone de un microprocesador externo que permite aplicar la carga de *bitstreams* almacenados en una memoria FLASH de la placa. Para la operativa de este microprocesador ha sido necesario realizar distintos programas con las siguientes funciones:

  - **El control de la carga de los *bitstreams* totales y parciales.** Este software ofrece, a través de un puerto serie a un ordenador PC, un menú con los distintos *bitstreams* parciales que se pueden aplicar a la FPGA.

    Con esta herramienta se han podido realizar ensayos de aplicación de reconfiguración parcial dinámica previos a las plataformas de validación reportadas en esta tesis.

  - **Grabación de los *bitstreams* totales y parciales en una memoria FLASH.** Se ha desarrollado un programa cargador para el microprocesador externo que permite tanto la carga del software para su memoria FLASH interna, como la grabación de los diferentes *bitstreams* en una memoria FLASH paralelo externa (AM29F080) a través de un puerto serie estándar.

    Para gestionar la carga de los *bitstreams* parciales se ha realizado una herramienta gráfica para ordenadores PC con sistema operativo Windows.

19.2. Publicaciones generadas a partir de esta Tesis

Se presentan a continuación las publicaciones mediante las que se ha divulgado el trabajo de investigación realizado en esta tesis. Están ordenadas siguiendo un criterio que sitúa inicialmente las publicaciones donde se presentan los elementos más concretos de `Tornado` (1-5); a continuación se resumen las publicaciones relacionadas con la arquitectura SoPC básica (6-9); y finalmente, las dedicadas a aspectos concretos de la reconfiguración dinámica y diseño de SoPC (10-14):

1. A. Astarloa, J. Lázaro, U. Bidarte, J. L. Martín y A. Zuloaga. *A Self-reconfiguration Framework for Multiprocessor CSoPCs*. Lecture Notes in Computer Science, Springer-Verlag, vol. 3203, pp. 1124-1126, 2004.

   En este artículo se resume el planteamiento teórico de `Tornado`.

2. A. Astarloa, U. Bidarte, J. Lázaro, A. Zuloaga y J. Arias. *Multiprocessor SoPC-Core for FAT Volume Computation*. Microprocessors and Microsystems, Elsevier, aceptada para publicación en el año 2005.

   En esta publicación se expone una topología de *Core Mixto* extendida, que incluye dos `nano-procesadores` trabajando en paralelo en el mismo *core* y está orientada a poder admitir el sistema `Tornado`. Se presenta de modo detallado, el diseño del mismo así como la metodología de desarrollo del *core* estático

3. A. Astarloa, U. Bidarte, A. Zuloaga, J. Arias y J. Jiménez. *Run-time Reconfigurable Hybrid Multiprocessor Cores*. Actas del congreso 2004 IEEE Internacional Conference on Industrial Technology. IEEE Society Press, 2004.

En esta publicación se expone el *Core Mixto* extendido presentado en la segunda publicación de esta lista, pero modificado para soportar la reconfiguración parcial dinámica mediante `Tornado`. Además se incluyen los resultados de los experimentos realizados para integrar ese módulo en diseños con múltiples IP-Cores.

4. A. Astarloa, J. Lázaro, J. Arias, U. Bidarte y A. Zuloaga. *Co-simulation Virtual Platform for Reconfigurable Multiprocessor Hybrid Cores Development.* Actas del congreso International Conference on Modeling, Simulation and Visualization Methods, MSV'04, pp. 17-22. CSREA Press, 2004.

   En esta comunicación se detalla la plataforma de verificación virtual desarrollada para simular y depurar sistemas con varios `Cores Mixtos` trabajando en paralelo (multiprocesador).

5. M. Garay, A. Astarloa, U. Bidarte, A. Zuloaga, y J.L. Martín. *Plataforma CSoPC para la Evaluación del Sistema de Reconfiguración Parcial Dinámica Tornado.* Actas de la las Jornadas de Computación Reconfigurable y Aplicaciones, JCRA'04, pp. 53-61. Universidad Autónoma de Barcelona, 2004.

   En esta publicación se recogen los detalles de diseño de la plataforma de verificación `Video-Multimaster` y los ensayos realizados con la misma.

6. A. Astarloa, U. Bidarte y A. Zuloaga. *A Reconfigurable SoC Architecture for High Volume and Multi-channel Data Transaction in Industrial Environments.* Actas del congreso International IEEE Conference on Industrial Electronics, Control and Instrumentation, IECON'02, pp. 2322-2327, 2002.

   Está publicación detalla el prototipo desarrollado para verificar `Tornado` sobre SoPCs, en el cual se incluye un dispositivo Spartan-II de Xilinx.

7. U. Bidarte, A. Astarloa, A. Zuloaga, J. Jiménez y I. Martinez de Alegría. *Core-Based Reusable Architecture for Slave Circuits with Extensive Data Exchange Requeriments.* Lecture Notes in Computer Science, Springer, vol. 2778, pp. 497-506, 2003.

   En este artículo se propone una arquitectura SoPC de bus compartido basada en *cores* con especificación Wishbone que ha servido de base para la generalización del modelo reconfigurable presentado en `Tornado` y para el desarrollo de las plataformas de validación.

8. U. Bidarte, A. Astarloa, A. Zuloaga, J. L. Martín y J. Jiménez. Capítulo titulado *Core-Based Architecture For Data Transfer Control in Soc Design* del libro New Algorithms, Architectures, and Applications for Reconfigurable Computing. Kluwer Academic Publishers, 2005.

   En esta publicación se detallan las características de la arquitectura genérica basada en *cores* para la transferencia masiva de datos.

9. U. Bidarte, A. Astarloa, J.L. Martín y J. Andreu. *Simulation Platform for Architectural Verfication and Performance Analysis in Core-Based SoC Design.* Lecture Notes in Computer Science, Springer-Verlag, vol. 3203, pp. 965-969, 2004.

   En esta publicación se presentan las herramientas software desarrolladas para realizar simulaciones RTL de sistemas basados en la plataforma genérica anterior.

10. A. Astarloa, U. Bidarte y A. Zuloaga. *Reconfigurable Microstepping Control of Stepper Motors using FPGA embedded RAM.* Actas del congreso 29th International IEEE Conference on Industrial Electronics, Control and Instrumentation, IECON03, pp. 2221-2227, 2003.

    En esta comunicación se presentan los primeros ensayos realizados para utilizar de manera intensiva los bloques de memoria dedicados y reconfigurables de las nuevas FPGAs.

11. U. Bidarte, A. Astarloa, J.L. Martín, J. Jiménez y C. Cuadrado. *On the Performance of Three-State and Multiplexor Logic Interconnection for Shared Bus SoC Design.* Actas del congreso XIX Design of Circuits and Integrated Systems Conference (DCIS'04), pp. 399-404, 2004.

    En este trabajo se comparan los resultados de implementar los buses compartidos empleados en las plataformas SoPC con *buffers* triestado o con multiplexores.

12. J. Lázaro, A. Astarloa, U. Bidarte, J. Arias y C. Cuadrado. *High Throughput Serpent Encryption Implementation.* Lecture Notes in Computer Science, Springer, vol. 3203, pp. 996-1000, 2004.

    Una de las mejoras propuestas en este artículo a la implementación en dispositivos FPGAs de Xilinx del algoritmo Serpent, consiste en la utilización de la reconfiguración parcial dinámica *intra-task*.

