UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Departamento de Ingeniería Electrónica
y Automática

Tesis Doctoral

Aportaciones a la metodología de diseño basada en Síntesis de Alto Nivel. Aplicaciones al diseño de IPs para procesado de eventos complejos y codificación de vídeo

Pedro Francisco Pérez Carballo

Noviembre 2015
Las Palmas de Gran Canaria

D. ROBERTO ESPER-CHAÍN FALCÓN, SECRETARIO DEL DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA DE LA UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA,

CERTIFICA,

Que el Consejo de Doctores del Departamento en su sesión de fecha 17 de noviembre de 2015, tomó el acuerdo de dar el consentimiento para su tramitación, a la tesis doctoral titulada "**Aportaciones a la metodología de diseño basada en síntesis de alto nivel. Aplicaciones al diseño de IPs para procesado de eventos complejos y codificación de vídeo**" presentada por el doctorando D. Pedro Francisco Pérez Carballo y dirigida por el Doctor D. Antonio Núñez Ordóñez

Y para que así conste, y a efectos de lo previsto en el Artº 6 del Reglamento para la elaboración, defensa, tribunal y evaluación de tesis doctorales de la Universidad de Las Palmas de Gran Canaria, firmo la presente en Las Palmas de Gran Canaria, a 17 de noviembre de dos mil quince.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Departamento de Ingeniería de Electrónica y Automática (DIEA)

Programa de Doctorado en Ingeniería de Telecomunicación Avanzada

Título de la Tesis:

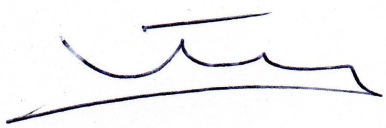
Aportaciones a la metodología de diseño basada en Síntesis de Alto Nivel. Aplicaciones al diseño de IPs para procesado de eventos complejos y codificación de vídeo

Tesis doctoral presentada por D. Pedro Francisco Pérez Carballo

Dirigida por el Dr. D. Antonio Núñez Ordóñez

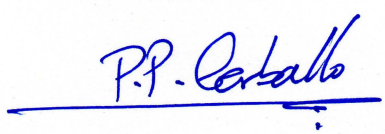
Las Palmas de Gran Canaria, 12 de noviembre de 2015

El Director



Fdo.: Dr. D. Antonio Núñez Ordóñez

El Doctorando



D. Pedro Francisco Pérez Carballo

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Tesis Doctoral

Aportaciones a la metodología de diseño basada en Síntesis de Alto Nivel. Aplicaciones al diseño de IPs para procesado de eventos complejos y codificación de vídeo

Pedro Pérez Carballo

Las Palmas de Gran Canaria, noviembre de 2015

A Carmen Amada y a Silvia …

A mis padres …

Agradecimientos

Después de varios intentos he podido completar este documento que recoge la conclusión de mi formación de doctorado. El objetivo se ha alargado más allá de lo esperado pero *nunca es tarde si la dicha es buena*. Y espero que así lo sea.

En primer lugar quiero agradecer a mi director de tesis, Antonio Núñez por haber tenido tanta paciencia para la dirección de mi tesis. A lo largo de la colaboración en distintos proyectos y trabajos, me ha dado la oportunidad de participar en proyectos de investigación a nivel europeo, nacional y local. La experiencia acumulada me ha permitido ir avanzando personal y profesionalmente. Agradezco también sus discusiones y oportunas correcciones que han ayudado a enfocar esta tesis.

Un trabajo de esta naturaleza precisa de apoyo material y humano para poder llevarlo a término. Por una parte el gran esfuerzo que hace el IUMA para disponer de entornos de diseño avanzados, a través del Servicio de Tecnologías y Herramientas (STH), que me ha facilitado de forma significativa el trabajo. Asimismo, agradezco el apoyo en tareas concretas de las personas que participan o han participado en esta aventura, que representa la adopción de nuevos métodos de diseño y tecnologías de implementación en un avance continuo. Mención especial

para aquellos que han colaborado en diferentes momentos en alguno de los temas tratados en esta tesis: Pedro, Antonio, Pablo, Carlos, Armando, Tomasz, Marcos, Adrián, Rita, Omar, Romén, Paloma, Mohamed, Alejandro, Rodrigo, David, Edgar y Benjamín y tantos otros. Igualmente ha sido clave el apoyo de los proyectos PCCMUTE, financiado por el Plan Nacional de I+D y RT-Core bajo convenio con Edosoft Factory. A los colegas de EUROPRACTICE (IMEC y RAL) que han respondido puntual y acertadamente a mis peticiones y dudas profesionales sobre las tecnologías y herramientas. A Enrique, Agustín y Luis que han proporcionado soporte para tener disponibles y a punto los entornos de trabajo, recursos claves para un proyecto de esta envergadura; con ellos he compartido muchas horas de laboratorio.

No quiero dejar pasar esta oportunidad sin agradecer a mis compañeros del IUMA esta aventura que supone participar en un Instituto de Investigación que a lo largo del tiempo ha demostrado con creces su razón de existir. Igualmente mi labor como docente me anima a superarme cada día, y la ilusión con que cada uno de los estudiantes aborda su formación personal pone en valor mi esfuerzo. Espero seguir aprendiendo de todos.

Todo esto no hubiese sido posible sin mi familia que me ha apoyado desde el comienzo. Mis padres me han respaldado desde el minuto cero y no dieron nunca un paso atrás. Mis hermanos y hermanas siempre han sido una referencia a pesar de la distancia y el mar. Igualmente al resto de la familia y amigos que me han apoyado de diversa forma.

Y no quiero terminar sin agradecer el apoyo de Carmen Amada y de Silvia por haber tenido la paciencia de soportar mis largas jornadas de trabajo en estos años. Por su constancia y apoyo tanto material como emocional. Tanto a Carmen Amada como a Silvia les admiro su afán por la lectura, persistencia con los idiomas, sus ganas de seguir aprendiendo y su valentía en abordar nuevos retos. Gracias además por los cafés y por el diseño de la portada de esta tesis.

Resumen

I. Escenario

El incremento de la capacidad de integración que se ha producido en la tecnología de semiconductores ha traído consigo un aumento en la complejidad de los sistemas electrónicos y por ende de su diseño. Esta capacidad de integración ha permitido abordar nuevos problemas que requieren alta capacidad de computación, con costes energéticos que han continuado reduciéndose, hasta el punto de poder realizarse en Sistemas en un solo chip (SoC). Actualmente, el uso de sistemas heterogéneos representa una solución generalizada al problema del diseño de SoCs. Los sistemas heterogéneos se basan en una descomposición que incluye el recurso a asignar tareas críticas de la computación a su ejecución por aceleradores específicos en *hardware*. Estos aceleradores de *hardware* se adaptan al problema con el fin de lograr resultados en tiempos de respuesta aceptables. En estos casos la síntesis de alto nivel (SAN) puede jugar un papel central, está siendo una tendencia ampliamente aceptada en la industria, y es objeto de intensa investigación.

Ya en los años 90 se adoptaron técnicas de diseño RTL, en las que los lenguajes de descripción de *hardware* (VHDL/Verilog) y la síntesis lógica desde RTL han contribuido a reducir la complejidad asociada a los problemas complejos. En la actualidad hay consenso en la comunidad científica internacional en el campo de Tecnología del Diseño en que es necesario subir el nivel de abstracción en la especificación del sistema para poder realmente aprovechar eficientemente el número de transistores disponibles. Más aún, ese nuevo nivel de abstracción es necesario para cubrir la brecha entre la creciente complejidad de los algoritmos y la gran densidad de transistores disponible. El problema planteado es el de obtener métodos y mejores prácticas que contribuyan a la eficiencia y productividad de la tecnología de diseño. Se trata de poder manejar ese muy elevado número de transistores de forma inteligente. La adopción de descripciones a nivel algorítmico y a nivel de sistema permite al diseñador centrar su atención en la funcionalidad y en la organización de la arquitectura del sistema y dejar a procedimientos automáticos guiados los detalles de la implementación. Sin embargo la excesiva automatización pretendida no siempre conduce a soluciones eficientes. La síntesis desde el nivel sistema o ESL es un campo en evolución y busca métodos para obtener implementaciones eficientes. ESL está demostrando su potencial en la exploración del espacio de diseño a nivel sistema buscando una correspondencia adecuada con el espacio de arquitecturas disponibles, pero está encontrando limitaciones en los intentos de convertir los flujos ESL en flujos de síntesis e implementación final. Un recurso disponible para ESL es la descomposición en algoritmos y la aplicación de SAN en el paradigma antes descrito de síntesis de aceleradores *hardware*.

Esta tesis se desarrolla en este escenario, y en este recurso metodológico de la descomposición del sistema en algoritmos como técnica de reducción de la magnitud del problema. Aquí la síntesis de alto nivel juega un papel central al permitir generar implementaciones RTL controladas y verdaderamente adaptadas a las condiciones de la aplicación. Si es importante en la productividad del diseño la reutilización de implementaciones *hardware* como bloques Intellectual Property (IPs), la SAN permite además introducir el concepto análogo de reutilización a nivel algorítmico. Para lograrlo ha sido un avance clave la introducción del modelado TLM al nivel de transacciones, ya que permite abordar por separado la implementación de la funcionalidad y de las interfaces. Esta estructuración del diseño está conduciendo a implementaciones más eficientes, dentro de un uso inteligente de los flujos de síntesis, y de un uso controlado y verificable de los resultados intermedios del proceso de síntesis. La utilización de lenguajes de alto nivel, tales como C/C++ y SystemC, permiten reducir la distancia semántica entre la descripción algorítmica y su implementación *hardware*. Sin embargo esta transformación requiere tener un conocimiento de las principales características de la tecnología de implementación para obtener la calidad deseable en los resultados finales, medida en términos de prestaciones (latencia y tiempo de ejecución), potencia y área (PPA), y en término de esfuerzo (tiempo y costo) de diseño.

II. Aportaciones

1. Esta tesis aporta en primer lugar un estudio de las diferentes opciones existentes para la realización de la síntesis de alto nivel. Este estudio está acompañado de la experiencia de diseño del autor en el seno del equipo de investigación en EDA en el IUMA. La tesis aporta un conjunto de recomendaciones probadas para el flujo de diseño mediante SAN,

organizadas en un conjunto de principios que hemos consolidado en numerosos diseños, un conjunto de ideas prácticas experimentales en el uso de las herramientas del flujo de diseño, y finalmente un flujo de síntesis que se ha contrastado cuantitativamente en esos diseños.

2. La tesis aporta en segundo lugar la implementación en FPGA y en ASIC de dos aplicaciones de aceleración *hardware* distantes entre sí, como casos de uso del flujo de síntesis establecido y como cuantificación, contraste y criba de las recomendaciones realizadas y obtención de recomendaciones y estrategias adicionales.

    a. En el primero de los casos se trata de una aplicación para el procesado de eventos complejos. Es una aplicación dominada por el flujo del control, es decir por la aplicación de numerosas reglas de decisión fuertemente entrelazadas y condicionadas, a partir de datos que llegan en ritmos exigentes para el procesado en tiempo real.

    b. En el segundo caso se trata de un filtro de bucle adaptativo para la decodificación de vídeo siguiendo el estándar H.264/AVC-SVC. Este bloque es de nuevo un bloque complejo y cuello de botella en la implementación de códec *hardware* del estándar. Es una aplicación dominada por el flujo de los datos e intensiva en procesamiento y en uso de memoria.

    c. Para ambos casos se ha utilizado el flujo SAN aportado en la primera parte. En este flujo se parte de SystemC como lenguaje de especificación. La descripción en SystemC se obtiene transformando y descomponiendo la especificación algorítmica de entrada. Se aplica SAN a esta descomposición y se guía la síntesis asegurando la verificabilidad de cada etapa.

    d. Para ambos problemas de referencia se ha realizado una implementación FPGA sobre una plataforma basada en un dispositivo Xilinx Virtex-5 FX, que incluye procesadores PowerPC440. Igualmente se ha realizado la implementación de ambos problemas en un IP ASIC. Para cada problema y para cada tecnología objetivo se ha seguido el flujo SAN establecido. Esta experimentación ha permitido refinar el flujo SAN utilizado y formular nuevas aportaciones sobre mejores prácticas de diseño y nuevas recomendaciones. Los resultados han sido comparados con el estado del arte y con resultados generados por otros flujos de diseño y otros enfoques.

3. Durante el desarrollo de este trabajo de investigación se han creado experiencias de uso que reflejan la necesidad de una integración vertical y sin costuras de los flujos de diseño para poder obtener una convergencia de los aspectos temporales. Obtener esta convergencia temporal se identifica como uno de los principales retos de la nueva generación de herramientas de síntesis de alto nivel. Se aportan orientaciones para explotar los recursos disponibles en las herramientas para lograr convergencia temporal.

4. Se ha observado la necesidad de mantener el nivel RTL como punto clave de la verificación del sistema, especialmente para la síntesis a partir de C/C++ donde no se dispone de información temporal ni de estructura del modelo, ya que solo el nivel RTL dispone de la información precisa necesaria tanto a nivel de precisión de ciclos de bus como de precisión

de bits y pines, para validar la implementación realizada durante la fase de síntesis de alto nivel.

5. Igualmente se han evaluado otros aspectos claves que afectan a la implementación del sistema, como puede ser la organización del modelo en cuanto a partición y jerarquía, la arquitectura de la memoria y la estructura de los datos. Igualmente se han tenido en cuenta aspectos relacionados con la ejecución paralela de funciones y bucles, y el control de los protocolos de comunicación.

6. El flujo de SAN para el diseño se ha organizado teniendo en cuenta los requisitos de integridad de los datos y facilidad de uso, requeridos para cubrir la brecha entre la poca productividad de los flujos de diseño y la productividad de la gran densidad de transistores disponible. La utilización de scripts basados en Tcl ha permitido definir estrategias de uso para cada caso concreto lo que facilita la reproducibilidad de resultados, y su portabilidad a distintas tecnologías. Todos los resultados se aportan como contribuciones a la creación de flujos SAN eficientes para diseños complejos.

Abstract

The increased capacity for circuit integration in semiconductor technology has brought about an increase in the complexity of electronic systems and therefore in their design. This integration capability has helped to address new problems requiring high computing capacity with ever reduced energy costs. Currently, the use of heterogeneous systems are a generalized solution. These *hardware* accelerators are adapted to the problem to be addressed in order to achieve results within acceptable response times. In these cases, high-level synthesis (HLS) can play a central role.

In the 90s, RTL techniques were adopted, including *hardware* description languages (VHDL/Verilog) and logic synthesis. They have helped to reduce the complexity associated with these designs. At the present time the design community has established the need to raise the level of abstraction in the system specification, in order to take advantage of the number of transistors available. The adoption of algorithmic descriptions and system level descriptions

allows the designer to focus on the functionality and architecture of the system, and leave to guided automatic procedures the details of its implementation.

This thesis is developed in this scenario, where high-level synthesis plays a central role, generating RTL implementations adapted to the conditions of the application. This introduces the concept of IP reuse at algorithmic level. The introduction of transaction level modelling (TLM) is key to this approach, separating implementation of functionality from interfaces. The use of high-level languages, such as C/C++ and SystemC, reduces the gap between the algorithmic description and *hardware* implementation. However the transformations required to have a knowledge of the main features of the implementation technology for a desirable quality of results in terms of performance (latency and throughput), power and area (PPA), and design effort (time and cost).

This thesis, after a study of the different options for performing high-level synthesis, in which the main contributions are indicated, presents a design flow based on high-level synthesis for hardware accelerators, and it is validated using two applications belonging to opposite application domains (control driven and dataflow driven). The first one is an application for complex event processing. The second, a deblocking filter for H.264/AVC-SVC video decoder. In both cases, the high-level synthesis flow starts from a model done in SystemC language, obtained from the transformation of algorithmic specification, and applying the HLS flow established. For both problems we have targeted FPGA implementations of a platform based on a Xilinx Virtex-5 FX device, including a PowerPC440 Hard IP, as well as IP ASIC implementations.

During the development of this research work, we have captured and extracted user experiences that reflect the need for a seamless vertical integration of the design flows to obtain a proper timing closure for performance, as well as a reduction in time and effort in the design cost. It is also applicable to objectives of power and area. This vertical integration is currently one of the main challenges of the new generation tools in high-level synthesis.

We observed the need to maintain the RTL level as a key point of the HSL and verification flow, especially for the synthesis from C/C++, where no temporal information nor model structure is available yet. In a C/C++ description, the necessary details in terms of bus cycles and bit and pin accurate signals, only available at RTL level, are included during the high-level synthesis. Other key issues affecting the implementation of the system were also evaluated, as the organization of the model in terms of partition and hierarchy, memory architecture and structure of the data. The recommendations also take into account issues related to function and loop parallelization, and issues concerning control of the communication protocols. The HLS design flow is organized taking into account aspects of data integrity and ease of use as required to bridge the gap between Design productivity and Semiconductor productivity (transistors density). By using Tcl scripts it is possible to define strategies to reproduce the synthesis results.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de acrónimos

0-	
µP	Microprocesador
A	
ABL	*A Block diagram Language*
ABLED	*A Block diagram Language Editor*
ABS	*Anti-lock Braking System*
ACE	*Advanced Configuration Environment*
ADN	Ácido DesoxirriboNucleico
AET	*Algo Execution Trading*
ALU	*Arithmetic Logic Unit*
AMBA	*Advanced Microcontroller Bus Architecture*
ANSI-C	*American National Standards Institute C standard*
API	*Application Programming Interface*
APU	*Auxiliary Processor Unit*
ARM	*Advanced RISC Machine*
ARP	*Address Resolution Protocol*

ASIC	*Application-Specific Integrated Circuit*
ASM	*Algorithmic State Machine*
ASSP	*Application Specific Signal Processor*
AT	*Algorithmic Trading o Algo Trading*
AVC	*Advanced Video Coding*
AXI	*Advanced eXtensible Interface*
AXI3	*Advanced eXtensible Interface* versión 3
AXI4	*Advanced eXtensible Interface* versión 4
AXI4-LITE	*Advanced eXtensible Interface* versión 4 modalidad *Lite*
AXI4S	*Advanced eXtensible Interface* versión 4 modalidad *streaming*
B	
BC	*Behavioral Compiler*
BCA	*Bus Cycle Accurate*
BRAM	*Block RAM*
BS	*Boundary Strength*
BSV	*Bluespec SystemVerilog*
C	
CA	*Cycle Accurate*
CABA	*Cycle-Accurate Bit-Accurate*
CAD	*Computer Aided Design*
CDC	*Clock Domain Crossing*
CDFG	*Control Data Flow Graph*
CE	Coprocesador de Eventos
CF	*Compact Flash*
CHP	*Customizable Heterogenuos Platform*
CI	Circuito Integrado
CIF	*Common Intermediate Format*
CLB	*Configurable Logic Block*
CMOS	*Complementary Metal–Oxide–Semiconductor*
CMU	*Carnegie Mellon University*
CONLAN	*CONsensus LANGuage*
CPU	*Central Processing Unit*
CSELT	*Centro Studi e Laboratori Telecomunicazioni*
CtoS	*Cadence C-to-Silicon Compiler*
CX	Ciclo aproXimado
D	
DC	*Design Compiler*
DCR	*Device Control Register*
DCT	*Digital Cosine Transform*
DDR	*Double Data Rate*
DDR2	*Double Data Rate* segunda generación
DF	*Deblocking Filter*
DHCP	*Dynamic Host Configuration Protocol*
DMA	*Direct Memory Access*
DPB	*Decoded Picture Buffer*
DSE	*Design Space Exploration*

DSP	*Digital Signal Processor*
DUT	*Device Under test*
DVFS	*Dynamic Voltage and Frequency Scaling*
DVI	*Digital Visual Interface*
E	
E/S	Entrada/Salida
EDA	*Electronic Design Automation*
EDIF	*Electronic Data Interchange Format*
EEUU	Estados Unidos
ELF	*Executable and Linkable Format*
EOF	*End Of Frame*
EOP	*End Of Payload*
EPLF	*Ecole Polytechnique Fédérale de Lausanne*
EPR	Elementos de Procesamiento Reconfigurables
ESL	*Electronic System Level*
F	
FF	*Flip-flops*
FFT	*Fast Fourier Transform*
FIFO	*First In, First Out*
FIR	*Finite Impulse Response*
FPGA	*Field Programmable Gate Array*
FSM	*Finite-State Machine*
FU	Factor de Utilización
G	
GaAs	*Gallium Arsenide*
GALS	*Globally Asyncronous Locally Synchronous*
GPGPU	*General Purpose computing on Graphics Processing Unit*
GPMC	*General Purpose Memory Controller*
GPP	*General Purpose Processor*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
H	
HDL	*Hardware Description Language*
HLS	*High-Level Synthesis*
HPC	*High Performance Computing*
HTG	*Hierarchical Task Graph*
HW	*Hardware*
HW/SW	*Hardware/Software*
I	
ICMP	*Internet Control Message Protocol*
ICON	*Integrated Controller*
IDaSS	*Interactive Design and Simulation System*
IDE	*Integrated Development Environment*
IDEA	*International Data Encryption Algorithm*
IES	*Incisive Enterprise Simulator*
IETR	*Institut d'Électronique et de Télécommunications de Rennes*

ILA	*In-Circuit Logic Analyzer*
ILP	*Instruction Level Paralelism*
IMEC	*Interuniversitair Micro-Elektronica Centrum*
IOB	*Input Output Block*
IoT	*Internet of Things*
IP	*Intellectual Property*
IR	*Intermediate Representation*
ISA	*Instruction Set Architecture*
ISO	*International Organization for Standardization*
ISP	*Instruction Set Processor*
ISPA	*Instruction Set Processor Architecture*
ISPL	*Instruction Set Processor Language*
ISPS	*Instruction Set Processor Specification*
ITRS	*International Technology Roadmap for Semiconductors*
IUMA	Instituto Universitario de Microelectrónica Aplicada
IUS	*Incisive Unified Simulator*
J	
JM	*Joint Model*
JTAG	*Joint Test Action Group*
K	
KARL	*Kaiserslautern Register Transfer Language*
L	
LAB	*Logic Array Block*
LEF	*Cadence Library Exchange Format*
LIS	*Latency Insensitivy System*
LL	*Low Leakage*
LLVM	*Low-Level Virtual Machine*
LSI	*EPFL Laboratoire des Systèmes Intégrés*
LUT	*LookUp Table*
M	
M1	Metal 1
MAC	Multiplicación/Acumulación
MB	Macro Bloque
MCI	*Memory Controller Interface*
MIPS	*Microprocessor without Interlocked Pipeline Stages*
MMU	*Memory Management Unit*
MPI	*Message-Passing Interface Standard*
MPI-3	*Message-Passing Interface Standard versión 3.0*
MPLB	*Master PLB*
MPSoC	*Multiprocessor System-on-Chip*
MPU	*Microprocessor Unit*
N	
NASDAQ	*National Association of Securities Dealers Automated Quotations*
NCD	*Xilinx Native Circuit Description*
NEC	*Nippon Electric Company*
NGC	*Xilinx Netlist Native Generic Compiler*

O	
OMAP	*Texas Instruments Open Multimedia Applications Platform*
OpenMP	*OpenMP API*
OpenSVC	*Open Scalable Video Coding*
OSCI	*Open SystemC Initiative*
OSI	*Open System Interconnection*
P	
P&R	*Placement and Routing*
PA	*Pin Accurate*
PB	*Picture Buffer*
PCAP	*Packet CAPture*
PCCMUTE	*Power Consumption Control in MUltimedia Terminals*
PDP	*Programmed Data Processor*
PLB	*Processor Local Bus*
PMS	*Processor Memory Switch*
PPA	Prestaciones, Potencia, Área
Q	
QCIF	*Quarter CIF*
QoE	*Quality Of Experience*
QoR	*Quality of Results*
R	
RAL	*Rutherford Appleton Laboratory*
RAM	*Random Access Memory*
RAMB	*BRAM*
RC	*RTL Compiler*
RGB	*Red Green Blue*
RMM	*Reuse Methodology Manual*
ROCCC	*Riverside Optimizing Compiler for Configurable Circuits*
RT	*Real Time*
RT-CORE	*Real Time - Core*
RTL	*Register Transfer Level*
S	
SAN	Síntesis de Alto Nivel
SDC	*Synopsys Desing Constraints*
SDK	*Software Developer Kit*
SDRAM	*Synchronous Dynamic Random-Access Memory*
SIF	*Synthesis Internal Format*
SIMD	*Single Instruction Multiple Data*
SL	Síntesis Lógica
SLEC	*Sequential Logic Equivalence Checking*
SoC	*System-on-Chip*
SOP	*Start Of Payload*
SP	*Standard Performance*
SPICE	*Simulation Program with Integrated Circuit Emphasis*
SPLB	*Slave Processor Local Bus*
SPMD	*Single Program Multiple Data*

SRAM	*Static Random Access Memory*
SRFF	*Synchronous Reset Flip Flop*
STH	Servicio de Tecnologías y Herramientas del IUMA
STR	Sistemas en Tiempo Real
SVC	*Scalable Video Coding*
SW	*Software*
T	
Tcl	*Tool Command Language*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TEMAC	*Tri-mode Ethernet MAC*
TFT	*Thin-Film Transistor*
TIC	Tecnologías de la Información y las Comunicaciones
TLF	*Timing Library Format*
TLM	*Transaction-Level Modeling*
TPB	*Temporal Picture Buffer*
TRS	*Term Rewriting System*
TTM	*Time To Market*
U	
UART	*Universal Asynchronous Receiver-Transmitter*
UMC	*United Microelectronics Corporation*
USB	*Universal Serial Bus*
V	
VCD	*Value Change Dump*
VCS	*Verilog Compiled Simulator*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VLSI	*Very-large Scale Integration*
VP	*Virtual Prototype*
X	
XML	*eXtensible Markup Language*
XPS	*Xilinx Platform Studio*
XST	*Xilinx Synthesis Technology*
Y	
YUV	*YUV Color Space*

Capítulo 1. Introducción

1.1. Planteamiento del problema

La necesidad de cómputo seguirá creciendo en los próximos años para resolver problemas de procesamiento complejos que hasta ahora no se han podido solucionar. Entre ellos se encuentran el estudio del clima y sus modelos, la gestión de las redes de energía, la variación del genoma humano, la astrofísica, la dinámica de los océanos o la gestión del tráfico en las grandes ciudades [1]. Otra clase de grandes problemas actuales deriva del incremento del número de dispositivos con conectividad a internet mediante conexiones inalámbricas, que crece a un 30% de tasa interanual hasta alcanzar 4.900 millones de dispositivos en 2015 y del orden de 25.000 millones en 2020 según un estudio realizado por Gartner [2] en lo que se denomina de forma genérica Internet de la cosas (IoT). En ambas clases, ya sea por los propios modelos de cada sistema o por sus interacciones, se genera un volumen importante de datos cuyo procesamiento requiere sistemas electrónicos de computación adaptados al problema en cuestión y sistemas de comunicación especializados. La red europea de excelencia *High Performance and Embedded Architecture and Compilation* (HiPEAC) [3] en su propia denominación y en su serie de estudios sobre la hoja de ruta previsible en ese campo, señala los

desafíos que se afrontan. Con frecuencia tanto el diseño de sistemas para computación de altas prestaciones como para sistemas empotrados necesitan incluir circuitos específicos o aceleradores. La consecuencia directa que afecta a las metodologías de diseño es disponer de métodos y herramientas que sean capaces de abordar estas necesidades.

Uno de los factores que afectan al coste de un sistema electrónico es su tiempo de desarrollo debido, por una parte, a la naturaleza de los productos obtenidos y, por otra, a los estrictos plazos que impone el mercado. En este sentido existen numerosas iniciativas tanto en la industria como en los centros de investigación y en la Academia que tratan de acortar el tiempo de diseño aumentando el nivel de abstracción y paralelizando tareas durante su desarrollo. A lo largo del tiempo se ha ido elevando el paradigma de modelado pasando desde la captura del *layout* (geometrías), al modelo eléctrico de transistores y puertas lógicas, a la transferencia entre registros, y hasta puros modelos funcionales o de comportamiento. La abstracción del diseño se ha demostrado como uno de los métodos más efectivos para controlar la complejidad y aumentar la productividad. Por ejemplo para un diseño de 1M de puertas se requieren unas 300K líneas de código RTL frente a las 30K-40K de un diseño modelado a nivel de comportamiento en C, C++ o SystemC. Esto supone una reducción de 7x-10X [4]. Este tipo de alternativas aprovechan mejor la gran capacidad de integración en silicio disponible.

La tendencia actual es integrar en un único sistema en chip (SoC) la mayor funcionalidad posible, debida a esa capacidad del silicio. La integración en un solo chip de tanta funcionalidad tiene limitaciones por la potencia que es posible disipar. Esta dependencia es cuadrática con la tensión y lineal con la frecuencia de operación. El aumento de prestaciones viene entonces dado no por un mayor aumento de la frecuencia de operación sino por la paralelización de los procesos de cómputo. En la figura 1 se muestra la tendencia de los modelos predictivos del *International Technology Roadmap for Semiconductors* (ITRS) [5] sobre la complejidad de los sistemas en cuanto a elementos de procesamiento, memoria y lógica incluida. Como se aprecia la tendencia es hacia un crecimiento exponencial en todos los aspectos citados. En este caso se trata de datos para productos portables, pero la tendencia indicada es similar en otros campos.

Todo este proceso de integración lleva consigo un incremento de coste de los proyectos que ha sido modelado por el IRTS [5]. Como se aprecia en la figura 2, el coste del diseño ha ido creciendo de forma significativa a lo largo del tiempo. Igualmente se muestra el efecto de la introducción de las diferentes herramientas CAD de diseño sobre el modelo de costes del SoC. Diferentes propuestas metodologías, fruto del trabajo de investigación en herramientas y métodos de automatización del diseño electrónico (EDA), han ido introduciendo cambios que han aportado puntos de inflexión y avances significativos en los flujos de diseño hacia una disminución de esos costes derivados del incremento de complejidad del diseño. En la figura 2 se resaltan en líneas verticales las novedades más significativas en tecnologías de diseño y su efecto de disminución del coste.

Junto con las herramientas CAD es cada vez más necesario considerar el creciente papel del *software* empotrado en el SOC final (figura 3). La disponibilidad de núcleos procesadores, procesadores programables, y gran cantidad de memoria disponible en los SoC, reclama ahora la gestión del *software* en los SoC. Es necesario por tanto introducir nuevos métodos de diseño que generen puntos de inflexión que frenen los costes.

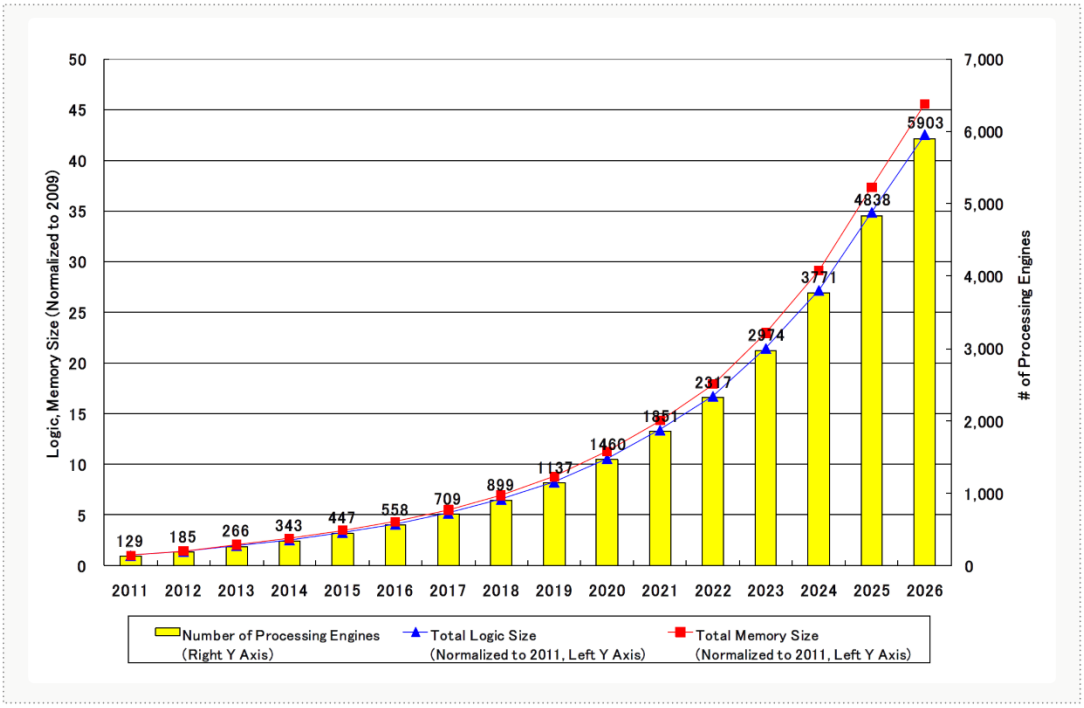


Figura 1. Crecimiento de la capacidad de integración de un SOC según el ITRS

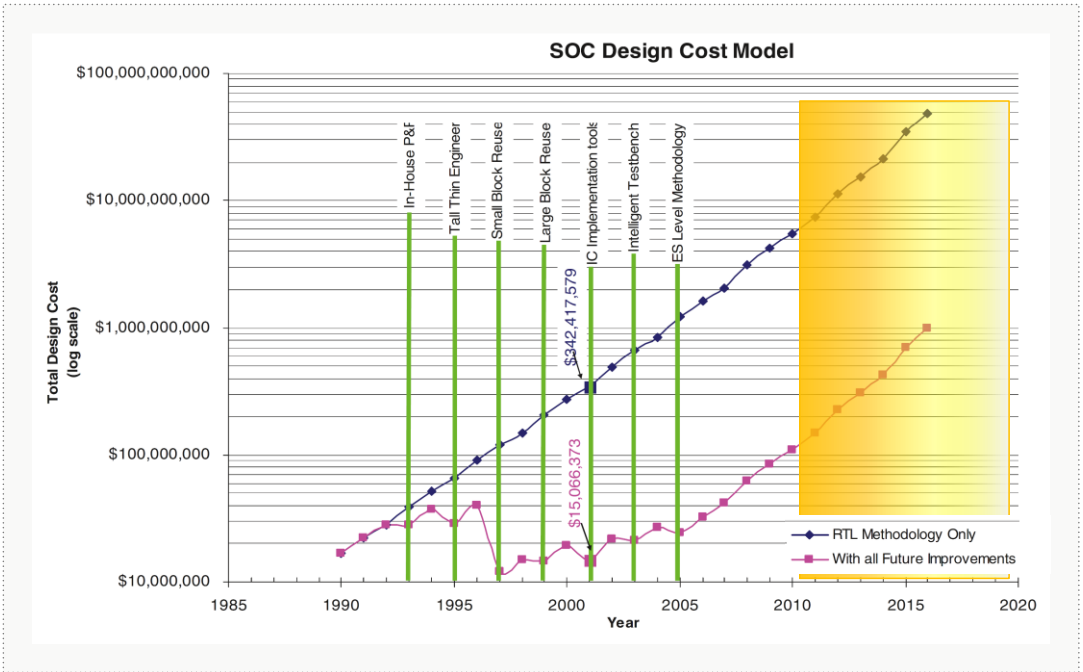


Figura 2. Evolución de los costes del CI y del efecto de la introducción de las herramientas de diseño (adaptada de [6])

El impacto de la ley de Moore y su evolución hacia propuestas *More* Moore (más densidad FinFET/CMOS) y *More than* Moore (más funciones, incluso MEMS) hace que durante la fase del diseño del sistema y su exploración la diversidad funcional que se puede incluir en el diseño se incremente de forma notable, por lo que para manejarla es necesario introducir niveles de abstracción que hagan de puente entre los niveles muy altos, a nivel de requisitos, y aquellos otros cuya misión sea la implementación electrónica directa de un bloque funcional, como es

por ejemplo el nivel RTL. La figura 3 muestra la gran brecha de diseño *software* que ha aparecido, y que se suma a la gran brecha existente entre la productividad del diseño hardware (herramientas CAD y diseño EDA) y la capacidad de integración.

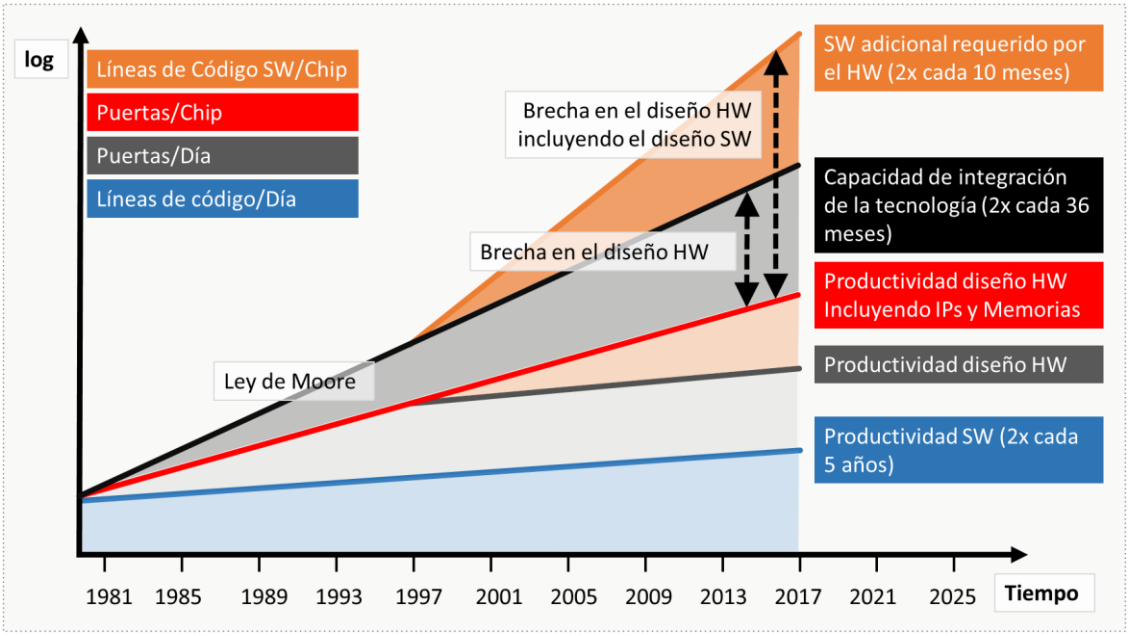


Figura 3. Evolución de la capacidad de integración y del diseño (adaptada de [7])

Ante este escenario ha surgido la integración de IPs específicos y memorias como medio de incrementar la productividad del diseño HW y reducir la brecha de diseño. Y la integración de más procesadores. Combinando ambas estrategias, los modelos de arquitectura de procesamiento que se proponen para la integración en un SoC son heterogéneos (figura 4), estando formados por uno o varios núcleos microprocesadores de propósito general, una unidad de E/S y un conjunto de elementos de procesamiento específico comunicados por un sistema de interconexión en chip que puede ser desde un simple bus, hasta una red conmutada en el chip, pasando por una arquitectura de interconexión con una variedad de buses en chip con capacidad de gestión de las transferencias [8][9]. Una consecuencia inmediata es la necesidad de crear diferentes esquemas de sincronización.

El diseño de estos elementos de procesamiento con funcionalidad diferenciada se realiza a partir de su funcionalidad o algoritmo. La transformación de estos algoritmos se realiza mediante síntesis de alto nivel SAN, apoyándose en el modelado a nivel de transacciones TLM para obtener la implementación final del sistema. En el diagrama en V de la figura 5 se indica la ubicación de la síntesis de alto nivel en el flujo de síntesis, partiendo de la especificación del diseño. La SAN enlaza con el diseño más tradicional basado en RTL. Por encima de RTL se suele considerar el *front-end* del diseño. Por debajo de RTL estamos en las etapas del *back-end*. Esta tesis no realiza aportaciones en los niveles de síntesis ESL (ni de exploración del espacio de diseño a niveles aún superiores). Vamos a suponer que en esos niveles se han hecho ya las adecuadas particiones del problema en tareas y éstas en tareas HW y en tareas SW. Sin embargo respecto al *back-end* las aportaciones que se hacen en esta tesis en SAN conducen en muchos casos a señalar las consecuencias que se derivan para la síntesis lógica y la síntesis física, así como para la verificación.

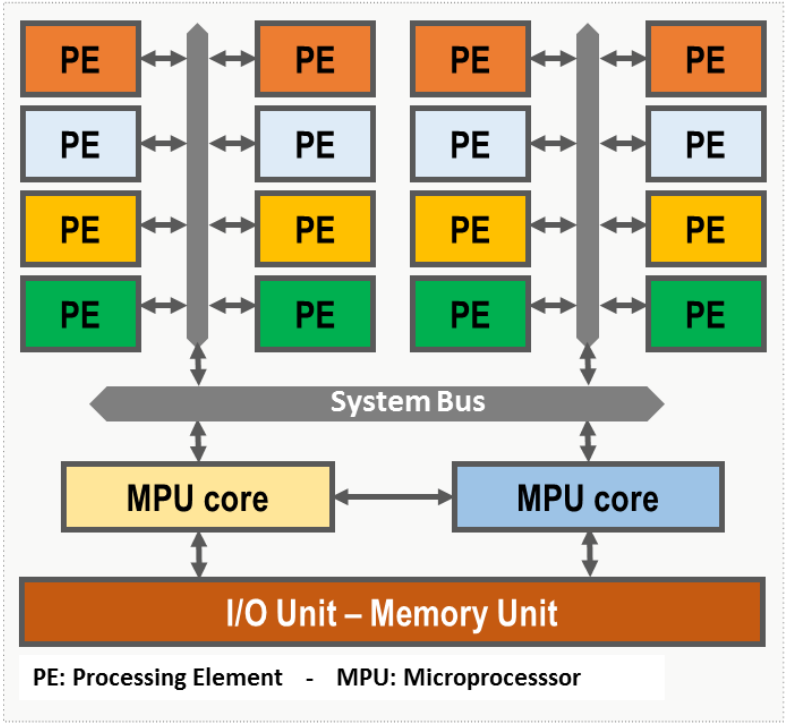


Figura 4. Ejemplo de sistema de procesamiento heterogéneo (adaptada de [5])

Hemos resaltado ya la necesidad de disponer de IPs que se modelen a nivel de comportamiento para mejorar la productividad de diseño HW. Este modelado a nivel comportamiento de los IPs hace que puedan adaptarse a diferentes casos de aplicación según las necesidades y aporta gran flexibilidad y productividad al proceso de diseño. EL nivel de reutilización de los IPs ahora se desplaza hacia una mayor abstracción que el nivel RTL, pero se requiere que todo este camino esté guiado por un conjunto de especificaciones y restricciones funcionales y de rendimiento, impuestas al diseño, acordes al nivel de abstracción, para obtener calidad en los resultados. El riesgo a evitar es perder la calidad natural de la síntesis RTL. Por esta razón todo el proceso de síntesis ahora ampliado al alto nivel debe ser acompañado de técnicas de verificación que permitan asegurar el correcto comportamiento funcional y temporal del sistema, tal como se describe en el esquema en V de la figura 5.

En estos niveles de abstracción, con la introducción de uno o varios núcleos microprocesadores se aporta aún otro nivel más de flexibilidad a los sistemas mediante *software* empotrado, tal como se indicó en figura 3. Para desarrollar *software* empotrado de forma productiva se están desarrollando modelos de alto grado de abstracción del sistema, tales que permitan hacer prototipado virtual del sistema. Una plataforma virtual VP (o VSP) es un conjunto de modelos del sistema empotrado que facilita el desarrollo del *software* empotrado en etapas tempranas del diseño. En esta plataforma se separa el desarrollo del diseño hardware del diseño *software* mediante la correspondiente interfaz API, que facilite la iteración del *software* empotrado, primero con la plataforma virtual o modelo de simulación abstracto durante etapas tempranas de diseño, y después con la plataforma real, una vez se ha completado su diseño hardware. La metodología de prototipado virtual está fuera del alcance de esta tesis. El lector puede encontrar más información en prototipado virtual en [10].

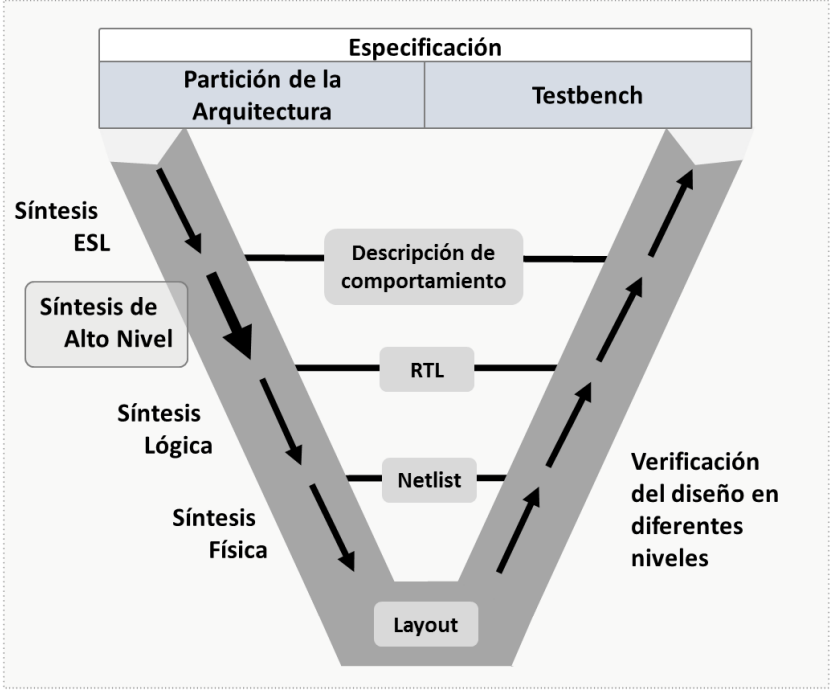


Figura 5. Diagrama en V del proceso de diseño electrónico (derivada de [7])

Podemos entonces justificar la necesidad de abordar un trabajo de investigación en técnicas de diseño basadas en síntesis de alto nivel SAN que permita extraer un conjunto de buenas prácticas asociadas a este cambio en el paradigma de diseño desde niveles RTL al diseño desde niveles algorítmicos y resaltar y cuantificar ventajas y posibles riesgos.

Este trabajo se centra en la síntesis del sistema hardware a partir de descripciones algorítmicas que serán transformadas mediante síntesis de alto nivel hacia su implementación en diferentes tecnologías, ya sean de tipo FPGA o ASIC. La clase de algoritmos a estudiar es la que corresponde a un elemento de procesamiento específico (PE) entre los integrados en los sistemas con procesadores heterogéneos, o aceleradores. Concretamente se parte siempre de una descripción de alto nivel del elemento de procesamiento que será integrado en el acelerador hardware, se define un flujo de diseño y se pasa a las fases de implementación e integración en la plataforma final.

La definición del flujo de diseño SAN se basa en tomar como objetivo de salida no directamente el silicio sino el nivel RTL. En varios trabajos ([11][12][13] [14][15][16]) se propone el concepto de flujo de diseño *Platform Based Design* o PBD, como el proceso para diseñar sistemas mapeando una aplicación del espacio de aplicaciones de entrada a una arquitectura de sistema procesador del espacio de arquitecturas de salida, tomando como punto intermedio la definición de plataforma de procesador al nivel *Instruction Set Architecture* (ISA), basando por lo tanto el sistema en la familia definida por su ISA. El mismo grupo extendió en estos trabajos el concepto PBD a otros campos incluido el diseño de circuitos desde una descripción algorítmica, tomando como punto intermedio precisamente el nivel RTL. Gráficamente este concepto ha sido capturado y expresado por un dibujo de “reloj de hora de arena” de la síntesis, una X (figura 6).

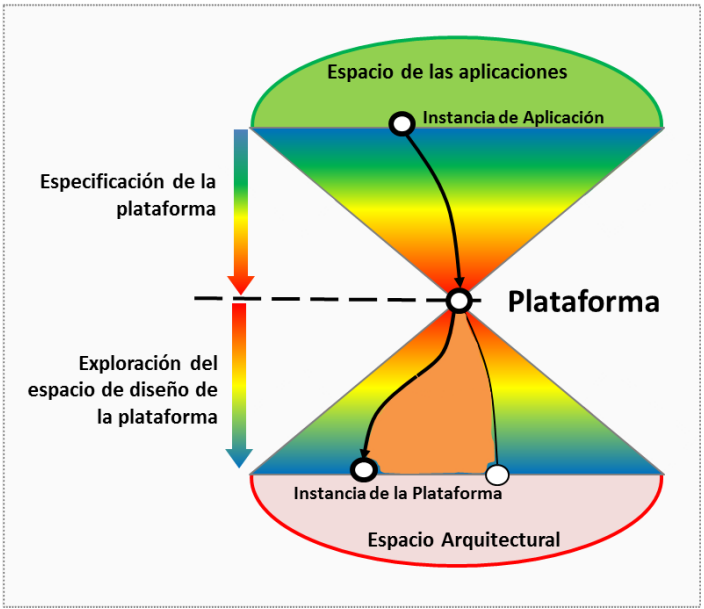


Figura 6. Diseño basado en plataformas – PBD (adaptada de [17])

En este trabajo reforzamos y argumentamos la idea de extender esta X a la etapa anterior a RTL, es decir a la SAN. Para ello el espacio de los algoritmos se mapea al espacio RTL pasando por un punto intermedio definido por la descomposición que se efectúa al nivel TLM. Es lo que denominamos flujo de síntesis en "Doble X". Las razones para la estandarización de ese punto intermedio TLM y para requerir a las herramientas su generación se aportan a partir de la experiencia de diseño del autor y del grupo de diseño del IUMA en el que trabaja, y a partir del estudio de las herramientas más actuales y su crítica y la evolución del estado del arte, y a partir de las dos experiencias adicionales de diseño que siguiendo esta metodología se presentan en esta tesis.

Este proceso de diseño "Doble X" se aplica a dos casos de uso, de los que se extraen recomendaciones adicionales, validación y cuantificación. Con ellos se cierra esta tesis. El primero es un procesador de eventos y el segundo es un filtro de los macrobloques en un decodificador H.264/AVC-SVC, concretamente el *deblocking filter* (DBF). Ambos casos son circuitos complejos, pero pertenecen a dominios muy distintos. El primero es un circuito *control driven*, dominado por el control. El segundo es un circuito *data driven*, dominado por los datos. Ambos circuitos son exigentes en memoria y en velocidad de ejecución. Ambos circuitos aceleradores hardware se diseñan con consideraciones también de bajo consumo.

Volviendo a las aplicaciones y recapitulando, podemos subrayar que la utilización de aceleradores que incluyen una FPGA como plataforma de implementación e integración es cada vez más atractiva desde el punto de vista de la computación de altas prestaciones (HPC). La utilización de estas plataformas basadas en FPGA permite reconfigurar el *blade* de cómputo haciéndolas adaptables al problema en ejecución. Esa adaptabilidad está basada en metodologías de síntesis de alto nivel. Un ejemplo es el caso de Microsoft que introduce este tipo de aceleradores en sus centros de cálculo [18], motores de búsqueda en Internet, para reducir la latencia en encontrar la respuesta o para aumentar el *troughput* (tasa de resultados

producidos), tal como se muestra en la figura 7. En este caso se trata de 1.632 servidores con FPGAs ejecutando el servicio de *ranking* de páginas de Bing (~30,000 líneas de C++).

Este tipo de aplicaciones HPC así como las aplicaciones en sistemas empotrados intensivos en computación, está produciendo que en el mercado de las FPGAs de última generación se esté utilizando cada vez más la síntesis de alto nivel. Por otro lado el mercado ASIC siempre ha requerido como entrada estable la descripción RTL. Este punto se mantiene en nuestra argumentación sobre el flujo XX. Además el mercado ASIC necesita el prototipado y depurado con emuladores de millones (miles de millones) de puertas lógicas. Estos emuladores en el *back-end* se construyen con FPGAs que reciben mapeados los diversos componentes del diseño implementado. La SAN de esos componentes en esas plataformas es también un caso de aplicación importante.

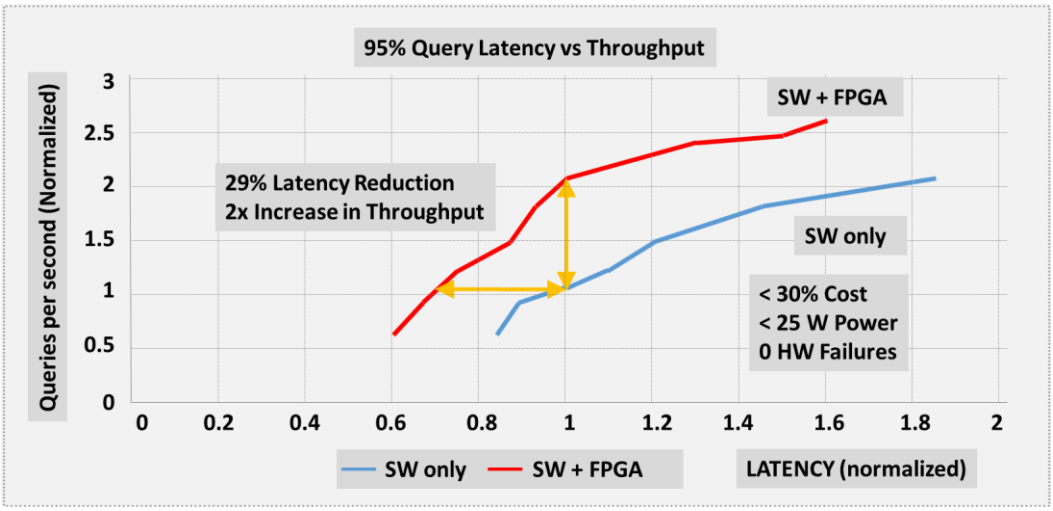


Figura 7. Reducción de los tiempos de búsqueda en la plataforma Bing de Microsoft (adaptada de [19])

1.2. Motivación de la investigación

El panorama internacional muestra una tendencia al uso extensivo de aceleradores hardware y de SoC heterogéneos en los que se incluyen aceleradores hardware (bloques modelados al nivel algorítmico que se implementan usando arquitecturas optimizadas y a medida del problema a resolver). Según ITRS hay una previsión del crecimiento del número de aceleradores presentes en un SoC con un incremento hacia más de 1000 en el año 2020 [5]. El diseño de estos aceleradores por tanto será un aspecto clave en la implementación del sistema de cara a obtener las prestaciones deseadas.

La síntesis de alto nivel ha sido objeto de interés en las últimas décadas según se muestra en [20] y se expone en detalle en el capítulo 2. Ello ha traído como consecuencia, no sin varios fracasos en el camino, el establecimiento de flujos de diseño de un sistema hardware desde niveles de abstracción superiores al RTL. Probablemente el mayor riesgo ha sido pretender un salto muy directo y muy grande, excesivamente automatizado.

a) Existen varios motivos que justifican la necesidad de abordar esta investigación en SAN de algoritmos hacia bloques IP para aceleradores hardware (FPGA o ASIC). En primer lugar el

diseño de sistemas complejos, con un alto grado de complejidad algorítmica no es abordable ya con las técnicas de diseño disponibles a nivel RTL. La partición del sistema, la definición de interfaces precisas a nivel de bits, el establecimiento de protocolos de comunicación a nivel de ciclo, entre otros aspectos, requieren largos procesos de verificación antes de tratar de generar el código RTL. No hacerlo conduce a la necesidad de rediseños, y a la pérdida del control de lo que está ocurriendo con la síntesis. En definitiva a la pérdida de productividad en diseño de hardware, a un mayor tiempo al mercado, y a un mayor coste. Estos procesos de verificación en alto nivel basados en simulación precisan de la creación de entornos de test complejos, laboriosos y propensos a errores con la posibilidad de no cubrir todos los casos de test necesarios. Durante años se ha tratado de crear técnicas que ayuden a solventar estos problemas a los diseñadores tratando de evitar que la verificación se convierta en el cuello de botella del diseño de un sistema.

Por ello uno de los primeros aspectos que justifican la creación de un punto intermedio en el modelado de alto nivel es la de simplificación de su verificación, utilizando lenguajes con mayor contenido semántico como pueden ser C/C++ o SystemC. Igualmente es necesario adoptar metodologías que soporten el nivel de transacciones TLM [21] y la síntesis o transformación desde el nivel algorítmico. La adopción de este tipo de modelado está justificada si el modelo intermedio creado (primera X del flujo SAN) se puede transformar mediante técnicas de síntesis en el diseño final.

La creación de modelos hardware a nivel de transacciones, en los que se separa de forma natural las interfaces de comunicación y su comportamiento, facilita el refinamiento de ambas partes del diseño por separado, poniendo énfasis en cada una de ellas en diferentes momentos en el proceso de diseño. De esta manera se puede mantener la interfaz de comunicación con niveles de abstracción altos cuando el foco de diseño se centra en la funcionalidad del mismo, disminuyendo la sobrecarga de protocolos de señalización entre componentes durante la simulación. De igual manera se puede mantener un modelo funcional del sistema mientras se pone énfasis en depurar los protocolos de comunicación. Esta separación de comunicación y funcionalidad permite cubrir de forma eficiente dos fases posteriores del proceso de diseño: la creación de la plataforma y su refinamiento, la segunda X del flujo propuesto.

TLM facilita la creación de modelos de simulación a nivel de transacciones. En ellos los accesos de los buses del sistema se hacen mediante operaciones de lectura y escritura. Podemos encontrarnos con dos tipos de modelos: sin referencia temporal (*untimed*) y con información temporal (*timed*). En el primero de los casos se describe la arquitectura del sistema. En el segundo se incluyen detalles temporales a nivel de microarquitectura.

Por otra parte, los modelos RTL incluyen información temporal precisa a nivel de señales. Por ello su conexión a los buses del sistema, modelados a nivel TLM, se realiza mediante adaptadores.

La transformación desde un modelo funcional (de entrada) a un nivel TLM (punto intermedio) y desde TLM a un modelo RTL es un paso estratégico en el flujo de diseño SAN. Implica la definición de la microarquitectura del sistema en chip y por tanto tiene impacto directo en las prestaciones finales del mismo.

Esta técnica de síntesis de alto nivel ha sido objeto de investigación durante décadas y en los últimos años ha tomado gran auge por diversas causas. Por una parte, la utilización en determinados dominios de aplicación concretos. Por otra su aplicación para el mapeado directo de algoritmos sobre plataformas FPGA que aceleren problemas complejos, en lo que se conoce como *hardware in the loop*. Se ha demostrado que desde el punto de vista del diseño de aceleradores hardware, los lenguajes HDL (VHDL/Verilog) no son aceptables para este tipo de soluciones, y requieren el uso de lenguajes y flujos de diseño basados en C/C++/SystemC. La inclusión de información tecnológica y restricciones directas impuestas por la tecnología, incrementa la calidad de los resultados. La utilización de síntesis de alto nivel supone entonces una reducción en el esfuerzo de diseño, aumentando la productividad y reduciendo el riesgo de diseño.

Es necesario integrar el modelo RTL obtenido en el flujo SAN con el resto de los módulos de la plataforma para poder realizar su verificación conjunta. Esta integración suele ser una tarea compleja y sujeta a errores. Por ello es conveniente automatizar esta tarea de tal manera que se obtenga una representación formalmente equivalente y verificable del conjunto.

Otro de los aspectos claves para poder abordar la complejidad y justificar la abstracción del modelo es la reutilización del diseño.

Ambos factores –verificación y reutilización– contribuyen de forma notable a incrementar la productividad del diseñador. La síntesis de alto nivel añade valor a los bloques de propiedad intelectual (IP) ya que facilita su reutilización en diferentes escenarios, en los que los parámetros como latencias y protocolos de comunicación se puedan variar sin esfuerzo adicional de diseño [22].

b) Una vez esbozado este escenario, es evidente el interés de toda metodología de diseño que incorpore un flujo de diseño basado en síntesis de alto nivel que, apoyado por distintas herramientas de diseño existentes, y siguiendo criterios metodológicos y buenas prácticas permita realizar la síntesis eficiente del sistema.

Diversos trabajos han identificado la síntesis de alto nivel como un elemento central de las metodologías de diseño más amplias basadas en ESL. Tanto SAN como ESL favorecen un salto en la productividad en el diseño de ASIC y FPGA a partir de especificaciones funcionales, ya sea sin información temporal o parcialmente temporal (protocolos), desde C/C++ o SystemC. Esta síntesis puede ser optimizada teniendo en cuenta los requerimientos temporales, las prestaciones, potencia y coste de un determinado sistema [23][24][25][26].

Todo este planteamiento no está exento de problemas prácticos a abordar debido a la naturaleza del lenguaje utilizado. La identificación de estos problemas y su solución práctica es una cuestión fundamentalmente experimental. Y como tal su resultado consiste en estrategias, recomendaciones, avisos, y guías de mejores prácticas. Uno de los primeros problemas que nos encontramos en la síntesis basada en C/C++ es la secuencialidad del lenguaje. El comportamiento intrínseco del hardware que implica concurrencia y tiempo se añade mediante librerías o *pragmas*. Igualmente se añade precisión a nivel de bits mediante librerías de tipos de datos especiales. Es necesario tener en cuenta que, en cualquier caso, el diseñador necesita entender y conocer la arquitectura de referencia que va a implementar.

El cambio de paradigma desde modelado RTL al modelado en alto nivel, aplicando las hipótesis indicadas, supone un reto para el diseñador hardware.

Las ventajas más destacables que aporta esta metodología se pueden resumir en los siguientes puntos:

- Mayor rapidez en la creación del diseño.
- Reducción en el número de líneas de código, por ejemplo 800 líneas de código C++, incluyendo 300 líneas para el *testbench*, generan 13.000 líneas de código RTL, en 10 minutos.
- Reducción en el tiempo de simulación y depurado, incrementando la cobertura de código y por tanto con menos errores. Ello implica más tiempo disponible para la exploración del espacio de diseño, lo que redunda en una mejora en la calidad de resultados en menos tiempo.
- Posibilidad de incluir el modelo en plataformas virtuales y de prototipado virtual, para verificar el sistema completo.
- Refinamiento incremental del diseño, separando las interfaces de su funcionalidad.
- Exploración de diferentes alternativas de la arquitectura del IP, analizando los resultados de síntesis en término de prestaciones, potencia y área (PPA).
- Cosimulación RTL/alto nivel, reutilizando el entorno de test.

En la figura 8 se muestra un flujo simplificado de síntesis del diseño, dejando para los tres capítulos posteriores la incorporación de detalles adicionales de transformación, síntesis y verificación, la introducción de puntos intermedios extensión del concepto de diseño basado en plataformas, y de las necesarias realimentaciones.

1.3. Objetivos

Como se ha indicado el objetivo principal de esta tesis es ofrecer una metodología de diseño basada en síntesis de alto nivel y un conjunto de mejores prácticas de diseño SAN, que permita que, partiendo de una descripción en alto nivel algorítmico, y mediante el uso de C/C++/SystemC, haga posible su transformación pasando por el nivel RTL -plenamente integrado- y su implementación, aplicando el flujo o flujos de diseño requeridos ya sea para un dispositivo FPGA como para un ASIC o un SoC.

El flujo de diseño debe cumplir requisitos tales como facilidad de creación del modelo sin requerir transformaciones costosas en el código original, verificación del modelo para asegurar que se sintetiza un modelo funcionalmente correcto, verificación del modelo RTL obtenido, calidad de los resultados en términos de prestaciones, potencia y área, soporte de los recursos tecnológicos existentes en las tecnologías de implementación (memorias, bloques de granularidad alta como DSPs, etc.), y facilidad para su integrada con los flujos existentes por la utilización de estándares EDA consolidados.

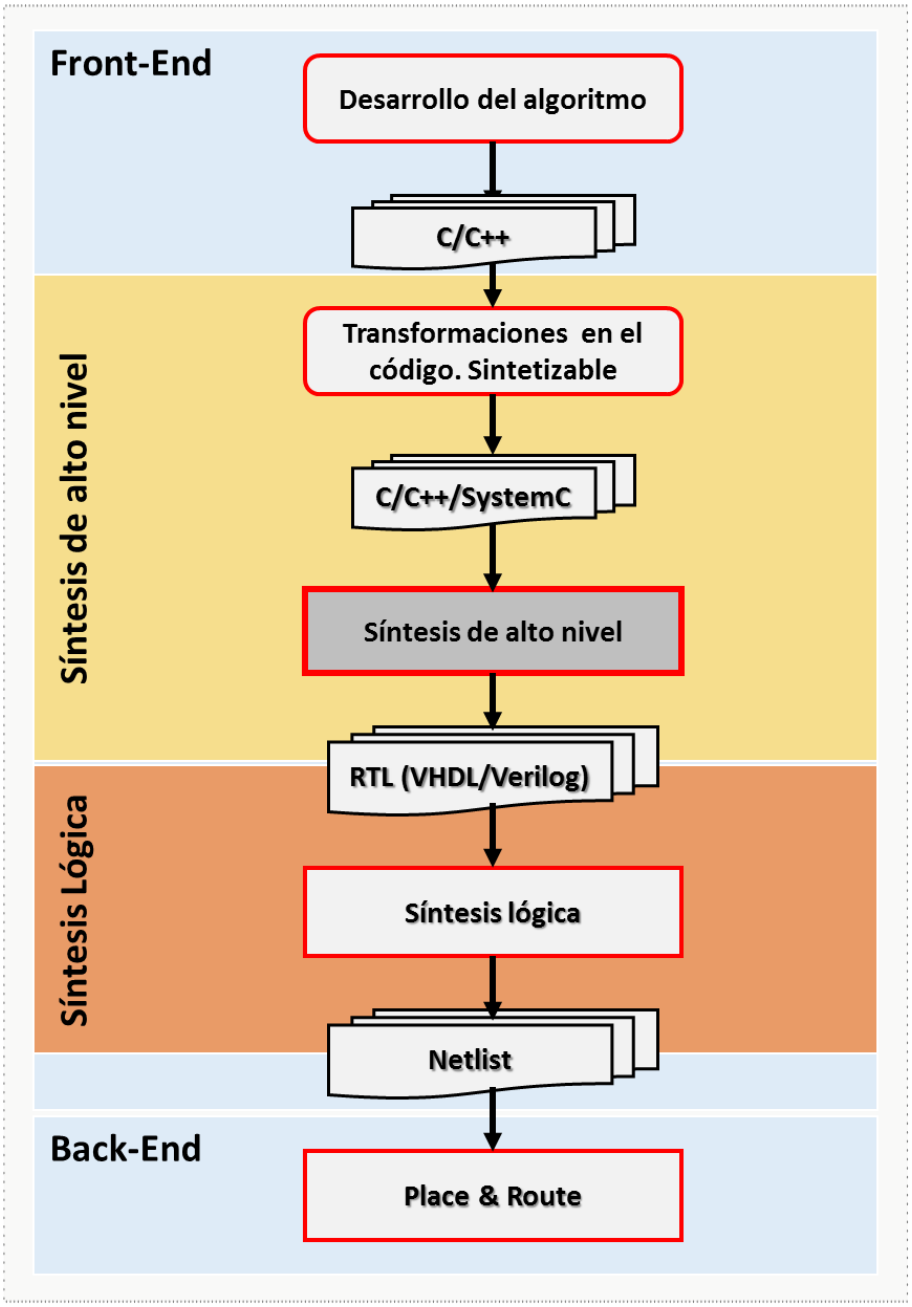


Figura 8. Flujo de síntesis del diseño

Este flujo de diseño debe ser validado. Una completa validación y crítica es la que provenga de su amplio uso. Sin embargo es necesario validarlo inicialmente. Al menos -tal y como se ha comentado- como medio para enriquecer desde la experiencia de estos diseños adicionales la propuesta de mejores prácticas obtenidas a lo largo de los años y de los diseños realizados, así como del estudio de los casos de uso de las herramientas más avanzadas. Se ha aplicado el flujo de síntesis y las recomendaciones a dos casos de uso: aceleración de una aplicación de procesamiento de eventos complejos y una aplicación de decodificación de vídeo. Para ambos casos se establecen los principios de uso del flujo, directrices de síntesis y estrategias que faciliten el diseño y verificación del sistema.

Por tanto se pretende cubrir los siguientes objetivos operativos:

1. Desde un punto de vista del modelado del sistema, definir las ideas generales de referencia con el suficiente grado de abstracción para facilitar su reutilización y reducción de los tiempos de desarrollo. Estas ideas están soportadas sobre lenguajes estandarizados y ampliamente aceptados tales como C/C++ y SystemC
2. Desde un punto de vista metodológico, establecer un conjunto de tareas y restricciones que permitan ofrecer una metodología de síntesis de alto nivel para la implementación en FPGA y ASIC/SoC a partir de descripciones algorítmicas en un lenguaje de alto nivel tal como los mencionados anteriormente.
3. Desde un punto de vista metodológico, propiciar la reutilización de bancos de test desde verificaciones algorítmicas hasta el nivel RTL, favoreciendo el incremento de la productividad y minimizando la posibilidad de errores introducidos por la intervención manual. En este mismo objetivo se tendrán en cuenta aspectos tales como las modificaciones temporales introducidas por la herramienta de SAN.
4. Identificar y establecer puntos singulares en el espacio de diseño, a tener en cuenta durante la implementación. Se trata de bloques que ofrecen discontinuidad en el espacio de soluciones (*hard macros*, memorias, bloques DSP). Identificar procedimientos y soluciones ya sean a medida o de tipo general.
5. Identificar y establecer mecanismos de continuidad en la transformación de restricciones de diseño desde alto nivel hasta restricciones de implementación física.
6. Establecer la metodología de análisis necesaria para comparar los resultados obtenidos, a diferentes niveles de abstracción, principalmente basados en parámetros PPA y latencias asociadas. Estos resultados se comparan al menos para dos implementaciones, una en FPGA y otra en ASIC.
7. Validación del flujo de diseño sobre demostrador para el caso de la implementación sobre FPGA. Y validación del flujo de diseño para ASIC/SoC en términos de *timing closure* y verificación post-*layout* (los circuitos ASIC no se fabrican por razones económicas).

1.4. Organización de la tesis doctoral

El trabajo desarrollado en esta tesis doctoral se ha estructurado en cinco capítulos, tal como se describe a continuación:

Capítulo 1. Introducción.

Se presenta el planteamiento del problema que ha motivado la realización de esta tesis doctoral, se esbozan las soluciones que se van a desarrollar y se detallan los objetivos de la misma. Para ello se da una visión global de las necesidades de capacidad de diseño para abordar el diseño de sistemas en chip (SoC) necesarios para la solución de problemas complejos actuales, en el ámbito de soluciones heterogéneas, en las que se integran aceleradores hardware adaptables a diferentes dominios de aplicaciones (IoT, HPC, Vídeo...). Se presentan los métodos de diseño de alto nivel más amplios que SAN, y se centra

el problema en la Síntesis de Alto Nivel (SAN) como la alternativa al diseño directo RTL. Se concreta el problema de la transformación algoritmo-RTL y se apunta el esquema de "Doble X" que se desarrolla en la tesis, introduciendo un primer punto intermedio de diseño TLM y un segundo punto intermedio RTL, en el camino hacia el hardware final. Se anticipan las ventajas en productividad del diseño, consistencia de la verificabilidad, flexibilidad, portabilidad, y reutilización no sólo como IP hardware sino también como IP a nivel de comportamiento.