13. J. Lázaro, J. Arias, J. L. Martin, A. Astarloa y U. Bidarte. *A Tiny Microprocessor Floating Point Implementation of a General Regression Neural Networks.* WSEAS Transactions on Computers, vol. 4, nº 2, pp. 280-285, 2005.

    En este artículo se detalla una aplicación de computación compleja para los `nano-procesadores` empleados en los `Cores Mixtos` presentados en esta tesis.

14. R. Sancho, U. Bidarte, A. Astarloa, C. Pérez y P. Ibáñez. *SOPC Design Methodology and its Application to the HW-SW Codesign of an Image Server.* Actas del congreso XVIII Design of Circuits and Integrated Systems Conference (DCIS'03), pp. 569-574, 2003.

    Este trabajo se enmarca en las experimentaciones realizadas con dispositivos reconfigurables para la integración de sistemas mixtos con procesamiento hardware y software, empleando en este caso tecnología de Altera.

20. Trabajo futuro

La experiencia acumulada por el autor y por el grupo de investigación en el diseño de sistemas empleando dispositivos reconfigurables, nos ha permitido adoptar un enfoque práctico de la auto-reconfiguración parcial dinámica. Durante varios años, el autor ha diseñado sistemas multi-procesador en una empresa dedicada a sistemas electrónicos de pesaje industrial. El trabajo con este tipo de diseños, que incluían varios procesadores distribuidos en distintas placas y trabajando de forma coordinada, ha inspirado la propuesta realizada en esta tesis. Realmente, el poder integrar de forma económica y eficiente múltiples unidades procesadoras y auto-reconfigurables en un único dispositivo es un reto apasionante y con grandes posibilidades de ser una línea de trabajo aplicable en diseños industriales y con continuidad en el futuro.

Sin embargo, para poder llevar a la práctica esta aspiración se ha necesitado analizar las implicaciones de un diseño de estas características. Realmente, ¿cómo puede afrontarse el cambio dinámico de las secciones hardware y software de un IP-Core mientras que el resto del sistema sigue funcionando? ¿Cuál es la manera más adecuada de gestionar las reconfiguraciones en este tipo de diseños, teniendo en cuenta que la información necesaria para determinar las mismas también va a estar integrada en el mismo dispositivo?

Por ello, teniendo en cuenta el estado del arte, las propuestas existentes y la tecnología disponible, en esta tesis se establecen las bases teóricas para disponer de un sistema que permita diseñar sistemas multi-procesador basados en *cores*. Se ha hecho un desarrollo práctico y se ha verificado la propuesta teórica en varios prototipos, de forma que se puede aplicar con garantías a nuevos diseños y aplicaciones.

Actualmente, tenemos abiertos varios proyectos y experimentos que continúan con la investigación en esta área:

- **Diseño de un controlador de reconfiguración con conexión con el puerto interno de configuración ICAP.** Uno de los mayores inconvenientes detectados en el prototipo realizado es la necesidad de salir al exterior del dispositivo para acceder al puerto de configuración. Esto obliga a incluir elementos externos a la FPGA para realizar la carga de los *bitstreams* parciales. Los nuevos dispositivos incluyen acceso interno al puerto de configuración, por lo que un controlador `TBC` que haga uso del mismo permitirá simplificar el proceso de carga y hará más atractiva la solución de la auto-reconfiguración.

- **Incluir la infraestructura `Tornado` en las nuevas herramientas para el diseño de sistemas basados en IP-Cores.** Las plataformas de verificación presentadas en esta tesis se han desarrollado empleando las herramientas de diseño a nivel de descripción RTL usando lenguajes HDL. Sin embargo, cada vez con más fuerza, se están extendiendo las herramientas de diseño a alto nivel con *cores*. Ejemplos de estos sistemas son los sistemas integrados Excalibur de Altera y el EDK de Xilinx.

Por ello, es necesario trasladar la infraestructura `Tornado` a este tipo de entornos de diseño, de forma que se pueda extender y simplificar su utilización.

En este sentido estamos realizando ensayos para *envolver* diseños realizados con el modelo especificado en esta tesis y con la infraestructura compatible Wishbone para Xilinx al bus Coreconnect^{TM}con el *wrapper* WB2OPB/OPB2WB [181]. Este módulo, disponible de forma pública, se integra en el entorno EDK como un *plug-in*, pudiéndose utilizar módulos descritos a nivel RTL para la especificación Wishbone sintetizados en el entorno ISE.

- **Incluir la *metacomputación* en sistemas operativos para procesadores de propósito general.** Las nuevas FPGAs pueden incluir potentes procesadores de propósito general embebidos en el propio silicio del dispositivo. El flujo de diseño utilizado con estas FPGAs se basa en las herramientas que trabajan a alto nivel con IP-Cores indicadas en el apartado anterior. Por ello, tras conseguir integrar la infraestructura `Tornado` en las plataformas indicadas se abre un nuevo campo de posibilidades: al poderse incluir los diseños que incorporan `Tornado` en estas plataformas, los procesadores pueden escribir palabras de configuración en el controlador de configuración, lo que posibilita que los sistemas operativos que estén funcionando en esos microprocesadores gestionen tareas hardware junto con las tareas software.

  En este sentido, disponemos actualmente de varios prototipos con dispositivos Virtex-II Pro que incluyen dos procesadores PowerPC 405 en el propio silicio. En uno de ellos se ejecuta el sistema operativo Linux, siendo un objetivo a corto plazo el desarrollo de un *driver* para acceder al controlador de reconfiguración `TBC` desde el mismo.

- **Ampliar los ensayos de reconfiguración *inter-task*.** La inestabilidad de las herramientas que en la actualidad proveen los fabricantes de lógica programable para experimentar con diseños reconfigurables dinámicamente ha limitado los ensayos relativos a la reconfiguración *inter-task*. Con el nuevo prototipo y con los elementos que se proponen como trabajo futuro, este tipo de reconfiguración adquiere un gran interés, pudiendo ser evaluada adecuadamente a medida que las herramientas disponibles la soporten de forma estable.

- **Incluir el sistema `Tornado` en una especificación estándar para interconexión de IP-Cores.** El planteamiento teórico realizado del sistema `Tornado` puede resumirse en una especificación que deben cumplir los integrantes de un sistema SoPC para que admita la auto-reconfiguración controlada. Se detallan los protocolos, señales y los requisitos que deben cumplir el controlador de reconfiguración, los `nano-procesadores` e incluso los *wrappers* para la adaptación de IP-Cores.

  Conseguir que `Tornado` se integre dentro de una especificación estándar para interconexión de IP-Cores como Wishbone, permitiría que se extienda el sistema propuesto a otros grupos de investigación de tal forma que se experimenten y desarrollen nuevas aplicaciones y módulos para el mismo.

- **Aplicar el sistema `Tornado` a diseños industriales.** En nuestro grupo, estamos trabajando en la utilización del sistema `Tornado` y de la infraestructura desarrollada para dispositivos Xilinx en aplicaciones de control de convertidores de potencia y en diseños industriales en general.

Bibliografía

[1] S. A. Guccione y D. Levi: *The Advantages of Run-Time Reconfiguration.* En *Proceedings of the Reconfigurable Technology: FPGAs for Computing and Applications (SPIE'99)*, páginas 87–92. The International Society for Optical Engineering, Septiembre 1999.

[2] G. E. Moore: *Cramming More Components onto Integrated Circuits.* Electronics, 38(8), Abril 1965.

[3] L. Waller: *The Big Question in Counting FPGA Gates: Should Memory Be Included.* EE Times Online, `http://www.eedesign.com/editorial/1997/fpgacolumn9710.html`, Octubre 1997.

[4] Y.W. Chan: *Physical design for SoC.* Tutorial Soc_pd2`http://http://cc.ee.ntu.edu.tw/~ywchang`, 2002.

[5] R. Rajsuman: *System-on-a-Chip. Design and Test.* Artech House Publishers, 2000.

[6] J. Borel: *SoC Design Challenges: the EDA MEDEA Roadmap.* Servicio de Publicaciones de la Universidad de Cantabria, 2001.

[7] A. Chao: *Platform Design System SoC.* Faraday Technology Corporation, `http://www.faraday-tech.com`, 2002.

[8] Virtual Socket Interface Aliance: *Specifications, Standars, Technical Documents.* `http://www.vsia.org`, 2003.

[9] G. Martin y H. Chang (Eds.): *Winning the SoC Revolution: Experiences in Real Design.* Kluwer Academic Publishers, Massachusetts, USA, 2003.

[10] M. Barr: *A Reconfigurable Computing Primer.* `http://www.netrino.com/Articles/RCPrimer/`, 1998.

[11] K. Compton y S. Hauck: *Reconfigurable Computing: A Survey of Systems and Software.* ACM Computing Surveys, 34(2):171–210, Junio 2002.

[12] J. Tuley: *Soft Computing Reconfigures Designer Options.* Embedded Systems, página 76, Abril 1997.

[13] K. Bondalapati y V.K. Prasanna: *Reconfigurable Computing Systems.* Proceedings of the IEEE, 90(7):1201–1217, Julio 2002.

[14] A. DeHon: *Reconfigurable Architectures for General Purpose Computing.* A.I. Technical report No. 1586. Artificial Intelligence Laboratory. Massachusetts Institute of Technology, Septiembre 1996.

[15] R. Tessier y W. Burleson: *Reconfigurable Computing for Digital Signal Processing: a survey.* Journal of VLSI Signal Processing, (28):7–27, Mayo 2001.

[16] R. Sidhu, A. Mei y V.K. Prassana: *Genetic Programming using Self-Reconfigurable FPGAs.* Lecture Notes in Computer Science, 1673:301–312, 1999.

[17] R. Sidhu y A. Mei V.K. Prasanna: *String Matching on Multicontext FPGAs using Self-Reconfiguration.* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'99)*, páginas 217–226, Febrero 1999.

[18] R. Sidhu y V.K. Prasanna: *Fast Regular Expression Matching using FPGAs.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, páginas 698–709, Abril 2001.

[19] J. Hauser y J. Wawrzynek: *Garp: A MIPS Processor with a Reconfigurable Coprocessor.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, páginas 12–21, Abril 1997.

[20] K.H. Leung, K.W. Wong y P.H.W. Leong: *FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, páginas 68–76, Abril 2000.

[21] J. Lázaro, A. Astarloa, U. Bidarte, J. Arias y C. Cuadrado: *High Throughput Serpent Encryption Implementation.* Lecture Notes in Computer Science, 3203:996–1000, 2004.

[22] W.J. Huang and N. Saxena y E.J. McCluskey: *A reliable LZ data compressor on reconfigurable coprocesors.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, páginas 249–258, Abril 2000.

[23] J. Burns, A. Donlin, J. Hogg, S. Singh y M. De Wit: *A Dynamic Reconfiguration Run-Time System.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, páginas 66–76, Abril 1997.

[24] J. Gause, P.Y.K. Cheung y W. Luk: *Reconfigurable Shape-Adaptive Template Matching Architectures.* En *Proceedigns of the IEEE FPGA Custom Computing Machine Conference 2002*, páginas 98–110, 2002.

[25] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene y S. Kaptanoglu: *An FPGA family optimized for high densities and reduced routing delay.* En *Proceedings of the IEEE Custom Integrated Circuits Conference*, páginas 31.5.1–31.5.4, 1990.

[26] H. Hsieh, W. Carter, J. Y. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin y L. Tinkey: *Third-generation Architecture Boosts Speed and Density of Field-programmable Gate arrays.* En *Proceedings of the IEEE Custom Integrated Circuits Conference*, páginas 31.2.1–31.2.7, 1990.

[27] Actel Corporation: *Accelerator Series FPGAs: ACT3 Family.* `http://www.actel.com`, 1997.

[28] Actel Corporation: *SX Family of High Performance FPGAs.* `http://www.actel.com`, 2001.

[29] S. Hauck, T.W. Fry, M.M. Hosler y J.P. Kao: *The Chimaera Reconfigurable Functional Unit.* IEEE Transactions on VLSI Systems, 12(2):206–217, Febrero 2004.

[30] C.R. Rupp, M. Landguth, T. Garverick, E. Comersall, H. Holt, J.M. Arnold y M. Gokhale: *The NAPA adaptive processing architecture.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, páginas 28–37, Abril 1998.

[31] Xilinx Corp.: *Using Block SelectRAM+ Memory in Spartan II FPGAs.* Xilinx Application Notes, `http://www.xilinx.com`, Deciembre 2000.

[32] Inc. Chameleon Systems: *CS2000 Advance Product Specification.* `http://www.isis.vanderbilt.edu/projects.asp`, 2000.

[33] Z. Li, K. Compton y S. Hauck: *Configuration Caching Techniques for FPGA.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, páginas 22–38, Abril 2000.

[34] T. Anderson: *System-on-Chip Design with Virtual Components.* Circuit Cellar Online, `http://www.circuitcellar.com`, Agosto 1999.

[35] S. Trimberger, D. Carberry, A. Johnson y J. Wong: *A time-multiplexed FPGA.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, páginas 22–28, Abril 1997.

[36] A. DeHon: *DPGA Utilization and Applications.* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'96)*, páginas 115–121, Febrero 1996.

[37] K. Danne, C. Bobda y H. Kalte: *Run-Time Exchange of Mechatronic Controllers Using Partial Hardware Reconfiguration.* Lecture Notes in Computer Science, 2778:272–281, 2003.

[38] Y. Chou, P. Pillai, H. Schmit y J.P. Shen: *PipeRench Implementation of the Instruction Path Coprocessor.* En *Proceedigns of the 33th Annual International Symposium on Microarchitecture (MICRO)*, páginas 147–158, 2000.

[39] Xilinx Corp.: *XC6200 Field Programmable Gate Arrays.* Xilinx Documentation, `http://www.xilinx.com`, 2002.

[40] Xilinx Corp.: *Virtex 2.5V Field Programmable Gate Array.* Xilinx Documentation, `http://www.xilinx.com`, Deciembre 2002.

[41] H. Schmit: *Incremental Reconfiguration for Pipelined Applications.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, páginas 16–18, Abril 1997.

[42] S. Hauck: *Configuration Prefetch for Single Context Reconfigurable Coprocessors.* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'98)*, páginas 65–74, Febrero 1998.