Capítulo 2. Trabajo previo, estado actual del arte y propuestas prácticas.

En este capítulo se da una visión global de la SAN, presentando la estructura general de toda herramienta de síntesis de alto nivel, un análisis de los hitos y las aportaciones consolidadas en la evolución histórica de las principales herramientas de SAN, juntamente con anotaciones de experiencias del autor, obtenidas durante su participación en diferentes proyectos y trabajos de investigación relacionadas con esta tesis doctoral. Igualmente se presentan los retos que deben abordar las herramientas de SAN para su total integración en el flujo de diseño, poniendo énfasis en su integración con la verificación del diseño y la calidad de los resultados obtenidos. A continuación se presenta una relación de mejores prácticas en flujos de síntesis de alto nivel, estructurada en principios, ideas prácticas, recomendaciones y estrategias generales. Ente estas estrategias y en el flujo de síntesis resultante se resalta el modelo "Doble X" que mantiene el nivel RTL e introduce el nivel TLM como puntos intermedios que ayudan eficientemente al progreso de la síntesis de los algoritmos. Se describen sus ventajas y los riesgos inherentes a otros flujos que intentan prescindir de estos puntos. El capítulo se cierra con un capítulo de conclusiones.

Capítulo 3. Caso de aplicación: diseño de un procesador de eventos.

Este es un capítulo central de la tesis donde se explica en detalle el proceso de síntesis de alto nivel, según el flujo propuesto y las recomendaciones aportadas, de un acelerador hardware para una aplicación de procesado de eventos desarrollada como parte de un proyecto más amplio realizado en el IUMA. Se explica la arquitectura de referencia, el flujo de diseño de SAN desarrollado, su conexión con flujos de implementación para FPGAs y se expande a la implementación hacia un ASIC. Se demuestra la validez del flujo explicado, integrando el IP obtenido en una plataforma FPGA y validando su diseño sobre una plataforma Xilinx ML-510 que incluye un dispositivo Virtex-5 FX, así como un ASIC en tecnología UMC CMOS 65nm.

Capítulo 4. Aplicación al diseño de un filtro de bucle adaptativo para H.264/AVC-SVC.

En este otro capítulo central de la tesis se extiende la validez de la metodología al diseño de un *Deblocking Filter* de un decodificador H.264/AVC-SVC desarrollado en el IUMA para el proyecto PCCMUTE. Igual que en el caso anterior, se valida la aplicación del flujo y la guía de mejores prácticas mediante

la realización de un prototipo sobre una plataforma FPGA Xilinx ML507 que soporta un dispositivo Virtex-5 FX como en el caso anterior, y un ASIC en tecnología UMC CMOS 65nm.

Capítulo 5. Conclusiones y trabajo futuro.

Se presentan las conclusiones del trabajo de investigación y se enumeran posibles trabajos futuros que se han anotado como necesarios durante el desarrollo de esta tesis doctoral.

Capítulo 2. Trabajo previo, estado actual del arte y propuestas prácticas

2.1. Introducción

El Diseño Electrónico es una tecnología, un método, unos procedimientos, unas herramientas y un arte. Es parte del proceso de creación intelectual, de concepción de una idea, y de cómo llevarla a la práctica mediante los principios físicos y procedimientos de ingeniería propios del campo. La Síntesis de Alto Nivel (SAN) ha aparecido en el camino de las innumerables experiencias de hacer diseños en la industria y la universidad, al ritmo de la evolución de las herramientas EDA y del reto de hacer posible el diseño abstrayendo la complejidad creciente de las implementaciones. Se trata de reducir las brechas entre la falta de productividad humana y la riqueza creciente que ha surgido por “arriba y por abajo en la escala de abstracción”, es decir tanto el crecimiento de riqueza y complejidad en la idea algorítmica, como el crecimiento del número de transistores disponibles en Silicio.

La expresión Estado del Arte se aplica en este contexto en su más directa acepción. Cómo está el arte actual del diseño, con qué herramientas se cuenta, qué aportaciones previas se han consolidado y se han establecido en herramientas actuales, qué aportaciones han fallado, cuáles son los retos pendientes, y qué lecciones ha aprendido la industria, es decir han aprendido los equipos de diseñadores durante el camino. Sin una visión evolutiva es muy difícil poder ver en las herramientas actuales tanto sus rasgos sobresalientes como sus carencias.

En este capítulo pretendemos aportar esa visión evolutiva y critica desde la visión que el autor tiene como testigo de algunos aspectos de esa evolución. En virtud de esa experiencia acumulada, decantada, y de éste breve meta-estudio o estudio de estudios, se pretende poder aportar un conjunto de recomendaciones de método, recomendaciones y elecciones sobre el uso de herramientas, y aportaciones sobre las mejores prácticas elaboradas en el grupo y personalmente, en ese proceso de maduración en el arte del diseño en problemas complejos.

Se presenta por tanto una visión del estado actual de las herramientas de síntesis de alto nivel y de las actividades de diseño asociadas. Se presentan igualmente los lenguajes utilizados para la captura, modelado y simulación de diseños, y el impacto de la elección del lenguaje en el estilo de síntesis y verificación a utilizar. Se hace especial énfasis en las características de las herramientas SAN de Synopsys (BC +DC), Cadence (CtoS, Cynthesizer, Stratus), Mentor Graphics (Catapult). Y se adopta la posición de contrastar las características destacadas en su comercialización técnica, con las características de la experiencia en el uso real, tanto directas como referenciadas.

Para poder subrayar las características sobresalientes, aquellas consolidadas, y las deficiencias de las herramientas, en donde quede patente la potencialidad, la eficacia, y las mejores prácticas de la SAN, se elabora una visión evolutiva que trata de poner en contexto los avances obtenidos y el sentido de las recomendaciones que se hacen en esta tesis.

Se presentan en primer lugar los conceptos establecidos en SAN (2.2), luego la visión evolutiva con la experiencia acumulada en el grupo y por el autor como diseñador y como consultor de muchos diseños realizados desde el STH del IUMA con una gran variedad de herramientas –lo que denominamos trabajo previo (2.3)– y también en las secciones sobre la situación actual en SAN y la hoja de ruta que puede preverse (2.4), dedicando una sección monográfica a los retos actuales de la SAN (2.5), campos objetos que están abiertos a la investigación.

Con este punto de partida se hace una reflexión sobre los flujos de diseño y las mejores prácticas disponibles, y se aportan otras nuevas (2.6) que cierran el capítulo, resumiendo finalmente estas a modo de conclusiones.

En los capítulos siguientes 3 y 4 se aplican estas mejores prácticas en cuanto al flujo a dos casos que tomamos como referencia, opuestos el uno al otro, para una vez más poner bajo escrutinio las recomendaciones y mejores prácticas y confirmar unas, desechar otras y aportar otras nuevas. Este escrutinio se hace apoyando las afirmaciones en las evidencias cuantitativas que arrojan los diseños, así como su comparativa con otros diseños o con la experiencia acumulada en los años de trabajo previo. El capítulo de conclusiones recoge el resultado de esta

criba metodológica, la identificación de deficiencias y las propuestas de nuevos avances. En fin se dará una visión desde el punto de vista de la calidad de la experiencia (QoE) obtenida.

2.2. Conceptos establecidos en síntesis de alto nivel

En términos generales, la síntesis es la traducción de una descripción del comportamiento (con frecuencia en lenguaje natural o algorítmico, o funcional, o con formulación matemática o analítica de "solución de problemas") en una representación estructural, donde cada componente de la descripción estructural es a su vez definido por su propia descripción de comportamiento [27] que en principio ha quedado rebajada en nivel de complejidad. El proceso de síntesis implica un refinamiento del sistema ya que se añade la información detallada requerida para el siguiente nivel (inferior) de abstracción. Este diseño con un nivel de detalle incrementado según se desciende en nivel de abstracción, debe satisfacer las restricciones de diseño proporcionadas en la primera descripción, restricciones que capturan limitaciones del entorno del diseño pretendido (tiempo, área, potencia por ejemplo).

En el contexto de esta tesis distinguiremos la SAN de otro nivel superior el ESL (*Electronic System Level*). Este último incluye la organización de una aplicación en tareas, la planificación de tareas, la descomposición HW/SW y su mapeado en arquitecturas, la asignación de tareas a elementos de proceso, y la sincronización entre tareas. La SAN parte de los resultados de exploración del espacio de diseño –o contribuye a esa exploración– y parte también de la descomposición del problema o aplicación en tareas. Este nivel ESL excede a los objetivos de la tesis aunque se ha aportado a diversos trabajos del grupo [23][28][29][30][31][32] en ese campo. Igualmente distinguiremos la SAN del nivel inferior o de Síntesis Lógica (SL) y del nivel de implementación física, aunque el tratamiento que hacemos señala como veremos que los procesos de SAN y de SL deben realizarse en estrecha cooperación [33].

La síntesis de alto nivel se denomina también síntesis del comportamiento (por ser la entrada el comportamiento, a nivel de una tarea o equivalente) o síntesis a nivel arquitectural [34] (por ser la salida una estructura arquitectural). Es la etapa o actividad que mapea tarea algorítmica en una arquitectura. La SAN es un paso en la evolución de las tecnologías de diseño que busca incrementar la productividad, moviendo las decisiones de diseño a niveles más altos de abstracción, tal como se recoge por ejemplo en el diagrama en Y de Gajski y Kuhn [35] (figura 9).

En el diagrama en Y se presenta el alto nivel como la abstracción de nivel superior a RTL y la síntesis de alto nivel como la transformación desde el dominio de comportamiento en alto nivel al nivel RTL. Con esta formulación de partida podemos decir por tanto que el principal objetivo de la SAN es la obtención de la microarquitectura del sistema electrónico planteada en base a unidades de procesamiento activadas por unidades de control asociadas al procesamiento.

La síntesis de alto nivel mejora la productividad del diseño automatizando el refinamiento desde el nivel algorítmico hacia RTL, de tal forma que las funciones escritas en un lenguaje de alto nivel (C/C++/Matlab) se transforman y detallan mediante un conjunto de operaciones bien

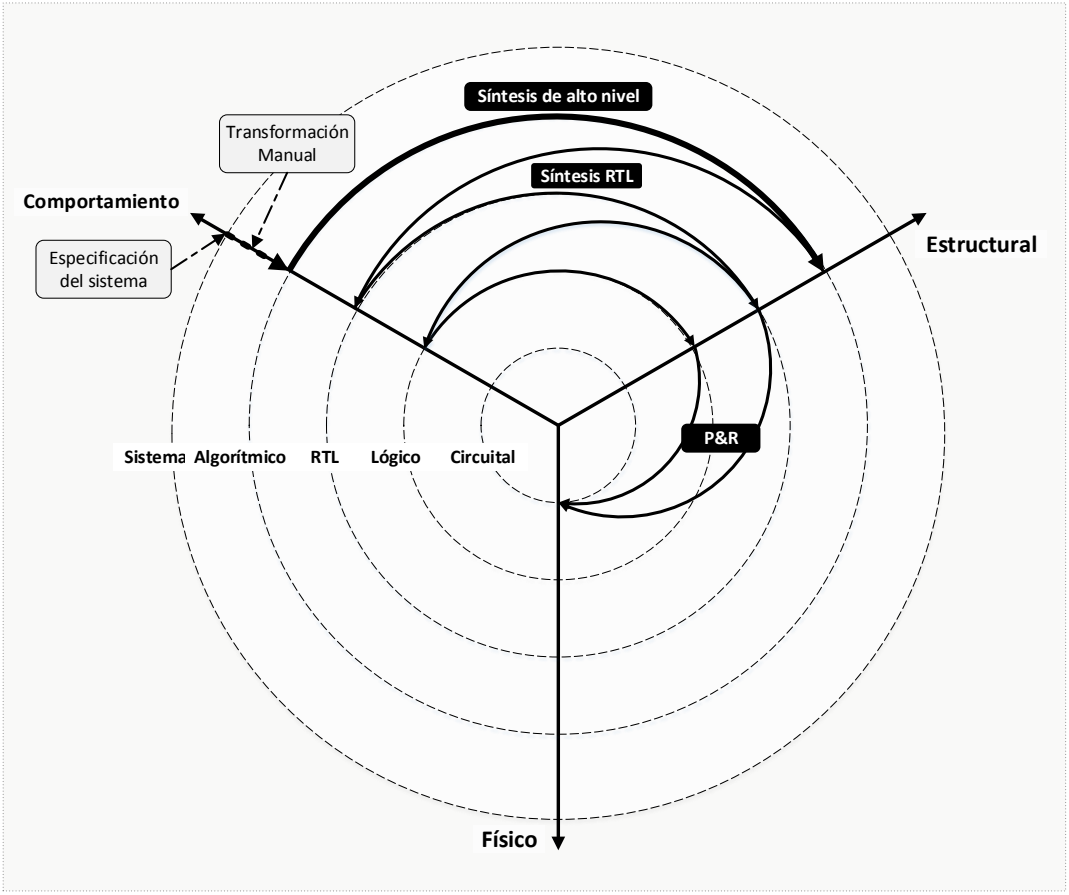


Figura 9. Diagrama en Y ubicando la síntesis de alto nivel

definidas, entre las que se encuentran las siguientes operaciones básicas plenamente establecidas (figura 10):

- Análisis del código y optimización (*lexical analysis*) para crear la estructura de representación de alto nivel generalmente formada por un Programa Ejecutable, una Descripción Ejecutable, una Máquina de Estados Algorítmica Generalizada (ASM), un Grafo del Flujo de Datos y del Control (CDFG), un Grafo de Especificación Ejecutable o un Grafo Jerárquico de Tareas (HTG) [36]
- Asignación de operadores y de variables (*allocation*), identificando el tipo de operadores y el tipo de almacenamiento necesarios, y asignación de cada variable y estructura de datos a determinada memoria o registro, así como el uso agrupado o potencialmente compartido de esas unidades (*binding* y *sharing*)
- Planificación temporal *(scheduling*) de tal forma que cada operación se asigna a uno o varios pasos de control o ciclo de reloj
- Asignación de cada operación a una unidad funcional determinada (*binding* y *sharing*)
- Síntesis de los elementos de interconexión y multiplexores de los que resulta el *datapath*, la ruta de datos o cauce de procesamiento del flujo de esos datos; y las necesidades de control del cauce, el flujo del control

- Segmentación del cauce o de las unidades, y adaptación de la estructura de control
- Visión clave del *datapath* como una estructura mixta formada por lógica combinacional de procesamiento conectada con registros cuyos valores se leen, trasfieren y escriben una vez procesados. Es la creación de la visión de nivel RTL. La SAN queda establecida como una síntesis que genera esta estructura RTL
- Síntesis de interfaces de Entrada/Salida

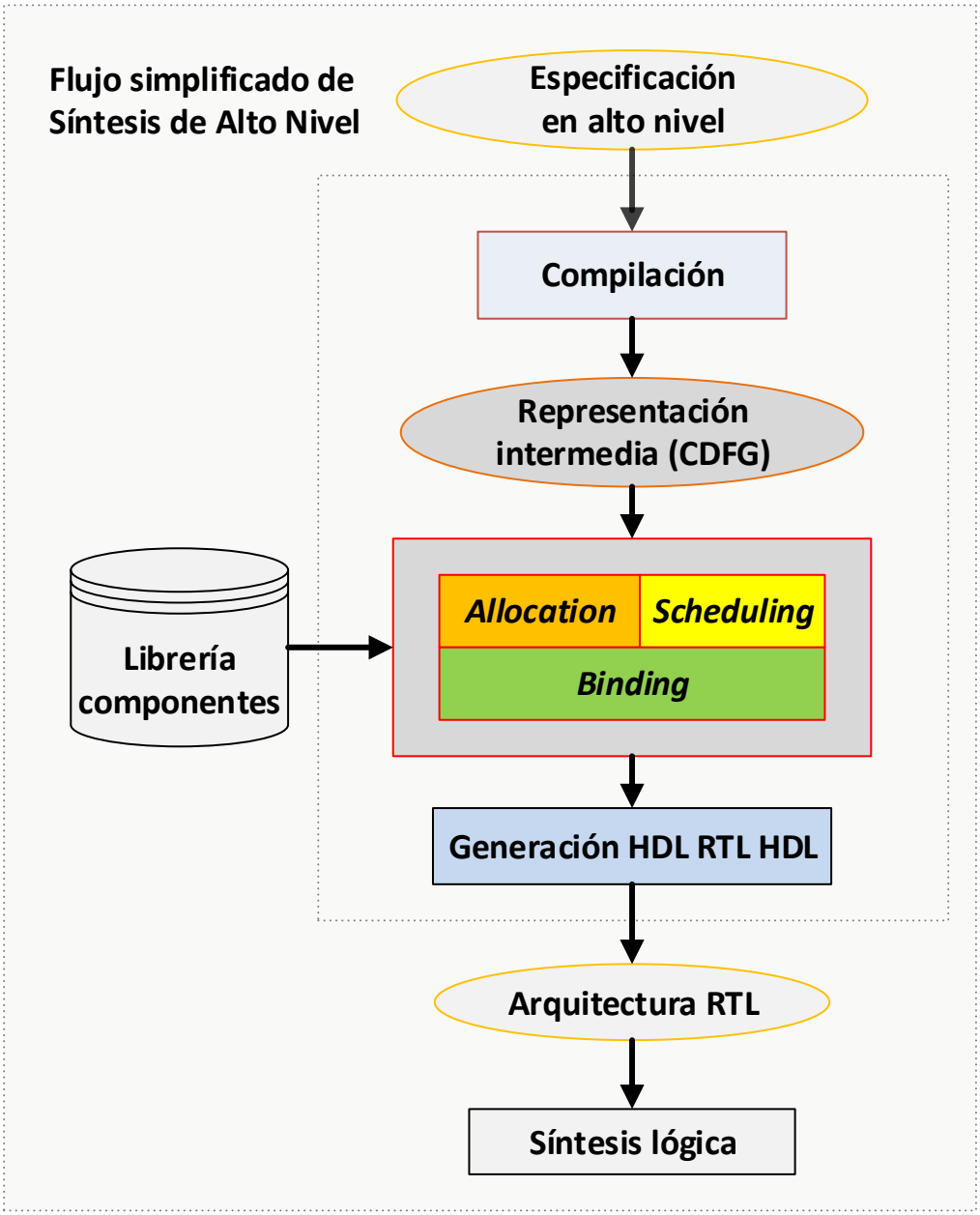


Figura 10. Flujo de síntesis de alto nivel

En todo caso, un entorno de síntesis de alto nivel debe proporcionar tres funciones básicas: mapear el comportamiento del sistema a su estructura, estimar y analizar las prestaciones del sistema con objeto de tomar decisiones y verificar la concordancia de la representación de alto nivel con la representación detallada obtenida.

Los parámetros que guían la toma de decisiones se obtienen normalmente de la utilización de librerías de operadores. En concreto se extrae información de área, retardos y potencia que se anotan en la herramienta de síntesis de alto nivel para elegir una u otra implementación del operador en función de objetivos y las restricciones del diseño. Esto permite realizar una exploración más amplia del espacio de diseño ya que las distintas variaciones arquitecturales tienen un impacto mayor sobre las prestaciones del diseño final que aquellas que se realizan más tarde y a un nivel más bajo: desde la síntesis lógica y la generación de máscaras del diseño físico (figura 11).

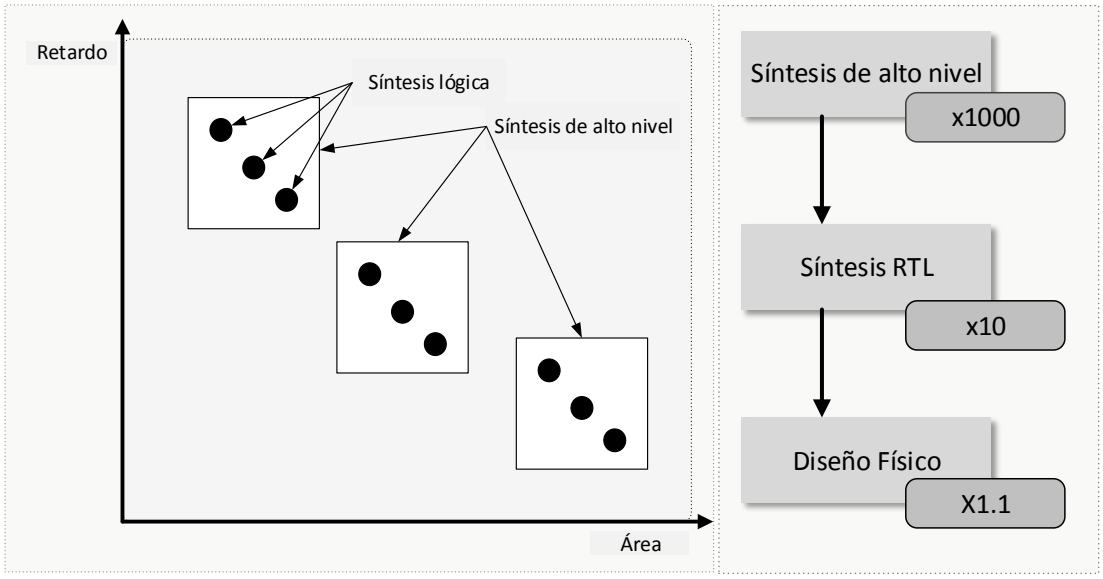


Figura 11. Efecto de la SAN en la exploración del espacio del diseño

Las herramientas de SAN facilitan también la exploración de las diferentes soluciones arquitecturales al utilizar diferentes técnicas tales como el desenrollado total o parcial de bucles, su segmentación, definición de diferentes arquitecturas de memoria, definición del paralelismo requerido, entre otras.

Otro de los aspectos claves del éxito de la SAN en la industria es la facilidad de la verificación del diseño, reduciendo el esfuerzo necesario y reutilizando la infraestructura de verificación durante el proceso de diseño. Y no menos importante es la facilidad que ofrece para la migración tecnológica del diseño, ya sea modificando la solución arquitectural en función del cumplimiento de las restricciones impuestas y la capacidad de la tecnología ya que una solución elegida para una determinada FPGA puede no ser adecuada para otra o para una tecnología ASIC y viceversa. En un proyecto los factores limitantes en la práctica industrial para la exploración de dichas soluciones son el tiempo y coste del proyecto.

Podemos concluir que a pesar de las dificultades que se hayan podido producir, existe un interés y un despliegue industrial de la SAN debido a:

a. Un incremento en la capacidad de integración del silicio que exige un mayor nivel de abstracción para controlar la complejidad y mejorar la productividad de los métodos

tradicionales basados en RTL (se reduce el tamaño del código a gestionar por el diseñador en un factor de 6 a 12 dependiendo del problema)

b. Es posible realizar la partición HW/SW de forma ágil moviendo bloques del *software* empotrado al HW con un coste reducido, explorando las alternativas de rendimiento de prestaciones en área, latencia y potencia

c. La reutilización del diseño a nivel de comportamiento facilita la migración a nivel de arquitectura (por ejemplo, modificando las interfaces del IP)

2.3. Visión de la evolución de la síntesis y experiencia previa

Como se describe acertadamente en [37] y [38] encontramos diferentes etapas en la evolución de las herramientas y metodologías de síntesis de alto nivel (figura 12).

Los primeros avances que se consolidan en herramientas de síntesis están orientados principalmente al diseño físico, al nivel circuital, como es el caso de SPICE [39] [40], y al diseño al nivel lógico con Espresso [41] y otras herramientas iniciadas en UC Berkeley por el grupo de Pederson.

En esta etapa de prehistoria, el nivel sistema se describe, por ejemplo, utilizando el modelo PMS (*Processor–Memory–Switch*). De hecho esta descomposición pasará a replicarse en el nivel inferior a sistema, el nivel alto, con el modelo de *Datapath* formado por unidades funcionales de proceso-registros-interconexiones, modelo del que deriva el concepto de lógica RTL. Esta estructura será básica en las primeras herramientas de SAN y en los primeros lenguajes HDL.

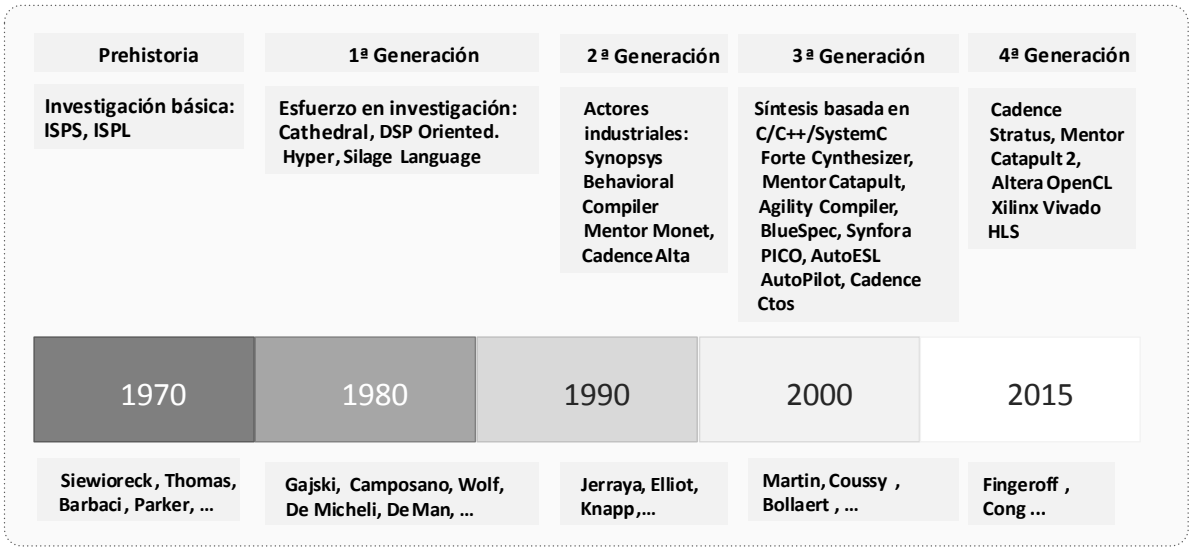


Figura 12. Breve evolución histórica de la síntesis de alto nivel

En este ambiente se referencian los trabajos de Mario Barbacci y otros en CMU en la creación de lenguajes de especificación como ISPS (*Instruction Set Processor Specificaction*) [42] y que encontramos más tarde en el paquete ENDOT. Este tipo de lenguajes tienen por objeto especificar y definir la arquitectura del procesador a nivel de juego de instrucciones (ISA), de donde puede construirse un primer nivel de verificación. En este sentido Barbacci indicó que "en teoría es posible compilar la especificación del juego de instrucciones en hardware", por

tanto estableciendo la noción de síntesis desde una especificación de alto nivel. Esto supone un paso en la estandarización de la captura del comportamiento del procesador y su posterior transformación en una implementación hardware. El entorno de diseño de referencia es CMU-DA HLS desarrollado en la Universidad de Carnegie-Mellon.

También en relación a esta etapa, el autor de este trabajo ha experimentado ampliamente con las herramientas MICRO/SILOSS de EPFL (desarrolladas en el Laboratorio LSI a finales de los 70s). Concretamente se experimentó con la descripción estructural y la simulación del juego de instrucciones para un microprocesador orientado al uso de pilas (en lugar de registros) eficiente en lenguajes procedurales estructurados del tipo Pascal, y también de un microcontrolador de acceso por clave [43]. Como aportación, MICRO y SILOSS hizo visible la facilidad con la que cualquier procesador podía crearse y simularse en base a un lenguaje de generación de microcódigo de control para su unidad de control y una descripción estructural apropiada para ejecutar su simulación. El paso hacia el diseño resultaba considerablemente ablandado.

El primer lenguaje de descripción hardware apareció al principio de la década de los 70's con una sintaxis similar a las de un lenguaje de programación tradicional [44] propuesto por Gordon Bell y Allen Newell [45] que introduce el concepto de *Register Transfer Level* (RTL) usado en el lenguaje ISP para describir los procesadores PDP-8 y PDP-16 de DEC. Se realizaron dos implementaciones de ISP: ISPL e ISPS. Este desarrollo es paralelo a la visión desarrollada en Hewlett Packard por Chris Clare [46] de un repertorio de instrucciones o ISA como un algoritmo especial, el algoritmo de interpretación de un juego de instrucciones, especificado como una Máquina de Estados Algorítmica o ASM, y la evolución de mecanismos de síntesis desde ASM hacia un flujo de datos en la ruta de datos y el flujo de su control, generados ambos a nivel de transferencias entre registros. La síntesis ASM deriva a la síntesis CDFG ya en esta década de los 70's.

La Universidad de Kaiserslautern desarrolló en 1979 un lenguaje orientado a RTL denominado KARL ("KAiserslautern Register transfer Language") que incluye muchas características para dar soporte al diseño VLSI, incluso a nivel de realización del plano de base del circuito. Para ello se desarrolló conjuntamente con CSELT en Torino el lenguaje ABL (A Block diagram Language), acompañado de un editor gráfico (ABLED). A ello le sigue el desarrollo de un entorno de diseño VLSI alrededor de KARL.

En una segunda etapa (1ª generación), ubicada temporalmente durante la década de los 80 y principios de los 90, se produce una gran expectación y un conjunto elevado de trabajos de investigación que ponen las bases de desarrollos posteriores en esta área.

A ISPS y a KARL [47] siguen en Europa Cathedral [48], COSSAP [49], CONLAN [50], IDaSS [51], y en EEUU Berkeley Hyper [52] [53] o Stanford Olympus (Hercules/Hebe) [54][55] entre otros. Cathedral se especializa en la síntesis de hardware para el procesado digital de señales. Hercules utiliza algoritmos especializados para analizar los árboles generados durante la lectura del código fuente que minimiza el número de registros.

En este periodo de tiempo un área de aplicación que concentra atención es la de circuitos intensivos en procesamiento de señal, tales como los DSP, donde la síntesis de la ruta de datos

para las operaciones de filtrado típicas del procesado de señales es un aspecto clave del problema y ofrece una estructura muy regular. Igualmente, el lenguaje de entrada del diseño está orientado a especificar de forma eficiente las rutas de datos. Ejemplos de lenguajes utilizados son una evolución de KARL en KARL-III [47], HardwareC [55] y Silage [56], un lenguaje especializado en el procesamiento de señales.

También en esta primera generación de los 80's se han producido contribuciones significativas de diferentes autores tales como Gajski [27], De Micheli [55], Camposano [57], Paulin y Knight [58] que han introducido conceptos claves para la síntesis de alto nivel, entre ellos los mencionados como básicos en SAN en la sección anterior (2.2). Pero las herramientas desarrolladas no tuvieron el éxito esperado en la industria ya que no se consideró su prioridad debido a la mayor necesidad de adoptar metodologías basadas en diseñar en nivel RTL directamente y el gran desarrollo que se experimenta en esos años con la aparición de los primeros HDL. Esta codificación RTL y síntesis lógica desde RTL fue considerada como la síntesis real del circuito. La única que permitía controlar la productividad en el diseño en términos significativos para la industria. Este desarrollo RTL creció acorde con el paso desde las necesidades de diseño LSI a las necesidades de diseño VLSI. Algunas herramientas SAN fueron demasiado específicas en cuanto a su dominio de aplicación, principalmente para la creación de unidades de procesamiento y rutas de datos, y no presentaron resultados aceptables en la lógica de control necesaria para los diseño de ASICs. Algunas herramientas SAN llegan a confundirse con “compiladores específicos de Silicio” desde alto nivel, en problemas muy específicos, saltando directamente la generación de un nivel RTL.

Unos buenos manuales de texto, muy creativos, donde ya aparecen avances consolidados aportados en esta primera generación son [59] y [60], que recogieron las visiones de UC Berkeley/Stanford U y de EPFL en esa época, respectivamente.

Algunas contribuciones y experiencias del autor se han realizado en varias de estas líneas. En primer lugar se ha contribuido en la descripción en KARL-III/ABL de un procesador *bit-slice* implementado en tecnología de GaAs [61][62][63]. En este trabajo se ha aprovechado la potencialidad de ABL para la creación de modelos a nivel de bloques del procesador, así como la flexibilidad y potencia de KARL-III para crear modelos funcionales del sistema completo.

Por otra parte se ha experimentado y se ha contribuido en el estudio e investigación en lenguajes orientados a la especificación de alto nivel y la experimentación y aplicación de diversos entornos y herramientas de síntesis de alto nivel de esta etapa. El trabajo más representativo ha utilizado Silage y Hyper, conectándolo al compilador de silicio EPOCH y al simulador de Synopsys, para generar diferentes bloques de procesamiento de señal (transformadas DCT, filtros FIR, transformadas FFT, etc.) [64]. Hyper es una herramienta de SAN orientado a la optimización de potencia usando transformaciones arquitecturales y de computación.

Se parte de una descripción funcional del bloque de procesamiento de señal y se obtiene la implementación física o *layout* mediante sucesivas transformaciones desde el dominio de comportamiento hasta el dominio físico, desde el nivel funcional al de *layout,* pero forzando pasar por el dominio estructural y los niveles RTL y lógico. El conjunto de transformaciones está

siempre basado en librerías que capturan las características funcionales, temporales y físicas de las células básicas. Estos trabajos reforzaron la convicción de establecer el nivel RTL como un nivel no prescindible en un proceso de síntesis eficiente.

En la 2ª generación de metodologías y herramientas orientadas a la síntesis de alto nivel, la de los años 90, podemos ubicar un conjunto de soluciones que tratan de complementar los niveles de abstracción que dejan sin ocupar las herramientas de síntesis lógica que se integran en el flujo de diseño. Es el momento de la aparición de Verilog y VHDL. En ese contexto se opta por utilizar los niveles de abstracción más altos soportados en los HDLs como lenguaje de entrada (VHDL/Verilog) al diseño, capturando así su funcionalidad. Esta elección de niveles excesivamente altos pero dentro de lo que podían describir los HDL resultó poco afortunada en la práctica para la síntesis en la industria. Ejemplos de herramientas comerciales de este periodo son Synopsys Behavioral Compiler (BC) [65], Mentor Monet [66] e IMEC (CoWare) OCAPI [67].

En el caso de BC el lenguaje utilizado es VHDL, donde se considera un único proceso y se permite al diseñador la introducción de sentencias *wait* para generar estados. La herramienta genera el CDFG y realiza la asignación de recursos y la planificación apoyándose en la infraestructura de síntesis de Synopsys (Design Compiler) a la que se añade un nuevo método de análisis orientado a la planificación del diseño. Da soporte a la generación de CDFG, la utilización de diferentes modos de planificación de interfaces de E/S (fijo, súper-estado en el que se permite alargar en número de ciclos la duración de la transferencia de las señales manteniendo el orden; y flotante en que se deja libertad al planificador para generar la secuencia de las señales), planifica las operaciones (soporta encadenamiento, segmentación y unidades multi-ciclo), se realiza la asignación de unidades funcionales y registros y se genera la correspondiente tabla de reservas (figura 13).

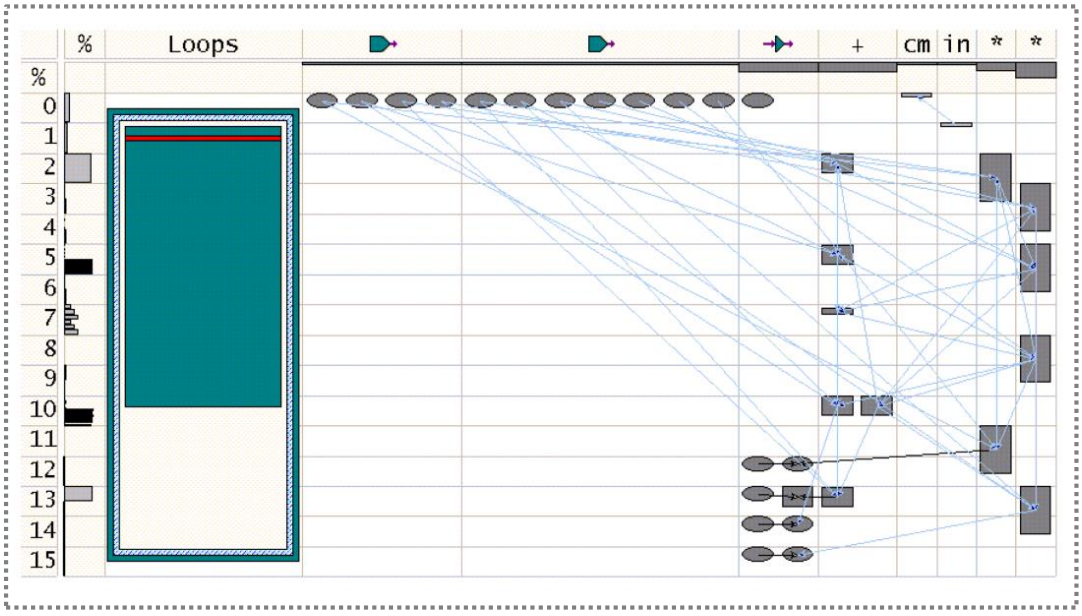


Figura 13. Tabla de reservas

Podemos citar dos problemas claves en BC. Por una parte la utilización de lenguajes de entrada basados en HDL, aunque fuera en sus niveles más altos, poco adaptados a la especificación algorítmica del problema lo que obliga a realizar la transformación de código

original, posiblemente en C/C++ a VHDL realizado con estilo de comportamiento (un proceso, múltiples sentencias *wait*).

La hipótesis de partida de que la elección del mismo lenguaje que el usado a nivel RTL facilita la introducción de nuevas metodologías fue errónea y la lección aprendida de esta decisión es que la elección del lenguaje apropiado a cada nivel de abstracción es un factor clave para el éxito de la metodología a usar.

Otro de los aspectos claves es la calidad de los resultados obtenidos, ya que no están en el rango de los valores aceptables y son dependientes de valores no controlables por el diseñador y por tanto no predecibles ni repetibles. No gestiona correctamente la presencia de bloques de memoria y sus interfaces, lo que hace que sea necesario dejar estos bloques fuera del diseño, con las dificultades de verificación que ello conlleva. En este sentido, la reutilización de entornos de test se limita a usar HDL y las interfaces externas a otros lenguajes que limitan la verificación de sistemas complejos.

Un ejemplo de la experiencia adquirida con el uso de esta herramienta es el diseño de un IP de cifrado/descifrado simétrico usando el algoritmo IDEA. El bloque se ha modelado a nivel algorítmico y se ha completado el flujo de diseño dentro del entorno de síntesis y simulación de Synopsys, implementándose en la tecnología UMC CMOS de 0,25 µm [68].

Por su parte Mentor Graphics Monet incluye diferentes herramientas interactivas de análisis tales como diagramas de transiciones, diagramas de bloques de la rutas de datos, diagramas de Gantt y enlaza estas ayudas gráficas con el código fuente de tal manera que el diseñador puede estudiar diferentes alternativas, añadiendo restricciones para controlar los resultados [66].

Las herramientas de la *Suite* OCAPI por su parte nacen en 1996 en IMEC, incorporando Cathedral III y otras utilidades, y ya plantean capturar la descripción de comportamiento no en HDL sino en C/C++. OCAPI está basado en lenguaje orientado a objetos y hace una traducción automática de C/C++ a VHDL comportamental para hacer luego una síntesis convencional [69][70]. Sin embargo ese paso queda corto al no afrontarse como objetivo obtener una salida HDL sintetizable, ni mucho menos RTL, y al resultar un paso de calidad limitada. El interés mayor estaba en poder hacer una descomposición HW/SW y un codiseño, incluyendo cosimulación a ese nivel. En la década posterior la propia CoWare (luego parte de Synopsys), y la mayor parte de la industria activa en SAN, cambió de paradigma, hacia la síntesis desde C/C++ y hacia la aparición de SystemC.

Otro entorno que empieza a utilizar SystemC es CoCentric de Synopsys [36]. Está orientado a la creación de aplicaciones de flujo de datos síncronos con máquinas de estado finito como modelo de control. Un ejemplo del desarrollo de un núcleo codificador y decodificador JPEP se puede consultar en [71].

En efecto, en la tercera generación, que podemos encuadrar a partir del año 2000, hay ya un cambio notable de aproximación al problema. En primer lugar se cambia el lenguaje de entrada a nivel algorítmico, utilizando aproximaciones más cercanas a la especificación original, ya sea en C/C++, SystemC o incluso en Matlab. Esto facilita las transformaciones iniciales del diseño, partiendo de una especificación ejecutable del problema.

Existe una larga lista de herramientas [72], entre las que podemos citar SpecC [73], Impulse-C [74], Calypto/Mentor Catapult C Synthesis [75] (figura 14), Forte Cynthesizer [76] (ahora adquirida por Cadence) (figura 15), Celoxica Agility [77] , Bluespec [78], NEC CyberWorkBench [4] (figura 16), Synopsys SynphonyC [79] (figura 17), AutoESL AutoPilot [80] (ahora Xilinx Vivado HLS) y Cadence C-to-Silicon (CtoS) [81]. Algunas de estas herramientas han aparecido en este periodo temporal y han evolucionado, soportando ahora nuevas funcionalidades y podemos incluirlas como herramientas de 4ª generación, la generación actual, como veremos.

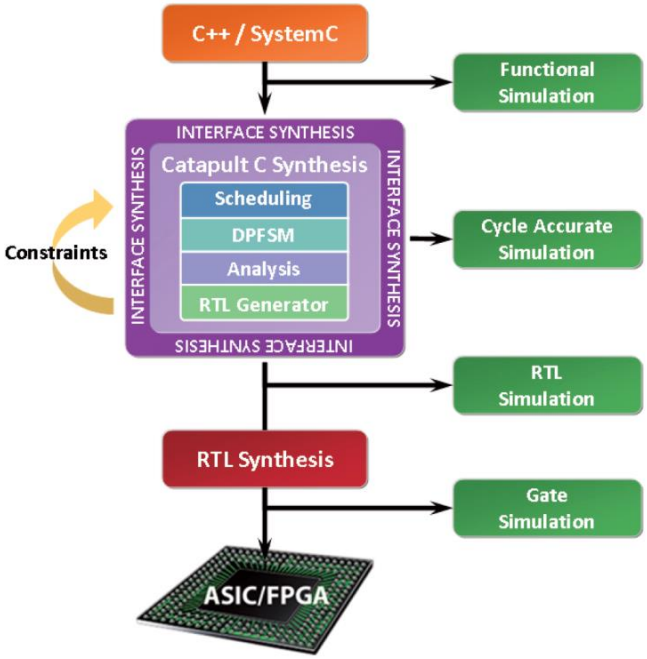


Figura 14. Calypto/Mentor Catapult HLS

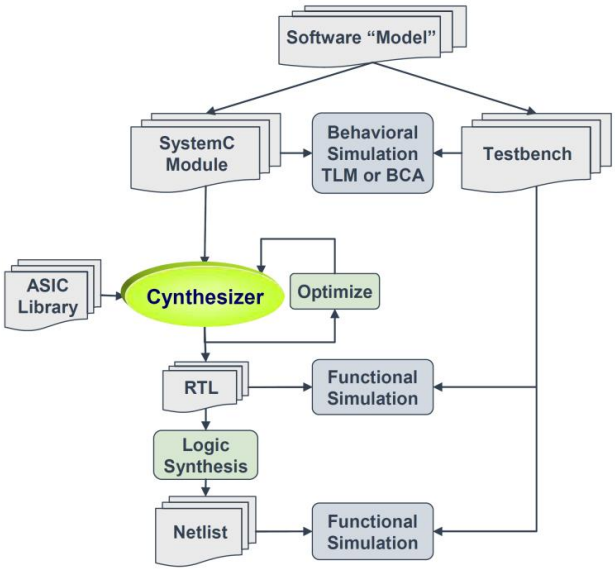


Figura 15. Flujo de diseño de Forte Cynthesizer

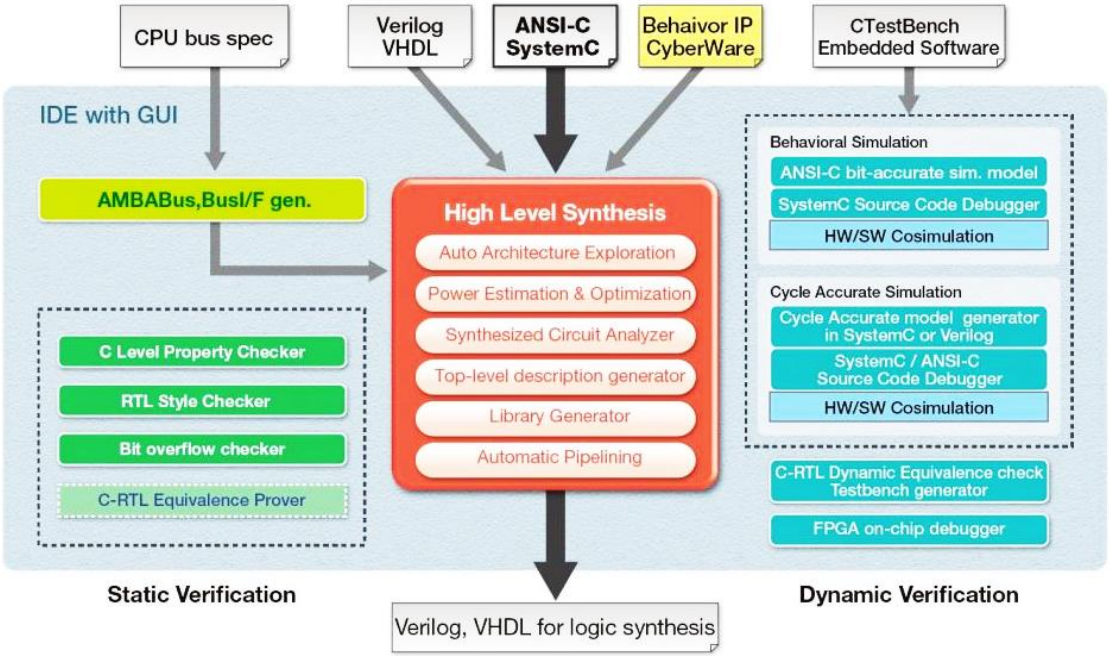


Figura 16. NEC Cyber Workbench (*partnership* con Aldec)

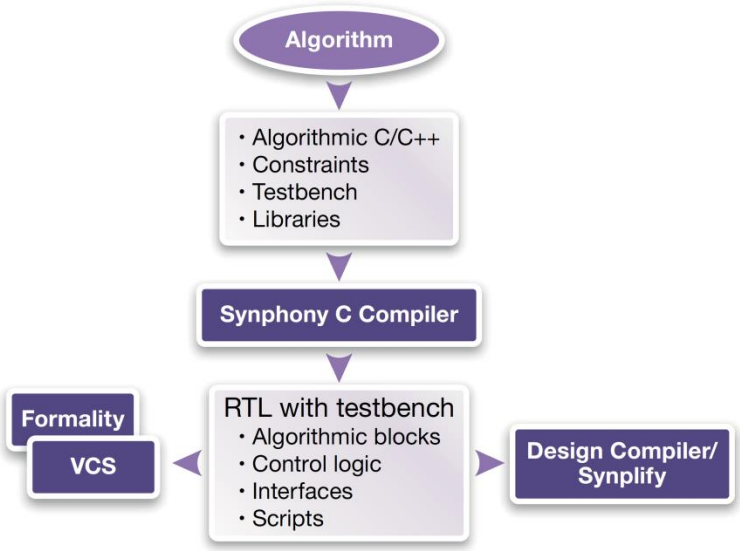


Figura 17. Arquitectura de Synopsys Synphony Compiler

Muchas de estas herramientas están centradas en ASIC y ASSP y otras son específicas para FPGAs. También algunas están centradas en el diseño de soluciones para DSPs (arquitecturas tipo *Dataflow*, dominadas por el flujo de datos) mientras que otras añaden soporte al diseño orientado a problemas dominados por el control. Como veremos más adelante, las herramientas actuales, que podemos considerar como de 4º generación soportan ambos dominios.

La entrada del diseño, como se indicó anteriormente, se realiza desde C/C++ o SystemC, si bien con determinadas librerías y estilos de especificación o variaciones del lenguaje con tipos de datos dedicados (Algorithmic C de Mentor [75]). Otras opciones utilizan SystemVerilog

(BlueSpec) o Esterel. Por ejemplo BlueSpec [78] parte de una descripción SystemVerilog o un subconjunto de SystemC y ofrece un entorno para realizar refinamientos sucesivos del diseño inicial en alto nivel que garantiza la corrección del diseño usando técnicas TRS [82]. Igualmente herramientas orientadas al diseño de circuitos de procesado de señal utilizan Matlab+Simulink como entorno de captura del diseño, si bien utilizan C como paso intermedio.

C/C++ soporta la captura de la funcionalidad del diseño pero no está preparado para modelar la estructura y el comportamiento temporal del mismo. Por ello la mayoría de las herramientas que utilizan C/C++ utilizan un conjunto de *pragmas* que permiten al diseñador guiar al compilador hacia la solución deseada. Este conjunto de *pragmas* soporta, por ejemplo la definición del protocolo de E/S, el control de los bucles o la protección de determinadas zonas de código para indicar la implementación de protocolos con dependencias estrictas. En cualquier caso el conjunto de directivas es finito y muy enfocado a un determinado estilo de diseño. SystemC en cambio ha sido diseñado para dar soporte a estos tres dominios de modelado del diseño. Al tratarse de clases de C++ y un *kernel* de simulación basado en eventos, soporta el modelado en varios niveles, como por ejemplo *untimed*, BCA, con abstracción de datos, etc.

SystemC soporta estructura en el diseño, permitiendo gestionar su complejidad [21]. Igualmente en este caso se separa la especificación de las comunicaciones del procesamiento de datos, utilizando metodologías orientadas a transacciones. De hecho la aparición de las clases de Verificación en C++ (VIP), y luego las clases de Interfaz en SystemC, han dado lugar en el 2001 al estándar *Transaction Level Modeling* (TLM) [83][84].

El uso del lenguaje apropiado al dominio de aplicación de la herramienta es un factor decisivo para su aceptación y para la obtención de la mejor calidad de resultados. Otro aspecto importante en la elección del lenguaje apropiado es la utilización de las mejoras en las técnicas de optimización de los compiladores, muchas de ellas comunes con las técnicas de optimización de los lenguajes de alto nivel.

Otro de los aspectos a tener en cuenta es que el nivel de integración a nivel de sistema (SoC) genera la necesidad de integrar diferentes tipos de unidades de procesamiento creando plataformas heterogéneas. Con ello las herramientas de SAN deben soportar diferentes dominios de diseño, ya sea de tipo flujo de datos (*dataflow*) o de control. La mayor parte de estos recursos se implementan conformando aceleradores hardware para algoritmos complejos (procesado de señal, multimedia, aplicaciones de bioingeniería, etc.). Aunque la gran industria apunta siempre a aceleradores en ASIC o ASSP o integrados en SoC, esos aceleradores muchas veces se integran en FPGAs, ya sea como etapa de prototipado ya sea como implementación final. Por lo tanto los parámetros a utilizar para la selección de la arquitectura son diferentes en FPGA que para el caso del ASIC o ASSP. En la FPGA, aparte de que se cumplan los requisitos temporales, se utiliza el criterio de si el acelerador se puede mapear o no en la FPGA con los recursos disponibles. Diferente situación se produce en el ASIC donde el espacio de soluciones área-tiempo es más flexible que para el caso de las FPGAs.

Esta tercera generación de herramientas SAN ha tenido en cuenta estos aspectos, diferenciando los tipos de soluciones y dominios, los lenguajes de entrada y conectando las

salidas a herramientas de implementación. En aquellos casos en los que se han especializado en implementaciones FPGAs, aprovechan los recursos de mayor granularidad de la FPGA tales como bloques de memoria y recursos tipo MAC/DSP para mapear variables y unidades aritméticas complejas.

La experiencia adquirida con estas herramientas ha facilitado una mejor comprensión de los procesos de síntesis que traen como consecuencia una mejora en la calidad de los resultados, al estar derivados del uso de mejores prácticas. En concreto comentamos ahora la utilización de Agility Compiler durante el desarrollo del proyecto ARTEMI+ [85] en el que se ha diseñado un decodificador de vídeo H.264/AVC-SVC con perfil *baseline* desde su descripción algorítmica basada en JM versión 12.4 [86]. Relacionados con este proyecto podemos citar además los trabajos de prototipado mediante síntesis a partir de SystemC del bloque de estimación de movimiento [87] y el diseño y simulación del bloque de mejora de vídeo [88].

Agility Compiler soporta el subconjunto sintetizable de SystemC, generando descripciones RTL para los flujos de diseño síntesis lógica y simulación para tecnologías ASIC y FPGAs (figura 18). Integra un IDE con soporte a los procesos de captura del diseño, simulación y síntesis. Dispone de generación automática de código para FPGAs de Actel, Altera y Xilinx. Al utilizar SystemC es necesario definir procesos, definir listas de sensibilidad, y describir el modelo al nivel de transacciones (TLM) separando los detalles de la comunicación entre módulos de su implementación, modelándose como canales. Estos canales se implementarán como FIFOs o buses. Las transacciones se gestionan a través de las funciones de las interfaces de los módulos.

Durante la fase de verificación es preciso escribir el *testbench* en SystemC y adaptarlo a nivel RTL. Algunas de las capacidades de optimización incluidas son la posibilidad de integrar bloques predefinidos como cajas negras, definir *resets* globales asíncronos e inicializar valores a las señales. Igualmente es posible realizar *retiming*, equilibrado y compartición de lógica y su reorganización en forma de árbol para reducir latencias. Asimismo puede hacer la reescritura de condiciones para ajustarlas a la lógica utilizada en el *netlist* final. Es posible la utilización de punteros que se resuelven en tiempo de compilación y desenrollar bucles bajo condiciones determinadas. La herramienta Agility Compiler genera informes de utilización de recursos y ayudas gráficas (CDFG). Utiliza formatos estándar para la salida del diseño, ya sea en VHDL, Verilog o EDIF para el caso de las FPGAs. Igualmente se genera SystemC a nivel RTL optimizado para simulación.

Durante la transformación del decodificador H.264/AVC-SVC desde una solución *software* a una implementación hardware, uno de los aspectos claves es realizar la partición del algoritmo de tal forma que se agrupe la funcionalidad en los bloques para disminuir los requerimientos de comunicación. En este trabajo se ha utilizado una aproximación TLM, separando interfaces y procesamiento para cada bloque principal. Para el modelado de los bloques se ha utilizado SystemC OSCI con el estilo de modelado soportado por Agility Compiler. Para modelar las comunicaciones se ha utilizado TLM-2.0 que después ha sido refinado de forma manual definiendo las interfaces finales utilizadas por los bloques principales. Dichas interfaces TLM se han sustituido por una interfaz de señalización *request/validate*, conjuntamente con la interfaz de datos correspondiente adaptada al tipo de datos a transmitir. Además se han definido las correspondiente interfaces a las memorias necesarias en el decodificador: *Temporal Picture*

*Buffer (*TPB), *Decoded Picture Buffer* (DPB), *Picture Buffer* (PB) y memorias internas. Las últimas versiones de Agility Compiler usadas en el proyecto soportan la capacidad de inferir memorias.

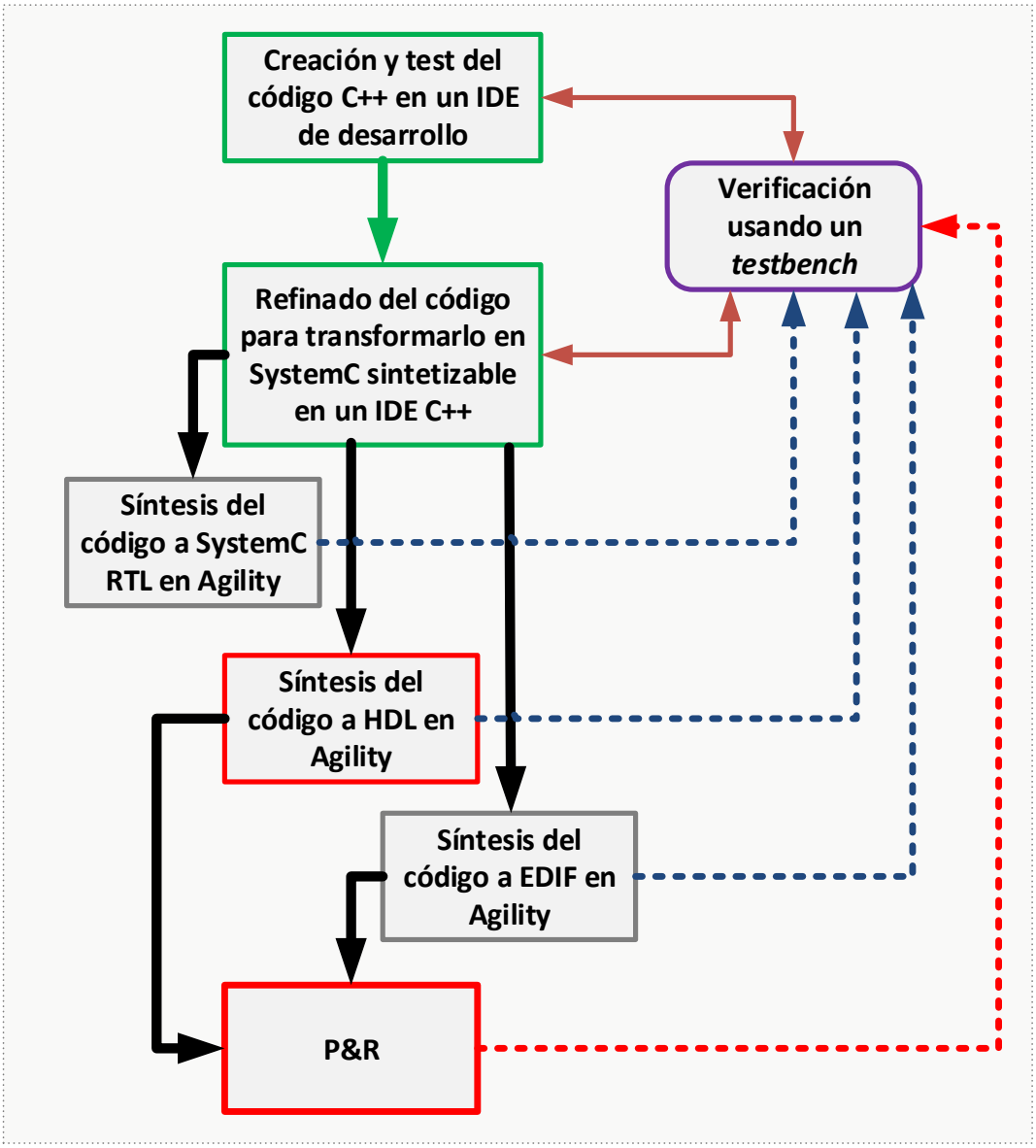


Figura 18. Flujo de diseño con Agility Compiler

Con objeto de mejorar las rutas críticas el proceso de planificación ha sido guiado por el diseñador, de tal manera que, apoyándose en la utilización de sentencias *wait*(), el diseñador puede segmentar las rutas críticas aumentando la frecuencia de funcionamiento, manteniendo o cumpliendo las restricciones de latencia requeridas. Esto no es automático. Se necesita acometer un proceso de mejoras iterativas que requiere la intervención del diseñador para la toma de decisiones. Igualmente los resultados obtenidos en alto nivel en términos de área/tiempo eran muy conservadores por lo que era preciso apoyarse en los resultados de las herramientas de síntesis lógica.

2.4. Situación actual en SAN

Hemos visto que la síntesis de alto nivel toma una descripción de comportamiento o algorítmica de un sistema digital y crea una estructura a nivel RTL que implementa dicho comportamiento. Las herramientas de SAN pueden generar representaciones RTL del diseño de una forma más eficiente, de mejor calidad y más rápida que si abordamos el diseño de forma manual en determinados dominios de aplicación.

Como se ha indicado, la SAN está ganando importancia en el diseño de sistemas electrónicos digitales ya que permite abordar la complejidad creciente del diseño del sistema, reduce los esfuerzos de verificación, facilita la exploración del espacio de diseño y la optimización del diseño para varios factores tales como latencia, potencia y área/recursos y facilita la migración entre diferentes tecnologías de implementación del diseño, ya sea ASIC, ASSP o FPGA.

Se ha producido un cambio de paradigma en los métodos de entrada usados para la SAN donde las representaciones de comportamientos basadas en HDL han sido sustituidas por lenguajes de alto nivel más cercanos a la creación de algoritmos y bloques de procesamiento de señal. Ello implica que se haya incrementado el interés de los usuarios industriales por la síntesis de alto nivel.

Esta tercera generación de la SAN se ha ido consolidando en el sector industrial de tal forma que en el año 2010, 30 de las mayores compañías de semiconductores a nivel mundial ya habían adoptado SAN como método de diseño de sus productos. En 2014 la adopción de la síntesis de alto nivel es un hecho en la empresa, quedando demostrado su valor desde el punto de vista industrial como método de diseño [89]. Consideramos ahora la situación actual, fijando el horizonte de 2015 como ya indicamos en la figura 12.

Los tipos de diseños en que actualmente se adopta la síntesis de alto nivel son aquellos con contenido algorítmico complejo, diseños cuyas especificaciones cambian rápidamente o los estándares evolucionan y aquellos diseños que deben implementarse rápidamente para tener un producto en el marcado, ya sea en FPGA o ASIC. Estos tipos de diseño se implementan mucho más rápidamente usando SAN que estilos de diseño RTL. Entre estos tipos de diseño podemos encontrar soluciones complejas desde *Wireless 3G/4GLTE*, hasta codificación/decodificación de vídeo H.264/AVC, H.265/HEVC, VP9, etc.

La mayor parte de las herramientas utilizadas en SAN actualmente utilizan C/C++ y SystemC como entrada del diseño. Para este trabajo vamos a realizar un estudio más detallado sobre los entornos de Calypto/Mentor Catapult, Altera OpenCL, Vivado HLS y Cadence CtoS, aportando también nuestra propia experiencia de diseño con ellos, para extraer mejores prácticas para entornos de producción de diseños. Otros entornos y herramientas están descritos en [90]. En el apartado siguiente (2.5) se indicarán algunos aspectos de tendencias en la evolución de estos entornos debido a avances científicos en la tecnología de diseño y de construcción de herramientas.

2.4.1. Calypto/Mentor Catapult

Catapult C Synthesis [91] parte de un algoritmo escrito en ANSI C++ y mediante un conjunto de directivas de síntesis genera un diseño a nivel RTL optimizado para una determinada tecnología, ya sea ASIC o FPGA. Las directivas no modifican el comportamiento funcional de la especificación de entrada sino que añaden la información precisa para crear la estructura del diseño, incluyendo las interfaces. Es posible su utilización tanto en aplicaciones intensivas en cómputo como de control o en sistemas que incluyan ambos dominios.

La entrada del diseño es totalmente funcional, sin incluir conceptos temporales o estructurales de forma explícita, tales como la arquitectura del diseño o sus interfaces de E/S. Esto permite una exploración completa del espacio de diseño a nivel de arquitectura. Las directivas especifican la tecnología de implementación, y sus componentes, el periodo de reloj, la síntesis de interfaces, la arquitectura de memorias, el tipo de paralelismo a aplicar en los bucles (desenrollado total o parcial, segmentación), jerarquía de bloques. Igualmente se dan restricciones de planificación tales como latencia o ciclos de reloj. También incluye directivas para el control de las tareas básicas de síntesis de alto nivel tales como planificación temporal (control de latencia/ciclos), o asignación de recursos (tipo de recursos y cantidad).

En cuanto al tipo de datos, se soportan tanto los tipos *integer* nativos de C++, enteros con operaciones con precisión de bits en C++ y coma flotante. Soporta los tipos propietarios definidos como Algorithmic C [75], incluyendo clases denominadas como Algorithmic C Window para portar *arrays* multidimensionales. Igualmente soporta tipos de datos en coma flotante (división, raíces, etc.) pero los transforma a coma fija. También soporta los tipos sintetizables de SystemC [92].

El proceso de creación del diseño con Catapult C se organiza en los siguientes pasos:

1. Crear/adaptar, encapsular y verificar el código fuente utilizando los tipos de datos y la estructura soportada. Es necesario identificar el nivel más alto de jerarquía mediante la correspondiente directiva o *pragma*.
2. Analizar el algoritmo con respecto a la tecnología a utilizar y al reloj definido, ya que representan las dos restricciones más importantes del diseño. Existen herramientas de análisis tales como el diagrama de Gantt. A este análisis sigue la generación de la arquitectura RTL.
3. Creación del diseño hardware. En este paso se hace la asignación de recursos hardware, las restricciones tecnológicas, la arquitectura de memoria en función de las restricciones de la tecnología y el esquema de E/S. Por último se genera el código RTL en HDL (VHDL o Verilog).
4. Realización de la simulación, con información precisa de ciclos.
5. Síntesis del diseño RTL apoyándose en herramientas de síntesis lógica integradas en el entono.

Tal como se ha indicado, las interfaces y la arquitectura del hardware generado puede controlarse mediante directivas de síntesis, ya sea como *pragmas* en el código fuente, como

directivas introducidas mediante la interfaz gráfica o un fichero Tcl, o incluso valores por defecto incluidos en la configuración del proyecto.

Todas las señales necesarias y las restricciones temporales se generan durante el proceso de síntesis de tal forma que el código RTL optimizado generado es conforme con las interfaces. Por ejemplo una variable de tipo *array* en C/C++ puede generar una interfaz de memoria (dirección/dato/*enable*/*write* o *read*) o una interfaz de flujo de datos *(Stream*) de tal forma que los datos son proporcionados de forma secuencial.

Catapult C incluye una base de datos orientadas hacia síntesis (Synthesis Internal Format – SIF) que da soporte a la exploración del espacio de diseño para diferentes soluciones y tecnologías. Durante la fase de síntesis de alto nivel se generan las restricciones necesarias para utilizarlas en la síntesis RTL. La síntesis de las interfaces se realiza para un conjunto definido de canales y con su correspondiente señalización, incluyendo interfaces de memoria, FIFOs, interfaces en flujo de datos, etc.

La jerarquía del diseño, que implica la creación de bloques que se ejecutan de forma concurrente, se especifica también con directivas de tal forma que una función se puede implementar en un bloque separado, reutilizando su implementación en caso de no ser requerida en el mismo ciclo. Igualmente se pueden especificar bloques interconectados mediante FIFOS, utilizando un modelo de redes de Kahn (KPN). Los bloques se pueden implementar en diferentes dominios de reloj, y la lógica de sincronización entre dominios se genera de forma automática. La comunicación entre bloques se optimiza utilizando FIFOs, Memorias *ping/pong* o señalización *request/validate*.

Para obtener convergencia en las prestaciones entre los resultados de alto nivel y RTL se utiliza una librería de componentes precaracterizados para la tecnología objeto, ya sea FPGA o ASIC, con diferentes alternativas retardo–área–potencia.

Durante el flujo de síntesis (figura 19) se genera el *testbench* que encapsula la descripción RTL obtenida en SystemC, reutilizando el *testbench* original en C/C++. Igualmente es posible realizar un análisis de potencia preciso a partir de la actividad obtenida del modelo RTL de tal forma que se obtiene una información detallada de las prestaciones del diseño con objeto de analizar varias opciones en la implementación de la arquitectura que cumplan con las restricciones del diseño.

Durante la realización de este trabajo se ha utilizado Catapult para la implementación de parte del procesador de eventos que se describe como caso de aplicación en el capítulo 3 a partir de la descripción SystemC. Sin embargo la versión universitaria utilizada estaba restringida a un único bloque por lo que los tiempos de síntesis para realizar la exploración del espacio de diseño eran prohibitivos dato el tamaño de la aplicación. Utilizar un nivel de granularidad más fino para implementar el acelerador implica un esfuerzo adicional en la creación de entornos de test e interfaces de comunicación y ello hace perder la ventaja de la utilización de una herramienta de SAN. Las versiones actuales ya no incluyen esta restricción, permitiendo controlar de forma eficiente la jerarquía de tal forma que cambios en un bloque no afectan totalmente al resto del diseño.

La herramienta presenta una curva de aprendizaje relativamente rápida, si bien para obtener un rendimiento elevado y mejorar la calidad de los resultados se exige una comprensión detallada de las alternativas de optimización. En otros casos de uso, con problemas de dimensiones relativamente menores, como es el caso de diferentes bloques del filtro de desbloqueo – DF del capítulo 4, las prestaciones están acorde a los valores obtenidos durante la síntesis lógica. En todos los casos se ha hecho uso de los modelos SystemC.

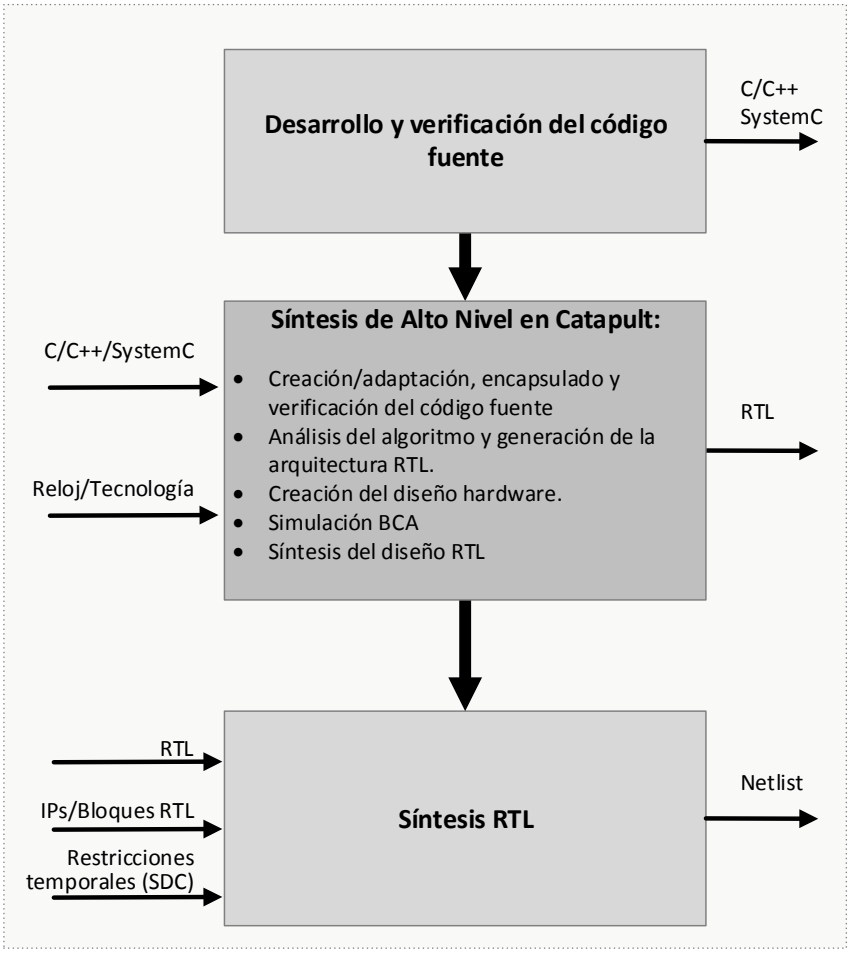


Figura 19. Flujo de diseño en Catapult

2.4.2. Altera OpenCL

Altera ha desarrollado un entorno de síntesis orientado a la implementación rápida de aceleradores hardware y basado en OpenCL [93]. Open Computing Language (OpenCL) es un estándar abierto para la programación de sistemas de computación heterogéneos que incluye un lenguaje de programación paralela para la parte del dispositivo y una API en C para la parte del host. Por tanto el *software* que se ejecuta en el host es totalmente secuencial y se puede ejecutar tanto en un procesador *soft-core* embebido en la FPGA (o en varios *cores*) como en un procesador externo, o en varios.