[43] S. Hauck, Z. Li y E. Schewabe: *Configuration Compression for the Xilinx XC6200 FPGA.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, páginas 138–147, Abril 1998.

[44] K. Compton, Z. Li, J. Cooey, S. Knol y S. Hauck: *Configuration Relocation and Defragmentation for Run-Time Reconfigurable Computing.* IEEE Transactions on VLSI Systems, 10(3):209–220, Junio 2002.

[45] D. A. Buel, J.M. Arnold y W.J. Kleinfelder: *Splash 2: FPGAs in a Custom Computing Machine.* Wiley-IEEE Computer Society Press, 1996.

[46] R.A. Keaney, L.H. C. Lee, D. J. Skellern, J.E. Vuillemin y M. Shand: *Implementation of Long Constraint Length Viterbi Decoders using Programmable Active Memories.* En *Proceedigns of the 11th Australian Microelectronics*, páginas 52–57. QLD Australia, Octubre 1993.

[47] J.R. Hauser: *Augmenting a Microprocessor with Reconfigurable Hardware.* Tesis de Doctorado, University of California, Berkeley, 2000.

[48] R. D. Wittig: *OneChip: An FPGA Processor With Reconfigurable Logic.* Tesis de Doctorado, Department of Electrical and Computer Engineering, University of Toronto, 1995.

[49] R. D. Wittig y P. Chow: *OneChip: An FPGA Processor With Reconfigurable Logic.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'96)*, páginas 126–135, Abril 1996.

[50] M. J. Wirthlin y B. L. Hutchings: *DISC: A dynamic instruction set computer.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'95)*, páginas 99–109, Napa Valley, California, Abril 1995.

[51] M. J. Wirthlin y B. L. Hutchings: *Sequencing Run-Time Reconfigured Hardware with Software.* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'98)*, páginas 122–128, Febrero 1996.

[52] P.C. Fren y R.W. Taylor: *A Self-reconfiguring Processor.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'93)*, páginas 50–59, Abril 1993.

[53] S.P. Seng, W. Luk y P.Y.K. Cheung: *Flexible Instruction Processors.* En *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems (CASES'00)*, páginas 193–200, 2000.

[54] S. Seng, W. Luk y P. Cheung: *Run-time Adaptive Flexible Instruction Processors.* Lecture Notes in Computer Science, 2438:545–555, 2002.

[55] J. Faura, M. Aguirre, J. Moreno, P. van Duong y J. Insenser: *FIPSOC: A Field Programmable System On a Chip*. En *Proceedings of the Design of Circuits and Integrated Systems Conference (DCIS'97)*, páginas 597–602, Noviembre 1997.

[56] U.C. Berkeley: *Pleaides Group*. `http://bwrc.eecs.berkeley.edu/Research/Configurable_Architectures`.

[57] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes y A. Abnous: *A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications*. En *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC'00)*, 2000.

[58] J. Becker: *Configurable Systems-on-Chip (CSoC)*. En *Proceedings of the 15th Symposium on Integrated Circuits and Systems Design (SBCCI'02)*, páginas 390–395, Septiembre 2002.

[59] J. Becker, T. Pionteck y M. Glesner: *An Application-tailored Dynamically Reconfigurable Hardware Architecture for Digital Baseband Processing*. En *Proceedings of the 13th Symposium on Integrated Circuits and Systems Design (SBCCI'00)*, página 341, Septiembre 2000.

[60] J. Becker and N. Liebau, T. Pionteck y M. Glesner: *Efficient Mapping of presynthesized IP-Cores onto Dynamically Reconfigurable Array Architectures*. Lecture Notes in Computer Science, 2147:584–589, 2001.

[61] ARM Ltd.: *AMBA 2.0 Specification*. `http://www.arm.com`.

[62] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach y M. Wreinhardt: *PACT XPP - A Self-Reconfigurable Data Processing Architecture*. The Journal of Supercomputing, 26:167–184, Septiembre 2003.

[63] PACT Corp.: *The XPP Communication System. Tecnical Report 15*. `http://www.pactcorp.com`, 2000.

[64] K. Sankaralingam, R. Nagarajan, H. Lui, C. Kim, J. Huh, D. Burger, S.W. Keckler y C.R. Moore: *Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture*. IEEE Micro, 23(6):46–51, Noviembre 2003.

[65] PACT Informationstechnologie GmbH: *1998a, WO 98/26356*. Patent, 1998.

[66] PACT Informationstechnologie GmbH: *2001c, XPP Tecnology White Paper*. `http://www.pactcorp.com`, 2001.

[67] J. Gaisler: *LEON 2 Processor*. `http://www.gaisler.com`, 2003.

[68] J. Becker, A. Thomas, M. Vorbach y G. Ehlers: *Dynamically Reconfigurable Systems-on-Chip: A Core-based Industrial/Academic SoC Synthesis Project*. En *Proceedings of the IEEE Workshop Heterogeneous Reconfigurable Systems-on-Chip*, Abril 2002.

[69] J. Becker, A. Thomas, M. Vorbach y V. Baumgarte: *An Industrial/Academic Configurable System-on-Chip Project (CSoC): Coarse-Grain XPP-/Leon-Based Architecture Integration*. En *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, páginas 1120–1121, Septiembre 2003.

[70] R. Enzel, C. Plessl y M. Plazer: *Virtualizing Hardware with Multi-context Reconfigurable Arrays*. Lecture Notes in Computer Science, 2778:151–160, 2003.

[71] R. Sidhu, S. Wadhwa, A. Mei y V.K. Prassana: *A Self-Reconfigurable Gate Array Architecture*. Lecture Notes in Computer Science, 1896:352–360, 2000.

[72] R. Bittner y P. Athanas: *Wormhole Run-time Reconfiguration*. En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'00)*, páginas 79–85, Febrero 1997.

[73] Triscend Corp.: *Configurable System-on-Chip Platforms*. `http://www.triscend.com`, 2004.

[74] Triscend Corp.: *Configurable System Interconnect (CSI) Bus User's Guide*. `http://www.triscend.com/support/index.htm`, 2004.

[75] Xilinx Corp.: *Spartan II 2.5V FPGA Family Datasheet*. Xilinx Documentation, Febrero 2001.

[76] M. George y P. Alfke: *Linear Feedback Shift Registers in Virtex Devices*. Xilinx Application Notes, `http://www.xilinx.com`, Enero 2001.

[77] Xilinx Corp.: *Using Delay-Locked Loops in the Spartan II FPGAs*. Xilinx Application Notes, `http://www.xilinx.com`, Enero 2000.

[78] U. Bidarte, A. Astarloa, J.L. Martín, J. Jiménez y C. Cuadrado: *On the Performance of Three-State and Multiplexor Logic Interconnection for Shared Bus SoC Design*. En *Proceedings of the Design of Circuits and Integrated Systems Conference (DCIS'04)*, páginas 399–404, Noviembre 2004.

[79] Xilinx Corp.: *Spartan II FPGA Family Configuration and Readback*. Xilinx Application Notes, `http://www.xilinx.com`, Deciembre 1999.

[80] Xilinx Corp.: *Virtex FPGA Series Configuration and Readback*. Xilinx Application Notes, `http://www.xilinx.com`, Julio 2002.

[81] Xilinx Corp.: *Virtex Series Configuration Architecture User Guide*. Xilinx Application Notes, `http://www.xilinx.com`, Febrero 2003.