OpenCL está basado ISO C99 y permite crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse en una serie de núcleos de computación disponibles, CPUs, GPUs, DSPs y FPGAs.

Altera ha creado su propio kit de desarrollo de *software* (SDK) [94] orientado hacia la implementación de aceleradores sobre FPGAs. El programa desarrollado en OpenCL para la parte del dispositivo se transforma en un núcleo IP en Verilog/VHDL que se toma como referencia para generar el *bitstream* de configuración de la FPGA. La parte del host se utiliza para configurar y controlar la FPGA para el procesado de datos.

La implementación de Altera de OpenCL [95] facilita la comunicación con el procesador, la planificación de los procesos y la transferencia de los datos en alto nivel. Dispone de un conjunto de funciones que abstraen la comunicación entre el procesador y el acelerador tal como se muestra en la figura 20. En la figura 21 se muestra la arquitectura de programación vista desde el host y en la figura 22 el flujo de diseño completo.

Altera OpenCL utiliza el compilador aoc (Altera Offline Compiler) para transformar la descripción algorítmica de los núcleos de procesamiento en una arquitectura RTL y genera la descripción Verilog que luego puede ser transformada por las herramientas de Quartus II en la implementación hardware de los núcleos. Durante el proceso se genera también un fichero binario (.aocx) que contiene la configuración de la plataforma. Los procesos realizados por el aoc son los siguientes:

- Compilación, cuyo objetivo es detectar errores sintácticos
- Emulación, que ejecuta los núcleos en una arquitectura x86 para depurar su funcionalidad
- Perfilado, que instrumenta el código Verilog generado para obtener perfiles de ejecución con objeto de optimizar la arquitectura del núcleo
- Ejecución del flujo, que genera los ficheros binarios .aocx para su implementación en la FPGA

El compilador aocx utiliza directivas basadas en *pragmas* y atributos para controlar el proceso de generación de la arquitectura RTL usando técnicas de SAN. Entre ellas podemos encontrar el control del desenrollado de bucles. Igualmente el compilador posee directivas de control para especificar el número de unidades de computación, replicar el número de núcleos en una arquitectura SIMD, o especificar la posición de los buffers de memoria, tanto local como global. Igualmente hace un uso extensivo del preprocesador de C para parametrizar la descripción.

```
main()
{
   Read_data( … );
   manipulate( … );
   clEnqueueWriteBuffer( … );        // Copiar datos a la FPGA
   clEnqueueTask(…, my_kernel, …);   // Procesar datos en la FPGA
   clEnqueueReadBuffer( … );         // Copiar datos desde la FPGA
   display_result( … );
}
```

Figura 20. Ejemplo de código OpenCL

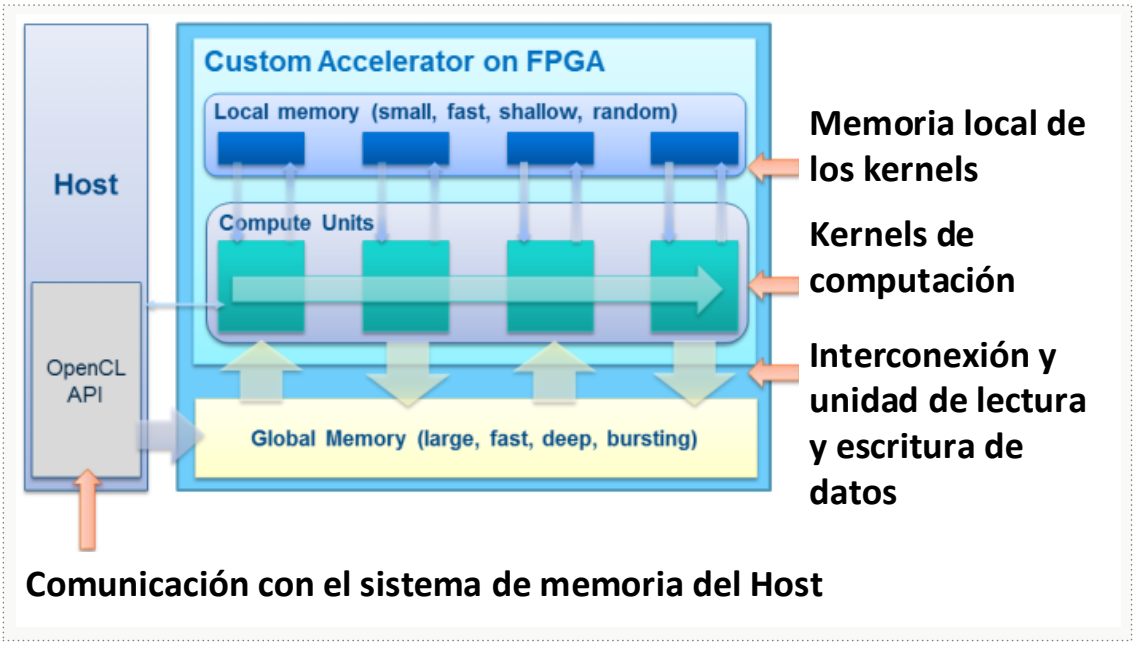


Figura 21. Vista del programador de OpenCL para el *host*

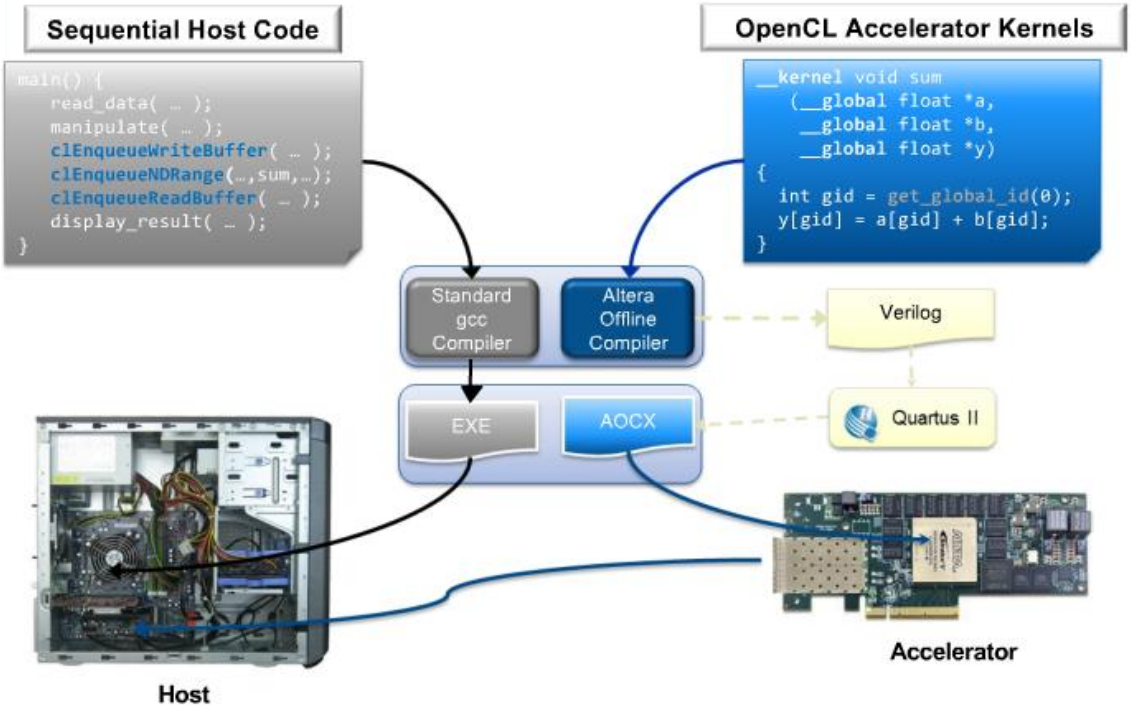


Figura 22. Flujo de diseño con Altera OpenCL

Con respecto a los canales de comunicación internos, permite combinar la utilización de FIFOs en estructuras de flujo de datos con zonas mapeadas en memoria. Los canales pueden ser bloqueantes o no bloqueantes. Soporta estructuras complejas de comunicación entre núcleos, ya sea desde una señalización simple hasta estructuras de memoria *ping-pong* o *pipes* con algunas restricciones.

2.4.3. Xilinx Vivado HLS

Vivado HLS, es la modificación y personalización realizada por Xilinx de las herramientas de SAN AutoESL AutoPilot que a su vez se deriva del entorno XPilot de la Universidad de California Los Angeles [96][97]. Vivado HLS sintetiza el modelo algorítmico en C/C++ y SystemC generando un bloque de propiedad intelectual (IP) o exportando el código RTL en VHDL/Verilog para síntesis e implementación en una FPGA de Xilinx (figura 23). Utiliza el entorno de compilación de código abierto LLVM [98] y se aprovecha de sus continuas optimizaciones. Vivado HLS soporta C, C++ y SystemC, lo que permite realizar la simulación del código C realizando un mínimo conjunto de modificaciones. Vivado HLS incluye un conjunto de clases y librerías propietarias en C++ (ap_[u]int<>) para dar soporte a enteros de precisión arbitraria necesarios para realizar la implementación hardware a nivel de bits, lo que permite operaciones con palabras de más de 64 bits. Soporta operaciones aritméticas, lógicas y de relaciones con signo y sin signo. Igualmente soporta tipos de coma fija (ap_[u]fixed) para gestionar aritmética de números reales en representación de punto fijo.

Vivado HLS realiza la síntesis del algoritmo, donde se produce la transformación de las funciones en la arquitectura RTL, incluyendo las tareas de planificación, asignación y enlazado de unidades funcionales. Estas operaciones afectan a la síntesis de las interfaces y son afectadas a su vez por esta.

La síntesis de las interfaces realiza la transformación de los argumentos y parámetros de las funciones en puertos RTL con el protocolo de comunicación elegido para la comunicación con otros bloques del diseño. Afecta a variables globales, argumentos de las funciones de mayor nivel de jerarquía y valores retornados por las funciones.

Entre los tipos de interfaces se encuentran señales registradas o no, con *handshake*, Buses (genéricos, AXI4, AXI4S, AXILite), FIFOs y RAM.

Las principales técnicas de optimización usadas por Vivado HLS son las siguientes:

- Optimización del flujo de datos o entre funciones (*Dataflow pipelining*)
- Segmentación de las funciones
- Optimización de latencias
- Enrollado y desenrollado de bucles
- Segmentación de bucles
- Modificación de los intervalos de iniciación
- Mapeado de *arrays* a memorias
- Limitación o compartición de recursos

La segmentación se puede aplicar –con distinta granularidad– como una optimización entre funciones o entre operaciones dentro de una función. También se puede aplicar a las operaciones dentro de un bucle o entre bucles. El principal objetivo es incrementar la capacidad de procesamiento, siempre que no haya dependencias de datos tales que una función deba esperar por los resultados de la anterior ("verdaderas dependencias" o no nominales).

El uso en Vivado HLS de *Dataflow pipelining* transforma un algoritmo codificado como una sucesión secuencial de funciones, en una partición y segmentación de bucles para crear una arquitectura paralela (figura 24 y figura 25).

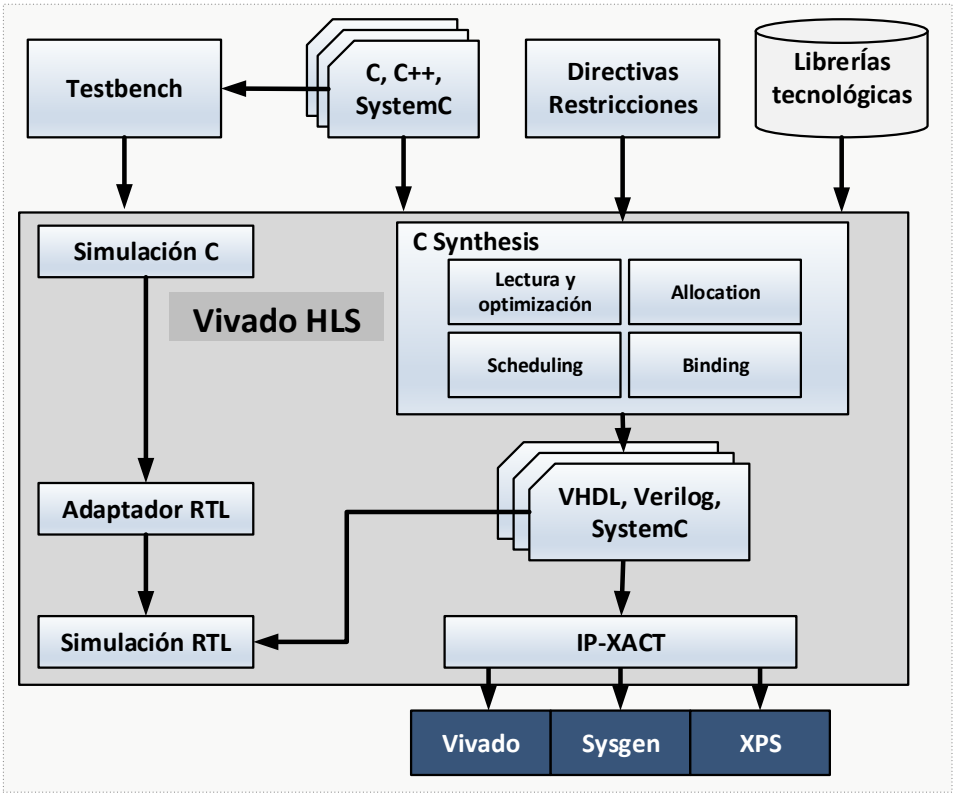


Figura 23. Flujo de diseño en Vivado HLS

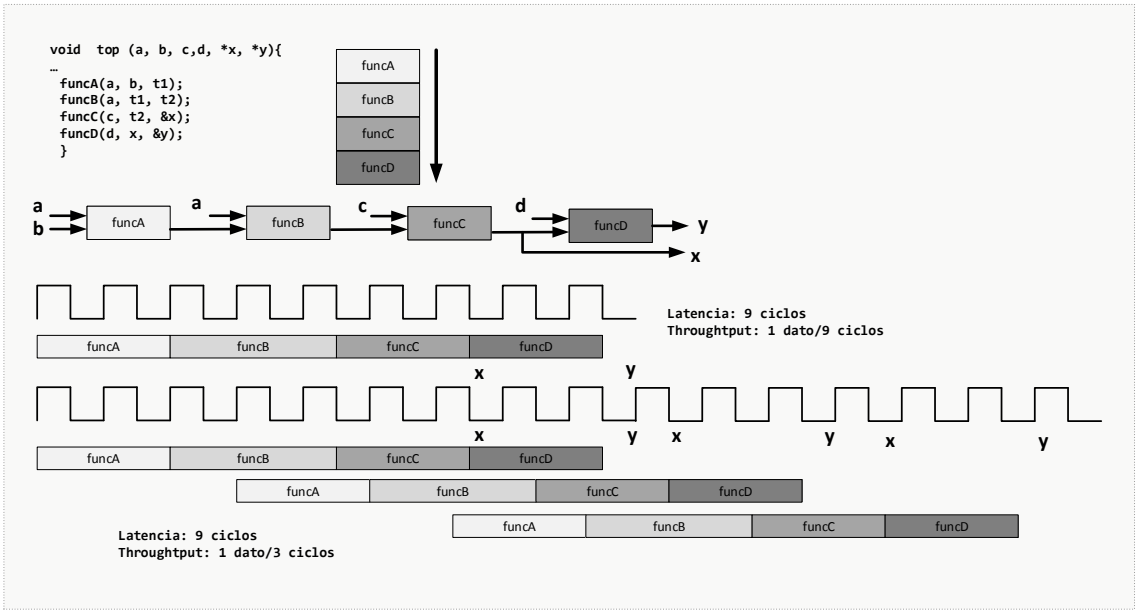


Figura 24. Ejemplo de transformaciones de segmentación de funciones

Vivado HLS contiene un conjunto de directivas que permiten al diseñador tener control sobre la segmentación de bucles:

- *unrolling*: desenrolla el bucle, ya sea de forma total o parcial para aumentar el paralelismo
- *merging*: une diferentes bucles sucesivos para su optimización
- *flattening*: reduce el nivel de profundidad de un bucle
- *dataflow*: paraleliza los bucles secuenciales para operar concurrentemente

- *pipelining*: paraliza la ejecución de los datos mediante segmentación del cauce de proceso
- *latency*: especifica la limitación en ciclos de la latencia del bucle

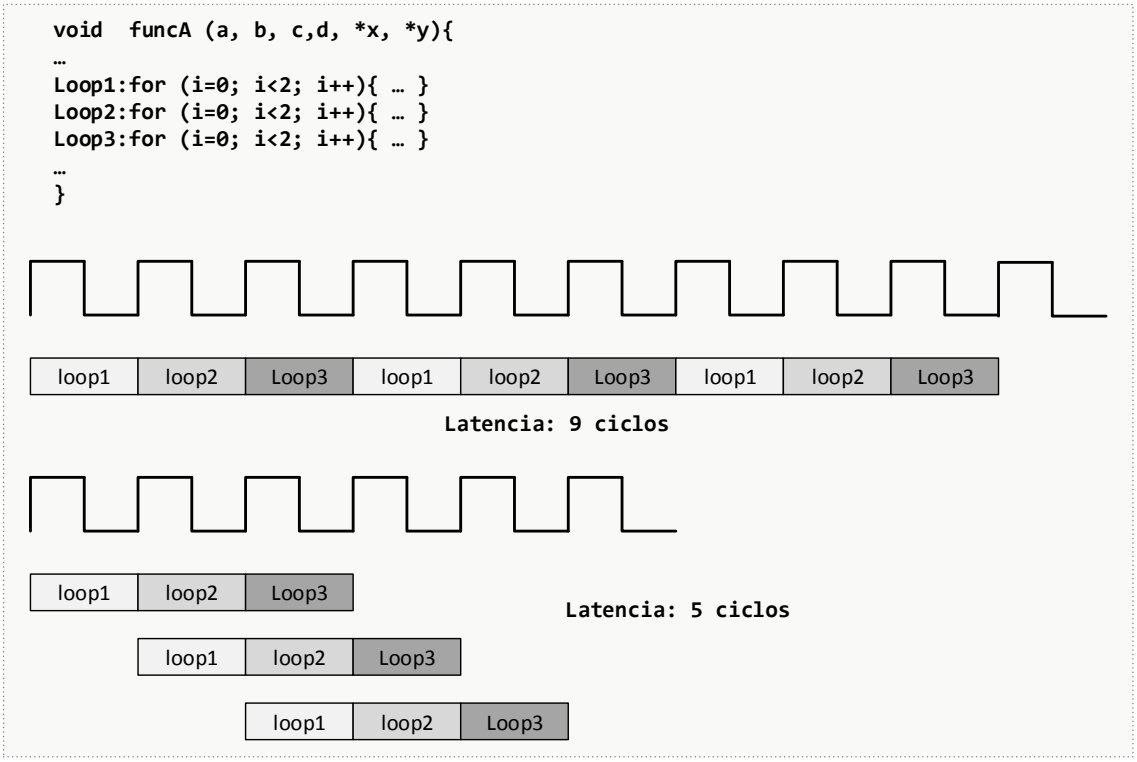


Figura 25. Segmentación de bucles

En cuanto al mapeado de *arrays* en memorias, Vivado HLS incluye un conjunto de directivas que permite la manipulación de los *arrays* de tal forma que optimiza su implementación en memoria. Entre las operaciones que están soportadas se encuentran la selección del recurso a utilizar, combinación de varios *arrays* para mapearlos en un único recurso de memoria, división de un *array* en varios para facilitar su mapeado para evitar cuellos de botella en el acceso a los datos, transformación de un *array* multidimensional para mapearlo en una memoria, mapeado en modo *stream* para implementarlo en una FIFO. Estas funcionalidades nos ayudan a realizar una exploración de la arquitectura del sistema.

Vivado HLS soporta la captura de restricciones como *pragmas* en el código de entrada, como directivas de usuario en un fichero en formato Tcl. La jerarquía del sistema se infiere de la estructura de funciones disponible en el código C/C++ o de la estructura de módulos explicita incluida en SystemC (figura 26). Se puede manipular la jerarquía haciendo copias *inline* de las funciones. Todas las transformaciones mencionadas afectan de forma directa a los recursos utilizados en la FPGA, a la latencia utilizada y a la potencia consumida en el diseño.

Uno de los aspectos claves que facilitan la utilización de Vivado HLS es la capacidad para la generación de interfaces estandarizadas en el ecosistema de diseño de Xilinx. Xilinx adoptó el estándar AMBA AXI (Advanced eXtensible Interface) con la familia Spartan-6. Por ello es de interés dotar a los IPs desarrollado de interfaces compatibles con AXI, ya sean de tipo *stream*, mapeadas en memoria o Lite (axis, m_axi o s_axilite, respectivamente). Estos tipos de interfaz

se generan en los puertos principales del bloque IP, controlando su generación mediante directivas de síntesis. Para el caso de que el modelo sea desarrollado en SystemC el soporte a la generación de interfaces se reduce a algunas interfaces de memoria y FIFOs, requiriendo librerías adicionales para la gestión de las interfaces AXI.

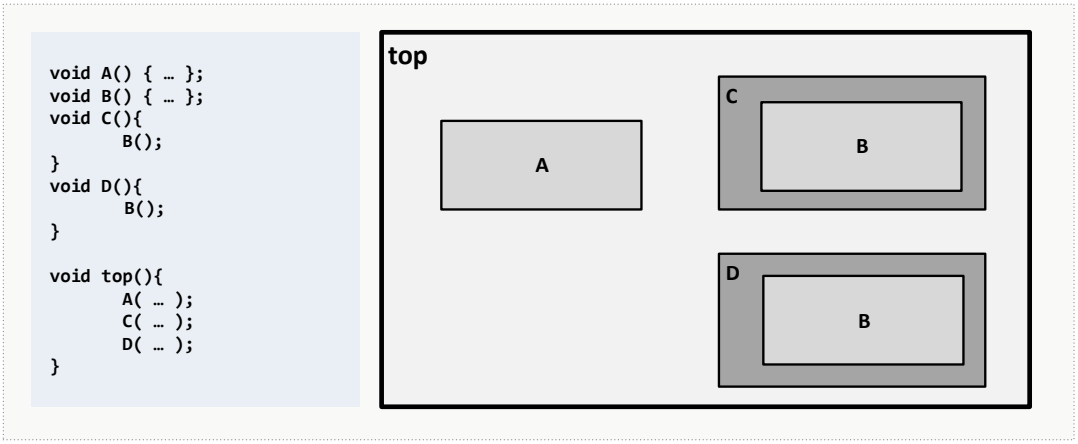


Figura 26. Jerarquía del diseño

Además de AXI, Vivado HLS soporta diferentes tipos de interfaces a nivel de bloque (ap_ctrl_none, ap_ctrl_hs, ap_ctrl_chain, ap_none, ap_hs), permitiendo que el diseñador genere su propio protocolo (se apoya además en directivas que protegen la zona de código que se desea se implemente a nivel *Bus Cycle Accurate* – BCA). Además genera interfaces para memoria, incluyendo BRAM y para FIFOs.

La utilización de Vivado HLS con estilos de modelado desde C/C++ y desde SystemC presenta diferencias notables. Si bien el estilo de modelado algorítmico es amigable y simplifica el diseño para casos de complejidad baja o media, para el caso de algoritmos complejos la eficacia y la eficiencia del proceso de modelado aconsejan la utilización de SystemC para facilitar el diseño modular. Esta es una práctica de diseño muy recomendable. La utilización de SystemC facilita la creación de modelos con ejecución concurrente y otro aspecto clave es poder simular el sistema completo sin que sea necesario transformarlo previamente en un modelo RTL. La generación de interfaces estandarizadas y la fácil integración del IP generado en el flujo de diseño de Vivado, al generar la información tanto hardware como *software* (por ejemplo direcciones internas de los registros mapeados en AXI), hacen de Vivado HLS un flujo prácticamente obligado cuando el diseño se vaya a implementar en FPGAs de Xilinx de última generación.

Un ejemplo de experiencia propia de diseño con Vivado HLS se puede encontrar en [99]. En este trabajo se ha usado Vivado HLS y toda su funcionalidad para implementar desde C/C++ un acelerador de búsqueda de caracteres en un flujo de datos recibido a través de una interfaz Ethernet soportando TCP/IP. El acelerador usa una interfaz AXI4 *Stream* para conectarse al resto de la plataforma. Se ha usado un dispositivo Zynq para implementar el diseño, con mejora en la calidad de los resultados y en el soporte a SystemC.

En un segundo trabajo [100] se ha enfrentado Vivado HLS a la evaluación de la reutilización de un diseño descrito en SystemC y sintetizado en Cadence CtoS para ser implementado también

en SystemC desde Vivado HLS. La experiencia obtenida nos ha permitido enfrentar dos flujos de diseño, Vivado HLS con un soporte maduro para diseños basados en C/C++ frente a Cadence CtoS claramente identificado con SystemC. En el trabajo no se ha explotado la capacidad de Vivado HLS para la síntesis de las interfaces ya que el diseño original ya dispone de su propio protocolo para la comunicación entre bloques (*request*/*validate*/data) o con el exterior (Xilinx *LocalLink* [101] orientado a *streaming*).

La adaptación del código fuente se ha realizado haciendo uso de las directivas de preprocesador de C para hacer compatibles ambas especificaciones (`#ifdef __SYNTHESIS__` para Vivado HLS y `#ifdef __CTOS__` para Cadence CtoS). Igualmente ha sido necesario capturar las estrategias de síntesis con las directivas de Vivado HLS, transformando los scripts Tcl de CtoS en otros equivalentes. Algunas adaptaciones adicionales consistieron en la modificación de las FIFOs por construcciones soportadas en Vivado HLS usando la librería TLM (sc_fifo_out<int> dout y sc_fifo_in<int> din). Otra modificación requerida es la transformación de módulos internos en funciones y la transformación de los procesos SC_THREAD por SC_CTHREAD. Por último podemos indicar que ha sido necesario modificar las interfaces con memoria para inferir memorias de bloque durante la síntesis.

Vivado HLS es una herramienta de síntesis en evolución, con una curva de aprendizaje razonable. Sin embargo, para un diseñador con experiencia en el diseño RTL, es necesario cambiar de paradigma de modelado desde una visión concurrente a otra secuencial. La utilización de SystemC como lenguaje de entrada supone un paso intermedio dentro de este proceso de adaptación.

2.4.4. C-to-Silicon (CtoS)

C-to-Silicon (CtoS) usa C/C++ y SystemC como lenguaje de entrada para capturar el diseño. Para diseños escritos en C, C++ o TLM 1.0 se genera un adaptador que encapsula el diseño y permite incluir implementaciones de los bloques TLM que describen las interfaces. CtoS soporta tanto el diseño de ASICs como de FPGAS y en este último caso soporta los principales fabricantes (Xilinx y Altera). Presenta una interfaz de usuario gráfica acompañada de funciones Tcl lo que permite automatizar el proceso de síntesis mediante *scripts*.

En relación a la generación de la ruta de datos, soporta librerías SystemC de aritmética de punto fijo de un número de bits arbitrario. Las tareas básicas incluyen: romper los bucles combinacionales, desenrollado total/parcial de los bucles, segmentación de la ruta de datos, asignación de variables a memoria, síntesis de la memoria en función de las interfaces de memoria definidas por el usuario, reorganización de la memoria, protección de zonas dedicadas a protocolo para soportar modelado BCA en SystemC, planificación automática o definida por el usuario, asignación de registros y generación de código Verilog a nivel RTL, así como el correspondiente *wrapper* de simulación en el simulador IUS de Cadence (figura 27).

Las librerías de diseño se generan apoyándose en herramientas de síntesis lógica (RC Compiler para librerías ASIC definidas en formato Liberty o herramientas del fabricante de la FPGA, XST en el caso de Xilinx) durante la fase de asignación de recursos para el diseño.

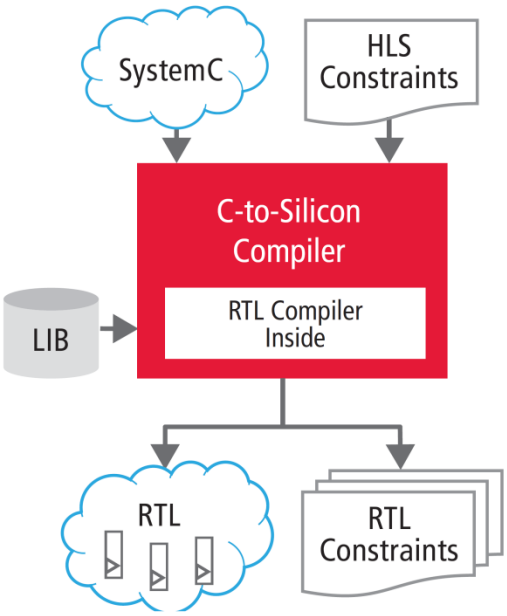


Figura 27. Diagrama de bloques de CtoS

CtoS utiliza un CDFG para mostrar el comportamiento de las funciones incluidas en el diseño, donde es posible imponer restricciones de diverso tipo, incluidas las de latencia (figura 28). Presenta un conjunto de utilidades para el análisis temporal, de área y potencia del diseño. Además del diseño RTL, exporta otros productos tales como el fichero de restricciones para su uso durante el proceso de síntesis lógica.

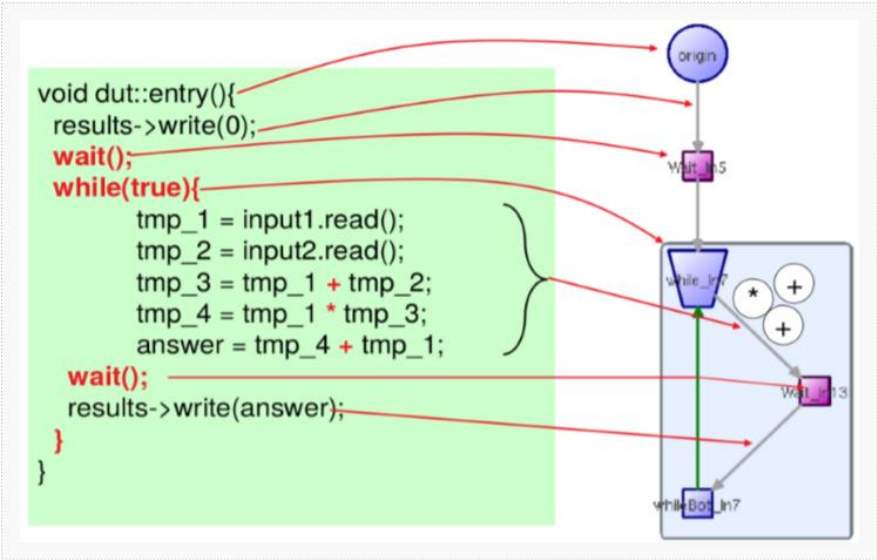


Figura 28. Ejemplo de CDFG de Cadence CtoS

La herramienta posee una interfaz de análisis enlazada con el código fuente en SystemC. La interfaz incluye el análisis de la ruta crítica, el mapa de memoria y de potencia, entre otras ayudas (figura 29).

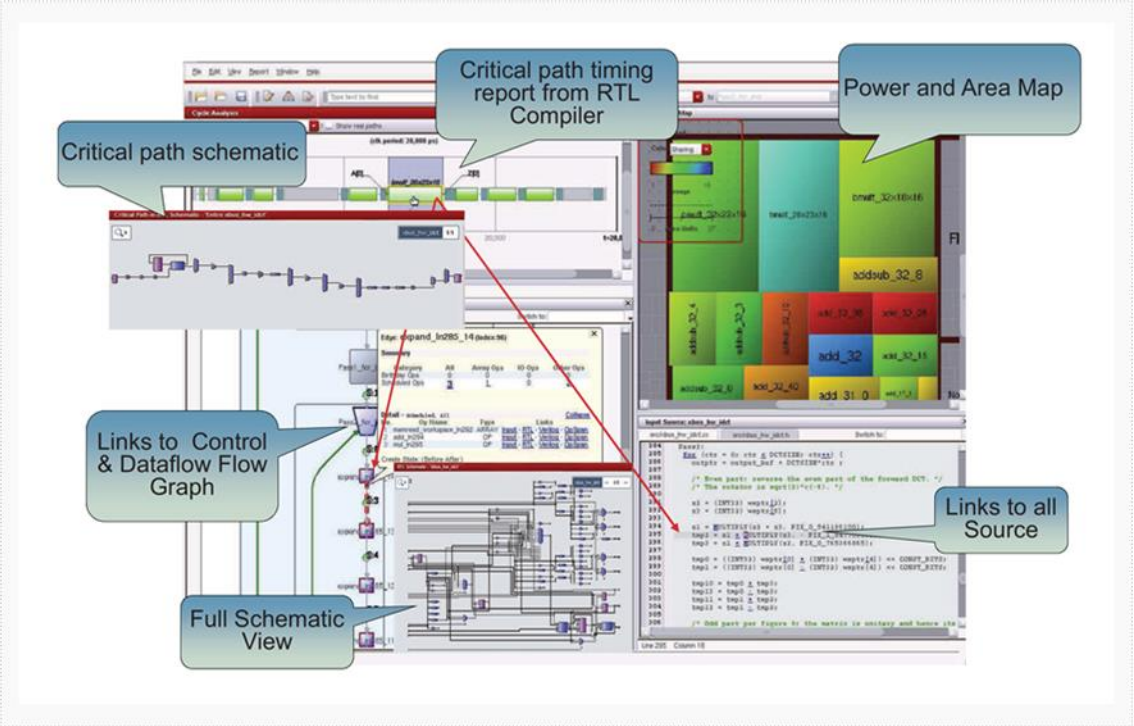


Figura 29. Interfaz de análisis de Cadence CtoS

La literatura muestra que CtoS puede conseguir diseños a partir de un modelo con una reducción x3 de las líneas de código y conseguir una reducción del 35% en área y 51% menos de potencia, con un incremento del 35% de prestaciones, que un diseño RTL a partir de las mismas especificaciones [102].

Nuestra experiencia de utilización de este entorno es amplia, demostrando el flujo de diseño basado en SystemC hasta su implementación hardware sobre FPGA y sobre ASIC [53, 54] partiendo de SystemC. En [105] se ha evaluado la capacidad de CtoS para soportar el modelado TLM orientado a la síntesis. En este caso se ha dotado de interfaz TLM a un bloque de predicción de movimiento de un decodificador de vídeo mediante los métodos `get()` y `put()` de la interfaz y se han modificado los puertos orientados al *handshake* de señales por FIFOs. Los resultados demuestran que las prestaciones de este estilo de modelado no se ven afectadas por el estilo de modelado y por el contrario hace que el proceso de modelado sea más simple.

Como esta herramienta ha sido extensivamente usada en este trabajo de investigación se mostrarán más detalles y conclusiones sobre mejores prácticas en los siguientes capítulos.

2.4.5. Otros entornos de síntesis de alto nivel

Aparte de los entornos citados que representan el estado de la industria EDA en cuanto a la SAN, existen diferentes grupos de investigación que están generando nuevas herramientas, generalmente de código abierto, que cubren diferentes aspectos ya sean desde el punto de vista metodológico o para aplicarlos en dominios especializados.

GAUT [38] es una herramienta de código abierto para la SAN. Parte de una especificación precisa a nivel de bits en C/C++ y genera de forma automática una arquitectura RTL en VHDL para implementarla en FPGAs (usando Quartus II o Xilinx ISE). Genera modelos de simulación

TLM y *Cycle-Accurate Bit-Accurate* (CABA) en SystemC para integrarlos en un prototipo virtual. Está orientado hacia el procesamiento de señales. Su modelo de arquitectura integra unidades de procesamiento, una unidad de memoria, una unidad de comunicación e interfaces GALS/LIS (*Globally Asyncronous Locally Synchronous/Latency Insensitive System*).

ROCCC (Riverside Optimizing Compiler for Configurable Circuits) [106] es una herramienta de código abierto para la SAN desarrollado en la Universidad de California Riverside. Utiliza Eclipse como entorno de diseño desde donde se ejecuta ROCCC. Utiliza un subconjunto de construcciones de C como lenguaje de entrada del algoritmo. Soporta el tipo de datos de enteros de ancho arbitrario usando construcciones `typedef` del lenguaje. Genera una descripción VHDL de salida que puede ser implementada en FPGAs. El objetivo es extraer las funciones, o parte de ellas, de la aplicación con mayor carga computacional y mapearlas en aceleradores *hardware*. Soporta el concepto de *Smart buffers* que hace un análisis de localidad de los datos, almacenando en memoria local aquellos datos reutilizados por el algoritmo, reduciendo así el tráfico con memoria. Esta técnica es de interés en aplicaciones de procesamiento de vídeo o de comunicaciones inalámbricas o celulares.

LegUp [107] acepta un algoritmo escrito en C estándar y lo compila en una plataforma heterogénea basada en FPGA que incluye un procesador *soft-core* MIPS y aceleradores *hardware* que se comunican a través de interfaces de buses estandarizados. El flujo de diseño incluye un compilador estándar de C que genera código para su ejecución en el procesador MIPS. Una vez se compila el código, este se perfila para extraer los núcleos de computación elevada que se transforman mediante técnicas de SAN en aceleradores *hardware*. LegUP está basado en la infraestructura de compilación LLVM (*Low-level Virtual Machine*) que transforma el código en una representación intermedia IR de tal forma que, mediante nuevos procesos de transformación y optimización, se convierte en una representación muy cercana al *hardware*. A partir de esta representación se realizan las operaciones de planificación, asignación de funciones y recursos. El resultado final es la obtención de una solución *hardware*/*software*, siendo de interés para este trabajo los aspectos relativos a la implementación de los aceleradores.

2.5. Retos de la síntesis de alto nivel y hoja de ruta que puede preverse

En este apartado vamos a describir los principales retos que debe acometer aún la SAN para consolidarse como metodología de diseño plenamente aceptada en la industria que cubra efectivamente la distancia entra la capacidad de integración existente y la capacidad de diseño.

A lo largo de este capítulo se ha explicado cómo diferentes entornos de diseño abordan el problema de transformar un algoritmo, con un comportamiento intrínsecamente secuencial, a una arquitectura *hardware* que ejecuta de forma concurrente diferentes procesos en los bloques que la forman. En el núcleo del problema, todas las herramientas realizan tareas de análisis del código, planificación, asignación de recursos y selección de unidades funcionales y, por último, realizan la tarea de generar código HDL a nivel RTL.

Sin embargo las principales diferencias estriban en las restricciones impuestas por los dominios de aplicación, ya que condicionan el tipo de solución arquitectural y por tanto imponen restricciones en el espacio de soluciones a explorar por la SAN.

Existen varios factores que están favoreciendo la plena adopción de la metodología de diseño de SAN, entre ellos:

- La necesidad de crear el diseño en niveles más altos de abstracción para poder cubrir la ya inmensa y creciente capacidad de integración del silicio
- Mejorar la reutilización del diseño adoptando los niveles de comportamiento
- La necesidad de disponer de modelos de alto nivel que faciliten la verificación
- La utilización masiva de aceleradores implementados en *hardware* que requieren los algoritmos actuales para una eficiente utilización del binomio tiempo y energía
- La utilización de plataformas heterogéneas que requieren que una parte del problema se solucione y desarrolle en un lenguaje de alto nivel
- En todo caso, la plena adopción de una nueva metodología está condicionada por varios factores claves: la calidad de los resultados obtenidos, la reducción de los tiempos de puesta en mercado del producto y de sus costes y la curva de aprendizaje de la metodología. A continuación se muestran algunos retos que aún debe abordar la metodología de SAN.

2.5.1. Lenguajes para SAN

Existen diferentes opciones para la captura del modelo de entrada a la SAN. C/C++ presenta facilidad de aprendizaje y velocidad de simulación mientras que SystemC incluye además características para dar soporte al diseño *hardware* a nivel de sistema. Por otro lado BSV (Bluespec SystemVerilog) soporta el nivel arquitectural para establecer estructuras del control.

BSV (Bluespec SystemVerilog) incluye varias características de un lenguaje de alto nivel (Haskell) y las características de SystemVerilog. El objetivo es reducir la distancia desde el modelo de comportamiento al modelo RTL. El lenguaje adopta el aspecto de transacciones atómicas para expresar y describir comportamientos concurrentes complejos. La ventaja de esta propuesta es ofrecer un lenguaje común en varios niveles de abstracción.

MATLAB y especialmente Simulink ha sido una herramienta ampliamente usada para el diseño algorítmico debido a que se trata de un lenguaje maduro con módulos especializados (*toolboxes*), conjuntamente con la posibilidad de integrar código C. El problema se centra entonces en migrar a *hardware* el algoritmo o parte de él.

Algunos proveedores de FPGAs han desarrollado *toolkits* de Simulink que permite sintetizar de forma eficiente algunas funciones predefinidas en sus dispositivos. Algunos ejemplos son Altera DSP Builder o Xilinx Vivado System Generator. Aunque se trata de entornos fáciles de usar, estas herramientas están limitadas a los bloques suministrados por el fabricante y están disponibles únicamente para dispositivos propios. Existen otros intentos de crear una herramienta genérica de síntesis a partir de código M de Matlab.

Podemos entonces resumir los requerimientos básicos que debe poseer un lenguaje especializado para SAN[108]. El primer requisito de un lenguaje de entrada a la SAN es que haya sido utilizado para escribir los algoritmos que se van a implementar en *hardware*. Para este requerimiento C/C++ o lenguajes basados en ellos es la elección apropiada. Se trata de lenguajes maduros, con soporte de diferentes herramientas tales como depuradores, compiladores, optimizadores, analizadores estáticos y dinámicos. El punto de partida es el amplio conjunto de algoritmos en C que es posible implementar en *hardware*.

El segundo requisito es que el lenguaje debe soportar mecanismos de abstracción. Si el lenguaje posee un único nivel de abstracción no es apto para la gran variedad de diseño *hardware* existente. Por tanto la idea es usar un lenguaje orientado a objetos (C++).

El tercer requisito es que el lenguaje debe soportar el nivel de abstracción relacionado con el *hardware*, proporcionando una conexión entre el modelo de alto nivel y la implementación *hardware*. Esto implica soportar una representación que se mapee directamente en RTL, proporcionando una integración directa entre los diferentes niveles de abstracción desde RTL. Los elementos que se requieren para esta integración son el soporte a jerarquía, concurrencia y precisión a nivel de bits. Sin estos elementos no es posible representar todos los aspectos del diseño *hardware*.

SystemC ha sido concebido para incluir estas características. Se trata de un conjunto de clases C++ que soportan las nociones de concurrencia, tiempo y otras características tales como aritmética de punto fijo. Además incluye un *kernel* de simulación orientado a eventos.

ANSI C++ es el lenguaje de modelado más ampliamente usado a nivel algorítmico en los niveles de abstracción de sistema. Proporciona clases y plantillas para modelar tipos de datos a nivel de precisión de bits, y soporta encapsulación modular y parametrización. La especificación secuencial en C++ es compacta, fácil de depurar y verificar y ofrece buena velocidad de simulación. Por tanto es un buen candidato para crear arquitecturas e interfaces *hardware* utilizando SAN.

SystemC se apoya en C++ como lenguaje y añade características específicas para el modelado de interfaces *hardware* y concurrencia para la integración y verificación de sistemas complejos, que incluyen buses de comunicación y componentes modelados a nivel RTL. La metodología TLM soportada por SystemC separa las especificaciones de los núcleos de computación sin información temporal por un lado, de las especificaciones de interfaces precisas a nivel de ciclo por otro lado. En la tabla 1 se muestran las principales características aportadas al diseño de alto nivel por los lenguajes citados.

La implementación eficiente desde algoritmos en C/C++ requiere un compilador con capacidad de paralelización y optimización para cumplir los objetivos de área, latencias y potencia requeridos. Todos los detalles *hardware* pueden inferirse desde el modelo de alto nivel y las restricciones de diseño.

El modelado a nivel de sistema y el desarrollo temprano del *software* requiere modelos que reflejan el paralelismo del diseño *hardware*. Aunque este paralelismo no se expresa directamente en la especificación C/C++, el modelo TLM en SystemC, que sí incluye paralelismo

hardware de forma explícita (microarquitectura) puede ser generado de forma automática por el compilador cumpliendo con las especificaciones.

Tabla 1. Soporte de los lenguajes a diferentes aspectos del diseño en alto nivel (adaptada de [26])

Lenguaje	Construcciones del lenguaje	Características aportadas al diseño de alto nivel
C	Tipos enteros de precisión arbitraria	Mejora de la calidad de los resultados al obtener diseños precisos a nivel de bits
	Tipos de datos para representación de reales con coma flotante	Soportan la aritmética de coma flotante
	Estructura y llamada de funciones en C	Soporte al diseño jerárquico
	Punteros	Eficiencia y flexibilidad en el uso de los recursos para el acceso a datos
	Estructuras y uniones	Encapsulación de datos
C++	Tipos de coma fija	Aritmética de coma fija, Compromiso precisión-costo
	Plantillas de C++	Diseño parametrizable
	Clases	Modelado orientado a objetos (encapsulado, herencia, polimorfismo …)
SystemC	Módulos y Procesos	Jerarquía y concurrencia a nivel de bloques
	Relojes SystemC	Soporte diseño síncrono y diseños con múltiples dominios de reloj. Protocolos
	TLM	Separación de la computación y comunicación. Simulación más rápida del modelo. Prototipos virtuales con precisión de ciclo
	Modelo de eventos	Simulación. Coexistencia de varios niveles de abstracción

Un lenguaje adecuado para SAN debe soportar por un lado descripciones funcionales, ya sean con información temporal o no, e igualmente soportar por otro lado la tecnología de síntesis apropiada para producir rutas de datos y lógica de control optimizadas a nivel RTL.

Los niveles de abstracción más elevados separan los núcleos de computación (*untimed*) de la implementación *hardware* ya que estos núcleos son más fáciles de simular y depurar, dejando a las herramientas de SAN la generación de las interfaces correspondientes, lo que conlleva la posibilidad de obtener varios tipos de arquitectura a partir de la misma descripción inicial.

TLM típicamente separa la descripción funcional de los módulos respecto de los medios y métodos usados para comunicarse entre ellos (separación de la computación de la comunicación). Ello permite un alto nivel de abstracción que puede usarse para maximizar las velocidades de simulación y para minimizar los tiempos de exploración de la arquitectura. Tanto la funcionalidad como la comunicación pueden ser refinadas de forma gradual hasta conseguir una comunicación precisa a nivel de señales.

Los núcleos de computación (*untimed C*) pueden usarse incorporados en un nivel TLM. Así, los algoritmos escritos a nivel TLM con SystemC aportan en sí mismos un medio de enlace entre

el código de alto nivel y su implementación RTL. Además la herramienta puede usarse durante la verificación mediante un bloque TLM que integre el bloque RTL optimizado.

Los SoCs diseñados hoy en día reutilizan del orden del 70% de los diseños a los que se añaden otros propios para diferenciar los productos y añadir características propias o nuevas del producto. Los nuevos bloques pueden ser tanto aceleradores *hardware* como bloques de control. Los aceleradores de aplicaciones implementados en *hardware* proporcionan ventajas en términos de prestaciones y de ahorro de potencia en relación a su implementación *software*. Estas funciones se diseñan normalmente en C/C++ para verificar su funcionalidad y para aprovechar las ventajas de la velocidad de simulación. La implementación de estas funciones directamente desde C/C++ proporciona el nivel más alto de abstracción y da la mayor ventaja en productividad del diseño.

Para el caso del diseño de aceleradores, el diseño del algoritmo está condicionado por las restricciones impuestas por la arquitectura particular en la que se va a ejecutar. A diferencia del diseño *software*, donde la arquitectura de la plataforma es fija, en el diseño de alto nivel, la arquitectura y el algoritmo se crean conjuntamente.

En contraste con los aceleradores de aplicaciones, los bloques de control interaccionan con el entorno y requieren una especificación detallada a nivel de ciclo. Las soluciones integradas que soporten ambos aspectos -bloques aceleradores (*dataflow)* y bloques de control a partir de especificaciones abstractas- son necesarias para los sistemas actuales.

Las herramientas poseen directivas para controlar la latencia, el tiempo de ciclo y el comportamiento temporal del diseño o de parte de él. Igualmente es posible interactuar para controlar la asignación de recursos, la asignación de entradas y salidas, la arquitectura de memoria y el uso de librerías de IPs.

El reto es reducir el tiempo de puesta en mercado del sistema electrónico, tanto del *hardware* como del *software* en forma de SoC complejos. Esto requiere lenguajes que vayan más allá del diseño lógico y den soporte al diseño del *software* empotrado mediante la creación de prototipos virtuales (VP) de altas prestaciones. SystemC permite la interoperatibilidad y estandarización de los modelos utilizados.

La verificación supone el mayor coste en la producción de nuevos SoC, por lo que la utilización de niveles tales como TLM, donde sea posible depurar el diseño a alta velocidad es una ventaja de productividad. Esto requiere que se adopten técnicas de SAN para los diferentes IPs incluidos en el diseño. Para poder realizar una exploración arquitectural del diseño, evaluar sus prestaciones y realizar su verificación, es necesario ejecutar las aplicaciones reales, a veces incluyendo un sistema operativo, sobre diferentes configuraciones que representan el sistema real.

Cada configuración incluye modelos, bancos de test, IPs usados para realizar la implementación en diferentes niveles de abstracción, por lo que se requiere un lenguaje estandarizado y aceptado por la comunidad de diseñadores para incluir esta diversidad.

Los aspectos de verificación son claves para validar el diseño obtenido. Por tanto las herramientas actuales tratan de verificar el diseño en dos etapas: durante la creación del

algoritmo que representa el código de entrada del diseño y durante una segunda etapa en la que se verifica el diseño RTL obtenido.

El primer paso se realiza mediante simulación mientras que el segundo se puede realizar mediante simulación pero también mediante verificación formal. El nivel de abstracción del modelo de entrada ofrece una velocidad de simulación mayor que el de los HDLs. Para verificar el modelo RTL la mayor parte de los flujos de diseño ofrecen procesos automáticos que permiten la reutilización de los bancos de test, verificando el modelo RTL contra el modelo funcional inicial.

La productividad, calidad de resultados (QoR) facilidad de verificación y reutilización son razones claves por las que los diseñadores han adoptado métodos de diseño basados en síntesis de alto nivel. El soporte del lenguaje es relevante si puede ayudar a estos objetivos. Las herramientas de SAN deben soportar el mayor conjunto de lenguajes de entrada posible que ayuden a describir otros aspectos del flujo de diseño.

Podemos concluir que la utilización de un lenguaje de especificación de alto nivel que permita la reutilización del diseño y soporte el paralelismo intrínseco del *hardware* contribuirá a la consolidación de la metodología. Debido a las continuas mejoras en las técnicas de compilación, se han aprovechado las optimizaciones introducidas en los compiladores para transferir de forma rápida esa evolución al diseño *hardware*. Esto requiere la especificación de algoritmos precisos a nivel de bits y la identificación o extracción de tareas que se puedan paralelizar, por la ausencia de dependencia de datos. C/C++ y SystemC suponen la alternativa elegida y consolidada como lenguaje por lo que el reto es establecer mecanismos de convergencia para la compatibilidad entre herramientas de síntesis, facilitando la reutilización de los IPs definidos a nivel de comportamiento (figura 30).

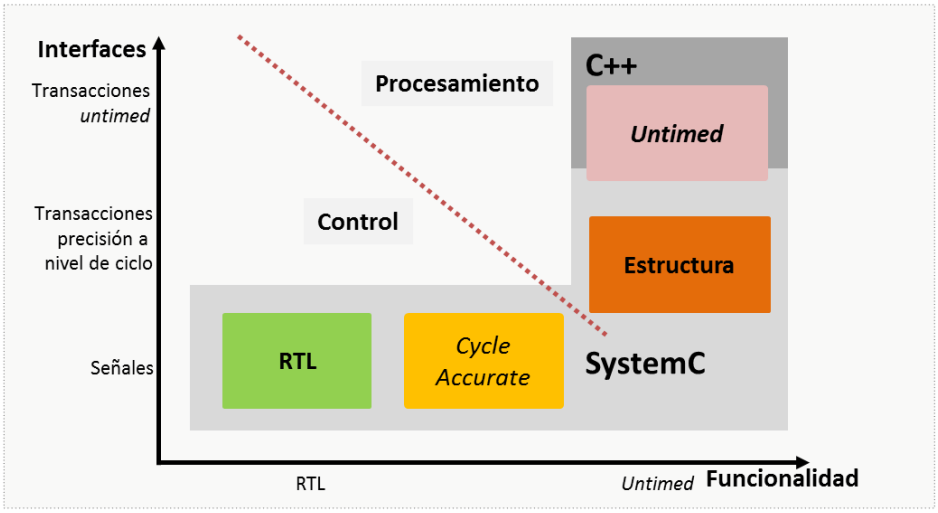


Figura 30. Niveles de modelado de C++ y SystemC (adaptada de [109])

2.5.2. Verificación del diseño a sintetizar

Como se ha indicado a lo largo de este capítulo la SAN presenta muchas ventajas sobre el diseño RTL, entre ellas la mejora en la eficiencia de la verificación del diseño en niveles de abstracción más altos. Parece evidente que en estos niveles de abstracción el número de errores

que se producen será menor pero siempre va a ser necesario verificar las transformaciones que se aplican al diseño desde su creación hasta que es implementado.

La mayor parte de las metodologías que utilizan el nivel algorítmico como entrada del diseño sin aportar información temporal (modelos *untimed*) requieren su transformación hacia nivel RTL para realizar la verificación. Ello se debe a que durante la transformación del diseño se han añadido dos aspectos claves y diferenciadores: el dominio estructural (modularidad, jerarquía, comunicaciones) y el dominio temporal (concurrencia, protocolos, entre otros). Es decir en RTL disponemos de un modelo preciso a nivel de ciclos de bus y preciso a nivel de bits (*Cycle Accurate Bus Accurate* – CABA) no existente al nivel algorítmico.

Esta falta de información en los dominios indicados trae consigo que, dependiendo de la complejidad del diseño, la verificación realizada a nivel algorítmico no represente el comportamiento de la arquitectura mapeada. La solución adoptada por la mayoría de los entornos de SAN estudiados es realizar la verificación del diseño solo a nivel RTL. Esto implica la necesidad de introducir adaptadores que transformen las señales RTL para obtener los valores de E/S del modelo simulado para su utilización en el *testbench.* Por tanto se quiere verificar que hay concordancia entre el modelo de comportamiento y el modelo RTL obtenido. Esta aproximación a la verificación del modelo de alto nivel es complicada y tediosa, difícil de verificar y produce resultados inesperados, especialmente en el dominio temporal, y más concretamente en los protocolos de comunicación entre bloques.

El modelo C/C++ no se puede depurar hasta que no se realice la síntesis de alto nivel ya que no se dispone de estructuras de comunicación ni de modelo temporal definido. Sin embargo, el encapsulado del algoritmo en SystemC facilita la verificación del modelo antes de ser sintetizado (figura 31). El modelo SystemC dispone de la información necesaria para dar soporte a los dominios de representación del modelo *hardware*.

La idea clave aquí es que el diseño debe ser verificado antes de su síntesis de alto nivel de tal forma que el diseño enviado a síntesis esté libre de errores [110] (figura 31). Para el incremento efectivo de la productividad es necesario verificar el diseño antes de su síntesis de alto nivel, y puede hacerse en SystemC. Como puede observarse, los modelos a nivel TLM como los descritos en [111][112][113] facilitan la simulación del sistema.

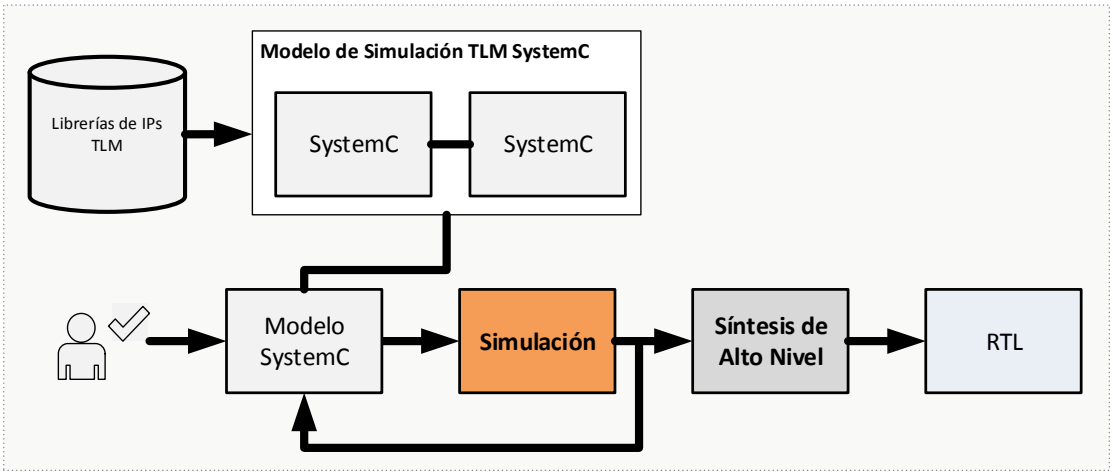


Figura 31. Simulación y síntesis del Modelo SystemC

2.5.3. Plataformas heterogéneas personalizadas

Numerosas aplicaciones de aceleración se están desarrollando bajo el paradigma de las plataformas heterogéneas personalizadas (*Customizable Heterogenuos Platform* – CHP) mediante la inclusión de aceleradores adaptados a la aplicación de tal manera que la mayor parte del cómputo se produce en los aceleradores, que son más eficientes en términos de energía. Por ejemplo en casos típicos de consumo de potencia, las prestaciones de transferencias de datos en una FPGA están en el orden de 2,5 Gbps/W mientras que en un procesador de propósito general dicha transferencia baja a 0,015 Gpbs/W [114].

En este escenario, la síntesis de alto nivel presenta ventajas bien definidas para la creación de dichos aceleradores a partir de algoritmos escritos en C/C++. En la literatura se pueden encontrar referencias a aplicaciones para el procesamiento que usan aceleradores basados en FPGA, creados a partir de especificaciones de alto nivel usando SAN. Algunos ejemplos son: identificación de ADN [115], aplicaciones financieras [116], de optimización de redes inalámbricas [117], de aceleración en bases de datos [118] o de computación en la nube [18]. Existen aproximaciones para crear aceleradores implementados mediante técnicas de reconfiguración parcial de la FPGA creando el concepto de FPGA Virtual [119]. Para ello es clave la utilización de técnicas de SAN automatizadas que resuelvan algunos problemas actuales que presentan estas plataformas heterogéneas:

- Análisis de la dependencia de datos para extraer el paralelismo necesario que guie al planificador.
- Inferencia de la arquitectura de memoria necesaria para optimizar el uso de recursos en la FPGA, especialmente para aplicaciones intensivas en datos o para tener en cuenta las restricciones físicas impuestas por la tecnología de implementación (número de puertos, tamaños de los bloques). Estas restricciones provocan problemas de contención y obligan a generar arquitecturas no optimizadas en términos de latencia.
- Aunque en el modelado TLM se separa la computación de la comunicación, lo que ocurre durante la fase de SAN es que hay una dependencia mutua: la elección de la arquitectura de comunicación influirá en la planificación y asignación de recursos y viceversa. Una comunicación orientada al flujo de datos generará una arquitectura también orientada a flujo de datos, con capacidad de optimización como ruta de datos segmentada. Pero esto también influye en la arquitectura de memoria. Se requiere por tanto una optimización de ambos aspectos de forma concurrente.
- Es necesario hacer una aproximación holística al problema, optimizando la transferencia de datos y el acelerador como un todo. En [18] se apuesta por integrar lenguajes especializados, tales como OpenCL, con la transformación automática del código C en aceleradores *hardware* haciendo uso de herramientas de síntesis de alto nivel para incrementar la productividad del diseño, especialmente para el caso de las FPGAs.

2.5.4. Convergencia temporal y calidad de los resultados

La calidad de los resultados (QoR) es un concepto recurrente en diseño electrónico. Mantener su coherencia entre diferentes niveles de abstracción ha sido un reto para todas las metodologías de diseño. Durante la SAN se toman decisiones que afectan de forma notable a las prestaciones finales del sistema, tal como se indicó en la figura 11. Por lo que se debe disponer de información con el grado de precisión necesaria para evitar errores que se propaguen al resto del flujo de diseño. En posteriores capítulos se incidirá en esta observación con datos cuantitativos al respecto sobre parámetros temporales, de área y de potencia.

Avances recientes en las herramientas de SAN tienen como principal objetivo la mejora de estos resultados. Por ejemplo, Cadence ha integrado los entornos CtoS y Forte Cynthesizer en su nuevo entono denominado Stratus HLS [120] (figura 32).

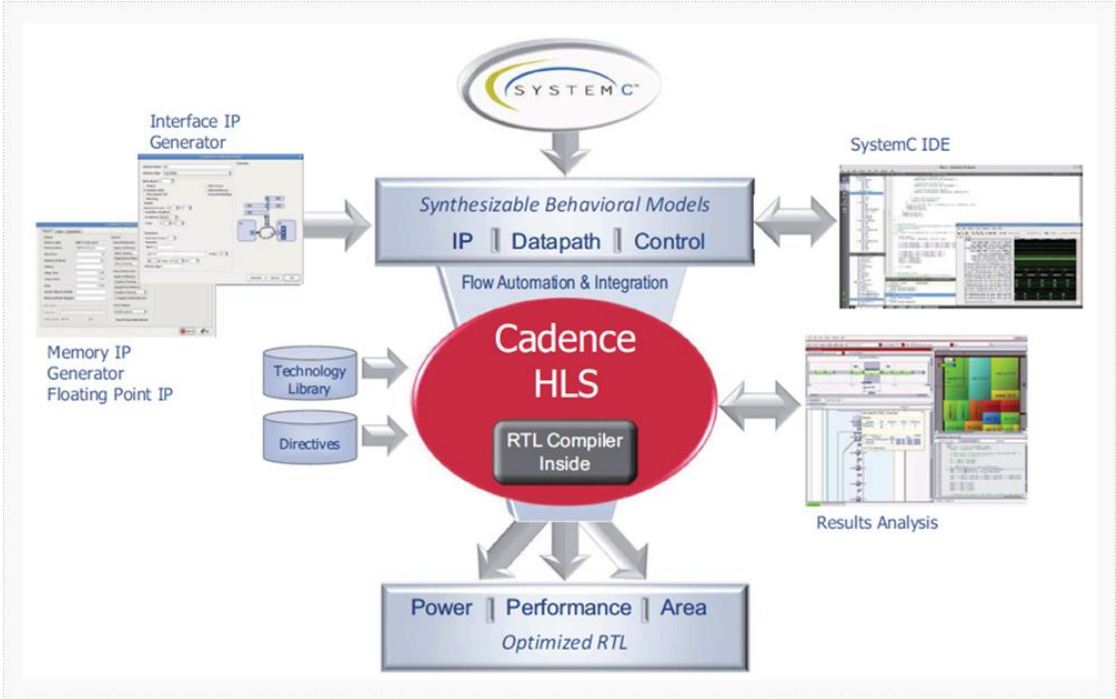


Figura 32. Entorno de síntesis Cadence Stratus

Dispone de una librería de IPs TLM sintetizables, que incluye:

- Tipos de datos para soportar aritmética de coma fija, de números complejos y de coma flotante
- Bloques de integración de la plataforma: librerías matemáticas, FIFOs, Memorias, CDC (*Clock Domain Crossing*), Buffers
- Bloques genéricos para comunicación: canales de comunicación punto a punto con la API put()/get()
- Interfaces de buses: genéricas, AMBA AXI3, AXI4, AXI4-Lite y las creadas por el usuario.

En esta evolución de las herramientas, uno de los aspectos más cuidados es la mejora en la convergencia temporal (*Timing Closure*). En las experiencias de trabajo con CtoS hemos encontrado notables divergencias en este sentido por lo que es necesario usar de forma sistemática la interacción e iteración entre síntesis de alto nivel y síntesis lógica para tener datos temporales fiables, por lo que será necesaria su evaluación detallada por parte del diseñador. Igualmente Vivado HLS mejora de forma notable dichos resultados según diferentes versiones de la herramienta.

Esta misma situación se produce en el caso de Vivado HLS con mejoras significativas en las prestaciones obtenidas y en la compatibilidad del lenguaje SystemC, entre otras.

2.6. Reflexiones sobre los mejores flujos y prácticas en SAN

En esta sección se presentan un conjunto de reflexiones extraídas de la experiencia acumulada en la realización de diferentes diseños con técnicas de SAN. Estas experiencias y prácticas se aplican y enriquecen con los diseños presentados en los capítulos 3 y 4. Se van a estructurar en **principios** e **ideas sobre el estilo de modelado y síntesis**.

Empezamos presentando los **principios básicos (**tabla 2**):**

1) El primer principio básico en el proceso de determinar el contenido del acelerador *hardware* a implementar usando SAN es determinar qué parte de la aplicación es necesario acelerar. Este paso no está directamente integrado dentro de las tareas de la SAN, pero está llamando la atención de los investigadores por su impacto directo en la complejidad del proceso siguiente. Esta partición *hardware*/*software* utiliza técnicas de perfilado de la aplicación y del establecimiento de los mecanismos de comunicación *hardware*/*software*. Estas tareas están no obstante fuera del alcance de este trabajo y no nos detenemos en ellas.

   En este primer paso es importante establecer como práctica que las decisiones sobre la arquitectura del sistema se deben tomar en alto nivel y no se deben retrasar al nivel RTL, una vez codificado en un HDL. La SAN facilita la evaluación de las posibles soluciones arquitecturales para elegir la más apropiada con respecto a las restricciones del diseño.

2) Un segundo principio es conocer la tecnología de implementación objetivo, pues permite extraer mejores resultados del diseño. Por ejemplo, tener en cuenta el número de puertos disponibles en las memorias de una FPGA condiciona totalmente el número de lecturas/escrituras simultáneas desde memoria. Conocer la tecnología de implementación permite tener claro el objetivo de los resultados esperados (memorias, DSPs, registros, …).

3) Un tercer principio es identificar de forma temprana problemas asociados a la calidad del código C/C++/SystemC para identificar problemas que afectan a los parámetros PPA (Prestaciones, Potencia, Área) del diseño final. Entre estos aspectos podemos citar la detección de código muerto, errores por lectura de memoria no inicializadas, casos no cubiertos en múltiples ramas de decisión, errores de desbordamiento y de división por cero entre otros.

4) En cuarto lugar hay que determinar la complejidad del diseño del sistema y la elección de la metodología para abordarla. Utilizar metodologías *top-down* pueden mejorar la calidad de las prestaciones al realizar una optimización global del sistema, pero puede conducir a tiempos de síntesis prohibitivos para el desarrollo del proyecto. Es necesario por tanto establecer la correspondiente partición del sistema, teniendo en cuenta principios de localidad en los datos, minimizando por tanto el flujo de datos. En su caso esto determina el estilo de la arquitectura utilizada (tipo *dataflow*, por ejemplo) y los canales de comunicación a definir. Una partición adecuada tiene la ventaja de poder aprovechar las capacidades de multiprocesamiento disponibles en las herramientas de síntesis.

5) Como quinto principio básico indicamos que se debe sintetizar el diseño únicamente después de que se ha simulado. Esta situación se facilita por la utilización de SystemC para el modelado del sistema. Sin embargo, SystemC incluye construcciones específicas del diseño *hardware* y puede ser extraño para los desarrolladores de algoritmos, normalmente escritos en C/C++. Para realizar esta simulación es necesario disponer de un conjunto de modelos del sistema, normalmente a nivel TLM. Sin embargo, en fases avanzadas del diseño, cuando se ha añadido un protocolo de comunicación al modelo preciso a nivel de bus (BCA) se requiere combinar los modelos TLM con esos modelos BCA. Esta integración se realiza mediante adaptadores o *wrappers*. Elegir un entorno de síntesis capaz de generar de forma automática estos adaptadores en diferentes estados de la transformación del diseño, desde alto nivel a RTL, aumenta la productividad del flujo de diseño.

   Tal como se ha indicado anteriormente, SystemC soporta el nivel de abstracción necesario para la captura del algoritmo para su implementación *hardware*. Combina las características de C++ (alto nivel, orientado a objetos) con las construcciones *hardware* que facilitan al diseñador la representación estructural (jerarquía, señales, puertos, relojes, *reset*). Esta combinación de características permite la simulación concurrente de los bloques que componen la arquitectura. Una vez se ha verificado su funcionalidad, los modelos simulados se pasan al flujo de síntesis.

6) En sexto lugar es básico automatizar desde SAN la verificación del diseño RTL, se realiza integrando el código fuente (*Golden Model*) y el diseño generado (como dispositivo bajo test) en el *testbench*. En este caso, el simulador puede aplicar los mismos estímulos al modelo de referencia y al DUT. Si se parte de un modelo funcional sin información temporal, caso típico cuando se utiliza C/C++ para la especificación del diseño, la comparación se realiza a nivel de entradas/salidas. Si el modelo de partida contiene información temporal, la SAN puede cambiar el comportamiento temporal, por lo que hay que aplicar técnicas de comparación inteligentes entre ambos modelos. Si el modelo de referencia incluye información temporal, la SAN no debe modificar el protocolo definido para los buses y señales de entrada/salida.

Tabla 2. Resumen de principios básicos

Principios	Descripción
Principio 1	Determinación de las funciones a acelerar que pasarán al proceso de SAN
Principio 2	Tener nociones de la tecnologías de implementación (ASIC, FPGA)
Principio 3	Calidad del código. Abordar problemas asociados al código fuente
Principio 4	Partición del sistema. Utilización de metodologías *top-down* o *bottom-up*. Optimización y paralelización de la síntesis
Principio 5	Sintetizar después de simular. SystemC soporta nociones temporales y de estructura además de funcionales
Principio 6	Automatización de la verificación. Reutilización del *testbench*

A continuación se indican **algunas ideas prácticas sobre el estilo de modelado y de síntesis** con objeto de optimizar las prestaciones de área y temporales:

1) Usar tipos de datos ajustados a nivel de bits para reducir el consumo de área o recursos en el caso de la FPGA. La utilización de tipos definidos por el usuario facilita la parametrización y reutilización del diseño al nivel algorítmico.

2) Utilizar tipos de datos abstractos para facilitar el encapsulado de datos. Un paquete se puede definir con un *struct* o una clase de tal forma que el modelo SystemC abstrae los detalles existentes en los niveles más bajos de abstracción. La combinación con la parametrización hace que el diseño sea completamente reutilizable.

3) Para aplicaciones que soportan operaciones en coma flotante usar librerías optimizadas o ajustar los tipos de datos de punto fijo para minimizar el error de cuantificación.

4) Utilizar el método `range()` para acceder a partes de la palabra en vez de utilizar máscaras, respetando la ordenación de los bits.

5) Definir valores por defecto para las salidas de las funciones y puertos externos.

6) El código fuente debe estar estructurado en funciones para facilitar su encapsulado, en el caso de usar SystemC, o su estructura como módulos si se parte de un código C/C++.

7) Crear funciones para secciones de código que se repite para facilitar su implementación como módulos, evitando consumir recursos adicionales. Ello tiene como contrapartida la necesidad de definir múltiples fuentes de entrada (muxes) e incrementar el *fanout* de salida. Si la función se va a invocar múltiples veces en el mismo ciclo, es necesario replicar en el algoritmo principal su cuerpo (*inline*) para facilitar su utilización. En caso contrario, el planificador deberá programar varios ciclos para las operaciones, incrementando la latencia. Disponer de las funciones de forma encapsulada facilita su manipulación durante el proceso de síntesis de alto nivel. En determinadas circunstancias realizar el *inline* de la función facilita la reducción de la ruta crítica ya que el planificador podrá programar su funcionalidad

en varios pasos de control. Realizar el *inline* de la función incrementa el uso de recursos. Es necesario establecer compromisos entre objetivos de latencia y de recursos o área utilizados.

8) Los errores en la comunicación entre procesos suelen ser una fuente importante de malfuncionamiento del diseño. Es necesario conocer en detalle cuales son los criterios utilizados por la herramienta de síntesis para la comunicación entre procesos. Como norma general la comunicación se realiza mediante señales si es escrita por un proceso y leída por otro en el mismo ciclo de reloj. Sin embargo es posible usar variables compartidas globales siempre y cuando sea escrita por un proceso y leída por otro. Aquellas variables globales que son accedidas únicamente por un proceso se pueden implementar como memorias. Se debe planificar la partición del sistema y en esa partición definir los tipos de canales de comunicación entre bloques.

9) A veces en el modelado de algoritmos se utilizan técnicas de recursividad en la invocación de las funciones. En una implementación *hardware* no se pueden utilizar estas técnicas y deben sustituirse por bucles.

10) Se debe tener en cuenta el protocolo a utilizar para comunicar el IP con el mundo exterior. En los modelos más abstractos de deben utilizar estrategias de modelado TLM que faciliten la depuración de la funcionalidad del sistema. En fases más avanzadas del desarrollo del modelo, si el entorno dispone de métodos para la síntesis de interfaces, hay que adaptar el código a los requisitos de la herramienta. En algunos casos es preciso usar punteros para acceder a interfaces de memoria. Generalmente este proceso es controlado mediante directivas y chequeos de la síntesis de interfaces. Por ejemplo, si se define un puntero que es accedido de forma secuencial se genera una FIFO. En cambio si hay un acceso fuera de orden se genera una interfaz con memoria. La determinación del protocolo de comunicación generalmente es una restricción impuesta por cada diseño concreto por lo que se deben establecer los mecanismos de modelado convenientes para la reutilización del diseño a nivel de comportamiento.

11) Igualmente, desde el punto de vista del protocolo, si el diseñador establece zonas dedicadas al protocolo en el código debe guiar al planificador para que no modifique su comportamiento.

12) Dividir expresiones complejas en varias líneas facilita mantener la referencia entre el código fuente y la implementación RTL de cara a realizar su depurado. En este mismo sentido, etiquetar bucles y referencias facilita su referenciación.

13) Desde el punto de vista de la gestión del proyecto, centralizar parámetros y tipos de datos, separar modelado de estructura y funcionalidad, manteniendo un único bloque por fichero, facilita su mantenimiento y simplifica la referencia en todo el flujo de diseño.

14) Se ha indicado anteriormente que el diseño debe ser reutilizable a nivel de SAN. Para facilitar la portabilidad entre entornos de diseño la utilización del preprocesador de C facilita mantener el código actualizado. Igualmente se deben hacer convivir aquellas partes del modelo que están orientadas a depurado con el modelo sintetizable.

15) Las variables, en general los objetos, de gran tamaño deben organizarse como *arrays* de cara a su mapeado como memorias. Con objeto de realizar un aprovechamiento de los recursos de almacenamiento local disponibles existen técnicas de reestructuración de la memoria que tienen en cuenta su tamaño, su planificación en cuanto a accesos y su disponibilidad. En una estructura compleja (*array* de *struct*), mapear cada componente del *struct* en una memoria diferente tiene la ventaja de poder hacer accesos atómicos a cada componente sin necesidad de leer todo el elemento del *struct* (con el consiguiente ahorro de consumo de potencia). Sin embargo mapear cada componente en un bloque de memoria separada puede conllevar un consumo excesivo de recursos de memoria. La utilización de plantillas, si están soportadas, ayuda a la reutilización, parametrizando el código que implementa el array (tamaño, tipo de datos).

16) En muchos diseños es preciso definir el estado inicial del contenido de los *arrays*. Si bien esto en C/C++ o en SystemC puede parecer trivial, adaptar el proceso para que el sistema no consuma ciclos en la funcionalidad requerida implica definir diferentes modos de funcionamiento del diseño. Generalmente la solución propuesta consiste en añadir un modo de configuración al sistema que facilite su depurado al poder cargar un contenido definido y poder llevar el sistema a un estado conocido en cualquier momento. Si el diseño está basado en una FPGA, existen métodos conocidos para realizar este proceso durante el flujo de implementación.

17) Durante la realización de la síntesis de alto nivel, aunque la mayoría de los entornos indicados en este trabajo soportan la inclusión de *pragmas* en el código fuente, la experiencia nos indica que es preferible utilizar *scripts*, generalmente en lenguaje Tcl para orientar la síntesis, con objeto de facilitar la reutilización del diseño. Durante la síntesis, una de las restricciones más importantes es la definición del reloj. El planificador, usado conjuntamente con la herramienta de análisis temporal planificará las operaciones para cumplir dicho periodo de reloj. Si se imponen restricciones de latencia (nº de ciclos máximos), el planificador planificará dentro de las restricciones impuestas. La prioridad en la decisión vendrá determinada por la política prioritaria del entorno de síntesis.

18) Los bucles requieren una atención importante. Por una parte no es posible planificar bucles combinacionales por lo que es necesario romper el bucle. La técnica a utilizar depende del grado de paralelismo a utilizar en el bucle: desenrollado completo, parcial, con segmentación o mantenerlo enrollado. La decisión tiene efecto directo en las prestaciones del diseño.

19) Se debe aplicar las directivas de síntesis necesarias, sin sobre-restringir el diseño, dejando libertad al planificador para que tome decisiones guiadas en función de los parámetros de prestaciones establecidos.

20) En el estado actual de la SAN –como se ha dicho antes– es preciso realizar la simulación de la descripción RTL obtenida, comparándola con la descripción inicial. La reutilización del *testbech* es una faceta ya obligatoria en toda metodología de SAN. Existen diferentes aproximaciones al problema dependiendo del estilo de modelado utilizado.

21) Las decisiones tomadas en alto nivel se deben propagar hasta la implementación final. Es necesario tener en cuenta que la calidad de los resultados finales no depende únicamente de la herramienta de SAN usada sino también de la integración con el flujo de diseño de las herramientas de implementación y con las librerías de diseño.

Todas estas recomendaciones se resumen en la tabla 3.

Tabla 3. Recomendaciones de modelado

N	**Recomendaciones sobre estilo de modelado y de síntesis**
1	Usar tipos de datos ajustados a nivel de bits para reducir área/recursos
2	Usar datos abstractos para facilitar su encapsulado
3	Usar librerías optimizadas para operaciones en coma flotante
4	Usar `range()` para acceder a los bits
5	Definir valores por defecto para salidas
6	Estructurar el código en funciones
7	Las funciones añaden modularidad al diseño *hardware*
8	Evitar problemas de comunicación entre procesos
9	No utilizar modelado recursivo ya que no está soportado en *hardware*
10	Modelado apropiado del protocolo de comunicación a utilizar
11	Proteger zonas dedicadas a protocolos de la acción del planificador
12	Dividir expresiones complejas en varias líneas para su referencia
13	Centralizar parámetros, separar estructura y comportamiento y utilizar un único bloque por fichero facilita la gestión del proyecto
14	Uso del preprocesador para facilitar la portabilidad entre entornos
15	Organizar los objetos de gran tamaño como *arrays* para su implementación como memorias
16	Inicialización del contenido de los *arrays*. Diferentes opciones en función de la tecnología
17	Utilizar restricciones en Tcl para facilitar la reutilización de los modelos
18	Atención a la planificación de bucles
19	Aplicar las directivas de síntesis sin generar sobre-restricciones al diseño
20	Realizar la simulación a nivel RTL
21	Las decisiones tomadas en alto nivel se deben propagar hasta la implementación

2.7. Conclusiones

En este capítulo se ha realizado una revisión global de la SAN, de su evolución y la presentación de herramientas significativas durante su evolución. Igualmente se han descrito diferentes casos de uso que han permitido crear la experiencia necesaria para abordar este trabajo. Se ha puesto especial hincapié en la necesidad de utilizar lenguajes cercanos a la creación del algoritmo, ya sea en C/C++ o SystemC. Se ha hecho una descripción del estado actual de las herramientas disponibles y se ha esbozado dentro del alcance de este trabajo sus principales características. Para terminar se han resumido, a modo de criterios generales y breves recetas prácticas, algunos de los aspectos que debe tener en cuenta el diseñador para la utilización de metodologías de diseño basadas en SAN con el mayor aprovechamiento posible. Constituyen unos modos de hacer o mejores prácticas en el diseño.

Como **principales conclusiones** de este capítulo podemos resaltar las siguientes:

1) La síntesis de alto nivel está siendo adoptada por la industria con el objetivo de mitigar la crisis de diseño y poder hacer frente a la demanda de aceleración de problemas de gran complejidad y magnitud en los datos. Igualmente está siendo adoptada como metodología de diseño en aquellos casos que requieren una rápida puesta en el mercado del producto y en aquellos casos donde los estándares cambian rápidamente.

2) Se ha producido una evolución y madurez de la SAN basadas, entre otros, en varios aspectos fundamentales: la adopción de un lenguaje apropiado al nivel del problema tratado, la facilidad de verificación en alto nivel y una mejora continua en la calidad de los resultados. Por ello, en SAN es muy importante dar soporte a la "síntesis para la verificación y calidad del diseño".

3) Desde el punto de vista de alto nivel hay un interés creciente en identificar de forma temprana problemas asociados a la calidad del código C/C++/SystemC para identificar problemas que afectan a los parámetros PPA (Prestaciones, Potencia, Área) del diseño final. Entre estos aspectos podemos citar la detección de código muerto, errores por lectura de memoria no inicializadas, casos no cubiertos en múltiples ramas de decisión, errores de desbordamiento y de división por cero entre otros.

4) A lo largo del capítulo se ha visto la importancia de mantener el nivel RTL como paso intermedio para la verificación y la síntesis de tal forma que no se puede prescindir de este nivel por dos razones principales: a) para verificar el diseño, ya sea mediante simulación, cosimulación o incluso mediante búsqueda de equivalencias con el código fuente (SLEC – *Sequential Logic Equivalence Checking*), y b) para conectar con flujos de diseño existentes en la industria con alto grado de calidad en los resultados de prestaciones, potencia y área (PPA). Diferentes enfoques para compilar desde el nivel algorítmico, o incluso desde el nivel TLM de transacciones hasta un *netlist* han resultado insuficientes. Solo en casos muy específicos un compilador de silicio desde el nivel algorítmico funciona bien (compilar una ALU, un filtro, un *array*...). Una de las mayores preocupaciones de los diseñadores que adoptan metodologías SAN aparece en la verificación tanto del modelo de alto nivel como de las discrepancias de comportamiento del modelo original y del modelo RTL.

5) Conviene introducir un nivel intermedio entre el algorítmico y el RTL. Este enfoque lo hemos llamado “Doble X”. Es muy útil el nivel de transacciones (TLM2.0 y otros lenguajes equivalentes). La razón es que la visión explícita de las transacciones es la que introduce por primera vez una visión de nivel arquitectural, donde la computación (o funciones algorítmicas), las estructuras de datos, y la comunicación, que componen toda arquitectura, aparecen definidas y separadas. Eso no lo hace el nivel algorítmico, ni mucho menos los modelos de computación. Ese nivel está muy bien soportado hoy por SystemC. Como consecuencia de visualizar y establecer estos dos niveles intermedios, es necesario proporcionar mecanismos que controlen la coherencia de los datos, de las descripciones y las adaptaciones de los resultados de una etapa de síntesis hacia la siguiente. Adaptar (y estandarizar) las salidas de una herramienta del flujo de diseño a la siguiente. Este aspecto de ingeniería del flujo de herramientas y datos en SAN es central en la experiencia de diseño, y en esta tesis. Ejemplos de este flujo se muestran en los casos de estudio y se han comentado en el apartado anterior, como es el caso de hacer uso del preprocesador de C para compatibilizar diferentes vistas del diseño y la utilización de flujos controlados con *Makefiles* para garantizar la actualización de los datos. Igualmente la utilización sistemática de scripts garantiza la repetitividad de los resultados obtenidos.

6) La adaptación de datos en la ingeniería del flujo de síntesis, incluye el proceso de controlabilidad de la verificación a distintos niveles de la síntesis, incluida la verificación funcional y arquitectural temprana. Es cuantificable el ahorro de esfuerzo de diseño y la mejora en la calidad del diseño si la verificación se hace mediante verificación por etapas. Igualmente la reutilización del *testbench* desarrollado con altos niveles de abstracción genera ahorros significativos en los tiempos de diseño.

7) Como se ha indicado, es cuantificable que la estructuración al nivel TLM/SystemC, es decir al primer nivel arquitectural, incide fuertemente en la optimización del diseño resultante, con efectos significativos en los parámetros de prestaciones PPA.

8) La síntesis de los mecanismos de generación de transferencias, la síntesis de protocolos, es determinante en el éxito de la adopción de técnicas de SAN. En parte debido al desacoplo computación-datos-comunicación (interconexión). En parte debido a la potencia de las infraestructuras de interconexión que han ido emergiendo, al ritmo del aumento del número de niveles de metal en los circuitos, y de la densidad de transistores para incluir la gestión de los buses y el control de protocolos.

9) Es importante y práctico encapsular la síntesis SAN en diseños IP, es decir completos y reutilizables, *standalone*. Es debido a que según el enfoque *platform based design* -de éxito industrial- la integración de estos bloques IP en plataformas permite ahorrar tiempo y esfuerzo en la etapa de transformación del diseño desde la plataforma intermedia al target final. Por eso es necesario seguir avanzando en técnicas para flexibilizar el diseño, el IP, para que sea portable para diferentes targets.

10) Se propone una visión de “Doble X” vertical (figura 33). En la primera se mapean tareas y funciones a estructuras TLM o de transacciones definidas por la arquitectura. En la

segunda se mapean estas arquitecturas a la clase de plataforma intermedia elegida. La salida de esta segunda X es la implementación del algoritmo ya estructurado y sintetizado en los IPs componentes de la arquitectura de la plataforma, en el target final.

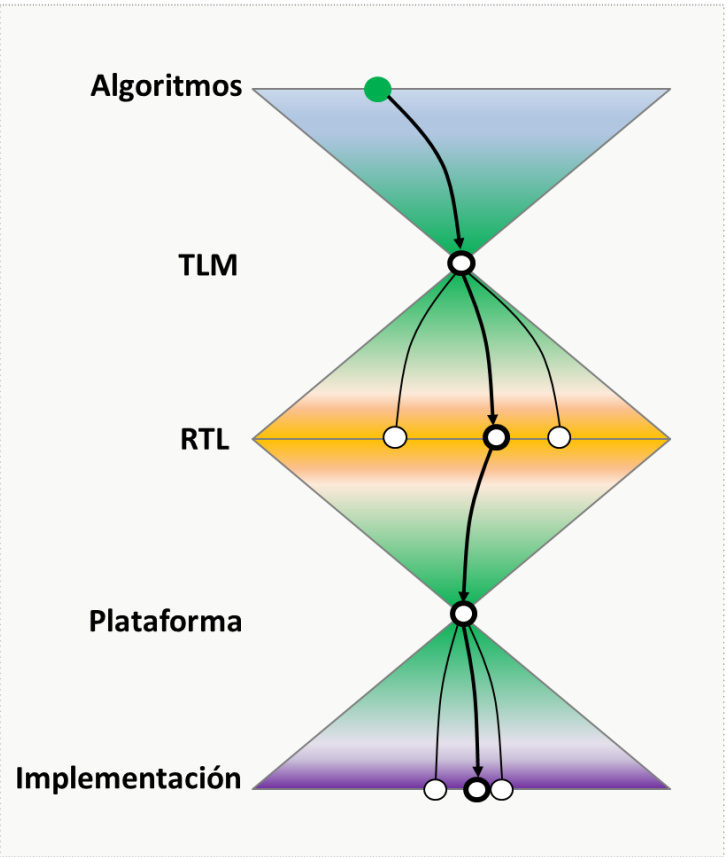


Figura 33. Flujo de diseño en "Doble X"

Un comentario final. Es útil la exploración del espacio de diseño DSE y el estudio de los diversos *algorithm to architecture mapping*, (transformación, correspondencia). Pero ese estudio, al igual que el *System Level ESL*, exceden la temática de esta tesis sobre SAN, que se refiere siempre a descender en la abstracción desde una funcionalidad definida a nivel tarea hasta niveles de implementación inferiores, más detallados, al menos RTL sintetizable, procesable por las herramientas de síntesis lógica. En *ESL y System Synthesis* debe tratarse apropiadamente la descomposición en tareas, la planificación de las tareas resultantes y su concurrencia y sincronización, la descomposición HW/SW, o la extracción de paralelismo que habilite un mapeado a MPSoC o múltiples núcleos en SW. Pero desde el punto de vista de la SAN para implementación, es importante no desenfocarse en su utilidad, en su objetivo: es mejor no centrarse en las búsqueda de maneras de "extraer" el paralelismo que pueda existir en una descripción secuencial, y centrarse en cambio en facilitar -para el arte del diseño- que los diseñadores puedan expresar fácilmente y con poco costo la concurrencia existente. Y a partir de ahí poder construir implementaciones eficientes -más eficientes- de aplicaciones útiles, significativas.

Capítulo 3. Caso de aplicación: diseño de un procesador de eventos

3.1. Introducción

Recientemente se ha producido un avance importante en las aplicaciones empotradas al poder incorporar aceleradores para aquellas tareas que tienen exigentes requisitos de funcionamiento en tiempo real. Algunos ejemplos de tales aplicaciones son la detección de fraudes para tarjetas de crédito, la detección de intrusión en redes de ordenadores, las transacciones en los mercados financieros o los sistemas de supervisión de la salud de los pacientes. En ellas se requiere el procesamiento de un gran volumen de datos a lo largo de intervalos temporales determinados, o de series temporales, para extraer información significativa. Esta extracción se puede caracterizar por estar basada en eventos complejos que incluyen además una gran cantidad de información a procesar. Se entiende que el procesamiento de eventos complejos es un paradigma de cómputo que partiendo de una secuencia de eventos en tiempo real, extrae información significativa basada en algoritmos o reglas definidos por el usuario y la transforma en nuevos datos que permitan manejar dichos eventos o tomar decisiones sobre ellos, y en otros datos, asociados, de forma integrada.

En muchos casos el tiempo de respuesta es un factor crítico de cara a la toma de decisiones. Por ejemplo, en aplicaciones del mercado electrónico de valores se especifican tiempos de respuesta del orden de 1 ms, tiempo en que el operador deberá tomar posición para obtener ventaja competitiva. En estos casos una solución completamente *software* pudiera no asegurar las cortas latencias que se necesitan, ni siquiera cuando explota al máximo el paralelismo de la aplicación. Por ello se puede considerar este tipo de sistemas como candidatos a su implementación con aceleradores *hardware*. La viabilidad científica e industrial de este proceso en aplicaciones muy variadas depende mucho de la creación y utilización apropiada de metodologías de diseño que faciliten la implementación de los algoritmos y reglas desde el código C/C++, desde el alto nivel.

3.1.1. Sistema de eventos complejos

El procesamiento de eventos complejos es un paradigma de computación que permite monitorizar los eventos que se reciben de forma continua en un escenario de tiempo real (*soft or hard real time*) y reaccionar ante ellos. Los aceleradores de procesamiento de eventos se utilizan en diferentes escenarios críticos tales como detección de fraudes, monitorización de tráfico, mercados de valores, gestión de redes o unidades de cuidados intensivos. En todos estos casos se necesita tener disponible capacidad de cómputo para cumplir con las prestaciones, entre las que frecuentemente se encuentran una alta capacidad de procesamiento y una baja latencia [121].

Los sistemas de procesamiento de eventos complejos se diseñan para procesar flujos de eventos recibidos en tiempo real. Soportan reglas de tipo reactivo que se disparan cuando se encuentran combinaciones de patrones específicos. Para ello se obtienen eventos desde el flujo de datos de entrada (por ejemplo, mediante un análisis sintáctico) obteniéndose una secuencia de eventos que se analiza de acuerdo a patrones de eventos complejos. Este es el caso, por ejemplo, de un *broker* que querría estar informado al instante de cuando un *stock* presenta movimientos al alza y a la baja (eventos derivados) [80]. Por ello se emplea el término de "mercado del microsegundo". En la edad del mercado de valores tecnológico, la latencia en la recepción y el procesamiento de los eventos del mercado es un aspecto crítico que determina una posición estratégica. El aumento de actividad del mercado de valores tecnológico es notable y está basado no sólo en las mejoras introducidas en las redes de datos, sino también en las soluciones algorítmicas para el procesamiento de eventos (78% de incremento entre 1995 y 2009 [122]).

3.2. Sistemas en Tiempo Real (STR)

Se ha indicado anteriormente que un procesador de eventos puede ser considerado como un sistema en tiempo real. Y se puede definir un sistema en tiempo real (STR) como aquel sistema digital que interactúa activamente con su entorno con una dinámica conocida entre sus entradas, salidas y restricciones temporales, para establecer un correcto funcionamiento de acuerdo con los conceptos de predictibilidad, estabilidad y controlabilidad [123].

Se puede hablar de dos tipos básicos de STR: *Hard* RT y *Soft* RT. Para el primero de los casos la restricción es completa, con tiempos de respuesta estrictos. Para el segundo, los tiempos de procesamiento pueden estar comprendidos en un determinado margen pero no son críticos cuando la salida sigue una determinada cadencia, aunque presente cierta latencia inicial.

Un ejemplo típico de sistema en tiempo real tipo *Hard RT* es el sistema antibloqueo de las ruedas de un automóvil (ABS). En este caso el tiempo límite en el que debe operar es aquel en el que las ruedas deban liberarse antes de que se bloqueen. Si el sistema libera las ruedas una vez que ya se han bloqueado, habrá fallado [5, 6]. Un ejemplo de sistema *Soft* RT puede ser un sistema de codificación, transmisión de vídeo y decodificación de vídeo [126].

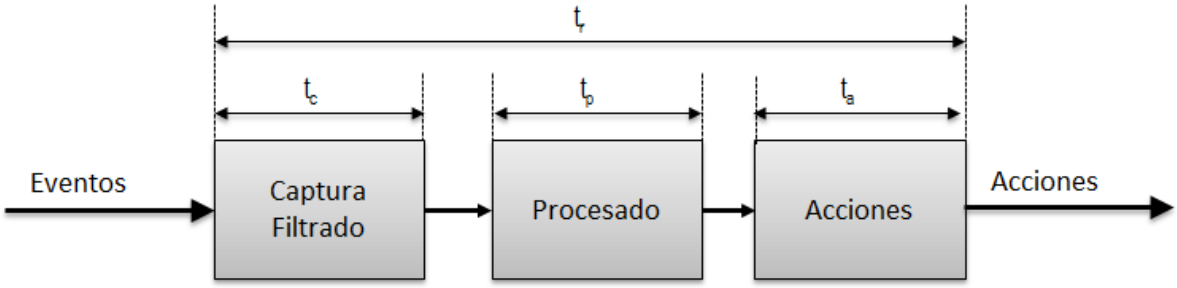


Figura 34. Esquema básico de un procesador de eventos complejos

Un sistema de procesado de eventos es un tipo de sistema en tiempo real. Existe una tasa de llegada de eventos que deben ser procesados en un tiempo, determinado por el periodo de muestreo de eventos. En general existirán unas condiciones y reglas que determinan la reacción del sistema ante determinados eventos. El procesado consiste en comprobar si cada evento de entrada cumple o no algunas de las condiciones impuestas con anterioridad y decidir, en base a ello, la acción a realizar. Un ejemplo son los sistemas automatizados de compra y venta en el mercado de valores. Estos sistemas deben decidir realizar una acción de compra o venta en función de los cambios y tendencia en los precios de los productos y de unas estrategias previamente establecidas [127].

El funcionamiento de los sistemas de procesado de eventos se puede dividir en las siguientes tareas: captura y filtrado de eventos, procesado de los eventos y generación de las acciones (figura 34). Durante la captura y filtrado, se requerirá que el sistema obtenga los eventos temporales que se producen en su entorno y filtre aquellos parámetros que sean de utilidad para su posterior procesado, descartando aquella información sobrante que pueda sobrecargar innecesariamente al resto de tareas. Durante el procesamiento se realizarán las operaciones definidas mediante reglas de procesamiento con el fin de comprobar si se cumplen una o varias condiciones previamente establecidas. En la fase final, la generación de acciones, se activan las acciones a realizar en función de las reglas activadas. Para poder procesar los eventos entrantes en el sistema debe cumplirse la siguiente restricción temporal:

$$t_c + t_p + t_a \leq t_r \quad (1)$$

donde t_c es el tiempo de captura, t_p es el tiempo de procesado, t_a es el tiempo requerido para la generación de acciones y t_r es la restricción impuesta de tiempo real.

3.3. Alternativas de implementación

Los sistemas de procesado de eventos se pueden ejecutar sobre diferentes plataformas arquitecturales con objeto de cumplir con la restricción temporal indicada anteriormente y con la flexibilidad requerida en cuanto a costes del sistema.

Las diferentes soluciones encontradas en la literatura científica van desde el uso de circuitos integrados de aplicación específica (ASIC) hasta el uso de microprocesadores de propósito general. La solución más flexible es ejecutar el STR sobre un procesador de propósito general (*General Purpose Processor* – GPP).

Un GPP está preparado para poder realizar cualquier tarea, mediante la ejecución de un determinado programa, lo que representa una solución muy adaptable a diferentes tipos de procesamiento. Sin embargo tiene la desventaja de ser poco eficiente para determinadas operaciones, especialmente las orientadas a tratar bits de forma individual, e igualmente su consumo de potencia puede no ser aceptable en determinados escenarios. Esta solución GPP se centra en el dominio *software*, donde las aplicaciones se pueden compilar, modificar y recompilar, permitiendo así optimizar el diseño tras varias iteraciones. La concurrencia explotable tiene las limitaciones propias del algoritmo y de la dependencia de datos, pero también varias limitaciones adicionales derivadas del compilador. Por contra, una ventaja es que la programación de las aplicaciones se puede hacer en lenguajes de alto nivel como Java, C++, etc.

En el lado opuesto se ubican soluciones a medida basadas en circuitos integrados de aplicación específica (ASIC). Al contrario que los GPP, se trata de soluciones circuitales diseñadas de forma optimizada para una aplicación concreta, restringiendo su programabilidad pero incrementando su eficiencia. Otra característica fundamental de esta alternativa es la posibilidad de concurrencia real entre tareas, al incluir en el circuito integrado diferentes módulos que realizan el procesamiento paralelo de los flujos de datos capturados.

En un ASIC, el flujo de diseño parte de una descripción, normalmente realizada en lenguajes de descripción *hardware* (HDL), tales como Verilog o VHDL, tradicionalmente a nivel RTL. Los bloques descritos en estos lenguajes se sintetizan posteriormente haciendo uso de las librerías tecnológicas de los fabricantes. Como se ha comentado ampliamente en el capítulo anterior, en la actualidad se ha incrementado el nivel de abstracción pasando a usar lenguajes de descripción de sistemas electrónicos (ESL), generalmente basados en C/C++ o SystemC [83], que tras una síntesis de alto nivel (HLS), transforma la descripción algorítmica en una microarquitectura a nivel RTL. La microarquitectura resultante se optimiza e implementa siguiendo flujos de síntesis de alto nivel [9, 10, 11].

El ASIC requiere un proceso de verificación complejo para garantizar su funcionamiento desde el primer diseño. Por ello las FPGAs facilitan las tareas iniciales de prototipado, y pueden también ser consideradas como alternativas de implementación frente a los GPP y los ASIC. Las FPGAs poseen recursos de memoria internos, bloques funcionales, en algunos casos incluso uno o varios núcleos procesadores, y otros bloques funcionales de interfaz que facilitan la implementación de un sistema electrónico en chip (SoC) [128].

Las principales ventajas del uso de FPGAs frente a otras soluciones de diseño electrónico son el bajo coste durante las fases de desarrollo del sistema, la producción de un reducido número de unidades (frente al coste de fabricar un ASIC) y su sencilla reprogramabilidad. Esta última propiedad permite una optimización post-diseño. Es aquí donde esta tecnología tiene una ventaja competitiva para este tipo de problemas de procesado de eventos.

Los kits de diseño que proporcionan los fabricantes incluyen placas sobre las que vienen interconectadas las FPGA con diversos recursos, entre ellos bloques de interfaz como pueden ser USB, Ethernet, etc. Además, también incluyen sus propias fuentes de reloj, bloques de memoria e incluso, como se ha dicho, CPUs.

Otra posible implementación es el uso de GPUs (*Graphics Processing Unit*). Los entornos de desarrollo proporcionados por los fabricantes, hacen que la GPU -originalmente creada para explotar el paralelismo existente en el procesamiento de gráficos- sea cada vez más flexible, permitiendo su uso para la ejecución de aplicaciones genéricas (GPGPU – *General Purpose computing on GPU*). Aunque su programación no es tan flexible como para el caso de las CPUs de propósito general, su gran cantidad de elementos de cómputo, junto a su arquitectura optimizada para la concurrencia mediante paradigmas SIMD (*Single Instruction Multiple Data*) y SPMD (*Single Program Multiple Data*), puede ser de utilidad en el procesamiento de eventos, cuando se requiera realizar la comparación de una gran cantidad de ellos sobre una misma condición.

Por último podemos citar los DSPs (*Digital Signal Processor*), que son procesadores optimizados para el procesado de señales, siendo normalmente especializados en operaciones del tipo MAC (Multiplicación/Acumulación). Este tipo de procesadores son más eficientes que las CPUs o las GPUs, pero mucho menos flexibles en su programación.

En la figura 35 se comparan las diferentes soluciones indicadas, teniendo en cuenta su eficiencia frente a su flexibilidad. Se aumenta la eficiencia del sistema moviendo funciones del dominio *software* –por tanto ejecutadas en un GPP, más flexibles– a su implementación en el dominio *hardware*, más eficientes pero menos flexibles. En este caso, con objeto de mantener una cierta flexibilidad en el desarrollo de estas funciones, se emplean técnicas de diseño basadas en la síntesis de alto nivel.

3.4. Arquitectura de los procesadores de eventos

Se define un procesador de eventos, en su versión más simple, como un sistema que ejecuta una acción en el caso de que se presenten determinadas condiciones en un conjunto de eventos de entrada. La arquitectura del procesador puede dividirse en cuatro grandes bloques: interfaz de entrada, núcleo de procesamiento, ejecutor de resultados e interfaz de salida (figura 36).

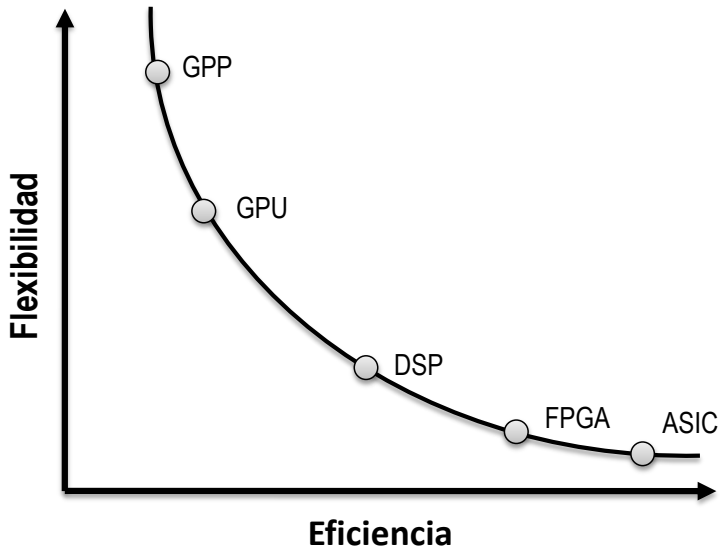


Figura 35. Comparativa de flexibilidad y eficiencia de las arquitecturas

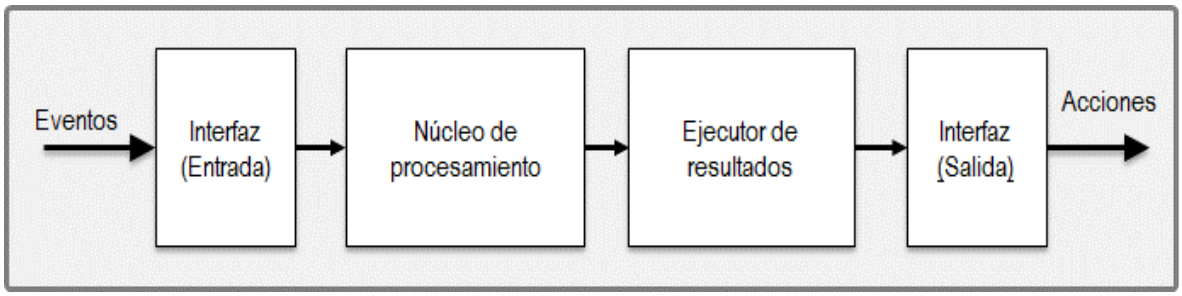


Figura 36. Arquitectura simple de un procesador de eventos

La interfaz de entrada recoge los eventos del medio de transmisión del que provengan, y los empaqueta de forma óptima para el núcleo de procesamiento. Este será el encargado de realizar las comprobaciones de cumplimiento de las reglas establecidas previamente. El ejecutor de resultados es un bloque esclavo del núcleo de procesamiento que realizará una acción predefinida, en caso de que el núcleo de procesamiento encuentre una coincidencia de su patrón de búsqueda. En caso de que el procesador de eventos esté comprobando más de una condición, y que cada una lleve asociada una acción diferente, existirá un código de operación que será comunicado por el núcleo al ejecutor de resultados. Por último, la interfaz de salida enviará el código de la correspondiente acción al sistema de actuación para completar la ejecución de la acción correspondiente.

En otras aplicaciones es necesario contemplar dependencias entre tramas, ya sea en base a fuentes de eventos diferentes, a códigos internos a cada evento, o simplemente a nuevas muestras temporales. Ejemplos de estas necesidades son:

- Condición de actuación asociada a la diferencia entre dos fuentes de eventos, como que dos sensores indiquen valores diferentes, y cada uno genere su propia trama.

- Actuación condicionada a las relaciones entre secuencias de eventos diferentes asociadas a productos o fuentes diferentes. Por ejemplo en un mercado de valores, si la condición de compra es una relación de precios entre un producto y otro.
- Cuando la condición de actuación se corresponde a una tendencia que requiere una memoria de valores históricos o serie temporal de tramas de eventos previos.

Este tipo de requisitos implica la necesidad de añadir soporte al almacenamiento de la información que contenga el estado actual del sistema y en el que se encuentre la información requerida sobre tramas pasadas y su evolución, si fuese necesario (figura 37).

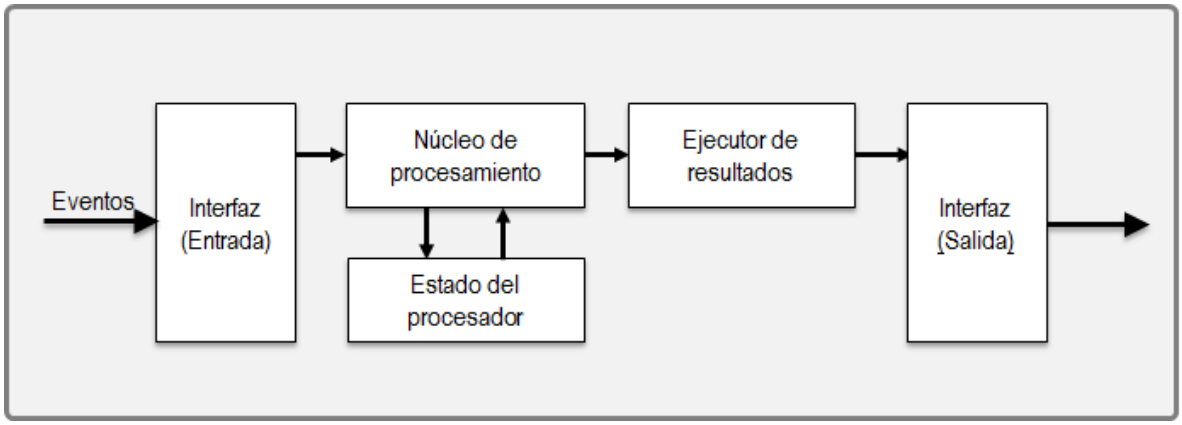


Figura 37. Arquitectura de procesador de eventos con memoria de estados

Para ambas arquitecturas, en el diseño del procesador de eventos es preciso incluir intrínsecamente las condiciones de actuación del procesador de eventos. Sin embargo, para aumentar la flexibilidad de su diseño, es común requerir que las condiciones de chequeo de eventos, así como las acciones asociadas a ellas, puedan modificarse en tiempo de ejecución.

Para ello, hay que modificar tanto el núcleo de procesamiento como el ejecutor de resultados, de tal manera que las condiciones y sus acciones, se puedan leer de las variables de estado. Ello conlleva el rediseño de la interfaz de entrada, para que sea capaz de discernir entre eventos y reglas de procesado, de tal manera que se envían unas al núcleo de procesamiento y las otras a las variables de configuración (figura 38).

Una propuesta de arquitectura más completa es la mostrada en la figura 39. En este caso se permite que las acciones del procesador de eventos modifiquen el estado del procesador. Es de aplicación para aquellos casos en que, si se cumplen las condiciones de procesamiento definidas, puede resultar de utilidad crear nuevas condiciones en el procesador para realizar estrategias de procesado más complejas. Es preciso que el bloque ejecutor de resultados pueda modificar el estado del procesador, generando nuevas condiciones y sus respectivas acciones.

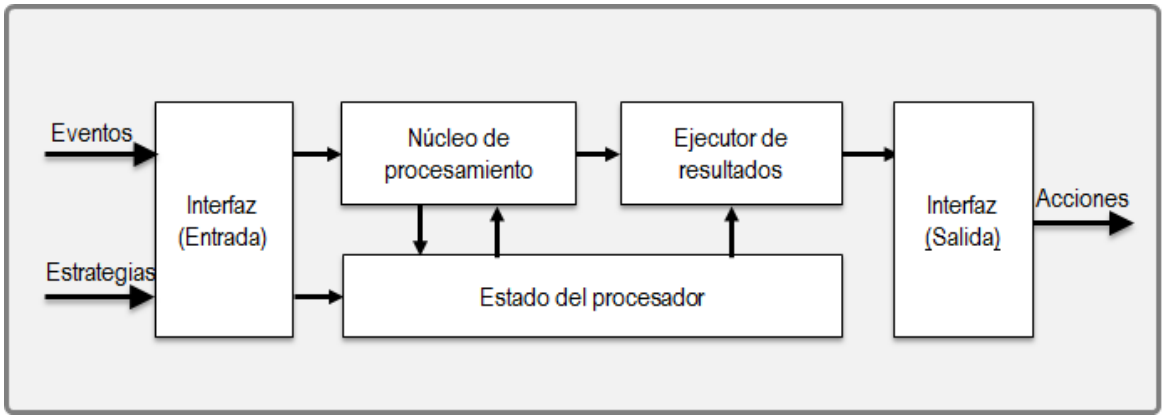


Figura 38. Procesador de eventos con estrategias modificables en tiempo de ejecución

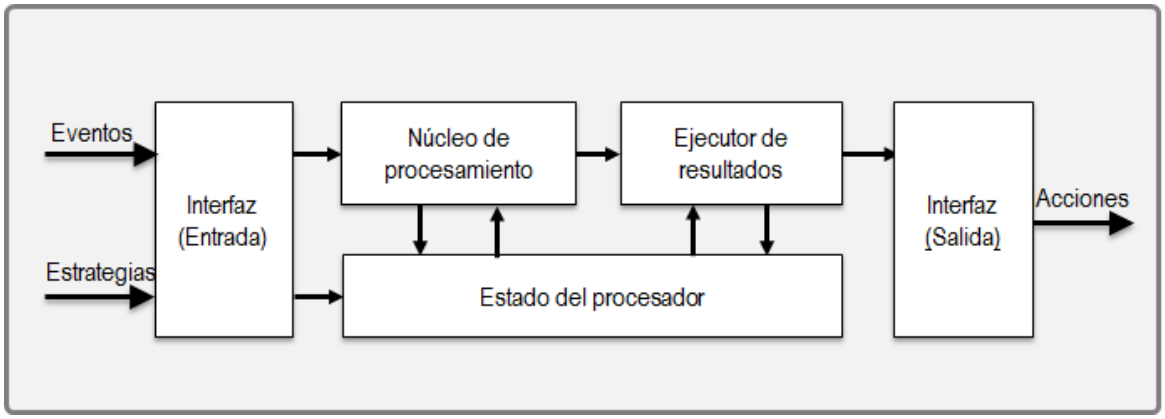


Figura 39. Arquitectura de un procesador de eventos con autoconfiguración

Se puede concluir que cada procesador de eventos puede tener unos u otros requerimientos que complementen en mayor o menor medida las arquitecturas descritas anteriormente.

3.5. Aplicación de referencia

Con el continuo avance de las Tecnologías de la Información y las Comunicaciones (TIC), el mercado financiero ha aplicado tecnologías TIC en diversos ámbitos de su actividad.

En los mercados financieros, cada orden realizada por un agente de bolsa, provoca cambios en los títulos que se intercambian en diferentes bolsas del mundo. Esta dependencia entre títulos, y los continuos cambios en sus precios, es aprovechada por los inversores para conseguir rentabilidad en sus inversiones a través de plusvalías.

Según NASDAQ [129], *Algorithmic Trading*[1] o simplemente *Algo Trading* (AT) hace referencia a la compraventa de acciones en el mercado bursátil mediante el uso de computadores utilizando algoritmos propietarios. Existen dos tipos de AT. Por una parte, *Algo Execution Trading* (AET) cuando una orden de compra/venta (normalmente una orden compleja) se ejecuta mediante AT. El programa para AT se diseña para obtener el mejor precio

[1]En la literatura también aparece referenciado como *High-Frequency Trading*

posible. Para ello se puede dividir la orden en partes y ejecutarlas en tiempos diferentes. El segundo tipo de AT consiste en buscar la mejor oportunidad de ejecución en el mercado.

Por su parte, The Economic Times [130] define AT como "un sistema de comercio que facilita la realización de decisiones sobre las transacciones en los mercados financieros usando herramientas matemáticas avanzadas".

En este tipo de sistemas se minimiza la intervención humana en la toma de decisiones, y por lo tanto son decisiones muy rápidas. Esto permite tomar ventaja de cualquier oportunidad de beneficio que aparezca en el mercado más rápidamente que lo haría un experto agente de bolsa.

Una característica distintiva de AT es la sofisticación de su sistema de análisis y toma de decisiones [131]. Estos sistemas están desplegados en mercados continuos y con alta volatilidad que exigen una gran dinámica a la hora de generar beneficios.

Los inversores generan la rentabilidad de su capital invertido a través de dos factores: dividendos y plusvalías. La rentabilidad por dividendo es a largo plazo y suele estar además vinculada al crecimiento de la cotización del valor. Las técnicas de AT sin embargo suelen ir asociadas a alcanzar a muy corto plazo una rentabilidad del capital invertido mediante la plusvalía generada entre la compra y venta de valores. En muchos casos, y gracias a los sistemas automatizados que se presentan, pueden generarse múltiples operaciones con valores de plusvalía muy pequeños, pero que repetidas muchas veces en cortos periodos de tiempo, generan beneficios para el inversor. Esto requiere en la práctica que las órdenes de compra y venta se realicen con latencias por debajo del milisegundo [132].

En las técnicas de mejora de este tipo de transacciones existen tres importantes líneas de actuación: definir las mejores estrategias de adquisición y venta de títulos, la creación de sistemas electrónicos capaces de generar las órdenes de mercado lo más rápidamente posible, adelantándose así a otros posibles interesados en realizar otras acciones que puedan afectar al mercado en el que se trabaje, y la de posicionar dichos sistemas lo más cerca posible (en lo que a dispositivos de red se refiere) del nodo del mercado en cuestión dentro de la red de comercio electrónico específica del mercado.

En este capítulo se usará como modelo de referencia una aplicación de AT dedicada a la generación de órdenes de mercado en base a estrategias definidas previamente por el inversor. No es objeto del mismo el desarrollo de esas estrategias. En particular, la aplicación de referencia utilizada, es propietaria y en este trabajo será denominada RT-CORE [133].

El procesador de eventos objeto de este desarrollo incluye las siguientes características:

- Permite crear estrategias compuestas por órdenes definidas por una acción a realizar y un conjunto dinámico de condiciones de disparo.
- Permite definir acciones de activación de nuevas órdenes y de estrategias previamente almacenadas en el sistema.
- Pueden definirse condiciones temporales de comienzo, finalización y duración de estrategias.

- Pueden definirse comparaciones entre un gran número de atributos de cada título, comparando diferentes atributos de diferentes productos. Además, las comparaciones pueden ser complejas, no solo de igualdad o de mayor o menor, sino también de valores calculados a partir de atributos, del estilo $K_1 \times Atributo + K_2$.

3.6. Requerimientos del sistema

A partir del estudio de la aplicación *software* de RT-CORE se deducen un conjunto de requerimientos que deben estar presentes en la arquitectura final del sistema.

- **Interfaz de entrada de eventos**. Los eventos se reciben a través de una interfaz TCP/IP, empaquetados ya sea en formatos estándares o propietarios. La tasa de tráfico a tratar es del orden de un millón de eventos complejos por segundo. Ello plantea la necesidad de disponer interfaces de red multiGigabit Ethernet. Para gestionar estas interfaces, si bien existen implementaciones *hardware* de la pila TCP/IP, por razones de flexibilidad se opta por una implementación *software*, por lo que se requiere la correspondiente infraestructura *hardware*/*software*.

- **Preparación/ordenación/procesamiento de eventos y reglas**. El procesamiento de los eventos de entrada requiere una fase de preparación y almacenamiento local en un formato que facilite su acceso por las unidades funcionales.

  Generalmente requiere estructuras especializadas por lo que será necesario crear unidades específicas para su tratamiento. Los requisitos impuestos por estas unidades dependen de la estrategia a utilizar, pero tienen en común la necesidad de almacenamiento local y de unidades lógicas (comparadores, etc.) rápidas. En el caso de usar estructuras complejas de memoria será necesario disponer de unidades que calculen las direcciones usando registros base y unidades aritméticas.

- **Almacenamiento local de los datos de entrada**. Normalmente los datos que provienen de la interfaz Ethernet se envía a colas FIFO para su posterior procesamiento. Estas colas temporales se volcarán a estructuras almacenadas en memorias mediante el uso de mecanismos de acceso directo a memoria (DMA).

- **Reconfigurabilidad**. Es posible añadir reconfigurabilidad al sistema ya sea incluyendo mecanismos de reconfigurabilidad *software* (programabilidad) mediante la utilización de uno o varios microprocesadores empotrados que ejecutan una aplicación *software*, y reconfigurabilidad *hardware* utilizando dispositivos programables tipo FPGA. El dispositivo FPGA deberá proporcionar mecanismos que soporten su reconfiguración total o parcial a través de puertos dedicados.

- **Jerarquía de memoria**. La necesidad de procesamiento de datos generalmente se organiza de forma jerarquizada, facilitando la localidad de los datos. Es necesario organizar la memoria en diferentes niveles ya sea en el propio dispositivo (BRAM), o disponer de un controlador de memoria externa con soporte de utilización de memorias de tipos SRAM o DDR de alta velocidad.

- **Otros recursos**. Como se deduce de los requisitos anteriores, la posibilidad de disponer de un procesador empotrado facilita la gestión de la pila TCP/IP y de otros parámetros de configuración del sistema final, aportando la flexibilidad *software* deseable.

3.7. Soluciones arquitecturales

Una vez definidos los requerimientos del sistema, se deben estudiar las diferentes posibilidades de implementación del RT-CORE. Existen diferentes alternativas de implementación, considerando aspectos de flexibilidad, capacidad de incorporación de reglas al sistema o capacidad de procesamiento.

3.7.1. Arquitectura basada en núcleos de tipo *soft-processor* integrados

Esta solución se basa en la creación de un sistema compuesto de N núcleos procesadores (dependiendo de la capacidad de la FPGA) que incrementa el paralelismo de procesamiento del sistema.

Se trata de procesadores de tipo *soft-core* que conforman el sistema multiprocesador. Los núcleos realizan el procesado de eventos de forma flexible, aprovechando la potencia de cómputo del núcleo. Esta solución se denomina de tipo multiprocesador homogéneo y es la extensión (masiva) de la utilización de una CPU multinúcleo.

La ventaja que ofrece esta solución es la facilidad de reprogramación y la adaptación del núcleo procesador, tamaños de caché y otros recursos a las características de la aplicación. Ejemplos de este tipo de núcleos de procesadores son MicroBlaze de Xilinx o Altera NIOSII.

Hay otras soluciones o variantes disponibles que usan núcleos procesadores *hard IP*, como es el caso de procesadores PowerPC o ARM, que pueden incluir arquitecturas superescalares de varios cauces de ejecución paralela (paralelismo ILP).

La figura 40 representa el diagrama de bloques de una arquitectura basada en núcleos de tipo *soft-processor* integrados. Las prestaciones de este tipo de soluciones están condicionadas por la arquitectura de interconexión de los procesadores y la arquitectura del subsistema de memoria diseñado. En general tienden a presentar elevados consumos de energía.

3.7.2. Arquitectura de implementación *hardware*

Una segunda alternativa, ubicada en el otro extremo del espacio de soluciones, es implementar el sistema en *hardware* específico. En este sentido, la implementación del procesamiento de las reglas se ejecuta en Elementos de Procesamiento Reconfigurables (EPR) implementados directamente como funciones lógicas. Esta solución presenta una ganancia de un orden de magnitud en las prestaciones del sistema (eventos/s) sobre la solución basada en PC.

A diferencia del caso anterior, el esfuerzo de diseño se centra en la síntesis de los procesadores de eventos, su interconexión y su integración con los módulos de entrada y salida de eventos.

En la figura 41 se representa la arquitectura general de esta solución. Como se puede observar, el sistema consta de una interfaz de red Gigabit Ethernet, conectada al microprocesador externo, que envía las tramas de eventos a los bloques de procesamiento (que incluyen la interfaz, el núcleo de procesamiento, el ejecutor de resultados y las memorias que almacenan el estado del procesador). Existe una interfaz con memoria de alta velocidad DDR usada por la CPU.

La principal característica de esta arquitectura es que las estrategias de comparación son las que definen el esquema lógico de los procesadores, consiguiendo una mayor tasa de procesado, a costa de la flexibilidad, ya que no se modifican las estrategias en tiempo de ejecución.

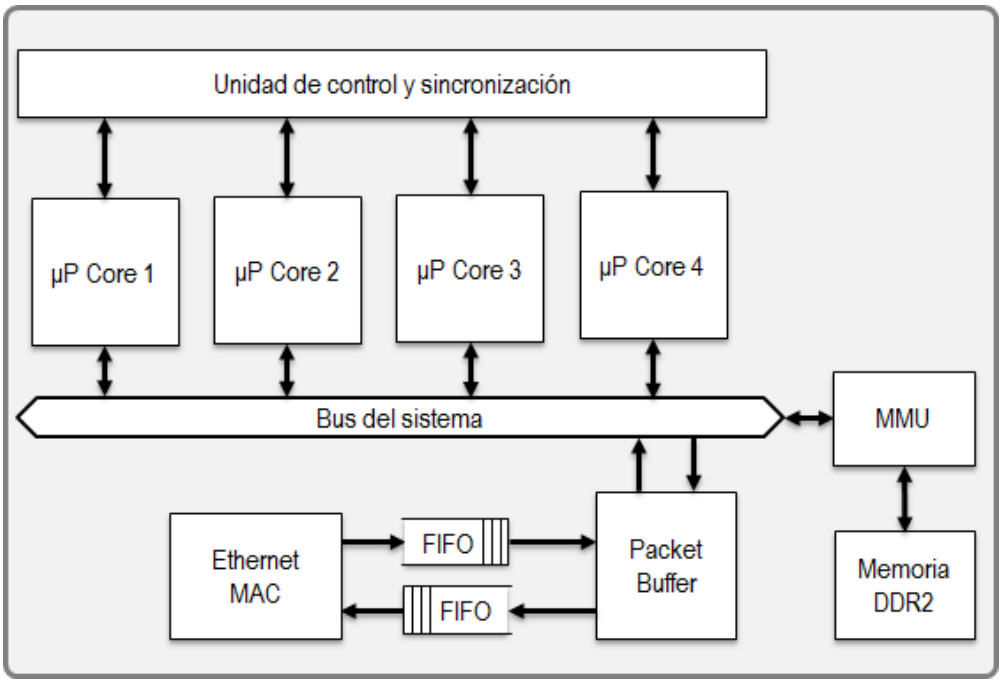


Figura 40. Arquitectura basada en *multi-core soft-processor*

3.7.3. Arquitectura de implementación híbrida

Esta solución plantea un punto intermedio en el espacio de soluciones, entre las dos descritas anteriormente. La principal característica de esta implementación es que dispone de núcleo CPU interno en tareas de planificación y supervisión y comunicación, y elementos *hardware* de proceso, en forma de aceleradores, cuyas estrategias de procesamiento y reglas se almacenan en memoria interna del dispositivo, en memoria de configuración de cada elemento de proceso.

Mantener la configuración en memoria, aunque los accesos a memoria ralentizan el procesado frente a la solución *hardware*, permite una mayor flexibilidad, ya que facilita la

modificación de las estrategias mediante la alteración de los parámetros y datos almacenados en dichas memorias. Además de esto, pueden reutilizarse los núcleos de procesamiento para varias estrategias, permitiendo así una mayor capacidad.

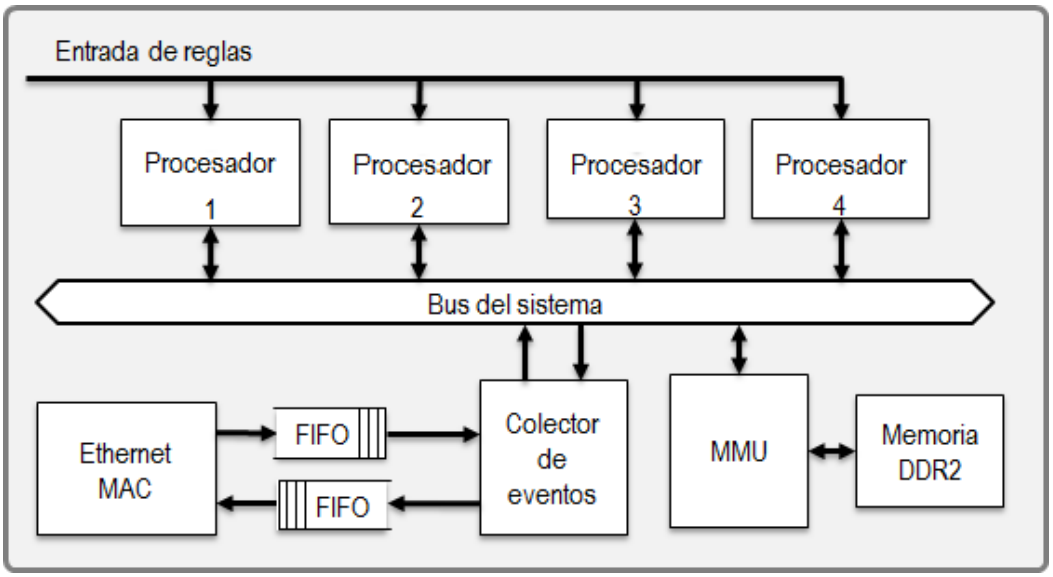


Figura 41. Arquitectura de implementación *hardware*

Para permitir la modificación del contenido de los bloques de memoria internos existen dos posibilidades:

- Modificarlas a través del puerto JTAG para una implementación basada en FPGA, opción dependiente de la tecnología.
- Integrar un bloque dedicado a la recepción de tramas de estrategia que modifiquen el contenido de las memorias, solución general.

Para implementaciones basadas en FPGA, en función de la ocupación del procesador de eventos y de la capacidad del dispositivo programable, puede replicarse el núcleo de procesamiento para aumentar el rendimiento del sistema final. En este caso es necesario implementar una unidad de despacho, que identifique eventos asociados a una u otra estrategia. En este tipo de soluciones, no es posible realizar procesado de eventos fuera de orden, al existir condiciones de disparo que hacen alusión a tendencias de atributos de títulos. En la figura 42 se muestra un ejemplo de arquitectura híbrida.

3.7.4. Comparativa entre arquitecturas

Las tres soluciones planteadas proporcionan tres puntos diferentes del espacio de soluciones entre prestaciones y flexibilidad/capacidad del sistema. En la figura 43 se presenta de forma cualitativa la relación entre estas arquitecturas, presentando los resultados sobre los ejes ya comentados. La relación entre las prestaciones y la carga de eventos se representa en la figura 44.

También cabe destacar que, en función de las necesidades del sistema, puede descartarse algunas de las soluciones. Un ejemplo es la necesidad de modificar las estrategias en tiempo de

ejecución, lo cual no es viable con una solución *hardware*, siendo necesaria una solución híbrida o basada en *soft-processor*.

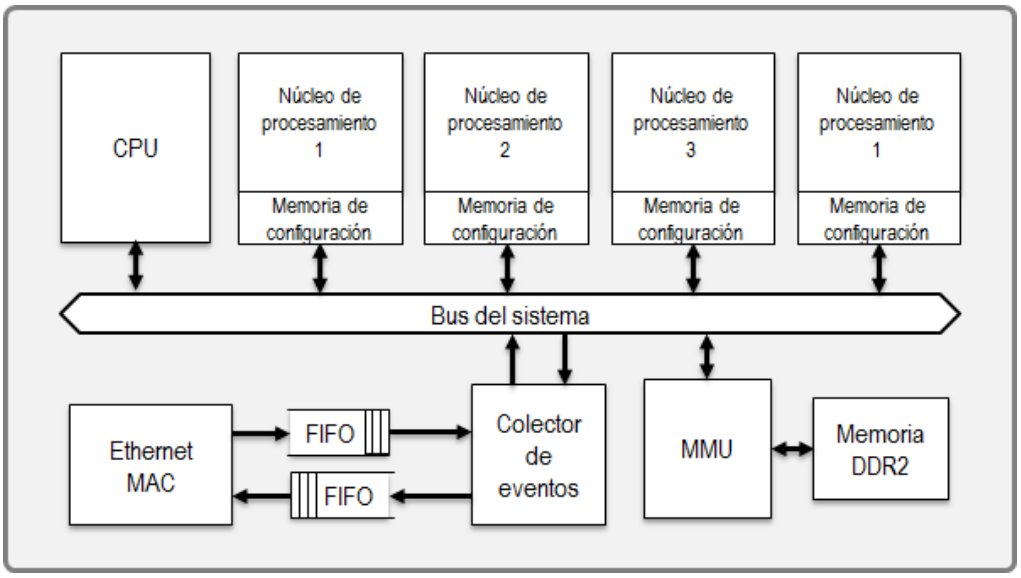


Figura 42. Arquitectura híbrida

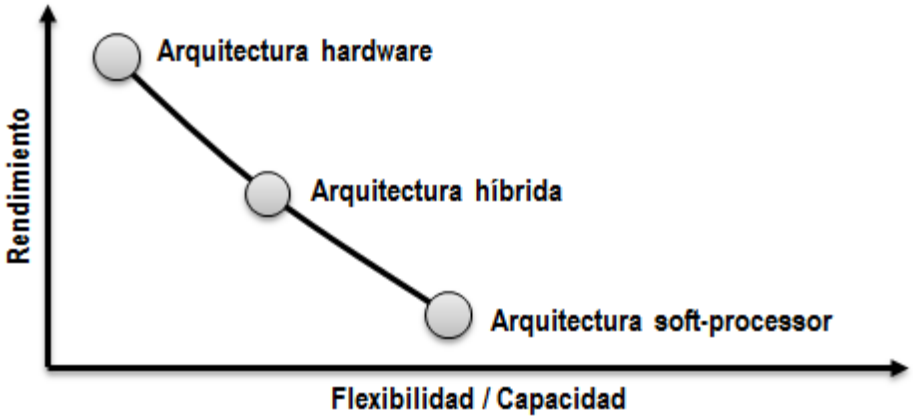


Figura 43. Comparativa entre arquitecturas

3.8. Paradigma de modelado

Para el modelado del sistema se parte de la aplicación RT-CORE ya desarrollada y escrita en C/C++, que servirá como modelo de referencia para la verificación de la funcionalidad. De igual forma se utilizará para la realización de las medidas de rendimiento y para las mejoras del diseño planteado.

A partir de la aplicación se realiza su perfilado estático y dinámico. El objetivo es determinar aquellas funciones, que representan el núcleo de procesamiento del sistema con mayor carga computacional, candidatas a ser implementadas como aceleradores *hardware*. Las técnicas utilizadas se pueden resumir en el análisis de ciclos de computación, ocupación de memoria, grafo de llamadas dinámico, iteraciones, saltos. Este tipo de análisis está soportado por herramientas tipo **gprof**, **valgrind** o **kcachegrind**.

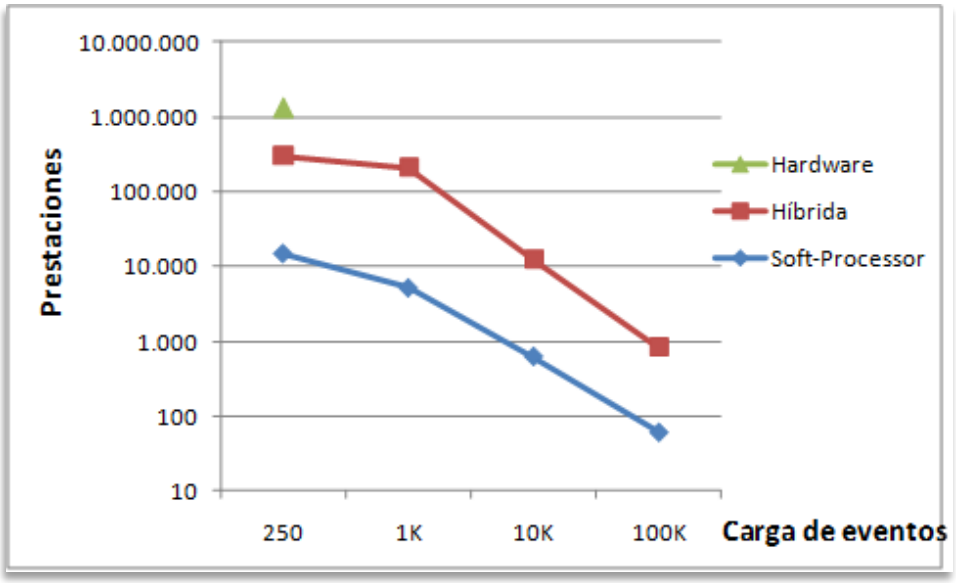


Figura 44. Efecto sobre las prestaciones de la carga de eventos procesados para las tres alternativas

A partir de este análisis se realiza la partición del sistema en bloques funcionales, utilizando módulos descritos en SystemC para cada bloque funcional. La aplicación se ha descompuesto en 34 módulos de procesamiento y 29 módulos de gestión de acceso a memoria local. Para la partición del sistema es necesario definir la interfaz de señalización entre módulos. Se ha utilizado un modelado a nivel de transacciones (TLM) de ciclo aproximado (CX)[134], en el que se abstraen los tipos de datos y se define un protocolo de señalización basado en petición respuesta (*Request*/*Validate*). La definición de los buses de datos y de señalización a nivel de ciclo será una de las tareas de la herramienta de síntesis de alto nivel para obtener una representación precisa a nivel de pines y de ciclos (*Pin Accurate/Cycle Accurate* – PA/CA). Durante el proceso de partición en bloques funcionales se ha mantenido agrupada la funcionalidad de la aplicación original para facilitar la localidad de los datos y facilitar su agrupamiento en estructuras fácilmente asignables a memorias locales.

El modelado de los tipos de datos es un aspecto clave por varias razones. En primer lugar la aplicación utiliza operaciones de suma, multiplicación y división en punto flotante cuya implementación en *hardware* es costosa. Para ello se han transformado las operaciones indicadas a sus operaciones equivalentes en punto fijo. En este proceso la pérdida de precisión puede ocasionar problemas en la funcionalidad del sistema (operaciones de comparación, etc.). Por ello ha sido necesario determinar los límites máximos y mínimos en los valores de los parámetros y variables de la aplicación. Para dar soporte a las operaciones se ha utilizado una librería de operadores optimizada para operaciones en punto fijo.

Por otra parte, para parametrizar los buses de datos y mantener la flexibilidad de propagación de cambios en el modelo se ha optado por definir tipos de datos organizados en estructuras de datos parametrizables. La compatibilidad de tipos de datos representa otra de las fortalezas del modelo, siendo necesario utilizar las funciones de conversión de tipos

aportados por SystemC de forma explícita para evitar incongruencias en la construcción del modelo final del sistema.

Los bloques de procesamiento pueden dividirse, según su funcionalidad, entre los bloques conceptuales presentados anteriormente: interfaz, núcleo de procesamiento, ejecutor de resultados y estado del procesador.

De acuerdo con la metodología de diseño y el flujo de síntesis propuesto el desarrollo del entorno de verificación se ha realizado también en SystemC. En este modelo SystemC se incluye una parte de las funciones que se implementarán posteriormente en el dominio *software* y que forman parte fundamental del sistema. Además será necesario incluir algunas funciones auxiliares, como por ejemplo la lectura de datos de test desde ficheros, que después serán sustituidas por lecturas desde los *sockets* de la interfaz Ethernet. Este entorno de verificación será reutilizado a lo largo del flujo de diseño como se ha comentado, mediante la creación de adaptadores o *wrappers* que adaptan las señales a nivel RTL.

3.9. Metodología de diseño

En todo diseño, ya sea *hardware* y/o *software* se pueden enumerar tres fases claves en la metodología de diseño: especificación, diseño y verificación.

En la fase inicial se analizan las especificaciones del sistema a realizar, ya sean estas funcionales, temporales o estructurales. Una vez definidas y acotadas, se procede al diseño del sistema en el que se decide la tecnología, las etapas de desarrollo, la partición del diseño (si la hubiese), etc. Esto produce una primera versión funcional del sistema, que se utiliza para una primera verificación a ese nivel funcional.

En el diseño del procesador de eventos, los datos de entrada al sistema se recogen como tramas TCP/IP. El tratamiento de la comunicación implica la extracción de la carga útil de datos de los paquetes y el tratamiento de los paquetes de control. Para dar solución a estos requerimientos se diseña un *System-on-chip (*SoC*)* heterogéneo del tipo híbrido definido más arriba, y formado por un microprocesador empotrado, un conjunto de aceleradores *hardware*, y un conjunto de periféricos interconectados (Buses, DMA, etc.). De esta forma, el procesador de eventos se comporta como un procesador dedicado integrado en una plataforma SoC heterogénea, siendo el microprocesador principal el encargado de preparar las tramas de datos y hacerlas disponibles.

Los datos de entrada del procesador de eventos se estructuran como un paquete de datos organizados al byte, con un formato definido y que son analizados por la aplicación *software* que es ejecutada en el microprocesador empotrado. Dichos datos, una vez compactados se envían al bloque acelerador que los procesa [23].

Es necesario por tanto abordar las siguientes tareas:

- Diseño de la plataforma, en el que se incluye la selección de aquellos bloques IP necesarios, así como del diseño del *software* embebido a ejecutar en el microprocesador.

- Diseño del procesador de eventos, tomando como especificación la aplicación *software* disponible.

La metodología incluye la etapa de validación. Para validar esta opción de diseño se han realizado dos implementaciones. En primer lugar se ha optado por la implementación sobre una FPGA Virtex-5 disponible en la tarjeta de desarrollo ML510 de Xilinx. Este dispositivo posee dos núcleos microprocesadores PowerPC440 empotrados en la FPGA como *hard* IP. De igual forma, la tarjeta posee dos puertos Gigabit Ethernet que permiten disponer de ancho de banda suficiente para la implementación del prototipo. En segundo lugar se ha realizado una implementación ASIC del bloque IP diseñado para el procesador de eventos.

En la figura 45 se muestra el flujo de diseño utilizado para el caso de la FPGA. Este flujo incluye el diseño de la plataforma. En general, se parte de la misma descripción funcional del sistema por lo que las etapas iniciales del flujo serán comunes con el diseño del ASIC, aunque optimizadas para la tecnología destino.

La síntesis de alto nivel y la síntesis lógica serán -en cambio- muy dependientes de la tecnología a utilizar, siendo el espacio de soluciones más reducido en el caso de las FPGAs. Además de esto, la fase de implementación está mucho más acotada en un dispositivo como una FPGA donde los recursos están pre-asignados a una posición definida, siendo necesaria únicamente su programación.

En las siguientes secciones se describen en más detalle las fases seguidas en el diseño.

3.9.1. Fase de análisis

En esta fase de análisis del diseño se obtiene el perfil de ejecución de la aplicación original. El objetivo es disponer de la información relativa a la complejidad de la aplicación, su modularidad, así como características de coste computacional y temporal de las distintas funciones que la componen. Esta información, junto a un estudio de la dependencia de datos, facilita la toma de decisiones relacionadas con la partición de la aplicación, las necesidades de comunicación y su conectividad durante la fase de diseño, produciendo como resultado la jerarquía del diseño del procesador y los elementos de procesamiento y sus respectivos recursos.

3.9.2. Diseño del procesador de eventos

En el diseño del procesador de eventos se sigue el flujo de síntesis de alto nivel establecido. La primera fase consiste en la captura del diseño, es decir, traducir la aplicación a una descripción funcional de alto nivel en SystemC. En esta primera fase se definen las particiones internas del procesador de eventos, los protocolos de comunicación y sincronismo, así como las memorias que son accedidas por distintos módulos. Es también en esta etapa donde debe definirse la interfaz del procesador de eventos mediante el protocolo con el que se comunicará con el resto del sistema siguiendo metodologías TLM.