[82] C. Carmichael, M. Caffrey y A. Salazar: *Correcting Single-Event Upsets Through Virtex Partial Configuration*. Xilinx Application Notes, `http://www.xilinx.com`, Junio 2000.

[83] Xilinx Corp.: *Configuration and Readback of Virtex FPGAs Using (JTAG) Boundary Scan*. Xilinx Application Notes, `http://www.xilinx.com`, Septiembre 2003.

[84] ATMEL: *AT40K Datasheet*. `http://www.atmel.com/atmel/acrobat/doc2818.pdf`, 2004.

[85] G. McGregor y P. Lysaght: *Self Controlling Dynamic Reconfiguration: A Case Study*. Lecture Notes in Computer Science, 1673:144–154, 1999.

[86] P. Lysaght: *Aspects of Dynamically Reconfigurable Logic.* IEE Coloquium on Reconfigurable Systems, Febrero 1999.

[87] R. Sidhu y V.K. Prasanna: *Efficient Metacomputation using Self-Reconfiguration.* Lecture Notes in Computer Science, 2438:698–709, 2002.

[88] N. Shirazi, W. L. y Peter Y. K. Cheung: *Automating Production of Run-Time Reconfigurable Designs.* En Kenneth L. Pocek y Jeffrey Arnold (editores): *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, páginas 147–156, Abril 1998.

[89] W. Luk and N. Shirazi y P.Y.K. Cheung: *Compilation tools for run-time reconfigurable designs.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, páginas 56–65, Abril 1997.

[90] W. Luk and N. Shirazi y P.Y.K. Cheung: *Modelling and optimising run-time reconfigurable systems.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'96)*, páginas 167–176, Abril 1996.

[91] Xilinx Corp.: *JBits 3.0 SDK for Virtex-II.* `http://www.xilinx.com/labs/projects/jbits/`, Agosto 2003.

[92] Altera: *LPM Quick Reference Guide.* `http://www.altera.com/literature/catalogs/lpm.pdf`, Deciembre 1996.

[93] Xilinx Corp.: *Core Generator Guide.* Xilinx Design Tool Center, `http://www.xilinx.com`, 2003.

[94] S. A. Guccione y D. Levi: *Run-Time Parametrizable Cores.* Lecture Notes in Computer Science, 1673:215–222, 1999.

[95] E. Lechner y S.A. Guccione: *The Java environment for reconfigurable computing.* Lecture Notes in Computer Science, 1304:284–293, 1997.

[96] S. A. Guccione y D. Levi: *XBI: A Java-based interface to FPGA Hardware.* En *Proceedings of the Configurable Computing Technology and its use in High Performance Computing, DSP and Systems Engineering (SPIE Photonics East)*, páginas 97–102. The International Society for Optical Engineering, Noviembre 1998.

[97] G. Brebner y A. Donlin: *Runtime Reconfigurable Routing.* Lecture Notes in Computer Science, 1388:25–30, 1998.

[98] Xilinx Corp.: *Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations.* Xilinx Application Notes, `http://www.xilinx.com`, Mayo 2002.

[99] M. Dyer y M. Wirz: *Reconfigurable System on FPGA.* Computer Engineering, Master Tesis: Swiss Federal Institute of Technology Zurich, 2002.

[100] P. James-Roxby y S. A. Guccione: *Automated Extraction of Run-Time Parametrisable Cores from Programmable Device Configurations.* páginas 153–161, Abril 1997.

[101] K. Chapman: *Constant Coeffucient Multipliers for XC4000E.* Xilinx Application Notes, `http://www.xilinx.com`, Deciembre 1996.

[102] E. L. Dickson: *Linear Groups: With an exposition of the Galois Field Theory.* Dover Publications, New-York, 1958.

[103] R. Payne: *Run-time Parameterised Circuits for the Xilinx XC6200.* Lecture Notes in Computer Science, 1304:191–172, 1997.

[104] P. Athanas y A. Abbott: *Real-Time Image Processing on a Custom Computing Platform.* IEEE Computer, páginas 16–24, Febrero 1995.

[105] A. Dandalis y V.K. Prasana: *Fast Parallel Implementation of DFT using Configurable Devices.* Lecture Notes in Computer Science, 1304:1400–1425, 1997.

[106] S. McMillan y S. A. Guccione: *Partial Run-Time Reconfiguration Using JRTR.* Lecture Notes in Computer Science, 1896:352–360, 2000.

[107] AT. Grötker, S. Liao, G. Martin y S. Swan: *System Design with SystemC.* Kluwer Academic Publishers, Massachusetts, USA, 2002.

[108] H. Hubert: *A Survey of HW/SW cosimulation techniques and tools.* Tesis de Doctorado, Royal Inst. of Tech., Sweden, Junio 1998.

[109] P. Gerin, S. Yoo, G. Nicolescu y A. Jerraya: *Scalable and Flexible Cosimulation of SoC Designs with Heterogeneous Multi-Processor Target Architectures.* En *Proceedings of the Asia South Pacific Design Automation Conference,*, páginas 63–68, 2001.

[110] H. Chang et al.: *Surviving the SOC Revolution.* Kluwer Academic Publishers, Massachusetts, USA, 1999.

[111] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A.L. Sangiovanni-Vicentelli, E. Sentovich, K. Suzuki y B. Tabbara: *Hardware-Software Co-Design of Embbeded Systems: The POLIS Approach.* Kluwer Academic Publishers, MA, USA, Mayo 1997.

[112] B. Tabbara, A. Tabbara y A. Sangiovanni-Vicentelli: *Function / Architecture Optimization and Co-Design of Embedded Systems.* Kluwer Academic Publishers, Massachusetts, USA, 2000.

[113] A. Gerstlauer, R. Dömer, J. Peng y D. Gajski: *SYSTEM DESIGN: A Practical Guide with SpecC.* Kluwer Academic Publishers, Massachusetts, USA, 2001.

[114] Open SystemC Initiative(OSCI): *SystemC User's Forum Presentation.* `http://www.systemc.org/news/systemc_user_forum.pdf`, Junio 2000.

[115] Open SystemC Initiative(OSCI): *SystemC User's Forum Presentation.* `http://www.systemc.org/news/systemc_dac_01.pdf`, Junio 2001.

[116] J. Connelland y B. Johnson: *Early Hardware/Software Integration Using SystemC 2.0.* En *Proceedings of the Embedded Systems Conference '02*, página 552, Febrero 2002.

[117] S. Swan: *An Introduction to System Level Modeling in SystemC 2.0.* Open SystemC Initiative (OSCI), 2001.

[118] A. Finand F. Fummiand D. Signoretto: *The Use of SystemC for Design Verification and Integration Test of IP-Cores.* En *Proceedigns of the IEEE ASIC-SOC 2001*, Washington, USA, Septiembre 2001.

[119] A. Fin, F. Fummi y D. Signoretto: *SystemC: A Homogenous Environment to Test Embedded Systems.* En *Proceedigns of the 9th International Workshop on Hardware/-Software Co-Design (CODES'01)*, páginas 17–22, Cophenagen, Abril 2001.

[120] K. Kwiat y W. Debany: *Reconfigurable Logic Modelling.* Integr. Syst. Des, `http://www.isdmag.com`, 1996.