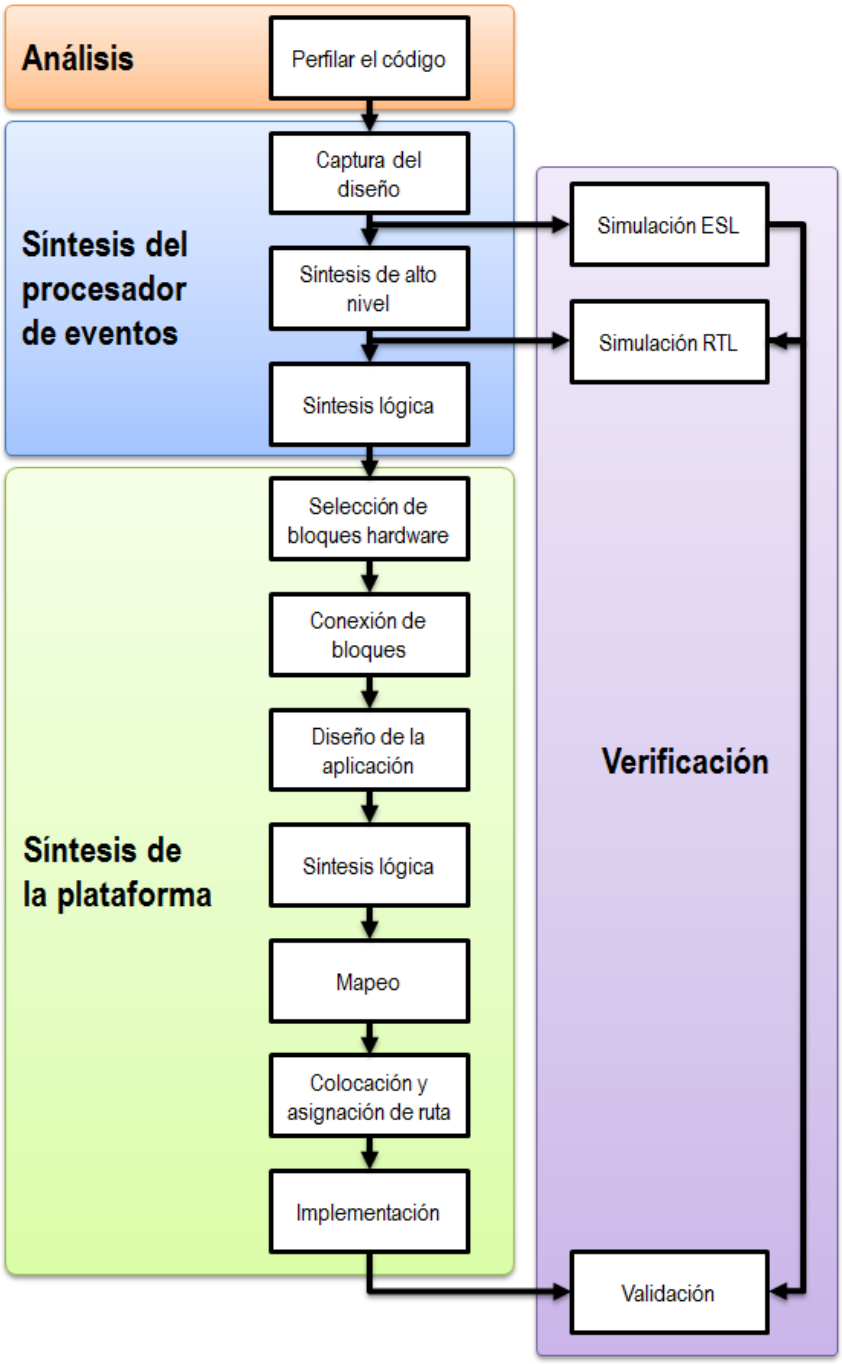


Figura 45. Diagrama del flujo de diseño

Cuando la descripción del procesador de eventos haya sido verificada mediante simulación a nivel ESL, comparando su funcionalidad con la de la aplicación original en términos de entrada/salida, se pasa a la primera fase de síntesis o síntesis de alto nivel, que traducirá el diseño a una descripción a nivel RTL en VHDL o Verilog. Existen parámetros objeto de optimización a la hora de sintetizar el diseño, entre los cuales se destacan las prestaciones, la potencia y el área (PPA). En muchos casos habrá que establecer una estrategia final ya que no es posible optimizar todos los parámetros indicados de forma simultánea. Es decisión del diseñador, en función de sus requisitos, guiar la síntesis con el fin de obtener la solución

deseada. Conviene tener presente que todo diseño es en última instancia un proceso creativo. Estas decisiones de optimización se incluirán en todos los niveles de síntesis, tanto de alto nivel y lógica, como en la síntesis física.

En la siguiente fase es preciso verificar que la solución obtenida tras la síntesis de alto nivel mantiene la funcionalidad especificada inicialmente. En caso contrario será necesario ajustar las restricciones impuestas al diseño e incluso la descripción de alto nivel, de forma que la funcionalidad de la solución obtenida a nivel RTL coincida con la de su diseño capturado, de entrada.

Una vez obtenida una descripción a nivel RTL, se realiza la fase de síntesis lógica que lo traduce a un diseño compuesto por la interconexión de células básicas de la librería tecnológica. Esta fase tiene por objetivo la obtención de un *netlist* (fichero de instancias de células básicas y sus conexiones), generalmente NGC o EDIF para el caso de FPGA, normalmente Verilog para el caso de un ASIC. Los parámetros objetivo son, por una parte, la determinación del tiempo de ciclo y, por otra, la optimización del área/recursos utilizados (figura 46).

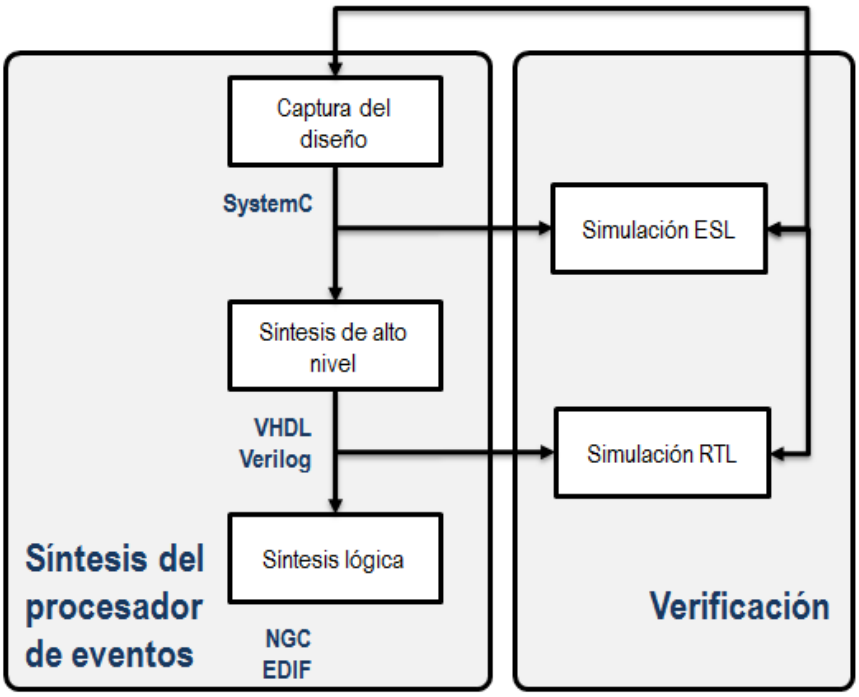


Figura 46. Diseño del procesador de eventos

3.9.3. Diseño de la plataforma

A continuación se acomete la implementación en la plataforma. El diseño de la plataforma *hardware* comienza con la creación del sistema, fase en la que se instanciarán los diferentes bloques IP necesarios, configurando aquellos atributos parametrizables en su caso. En este paso se importará también el procesador de eventos y se conectará al resto del sistema usando una interfaz de interconexión estandarizada, en este caso LocalLink, bus propietario de Xilinx [23]. Será necesario seguir el flujo de diseño apropiado para la implementación sobre una FPGA, es decir, sintetizar el diseño (desde la descripción HDL correspondiente), asignarlo o hacerlo

corresponder (mapearlo) sobre los elementos disponibles en la tecnología sobre la que se implementa el diseño, y decidir la colocación de cada uno de estos elementos y su ruta de interconexión.

El siguiente paso consiste en el diseño del *software* del sistema, eligiendo el sistema operativo sobre el que se ejecutará la partición *software* de la aplicación y las tareas auxiliares, así como aquellas librerías necesarias para dar soporte a los periféricos. El último paso es añadir la aplicación de usuario, en este caso la aplicación que recibe los eventos por la interfaz de entrada al sistema –en el caso que nos ocupa se trata de tramas TCP/IP–, que realiza su pre-procesamiento si fuera necesario, y los envía al acelerador *hardware* diseñado anteriormente. A continuación, la aplicación *software*, así como las librerías asociadas, deberán compilarse y enlazarse para generar un fichero ejecutable, típicamente en formato ELF, que se ejecutará en el microprocesador de la plataforma.

El siguiente paso es la validación de la plataforma. Con el fin de validar el correcto funcionamiento del sistema se deberá descargar el fichero de configuración de la FPGA generado tras la implementación, así como el ejecutable para el microprocesador. Hecho esto, se enviarán eventos a través de la interfaz Gigabit Ethernet y se esperarán las tramas de salida del mismo.

3.9.4. Verificación y validación del sistema

La fase de verificación se realiza a distintos niveles de abstracción, así como sobre distintas vistas del diseño. Para ello, se realiza la verificación funcional del sistema tanto en su descripción ESL como a nivel RTL, la verificación física tras su implementación, y la verificación del comportamiento temporal mediante un análisis del *slack* (u holguras) de todas las rutas lógicas del diseño. Finalmente se realiza el prototipado del sistema, verificando así su funcionalidad final y en tiempo real. Esta última verificación la denominamos validación del sistema.

3.10. Síntesis del procesador de eventos

En este apartado se describen las etapas de síntesis de alto nivel y verificación del diseño RTL generado para el procesador de eventos del sistema. Se pretende mostrar el flujo de diseño partiendo de un *software* de referencia que se transforma a una especificación funcional en SystemC (captura del diseño), y desde ella obtener un modelo sintetizado y optimizado, que pueda posteriormente integrarse en la plataforma para el desarrollo y validación del sistema final.

Se trata de obtener un diseño SystemC, particionado y basado en la aplicación *software* original que describe su funcionalidad, transformado la aplicación para ser así ejecutada en *hardware* aprovechando las mejoras que esto supone. Desde SystemC se trata de sintetizar el sistema.

Se pretende en este apartado definir directrices sobre los pasos a seguir para la síntesis y verificación de un sistema *hardware* complejo formado por un conjunto de bloques funcionales interconectados.

3.10.1. Criterios de optimización

En general, a la hora de realizar la transformación de una descripción funcional a una microarquitectura, existen varios factores a optimizar: ciclos de latencia, periodo de reloj, uso de recursos y potencia consumida, entre otros. En general, estos factores dependen entre sí, de tal forma que disminuir el tiempo de ciclo de reloj puede ir asociado a un incremento en el uso de recursos o a un mayor número de ciclos de latencia. Esta dependencia entre los factores hace imposible optimizar todos conjuntamente, siendo necesario establecer un compromiso en función de los requisitos del sistema como veremos.

3.10.2. Estrategias de síntesis

A la hora de realizar la síntesis, existen principalmente dos estrategias a valorar: *top-down* y *bottom-up*. Ambas se basan en la partición del sistema en módulos más simples y ligeros, que al interconectarlos consiguen la funcionalidad del módulo principal. En general, al módulo que integra e interconecta al resto se le conoce como "*top* del diseño".

La estrategia *top-down* (figura 47) consiste en crear un proyecto único para la síntesis, que tenga como módulo principal el *top*, el cual instancia el resto de bloques, y causa también que la propia herramienta los sintetice y conecte.

Por el contrario, una estrategia *bottom-up* (figura 48) se basa en la síntesis por separado de cada bloque integrado en el sistema para finalmente integrarlos todos mediante un módulo top dedicado únicamente a definir la conectividad entre ellos. Es necesario también tener en cuenta los efectos en los bordes de los módulos para facilitar la interconexión. Veremos más adelante que ambos procesos se pueden combinar para llegar a un compromiso entre tiempo de síntesis y calidad de los resultados.

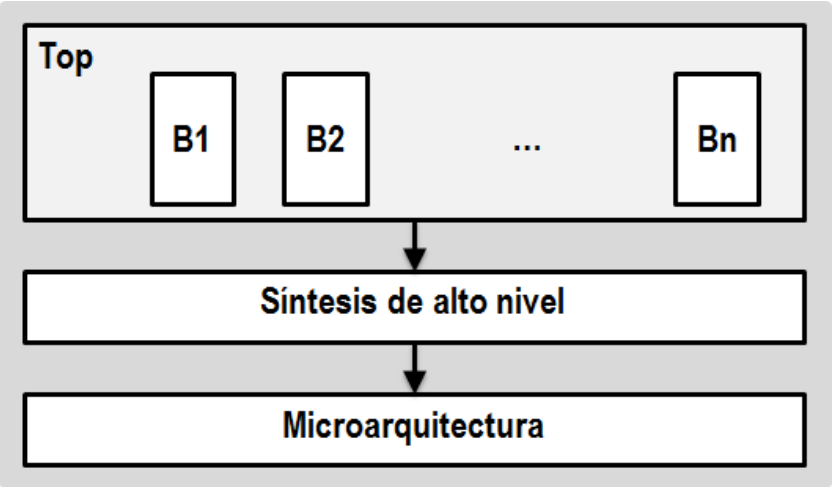


Figura 47. Estrategia *top-down*

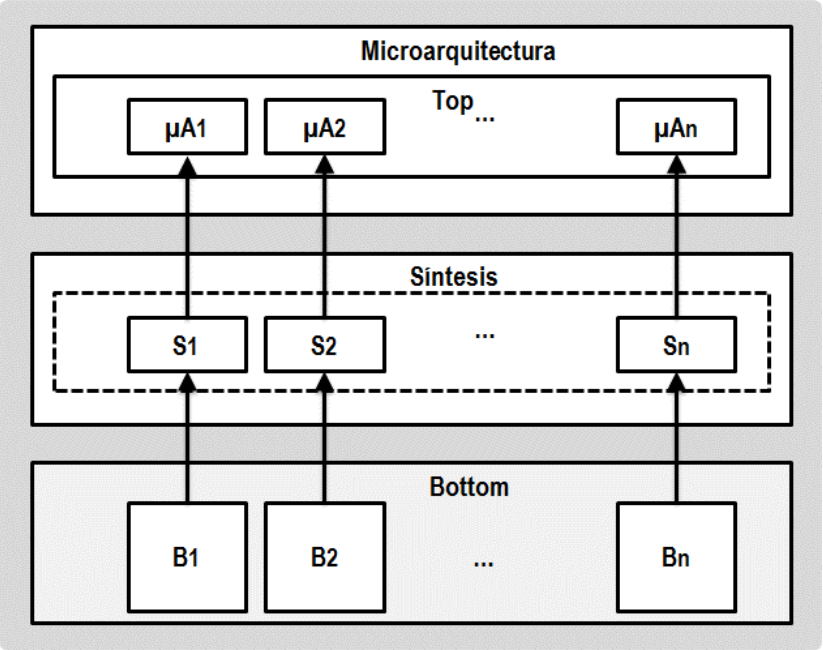


Figura 48. Estrategia *bottom-up*

En el procesador de eventos se ha optado por una metodología *bottom-up* debido a las posibles ventajas inherentes:

- **Paralelización de la síntesis**. Gracias a la creación de proyectos independientes para cada bloque, y aprovechando los recursos multi-cpu disponibles en el entorno de diseño, puede lanzarse simultáneamente la síntesis de un alto número de bloques, minimizando el tiempo de síntesis.
- **Disminución del tiempo de depurado**. Gracias a la separación de la síntesis de cada bloque, durante la etapa de depurado en la verificación RTL pueden realizarse modificaciones en un único bloque y conseguir una nueva versión del diseño sin tener que sintetizar el resto de bloques.

3.10.3. Síntesis de alto nivel

Para la síntesis de alto nivel se ha optado por hacer uso de la herramienta C-to-Silicon (CtoS) de Cadence, la cual ofrece la posibilidad de realizar la síntesis partiendo de un diseño en SystemC, y obteniendo como resultado la descripción RTL del módulo en Verilog. El proceso de síntesis de alto nivel se guía creativamente para recorrer el espacio de diseño y obtener aquellas soluciones optimizadas que cumplen con los requisitos de funcionalidad. Los principales parámetros controlados por el diseñador durante la fase de síntesis se explican en los apartados siguientes. El mecanismo utilizado es la creación de *scripts* en lenguaje Tcl (*Tool Command Language*) que incorporan la información requerida.

Frecuencia de reloj

Es la principal restricción impuesta al diseño que determina el tiempo de ciclo y también la latencia total del sistema. Se especifica su frecuencia, así como su ciclo de trabajo (los valores se dan en términos de periodo de reloj y *offset* del flanco de bajada).

```
define_clock -name clk -period 5000 -rise 0 -fall 2500
```

Los valores de periodo, flanco de subida y flanco de bajada están indicados en picosegundos. En el ejemplo se muestra la definición de un reloj denominado *clk* que tiene un periodo de 5ns (200 MHz) y un ciclo de trabajo del 50%.

Bucles combinacionales

En el modelo SystemC es normal la existencia de bucles combinacionales, típicamente para realizar determinadas operaciones en *arrays*, que desde el punto de vista funcional deben ejecutarse en un único ciclo de reloj. Para su planificación puede optarse por varias soluciones:

- **Romper el bucle.** Los datos se registran al final de cada iteración (representados mediante una sentencia `wait()` en SystemC). Cada iteración se realiza en un ciclo de reloj. Minimiza el uso de recursos y el periodo de reloj, pero aumenta la latencia de procesamiento.
- **Desenrollado completo del bucle**. Se implementan todas las iteraciones del bucle en un único ciclo, replicando la lógica interna del bucle, una por cada iteración. Favorece una minimización del número de ciclos, aunque puede tener impacto en el incremento del periodo de reloj. Por el contrario aumentan los recursos *hardware* consumidos, aunque facilita la optimización de la lógica generada.
- **Desenrollado parcial del bucle.** Para reducir la latencia también es posible realizar el desenrollado parcial del bucle de tal manera que es posible optimizar la lógica incluida para la ejecución del conjunto de iteraciones del bucle sin impactar en el ciclo de reloj. El número de iteraciones a desenrollar depende de si el número de iteraciones del bucle es constante o es necesario definir condiciones de salida para bucles con iteraciones variables.
- **Segmentación y solapamiento del bucle.** En este caso la ejecución de las iteraciones del bucle se solapan con el objetivo de reducir la latencia final. Se define un intervalo de inicialización (II) que se corresponde con el número de ciclos de reloj requeridos para el comienzo de una nueva iteración, respetando las posibles dependencias de datos. De igual forma es necesario tener en cuenta los problemas de bloqueos (*stall*) de tal forma que se permita vaciar el *pipeline* (cauce de procesamiento) cuando la dependencia sea inevitable o no se pueda ocultar. Esta solución mejora el rendimiento para bucles con alta carga computacional en cada iteración, pero es ineficiente para bucles de poca profundidad ya que precisa de lógica adicional para el control del *pipeline*.

Como estrategia general, y con objeto de minimizar los recursos usados, la estrategia seguida consiste en aplicar la técnica de rotura del bucle en todos aquellos bucles

combinacionales presentes. El motivo es minimizar el conjunto de recursos utilizados teniendo en cuenta que el sistema se va a implementar en una FPGA. Para Cadence CtoS, las órdenes utilizadas se muestran a continuación.

```
set combo_loops [find_combinational_loops]
foreach loop $combo_loops {
    break_combinational_loop  $loop
}
```

Funciones no planificables

En la descripción del diseño en alto nivel se permiten las llamadas a funciones globales. En general estas llamadas las resuelve el planificador mediante la síntesis de un bloque independiente y un mecanismo de comunicación haciendo uso de un protocolo de petición y respuesta. Sin embargo, en algunas ocasiones, especialmente cuando la latencia no es constante o se realizan llamadas desde distintos procesos concurrentes, no es posible aplicar esta solución. Esas funciones son entonces no planificables.

Para estos casos, es necesario optar por realizar el *inline* de la función, es decir, sustituir la llamada a la función por una copia de la misma. La inmediata consecuencia es el incremento en el consumo de recursos.

Realizar el *inline* de las funciones facilita la optimización de la jerarquía de funciones, permitiendo en muchos casos eliminar lógica innecesaria. Por ello parece de interés realizar incluso un *inline* de las funciones que no producen conflictos de acceso concurrente. Sin embargo, llevar esta acción sobre todas las funciones del diseño trae como consecuencia una explosión en el consumo de recursos.

A continuación se muestra la sintaxis de CtoS para realizar *inline* de una función específica funcion1.

```
inline /designs/modulo/modules/behaviors/funcion1
```

Asignación de memorias

Las memorias siempre suponen un punto singular necesario en la creación de la microarquitectura del sistema. Se generan memorias cuando existen *arrays* de datos en la descripción SystemC del diseño, y se debe decidir los recursos a utilizar para almacenarlos. Existen diferentes opciones, que se enumeran a continuación:

- **RAM interna**. Se crea un modelo RTL para la memoria, que incluye los puertos de dirección y datos y el protocolo de lectura y escritura. Para el caso de una implementación basada en FPGA, se usan como recursos los bloques BRAM o las LUTs de los SLICEM para almacenar la información. El uso de una u otra opción depende de los ciclos de espera introducidos entre la generación de la dirección y la lectura del dato.
- **Registros**. Todos los datos se almacenan en registros. Esta solución está orientada a *arrays* de tamaño reducido.

- **RAM de terceros**. Es posible incluir el modelo RTL de una memoria RAM de un tercero. Para ello es necesario crear un *wrapper* y respetar las interfaces de la memoria.

El uso de registros o BRAM/LUTs dependerá de la disponibilidad de recursos del dispositivo y del uso que haga el resto del diseño. Por el alto uso de registros en el diseño, en el presente trabajo se ha decidido usar RAM interna para todos los *arrays* del diseño, como se muestra a continuación.

```
set memories [find $top_path/modules/*/arrays/*]
foreach mem $memories {
    allocate_builtin_ram $mem
}
```

Además es posible realizar diferentes operaciones para gestionar las memorias. En primer lugar, cuando los *arrays* se crean con tipos de datos heterogéneos tipo *struct* es posible utilizar modelos monolíticos, en los que se concatenan en la misma fila toda la estructura, o por el contrario usar un modelo distribuido, en el que se genera una memoria para cada componente de la estructura. Es posible controlar qué modelo se utiliza para cada *array*.

De igual forma existen mecanismos para combinar varios *arrays* con el objetivo de optimizar la ocupación de los bloques de memorias, ya sea los que existen en la FPGA o los que se han incluido en el diseño del ASIC.

Existen dos técnicas clásicas para agrupar los *arrays* en memorias. En la asignación (o mapeado) horizontal, los *arrays* definidos en el modelo C/C++/SystemC se concatenan en memorias de mayor tamaño y cuyo ancho es el ancho del tipo de datos más ancho. Se realiza por tanto una "agrupación por direcciones". En este caso los *arrays* comparten los mismos puertos de lectura y escritura, lo cual puede crear contención en el acceso a la memoria.

Por el contrario, en la asignación (o mapeado) vertical, se agrupan los *arrays* para crear memorias más anchas, con el ancho de al menos la suma de los anchos de los *arrays* y con el tamaño del mayor de los *arrays*. En este caso se habla de "agrupación de datos".

De igual forma es posible descomponer un *array* en varios, por ejemplo para acceder a varias zonas del *array* en el mismo ciclo de reloj. Además de estas agrupaciones y descomposiciones de *arrays*, existen diferentes técnicas y opciones que permiten afinar la optimización del uso de los recursos de memoria disponibles.

Uso de DSPs

En el caso del uso de FPGAs, es posible utilizar los recursos DSPs disponibles en la FPGA para disminuir el consumo de *slices* o LABs. Para ello las operaciones aritméticas complejas se asignan o mapean en bloques DSPs, requiriendo procesos de reestructuración interna de operaciones. Es posible limitar la latencia de las unidades para cumplir las restricciones temporales. De igual forma se puede definir el ancho mínimo en el número de bits de los operandos a partir del cual el uso de bloques DSP produce ahorro en consumo de recursos del diseño.

```
use_dsp /designs/$modulo
```

Relajación de la latencia

Esta opción permite a CtoS incrementar el número de ciclos de latencia de una función, con el fin de facilitar la planificación de la misma. La relajación de latencias permite añadir ciclos de reloj para cumplir las restricciones temporales de tiempo de ciclo. Sin embargo es importante restringir estas operaciones tanto en aquellas funciones que implementan los protocolos de comunicación externos, como en aquellas que realizan los accesos a memoria o que se dedican a leer y escribir sobre los puertos de control de E/S, ya que añadir estados intermedios impide el correcto funcionamiento del protocolo. Existen mecanismos de control para aplicar esta técnica en distintas funciones del modelo.

```
set behaviours [find $top_path/modules/*/behaviors/*]
foreach beh $behaviours {
    set_attr relax_latency "false" $beh
}
```

Automatización de la síntesis

Para la automatización del proceso de síntesis, las opciones y restricciones impuestas mediante órdenes Tcl se organizan en *scripts* Tcl para lanzarlas en modo *batch*. La combinación de este proceso con la estrategia *bottom-up* permite acelerar la síntesis de los diferentes bloques del diseño.

```
ctos scripts/ctos.tcl -log log/ctos.log
```

En el *script* Tcl existirán dos etapas diferenciadas. La primera es la que configura el proyecto de CtoS, indicando los ficheros fuente, la frecuencia de reloj, así como el dispositivo de prototipado final. La segunda define las estrategias de síntesis guiada, con las opciones y restricciones elegidas.

```
# Properties
new_design $modulo
set_attr auto_write_models "true" /designs/$modulo
define_sim_config -model_dir "./model" /designs/$modulo
set_attr source_files [list src/$modulo.cpp] /designs/$modulo
set_attr compile_flags " -w -I../../include -I./src" /designs/$modulo
set_attr auto_save_dir $modulo /designs/$modulo
define_clock -name clock -period 10000 -rise 0 -fall 5000
set_attr implementation_target FPGA [get_design]
set_attr  fpga_target  [list  Xilinx  Virtex5  xc5vfx130t-1-ff1738]
[get_design]
set_attr fpga_install_path [exec which xst] [get_design]
set_attr fpga_work_dir "./fpga_work" [get_design]

# Strategies
set combo_loops [find_combinational_loops]
foreach loop $combo_loops {
    break_combinational_loop  $loop
}
foreach mem [find $top_path/modules/*/arrays/*] {
    allocate_builtin_ram $mem
}
```

```
foreach beh [find $top_path/modules/*/behaviors/*] {
    set_attr relax_latency "true" $beh
}
schedule -effort high -passes 200 /designs/$modulo
allocate_registers
write_rtl -recursive -o [concat ./[string trim $model]] $top_path
/modules/$modulo
```

3.10.3.1. Análisis de resultados

A continuación se detallan algunos resultados obtenidos de la síntesis de alto nivel. Su estudio debe guiar siempre al diseñador en el análisis y optimización del diseño.

Control and Data Flow Graph (CDFG)

En el CDFG (figura 49) se representan tanto el flujo de datos como el de control del algoritmo implementado, incluyendo bucles, así como las bifurcaciones generadas en base a determinadas condiciones. Este análisis temporal se realiza en etapas tempranas de la síntesis de alto nivel con el objeto de capturar y representar el flujo de ejecución de la función bajo estudio.

El CDFG permite analizar las latencias del sistema para cada posible camino. En un flujo interactivo es posible insertar ciclos de *retiming* para igualar la latencia y simplificar las máquinas de estado de control de la función representada, al igual que es posible limitar la latencia máxima en ciclos entre dos puntos definidos en el código fuente.

Análisis de la ruta crítica

El análisis de la ruta crítica permite obtener el periodo de reloj del sistema. Para ello estima los retardos de cada uno de los bloques obtenidos durante la planificación, realizando un análisis temporal según la información obtenida de las librerías tecnológicas indicadas. En la información obtenida se representa el tiempo de *hold* del registro origen de la ruta crítica, retardos de la propagación de la señal (separación entre tiempos de los bloques), el *slack* y retardo por *fan-out*.

En el ejemplo de la figura 50, se ha indicado una restricción en frecuencia de funcionamiento de 200 MHz, es decir, se trata de una síntesis *time-constrained* para un periodo de reloj de 5 ns. Como puede observarse, existe un *slack* negativo de 1,56 ns, lo que implica que la frecuencia propuesta no es alcanzable con el diseño actual. Por tanto será necesario introducir restricciones adicionales relajando el periodo de reloj, o introduciendo ciclos adicionales de reloj, ya sea añadiendo sentencias `wait()` en el modelo SystemC, o, lo que resulta más adecuado, introduciendo restricciones adicionales directamente en la síntesis en CtoS.

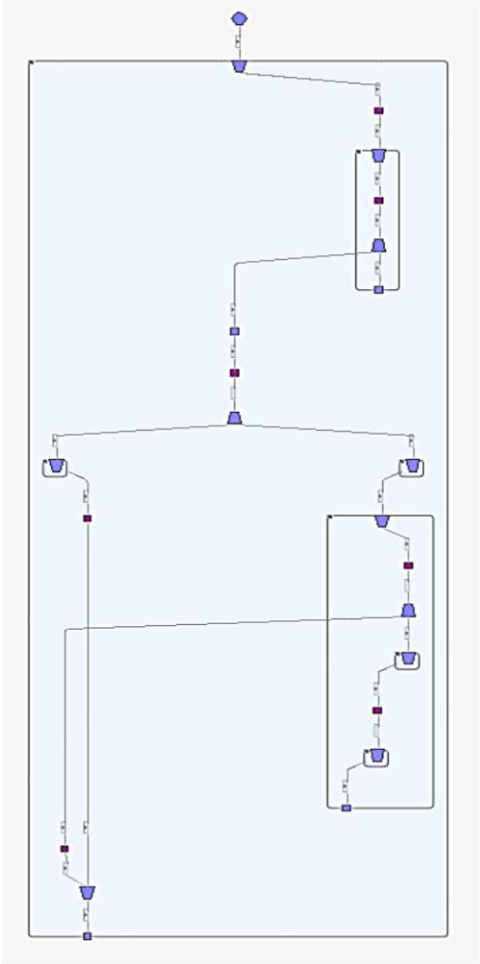


Figura 49. Vista del *Control and Data Flow Graph (CDFG)*

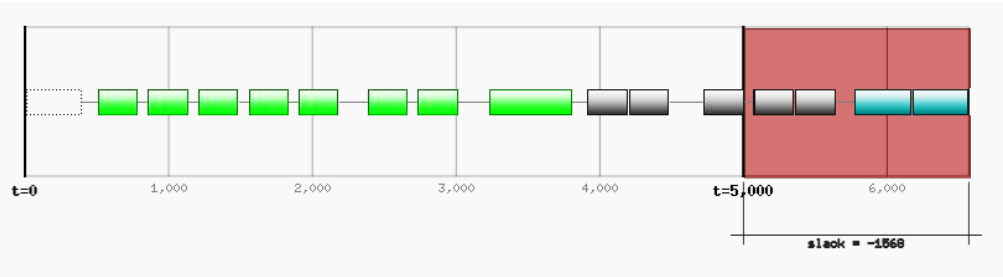


Figura 50. Análisis de ruta crítica

Análisis de área

En un diseño sobre FPGAs los análisis de área están orientados a conocer los recursos consumidos por el diseño. En el caso del ASIC por el contrario, este análisis mostrará los distintos tipos de recursos utilizados por el diseño, en función de la información disponible en la librería tecnológica del fabricante. La herramienta de síntesis de alto nivel permite, tras obtener la arquitectura RTL del módulo estimar el área consumida por cada módulo del diseño, tal y como se muestra en la figura 51, organizado como área combinacional y área secuencial. La herramienta de síntesis genera un bloque para cada función del código SystemC, y además

incluye lógica adicional para las llamadas a cada función. El consumo de área puede visualizarse organizado por funciones.

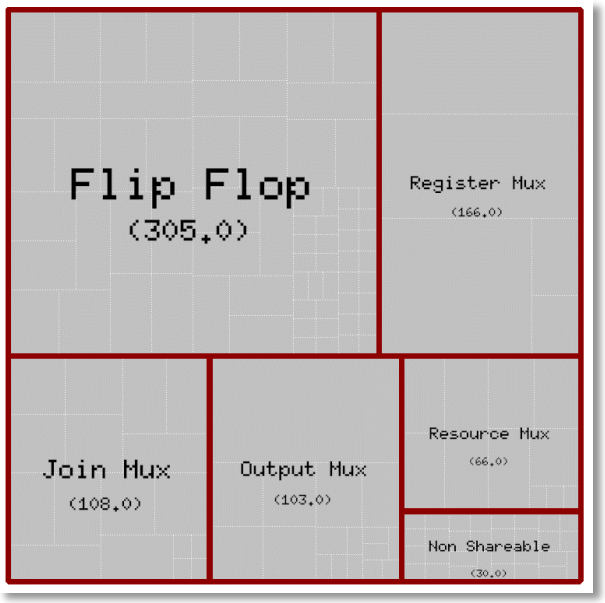


Figura 51. Análisis de área

Análisis de potencia

Otro de los resultados a analizar para determinar la calidad de la arquitectura obtenida es el "consumo" o nivel de potencia y por tanto la energía consumida. La potencia influye en el diseño de la fuente de alimentación y en el diseño de las interconexiones. La potencia disipada en el circuito es un factor determinante de su coste y fiabilidad. En el caso de que el circuito vaya a ser utilizado en sistemas de altas prestaciones es preciso reducir los costes de enfriamiento (encapsulados, sistemas de refrigeración) e incrementar la fiabilidad de los mismos. Para el caso de sistemas portables alimentados por baterías, el objetivo es alargar su duración entre recargas y minimizar el número de recargas durante su vida útil. Aquí el factor predominante es la energía determinada por la expresión $E = \sum_i P_i \times t_i$.

Las decisiones tomadas en la síntesis de alto nivel tienen gran impacto en términos de calidad de resultados (QoR) (figura 52). Esto es aplicable tanto a nivel de recursos (área) como de prestaciones (potencia consumida y tiempo de ejecución). Es posible reducir hasta el 80% el consumo con decisiones adecuadas en alto nivel. Estas decisiones en alto nivel intervienen de forma determinante en el comportamiento del dispositivo final [135].

A nivel RTL, los principales métodos referenciados de reducción del nivel de potencia son los siguientes [136]:

- *Clock gating*: utilizado para reducir la potencia dinámica de conmutación en la ruta del reloj. Es el método más utilizado (54%).
- *Power gating:* permite la reducción de la potencia disipada por corrientes de fuga desconectando la fuente de alimentación en las zonas no operativas del circuito

para determinados modos de operación. Para ello se utilizan elementos básicos tales como *switches* de control de VDD y GND, células de aislamiento y Flip-Flops de retención de estados SRFFs. Su utilización es del orden del 48%.

- Combinación entre *clock gating*, y escalado dinámico de tensión y frecuencia (DVFS) (40%) (figura 53) [135].

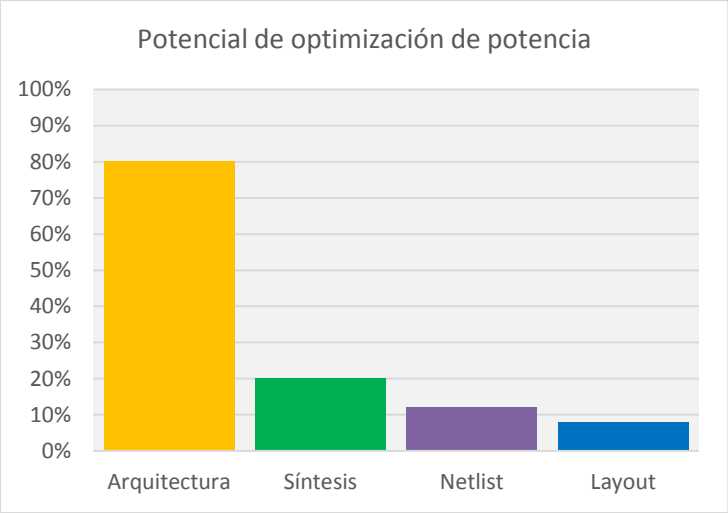


Figura 52. Oportunidades de optimización de potencia en diferentes etapas del diseño

A diferencia de los métodos citados a nivel RTL, en síntesis de alto nivel es posible optimizar el consumo de potencia con otros métodos. Durante la fase de modelado, la realización de optimizaciones en el código fuente ya permite reducir el consumo de potencia, ya sea en la fase de computación, de comunicación o de almacenamiento del modelo, tal como se muestra en la figura 54.

a) Impacto de la arquitectura de memorias en la potencia

El diseño de alto nivel de la arquitectura de memoria igualmente afecta al consumo de potencia, al igual que a las prestaciones del sistema final. Ejemplos de medidas a adoptar pueden ser, entre otras, las siguientes: acceso a datos locales para disminuir el tráfico de datos con memoria, organización de la memoria en bloques para activar únicamente aquellos que son accedidos y dejar el resto inactivos.

Durante la síntesis de alto nivel es también posible utilizar diferentes configuraciones de direccionamiento, ancho de palabras y relación de aspecto de los *arrays* de tal forma que se habiliten los recursos precisos, especialmente sensibles para el caso de las FPGAs. Por ejemplo en CtoS, durante la fase de definición de la microarquitectura es posible realizar la separación (*split*) en múltiples bancos de memoria independientes y la unión (*merge*) del *array* para su posterior implementación como una memoria o banco de registros.

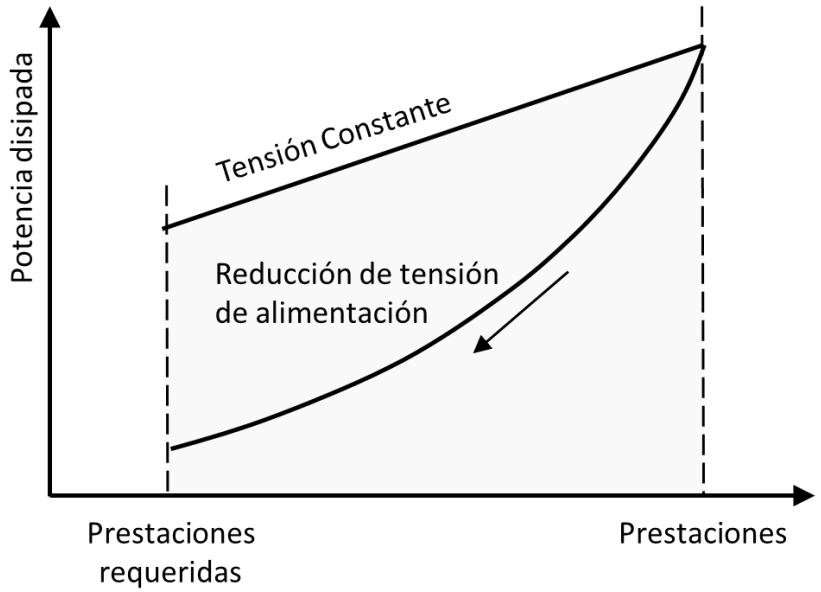


Figura 53. Efecto de la reducción de la tensión de alimentación

Computación:	X*Y+X*Z	X*(Y+Z)
Comunicación:	for (i= 1..n) read (X) Y = n * X end	Read (X) for (i= 1..n) Y = n * X end
Almacenamiento:	for (i= 1..n) Z[i] = X[i] * Y[i] end for (i= 1..n) W[i] = Z[i] - 1 end	for (i= 1..n) W[i] = X[i] * Y[i] - 1 end

Figura 54. Modificaciones del código fuente para aumentar la calidad de resultados

Este proceso puede basarse en el esquema de direccionamiento (por ejemplo, se implementan en bloques separados las direcciones pares e impares –bit menos significativo– o cualquier otro) o en datos (por ejemplo, se separa la palabra en dos partes y se implementa cada una de ellas en bloques de memoria separada). La primera opción produce mejores resultados en términos de consumo de potencia ya que se activa únicamente el bloque en el que se produce la lectura/escritura. Si generalizamos este concepto, desde el punto de vista de la optimización de potencia es más eficiente partir la memoria local y asignarla a los diferentes bloques de procesamiento. Esta decisión de diseño, especialmente para el caso de utilización de bloques de memoria RAM (BRAM) en las FPGAs puede generar implementaciones poco eficientes desde el punto de vista de los recursos.

Es posible reestructurar los *arrays* multidimensionales presentes en el modelo funcional para su implementación RTL, realizando una utilización eficiente de los recursos, a la vez que una reducción del consumo de potencia (figura 55). Con ello es posible adaptar el *array* al tamaño real del bloque disponible, ya sea para librerías ASICs o de FPGAs, en las que existen unos determinados módulos, con una relación de aspecto y tamaños definidos. Mediante esta operación, el diseñador puede acomodar el *array* original en una librería de IPs concreta.

En el diseño objeto de este trabajo se ha utilizado un modelo monolítico de memoria de tal manera que las estructuras de datos definidas por el diseñador se mapean en memoria de forma directa, concatenando los campos de las estructuras, para obtener memorias cuyo ancho de palabra es la suma de los campos componentes y el tamaño en palabras es equivalente al definido en el *array* cuyo tipo depende de la estructura definida. La alternativa es usar un modelo fragmentado por campos de la estructura de datos. Si bien esto facilita el acceso a cada componente de la estructura al estar ubicada en diferentes bloques de memoria, se incrementa de forma notable el correspondiente uso de recursos para el caso de la FPGA e incrementa la complejidad de gestión de los bloques de memoria en la implementación ASIC.

b) Impacto de la arquitectura de conexiones en la potencia

La utilización de buses locales dedicados entre bloques contribuye a la reducción del consumo de potencia ya que reduce el acceso a buses globales, con alto grado de actividad. El objetivo a conseguir consiste en añadir recursos para consumir menos potencia, hasta un cierto compromiso. Esto puede ir acompañado de técnicas de explotación de la regularidad (productor/consumidor) que simplifiquen el esquema de interconexión.

c) Impacto de la arquitectura de la ruta de datos en la potencia

Igualmente, la utilización de técnicas de segmentación en la ruta de datos y de aumento del paralelismo permite aumentar o disminuir la frecuencia de funcionamiento para obtener las prestaciones requeridas, con la correspondiente consecuencia en el consumo de potencia.

d) Guiado de la síntesis en la planificación: CtoS, Cynthesizer, Vivado

La síntesis de alto nivel da soporte a todas estas decisiones arquitecturales durante la etapa de planificación temporal o *scheduling.*

CtoS posee métodos para optimizar el diseño para bajo consumo de potencia usando la opción `-low_power` durante la fase de planificación `schedule -low_power`. En este modo el planificador realiza un análisis temporal multi-ciclo y evalúa las operaciones que están en la ruta crítica. Usando estos resultados, el planificador elige el mejor conjunto de recursos en términos de retardo, área y potencia. El análisis se repite en diferentes etapas del proceso de planificación. Aquellas operaciones no críticas se sintetizan e implementan usando recursos más eficientes en término de uso de recursos y potencia, para obtener menor área. Igualmente, en el caso de que aparezca *slack* negativo, este será más equilibrado o balanceado entre las etapas planificadas, y por tanto será de más fácil corrección.

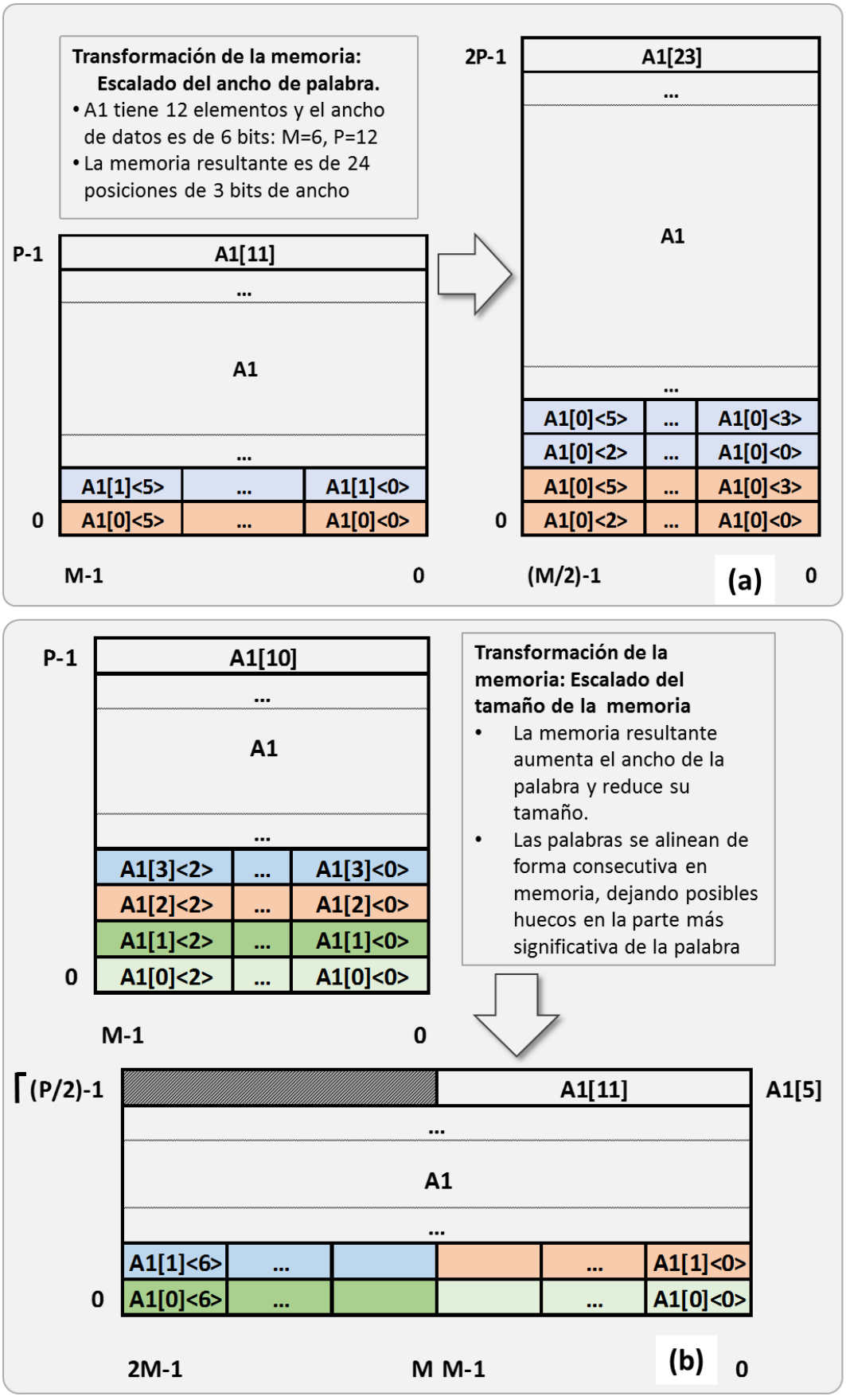


Figura 55. Modificación del factor de forma de los *arrays* para su implementación[81]

Se puede obtener un informe detallado del consumo de potencia estimado de todos los recursos del diseño mediante la orden `report_power` después de utilizar el comando `analyze -power` para anotar la información de potencia en la base de datos. Esta información puede obtenerse antes de la planificación del diseño o post-planificación, siendo más precisa en este último caso. Desde el punto de vista del flujo de diseño es preferible hacer una planificación con poco esfuerzo (`schedule_effort low`) y anotar los resultados obtenidos post-planificación.

El número de diferentes arquitecturas que es posible explorar está limitado principalmente por la duración del proyecto, siendo este un factor crítico que afecta a la calidad de los resultados. Por ello la síntesis de alto nivel ayuda a la toma de decisiones sobre la determinación de la arquitectura final. A partir de la decisión tomada, la síntesis de alto nivel proporciona un rápido camino hacia la implementación final. Como se ha indicado, la arquitectura de memoria posee un gran impacto en el consumo de potencia y por tanto es preciso darle prioridad en el proceso de diseño.

Cynthesizer [137] pone énfasis en tres aspectos de la optimización de potencia. Por una parte utiliza la técnica de *clock-gating* en los registros del diseño. Igualmente reduce el porcentaje de nodos que conmutan en las salidas de las máquinas de estado finito eligiendo la codificación apropiada, y por último, optimiza (reduce) el número de lecturas especulativas de memoria. Stratus [120] minimiza el consumo de potencia reordenando la planificación de operaciones.

En general, el consumo de potencia se incrementa con el uso agresivo de las técnicas de *pipelining* ya que incrementa la utilización de los recursos, pero con un comportamiento no lineal ya que el uso de los recursos en el caso de que no se use segmentación (puramente secuencial) es muy bajo y poco eficiente [138].

Vivado HLS [139] es un entorno de síntesis de alto nivel desde C/C+/SystemC integrado en el entorno Vivado de Xilinx [140]. Los criterios de optimización de potencia aplicados son de tipo genéricos, ya citados en el presente apartado.

3.11. Verificación RTL

Durante el proceso de síntesis de alto nivel el diseño se transforma desde una descripción algorítmica -con información temporal en las interfaces de comunicaciones y relajada en la de procesamiento y, en nuestro caso, con interfaces de datos abstractas- a una descripción RTL, constituida por bloques en HDL (VHDL/Verilog) e interfaces precisas a nivel de señales y de ciclos.

Durante el proceso de síntesis de alto nivel, con el objeto de cumplir las prestaciones requeridas, se toman decisiones que pueden afectar a la funcionalidad del diseño. Esta situación se presenta tanto en la planificación de las interfaces (mapeado o asignación de datos a señales, mapeado de estados, precisión del modelo de datos utilizado, mapeado de interfaces con memoria) como en la generación de la ruta de datos.

Otra situación que es normal en el flujo de diseño de un sistema complejo es realizar la partición del sistema en bloques manejables por el diseñador, creando la correspondiente jerarquía de diseño, sintetizando cada bloque por separado.

Las situaciones descritas requieren una coverificación del código RTL generado, con el objetivo de comprobar que el modelo continúa siendo funcionalmente equivalente al diseño original. Las descripciones en C, SystemC y RTL son ejecutables. Esto conduce a una verificación mediante simulación, que puede ser interactiva y cooperativa. Las ventajas de la cosimulación radican en que se generan transacciones desde el *testbench* original en alto nivel (C/C++/SystemC/SystemVerilog) hacia el modelo RTL encapsulado en un adaptador o *wrapper.* La comparación de las respuestas obtenidas con ambos modelos, funcional y RTL, permite observar las posibles diferencias de comportamiento entre ambos modelos. Por tanto, se utiliza el modelo funcional como modelo de referencia (*golden reference*).

Generalmente la arquitectura del *wrapper* se diseña para obtener la salida del módulo RTL (objeto de esta etapa de verificación) aplicando los mismos estímulos de entrada, mientras que las salidas del módulo de referencia se utilizan para realizar la comparación de dichas salidas. Igualmente es posible cosimular únicamente el módulo RTL utilizando el *testbench* original.

En CtoS se pueden encontrar ejemplos de utilización de este tipo de arquitectura de verificación. CtoS genera un *wrapper* en SystemC que instancia un modelo en Verilog durante la síntesis de alto nivel. Permite así realizar una simulación usando el mismo *testbench* en SystemC y RTL. Y, por otro lado, este *wrapper* realiza dos instancias del diseño en cuestión, una del modelo original en alto nivel en SystemC y otro del sintetizado en Verilog. Este mismo mecanismo de verificación también está implementado en Xilinx Vivado HLS.

La figura 56 muestra la jerarquía de bloques del *testbench* del sistema en alto nivel y la figura 57 representa el proceso de reutilización del *testbench* para simular el diseño sintetizado a nivel RTL. Además de la información funcional y dependiendo de las estrategias de diseño del *testbench*, durante la simulación RTL es necesario tener en cuenta las latencias de procesamiento obtenidas durante la síntesis de alto nivel de tal manera que el comportamiento sea concordante en ciclos de procesamiento.

Todo el proceso de verificación a diferentes niveles de abstracción se realiza en el entorno de simulación de Cadence Incisive Enterprise Simulator (IES). Al igual que la herramienta de síntesis, permite lanzarlo tanto en modo *batch* como en modo gráfico en el que se representan las señales como formas de onda y otras vistas gráficas y de depurado del diseño.

3.11.1. Diseño del *testbench*

Para el diseño del *testbench* del procesador de eventos cabe destacar que los datos de entrada se encuentran almacenados, individualmente en ficheros binarios, en el formato en el que son recibidos por la red. Por eso, teniendo en cuenta la necesidad de preparar las tramas para su posterior procesado (etapa de pre-procesado que en el sistema final será realizada por el procesador empotrado PowerPC 440 de la FPGA), esta tarea deberá realizarla el *testbench* antes de enviar los datos al CE.

Además de leer los ficheros binarios y realizar el pre-procesado de las tramas, el *testbench* debe implementar el protocolo de comunicación del bloque a simular, en este caso LocalLink. Esto implica que el *testbench* pasa de ser un simple generador de tramas, a ser un sistema que se realimenta del dispositivo bajo test, sincronizando las transmisiones cuando este deja de estar listo para la recepción de nuevos datos. Está formado por dos hilos de ejecución para modelar las interfaces LocalLink, proporcionando así una comunicación *full-duplex*. La figura 58 representa en un diagrama de flujo la ejecución del *testbench*.

3.11.2. Simulación del diseño

Para la simulación y consecuente verificación del diseño generador por CtoS se ha usado el *wrapper* que instancia el modelo original en alto nivel y el diseño sintetizado RTL. La estrategia de verificación implica los siguientes pasos. Se envía una regla de configuración, generándose notificaciones de creación de la regla en el sistema, y de las órdenes de ejecución que esta contiene. En la figura 59 se han introducido cinco marcadores, con nombres *TimeA*, *TimeB*, *TimeC*, *TimeD* y *TimeE*. El primero, marca el comienzo de transmisión de la regla, por parte del *testbench*, y con destino al modelo *hardware* bajo test. Los sufijos de *tx* y *rx* para denominar las señales de un sentido u otro de la comunicación han sido nombrados desde el punto de vista del bloque *hardware*, por lo que las señales con sufijo *rx* implican el sentido con origen el *testbench* (o el DMA en el sistema completo) y destino el procesador de eventos, y viceversa para las señales *tx*.

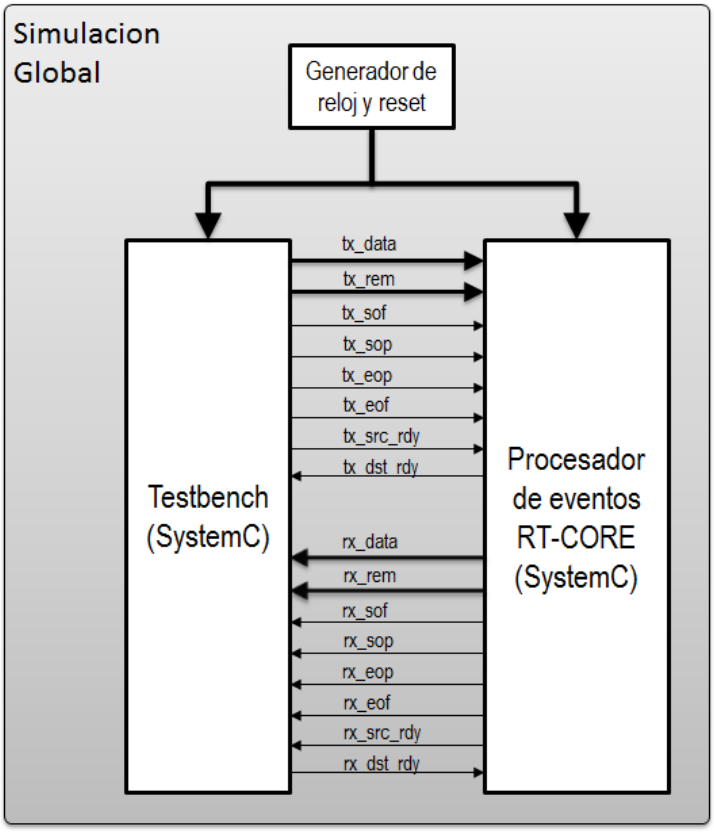


Figura 56. Simulación en alto nivel

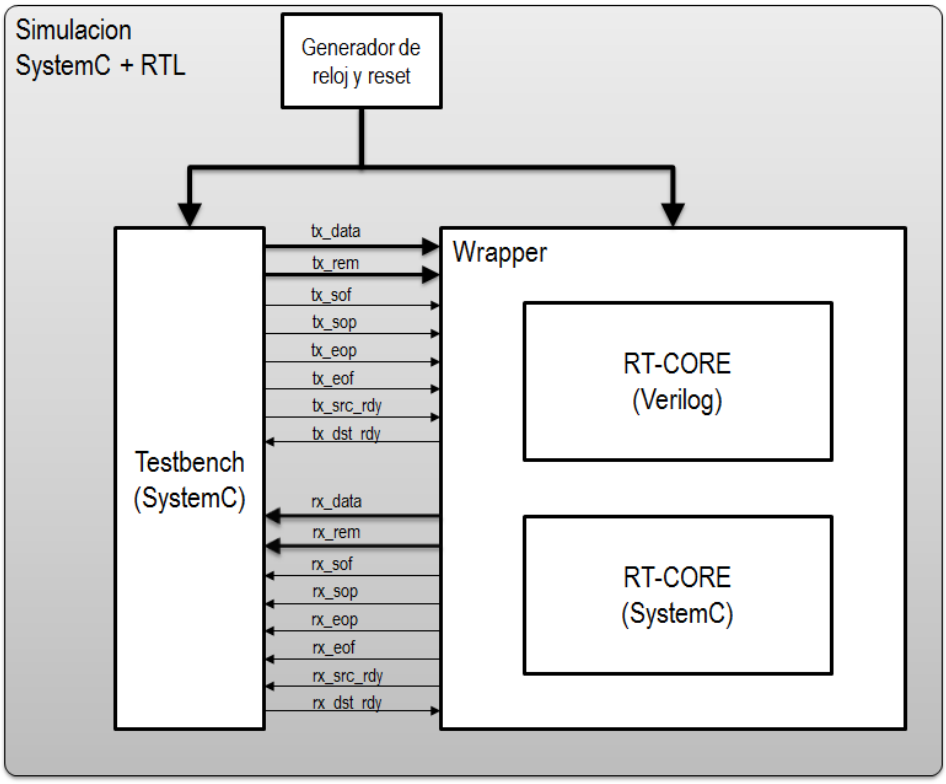


Figura 57. Esquema de simulación RTL usando el modelo SystemC como *Golden Reference*

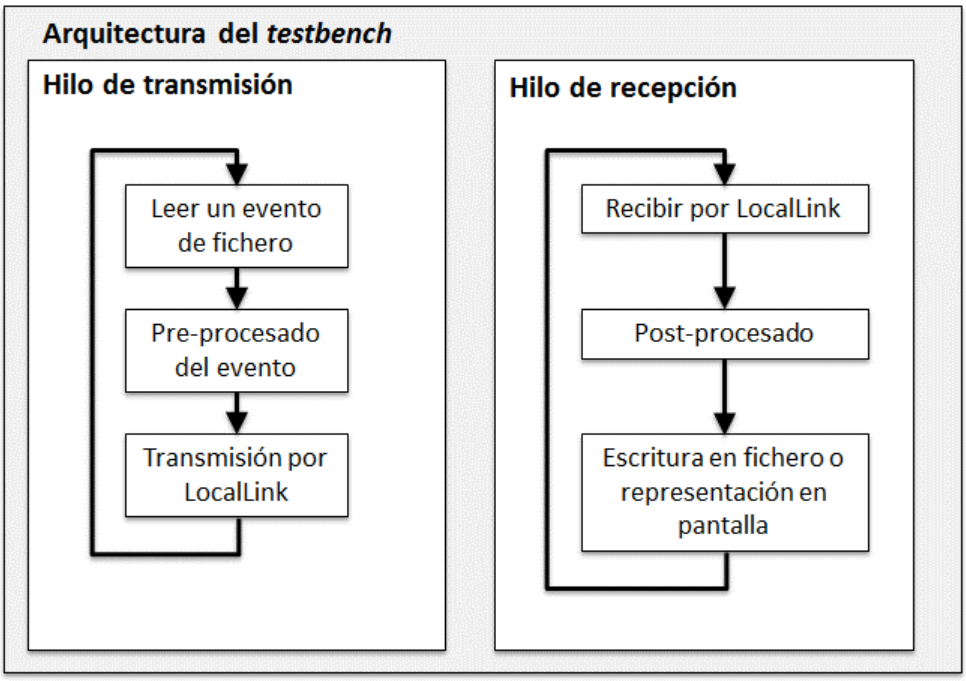


Figura 58. Arquitectura del *testbench* incluyendo la interfaz LocalLink

En las transmisiones, tanto de tramas de reglas como de eventos, se han simulado las interrupciones en la comunicación que se producirán en el sistema real, debido a la compartición de la matriz de interconexión interna del bloque de procesador, que impedirá que todos los

envíos se realicen en una sola ráfaga. Al comienzo de la transmisión de la trama de reglas, puede observarse que la señal *ll_src_rdy_rx_n* se desactiva durante unos instantes, para luego seguir por el mismo punto del paquete datos. Estas interrupciones deben ser previstas por el destino, poniéndose a la espera de que el origen vuelva a estar disponible, y seguir con la recepción de datos.

Puesto que el diseño en SystemC ya ha sido testado funcionalmente, y comprobado su correcto funcionamiento en una etapa anterior del flujo de diseño, la correlación –igualdad– entre las señales del modelo RTL de esta fase y el modelo SystemC de la fase anterior implican un correcto funcionamiento del modelo sintetizado.

Como ya se adelantaba, la relajación de latencia en las opciones de síntesis de CtoS provoca que, en ocasiones, los instantes temporales en los que ocurren ciertos eventos en el sistema, no sean los mismos en el diseño original y en el sintetizado. En el instante *TimeE*, por ejemplo, puede observarse que comienza la transmisión de una notificación (trama de salida del procesador de eventos), mediante la activación de la señal *ll_sof_tx_n* en el conjunto de señales del modelo en Verilog. Sin embargo, su señal correspondiente en el modelo SystemC ya había sido activada unos instantes antes. Es decir, el modelo en Verilog tiene una latencia mayor en su ejecución, lo que produce que se envíe más tarde la trama de salida.

Como es evidente, para la comprobación del correcto funcionamiento del modelo RTL, una vez comprobado de forma detallada el protocolo de comunicación, se realiza una simulación en modo *batch*. Para ello, el *testbench* se prepara para que, al igual que se hace con las tramas de entrada, se escriban en fichero todas las tramas de salida que genere el procesador de eventos. La detección de errores se realiza mediante aserciones, comparando los ficheros de salida generados por el modelo SystemC y el modelo RTL.

3.11.3. Interacción RTL-SystemC

Idealmente la herramienta de síntesis debe generar un diseño RTL que se comporte de forma equivalente al diseño de entrada en alto nivel. La naturaleza de nivel de abstracción más alto hace que durante el proceso de transformación se puedan realizar por parte del compilador diferentes interpretaciones, provocando discordancias entre los diseños. Algunos aspectos a tener en cuenta se describen a continuación a modo de reflexión para obtener mejores prácticas.

Uso de canales versus variables globales

Uno de los mecanismos de comunicación entre diferentes funciones en C es el uso de variables globales, que abstraen el concepto de memoria compartida en computación. Pero aunque las variables globales de C están soportadas por SystemC y también por la herramienta de síntesis CtoS, pueden provocar errores funcionales en el diseño RTL (Verilog) si se hace un uso incorrecto durante la fase de modelado.

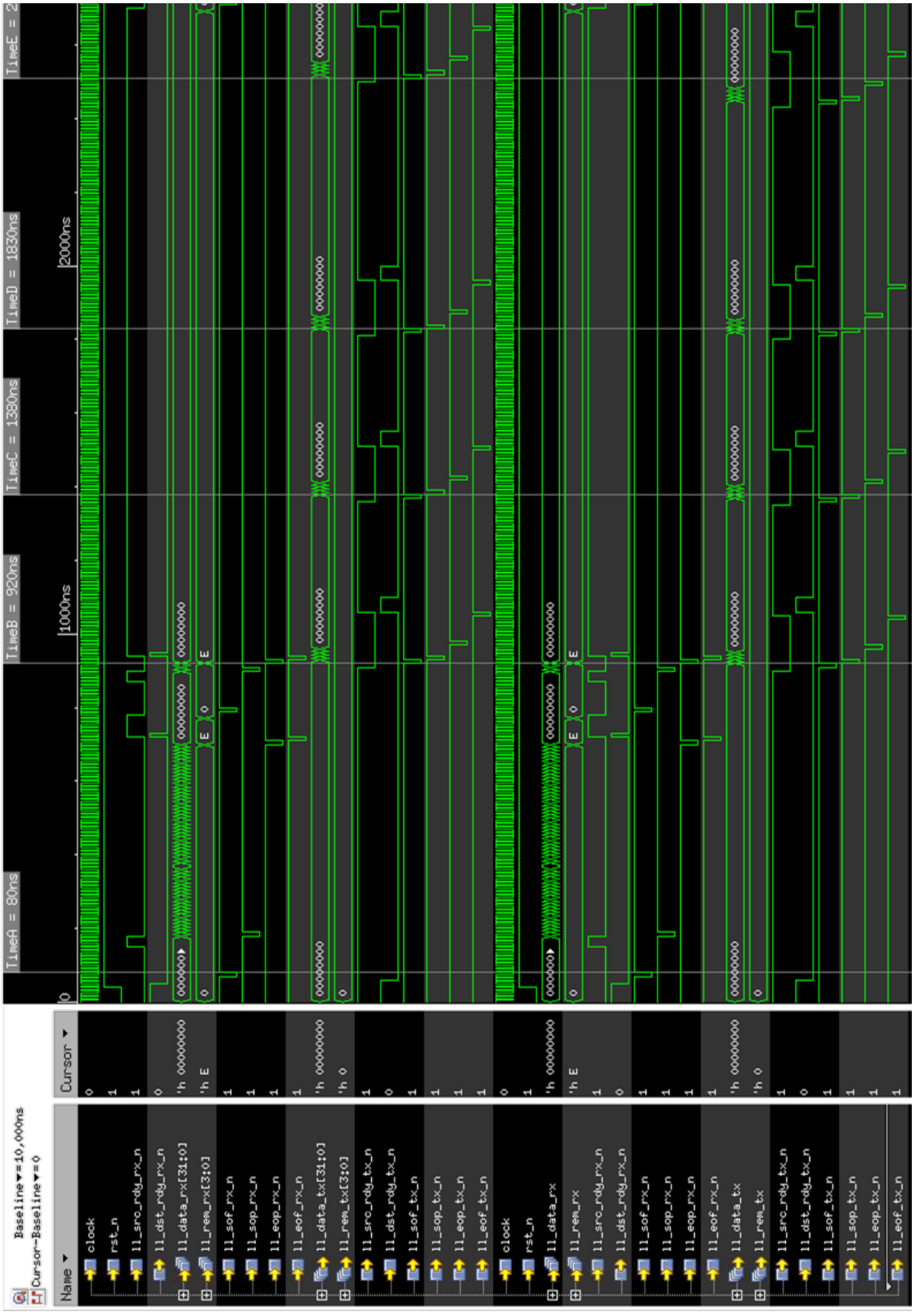


Figura 59. Simulación del diseño RTL y SystemC

Concretamente, el modelo de variables globales en SystemC y soportado por CtoS va destinado a su uso como medio de comunicar funciones de un mismo proceso, de forma que la variable representa un espacio de almacenamiento común tanto a la función principal ejecutada

por el hilo de ejecución como a las rutinas que sean llamadas desde esta. Por tanto no existen, no pueden existir, varias funciones escribiendo en la variable de forma concurrente ya que dentro de un proceso la ejecución es secuencial.

Sin embargo, en módulos en los que existan dos o más procesos o hilos (*SC_THREAD, SC_CTHREAD*), es necesario utilizar un canal SystemC, como por ejemplo una señal SystemC (`sc_signal`) del tipo de dato a comunicar. Una señal SystemC es un canal –primitiva del lenguaje– dedicado por tanto a definir protocolos de comunicación y sincronización de eventos. Otros tipos de canales simples que pueden ser utilizados en SystemC son: `sc_buffer`, `sc_fifo`, `sc_mutex`, `sc_semaphore`, `sc_signal_rv`. La utilización de métodos implementados en las correspondientes clases garantiza el acceso a los datos de forma ordenada, incluyendo los métodos `read()`, `write()` para interactuar con el canal. Se incluye también un conjunto de métodos de depurado para ver el estado del canal (`print()`, `dump()`, `trace()`).

Ancho de los datos

Al realizar asignaciones de datos de distinto ancho es importante que dichas asignaciones sean controladas de forma explícita por el diseñador, mediante los métodos de obtención del rango de los datos (`range()`). Con ello se evita la diferente interpretación que puede realizar el compilador de C/C++ con respecto a la herramientas de síntesis de alto nivel que puede provocar una diferencia de comportamiento entre ambas simulaciones.

Relajación de latencias durante la síntesis

Se ha mencionado con anterioridad que durante la fase de planificación es posible relajar latencias con el objetivo cumplir con las especificaciones temporales, e incluso mejorar algunas prestaciones del sistema, aumentando por ejemplo la frecuencia pero añadiendo ciclos de latencia de procesamiento. Es preciso que el diseñador establezca las zonas de mejora, respetando las interfaces de comunicación y el protocolo definido en los puertos de E/S, sobre todo cuando se ha realizado un modelado preciso a nivel de ciclos de bus. Por ejemplo, si se permite relajar la latencia, en una función de control de lectura o escritura de una FIFO, esto puede provocar que se inserten datos duplicados en la cola, o que se extraigan varios elementos, lo que corrompe la secuencia de escritura y lectura de los datos del sistema. Un caso similar se produce cuando se permite relajar la latencia en las funciones que gestionan el protocolo de comunicaciones LocalLink.

Inicialización de las memorias

Debido a la naturaleza del diseño es preciso realizar la inicialización de las memorias incluidas en los diferentes bloques del sistema. En SystemC este proceso se podría realizar en el constructor del módulo o mediante un bucle combinacional en el ciclo de *reset* del módulo. Si bien es una opción válida para la simulación, normalmente las herramientas de síntesis no tienen en cuenta las inicializaciones realizadas en la fase de elaboración del modelo. Y para la segunda alternativa, la presencia de un bucle combinacional durante el *reset* es inviable en un diseño síncrono. Existe sin embargo una tercera alternativa que consiste en ubicar la fase de inicialización de las memorias entre la fase de *reset* y el bucle infinito que modela el proceso de

tipo SC_THREAD o SC_CTHREAD. Esto consumirá ciclos de reloj para inicializar la memoria, que dependiendo de su tamaño produce entonces una latencia que pudiera ser aceptable o no para el diseño.

Para facilitar todo este proceso, se ha optado por delegar estas operaciones de inicialización a las herramientas de implementación incluyendo en el código RTL generado en Verilog sentencias de tipo `$readmemb`, que inicializan la memoria a partir de un fichero externo.

La integridad de este proceso se garantiza con la utilización de modelos precisos a nivel RTL de las memorias a sintetizar y con la creación de herramientas que complementan a las existentes para simulación y síntesis. Las herramientas parten de la información generada desde alto nivel y anotan la información necesaria en los modelos Verilog obtenidos durante la fase de síntesis de alto nivel. Para ello se asegura que las funciones que leen y escriben en memoria cumplen con las restricciones de latencia exigidas por los modelos RTL, que su vez están obtenidas a partir de la información de la librería de FPGAs utilizada. Al final del proceso todo el contenido de las memorias BRAM que se inferirán durante la fase de síntesis lógica e implementación se incluye en el fichero de programación, y por tanto se permite así inicializar correctamente las memorias. En la figura 60 se muestra la arquitectura usada para el módulo de memoria, incluyendo el controlador de lectura y escritura y el bloque de almacenamiento. Para el modelado de este último bloque se ha usado un *template C++* lo que ha facilitado la reutilización del código SystemC.

Para el caso de la implementación ASIC se genera un *wrapper* a nivel RTL que conecta los módulos de memoria a bloques disponibles en la librería ASIC.

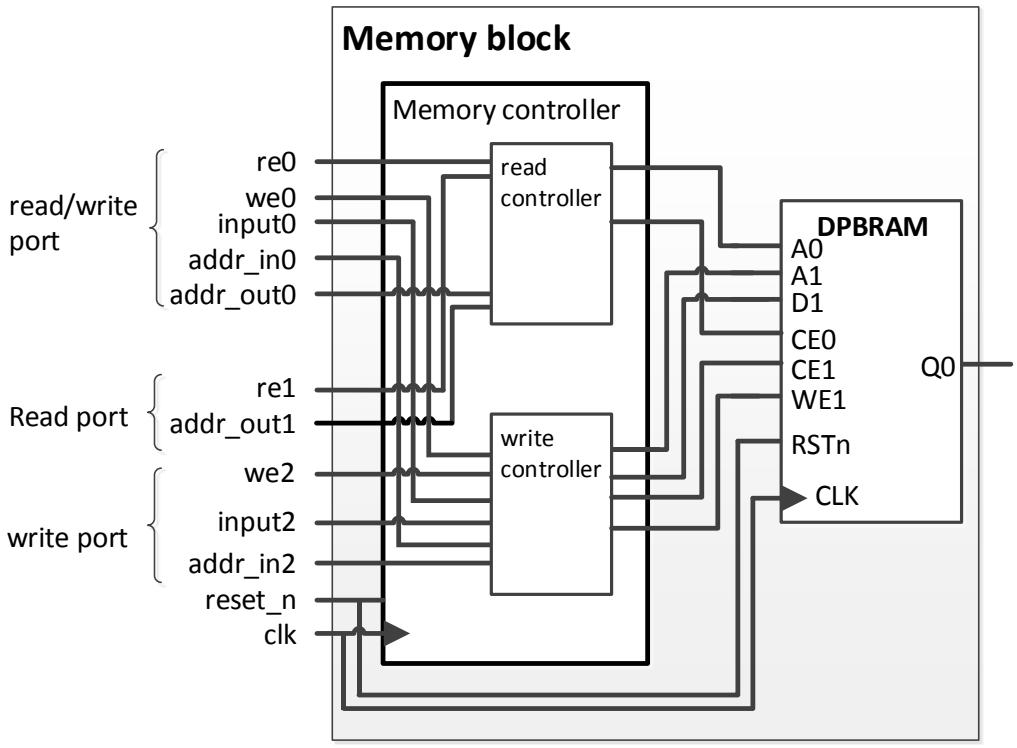


Figura 60. Diagrama de bloques de los módulos de memoria

3.12. Implementación FPGA

En esta sección se describe la implementación FPGA realizada.

3.12.1. Síntesis lógica

Para la realización de la síntesis lógica se parte de la descripción RTL obtenida y verificada siguiendo el proceso definido en el apartado anterior. Durante el proceso de síntesis se han seguido diferentes estrategias para la obtención del *netlist* en formato EDIF, listo para ser mapeado en la FPGA o ser enviado al flujo de *placement & routing* para el diseño ASIC.

3.12.2. Estrategias de síntesis lógica

Centrándonos en el caso de la implementación de la FPGA, se plantean dos estrategias posibles para la síntesis lógica: síntesis *bottom-up* y síntesis *top-down.* Durante el proceso de síntesis lógica se ha utilizado Synopsys Synplify Premier. Se han impuesto como restricciones del diseño aquellas obtenidas durante la fase de síntesis de alto nivel.

Estrategia *bottom-up*

La complejidad del diseño condiciona los tiempos de síntesis. Por ello es necesario adoptar estrategias *bottom-up* que reduzcan los tiempos de síntesis sin afectar a la calidad del diseño. Este tipo de estrategia ya había sido adoptada durante la etapa de síntesis de alto nivel. Esta estrategia está soportada mediante un sistema de gestión de proyectos y sub-proyectos con restricciones globales comunes: un proyecto para el *top* y en este incluir como sub-proyectos los proyectos anteriormente creados. Se realiza la síntesis de cada módulo por separado, facilitando la posibilidad de ejecutar múltiples síntesis, para luego realizar una optimización global de todo el diseño.

Estrategia *top-down*

Por otra parte es posible seguir una estrategia *top-down*, en la que se crea un único proyecto que incluye todos los modelos RTL de los módulos. Se genera la jerarquía del diseño mediante las instancias realizadas en los distintos módulos, sintetizando los que conformen el diseño completo. Los resultados obtenidos generalmente son mejores en términos de recursos y de frecuencia de funcionamiento debido a la mayor capacidad de optimización global.

Comparativa entre estrategias

Con el creciente incremento de la capacidad de las FPGAs los tiempos de síntesis e implementación han ido creciendo proporcionalmente [141], a pesar de los avances en la potencia de cálculo de los servidores de herramientas (figura 61). En aquellos casos donde el diseño es demasiado complejo una estrategia *top-down* puede ser inviable debido al tiempo necesario para realizar la síntesis o por los altos requerimientos de memoria del diseño.

En los flujos *bottom-up* cada sub-módulo se desarrolla mediante un flujo estándar de diseño, y luego se integra en un diseño final de módulos unidos. Durante la síntesis del sistema

completo es posible realizar las síntesis de varios de estos sub-módulos en paralelo, reduciendo así los tiempos de dicha etapa del flujo. Sin embargo esta estrategia puede llevar consigo un empeoramiento en la calidad de los resultados finales, puesto que la optimización de cada módulo tiene muchas restricciones.

La utilización de puntos de compilación intermedios da soporte a la síntesis incremental, creando particiones en el diseño de forma automática de tal manera que es posible reutilizar los resultados de síntesis de las particiones que no han sufrido cambios. Si además se activa el modo de multi-procesado cada partición o *Compile Point* es sintetizado en paralelo a los demás, reduciendo así los tiempos de síntesis. Como se puede apreciar en la figura 62 este mecanismo aumenta la calidad de los resultados obtenidos.

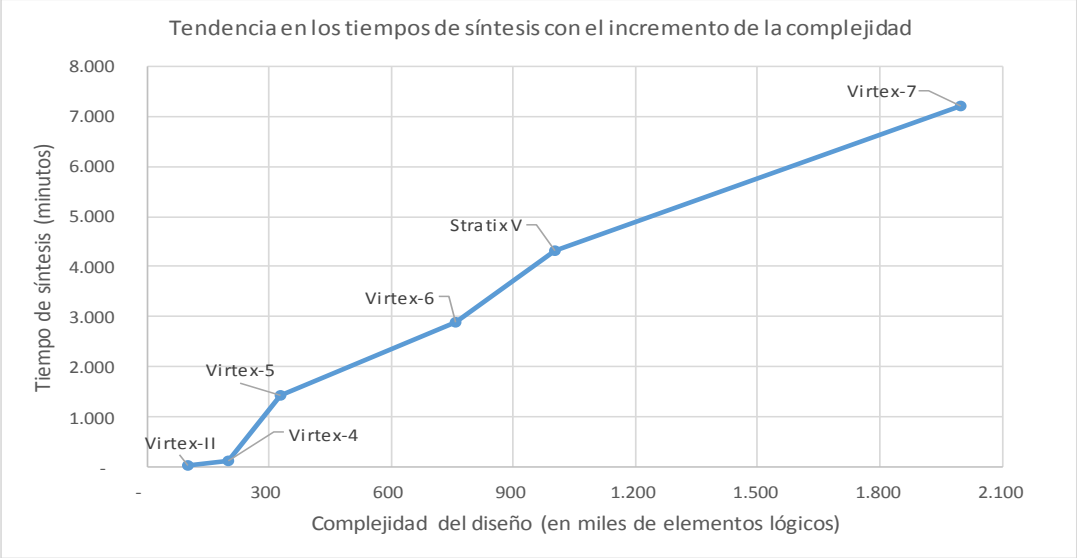


Figura 61. Tiempos de síntesis e implementación en familias de FPGA de Xilinx (derivada de [142])

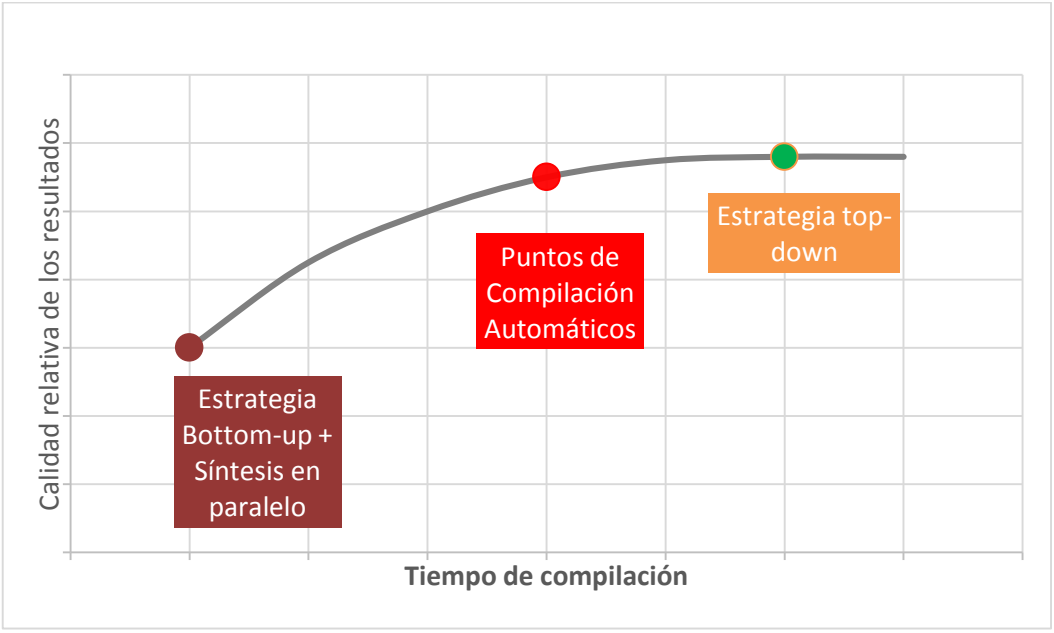


Figura 62. Calidad relativa de los resultados usando diferentes estrategias de síntesis

En el caso de la estrategia *bottom-up*, normalmente se realiza una optimización global tras la síntesis de todos los sub-módulos del diseño, incrementando ligeramente el tiempo de síntesis pero consiguiendo una mejora notable en la calidad de los resultados, ahorrando recursos innecesarios en los bordes de los bloques.

3.12.3. Opciones de síntesis

Con el objetivo de obtener un *netlist* optimizado para incorporarlo al diseño de la plataforma se incluyen diferentes opciones durante la fase de síntesis lógica del bloque IP.

Disable I/O Insertion

Normalmente durante la fase de síntesis lógica para la FPGA se insertan los bloques de entrada/salida (IOB) del diseño, correspondientes a las señales de entrada y salida en el bloque de mayor jerarquía del IP, entendiendo que dichas señales estarán asociadas a pines de la FPGA. En nuestro caso el diseño formará parte de una plataforma en la que se ubicarán las interfaces de E/S.

FSM Compiler

La utilización de un compilador simbólico de máquinas de estado (FSM) permite optimizar la codificación de los estados en términos de área y retardos. La elección de un esquema de codificación se realiza en función del número de estados y del impacto de la lógica de decodificación. Por ejemplo, una codificación binaria se utiliza cuando el número de estados es reducido (<5) o una codificación de Gray cuando el número de estados es alto (>25). Normalmente, para el caso de FPGAs, la utilización de codificación *one-hot* es la alternativa adecuada para la mayoría de los casos no contemplados entre los extremos indicados anteriormente.

Resource Sharing

Se habilita la opción de compartición de recursos de procesamiento, con el fin de reducir el consumo de recursos en el dispositivo, para aquellos casos en los que es posible hacerlo porque su uso es mutualmente excluyente como por ejemplo en una sentencia **case.** Esto implica la introducción de multiplexores en las entradas y salidas, incrementando los retardos en la lógica. Otro efecto es el incremento en el *fan-out* de los puertos de salida. Si los recursos están en la ruta crítica del diseño es preferible no compartir recursos y evitar así retardos adicionales.

Pipelining

Durante el proceso de optimización de la síntesis lógica es posible segmentar la lógica para incrementar las prestaciones del sistema, permitiendo que se estén procesando diferentes datos en diferentes etapas de la unidad de procesamiento. Esta segmentación es posible bajo determinadas condiciones, compartiendo señales de reloj, de *reset* y de *enable*. También es posible segmentar las propias unidades funcionales (multiplicadores, memorias, DSPs).

Retiming

El proceso de *retiming* está orientado a la optimización temporal y también a la reducción del uso de recursos. La aplicación de *retiming* implica el movimiento legal de registros y lógica sin alterar el comportamiento temporal del sistema en cuanto al número de ciclos totales. Dado que el tiempo de ciclo viene definido por la etapa lógica más lenta del sistema, el *retiming* tiene por objetivo equilibrar las etapas lógicas, reduciendo el tiempo de ciclo y por tanto reduciendo la latencia global del sistema. Igualmente es posible aplicar técnicas de *retiming* para mover registros desde las entradas a las salidas y por tanto reducir, en algunos casos, la utilización de registros.

3.12.4. Resultados

En este apartado se presenta la comparación ente los resultados de síntesis, en términos de consumo de recursos y frecuencia máxima de utilización, para las estrategias de síntesis *bottom-up* y *top-down*.

Los resultados, en los casos de estrategia *bottom-up*, serán además presentados desglosados para los módulos de la partición realizada del diseño: interfaz, estado del sistema, núcleo de procesado y ejecutor de resultados. La interfaz incluye la funcionalidad de control del protocolo LocalLink y las FIFOs de comunicación. El módulo de estado del sistema comprende todas las memorias internas que almacenan las reglas activas y por activar, que actualmente ha recibido el sistema, así como el bloque encargado de reconocer una trama de reglas recibida por LocalLink y mapearla en las memorias internas. El núcleo de procesado es el encargado de comprobar -una vez recibido un evento- si los valores recibidos están asociados a una regla activa y realizar las operaciones necesarias. En caso de que se cumpla la condición, el ejecutor de resultados realizará las tareas asociadas a dicha regla.

En la tabla 4 y en la figura 63 se muestran los resultados obtenidos durante la fase de síntesis lógica utilizando una estrategia *bottom-up.* La mayor parte de lógica programable, como son las LUTs y los FlipFlops se utiliza en los bloques del núcleo de procesamiento y el ejecutor de resultados. Las LUTs y los FlipFlops del bloque de estado son los usados por el módulo responsable de actualizar el estado del procesador cuando este recibe nuevas estrategias para su preparación y almacenamiento en memoria.

Tabla 4. Resultado de la síntesis lógica con estrategia *bottom-up*

Bloque	Ruta crítica (ns)	Frecuencia (MHz)	LUTs	DSPs	FlipFlops	BRAM
Interfaz	7,6	131,8	3.218	0	1.477	0
Estado	5,7	176,2	9.890	0	14.223	53
Núcleo	5,4	183,6	10.120	4	18.800	0
Ejecutor	6,8	147,5	11.544	24	14.484	0
Total agregado	**7,6**	**131,8**	**35.772**	**28**	**48.984**	**53**
Sistema optimizado	**6,7**	**149,5**	**35.772**	**28**	**36.922**	**53**

Los bloques DSPs son usados únicamente por el núcleo de procesamiento y el ejecutor de resultados, ya que realizan operaciones con números reales. A su vez los bloques BRAMs están concentradas en el módulo de estado del sistema ya que este se encarga de almacenar las estrategias, acciones e histórico de cotizaciones. Igualmente el módulo de interfaz hace uso de LUTs para implementar RAM distribuida haciendo uso de los SLICEM de la FPGA.

En cuanto a la frecuencia máxima de funcionamiento, todo el diseño se ha referenciado a un único reloj de entrada para el IP. La frecuencia de funcionamiento del IP acelerador vendrá por tanto, determinada por la ruta crítica del sistema, que en este caso se encuentra en la interfaz de entrada/salida (figura 64). Esto se debe a la funcionalidad incluida en dicho módulo de desempaquetado de datos que afecta a la ruta crítica del sistema. Aunque parezca que interfiere en las prestaciones finales del sistema, la ventaja de recibir los datos empaquetados es que disminuye la latencia debida a la transferencia de datos y por tanto en realidad mejora las prestaciones globales del sistema.

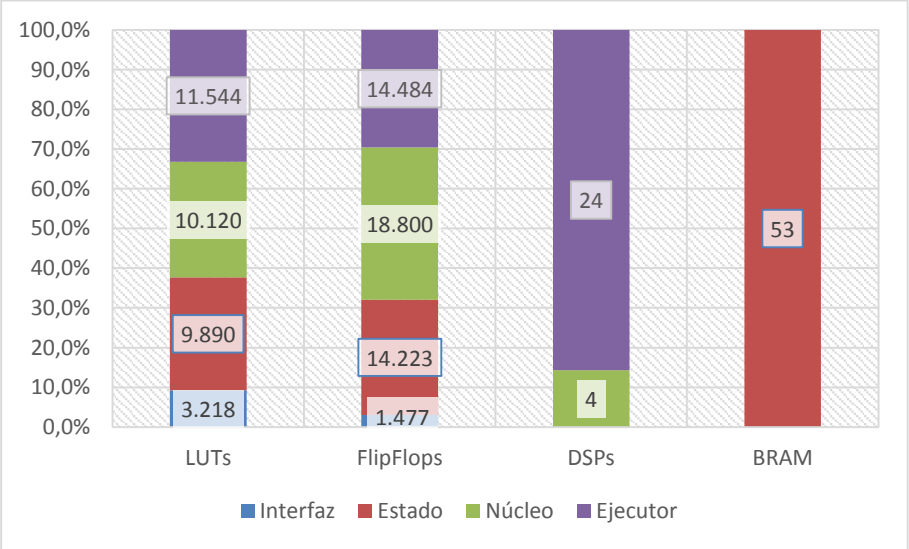


Figura 63. Distribución de recursos utilizados en cada módulo del diseño

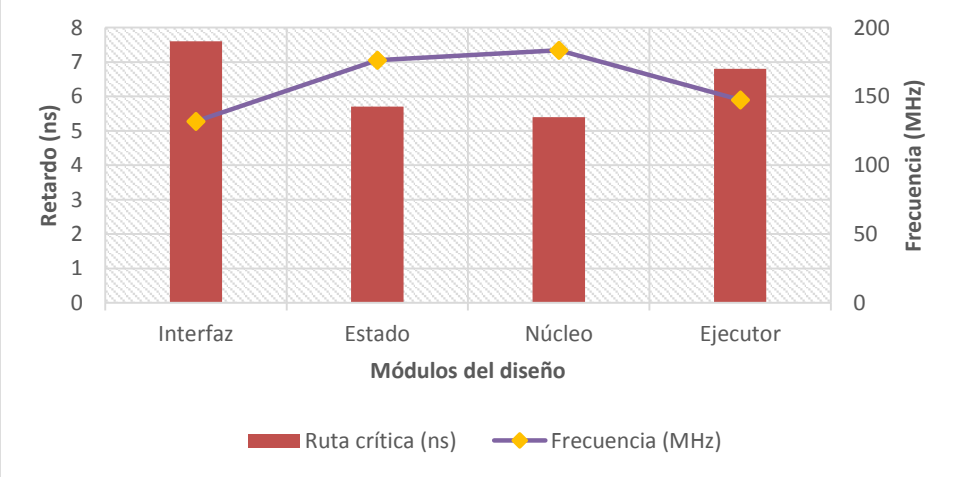


Figura 64. Frecuencia máxima de cada bloque

Igualmente se muestran los resultados de la síntesis lógica realizada con estrategia *top-down*, en la que se muestran las esperadas mejoras en cuestión de consumo de recursos arquitecturales frente a la síntesis *bottom-up* (tabla 5 y figura 65).

Tabla 5. Resultados de síntesis con estrategias *bottom*-up y *top-down*

Estrategia	Ruta crítica (ns)	Frecuencia (MHz)	LUTs	DSPs	FlipFlops	BRAM
bottom-up	6,688	149,5	35.772	28	36.922	53
top-down	6,888	145,2	30.708	28	32.291	46
Diferencia	-3,0%	-2,9%	14,2%	0,0%	12,5%	13,2%

Puede observarse que, menos en el caso de los bloques DSPs, el resto de resultados varían con la estrategia *top-down*, mejorando en el consumo de recursos, pero empeorando en la frecuencia máxima del diseño.

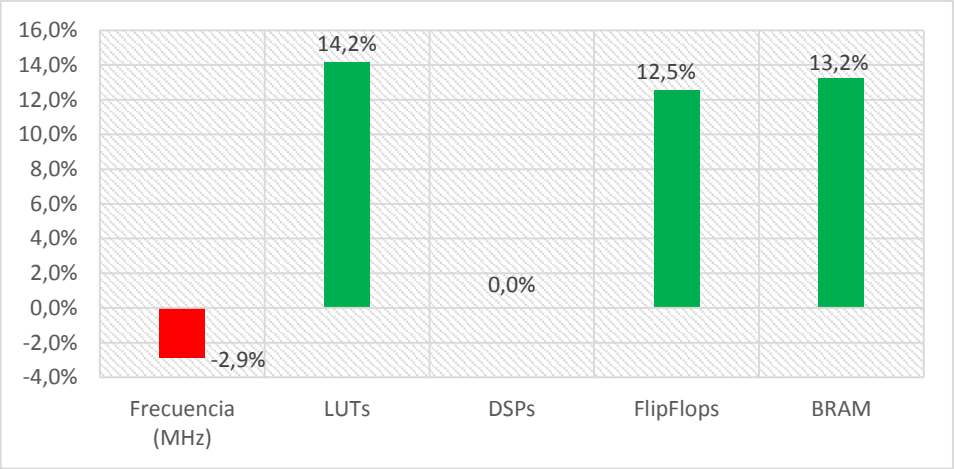


Figura 65.Comparación entre los resultados de las dos estrategias utilizadas

3.13. Implementación física y validación de la plataforma

El flujo de diseño establecido prevé como siguientes etapas realizar la implementación física integrando el diseño en la plataforma, y entonces validar la implementación completa en la plataforma. Para validar los resultados obtenidos en las etapas de síntesis se realiza en primer lugar la integración del acelerador *hardware* en la plataforma y luego la implementación final de ésta. Con ello se genera un fichero de configuración para la programación de la FPGA. Igualmente es preciso generar el *software* empotrado para la generación y envío de tramas de validación a través de la interfaz de red Ethernet.

3.13.1. Plataforma de validación

Para la implementación física del procesador y para su validación se ha utilizado la plataforma mostrada en la figura 66. El sistema se organiza alrededor del bloque empotrado que incluye el PowerPC 440 y un conjunto de elementos de interconexión para crear la infraestructura necesaria (interfaces de bus PLB maestra y esclavas, interfaces LocalLink con sus

correspondientes bloques DMA, así como otras interfaces para el controlador de memoria y otras interfaces de control. Asimismo el sistema incluye un conjunto de IPs interconectados mediante el bus CoreConnect PLB así como interfaces LocalLink dedicadas (figura 67). La plataforma incluye un controlador de acceso a memoria DDR, un controlador TEMAC para acceso Ethernet, una UART, un controlador de interrupciones, así como un controlador para acceder a la memoria CompactFlash. Asimismo incluye un bloque ILA/ICON para el depurado de la plataforma mediante un analizador lógico integrado. El acelerador *hardware* se conecta al PowerPC mediante un enlace punto a punto *full-duplex* LocalLink que permite las transferencias de datos a través de DMAs dedicados.

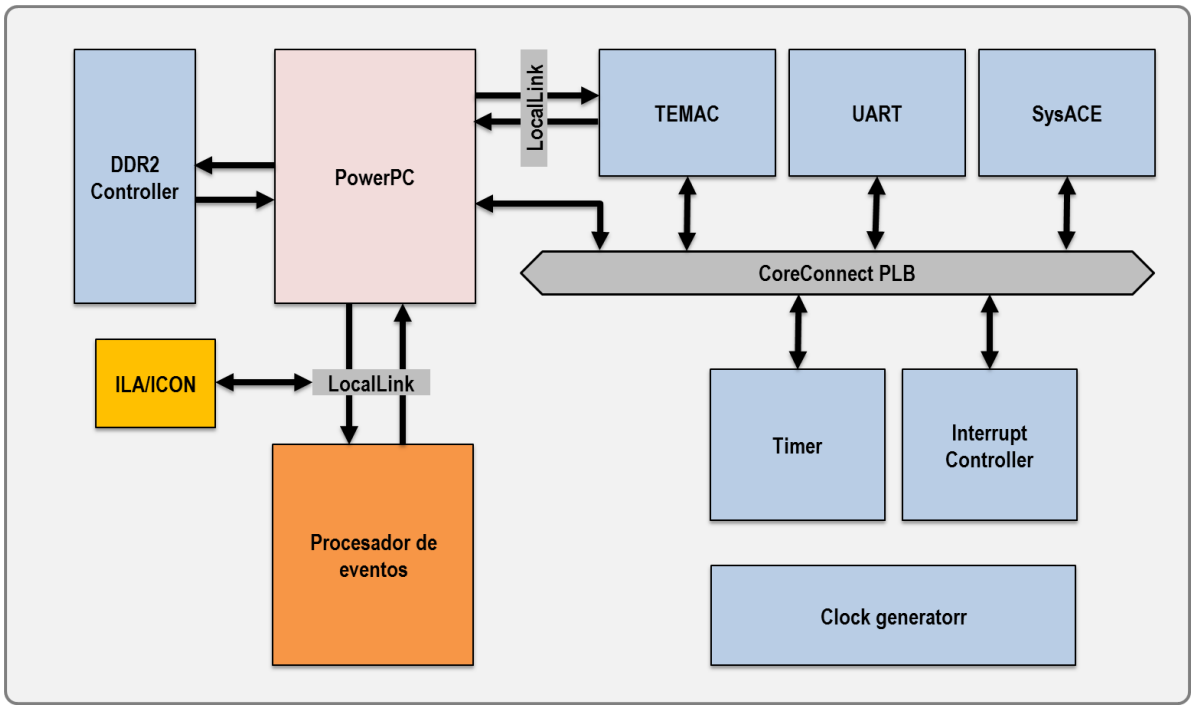


Figura 66. Plataforma de referencia para la validación del acelerador *hardware*

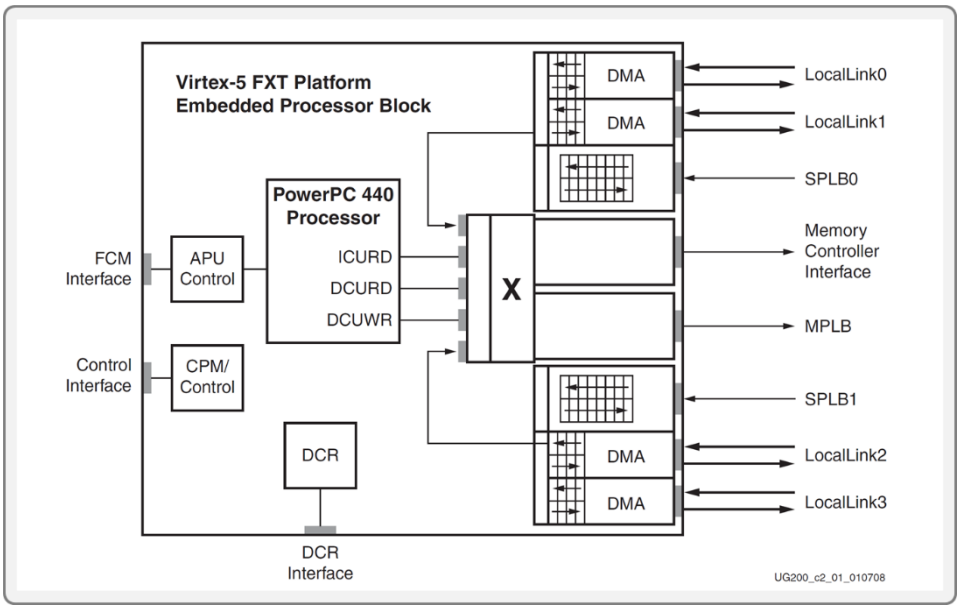


Figura 67. Bloque procesador empotrado Virtex-5 PowerPC 440 [143]

La integración del procesador de eventos en la plataforma exige la creación de un *wrapper* VHDL para conectar el *netlist* EDIF generado en Synplify durante la fase de síntesis lógica que será tratada como una *blackbox*. A continuación se muestra el *wrapper* VHDL utilizado.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity rt_core is
    port(
        ll_clk           : in  std_logic;
        ll_rst           : in  std_logic;
        tx_data          : in  std_logic_vector(31 downto 0);
        tx_rem           : in  std_logic_vector(3 downto 0);
        tx_sof_n         : in  std_logic;
        tx_eof_n         : in  std_logic;
        tx_sop_n         : in  std_logic;
        tx_eop_n         : in  std_logic;
        tx_src_rdy_n     : in  std_logic;
        tx_dst_rdy_n     : out std_logic;
        rx_data          : out std_logic_vector(31 downto 0);
        rx_rem           : out std_logic_vector(3 downto 0);
        rx_sof_n         : out std_logic;
        rx_eof_n         : out std_logic;
        rx_sop_n         : out std_logic;
        rx_eop_n         : out std_logic;
        rx_src_rdy_n     : out std_logic;
        rx_dst_rdy_n     : in  std_logic
    );
end rt_core;

architecture arch of rt_core is
  component procesador_rtl
    port(
        clock                : in  std_logic;
        rst_n                : in  std_logic;
        ll_src_rdy_rx_n      : in  std_logic;
        ll_dst_rdy_rx_n      : out std_logic;
        ll_sof_rx_n          : in  std_logic;
        ll_sop_rx_n          : in  std_logic;
        ll_data_rx           : in  std_logic_vector(31 downto 0);
        ll_rem_rx            : in  std_logic_vector(3 downto 0);
        ll_eop_rx_n          : in  std_logic;
        ll_eof_rx_n          : in  std_logic;
        ll_src_rdy_tx_n      : out std_logic;
        ll_dst_rdy_tx_n      : in  std_logic;
        ll_sof_tx_n          : out std_logic;
        ll_sop_tx_n          : out std_logic;
        ll_data_tx           : out std_logic_vector(31 downto 0);
        ll_rem_tx            : out std_logic_vector(3 downto 0);
        ll_eop_tx_n          : out std_logic;
        ll_eof_tx_n          : out std_logic
    );
  end component;
begin
 procesador_inst : procesador_rtl port map (
              clock  => ll_clk,
              rst_n  => ll_rst,
```

```
      ll_src_rdy_rx_n => tx_src_rdy_n,
      ll_dst_rdy_rx_n => tx_dst_rdy_n,
          ll_sof_rx_n => tx_sof_n,
          ll_sop_rx_n => tx_sop_n,
           ll_data_rx => tx_data,
            ll_rem_rx => tx_rem,
          ll_eop_rx_n => tx_eop_n,
          ll_eof_rx_n => tx_eof_n,
      ll_src_rdy_tx_n => rx_src_rdy_n,
      ll_dst_rdy_tx_n => rx_dst_rdy_n,
          ll_sof_tx_n => rx_sof_n,
          ll_sop_tx_n => rx_sop_n,
           ll_data_tx => rx_data,
            ll_rem_tx => rx_rem,
          ll_eop_tx_n => rx_eop_n,
          ll_eof_tx_n => rx_eof_n);
end arch;
```

Para realizar la implementación física del diseño, y a la vista del elevado número de recursos utilizados de la FPGA, se han utilizado distintas estrategias con el objetivo de obtener una implementación válida que cumpla con los objetivos temporales.

Se parte de un *netlist* completo de la plataforma creado a partir de sus bloques IPs componentes en sus formatos nativos (EDIF, NGC, etc.) y de la definición de las interconexiones de los mismos obtenidos en Xilinx Platform Studio (XPS).

PlanAhead soporta la ejecución paralela (en diferentes servidores de cómputo) con diferentes estrategias, con y sin optimización global, para luego optar por aquella válida con un mejor análisis temporal. En la figura 68 se muestran distintas estrategias disponibles en PlanAhead, algunas de ellas optimizadas en área mientras que otras están diseñadas para la optimización temporal.

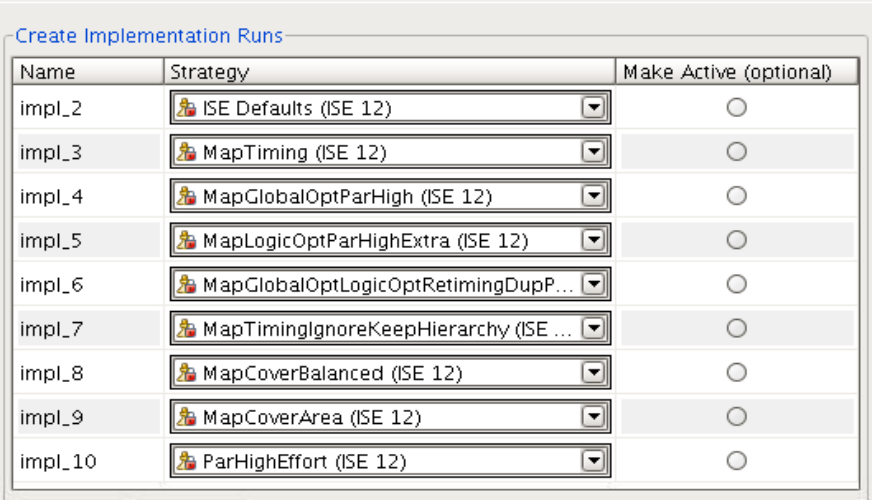


Figura 68. Estrategias de las implementaciones

En la figura 69, se observa el resultado de los procesos de implementación planificados con diferentes estrategias, en total nueve. De ellas se ha finalizado una completamente, en concreto

la impl_10, que aplica un esfuerzo alto de colocado y ruteado. Dos de ellas (impl_3 e impl_5) han completado las fase de colocado y ruteado aplicando igualmente esfuerzos elevados de optimización temporal guiada por prestaciones temporales (en impl_3) y de mapeado con un esfuerzo adicional (en impl_4). La impl_7 y la impl_8 han fallado durante la fase de mapeado y el resto no ha completado esta fase de diseño, lo que suele indicar que no se encontrará una solución válida. Como se ve, la utilización de diferentes estrategias de implementación es esencial y permite encontrar soluciones aceptables desde el punto de vista de ocupación de la FPGA y de cumplimiento de los requisitos temporales.

En cuanto a los resultados finales obtenidos, se resume en la tabla 6 y figura 70, en términos de recursos utilizados, la utilización de los bloques de la FPGA y la frecuencia de funcionamiento de la FPGA y su ruta crítica.

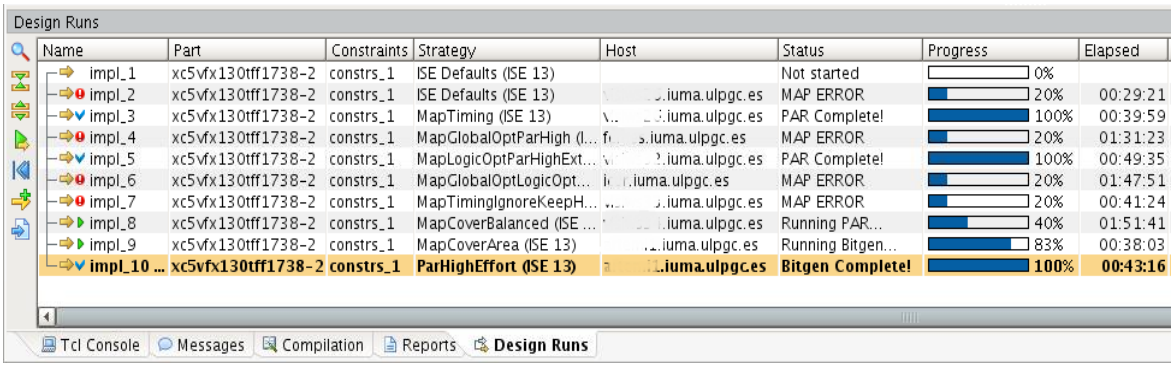


Figura 69. Estrategias de implementación utilizadas

Tabla 6. Utilización de recursos

Recursos	BRAMs	DSPs	LUTs	FlipFlops	Slices
Utilizados	62	28	38.777	52.982	**18.562**
Totales	298	320	81.920	81.920	**20.480**
Porcentaje de ocupación	21%	9%	47%	65%	**91%**

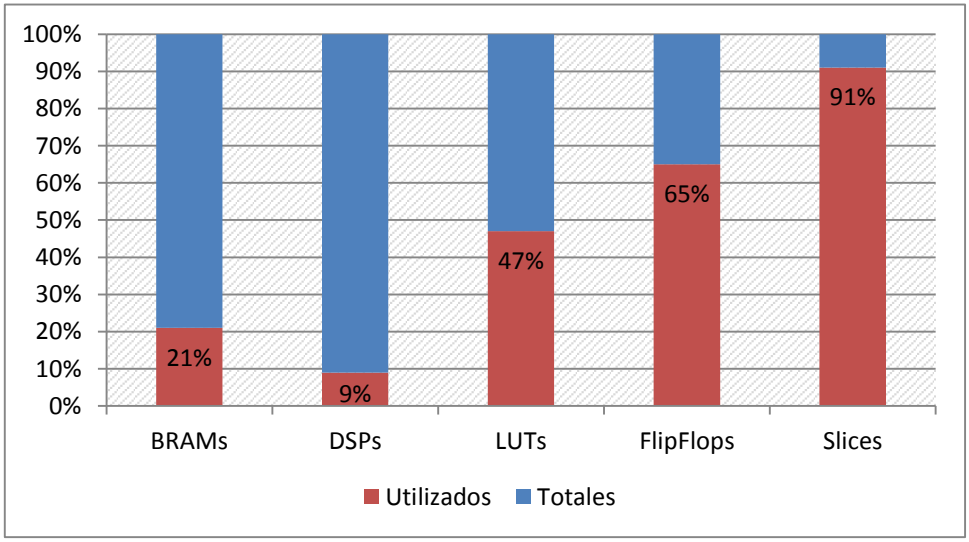


Figura 70. Ocupación de la FPGA Virtex-5 FX 130T

Para hacer un análisis del área ocupada por el diseño y de su complejidad, se debe observar el número de *Slices* utilizados, ya que el *Slice* es la unidad básica de programabilidad de las FPGAs de la familia Virtex-5 de Xilinx.

En la arquitectura Virtex-5 cada *Slice* está formado por cuatro *LUTs* y cuatro *FlipFlops (FF)*, por lo que dado un número de *LUTs* y *FlipFlops* usados, el factor de utilización (FU) de los *slices* se puede calcular, tanto para LUTS como para FFs, como:

$$FU_{LUTs} = \frac{N_{LUTS}}{N_{SLICES}} = 2{,}85 \qquad FU_{FF} = \frac{N_{FF}}{N_{SLICES}} = 2{,}09$$

Los factores de utilización indicados dependen de la naturaleza del diseño y del estilo de diseño utilizado. En la FPGA para poder incrementar la utilización de *slices* es necesario que compartan las señales de control. La utilización de *resets* globales puede aumentar del orden del 10% el factor de utilización, aunque disminuye el control sobre el diseño durante su simulación. Este proceso de empaquetado de señales de control está influenciado por las restricciones temporales impuestas al diseño. La tabla 7 resume los resultados del factor de utilización para diferentes diseños realizados siguiendo el flujo de diseño mostrado y comparándolo con flujos de diseño basados en Vivado HLS.

Tabla 7. Factor de utilización de los *slices*

	Tipo diseño	**Herramienta**	**FPGA**	**Factor de Utilización**	
				LUTS	***FFs***
Diseño 1	Plataforma + IP	Vivado HLS	Xilinx Zynq 7z020	2,21	2,90
Diseño 2	IP	CtoS	Xilinx Zynq 7z045	3,32	1,64
Diseño 3	Plataforma + IP	Vivado HLS	Xilinx Zynq 7z045	2,29	3,05
Diseño 4	IP	Vivado HLS	Xilinx Zynq 7z045	2,12	2,79
Este diseño	Plataforma + IP	CtoS	Xilinx Virtex-5 FX 130t	2,85	2,09

En la figura 71 se muestra el *layout* final del diseño. En a) se presentan resaltados los principales bloques de la plataforma, mostrando el acelerador *hardware*, el controlador DDR2, el adaptador del bloque TEMAC y el procesador empotrado PowerPC 440. En b) se muestra la distribución de los bloques que componen el acelerador *hardware*, y que incluye los bloques BRAM, organizados en columnas, el bloque de estado del procesador de eventos, y el bloque de interfaz, más próximo al PowerPC. Igualmente se incluye el bloque de procesamiento y el bloque ejecutor de resultados.

En cuanto al análisis temporal estático realizado después de la implementación se ha comprobado que el sistema cumple las restricciones temporales para una frecuencia a 100 MHz para el acelerador *hardware*, tal como se especifica en el fichero de restricciones temporales. Los resultados obtenidos son los siguientes:

- Periodo mínimo: 9,996 ns
- Máxima frecuencia: 100,040 MHz

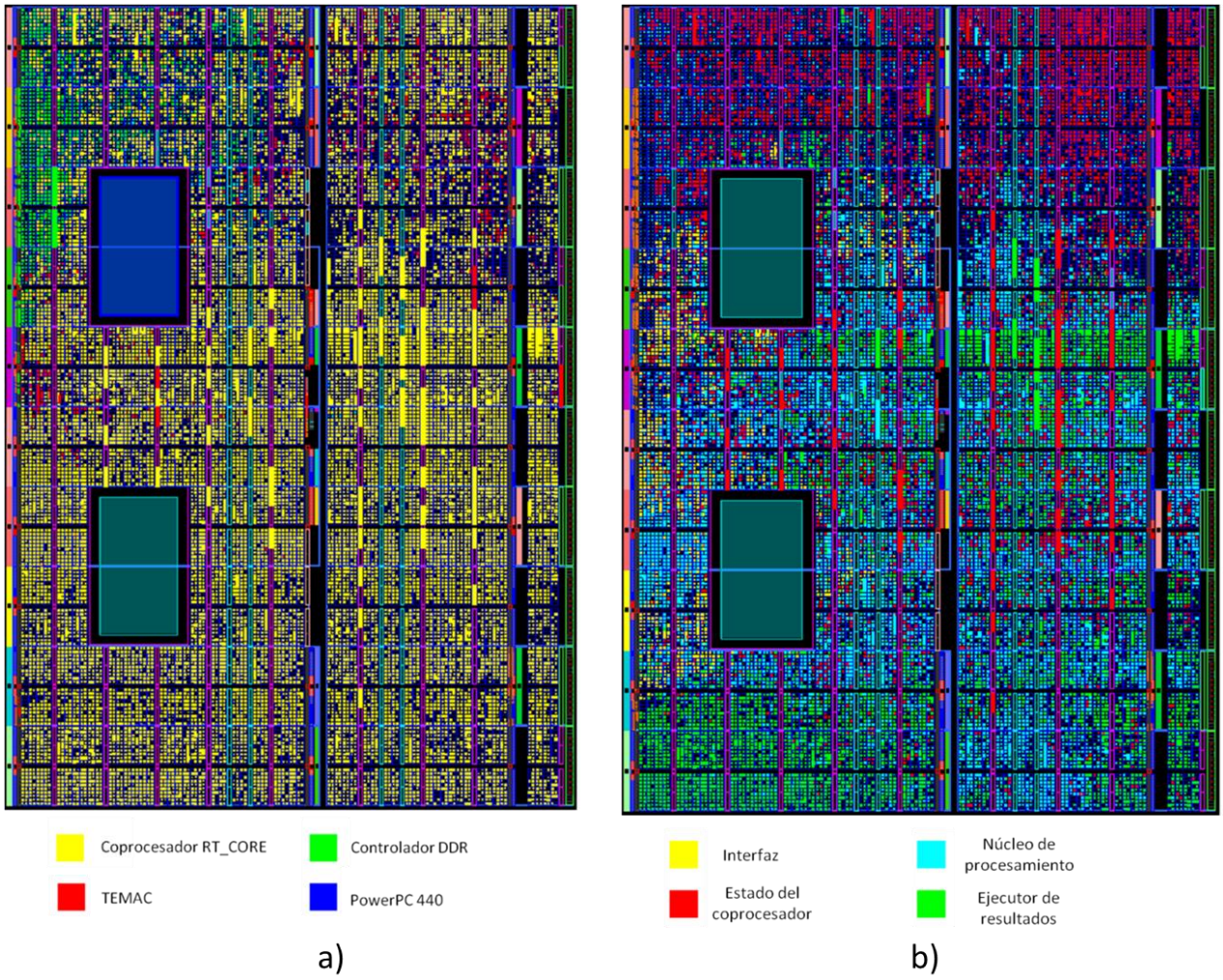


Figura 71. *Layout* final del diseño: a) bloques de la plataforma. b) bloques del acelerador *hardware*

En el informe obtenido durante el análisis temporal se presentan las rutas con los correspondientes *slack* para cada reloj del sistema. A continuación se muestra el resumen de una ruta con su correspondiente *slack* del procesador de eventos.

```
Slack: 0.024ns
  (requirement - (data path - clock path skew + uncertainty))
  Source:  .../state_leer_acciones_0_rep1 (FF)
  Destination: .../mux_limite_m_mant_ln392_Z_0_0[6] (FF)
  Requirement:          10.000ns
  Data Path Delay:      9.896ns (Levels of Logic = 3)
  Clock Path Skew:      0.001ns (1.241 - 1.240)
  Source Clock:         ll_clk_100MHz rising at 0.000ns
  Destination Clock:    ll_clk_100MHz rising at 10.000ns
  Clock Uncertainty:    0.081ns
  Clock Uncertainty:    0.081ns  ((TSJ^2 + DJ^2)^1/2) / 2 + PE
    Total System Jitter (TSJ):  0.070ns
    Discrete Jitter (DJ):       0.146ns
    Phase Error (PE):           0.000ns
```

```
  Maximum Data Path: .../state_leer_acciones_0_rep1 to
.../mux_limite_m_mant_ln392_Z_0_0[6]
Location          Delay type       Delay(ns)
  Physical Resource
  Logical Resource(s)
-------------------------------------------------------------------
SLICE_X61Y114.BQ  Tcko             0.375
  .../read_estado_in_Z_0_0[7]
  .../state_leer_acciones_0_rep1
SLICE_X73Y133.B2  net (fanout=36)  4.031
  .../state_leer_acciones_0_rep1
SLICE_X73Y133.B   Tilo             0.086
  .../leer_comandos_ln2023_...TipoV_posVariable_out_0_0[11]
  .../driver_sib_tx...binational.ctrlOr_ln266_w_i_o2
SLICE_X63Y138.B   Tilo              0.086
  .../n_combinat...merged_I_mux_eq_ln539_Z_0_0_i_a2_1_4[29]/O
  .../mbinational.merged_I_mux_eq_ln539_Z_0_0_i_a2_1[29]
...
-------------------------------------------------------------------
Total                     9.896ns (0.827ns logic, 9.069ns route)
                                  (8.4% logic,    91.6% route)
```

3.13.2. Validación

Finalmente se valida el diseño mediante un generador de tramas de datos que se inyectan a través del puerto Ethernet disponible en la placa de prototipado Xilinx ML510. El generador de tramas extrae las tramas capturadas directamente de la red en formato PCAP que incluye las cabeceras de las capas OSI, además de la carga útil. Estos datos fueron tratados utilizando la librería *libpcap* [144] para extraer la carga útil desde tramas TCP a un fichero binario, desechando tramas que no contienen datos válidos para la aplicación (tramas ARP, DHCP, ICMP, etc.). Dicho fichero contiene el *payload* en un formato comprimido para disminuir el número de transferencias hacia el procesador de eventos. Este generador de tráfico se ha reutilizado desde la etapa de verificación del sistema. En [145] se muestran más detalles de la implementación del *software* del sistema.

Para depurar el sistema se integra un conjunto de utilidades del tipo ILA (In-Circuit Logic Analyzer) que facilitan la captura y presentación del estado real de las señales en la plataforma. Las señales se capturan a través de la interfaz de programación y depurado JTAG y se visualizan mediante el entorno virtual de análisis lógico *ChipScope Analyzer*.

En la figura 72 se representa el proceso de transferencia de datos a través de la interfaz LocalLink obtenido para la depuración del sistema sobre el prototipo durante la transferencia de datos al acelerador *hardware*.

El comienzo de la trama viene definido por la activación, a nivel bajo, de la señal SOF de LocalLink. En el caso mostrado, el DMA interrumpe la transmisión al desactivar la señal SrcRdy de LocalLink. Una vez que reanude la transmisión, comienza la carga útil activando la señal SOP. Para señalar el fin de la trama, el DMA del bloque de procesador activa la señal de fin de carga

útil (EOP) en el mismo instante en el que se envía la palabra, para, en el siguiente ciclo, activar la señal de fin de trama (EOF).

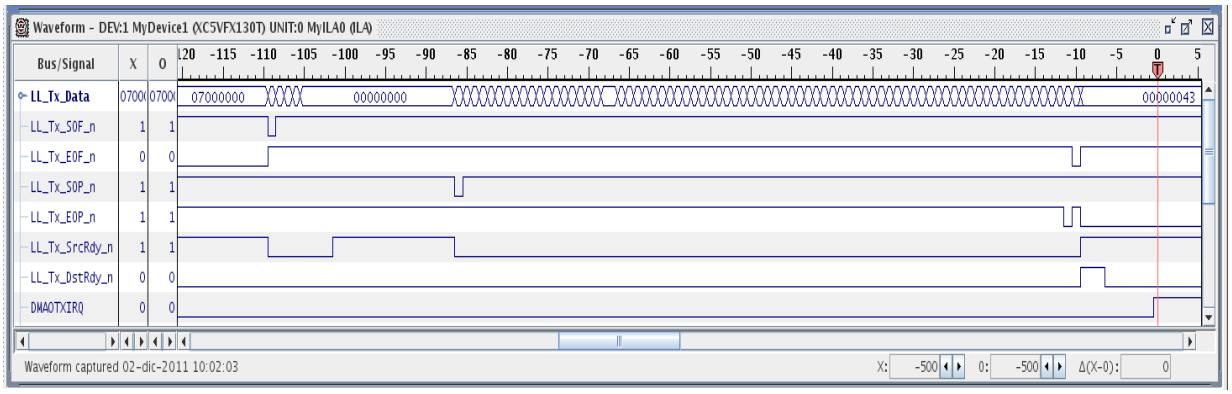


Figura 72. Depurado sobre el prototipo de la transferencia con ChipScope

```
ila connect -force chipscope_ila_0/chipscope_ila_0/i_chipscope_ila_0/
trigger 0 rt_core_LLINK0_LL_Tx_Data[0]
ila connect -force chipscope_ila_0/chipscope_ila_0/i_chipscope_ila_0/
trigger 1 rt_core_LLINK0_LL_Tx_Data[1]
...
```

Además, es posible acceder a las señales internas del procesador de eventos para capturar el estado de las señales de módulos de granularidad más baja del diseño.

Para calcular el factor de aceleración introducido mediante el acelerador *hardware* implementado en la FPGA se ha hecho uso del *timer* introducido en la plataforma que ha permitido obtener marcas de tiempo de ejecución del sistema.

El rendimiento del sistema se define como el conjunto de eventos procesados por unidad de tiempo y el factor de aceleración como los cocientes entre ambos rendimientos, con o sin acelerador *hardware* (tabla 8).

$$Rendimiento = \frac{Total\ de\ eventos\ procesados}{Unidad\ de\ tiempo} \qquad F_a = \frac{Rendimiento_{HW}}{Rendimiento_{SW}}$$

Tabla 8. Factor de aceleración alcanzado con la integración del acelerador HW

	Tramas	Eventos por trama (promedio)	Tiempo de procesado (ms)	Rendimiento (eventos/s)
Solución PC (1)	40.000	10	666,6	600.000
Solución con Aceleración HW (2)	40.000	10	80,0	5.000.000
Factor de Aceleración (Fa) (2)				8,33
(1) Servidor Intel Xeon a 2,67 GHz, 8GB de Memoria RAM DDR3 (2) La versión optimizada de este diseño alcanza el procesamiento de 22.000.000 de eventos/s con un factor de aceleración de 36				

3.14. Implementación ASIC

Se presentan en este apartado la implementación realizada del acelerador *hardware* utilizando una metodología de diseño orientada hacia la implementación ASIC desde síntesis de alto nivel. Al igual que se ha realizado para el caso de la implementación en FPGA, el objetivo es demostrar la viabilidad del flujo de diseño adoptado, si bien en este caso se cubrirán solo las etapas esenciales del flujo de diseño: síntesis de alto nivel, síntesis lógica, colocado, ruteado y análisis temporal. No se han tenido en cuenta aspectos como la optimización en potencia ni aspectos del diseño para test en esta primera aproximación al diseño, como sí se ha hecho para la implementación FPGA.

Para la implementación ASIC se parte de la misma descripción funcional del diseño en SystemC, verificado funcionalmente. Sin embargo, como veremos a continuación será necesario adaptar el flujo de diseño debido a la integración de bloques de memoria desde las fases iniciales del flujo de diseño. Esta disponibilidad de bloques de *foundry* es típica del diseño ASIC. En este caso es necesario separar la funcionalidad de las memorias y crear *wrappers* o envoltorios que permitan integrar la funcionalidad de los bloques de memoria proporcionados por la *foundry.*

3.14.1. Tecnología elegida

Para esta implementación se ha optado por la librería de células estándar de UMC CMOS 65nm. Se trata de una tecnología madura, bien soportada por los flujos de diseño de síntesis e implementación. Las principales características de esta tecnología son:

- Dispone variantes para prestaciones estándar (SP) y para baja corriente de fuga (LL)
- Tiene 1 capa de Poly y hasta 10 capas de metal
- El retardo de una puerta NAND básica es de 30 ps
- El consumo de potencia es de 2 pJ/MHz

La librería utilizada está desarrollada por UMC utilizando la variante LL de la tecnología. Las células de la librería ocupan 1,8 µm de altura (9 pistas de 0,2 µm por pista) y utilizan un único metal (M1). Las células poseen una longitud de puerta de 0,06 µm y el ancho del bus de alimentación y tierra (POWER/GND) es de 0,3 µm. Incluye 216 tipos de células con un total de 1088 células diferentes. La tensión de alimentación varía entre 0.9 y 1,32 V. La librería incluye modelos estructurales, temporales y físicos en formatos estandarizados (VHDL/Verilog, Cadence LEF, Synopsys Liberty, etc.)

Además se ha hecho uso de los generadores de bloques de memoria proporcionados por Faraday Technology [146]. En concreto se ha usado el generador de memorias RAM de doble puerto para utilizarlas en el diseño. Los bloques de memoria se han personalizado para ocupar la menor área posible, de tal forma que se han utilizado 24 bloques diferentes según se muestra en la tabla 9.

Todos los bloques de memoria disponen de doble puerto de lectura y escritura, con relojes independientes, lo que permite lecturas y escrituras simultaneas en la memoria, a frecuencias

de reloj diferentes. Para este diseño se ha utilizado un único dominio de reloj tanto para lecturas como para escrituras. Se utiliza la misma tensión de alimentación (1,2 V) que para el resto del diseño, y se usan hasta 4 niveles de metal internamente en el bloque, en una celda de 8 transistores [147].

Como se ha indicado ya, y se explicará más adelante, la utilización de estos bloques de memoria introduce modificaciones en el flujo de diseño ya que obliga a encapsular los bloques para que sean utilizados en la herramienta de síntesis del alto nivel.

Tabla 9. Bloques de memoria utilizados en el acelerador

Tipo de Memoria	Tamaño	Ancho	Bits	Bytes
Dual Port RAM	1024	40	40960	5120
Dual Port RAM	1024	32	32768	4096
Dual Port RAM	1024	11	11264	1408
Dual Port RAM	128	128	16384	2048
Dual Port RAM	128	81	10368	1296
Dual Port RAM	128	48	6144	768
Dual Port RAM	128	35	4480	560
Dual Port RAM	128	120	15360	1920
Dual Port RAM	128	104	13312	1664
Dual Port RAM	128	88	11264	1408
Dual Port RAM	128	65	8320	1040
Dual Port RAM	128	57	7296	912
Dual Port RAM	128	32	4096	512
Dual Port RAM	64	128	8192	1024
Dual Port RAM	64	88	5632	704
Dual Port RAM	64	8	512	64
Dual Port RAM	64	33	2112	264
Dual Port RAM	64	98	6272	784
Dual Port RAM	64	56	3584	448
Dual Port RAM	64	40	2560	320
Dual Port RAM	32	128	4096	512
Dual Port RAM	32	76	2432	304
Dual Port RAM	32	16	512	64
Dual Port RAM	32	8	256	32

3.14.2. Síntesis de alto nivel

El flujo de diseño de alto nivel que establecemos para la implementación ASIC es similar al que hemos establecido para la implementación FPGA. La descripción de la tecnología, suministrada en formato Liberty, incluye información temporal, de área y potencia.

El sistema utiliza el entorno de síntesis lógica de Cadence, RC Compiler, para generar la información de área y la información temporal de las unidades funcionales. Estos datos son necesarios para tomar decisiones de asignación de recursos y de planificación.

```
set asic_clock_period 3000
set asic_lib_name uk65lscllmvbbr_120c25_tc
set asic_lib [concat [string trim $asic_lib_name].lib]
...
define_clock -name clock -period $asic_clock_period -rise 0 -fall
[expr $asic_clock_period/2]
set_attr tech_lib_names lib/$asic_lib [get_design]
```

En el caso de los módulos de memoria, que incluyen el controlador y la unidad de almacenamiento, es necesario realizar su encapsulado mediante un adaptador XML. En el fichero de parámetros XML se especifica el *wrapper* en formato Verilog -ver código más abajo- que adapta la conexión del bloque de memoria al diseño, su información temporal, y de área y potencia, el tamaño de la memoria, su ancho y las principales características de su interfaz (tipo de lectura, *reset*, polaridad del reloj, entre otras). Entonces la síntesis de alto nivel del módulo de memoria se simplifica mediante la creación de un adaptador de interfaces, tal como se muestra en la figura 73.

```
<?xml version="1.0"?>
<ctos_ip_definitions>
<RAMDef>
<name>wrap_ram_6_DatVariable_64_</name>
<wrapper_filename>wrap_ram_6_DatVariable_64_.v</wrapper_filename>
<liberty_filename>SJKA65_64X56X1CM4_tt1p2v25c.lib</liberty_filename>
<verilog_filename>SJKA65_64X56X1CM4.v</verilog_filename>
<num_words>64</num_words>
<width>56</width>
<min_write_width>0</min_write_width>
<interface_types>
  <elem>sync_read_write</elem>
  <elem>sync_read_write</elem>
</interface_types>
<has_reset>false</has_reset>
<reset_async>false</reset_async>
<reset_high>true</reset_high>
<clock_posedge>true</clock_posedge>
</RAMDef>
</ctos_ip_definitions>
```

```
module wrap_ram_6__DatVariable_64_ (CLK, RSTn, CE0, A0, D0, WE0, CE1,
WE1, A1, D1, Q1, Q0);
  parameter AWIDTH  = 6 ;
  parameter DWIDTH  = 56 ;
  parameter MLENGTH = 64 ;

  input CLK;
  input RSTn;
  input CE0;
  input WE0;
  input CE1;
```

```
  input WE1;

  input [AWIDTH-1 : 0] A0;
  input [DWIDTH-1 : 0] D0;

  input [AWIDTH-1 : 0] A1;
  input [DWIDTH-1 : 0] D1;

  output reg [DWIDTH-1 : 0] Q0;
  output reg [DWIDTH-1 : 0] Q1;

  wire CE0N, CE1N, WE1N;
  assign CE0N = ~ CE0;
  assign CE1N = ~ CE1;
  assign WE1N = ~ WE1;