[121] J. Faura, J.M. Moreno, J. Madrenas y J.M. Insenser: *VHDL Modelling of Fast Dynamic Reconfiguration on Novel Multicontext RAM-based Field Programmable Devices.* VHDL User´s Forum in Europe (VUFE'97), 1997.

[122] D. Robinson y P. Lysaght: *Methods of exploting simulation technology for simulating the timing of dynamically reconfgurable logic.* IEE Proceedings - Computers and Digital Tecniques 143:(3), páginas 175–180, Mayo 2000.

[123] I. Robertson, J. Irvine, P. Lysaght y D. Robinson: *Improved Functional Simulation of Dynamically Reconfigurable Logic.* Lecture Notes in Computer Science, 2438:152–161, 2002.

[124] M. Vasilko y D. Cabanis: *Improving Simulation Accuracy in Design Methodologies for Dynamically Reconfigurable Logic Systems.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, páginas 20–23, Abril 1999.

[125] P. Lysaght y J. Stockwood: *A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays.* IEEE Transactions on VLSI Systems, 4(3):381–390, Septiembre 1996.

[126] D. Robinson y P. Lysaght: *Verification of Dynamically Reconfigurable Logic.* Lecture Notes in Computer Science, 1896:141–150, 2000.

[127] I. Robertson, J. Irvine, P. Lysaght y D. Robinson: *Timing Verfication of Dynamically Reconfigurable Logic for the Xilinx Virtex FPGA Series.* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'02)*, páginas 127–135, Febrero 2002.

[128] A.M. Rincon, C. Cherichetti, J.A. Monzel, D.R. Stauffer y M.T. Trick: *Core Design and System-on-a-Chip Integration.* IEEE Design & Test of Computers, páginas 26–35, Octubre 1997.

[129] A. Astarloa, U. Bidarte, A. Zuloaga, J. Arias y J. Jiménez: *Run-time Reconfigurable Hybrid Multiprocessor Cores.* En *Proceedings of the 2004 IEEE Internacional Conference on Industrial Technology*, Deciembre 2004.

[130] K.A. Vissers: *Parallel Processing Architectures for Reconfigurable Systems.* En *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, páginas 10396–10397, Septiembre 2003.

[131] J. MacBeth y P. Lysaght: *Dynamically Reconfigurable Cores.* Lecture Notes in Computer Science, 2147:462–472, 2001.

[132] J. MacBeth y P. Lysaght: *Dynamically Reconfigurable Intelectual Property.* En *Proceedings of the Postgraduate Research in Electronics, Photonics, Communications and Software (PREP'01)*, Abril 2001.

[133] J. MacBeth y P. Lysaght: *Investigating Dynamic Reconfiguration of FPGA Based IP Cores.* En *Proceedigns of the IP 2001 System-on-Chip Conference*, Lyon, France, Octubre 2001.

[134] M. Dyer, C. Plessl y M. Platzner: *Partially Reconfigurable Cores for Xilinx Virtex.* Lecture Notes in Computer Science, 2438:292–301, 2002.

[135] J. W. Lookwood and N. Naufel, J.S. Turner y D.E. Taylor: *Reprogrammable Network Packet Processing on the Field Programable Port Extender (FPX).* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'01)*, páginas 87–93, Febrero 2001.

[136] F. Braun, J. W. Lockwood y M. Waldvogel: *Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware.* Technical Report WUCS-01-10. Applied Research Lab. Department of Computer Science. Washignton University, Julio 2001.

[137] F. Braun, M. Waldvogel y J. Lockwood: *OBIWAN - an Internet Protocol Router in Reconfigurable Hardware.* Technical Report WUCS-01-11. Applied Research Lab. Department of Computer Science. Washignton University, Julio 2001.

[138] J. W. Lockwood, C. Neely, C. Zuver, J. Moscola, S. Dharmapurikar y D. Lim: *An Extensible, System-On-Programmable-Chip, Content-Aware Internet Firewall.* Lecture Notes in Computer Science, 2778:859–868, 2003.

[139] D.E. Taylor, J.S. Turner, J. W. Lookwood y E. L. Horta: *Dynamic Hardware Plugins (DHP): Exploiting reconfigurable hardware for high-performance programmable routers.* Computer Networks, 38(3):295–310, Febrero 2002.

[140] E. L. Horta y J. W. Lookwood: *PARBIT: A Tool to Transform BitFiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs).* Technical Report. Applied Research Lab. Department of Computer Science. Washignton University, Julio 2001.

[141] Edson L. Horta, John W. Lockwood, David E. Taylor y David Parlour: *Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration.* En *Proceedings of the Design Automation Conference (DAC'02)*, páginas 343–348, New Orleans, LA, Junio 2002.

[142] A. Donlin: *Self Modifying Circuitry - A Platform for Tractable Virtual Circuitry.* Lecture Notes in Computer Science, 1482:200–208, 1998.

[143] Douglas W. Jones: *The Ultimate RISC.* ACM Computer Architecture News, 16(3):48–55, Junio 1988.

[144] P. W. Fould y L.D. Hodson: *Data Folding in SRAM Configurable FPGAS.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'93)*, páginas 163–171, Abril 1993.

[145] R. J. Fong, S. J. Harper y P. M. Athanas: *A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguration.* En *Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping (RSP'03)*, páginas 117–123, Junio 2003.

[146] V. Eck, P. Kalra, R. LeBlanc y J. McManus: *In-Circuit Partial Reconfiguration of RocketIO Attributes.* Xilinx Application Notes, `http://www.xilinx.com`, Enero 2003.

[147] Inc. IBM: *Coreconnect Spec.* IBM web site: `http://www.chips.ibm.com/products/coreconnect`, 2003.

[148] Altera: *Avalon Bus Specification Reference Manual 2.3.* `http://www.altera.com/literature/manual/mnl_avalon_bus.pdf`, Julio 2003.

[149] ATMEL: *FPSLIC Home Page.* `http://www.atmel.com/products/FPSLIC/`, Julio 2003.

[150] Kyeong Keol Ryu, Eung Shin y Vincent J. Mooney: *A Comparison of Five Different Multiprocessor SoC Bus Architectures.* En *Proceedings of the Euromicro Symposium on Digital Systems Design (DSD'01)*, página 202, Septiembre 2001.

[151] Silicore Corporation: *Summary of SoC Interconnection Buses.* `http://www.silicore.net`, 2002.

[152] U. Bidarte: *Arquitectura Basada en Cores para Control de Transferencias de Datos en SoC.* Tesis de Doctorado, Universidad del País Vasco, Junio 2004.

[153] Rudolf Usselmann: *SoC Bus Review.* `http://www.opencores.org`.

[154] Silicore Corporation: *Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores Revision: B.3.* `http://www.opencores.org`, Septiembre 2002.

[155] Xilinx Corp.: *ISE 6.1 Xilinx Libraries Guide.* `http://support.xilinx.com`, 2003.

[156] Inc. Atmel: *FPSLIC on-chip Partial Reconfiguration of the Embedded AT40K FPGA.* Atmel Application Note, 2002.

[157] B. Blodget, P. James-Roxby, E. Keller, S. McMillan y P. Sundararajan: *A Self-reconfiguring Platform.* Lecture Notes in Computer Science, 2778:565–574, 2003.