  // Instancia 0
  SJKA65_64X56X1CM4 ram_block_acciones_av_inst(
          .A(A1),.B(A0),.DOA(),.DOB(Q0),.DIA(D1),.DIB(56'b0),
          .CKA(CLK),.CKB(CLK),.CSAN(CE1N),.DVSE(1'b0),.DVS(4'b0),
          .WEAN(WE1N),.WEBN(1'b1),.CSBN(CE0N));
endmodule
```

Para el ASIC el proceso de síntesis de alto nivel se realiza siguiendo una estrategia *bottom-up* al igual que se hizo para el caso de las FPGAs. Igualmente se han protegido las interfaces no permitiendo la modificación de la latencia en la síntesis para respetar el estilo de modelado preciso en ciclos de bus BCA, realizado en la comunicación entre bloques y con el exterior. Por otra parte se han relajado las restricciones de latencia en los bloques de procesamiento para facilitar la planificación de los bloques cumpliendo las restricciones temporales del diseño. En la figura 74 se muestra el flujo de diseño utilizado.

Los resultados obtenidos medidos en términos de área, retardo y potencia, agregados para los módulos del diseño se muestran en la tabla 10, figura 75 y figura 76 para distintas frecuencias objetivos. Como se puede apreciar la síntesis de alto nivel presenta resultados similares una vez alcanzados los objetivos temporales del diseño. En la tabla 10 se puede apreciar que para T_c = 2 ns no se ha conseguido cumplir con las restricciones temporales del diseño durante la fase de síntesis de alto nivel. Como se mostrará más adelante, este objetivo se alcanzará durante la optimización realizada en la síntesis lógica.

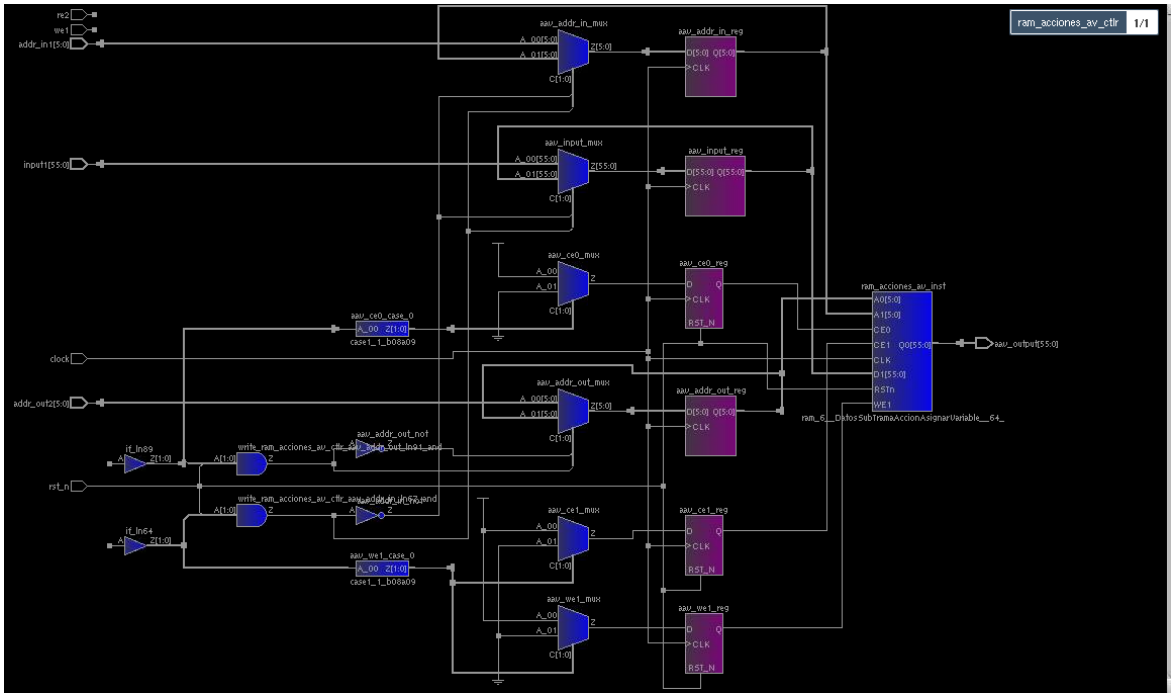


Figura 73. Adaptador de interfaz del bloque de memoria

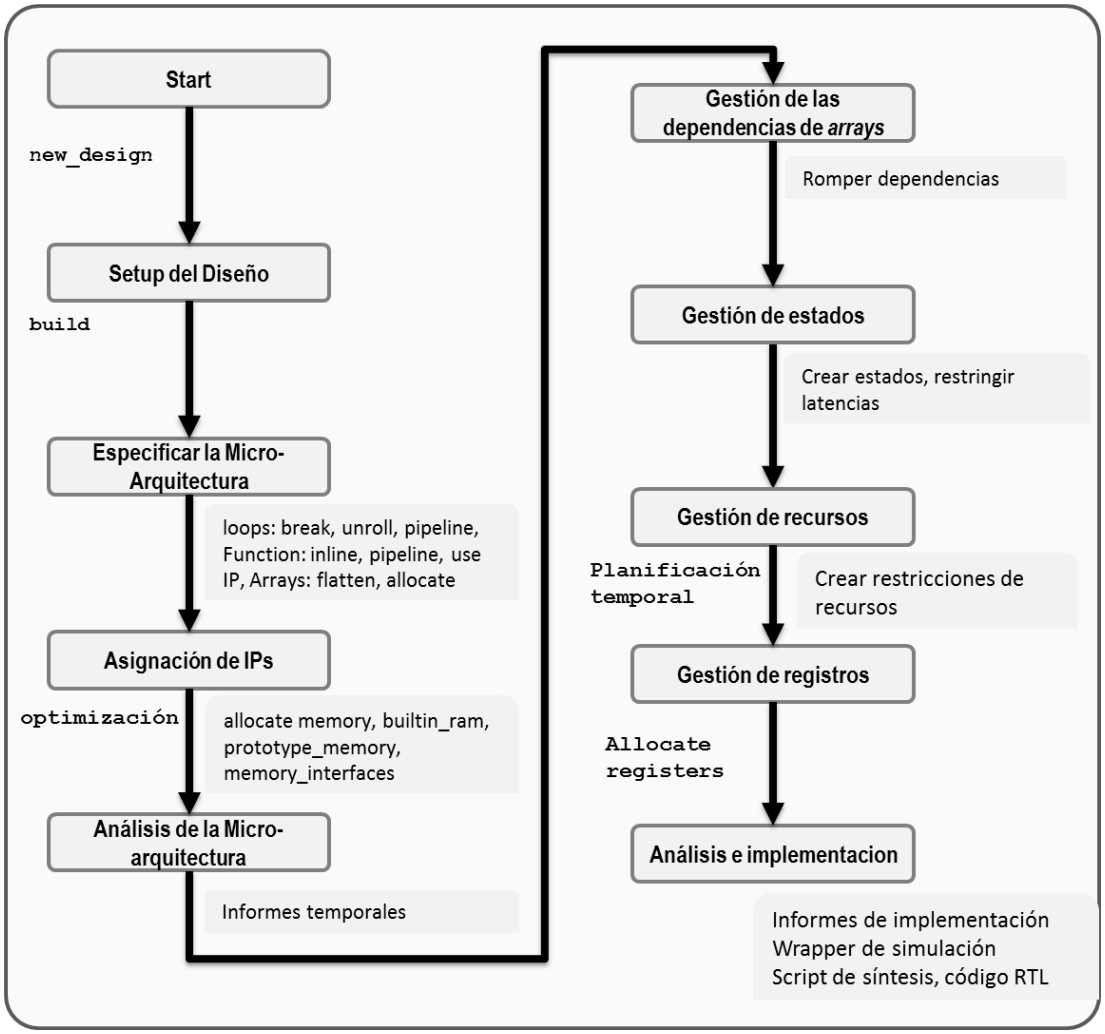


Figura 74. Flujo de diseño utilizado para la síntesis de alto nivel en CtoS para ASIC

Tabla 10: Resultados de la Síntesis de Alto Nivel

Síntesis de Alto Nivel					
TC=10 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	226.690,1	901	982	1.018,32	2,5319
Procesador de eventos	1.260.797,2	7.700	2.300	434,78	14,5995
Estado	1.512.427,6	9.383	617	1.620,74	12,2035
Ejecutor de Resultados	868.803,3	8012	1.988	503,01	3,8497
Total	3.868.718,2	7.700	2.300	434,78	33,1846
TC = 5 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	226.690,1	4.018	982	1.018,32	2,5319
Procesador de eventos	1.260.797,2	2.700	2300	434,78	14,6363
Estado	1.512.427,6	4.383	617	1.620,74	12,2035
Ejecutor de Resultados	868.803,3	3.012	1988	503,01	3,8886
Total	3.868.718,2	2.700	2300	434,78	33,2603
TC = 4ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	226.690,1	3.018	982	1018,32	2,5319
Procesador de eventos	1.260.797,2	1.700	2.300	434,78	14,7009
Estado	1.512.427,6	3.383	617	1.620,74	12,2035
Ejecutor de Resultados	868.803,3	2.012	1988	503,01	3,933
Total	3.868.718,2	1.700	2.300	434,78	33,3693
TC = 3 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	226.690,1	2018	982	1018,32	2,5364
Procesador de eventos	1.260.797,2	700	2.300	434,78	14,7573
Estado	1.512.427,6	2.383	617	1.620,74	12,2035
Ejecutor de Resultados	868.803,3	1.012	1988	503,01	4,0526
Total	3.868.718,2	700	2.300	434,78	33,5498
TC = 2 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	226.690,1	1.018	982	1.018,32	2,5365
Procesador de eventos	1.260.797,2	-300	2300	434,78	14,7758
Estado	1.516.207,2	1.383	617	1.620,74	12,2035
Ejecutor de Resultados	869.231,3	12	1988	503,01	4,1693
Total	3.872.925,8	-300	2300	434,78	33,6851

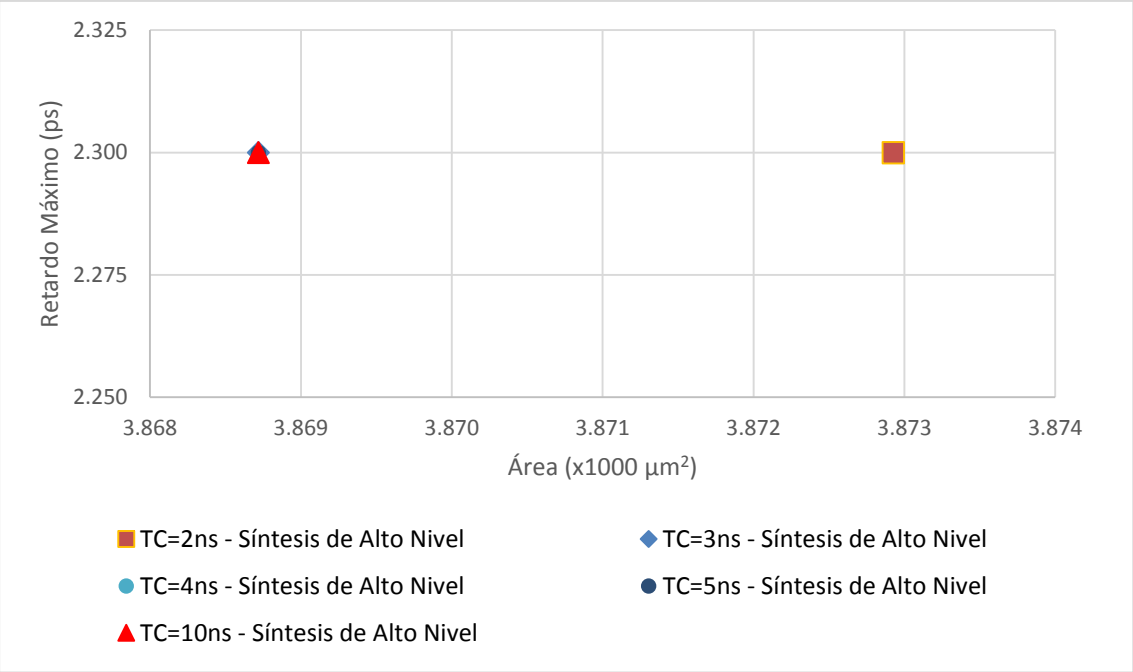


Figura 75. Espacio de diseño para diferentes restricciones en tiempo de ciclo

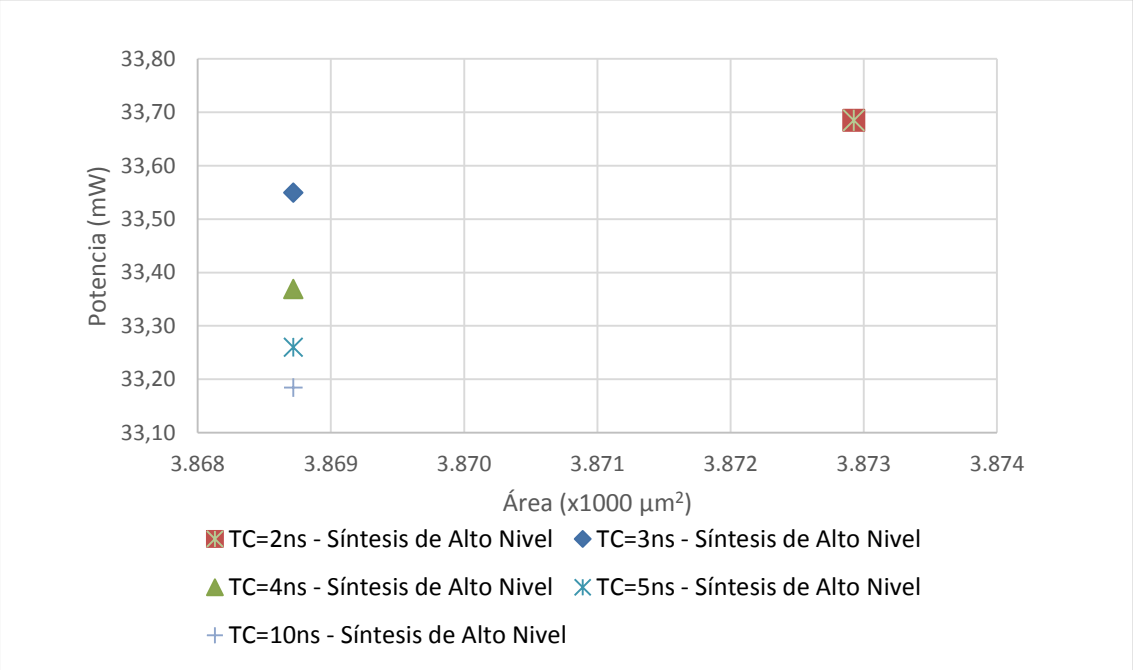


Figura 76. Potencia disipada versus área del diseño para diferentes tiempos de ciclo

3.14.3. Síntesis lógica

Durante la fase de síntesis lógica se parte de la descripción RTL obtenida durante la síntesis de alto nivel, con la jerarquía apropiada. Igualmente se dispone de la información temporal, de área y potencia correspondiente a la tecnología UMC y de los bloques de memoria en formato Liberty.

El proceso de síntesis lógica se ha realizado tanto con estrategia *bottom-up* o con estrategia *top-down*. Para todos los casos se han impuesto restricciones al diseño con el objetivo de minimizar el área, cumpliendo las restricciones de *slack* positivo en el diseño. Se ha diferido la optimización del árbol de reloj a las herramientas de síntesis física. Para el proceso de síntesis lógica se ha utilizado la herramienta Design Compiler de Synopsys. El flujo de síntesis se muestra en la figura 77.

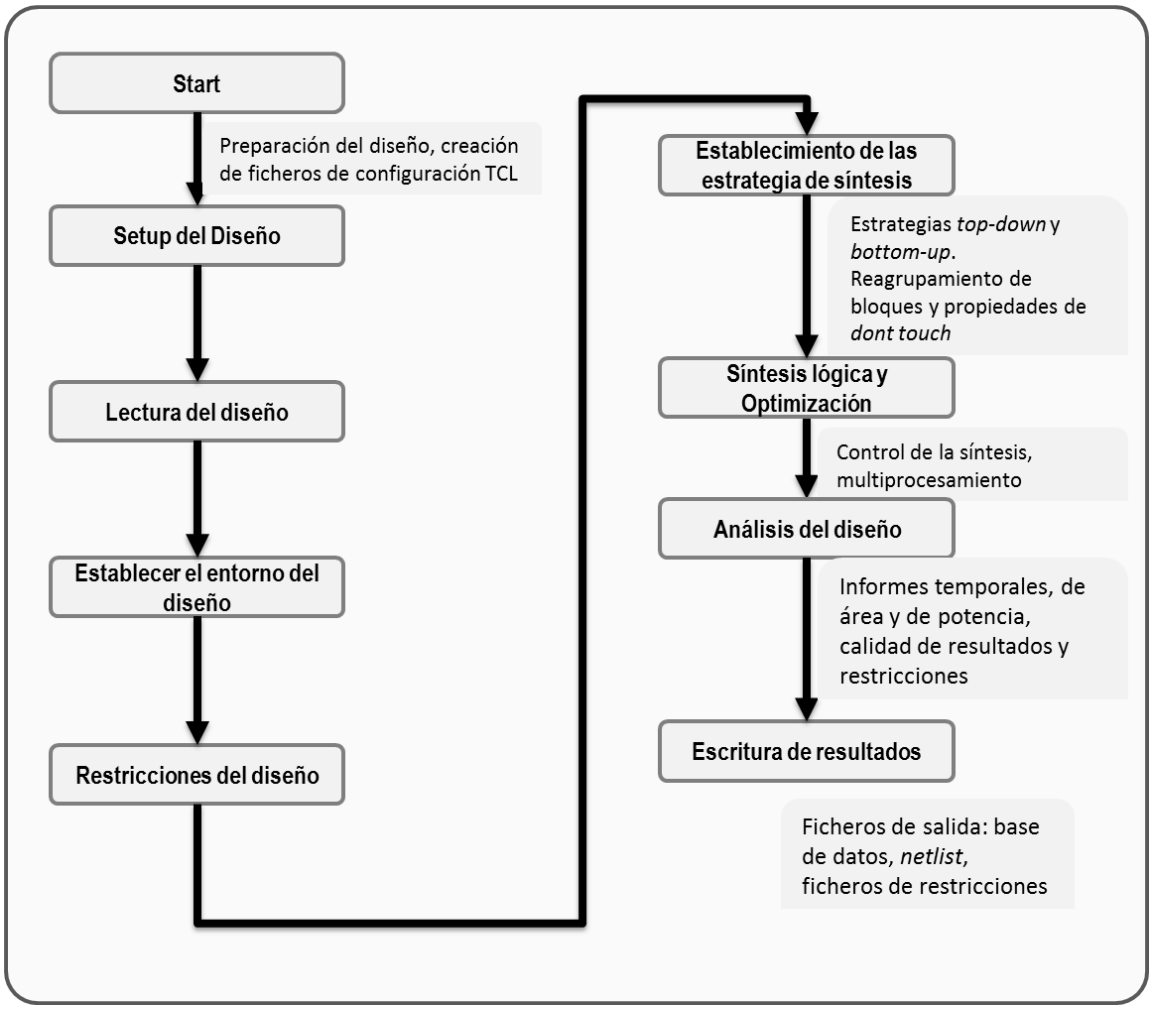


Figura 77. Flujo de síntesis lógica

Durante la fase de síntesis lógica se han conseguido los objetivos temporales, optimizando el área y potencia con respecto a los resultados preliminares obtenidos en la fase de síntesis de alto nivel, alcanzando una diferencia notable en área (40% de reducción). En cuanto a la potencia, las estimaciones son muy optimistas durante la fase de síntesis de alto nivel, y ciertamente pesimistas en la fase de síntesis lógica, con una factor que se incrementa con la frecuencia de diseño (x5 para 10 ns, x10 para 5ns y x28 para 2 ns). Los datos completos se muestran en la tabla 11, en la figura 78 se muestra la relación área-retardo y en la figura 79 la relación área-potencia.

Tabla 11. Resultados de la síntesis lógica

Síntesis Lógica					
TC=10 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	143.006,4	7.060	2.940	340,1	9,8385
Procesador de eventos	458.626,7	3.280	6.720	148,8	29,9412
Estado	1.402.543,3	8.280	1.720	581,4	125,4946
Ejecutor de Resultados	282.552,1	3.430	6.570	152,2	14,9191
Total agregado	2.286.728,5	3.280	6.720	148,8	180,1934
TC = 5 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	143.022,2	2.320	2.680	373,1	19,6777
Procesador de eventos	458.859,2	0	5.000	200,0	59,8347
Estado	1.402.523,5	3.340	1.660	602,4	250,7510
Ejecutor de Resultados	283.748,8	0	5.000	200,0	29,8551
Total agregado	2.288.153,8	0	5.000	200,0	360,1185
TC = 4ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	143.019,4	1.320	2.680	373,1	24,5952
Procesador de eventos	460.977,8	0	4.000	250,0	75,6620
Estado	1.402.495,8	2.050	1.950	512,8	313,4373
Ejecutor de Resultados	292.078,8	0	4.000	250,0	39,2089
Total agregado	2.298.571,8	0	4.000	250,0	452,9034
TC = 3 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	143.021,2	320	2.680	373,1	32,7914
Procesador de eventos	461.265,8	0	3.000	333,3	100,2586
Estado	1.402.559,1	1.050	1.950	512,8	418,2281
Ejecutor de Resultados	295.278,1	0	3.000	333,3	53,7536
Total agregado	2.302.124,3	0	3.000	333,3	605,0317
TC = 2 ns	**Área (um2)**	***Slack* (ps)**	**Ruta Crítica (ps)**	**Frecuencia máx. (MHz)**	**Potencia (mW)**
Interfaz	143.203,3	0	2.000	500,0	49,1828
Procesador de eventos	462.259,4	0	2.000	500,0	150,3737
Estado	1.402.380,6	460	1.540	649,4	626,8878
Ejecutor de Resultados	297.886,3	0	2.000	500,0	80,9475
Total agregado	2.305.729,7	0	2.000	500,0	907,3918
Procesador completo *(top-down)*	2.319.852,4	0	2.000	500,0	799,0269
Procesador completo *(bottom-up)*	2.305.025,1	-4.720	6.720	148,8	804,5374

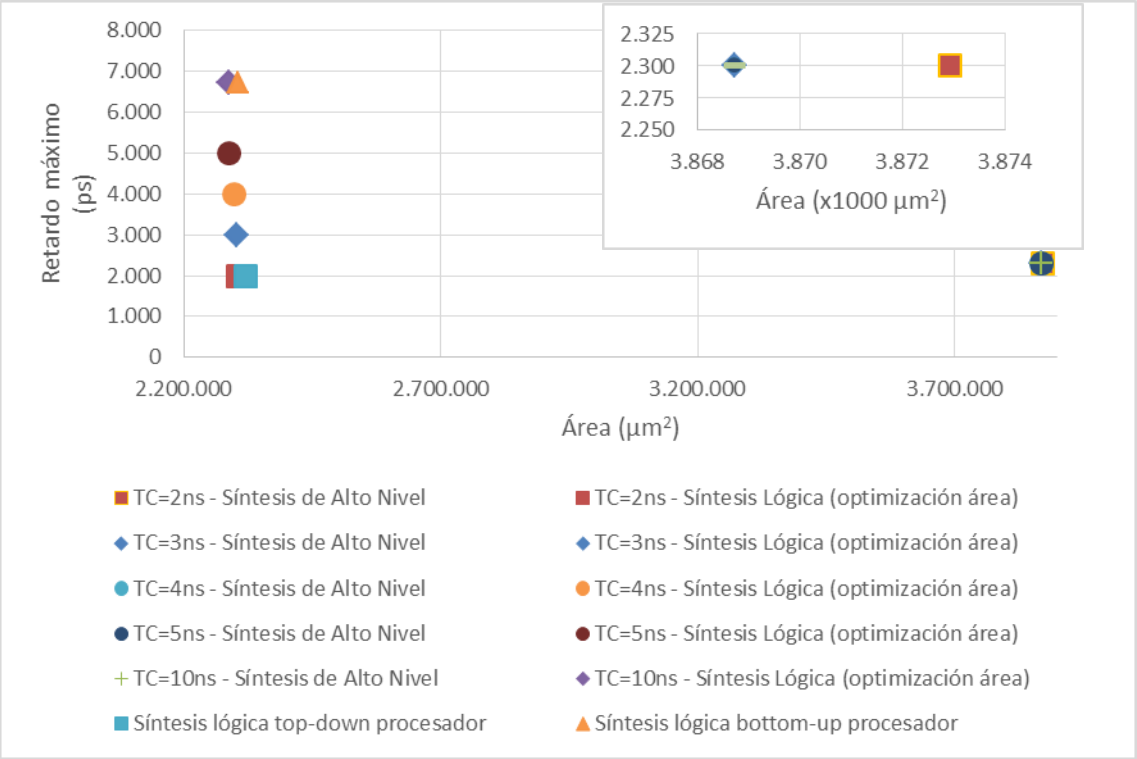


Figura 78. Espacio de diseño Retardo - Área para la síntesis lógica

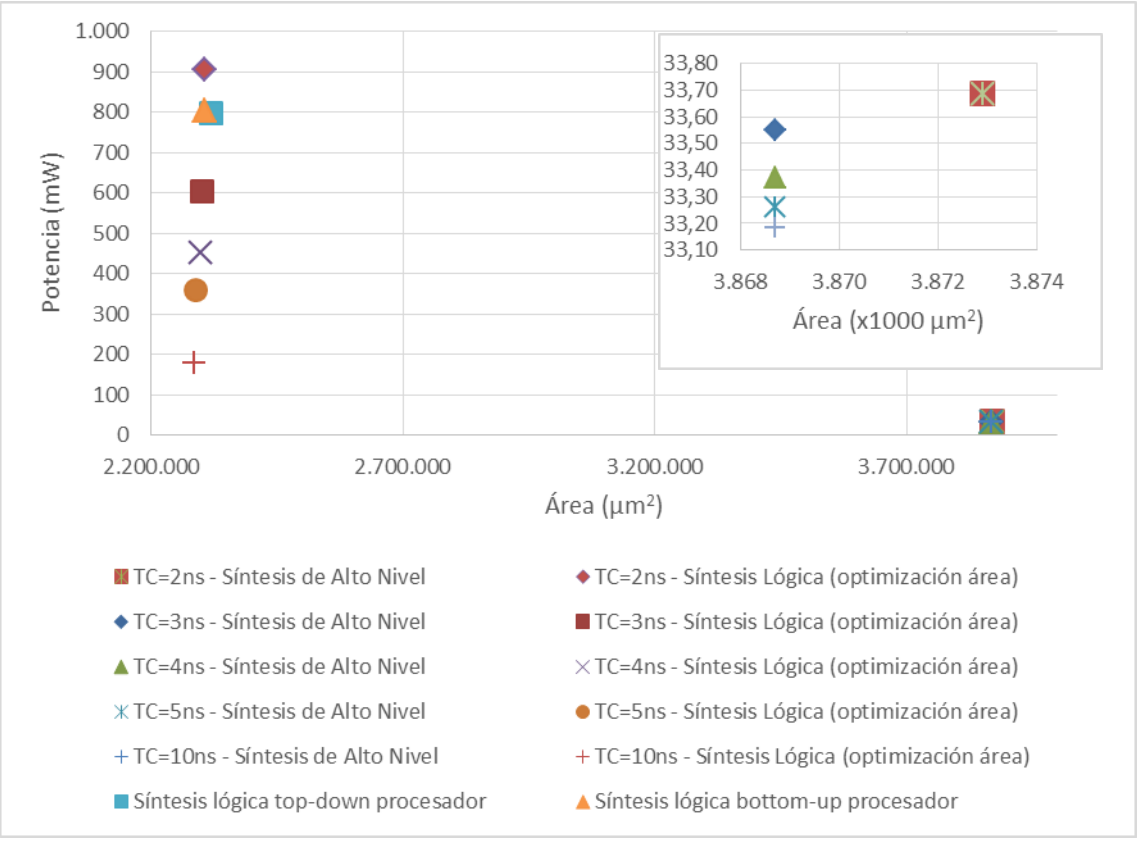


Figura 79. Espacio de diseño potencia - área para la síntesis lógica

Los datos resumen que se presentan al final de la tabla 11 son el resultado de aplicar la metodología de diseño a los diferentes bloques que componen el diseño. Como se puede observar, en la metodología *bottom-up* se presentan problemas de optimización temporal, lo que precisa la aplicación de metodologías de síntesis y caracterización de cada bloque, una vez identificados los bloques que incumplen las restricciones temporales. Por ello se opta por aplicar metodologías *top-down*, consiguiendo alcanzar el objetivo temporal. Como se puede observar en las siguientes gráficas, los resultados obtenidos representan el área mínima para cada una de las frecuencias de funcionamiento.

La comparación de las distintas unidades funcionales del diseño muestra que una parte importante del área y del consumo de potencia está concentrada en los bloques de memoria, dada la necesidad de reprogramar sus parámetros en tiempo real, así como de mantener estructuras de datos complejas para su procesamiento. En concreto, las áreas ocupadas por los bloques de memoria que conforman la unidad de estado y las correspondientes potencias disipadas se comparan en la tabla 12 y figura 80.

Tabla 12. Comparación de área y potencia de la unidad de estado

	Área µm²	**(% total)**	**Potencia mW**	**(% total)**
Unidad de estado	1.402.380,60	(60%)	626,89	(78%)
Bloques de memoria	1.320.939,50	(57%)	514,33	(64%)
Controladores de memoria	81.441,10	(4%)	112,56	(14%)
Procesador	2.319.852,40	(100%)	799,03	(100%)

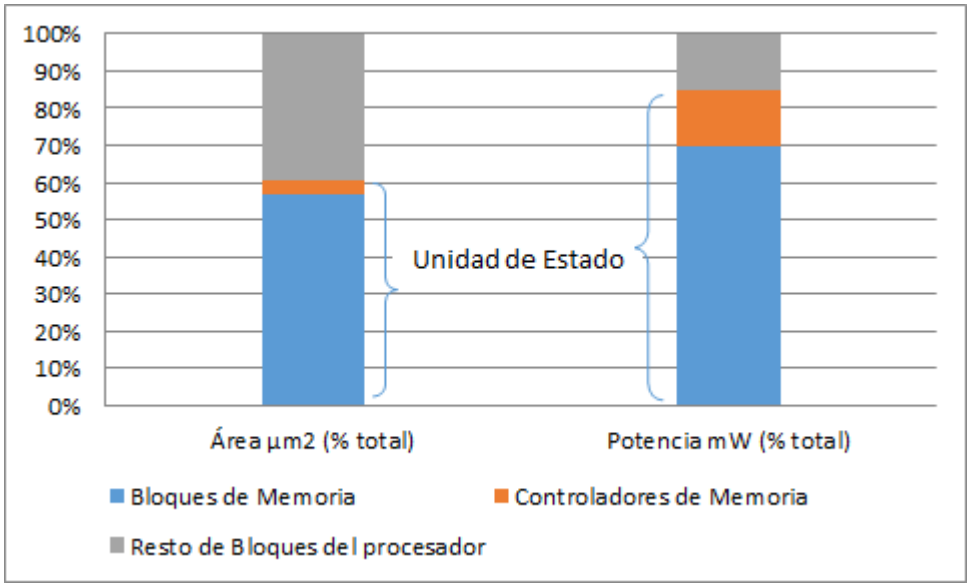


Figura 80. Impacto en términos de área y potencia de la Unidad de Estado

3.14.4. Implementación física

Con objeto de completar el flujo de diseño se ha realizado la implementación física del diseño, sintetizado para un tiempo de ciclo de 2 ns obtenido mediante una metodología *top-down*. El resultado de la implementación es la creación de un bloque IP que implementa la funcionalidad del procesador.

Se ha utilizado el flujo de implementación física de Cadence Encounter, partiendo de la información tecnológica de la librería de células estándar de UMC CMOS 65 nm mencionada anteriormente.

El flujo de diseño utilizado se muestra en la figura 81, obteniéndose como resultado el *layout* presentado en la figura 82 y figura 83.

El resultado final es sensible a la colocación de los bloques de memoria, a su interacción con los *stripes* de alimentación y los anillos correspondientes al bloque de memoria. Al utilizar una arquitectura de memoria distribuida es preciso ubicar los bloques correctamente para minimizar la longitud de las interconexiones, reduciendo por tanto el área y los retardos. A diferencia del caso de la FPGA, donde los bloques de memoria presentan ubicaciones fijas en el dispositivo, aquí es posible realizar una ubicación optimizada de los bloques de memoria y aumentar las prestaciones del diseño.

El circuito diseñado no posee *pads* de E/S ya que está pensado para ser integrado en un SoC conjuntamente con el resto de bloques necesarios para constituir una plataforma similar a la presentada en el caso de la FPGA.

El diseño completo del procesador de eventos ocupa un área de 6.735.458,60 μm^2 de los cuales el 65% corresponde al área de interconexionado y buses de alimentación y un 35% a lógica y bloques de memoria. El área de interconexionado incluye los halos de guarda necesarios para facilitar la conexión de los buses de datos y las señales de control a los bloques de memoria. La diferencia entre los resultados obtenidos en cuanto a área entre la síntesis lógica y la implementación física se debe principalmente al factor de inclusión de los halos necesarios para el ruteado de los bloques de memoria, que no están contemplados en los modelos utilizados durante la fase de síntesis lógica. Igualmente los modelos de interconexión requieren anotaciones más precisas de área, solo disponibles una vez realizado el *floorplanning* del diseño.

Todo el proceso de colocado de bloques de memoria y de células estándar, generación del árbol de reloj y ruteado han sido guiados por las restricciones temporales exportadas desde la herramienta de síntesis lógica en formato SDC, tratando de conseguir el acercamiento entre la información temporal obtenida en síntesis lógica y la final del diseño una vez implementado (*timing closure*). El diseño finalmente se somete a un análisis temporal estático. De los resultados obtenidos se desprende que la ruta crítica, una vez ruteado el diseño, requiere un tiempo de ciclo de 2,81 ns, lo que implica un funcionamiento a una frecuencia de 355 MHz.

Igualmente el diseño se somete a la verificación del diseño físico para comprobar el respeto a las reglas de diseño, eliminar la posibilidad de segmentos de interconexión que presenten efecto antena y otros aspectos relacionados con el diseño final del *layout* del circuito.

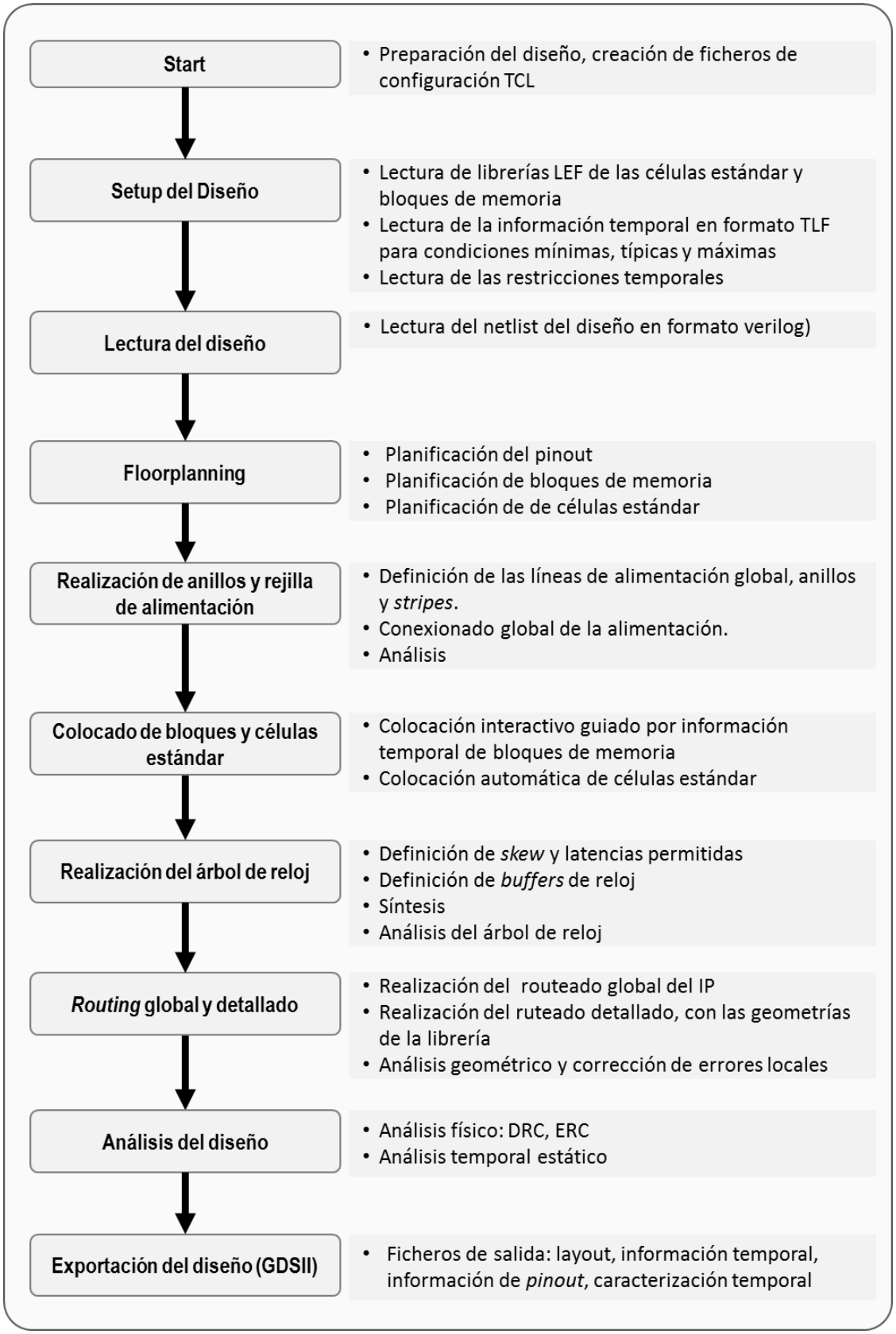


Figura 81. Flujo para la implementación física del diseño

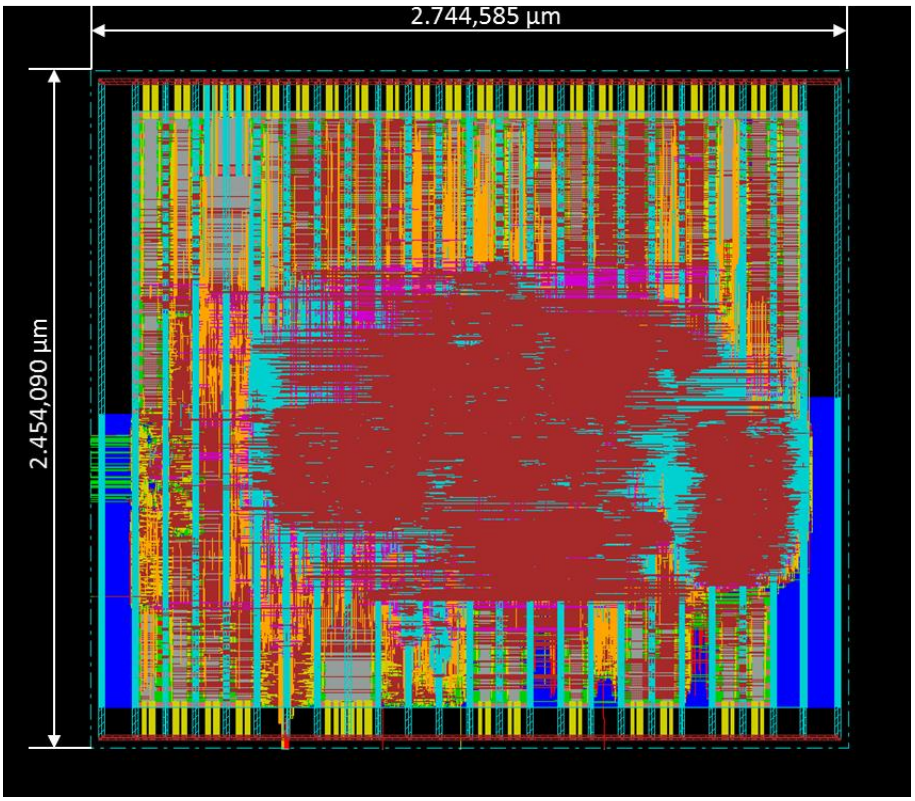


Figura 82. *Layout* del diseño

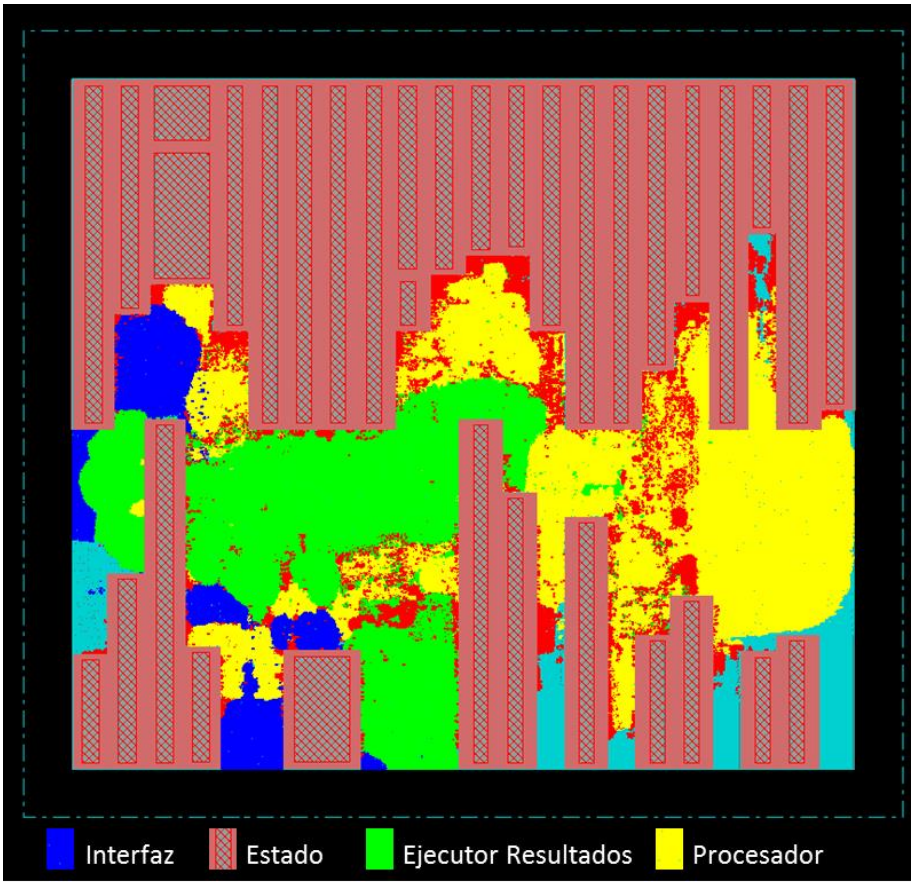


Figura 83. Distribución de bloques en el circuito

El circuito diseñado incluye un total de 781.896 células estándar organizadas en 1196 filas y 35 *hard* macros, que implementan los bloques de memoria. Posee 87 pines de E/S. El circuito se ha ruteado utilizando 9 capas de metal, con una mayor utilización de la capa ME5 tal como se muestra en la figura 84. La densidad efectiva de utilización del núcleo del chip que incluye células estándar y bloques, es del 65,2% mientras que esta utilización disminuye si consideramos los anillos de alimentación al 53,8% (tabla 13).

Tabla 13. Datos de utilización del Chip

Parámetros	Valores
Nº. de células	781.931
Nº. de *Hard* Macros	35
Nº. de células estándar	781.896
Nº. de Nodos	328.131
Pines de E/S	87
Pines totales del diseño	1.188.688
Promedio de *pines* por señal	3,62
Capas de ruteado	9
Capas usadas en los pines	6
Nº. de *Nets*	315.229
Nº. de conexiones	873.538
Longitud total de interconexión (µm)	24.504.653,18
Longitud promedio por nodo	74,67
Área del núcleo (µm^2)	5.263.424,13
Área total del chip (µm^2)	6.735.458,60
Nº. de filas de células estándar	1196
% Densidad de las células estándar	53,82%
% Densidad de utilización efectiva del núcleo	65,26%
% Densidad de utilización efectiva del Chip	51,02%

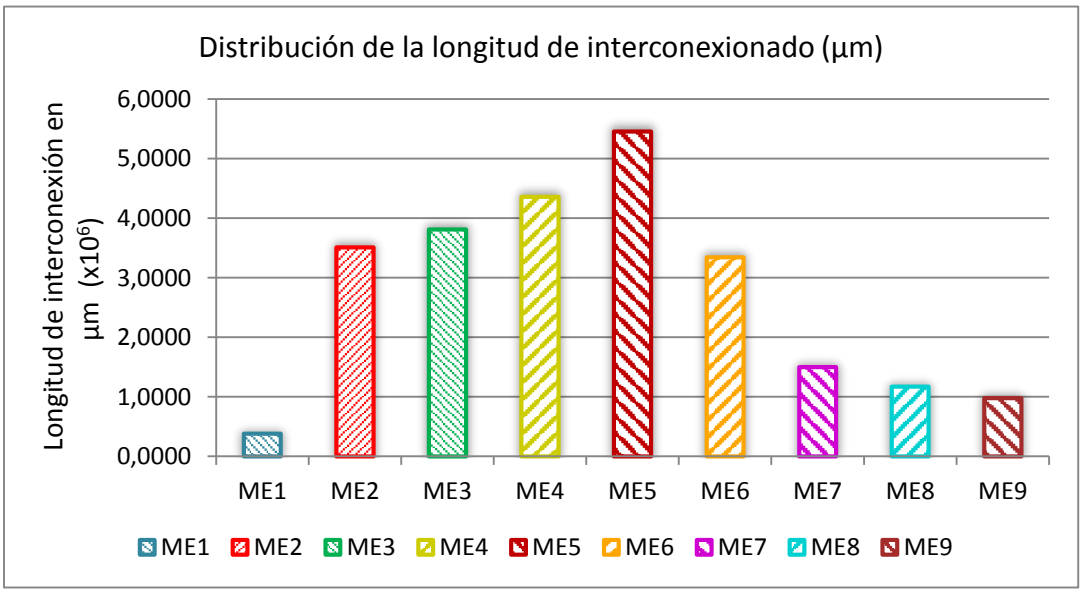


Figura 84. Distribución de longitudes de interconexionado por capas de ruteado

3.15. Conclusiones

En este capítulo se presenta la aplicación de referencia para el diseño del procesador de eventos, las arquitecturas disponibles para encontrar una solución de cara a la implementación del acelerador *hardware*, y la efectiva implementación tanto en FPGA como en ASIC.

Se evalúan diferentes alternativas arquitecturales comparándolas en cuanto a la relación entre su flexibilidad de configuración y sus prestaciones y se muestra el efecto de la carga de eventos sobre las prestaciones medidas en número de eventos procesados por segundo para cada una de las arquitecturas.

Igualmente se presentan los detalles de la metodología de síntesis de alto nivel utilizada para el diseño de un procesador de eventos complejos partiendo de un modelo funcional desarrollado en C/C++. El modelo se transforma en un conjunto de bloques funcionales en SystemC, partiendo de la información de la carga computacional y *profiling* de la aplicación inicial.

Este modelo de alto nivel creado, en SystemC, con precisión a nivel de ciclos pero abstracto a nivel de pines, es el punto de entrada a la síntesis de alto nivel. Se han explorado estrategias adecuadas para la gestión de la complejidad del diseño, utilizando metodologías *bottom-up* con interfaces bien definidas en alto nivel, que son propagadas al modelo RTL. En los distintos bloques se han aplicado las técnicas de síntesis de alto nivel, tales como el *inline* de funciones, la rotura de bucles combinacionales, el desenrollado de bucles o su segmentación donde ha sido necesario, la planificación y asignación de unidades funcionales y la asignación de registros.

Se ha restringido la modificación de las latencias de en las interfaces de comunicación, que en este trabajo han sido controladas por el diseñador. Este aspecto ha sido importante para mantener la coherencia en los protocolos de comunicación entre bloques, sobre todo cuando se ha realizado una integración *bottom-up* del modelo.

Otra restricción, dado el alto número de memorias utilizadas en el diseño, ha sido la gestión del protocolo de comunicación con memoria. Para ello se ha tenido en cuenta el comportamiento final de los bloques a utilizar, ya sea para tecnologías FPGA o ASIC, creando modelos sintetizables en alto nivel y controladores y adaptadores para ambas tecnologías.

El modelo RTL obtenido ha sido verificado comparándolo con el modelo SystemC de referencia, utilizando el mismo entorno de test desarrollado inicialmente para el proyecto, también en SystemC. Ello ha supuesto un incremento de productividad significativa a la hora de abordar un proyecto tan complejo.

El diseño RTL funcionalmente correcto ha sido prototipado sobre una FPGA Virtex-5 validando el diseño original. Igualmente se ha realizado la implementación ASIC con el objetivo de mostrar la viabilidad del flujo de síntesis propuesto para la implementación en tecnologías basadas en células estándar.

En todo este flujo de diseño se han utilizado herramientas comerciales disponibles, tales como Cadence CtoS, Cadence Incisive, Synopsys Sinplify Premier, Xilinx ISE+PlanAhead, Synopsys Design Compiler y Cadence Encounter. La utilización de lenguajes y formatos estandarizados, tales como Verilog y EDIF, ciertamente facilitan la integración sin costuras del flujo de diseño.

Sin embargo, se ha constatado de nuevo que durante la generación de código los fabricantes incluyen aspectos particulares de sus entornos de diseño que hacen precisa la creación de utilidades de adaptación, de manera que se pueda garantizar la compatibilidad entre las diversas herramientas conectadas en el flujo.

En este trabajo se ha hecho uso intensivo de herramientas basadas en UNIX que facilitan la integración y adaptación anteriormente citada, así como la gestión de los proyectos de diseño asociados a cada herramienta. En concreto, todo el proyecto de diseño está soportado por la gestión de *Makefiles* cuya gestión de dependencias asegura la integridad de los datos. La mayor parte de las herramientas se utilizan en modo de línea de comando, con *scripts* (la mayoría en Tcl) que capturan el conocimiento adquirido durante el trabajo con los que se asegura la repetitividad de los resultados obtenidos. Igualmente, para facilitar la obtención de resultados se hace uso intensivo de la ejecución basada en multiprocesamiento y distribuida entre diferentes equipos de computación, que facilita la obtención de resultados globales en un tiempo reducido.

El trabajo global incluye 26.380 líneas de código fuente sintetizable del modelo SystemC y 5167 líneas de *testbench.* Durante el proceso de síntesis se ha generado un total de 224.333 líneas de código Verilog y 787 de código SystemC/C++ para el *wapper* de cosimulación RTL. Igualmente el proceso de síntesis lógica ha generado 1.008.616 líneas EDIF para la implementación FPGA y 1.665.716 líneas Verilog a nivel de *netlist* (tabla 14).

Tabla 14. Dimensiones del esfuerzo de diseño

Modelos	Líneas de código	Factor
SystemC (incluyendo *testbench*)	41.311	x1
Verilog RTL	224.333	x5
EDIF (*netlist* FPGA)	1.008.616	x24
Verilog (*netlist* ASIC)	1.665.716	x40

En un trabajo de investigación y un diseño de este tipo es importante mantener la correlación de información entre el código fuente y la implementación de tal manera que sea posible conocer el origen de los fallos detectados en etapas posteriores del diseño ya sea por una especificación incorrectamente modelada o por un error en el proceso de diseño. En este flujo es posible exportar marcas desde el código fuente hasta el *netlist* final del diseño.

Otro de los aspectos a considerar en la metodología de diseño utilizada es la divergencia en la precisión de la estimación de los valores de parámetros del diseño, especialmente temporales, obtenidos durante la síntesis de alto nivel. Como se ha indicado, la estimación se realiza utilizando la propia biblioteca de funciones de Xilinx y su tecnología de síntesis, para el caso de la FPGA utilizada (o de Altera en su caso), y a partir de la información contenida en los ficheros de tecnología del proceso UMC en formato Liberty para el caso del ASIC.

La experiencia adquirida y la acumulada durante años ha mostrado que una vez que se ha obtenido la arquitectura RTL del sistema, existe aún un amplio margen de mejora en la síntesis lógica -no tanto en fases posteriores- por lo que antes de decidir aceptar o no la arquitectura

RTL obtenida desde síntesis de alto nivel es fundamental realizar la síntesis lógica del bloque y tomar la decisión en base a las prestaciones obtenidas. Por ello normalmente se realiza una iteración entre la herramienta de síntesis de alto nivel y la de síntesis lógica para optimizar la ruta crítica del diseño. Para ello es de vital importancia prestar atención a las anotaciones recibidas en el código RTL desde el modelo SystemC y desde el proceso de síntesis de alto nivel indicado anteriormente, y a sus opciones de optimización en alto nivel y en síntesis lógica. En esta iteración a veces es necesario modificar el modelo SystemC y/o añadir restricciones adicionales durante el proceso de síntesis de alto nivel, que después se propagan a la herramienta de síntesis lógica. Este proceso de mejora iterativa se ve facilitada por la metodología *bottom-up* utilizada para la síntesis de alto nivel, que permite tiempos de síntesis más aceptables para el diseñador que la metodología *top-down* en alto nivel. El potencial de una efectiva metodología *bottom-up* en alto nivel es muy dependiente de la captura del diseño, en nuestro caso y en nuestro criterio mediante una correcta descomposición SystemC. En síntesis lógica en cambio son fundamentales ambos enfoques.

Capítulo 4. Aplicación al diseño de un filtro de bucle adaptativo para H.264/AVC-SVC

4.1. Introducción

En la actualidad hay numerosos estudios que describen implementaciones *hardware* del filtro de bucle adaptativo o *deblocking filter* (DF) para el estándar de codificación de vídeo H.264/AVC-SVC. Muchas de ellas se orientan a reducir el consumo de potencia para diseños empotrados, en los cuales la duración de la batería es un factor clave. Algunas implementaciones se describen en los trabajos referenciados en [148], [149] y [150].

Este trabajo está orientado en cambio al estándar H.264/AVC-SVC, que posee como característica la capacidad de mejorar la vida de la batería de los dispositivos usando diferentes tipos de escalabilidad: temporal, espacial o de calidad (figura 85), dependiendo de los perfiles de consumo de potencia o del estado de la batería. El impacto del uso de cada tipo de escalabilidad en el consumo de energía se explica a continuación.

Escalabilidad temporal. El objetivo es cambiar en tiempo real el número de fotogramas que se decodifican, reduciendo el número de fotogramas que se decodifican por segundo. Esto produce una reducción en las prestaciones y en las operaciones y funciones del dispositivo y por tanto en su consumo de potencia. Si bien esta funcionalidad ya se incluye en AVC, SVC proporciona mecanismos mejorados para facilitar su utilización. Varios estudios conducentes al estándar han determinado la forma eficiente de establecer dependencias entre los fotogramas de tal forma que se facilite la toma de decisiones sobre cuál de los fotogramas no deba ser decodificado y así poder reducir el ratio de fotogramas decodificados por segundo [151].

Escalabilidad espacial. Habilita la reducción de la resolución de la imagen, procesando por tanto un número más reducido de pixeles, lo cual trae como consecuencia reducir el consumo de energía [5]. Para ello el vídeo se codifica con diferentes resoluciones, pudiendo utilizar los datos de las resoluciones bajas para predecir las muestras de resoluciones más altas, reduciendo así la tasa de bits transmitida.

Escalabilidad de la calidad. La variación de los parámetros de cuantización permite seleccionar diferentes niveles de calidad. La reducción de calidad basada en los valores de estos parámetros produce también una reducción en el consumo de potencia [152].

Es posible combinar los tres tipos de escalabilidad para obtener reducciones significativas en el consumo de potencia, y en las tasas de bits que se transmiten (figura 85).

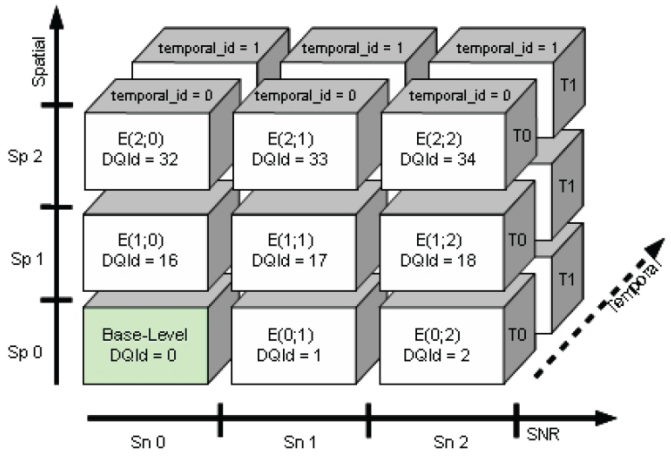


Figura 85. Ejemplo de combinación de escalabilidad: espacial, temporal y de calidad [153]

4.2. Decodificador OpenSVC

OpenSVC[153] ha sido desarrollado por IETR a partir de una librería en lenguaje C para la especificación del perfil *Scalable Baseline* con capacidad para dar soporte a la escalabilidad espacial, temporal y de calidad. El decodificador OpenSVC se basa en el estándar H.264/AVC-SVC con perfil *Baseline* añadiendo las características principales del *Main Profile* [154]. Es la referencia algorítmica que utilizamos en esta investigación.

Contrariamente a la implementación JSVM que decodifica la capa con mayor escalabilidad, por ejemplo, la capa de mejora con la escalabilidad espacial, temporal y de calidad, el decodificador OpenSVC puede decodificar el flujo de bits de una capa específica con una

determinada escalabilidad temporal. Esta particularidad proporciona adaptabilidad a diferentes plataformas de decodificación, seleccionando la capa correcta para obtener la decodificación en tiempo real. La librería también contiene mecanismos para intercambiar estas capas durante el proceso de decodificación, permitiendo al usuario seleccionar la capa a presentar mediante determinados comandos especificados por el usuario. Sin embargo, el decodificador puede también cambiar la capa automáticamente cuando se produce un error de transmisión que puede producir pérdidas en la calidad del vídeo. En caso de una decodificación parcial del flujo de bits, el decodificador desechará las capas no necesarias.

La síntesis de alto nivel minimiza el esfuerzo de diseño. En esta investigación sobre flujos de síntesis de alto nivel se ha realizado en primer lugar la adaptación del decodificador OpenSVC a una descripción de alto nivel en SystemC como ruta eficiente para la síntesis. Con este punto de partida y siguiendo la metodología de diseño ESL con las mejores prácticas propuestas se obtiene una implementación *hardware*. La utilización de técnicas de perfilado nos ha permitido observar los bloques de mayor carga computacional y críticos para el sistema. Se han asignado al *software* los bloques menos exigentes. Se ha decidido acelerar en *hardware* el DF ya que este bloque introduce la mayor complejidad y coste computacional, aproximadamente un tercio del coste computacional total del decodificador [155]. La implementación completa del decodificador SVC ha sido objeto del proyecto PCCMUTE [156][157] en que el decodificador H.264/AVC-SVC [103][104], se implementa sobre una arquitectura híbrida basada en un SoC OMAP 3530 que contiene la mayor parte de los bloques del sistema, con excepción del DF que se implementa haciendo uso de un acelerador *hardware* diseñado sobre una FPGA Virtex-5 XC5VFX70T de Xilinx que incluye un PowerPC 440. La comunicación entre el OMAP y la FPGA se realiza a través de la interfaz GPMC [158]. En esta investigación se explora además una implementación ASIC del diseño para obtener ganancias adicionales en prestaciones y en reducción de área y consumo de potencia.

4.3. Funcionalidad del DF

El DF es responsable de eliminar las diferencias entre los bordes de los bloques, para crear una imagen suavizada. En el proceso de descodificación, la unidad de predicción busca bloques similares al actual. El bloque encontrado no es exactamente idéntico, produciendo errores de predicción. Para mejorar la eficiencia en la codificación, estos errores se transforman y se cuantifican. Después de la decodificación, el bloque reconstruido es diferente del bloque original. Estas diferencias se destacan especialmente en los bordes. Para reducir el grado de discontinuidad se aplica el DF.

Además, el DF tiene un alto coste computacional, ya que es necesario filtrar todos los pixeles de los bordes de los bloques adyacentes (figura 86). Esta es la razón principal para plantearse una implementación *hardware* de este bloque.

La funcionalidad del DF se divide principalmente en dos etapas:

a) Una primera etapa donde se calculan los parámetros de filtrados asociados con cada bloque. Los parámetros de filtrado deben calcularse para cada eje horizontal y vertical, definiendo estos ejes como los límites entre cada par de bloques de 4x4 de

luminancia y 2x2 de crominancia. La figura 87 muestra los ejes de filtrado y la figura 88 presenta las particiones separadas por los parámetros de filtrado.

b) Una segunda etapa que realiza el filtrado en varios niveles usando los parámetros calculados anteriormente.

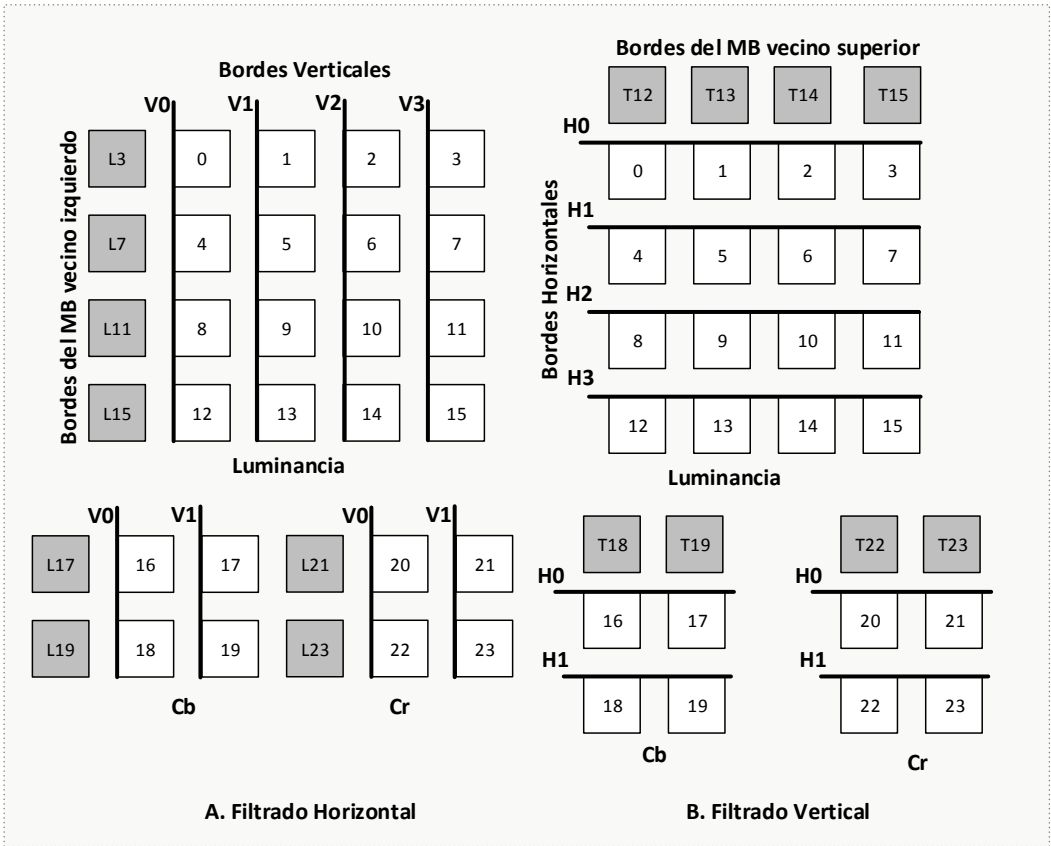


Figura 86. Distribución de datos en los MB del DF (adaptado de [159])

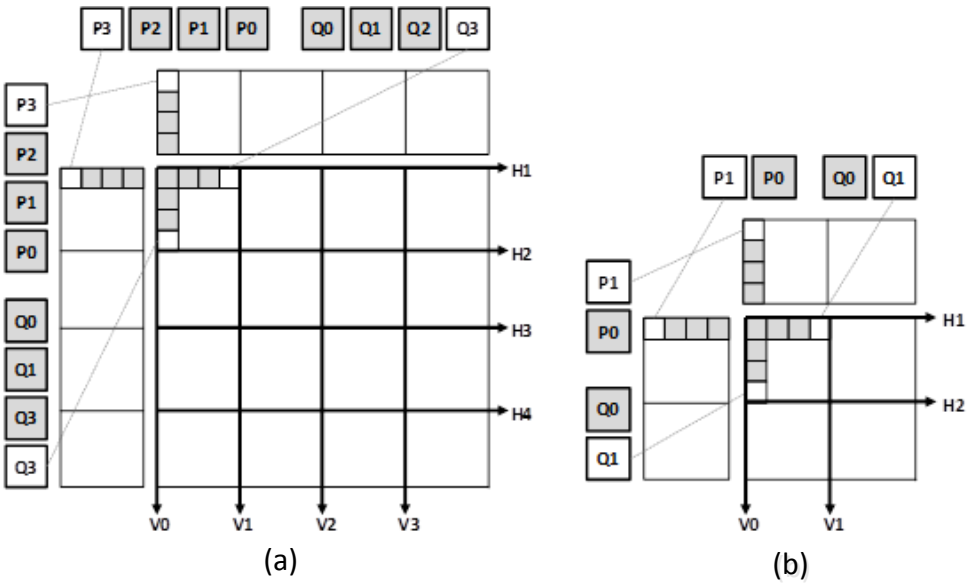


Figura 87. Ejes de filtrado horizontal y vertical para el filtrado de la luma (a) y croma (b)

Este proceso de filtrado es adaptativo ya que la intensidad del filtro depende del tipo de MacroBloque (MB) de entrada y sus valores. Dicha intensidad se conoce como *Boundary Strength* (BS) y se calcula siguiendo el diagrama de decisión mostrado en la figura 89.

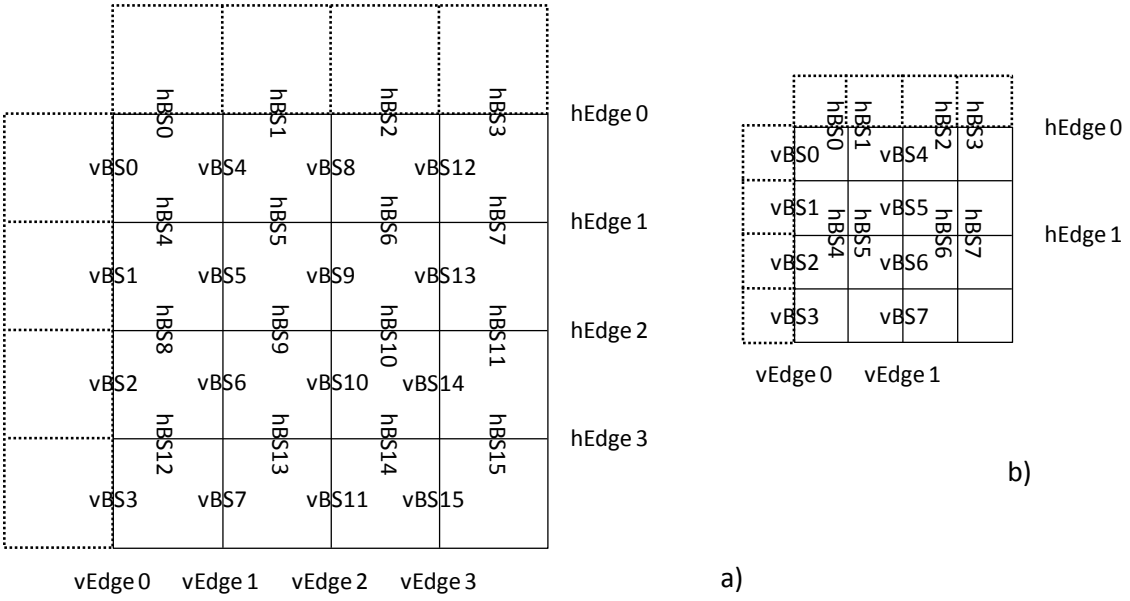


Figura 88. Parámetros para el filtrado de la luma (a) y croma (b)

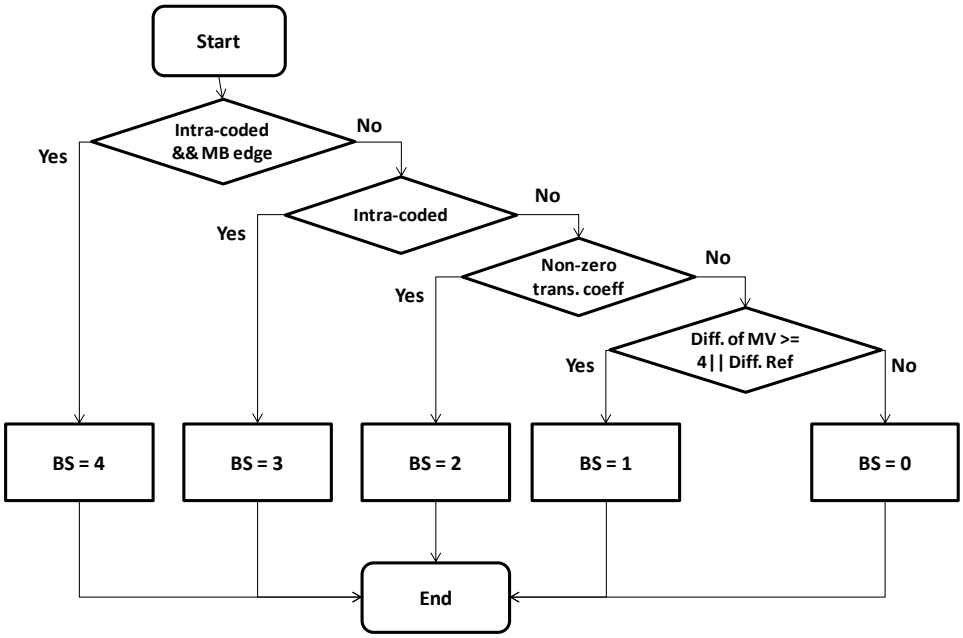


Figura 89. Diagrama de decisión para calcular el *Boundary Strength*

Una vez se calcula el BS, son necesarios los valores de α y β que previenen del sobrefiltrado de los bordes que realmente están presentes en la imagen. Este sobrefiltrado podría causar pérdidas de claridad en la imagen filtrada. Estos parámetros se calculan basándose en los parámetros de cuantización de los bloques siguiendo los siguientes pasos:

- Paso 1: Los parámetros de cuantización para filtrar los bloques de 4x4 se conocen como `QPP` y `QPQ`

- Paso 2: Se calcula el parámetro de cuantización promedio:
- `QPav = (QPP + QPQ + 1) >> 1`
- Paso 3: Se calcula el índice de la tabla de parámetros α:
- `indexA clip3 = (0, + FilterOffsetA QPav, 51)`
- Paso 4: Se calcula el índice de la tabla de parámetros β:
- `indexB clip3 = (0, + FilterOffsetB QPav, 51)`
- Paso 5: `α = Alpha_Table(indexA)`
- Paso 6: `β = Beta_Table(indexB)`

La función Clip3 verifica que un valor está dentro de los rangos especificados, en este caso de 0 a 51, saturando los valores en el caso de que sea menor de 0 ó mayor de 51 respectivamente.

El último parámetro a calcular es el *flag* de filtrado, que decide, basándose en los valores calculados previamente y en los valores de la muestra, si filtrar o no esta muestra actual. Las condiciones que debe cumplir el filtro son las siguientes:

- Condición 1: `BS != 0`
- Condición 2: `Abs(p0-q0) < α`
- Condición 3: `Abs(p1-p0) < β`
- Condición 4: `Abs(q1-q0) < β`

En la segunda etapa se realiza el proceso de filtrado de la luma y la croma. Para ello, se toman cuatro muestras de cada uno de los dos bloques que tienen un lado en común. Dependiendo de los parámetros de BS el proceso de filtrado será intenso o débil, lo que implica la ejecución de un flujo diferente de operaciones en cada caso. Una exposición más completa y detallada de estos flujos de procesamiento se da en [160].

4.4. Arquitectura propuesta del DF

La figura 90 muestra la arquitectura interna del DF, que está formada por cinco bloques principales:

- El primer bloque **Interface** constituye la interfaz responsable de proporcionar los datos de entrada al DF y salida del DF, así como la generación de señales de control para otros bloques.
- El bloque **Param_BS** calcula el BS para cada eje de filtrado tanto para luminancia como para la crominancia.
- El bloque **Param_Clip** es responsable de calcular los parámetros umbrales α y β, y los parámetros C en el caso de filtrado débil.
- El bloque **Luma_core** verifica si las condiciones para realizar el filtrado se cumplen, y

en este caso realizar el filtrado de la luminancia en cada eje del bloque.

- El bloque **Chroma_core** posee la misma funcionalidad que el bloque luma_core, pero esta vez para procesar las muestras de croma.

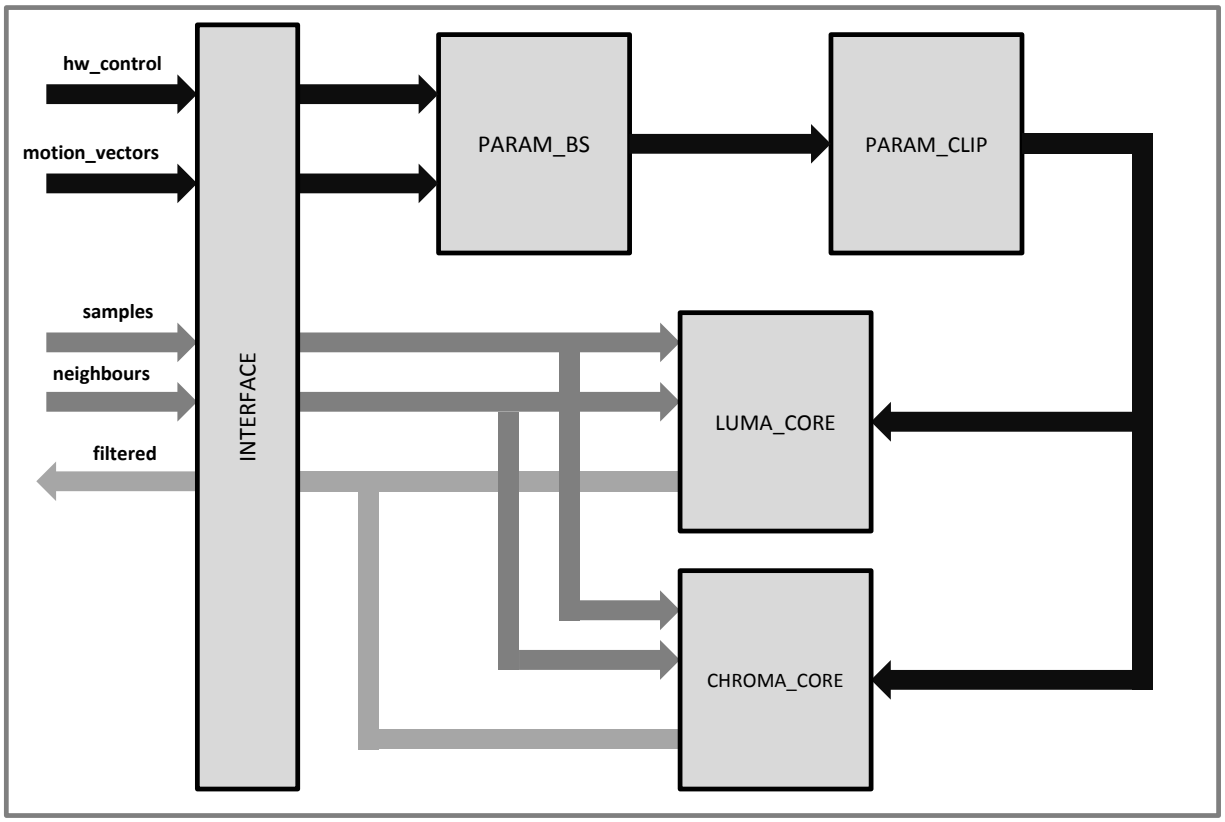


Figura 90. Arquitectura interna del DF

4.5. Implementación FPGA

Es este apartado se muestra la implementación realizada del DF utilizando el flujo de diseño aportado. Se parte de la descripción SystemC del bloque y se realizan los pasos de verificación, síntesis de alto nivel, síntesis lógica, diseño de la plataforma e integración del IP e implementación en la FPGA. Igualmente es preciso realizar el *software* empotrado integrado en la plataforma.

4.5.1. Arquitectura del DF

La arquitectura del DF se muestra en la figura 91. Esta arquitectura se divide en cuatro bloques principales:

- Un primer bloque implementa una interfaz LocalLink para conectar el DF a un bloque DMA desde el procesador empotrado disponible en la FPGA.
- A continuación se implementa una memoria BRAM de doble puerto, que almacena las muestras de píxeles de la imagen a filtrar (hasta formato CIF), más información de entrada de otros bloques del decodificador necesaria para el proceso de filtrado

- Un gestor de E/S que gestiona las peticiones de datos realizadas por el DF y realiza las operaciones de lectura/escritura en la memoria principal
- Finalmente, el DF posee cinco interfaces de E/S: entrada de muestras sin filtrar, entrada de muestra de los macrobloques vecinos, entrada de los vectores de movimiento, entrada de señales de control del bloque y salida de las muestras filtradas.

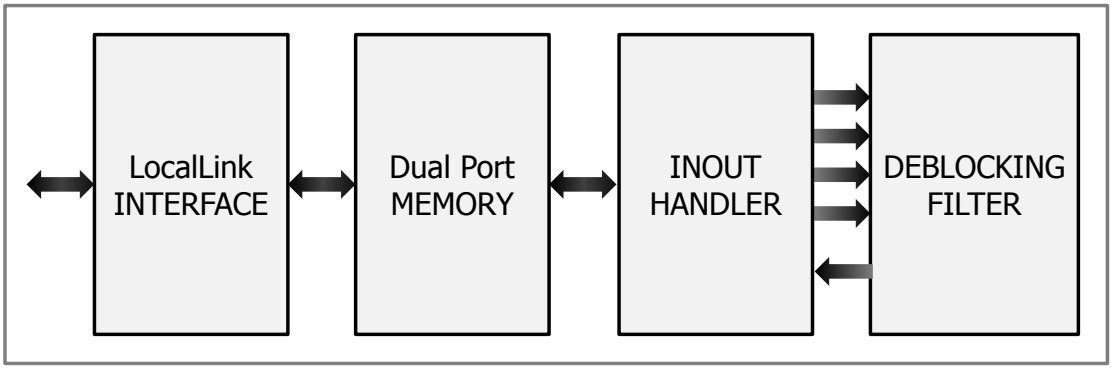


Figura 91. Arquitectura para la implementación FPGA del DF

4.5.2. Interfaz LocalLink

Debido a la importancia de las comunicaciones entre el acelerador *hardware* y el procesador, se presenta la interfaz utilizada en esta investigación. LocalLink [101] es una especificación de protocolo de comunicación propuesto por Xilinx para transacciones de datos en forma de tramas. En particular, es de interés para este trabajo debido a que la conexión entre los bloques DMA del bloque de procesador y *deblocking filter* al que se conecta se hace con esta especificación. Se trata de un protocolo síncrono y punto a punto.

Una interfaz LocalLink se compone de un conjunto de señales de control, un puerto de datos, además de las señales de reloj y de *reset*. La comunicación es *simplex* (en un solo sentido), siendo necesarios dos canales LocalLink simétricos para conseguir una comunicación *full-duplex*.

Las señales de control son en ambos sentidos para cada uno de los canales, ya que, como se comentará posteriormente, el destino de la transmisión avisará a la fuente si está o no preparado para la recepción de datos. En la figura 92 se muestra un ejemplo básico de una comunicación LocalLink en un único sentido.

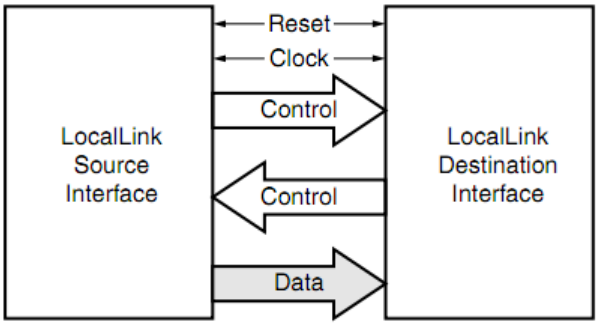


Figura 92. Ejemplo de comunicación LocalLink

La especificación LocalLink es genérica, pudiéndose parametrizar para cada uso específico. De esta forma, una trama de LocalLink está formada por una cabecera, una carga útil o *payload* y una cola, siendo la cabecera y la cola opcionales.

En la tabla 15 se enumeran las señales que obligatoriamente deben aparecer en un canal unidireccional LocalLink genérico.

Tabla 15. Señales obligatorias de una interfaz LocalLink

Señal	Dirección	Descripción
DATA	Fuente → Destino	Bus de datos de ancho parametrizable.
SRC_RDY_N	Fuente → Destino	Señal activa a nivel bajo. Indica si la fuente está preparada para enviar datos.
DST_RDY_N	Fuente ←Destino	Señal activa a nivel bajo. Indica si el destino está preparado para recibir datos.
SOF_N	Fuente → Destino	Señal activa a nivel bajo. Indica el comienzo de una trama.
EOF_N	Fuente → Destino	Señal activa a nivel bajo. Indica el fin de una trama.

El modelo SystemC de la interfaz LocalLink se muestra a continuación.