[158] B. Blodget, S. McMillan y P. Lysaght: *A lightweight approach for embedded reconfiguration of FPGAs.* En *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, página 1530, Septiembre 2003.

[159] Xilinx Corp.: *Virtex II Platform FPGA User Guide.* Xilinx Documentation, `http://www.xilinx.com`, Deciembre 2002.

[160] Xilinx Corp.: *Virtex II PRO Platform FPGA User Guide.* Xilinx Documentation, `http://www.xilinx.com`, Febrero 2003.

[161] S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget y D. Levi: *A High I/O Reconfigurable Crossbar Switch.* En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'03)*, páginas 3–11, Abril 2003.

[162] M. Ullmann, M. Hübner, B. Grimm y J. Becker: *On-Demand FPGA Run-Time System for Dynamical Reconfiguration with Adaptive Priorities.* Lecture Notes in Computer Science, 3203:454–463, 2004.

[163] M. Ullmann, M. Hübner, B. Grimm y J. Becker: *An FPGA Run-Time System for Dynamical On-Demand Reconfiguration.* En *Proceedings of the 11th Reconfigurable Architectures Workshop (RAW/IPDPS)*, Abril 2004.

[164] M. Hübner, M. Ullmann, L. Braun, A. Klausmann y J. Becker: *Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems.* Lecture Notes in Computer Science, 3203:1037–1041, 2004.

[165] A. Jantsch y H. Tenhunen: *Networks on Chip.* Kluwer Academic Publishers, 2003.

[166] T. Marescaux, J-Y. Mignolet, A. Bartic, W. Moffat, D. Verkest, S. Vernalde y R. Lauwereins: *Networks on Chip as Hardware Components of an OS for Reconfigurable Systems.* Lecture Notes in Computer Science, 2778:595–605, 2003.

[167] D. E. Culler y J. P. Singh: *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann Publishers, 1999.

[168] L. Möller and N. Calazans, F. Moraes, E. Briao y E. Carvalho snd D. Camozzato: *FiPRe: An Implementation Model to Enable Self-Reconfgurable Appliactions.* Lecture Notes in Computer Science, 3203:1042–1046, 2004.

[169] J.C. Palma, A.V. de Mello, L. Möller y F. Moraes and N. Calazans: *Core Communication Interface for FPGAs.* En *Proceedigns of the ACM 17th South Microelectronics Seminar (SIM'02)*, páginas 183–190, Junio 2002.

[170] F. Moraes and N. Calazans: *R8 Processor Architecture and Organization Specification and Design Guidelines.* `http://www.inf.pucrs.br/~graph/Projects/R8/pulbic/R8_arq_apec_eng.pdf`, Septiembre 1996.

[171] F.G. Moraes, D. Mesquita, J.C. Palma y L. Möller and N. Calazans: *Development of a Tool-Set for Remote and Partial Reconfiguration of FPGAs.* En *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, páginas 11122–11123, Febrero 2003.

[172] H. Walder y M. Platzner: *Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations.* En *Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*, páginas 284–287. CSREA Press, Junio 2003.

[173] H. Walder y M. Platzner: *A Runtime Environment for Reconfigurable Hardware Operating Systems.* Lecture Notes in Computer Science, 3203:831–835, 2004.

[174] H. Walder, S. Nobs y M. Platzner: *XF-Board: A Prototyping Platform for Reconfigurable Hardware Operating Systems.* En *Proceedigns of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*, página 306. CSREA Press, Junio 2004.

[175] C. Steiger, H. Walder y M. Platzner: *Heuristics for Online Scheduling Real-Time Tasks to Partially Reconfigurable Devices.* Lecture Notes in Computer Science, 2778:575–584, 2003.

[176] A. Segard y F. Verdier: *SOC and RTOS: Managing IPs and Tasks Communications.* Lecture Notes in Computer Science, 3203:710–718, 2004.

[177] *A D&T Roundtable: Are Single-Chip Multiprocessors in Reach?* IEEE Design & Test of Computers, 18(1):82–89, 2001.

[178] K. Chapman: *KCPSM Constant(k) Coded Programmable State Machine.* Xilinx Application Notes, `http://www.xilinx.com`, 2000.

[179] K. Chapman: *PicoBlaze 8-Bit Microcontroller for Virtex-E and Spartan II/IIE Devices.* Xilinx Application Notes, `http://www.xilinx.com`, Febrero 2003.

[180] G. Ferrante: *CPUGEN Tutorial V1.00.* `http://www.opencores.org/projects/cpugen`, 2003.

[181] OpenCores Comunity: *OpenCores: Free open source IP Cores and Chip Design.* `http://www.opencores.org`, 2004.

[182] A. Astarloa, J. Lázaro, J. Arias, U. Bidarte y A. Zuloaga: *Co-simulation Virtual Platform for Reconfigurable Multiprocessor Hybrid Cores Development.* En *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods, (MSV'04)*, páginas 17–22. CSREA Press, Junio 2004.

[183] U. Bidarte, A. Astarloa, J.L. Martín y J. Andreu: *Simulation Platform for Architectural Verfication and Performance Analysis in Core-Based SoC Design.* Lecture Notes in Computer Science, 3203:965–969, 2004.

[184] K.Ñ. Virkam y V. Vasudevan: *Hardware-Software Co-simulation of Bus-based Reconfigurable Systems.* Microprocessors and Microsystems, 29(4):133–144, 2005.

[185] B.D. Theelen, A.C. Verchueren, V.V. Reyes Suárez, M.P.J. Stevens y A. Nunez: *A Scalable Single-chip Multi-processor Architecture with On-chip RTOS kernel.* Journal of Systems Architecture, (49):619–639, 2003.

[186] H. V. Kampen: *PicoBlaze Rijndael (AES-128) block cipher.* `http://www.mediatronix.com`, 2003.

[187] J. Lázaro, J. Arias, J. L. Martin, A. Astarloa y U. Bidarte: *A Tiny Microprocessor Floating Point Implementation of a General Regression Neural Network.* WSEAS Transactions on Computers, 4(2):280–285, Febrero 2005.

[188] A. Baghdadi, N. E. Zergainoh, W. Cesario, T. Roudier y Ahmed Amine Jerraya: *Design Space Exploration for Hardware/Software Codesign of Multiprocessor Systems.* En *Proceedings of the IEEE International Workshop on Rapid System Prototyping*, páginas 8–13, 2000.

[189] Xilinx Corp.: *CryptoBlaze: 8-Bit Security Microcontroller.* Xilinx Application Notes, `http://www.xilinx.com`, Septiembre 2003.

[190] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi y A. A. Jerraya: *A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design*, 2001.

[191] D. Lyonnard, S. Yoo, A. Baghdadi y A. A. Jerraya: *Automatic Generation of Application-Specific Architecture for Heterogeneous Multiprocessor System-on-Chip.* En *Proceedings of the Design Automation Conference (DAC'01)*, páginas 518–523, Junio 2001.

[192] H. Fu y J.W. Lockwood: *The FPX KCPSM Module: An Embedded, Reconfigurable Active Processing Module for the Field Programmable Port Extender (FPX).* Technical Report. Applied Research Laboratory Department of Computer Science. Washignton University, Julio 2001.

[193] A. Astarloa: *FFR16: First File Reader FAT16.* `http://www.opencores.org/projects/FFR16`, Agosto 2003.

[194] N. Palermo: *A 9 MHz Digital SSB Modulator.* `http://www.microtelecom.it/ssbdex-e.htm`, Septiembre 2002.

[195] MEET Electronics Ltd.: *Three Phase Motor Controller.* Xilinx AlianceCORE IP Center, Septiembre 2002.

[196] A. Astarloa, U. Bidarte, J. Lázaro, A. Zuloaga y J. Arias: *Multiprocessor SoPC-Core for FAT Volume Computation.* Microprocessors and Microsystems, in press, 2005.

[197] A. Astarloa: *FFR16: First File Reader FAT16.* `http://www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm`, Agosto 2003.

[198] Microsoft Corporation: *Microsoft Extensible Firmware Initiative FAT32 File System Specification - 1.03* . Hardware White Paper - `http://www.microsoft.com`, Deciembre 2000.

[199] Compact Flash Association CFA: *CF+ and Compact Flash Specification Revision 1.4.* `http://www.compactflash.org`, 1999.

[200] A. Ramanathan: *Development of ATA/ATAPI-5 IDE Controller Core.* Project Report: `http://www-unix.ecs.umass.edu/~aramanat/RCG/report.doc`, Deciembre 2001.

[201] P. Nanthanavoot y P. Chongstitvatana: *Development of a data reading device for a CD-ROM drive with FPGA technology.* Conference of Electrical Engineering, Thailand, 2002.

[202] ASTEK Corp.: *IP Cores: ATA-4/6 Cores.* `http://www.astekcorp.com`, 2004.

[203] Palmchip Corp.: *ATA IDE Host Controller Cores.* `http://www.palmchip.com`, 2004.

[204] ACARD Technology Corp.: *The Secrets of the Best Burning Engine.* `http://www.acard.com/eng/press/2003/engine.html`, 2003.

[205] A. Astarloa, U. Bidarte y A. Zuloaga: *A Reconfigurable SoC Architecture for High Volume and Multi-channel Data Transaction in Industrial Environments.* En *Proceedings of the International IEEE Conference on Industrial Electronics, Control and Instrumentation IECON'02*, páginas 2322–2327, Noviembre 2002.

[206] R. Herveille: *OCIDEC: Opencores IDE Controller.* `http://www.opencores.org/projects.cgi/web/ata/overview`, 2001.

[207] P. Pelgrims, D. Driessens y T. Tierens: *Evaluation report OCIDEC-case V1.0.* Technical Report. DE NAYER Instituut. `http://emys.denayer.wenk.be`, 2003.

[208] Xilinx Corp.: *MicroBlaze^{TM} Soft Processor Core.* Xilinx Processor Central, `http://www.xilinx.com`, 2004.

[209] R. A. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz y M. Muhlada: *Automating the Design of SOCs Using Cores.* IEEE Design & Test of Computers, 18(5):32–45, 2001.

[210] Y. Zorian R. K. Gupta: *Introducing Core-Based System Design.* IEEE Design & Test of Computers, 14(4):15–25, 1997.

[211] J. Resano, D. Mozos, D. Verkest, S. Vernalde y F. Catthoor: *Run-Time Minimization of Reconfiguration Overhead in Dinamically Reconfigurable Systems.* Lecture Notes in Computer Science, 2778:585–594, 2003.

[212] A. Astarloa, J. Lázaro, U. Bidarte, J. L. Martín y A. Zuloaga: *A Self-reconfiguration Framework for Multiprocessor CSoPCs.* Lecture Notes in Computer Science, 3203:1124–1126, 2004.

[213] P. Pelgrims, D. Driessens y T. Tierens: *Overview: Excalibur, Leon, Microblaze, Nios, OpenRISC, VIRTEX II PRO.* Technical Report. DE NAYER Instituut. `http://emys.denayer.wenk.be`, 2003.

[214] U. Bidarte, A. Astarloa, A. Zuloaga, J. Jiménez y I. Martinez de Alegría: *Core-Based Reusable Architecture for Slave Circuits with Extensive Data Exchange Requeriments.* Lecture Notes in Computer Science, 2778:497–506, 2003.

[215] Xilinx Corp.: *PowerPC Embedded Processor Solution.* Xilinx Hard Processors, `http://www.xilinx.com`, 2004.

[216] Altera: *Nios 3.0 CPU Datasheet.* Literature: Nios Processor, `http://www.xilinx.com`, 2004.

[217] Chris Dunlap y Tom Fischaber: *Configuring Xilinx FPGAs Using an XC9500 CPLD and Parallel PROM.* Xilinx Application Notes, `http://www.xilinx.com`, Julio 2000.

[218] Xilinx Corp.: *System ACE CompactFlash Solution.* `http://support.xilinx.com`, Abril 2002.

[219] Xilinx Corp.: *Gate Count Capacity Metrics for FPGAs.* Xilinx Application Notes, `http://www.xilinx.com`, Febrero 1997.

[220] K. Chaplin: *Reconfiguring BlockRAMs.* TechXclusives, `http://www.xilinx.com`, 2003.

[221] D. Gajski, J. Zhu, A. Gerstlauer, R. Dömer y S. Zhao: *SpecC: Specification Languaje and Methodology.* Kluwer Academic Publishers, Boston, USA, Febrero 2000.

[222] J.M.P Cardoso y H.C. Neto: *Compilation for FPGA-Based Reconfigurable Hardware.* IEEE Design & Test of Computers, páginas 65–75, Marzo 2003.

[223] Xilinx Corp.: *Xilinx Development Guide.* `http://support.xilinx.com`, Abril 2003.

[224] J. B. Stephensen: *Software-Defined Hardware for Software-Defined Radio.* QEX, páginas 41–50, Septiembre 2002.

[225] G. Youngblood: *A Software-Defined Radio for the Masses, Part 1.* QEX, páginas 13–21, Julio 2002.

[226] R. Adraka: *A survey of CORDIC algorithms for FPGAs.* En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'98)*, páginas 192–200, Febrero 1998.

[227] R. Herveille: *CORDIC core.* `http://www.opencores.org/projects.cgi/web/cordic`, 2001.

[228] J. Arias, A. Zuloaga, J. Lázaro y A. Astarloa: *Malguki: An RSSI based Ad-Hoc Location Algorithm.* Microprocessors and Microsystems, 28(8):403–409, Octubre 2004.

[229] J. Logue: *Virtex Synthetizable Delta-Sigma DAC.* Xilinx Application Notes, `http://www.xilinx.com`, Septiembre 1999.

[230] J. Logue: *Virtex Analog to Digital Converter.* Xilinx Application Notes, `http://www.xilinx.com`, Septiembre 1999.

[231] FTDI: *FT8U245AM Datasheet.* `http://www.ftdichip.com`, 2001.

[232] APERT: *Applied Electronics Research Team, Universidad del País Vasco.* `http://det.bi.ehu.es/~apert`, 2004.

[233] M. Garay, Astarloa, U. Bidarte, A. Zuloaga y J.L. Martín: *Plataforma CSoPC para la Evaluación del Sistema de Reconfiguración Parcial Dinámica Tornado*. En *Proceedigns of the Jornadas de Computación Reconfigurable y Aplicaciones (JCRA'04)*, páginas 53–61. Universidad Autónoma de Cataluña, Septiembre 2004.