```
SC_MODULE(ll_interface) {
    // Generic Ports
    sc_in <bool> clk;   // clock input
    sc_in <bool> rst_n; // reset input
    // LocalLink Interface TX (PowerPC -> IP Core)
    sc_out<bool >             ll_dst_rdy_tx_n;
    sc_in <bool >             ll_src_rdy_tx_n;
    sc_in <bool >             ll_sof_tx_n;
    sc_in <bool >             ll_sop_tx_n;
    sc_in <bool >             ll_eof_tx_n;
    sc_in <bool >             ll_eop_tx_n;
    sc_in <sc_uint<32> > ll_data_tx;
    sc_in <sc_uint<4> >  ll_rem_tx;

    // LocalLink Interface TX (IP Core -> PowerPC)
    sc_in <bool >             ll_dst_rdy_rx_n;
    sc_out<bool >             ll_src_rdy_rx_n;
    sc_out<bool >             ll_sof_rx_n;
    sc_out<bool >             ll_sop_rx_n;
    sc_out<bool >             ll_eof_rx_n;
    sc_out<bool >             ll_eop_rx_n;
    sc_out<sc_uint<32> > ll_data_rx;
```

```
sc_out<sc_uint<4> >  ll_rem_rx;

// Memory Interface
sc_out<sc_uint<17> > addr;
sc_out<sc_uint<32> > data_wr;
sc_out<bool >        ce;
sc_out<bool >        we;
sc_in <sc_uint<32> > data_rd;

// Control Interface
sc_in <bool >             read_handler_request;
sc_out<bool >             read_handler_validate;
sc_in <bool >             write_handler_request;
sc_out<bool >             write_handler_validate;
```

4.5.2.1. Interfaz LocalLink de un bloque DMA

Como se indicó anteriormente, los bloques DMA del procesador utilizan el protocolo de comunicación LocalLink para enviar y recibir flujos de datos en forma de tramas entre procesador y acelerador *hardware*. En particular, cada bloque DMA tiene dos interfaces LocalLink, uno para transmisión y otro para recepción, además de las señales de reloj y *reset* comunes a ambas interfaces. En la figura 93 se muestra el diagrama de bloques de un canal DMA.

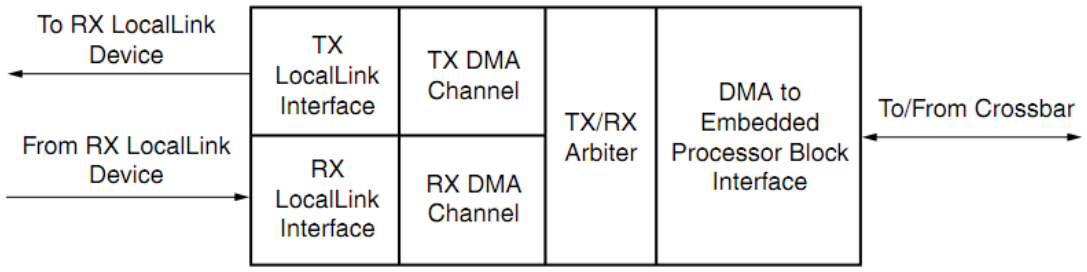


Figura 93. Interfaces LocalLink de un bloque DMA

El dispositivo de transmisión del bloque DMA recoge los datos de una o varias regiones de la memoria, creando la carga útil a transmitir por la interfaz LocalLink de transmisión. Por su lado, el dispositivo de recepción recibe los datos desde la interfaz LocalLink de recepción y los almacena en una o varias regiones de la memoria. Estas zonas de memoria se señalan por unos registros de control llamados descriptores.

Cada canal DMA se controla por descriptores independientes, y sus correspondientes estructuras de datos se inicializan por el PowerPC 440 de la FPGA Virtex-5 antes de que las operaciones del DMA comiencen. Entre otras cosas, estos descriptores controlan cuántos datos serán transferidos y la localización de los datos en el sistema de memoria.

Los descriptores pueden estar encadenados, permitiendo formar una secuencia de bloques de memoria separados, que se combinan en un único paquete al ser transmitido. De la misma forma, se puede dividir el paquete recibido en una secuencia de bloques de memoria separados.

El PowerPC inicializa el DMA creando una secuencia de descriptores en memoria, y luego escribiendo la dirección del primer descriptor en el registro DCR de Puntero de Descriptor Actual. Finalmente, la CPU comienza las operaciones del DMA escribiendo la dirección del último descriptor de la secuencia en el registro DCR de Puntero de descriptor de cola.

Esta última escritura hace que el DMA busque un nuevo descriptor de la dirección apuntada por el registro Puntero al Descriptor Actual. En el caso del canal de transmisión, el DMA comienza por buscar los datos de la dirección indicada en el descriptor y comienza el proceso de creación y envío de un paquete de datos. Después de transmitir todos los datos indicados por el descriptor actual, el DMA busca el siguiente descriptor, si existe, y continúa enviando los datos indicados por este descriptor.

En el caso del canal de recepción, el DMA espera por los datos recibidos del periférico externo, y comienza a copiarlos en la región de memoria apuntada por el descriptor. Si se reciben más datos, se busca el nuevo descriptor y se escriben en la región indicada por este. Este proceso continua hasta el final de la carga útil recibida. En la figura 94 se muestra un diagrama con las señales que aparecen en la interfaz LocalLink de transmisión del DMA.

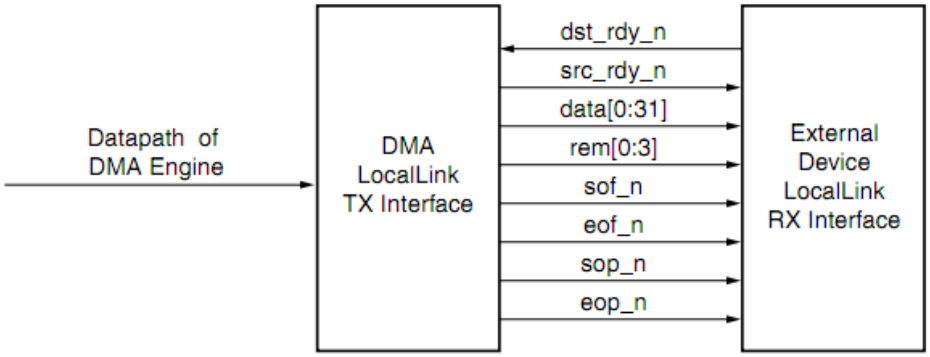


Figura 94. Diagrama de bloques de la interfaz de transmisión LocalLink de un DMA

4.5.3. Flujo de diseño

La figura 95 muestra el flujo de diseño completo usado para realizar el diseño del DF en la plataforma FPGA. Tal como se ha explicado anteriormente, el flujo de diseño incluye las siguientes etapas:

4.5.3.1. Análisis del diseño

En la fase de análisis del diseño se realiza la obtención del perfil de ejecución del decodificador original. Con ello se pretende obtener información acerca de la complejidad de la aplicación, su modularidad, así como de características de coste computacional y temporal de las distintas funciones que la componen. Esta información, junto a un estudio conceptual de la dependencia de datos, facilita durante la fase de diseño la toma de decisiones relacionados con

la partición, las necesidades de comunicación y su conectividad, produciendo como resultado la jerarquía del diseño.

Por tanto, después de una etapa de análisis del decodificador OpenSVC en la que se identifican los núcleos con mayor carga computacional, se decide realizar la implementación *hardware* el DF. Para explorar en esta investigación la aceleración del ciclo de diseño se ha seleccionado una metodología de diseño basada en síntesis de alto nivel con algunas aportaciones adicionales. Al DF se le trata por tanto como sistema.

4.5.3.2. Captura del diseño

Durante esta etapa se realiza la adaptación del *software* de referencia a los módulos encapsulados en lenguaje SystemC, realizando para ello las particiones que se estimen oportunas. Las funciones en lenguaje C del DF se han agrupado por su funcionalidad y se han encapsulado como módulos SystemC. La mayoría de las decisiones de diseño se han realizado durante esta fase de síntesis de alto nivel, incluyendo el análisis del impacto en el uso de recursos y latencias.

4.5.3.3. Simulación ESL

Durante esta simulación se verifica que la funcionalidad es correcta con respecto al modelo de referencia. Para verificar el funcionamiento se han extraído las entradas y salidas del modelo de referencia y se han almacenado en ficheros. A partir de aquí, mediante un *testbench* escrito en SystemC se han cargado los fotogramas no filtrados, los parámetros del filtro y se obtienen los fotogramas filtrados, que se comprueban en su integridad.

4.5.3.4. Síntesis de alto nivel

Una vez verificado el sistema, se pasa a adaptar su código al de entrada a la herramienta CtoS que se encarga de generar la descripción del sistema a nivel RTL en Verilog HDL.

La síntesis de alto nivel incluye las siguientes operaciones: romper bucles combinacionales, realizar la copia *inline* de las funciones, aplanar *arrays* para su implementación en registros, asignar otros *arrays* a memoria RAM, hacer la planificación temporal, hacer la asignación de registros, y generar finamente el código RTL.

A continuación se indican algunas estrategias realizadas durante la síntesis de alto nivel. Para esta implementación, la mayor parte de las variables del modelo funcional se implementan como registros, excepto *arrays* de gran tamaño que se implementan como memorias. Las funciones de comunicación con memoria (lectura/escritura) se han transformado mediante *inline* para facilitar el acceso concurrente a las variables que se han implementado como bloques de memoria. De igual forma, se facilita el uso de bloques DSP para mapear operaciones aritméticas (multiplicación-acumulación) en los bloques DSP de la FPGA. Se ha permitido la relajación de latencias para introducir ciclos adicionales (de forma limitada) con el objetivo de cumplir las restricciones temporales impuestas (ciclo de reloj de 10 ns). Estas restricciones para relajar la latencia se han evitado en las interfaces de comunicación externas porque requieren protocolos precisos a nivel de ciclo (*cycle-accurate*) y en la comunicación con las memorias de bloque. Después de estos pasos se obtiene una descripción RTL sintetizable del DF.

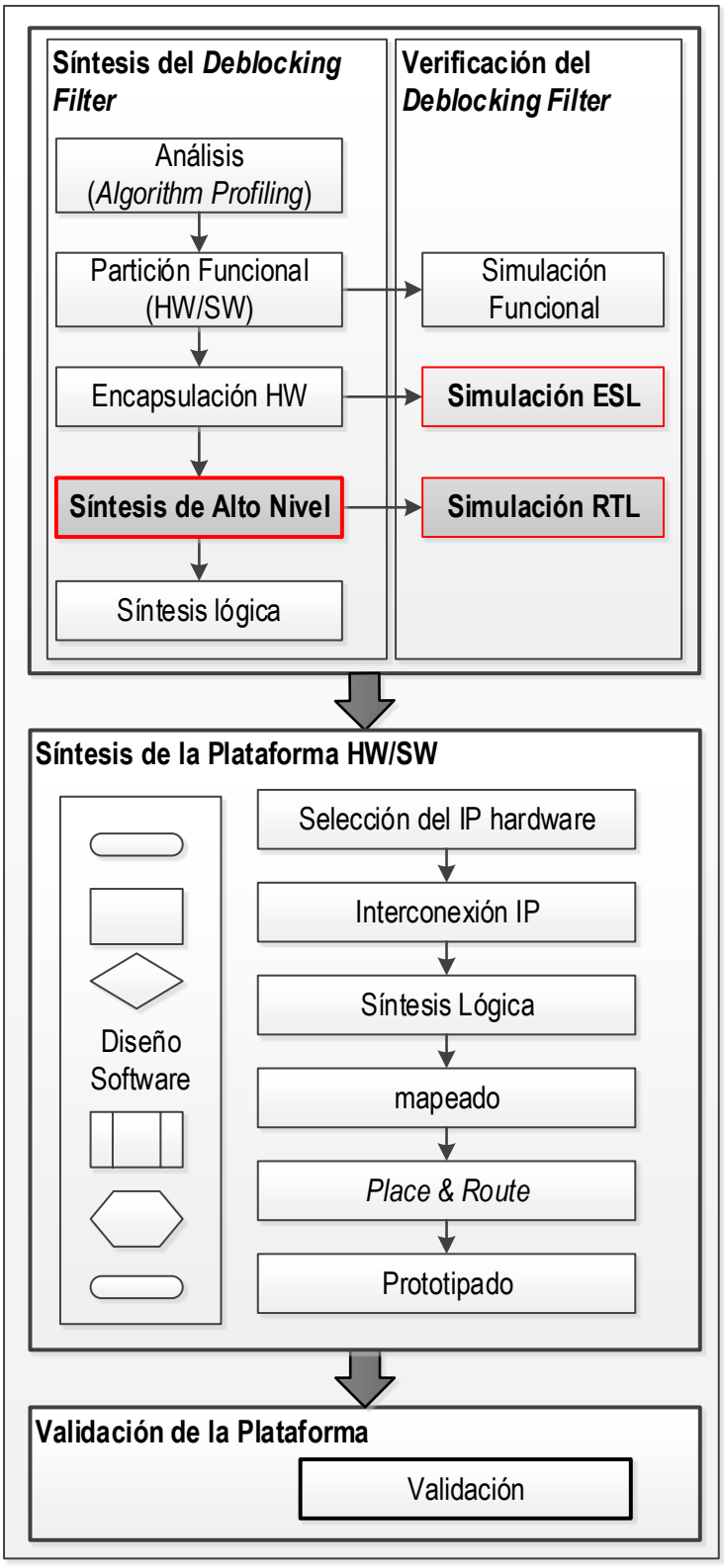


Figura 95. Flujo de diseño finalmente establecido para la FPGA

A continuación se muestra un extracto de la implementación de las estrategias de síntesis para CtoS en scripts del lenguaje de comandos para herramientas Tcl.

```
set modulo d_filter
source scripts/ctos_setup.tcl

set top_path "/designs/$modulo"
set modules $top_path/modules

set combo_loops [find_combinational_loops]
foreach loop $combo_loops {
    puts "breaking loop: $loop"
    break_combinational_loop  $loop
}
inline \
    $modules/d_filter_luma_core/behaviors/write_work_luma_word \
    $modules/d_filter_luma_core/behaviors/write_filtered_luma_word \
    ...
    $modules/d_filter_param_clip/behaviors/process_0

flatten_array  \
    $modules/d_filter_param_bs_svc/arrays/vec_y_up  \
    $modules/d_filter_chroma_core/arrays/q_samples \
    ...
    $modules/d_filter_param_bs_svc/arrays/bs_boundary_hor_luma

allocate_builtin_ram
    $modules/d_filter_param_clip/arrays/alpha_value_up_ver4_luma
...
allocate_builtin_ram
    $modules/d_filter_param_clip/arrays/alpha_value_up_ver4_chroma0
...

use_dsp /designs/$modulo

# Modules with relax_latency true
set modules_rlt [list d_filter_param_clip d_filter_param_bs_svc
    d_filter_luma_core d_filter_chroma_core]
# Modules with relax_latency false
set modules_rlf [list d_filter_interf ram8x1024 ram8x128 ram64x32
    ramMAX_REF_BITSx128 ram8x512 ram8x32 ram8x16 ram8x64]
foreach m $modules_rlt {
  set modulo_rlt $top_path/modules/$m
  foreach beh [find $modulo_rlt/behaviors/*] {
    set_attr relax_latency "true" $beh
  }
  schedule -verbose -effort high -passes 200 $modulo_rlt
  allocate_registers $modulo_rlt
}
foreach m $modules_rlf {
  set modulo_rlf $top_path/modules/$m
  foreach beh [find $modulo_rlf/behaviors/*] {
    set_attr relax_latency "false" $beh
  }
  schedule -verbose -effort high -passes 200 $modulo_rlf
  allocate_registers $modulo_rlf
}
write_rtl -recursive -o [concat ./[string trim $model]]
    $top_path/modules/$modulo
report_summary > $ctos_log/summary.txt
report_resources > $ctos_log/resources.txt
report_timing > $ctos_log/timing.txt
report_area -detail > $ctos_log/area.txt
report_registers -detail > $ctos_log/registers.txt
```

4.5.3.5. Simulación RTL

Para comprobar la corrección de la traducción se realiza una verificación RTL reutilizando el mismo *testbench* con el que se hizo la primera comprobación de funcionalidad en SystemC. Para realizar la simulación RTL del diseño se utiliza también el mismo entorno de simulación que el utilizado a nivel ESL. La herramienta de síntesis de alto nivel genera un adaptador (*wrapper*) que adapta los tipos de datos abstractos usados para el modelo funcional a un modelo RTL, preciso a nivel de pines. Durante estas transformaciones del diseño es necesario prestar atención a la comunicación con los bloques de memoria disponibles en la FPGA manteniendo el protocolo de acceso síncrono soportado por las RAMB de Xilinx. Un modelo de comunicación optimizado para las memorias evita tiempos largo de síntesis y de depuración para las síntesis de interfaces de memoria. A continuación se muestra el ejemplo del uso del *wrapper* para incluir el modelo RTL Verilog y/o el modelo original SystemC.

```
#include "top_testbench.h"
#include "d_filter_top.h"
#include <systemc.h>

#ifdef CTOS_MODEL
#include "d_filter_top_ctos_wrapper.h"
#endif

//------------------------------------------------------------------------
    int sc_main (int argc, char *argv [ ] ) {

    sc_signal<bool>          rst_n;

    // LocalLink Interface TX (PowerPC -> IP Core)
    sc_signal<bool>                 ll_dst_rdy_tx_n;
    sc_signal<bool>                 ll_src_rdy_tx_n;
    ...
    sc_signal<sc_uint<4> >  ll_rem_tx;

    // LocalLink Interface RX (IP Core -> PowerPC)
    sc_signal<bool >          ll_dst_rdy_rx_n;
    sc_signal<bool >          ll_src_rdy_rx_n;
    ...
    sc_signal<bool >          ll_eop_rx_n;

    sc_clock clk("clk", 10, SC_NS, 0.5, 0.0, SC_PS);

#ifdef NC_SYSTEMC
#include "d_filter_top_main_ncsc_names.h"
#endif

#ifdef CTOS_MODEL
    d_filter_top_ctos_wrapper
    d_filter_top_inst("d_filter_top_inst","_rtl");
#else
    d_filter_top d_filter_top_inst("d_filter_top_inst");
#endif

    d_filter_top_inst.clk(clk);
    d_filter_top_inst.rst_n(rst_n);
    d_filter_top_inst.ll_dst_rdy_tx_n(ll_dst_rdy_tx_n);
    ...
    d_filter_top_inst.ll_rem_rx(ll_rem_rx);


    top_testbench top_testbench_inst("top_testbench_inst");
```

```
    top_testbench_inst.clk(clk);
    top_testbench_inst.rst_n(rst_n);
    top_testbench_inst.ll_dst_rdy_tx_n(ll_dst_rdy_tx_n);
    ...
    top_testbench_inst.ll_rem_rx(ll_rem_rx);

    sc_start();

#ifndef NC_SYSTEMC
    system("pause");
#endif

    return 0;
};
```

4.5.3.6. Síntesis lógica

El flujo de síntesis continúa con la síntesis lógica para el dispositivo FPGA. Esta síntesis se ha realizado con Synopsys Synplify Premier, obteniéndose un fichero EDIF con la implementación del diseño. Se ha seguido una aproximación *top-down* como estrategia de síntesis lógica y se ha permitido a la herramienta de síntesis que obtenga el diseño más rápido (120,3 MHz – 8,315 ns). La etapa final de la síntesis lógica es la obtención de la implementación completa en el entorno de desarrollo de Xilinx (ISE, XPS, PlanAhead y SDK), incluyendo tanto el DF como la plataforma de validación. La síntesis lógica está guiada por restricciones temporales con el objetivo de obtener la frecuencia máxima de funcionamiento.

```
set modulo d_filter_top

set mypath [pwd]
...

set_option -technology Virtex5
set_option -part XC5VFX70T
set_option -speed_grade -1
set_option -package FF1136
...
set_option -frequency auto
set_option -disable_io_insertion 1

set_option -run_prop_extract 1
set_option -pipe 1
set_option -update_models_cp 0
set_option -retiming 0
set_option -no_sequential_opt 0
set_option -fixgatedclocks 3
set_option -fixgeneratedclocks 3
set_option -enable_prepacking 1

set_option -resource_sharing 1
set_option -multi_file_compilation_unit 1

add_file -verilog "../model/d_filter_top_rtl.v"
add_file -verilog "../model/d_filter_param_bs_svc_rtl.v"
...
add_file -verilog "../model/d_filter_rtl.v"

...

#VIF options
```

```
set_option -write_vif 1

#automatic place and route (vendor) options
set_option -write_apr_constraint 1

#set result format/file last
project -result_file "./rev_1/d_filter_top.edf"

project -run synthesis
project -save $mypath/[concat [string trim $modulo].prj]

set_option -num_startend_points 5
set_option -reporting_margin -1.0
set_option -reporting_filename [concat [string trim $modulo].ta]
set_option -reporting_output_srm 0
project -run timing
```

4.5.3.7. Validación del diseño

Esta sección describe la parte final del flujo de síntesis de alto nivel (figura 95): la arquitectura de la plataforma de validación diseñada usando la herramienta de Xilinx Platform Studio (XPS) y la metodología de validación seguida.

El procesador PowerPC empotrado en la FPGA realiza tareas auxiliares: genera los fotogramas de entrada para el DF, recibe las muestras filtradas y realiza la conversión 4:2:0 a 4:4:4 y la conversión del espacio de colores de YUV a RGB para la visualización del vídeo en una pantalla TFT. La figura 96 muestra la arquitectura de la plataforma de validación. Los bloques del sistema se describen a continuación.

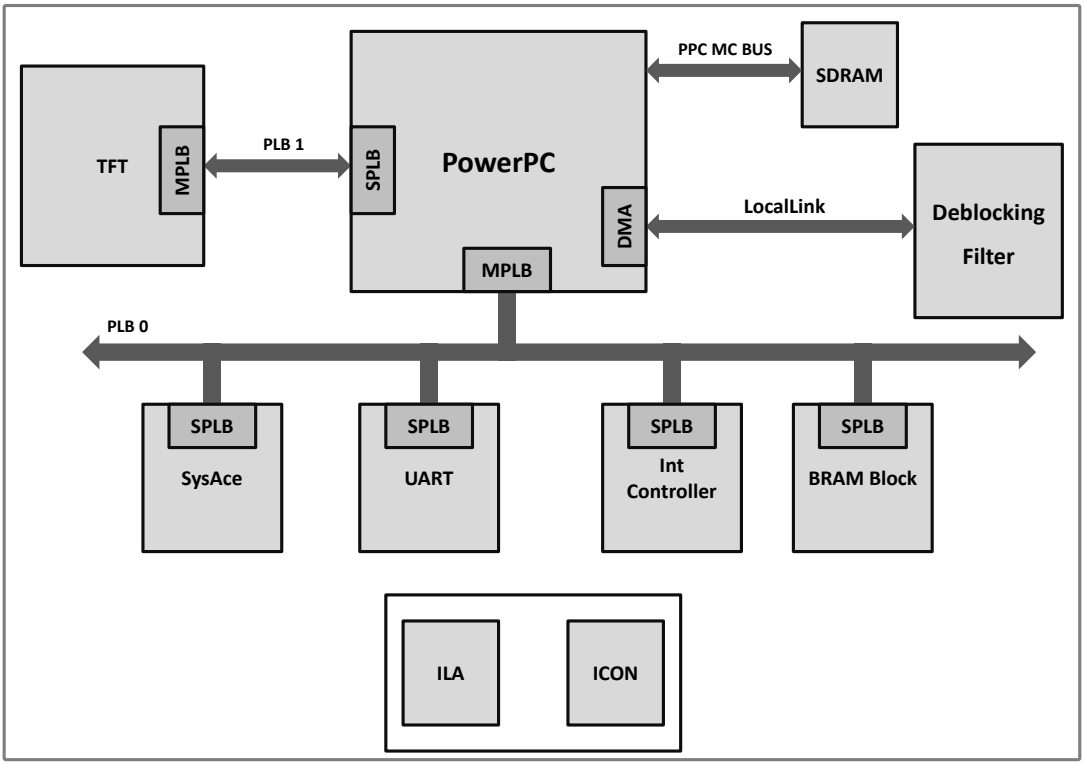


Figura 96. Diagrama de bloques de la plataforma de validación

- **Procesador Empotrado PowerPC**. Ejecuta el *software* empotrado, creando las estructuras de datos de entrada necesarias para el DF y recibiendo las imágenes

filtradas. Esta comunicación se realiza mediante la interfaz LocalLink y su correspondiente DMA.

- **Controlador de Memoria SDRAM**. El controlador de memoria permite a la FPGA usar la memoria externa, incorporada en la tarjeta de la FPGA, como memoria principal del PowerPC440. El bloque del procesador empotrado se conecta a este controlador mediante el *Memory Controller Interface* (MCI).

- **SysAce**. El controlador XPS System ACE Interface representa la interfaz entre el bus de sistema en chip PLB y la interfaz del microprocesador (*Microprocessor Interface* - MPU) del periférico System ACE Compact Flash. Esta tarjeta de memoria se usa como sistema de ficheros para almacenar los datos de entrada al DF y para escribir las muestras filtradas durante la etapa de validación.

- **Transceptor Serie UART**. Se usa una comunicación serie para monitorizar el sistema. Por tanto, cualquier mensaje del sistema se enviará mediante el puerto serie al sistema de monitorización. Este periférico, como otros definidos en la plataforma, presenta una interfaz esclava PLB y varios atributos configurables como los baudios de la transmisión, paridad, etc.

- **Controlador de interrupciones.** El procesador empotrado PowerPC440 posee una única entrada de interrupción. Por tanto, es necesario incluir un controlador con varias entradas de interrupción desde los dispositivos periféricos y que genere una señal de interrupción hacia el procesador. El control, la activación y los registros de máscara se acceden a través de una interfaz PLB esclava.

- **Integrated Logic Analyzer (ILA).** El bloque ILA se implementa en la lógica configurable y se utiliza para mostrar señales internas del sistema, actuando como un analizador lógico integrado. Se trata de un bloque síncrono y por tanto debe cumplir las restricciones temporales del resto del diseño. Las señales del sistema se conectan al bloque ILA y se muestrean a la frecuencia del diseño en tiempo real. Es necesario planificar el número de señales a muestrear y el tamaño ocupado por la memoria de muestras (las muestras se guardan en memoria interna BRAM de la FPGA). La comunicación con los bloques ILA se realiza a través del puerto JTAG de la FPGA usando el bloque de control ICON, que realiza la tarea de comunicación entre los bloques ILA y el bloque *Boundary Scan* JTAG de la FPGA.

- **Block RAM**. Implementa la memoria de programa que almacenan las instrucciones a ejecutar en el microprocesador.

- **Controlador de *display* TFT.** Este bloque es responsable de generar las señales de control para mostrar el video en una pantalla TFT usando una interfaz DVI. Para realizar esta tarea, reserva un área de memoria de tamaño doble al de un fotograma para implementar un esquema de memoria de doble *buffering,* almacenado muestras en formato RGB. Este bloque tiene una interfaz maestra PLB que permite acceder a la memoria SDRAM principal *off-chip* a través de una interfaz PLB esclava del procesador.

4.5.4. Flujo de datos

El Flujo de datos durante la validación sobre la plataforma definida parte desde los datos en la tarjeta Compact Flash (CF) hasta que se muestran en la TFT una vez filtrados:

1. El primer flujo de datos consiste en la lectura de los datos de un *frame* desde la memoria CF externa y su almacenamiento en memoria principal, en este caso DDR.
2. A continuación el PowerPC ordena los datos y los envía al DF mediante la interfaz de LocalLink DMA definida. Los datos se filtran en el IP acelerador DF sintetizado.
3. La salida filtrada será almacenada por el bloque DMA en memoria principal.
4. Por último, el *frame* o trama de datos es devuelto a memoria CF y presentado en pantalla TFT.

4.5.5. *Software* empotrado

Conjuntamente con la plataforma *hardware*, es necesario definir la plataforma *software*, es decir, el conjunto de librerías, *drivers*, y sistemas operativos, disponibles para el diseño de las tareas auxiliares a ejecutar en el procesador empotrado del sistema DF.

La librería de soporte estándar de C consta de la librería *libc*, que contiene las funciones *estándar* de las librerías *stdio*, *stdlib* y *string*. La librería matemática *libm* provee las rutinas estándar de operaciones aritméticas. En Virtex-5 se proporcionan dos posibles sistemas operativos para el PowerPC440: la plataforma *Standalone* y *Xilkernel*.

El *software* está organizado en las siguientes funciones:

- **XLlDma_Initialize().** Es la función que inicializa un bloque DMA y lo asocia a una variable que será el identificador virtual del bloque *hardware*.
- **XLlDma_BdRingCreate().** Crea un anillo de descriptores. Un descriptor es una estructura de datos que se utiliza para identificar transferencias a través del DMA. Cada vez que se desea enviar una ráfaga de datos a través del DMA, se usa uno de los descriptores libres del anillo como medio de comunicación con el DMA. En el descriptor incluye datos tales como dirección de memoria donde están los datos, longitud de la ráfaga, etc.
- **XIntc_Connect().** Es la encargada de vincular rutinas de interrupción a los vectores de interrupción correspondientes. Debido a que el DMA realiza las transferencias en paralelo a la ejecución de otras tareas de la CPU, es necesario disponer de interrupciones cuando una transferencia haya finalizado.
- **XIntc_Enable().** Habilita un vector de interrupción en particular.
- **RxHandler()** y **TxHandler().** Son las rutinas de interrupción de las interrupciones de trama recibida y trama enviada con éxito.

- **XLlDma_BdRingIntEnable().** Indica al DMA que genere interrupciones cuando termine de enviar o recibir una trama.
- **sysace_fopen(), sysace_fclose(), sysace_fread(), sysace_fwrite().** Son las funciones utilizadas para leer y escribir en la CompactFlash.
- **convert_to_444().** Esta función se encarga de pasar de formato 4:2:0 a una representación 4:4:4 donde las tres componentes coinciden en dimensiones.
- **convert_to_RGB().** Pasa de una representación YUV a una representación RGB para su almacenamiento en el buffer de representación.

4.5.6. Resultados de síntesis

En los siguientes apartados se muestran los resultados obtenidos de la implementación FPGA realizada del DF y la arquitectura de validación.

4.5.6.1. Plano de base de la FPGA

La figura 97 presenta la organización de la implementación realizada sobre el dispositivo FPGA de los diferentes bloques que conforman la implementación del DF para el decodificador OpenSVC en el dispositivo XC5VFX70T de Xilinx.

4.5.6.2. Recursos FPGA

La tabla 16 resume los recursos utilizados por el sistema que implementa la plataforma una vez se han implementado en la FPGA.

4.5.6.3. Rendimiento y frecuencia

Para medir el rendimiento del sistema, se considera como comienzo de la decodificación el instante en que el primer dato atraviesa la interfaz LocalLink de entrada. Por ello se consideran incluidos los retardos de comunicación para cada fotograma.

El tiempo transcurrido entre la fuente de muestras y el DF, y entre el DF y el receptor de imágenes filtradas también se ha tenido en cuenta a la hora de presentar los datos de las tablas anteriores. Esta comunicación se realiza a través de un bus LocalLink de 32 bits a una frecuencia de bus de 100 MHz, lo que produce un ancho de banda teórico de 3.2 Gbps. El tiempo de comunicación es aproximadamente el 5% del tiempo total, mientras que el 95% del tiempo restante se utiliza para decodificar el fotograma (figura 98). El DF puede decodificar 130 fps de un vídeo en formato QCIF (176x144 pels) y 37 fps de un vídeo en formato CIF (352x288 pels) con una latencia promedio de 6.500 ciclos de reloj por macrobloque (tabla 17).

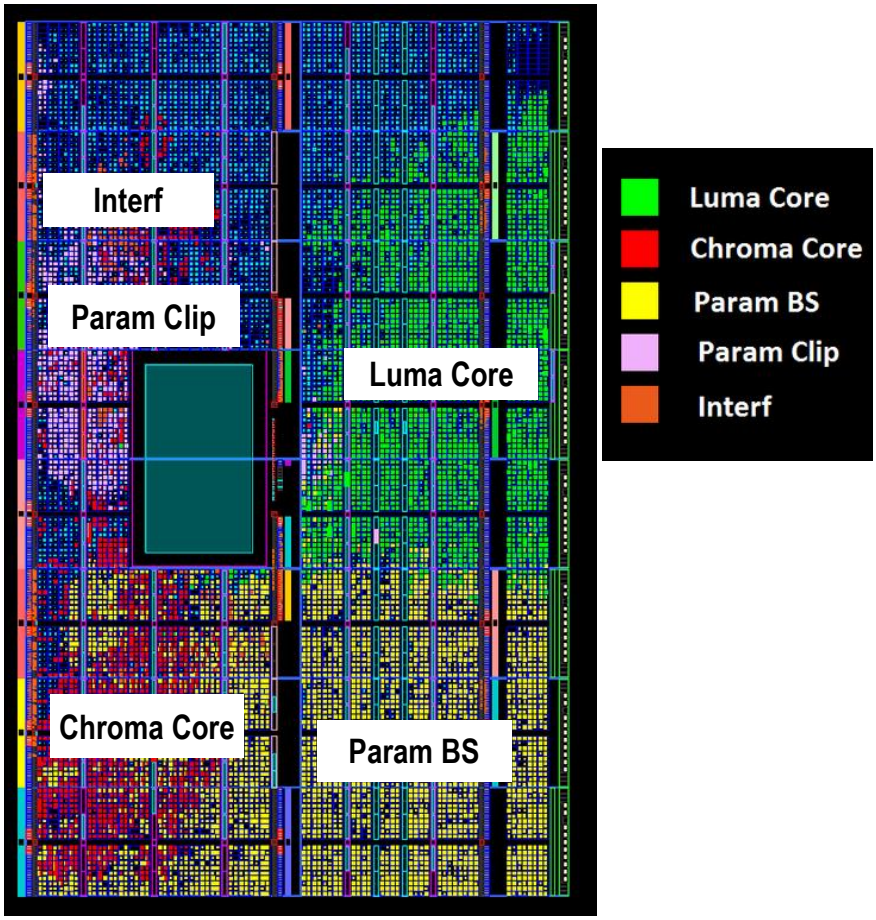


Figura 97. Organización de la FPGA mostrando los diferentes bloques: Núcleos de procesamiento Luma y Croma, Param_BS, Param_Clip e Interface

Tabla 16. Recursos utilizados en la implementación FPGA

	LUTs	Registers	DSPs	BRAM
d_filter	27.192	18.359	1	10
Interface	637	446	0	0
param_BS	9.552	9.523	0	0
param_clip	1.911	877	1	0
luma_core	9.832	4.170	0	1
chroma_core	4.470	3.099	0	1
ll_interface	336	414	0	0
inout_handler	6.599	3.413	2	0
Memory	2	0	0	96

Tabla 17. Resultados de prestaciones

Parámetro	Valor
Tecnología	FPGA Virtex5 (CMOS 65 nm)
Frecuencia	100 MHz
Rendimiento (Throughput)	QCIF (176x144): @130 fps - CIF (352x288): @37 fps
Latencia	6,500 ciclos/MacroBlock

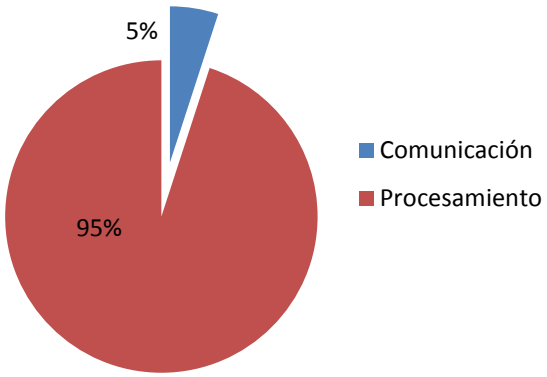


Figura 98. Distribución de latencias totales del sistema: comunicación y procesamiento

4.5.6.4. Análisis del Consumo de potencia

Para realizar la medida del consumo de potencia del DF así como del resto del sistema de prototipado, implementación y validación, se utiliza la herramienta XPower Analyzer de Xilinx. Para aumentar el nivel de precisión de la estimación realizada se ha optado por utilizar el diseño completamente ruteado (en formato NCD), anotando la actividad de los nodos obtenida desde la simulación RTL, con la información de decodificación de entrada (en formato VCD). La tabla 18 muestra el consumo de potencia del DF. La tabla 19 compara los resultados obtenidos con otros obtenidos de publicaciones científicas que describen implementaciones de DF de bajo consumo. La información debe escalarse en función de la tecnología usada, que afecta al consumo de potencia. En [161] se indica que la reducción de potencia entre tecnologías de 130nm y de 65nm es del 50%.

Tabla 18. Consumo de potencia del DF

Bloque	Potencia (mW)
DF	18,69

Tabla 19. Comparación de Potencia Disipada

Referencia	Tecnología	Frequencia (MHz)	Potencia (mW)
[148] **Nadeem et al.**	Xilinx Virtex-2 (CMOS 130nm)	76	43,00
[150] **Parlak et al.**	Xilinx Virtex-2 (CMOS 130nm)	72	259,13
Esta implementación	Xilinx Virtex-5 (CMOS 65nm)	100	18,69

4.5.7. Prototipado del sistema

La figura 99 muestra el prototipo del sistema implementado en una placa de desarrollo Xilinx ML507 que incluye el dispositivo XC5VFX70T.

Figura 99. Prototipo del sistema funcionando en una placa Xilinx ML507

4.6. Implementación ASIC

La simulación funcional en SystemC garantiza la corrección del DF para los casos de test. De la misma forma, el prototipo implementado sobre la FPGA representa la validación de la arquitectura del DF para decodificar vídeo en formatos QCIF y CIF. Con objeto de explorar una implementación ASIC del decodificador SVC completo se realiza la implementación del DF siguiendo una metodología de diseño para tecnologías sub-micra.

Se ha realizado una implementación ASIC del DF en tecnología UMC CMOS 65nm en su variante de SP (Standard Performance) con 7 capas de metal. La elección de esta tecnología viene motivada para comparar los resultados de prestaciones obtenidas y potencia con el dispositivo FPGA Virtex-5 implementado en CMOS 65nm. La implementación se realiza usando células estándar de UMC.

4.6.1. Flujo de diseño

La figura 100 muestra el flujo de diseño definido para la implementación ASIC del *Deblocking filter*.

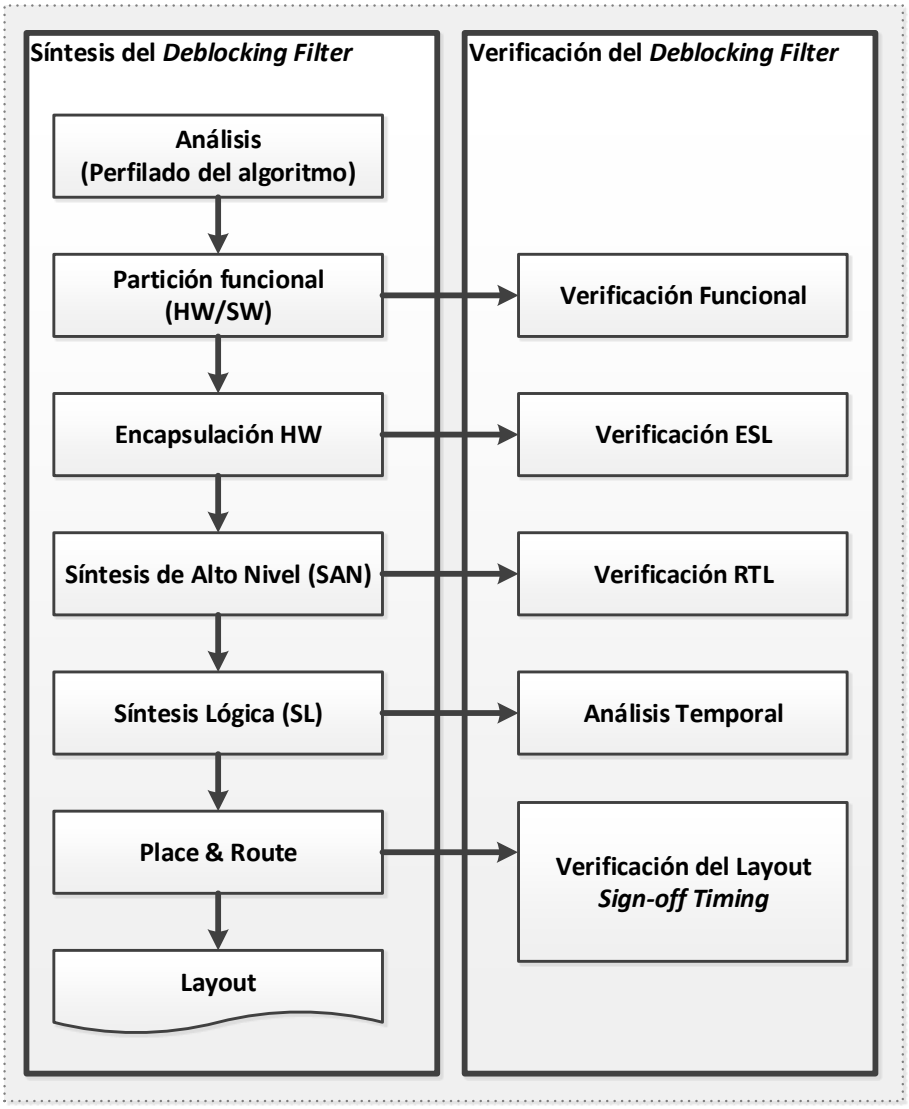


Figura 100. Flujo de diseño finalmente establecido para la implementación ASIC

El flujo de diseño parte de nuevo del modelo SystemC del DF como entrada a la herramienta de síntesis de alto nivel. Las memorias de Chroma, Luma y Control se han obtenido mediante generadores automáticos de *layout* desde las librerías de Faraday. Es necesario capturar la funcionalidad y las características temporales de los bloques de memoria para que se tengan en cuenta en los procesos de síntesis de alto nivel poniendo especial atención a los protocolos de comunicación con los bloques de memoria. Este problema de protocolos se ha resuelto creando adaptadores que permiten elegir distintos tipos de protocolos síncronos de lectura/escritura.

La síntesis de alto nivel se ha realizado desde SystemC usando la herramienta de síntesis CtoS. La herramienta utiliza la información temporal y de área de la librería en formato Liberty de Synopsys. Esta información es utilizada por el analizador temporal interno para obtener información preliminar sobre el mapeado de las unidades funcionales básicas que implementan

las operaciones durante los pasos de planificación temporal y asignación de recursos. Esta librería también se utiliza para realizar el análisis temporal global para determinar el ciclo de reloj. La información recogida en este paso proporciona una primera idea de la ruta crítica del diseño. De igual forma es posible realizar un análisis del área ocupada y de la potencia consumida. Todo ello permite al diseñador tener una noción temprana de las prestaciones del módulo.

Como se realizó en el flujo de diseño para la FPGA, la descripción RTL obtenida del DF durante la síntesis de alto nivel para ASIC se ha simulado para verificar que las decisiones tomadas durante la síntesis de alto nivel producen un módulo que es funcionalmente correcto. Se realiza una co-simulación SystemC-Verilog, en la que el *testbench* se reutiliza el de SystemC y el DUT es ahora el modelo RTL Verilog del ASIC.

La descripción RTL obtenida se usa como entrada de la herramienta de síntesis lógica. Se ha utilizado Synopsys Design Compiler en el flujo de diseño. Debido al objetivo de aumentar las prestaciones en la implementación ASIC, se ha seleccionado un periodo de reloj de 5 ns (200 MHz). De igual forma, se han utilizado las condiciones nominales para la librería de UMC. Para modelar el efecto del retardo en la red del reloj se han incluido restricciones para latencia, tiempos de subida y bajada e incertidumbre del reloj para guiar a la síntesis lógica. Estas restricciones se usarán más tarde para generar el árbol del reloj durante la implementación del diseño físico. La estrategia de síntesis lógica ha sido *top-down*, desde la descripción RTL en Verilog, marcando con atributos "*dont touch*" los bloques de memoria de chroma, luma y de control. A continuación se muestra un extracto del *script* de síntesis del Design Compiler utilizado para sintetizar el diseño, incluyendo las memorias utilizadas. En este caso se han utilizado las condiciones típicas del proceso.

```
# Define the UMC 65nm library
set UMC65 .../d_filter_sp_mm_ll/db
# Define the libraries and search path
set search_path ${UMC65}
# Target library GENERIC_CORE_1D2V and memories
set target_library {.../synthesis/uk65lscsp10bbrccs_100c25_tc_ccs.db
                    .../synthesis/SJKA65_32X64X1CM4_tt1p2v25c.db
                    .../synthesis/SJKA65_128X8X1CM8_tt1p2v25c.db
                    .../synthesis/SJKA65_128X17X1CM4_tt1p2v25c.db
                    .../synthesis/SJKA65_512X8X1CM8_tt1p2v25c.db
                    .../synthesis/SJKA65_1024X8X1CM8_tt1p2v25c.db}

set link_library [concat "*" $target_library]
set symbol_library .../symbol/uk65lscsp10bbrccs.sdb

define_design_lib WORK -path ./WORK

set_units -time ns -resistance kOhm -capacitance pF -voltage V -current mA
set_operating_conditions uk65lscsp10bbrccs_100c25_tc -library  \
uk65lscsp10bbrccs_100c25_tc
set_wire_load_model -name wl0 -library uk65lscsp10bbrccs_100c25_tc
create_clock [get_ports clk]  -period 6.5  -waveform {3.25 6.5}
set_clock_latency 0.065  [get_clocks clk]
set_clock_uncertainty 0.195  [get_clocks clk]
set_clock_transition -max -rise 0.325 [get_clocks clk]
set_clock_transition -max -fall 0.325 [get_clocks clk]
set_clock_transition -min -rise 0.325 [get_clocks clk]
set_clock_transition -min -fall 0.325 [get_clocks clk]
```

4.6.2. Resultados de síntesis

Las prestaciones obtenidas se resumen en la tabla 20. La frecuencia obtenida en esta etapa es de 200 MHz. La figura 101 muestra la distribución de *slacks* obtenida para las rutas críticas (peor caso) del diseño.

Tabla 20. Métricas para la implementación ASIC del DF

Block	Área total (µm2)	Retardo (ns)	Potencia (mW)
d_filter	843.305,16	4,92	9,914
d_filter_interf	14.772,92	3,18	-
d_filter_param_bs_svc	274.334,16	4,89	-
d_filter_param_clip	85.706,34	4,84	-
d_filter_luma_core	179.595,18	4,89	-
d_filter_chroma_core	119.646,10	4,88	-

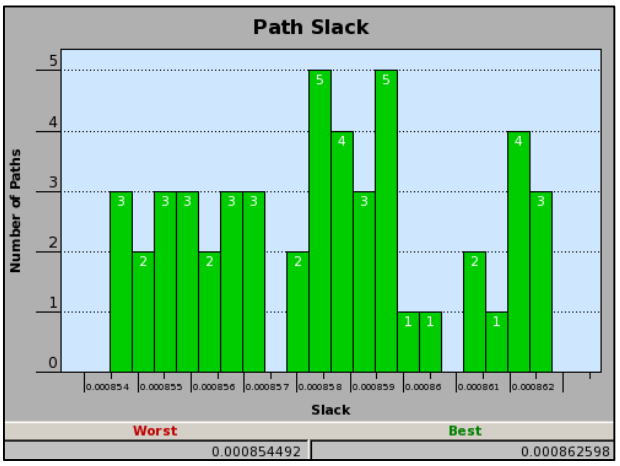


Figura 101. *Path Slack histogram* para una implementación de 5 ns de ciclo de reloj

4.6.3. Implementación física

Una vez que se ha asegurado que el diseño cumple con las restricciones temporales a nivel lógico, se realiza el mapeado y la optimización en dicho nivel y se genera el *netlist* correspondiente en Verilog. Dicho *netlist* se utiliza como entrada a las herramientas de implementación física. Para las tareas asociadas al diseño físico, colocado y ruteado, se utiliza Cadence SoC Encounter. La implementación física se guía haciendo uso de las restricciones temporales extraídas durante la fase de síntesis lógica usando técnicas de convergencia temporal (*timing closure*) para asegurar que al final del flujo de diseño se satisfacen los requerimientos temporales. Los ficheros de restricciones en formato SCF (Synopsys Design Constraint) se exportan desde la herramienta de síntesis lógica (Design Compiler) hasta SoC Encounter para facilitar esta tarea. Se utilizan librerías de células estándar en formato LEF y TLF para incorporar la información física y temporal de las celdas respectivamente.

Los pasos que componen el diseño físico son: realización del plano de base, planificación de alimentaciones y tierras, colocación de células estándar, síntesis del árbol de reloj y ruteado global y detallado. A ello le sigue un análisis temporal completo a nivel de *layout* con objeto de asegurar que las restricciones temporales se cumplen. La frecuencia final obtenida es de 181,8 MHz, ocupando un área de 596.394,4 μm^2. La figura 102 presenta el plano de base del DF. Los bloques de memoria para *Chroma* y *Luma* están ubicadas en las partes superior izquierda y derecha del *layout* respectivamente. La memoria de control, por el contrario está en la parte inferior izquierda del *layout*. Una de las consecuencias de la escalabilidad de SVC es el incremento de la complejidad del bloque que gestiona los parámetros del *boundary strength* lo que le lleva a consumir área adicional en su implementación. El resto de los bloques son los núcleos de procesamiento de *Luma* y *Chroma*, el bloque de gestión de los parámetros de *clipping* y el bloque de interfaces de E/S. El *layout* final se muestra en la figura 103.

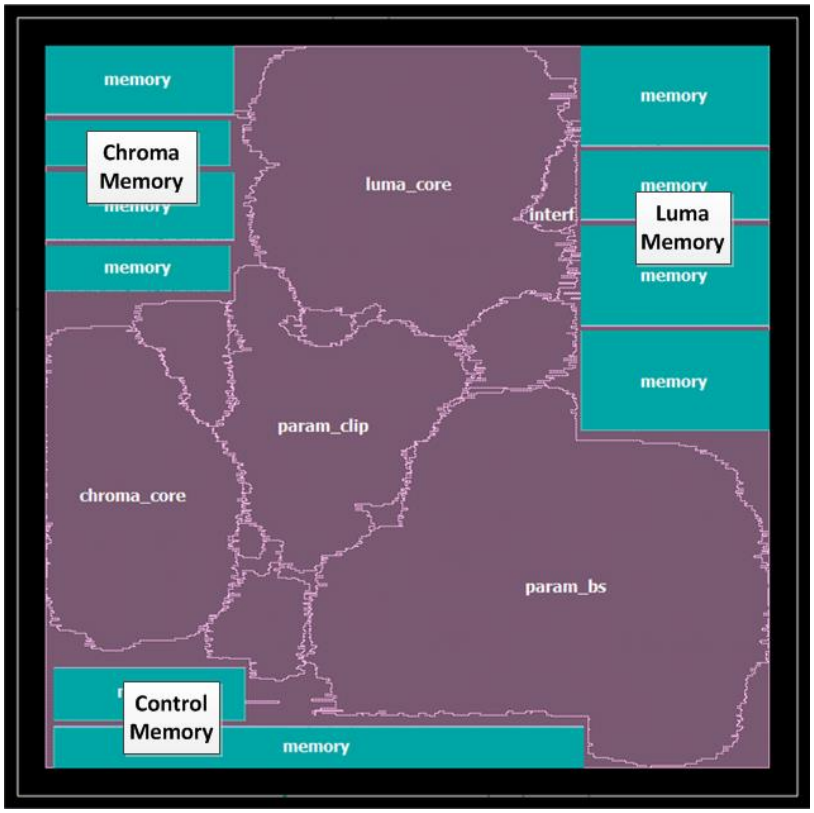


Figura 102. Plano de base de la implementación ASIC del DF

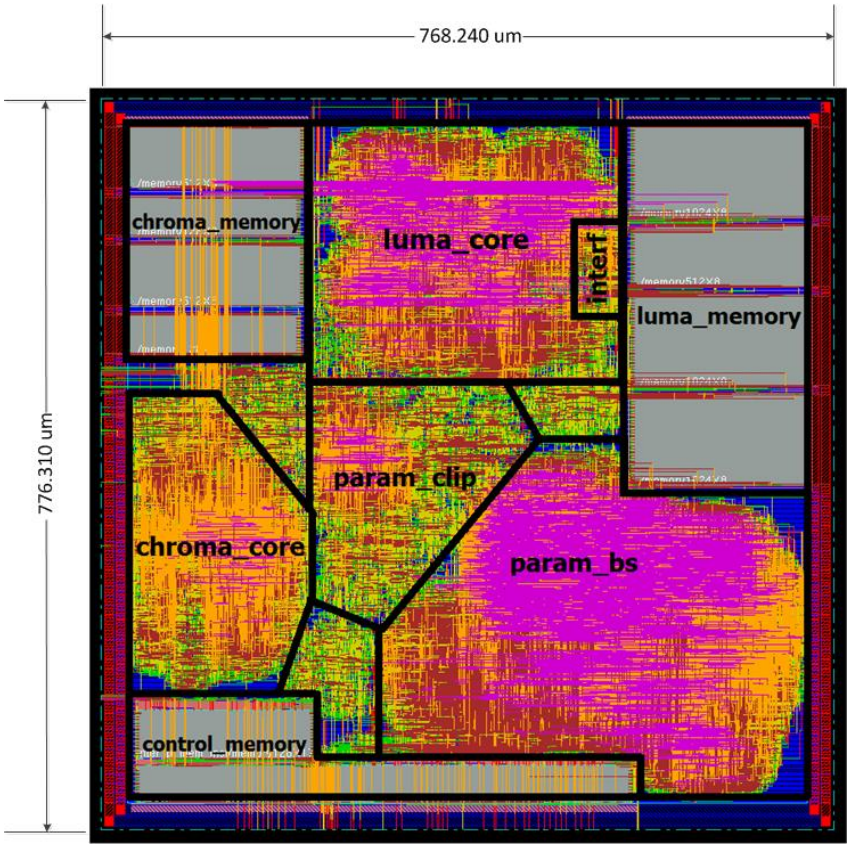


Figura 103. Implementación CMOS 65nm del DF

4.7. Conclusiones

En este capítulo se ha presentado la implementación del DF de un códec de vídeo H.264/AVC-SVC, siguiendo el flujo de diseño *hardware* propuesto, que parte de la solución OpenSVC adaptada para crear un modelo SystemC a nivel ESL. Desde este punto de partida, se ha seguido un flujo basado en síntesis de alto nivel para obtener una arquitectura *hardware*. Se ha diseñado una plataforma de validación basada en un SoC configurable para verificar el correcto funcionamiento del diseño. De igual forma se ha realizado una implementación ASIC para demostrar su viabilidad de implementación integrada y para evaluar sus prestaciones finales.

Capítulo 5. Conclusiones y líneas futuras

5.1. Introducción

En esta tesis doctoral se ha realizado un estudio de las principales ventajas que aporta la introducción de la síntesis de alto nivel en el flujo de diseño de sistemas electrónicos complejos, poniendo especial énfasis en la creación de aceleradores *hardware* para distintas aplicaciones. El escenario de la Tecnología de Diseño es rápidamente cambiante, pero al mismo tiempo hay avances clave que se resaltan en esta tesis. Como en todo proceso de creatividad humana y diseño, la componente de Arte y estilo personal de los diseñadores no solo es inevitable sino conveniente. Por esta razón los diseños experimentales y las métricas de su validación arrojan luz sobre las mejores prácticas que se recomiendan en el proceso de diseño. Esta tesis refuerza el criterio de introducir en Síntesis de Alto Nivel el punto intermedio del nivel TLM como plataforma necesaria en el flujo de síntesis, y sostiene que el nivel RTL no es cancelable, a no ser que se quieran afrontar riesgos e ineficacias en la implementación que alejarían cualquier metodología de los procedimientos de síntesis realmente útiles en la industria. En este capítulo de conclusiones condensamos nuestra visión de los puntos claves del escenario de la Tecnología

de Diseño, las conclusiones aportadas en la tesis en Síntesis de Alto Nivel para IPs en sistemas heterogéneos, y las líneas de investigación futuras que esta tesis deja iniciadas en el grupo de investigación.

5.2. Conclusiones

5.2.1. Escenario

El incremento de la capacidad de integración que se ha producido en la tecnología de semiconductores ha traído consigo un aumento en la complejidad de los sistemas electrónicos y por ende de su diseño. Este incremento de la capacidad de integración gobernado por la ley de Moore y por aspectos cubiertos por aproximaciones *More than Moore* ha permitido abordar nuevos problemas que requieren alta capacidad de computación, con costes energéticos que han continuado reduciéndose, hasta el punto de poder realizarse en Sistemas en un solo chip (SoC). Actualmente, el uso de sistemas heterogéneos representa una solución generalizada al problema del diseño de SoCs. Los sistemas heterogéneos se basan en una descomposición que incluye el recurso a asignar tareas críticas de la computación a su ejecución por aceleradores específicos en *hardware* en entornos dominados por soluciones basadas en varios microprocesadores o varios núcleos de proceso en un chip. Estos aceleradores de *hardware* se adaptan al problema con el fin de lograr resultados en tiempos de respuesta aceptables. En estos casos, la síntesis de alto nivel (SAN) puede jugar un papel central, está siendo una tendencia ampliamente aceptada en la industria, y es objeto de intensa investigación.

Ya en los años 90 se adoptaron técnicas de diseño RTL, en las que los lenguajes de descripción de *hardware* (VHDL/Verilog) y la síntesis lógica desde RTL han contribuido a reducir la complejidad asociada a los problemas complejos. En la actualidad hay consenso en la comunidad científica internacional en el campo de Tecnología del Diseño en que es necesario subir el nivel de abstracción en la especificación del sistema para poder realmente aprovechar eficientemente el número de transistores disponibles. Más aún, ese nuevo nivel de abstracción es necesario para cubrir la brecha entre la creciente complejidad de los algoritmos y la gran densidad de transistores disponible.

El problema planteado es el de obtener métodos y mejores prácticas que contribuyan a la eficiencia y productividad de la tecnología de diseño. Se trata de poder manejar ese muy elevado número de transistores de forma inteligente. La adopción de descripciones a nivel algorítmico y a nivel de sistema permite al diseñador centrar su atención en la funcionalidad y en la organización de la arquitectura del sistema y dejar a procedimientos automáticos guiados los detalles de la implementación. Sin embargo la excesiva automatización pretendida no siempre conduce a soluciones eficientes. La síntesis desde el nivel sistema o ESL es un campo en evolución y busca métodos para obtener implementaciones eficientes. ESL está demostrando su potencial en la exploración del espacio de diseño a nivel sistema buscando una correspondencia adecuada con el espacio de arquitecturas disponibles, pero está encontrando limitaciones en los intentos de convertir los flujos ESL en flujos de síntesis e implementación final. Un recurso disponible para ESL es la descomposición en algoritmos y la aplicación de SAN en el paradigma antes descrito de síntesis de aceleradores *hardware*.

Esta tesis se desarrolla en este escenario, y en este recurso metodológico de la descomposición del sistema en algoritmos como técnica de reducción de la magnitud del problema. Aquí la síntesis de alto nivel juega un papel central al permitir generar implementaciones RTL controladas y verdaderamente adaptadas a las condiciones de la aplicación. Si es importante en la productividad del diseño la reutilización de implementaciones *hardware* como bloques *Intellectual Property* (IPs), la SAN permite además introducir el concepto análogo de reutilización a nivel algorítmico. Para lograrlo ha sido un avance clave la introducción del modelado TLM al nivel de transacciones, ya que permite abordar por separado la implementación de la funcionalidad y de las interfaces. Esta estructuración del diseño está conduciendo a implementaciones más eficientes, dentro de un uso inteligente de los flujos de síntesis, y de un uso controlado y verificable de los resultados intermedios del proceso de síntesis. La utilización de lenguajes de alto nivel, tales como C/C++ y SystemC, permite reducir la distancia semántica entre la descripción algorítmica y su implementación *hardware*. Sin embargo esta transformación requiere tener un conocimiento de las principales características de la tecnología de implementación para obtener la calidad deseable en los resultados finales, medida en términos de prestaciones (latencia y tiempo de ejecución), potencia y área (PPA), y en términos de esfuerzo (costo y tiempo) de diseño.

La transición desde metodologías RTL a metodologías de alto nivel requiere un cambio de paradigma en los aspectos de modelado y transformaciones a realizar en el diseño. En primer lugar, el diseño se verifica sin información temporal, con una visión del problema desde el punto de vista funcional cuando se parte de descripciones basadas en C/C++, o con una visión aproximada en el caso de utilizar SystemC. La transformación del diseño precisa ahora no solo de tareas de planificación funcional, asignación de operaciones, asignación de registros, multiplexores, almacenamiento en memoria y otras tareas asociadas, sino aportar una mayor visibilidad a la arquitectura de comunicaciones o interconexión, a los protocolos de comunicación, y a la creación de tipos de datos extendidos. La encapsulación de los datos, su cuantificación y soporte para operaciones en coma fija o en coma flotante son tareas que afectan a los resultados finales medidos en término de prestaciones, potencia y área.

Durante la realización de este trabajo se han realizado además, como parte central de la experiencia, dos implementaciones de referencia, pertenecientes a dominios alejados entre sí, y de elevada complejidad y exigencia. Se ha aplicado el flujo de síntesis y las recomendaciones propuestos, y se han adoptado las estrategias de optimización requeridas en función de las limitaciones de latencia y *throughput* de cada aplicación. Manteniendo el mismo flujo de SAN propuesto se han recorrido diferentes estrategias en función de las necesidades concretas impuestas por la plataforma de implementación. Por ejemplo, el protocolo de integración del IP (LocalLink) requiere una secuencia definida en su protocolo que durante la síntesis no puede ser modificada. Situación similar ocurre con las interfaces de memoria. En todos estos casos típicos es necesario proteger estas zonas del modelo de posibles modificaciones realizadas por el planificador. La experiencia nos demuestra también que para el resto de los casos los mejores resultados se obtienen en cambio cuando se permite que el planificador realice una planificación guiada por las prestaciones temporales. La inclusión en línea de las funciones crea oportunidades para mejorar la planificación del bloque, pero puede producir incrementos

significativos de los recursos consumidos en el diseño. Lo mismo ocurre para el caso del tratamiento de los bucles, cuando se intenta optimizar la latencia final del sistema.

Para las dos implementaciones realizadas de cada caso de estudio, tanto FPGA como ASIC, el análisis temporal obtenido en la síntesis de alto nivel tiene dispersiones significativas con respecto al obtenido en RTL. Esta es una evidencia adicional sobre el papel fundamental del nivel RTL y la inconveniencia de crear flujos de diseño desde alto nivel que lo cancelen. El apoyo de la herramienta de síntesis lógica ha sido clave a la hora de tomar decisiones temporales, que se retroalimentan de forma interactiva en el entorno de SAN para realizar mejoras en la planificación y en la asignación de recursos. Otro tanto ocurre en SAN respecto del tratamiento de las memorias, con objeto de aprovechar la ventaja de las memorias de bloques disponibles en la FPGA y de los bloques de memoria generados para el caso del ASIC. En ambos casos el objetivo es disminuir costes en recursos (área para el ASIC). En estos casos es necesario compatibilizar la planificación realizada con los modelos funcionales de la memoria, que se convierten en una restricción para la planificación del correspondiente bloque. Por tanto es necesario conocer bien la tecnología de implementación del sistema, incluso desde un flujo SAN.

5.2.2. Aportaciones

1. Esta tesis aporta en primer lugar un estudio de las diferentes opciones existentes para la realización de la síntesis de alto nivel. Este estudio está acompañado de la experiencia de diseño del autor en el seno del equipo de investigación en EDA en el IUMA. Se da una visión general de diferentes etapas que ha recorrido la síntesis de alto nivel, hasta alcanzar la situación actual, dejando atrás la prescripción de utilizar HDLs como entrada y adoptando metodologías de síntesis a partir de C/C++/SystemC. La tesis aporta un conjunto de recomendaciones probadas para el flujo de diseño mediante SAN, organizadas en un conjunto de principios que hemos consolidado en numerosos diseños, un conjunto de ideas prácticas experimentales en el uso de las herramientas del flujo de diseño, y finalmente un flujo de síntesis que se ha contrastado cuantitativamente en esos diseños. Se indican los procesos de cosimulación a seguir para la verificación del diseño antes y después de la SAN.
2. Es importante y práctico encapsular la síntesis SAN en diseños IP, es decir completos y reutilizables, *standalone*. Es debido a que según el enfoque *platform based design (PBD)* –de éxito industrial– la integración de estos bloques IP en plataformas permite ahorrar tiempo y esfuerzo en la etapa de transformación del diseño desde la plataforma intermedia al target final. Por eso es necesario seguir avanzando en técnicas para flexibilizar el diseño, el IP, para que sea portable para diferentes targets. Se propone una visión de “Doble X” (figura 33). En la primera X se mapean tareas y funciones a estructuras TLM o de transacciones definidas por la arquitectura. En la segunda se mapean estas arquitecturas a la clase de plataforma intermedia elegida. La salida de esta segunda X es la implementación del algoritmo ya estructurado y sintetizado en los IPs componentes de la arquitectura de la plataforma, en el target final. Una amplia guía de recomendaciones prácticas y estrategias basadas en esta “Doble X” se detalla en el capítulo 2.
3. La tesis aporta en segundo lugar la implementación en FPGA y en ASIC de dos aplicaciones de aceleración *hardware* distantes entre sí, como casos de uso del flujo de síntesis

establecido y como cuantificación, contraste y criba de las recomendaciones realizadas y obtención de recomendaciones y estrategias adicionales.

a. En el primero de los casos se trata de una aplicación para el procesado de eventos complejos. Es una aplicación dominada por el flujo del control, es decir por la aplicación de numerosas reglas de decisión fuertemente entrelazadas y condicionadas, a partir de datos que llegan en ritmos exigentes para el procesado en tiempo real.

b. En el segundo caso se trata de un filtro de bucle adaptativo (*Deblocking Filter*) para la decodificación de vídeo siguiendo el estándar H.264/AVC-SVC. Este bloque es de nuevo un bloque complejo y cuello de botella en la implementación de códecs *hardware* del estándar. Es una aplicación dominada por el flujo de los datos e intensiva en procesamiento y en uso de memoria.

c. Para ambos casos se ha utilizado el flujo SAN aportado en la primera parte. En este flujo se parte de SystemC como lenguaje de especificación. La descripción en SystemC se obtiene transformando y descomponiendo la especificación algorítmica de entrada. Se aplica SAN a esta descomposición y se guía la síntesis asegurando la verificabilidad de cada etapa mediante la mencionada introducción del nivel TLM y el flujo en “Doble X”.

d. Para ambos problemas de referencia se ha realizado una implementación FPGA sobre una plataforma basada en un dispositivo Xilinx Virtex-5 FX, que incluye procesadores PowerPC440. Igualmente se ha realizado la implementación de ambos problemas en un IP ASIC. Para cada problema y para cada tecnología objetivo se ha seguido el flujo SAN establecido. Esta experimentación ha permitido refinar el flujo SAN utilizado y formular nuevas aportaciones sobre mejores prácticas de diseño y nuevas recomendaciones. Los resultados han sido comparados con el estado del arte y con resultados generados por otros flujos de diseño y otros enfoques.

4. Durante el desarrollo de este trabajo de investigación se han creado experiencias de uso que reflejan la necesidad de una integración vertical y sin discontinuidades de los flujos de diseño para poder obtener una convergencia de los aspectos temporales. Obtener esta convergencia temporal se identifica como uno de los principales retos de la nueva generación de herramientas de síntesis de alto nivel. Se aportan orientaciones para explotar los recursos disponibles en las herramientas para lograr convergencia temporal.

5. Se ha observado la necesidad de mantener el nivel RTL como punto clave de la verificación del sistema, especialmente para la síntesis a partir de C/C++ donde no se dispone de información temporal ni de estructura del modelo, ya que solo el nivel RTL dispone de la información precisa necesaria tanto a nivel de precisión de ciclos de bus como de precisión de bits y pines, para validar la implementación realizada durante la fase de síntesis de alto nivel.

6. La experiencia ha demostrado que la utilización modelos SystemC como entrada a la síntesis facilitan la verificación del diseño en niveles altos de abstracción, pasando a la síntesis únicamente aquellos modelos que han sido verificados. La utilización de aproximaciones

TLM, separando el refinamiento de las interfaces de su funcionalidad aporta ventajas significativas al proceso de diseño. Ese aspecto es de especial interés cuando el propio entorno de diseño de alto nivel soporta la síntesis de interfaces y protocolos.

7. Igualmente se han evaluado otros aspectos claves que afectan a la implementación del sistema, como puede ser la organización del modelo en cuanto a partición y jerarquía, la arquitectura de la memoria y la estructura de los datos. Igualmente se han tenido en cuenta aspectos relacionados con la paralelización de funciones y bucles, y el control de los protocolos de comunicación.
8. El flujo de SAN para el diseño se ha organizado teniendo en cuenta los requisitos de integridad de los datos y facilidad de uso, requeridos para cubrir la brecha entre la poca productividad de los flujos de diseño y la productividad de la gran densidad de transistores disponible. La utilización de scripts basados en Tcl ha permitido definir estrategias de uso para cada caso concreto lo que facilita la reproducibilidad de resultados, y su portabilidad a distintas tecnologías. Todos los resultados se aportan como contribuciones a la creación de flujos SAN eficientes para diseños complejos.
9. La validación tanto sobre FPGA como sobre ASIC ha puesto en valor la necesidad de tener referencias hacia el modelo inicial durante la implementación y prototipado de tal forma que se tenga un enlace único a la fuente de los problemas encontrados durante el prototipado.

5.3. Líneas futuras

A partir de los resultados obtenidos se plantean nuevas visiones del problema y se abren diferentes líneas de actuación posibles.

a) El punto clave que debemos abordar es profundizar en el estudio e implementar las estrategias necesarias para mejorar los aspectos de convergencia en los resultados de prestaciones, potencia y área. Este creemos que es un problema clave a la hora de tomar decisiones en alto nivel que no condicionen las siguientes etapas del flujo de diseño. Algunas líneas exploratorias se han seguido, con resultados todavía inciertos ya que es necesario aislar la dependencia con el tipo de problema estudiado. Para el caso de FPGAs, como por ejemplo Xilinx, su entorno de SAN genera información para guiar a la herramienta de síntesis lógica indicando cómo debe proseguir en el flujo de diseño (por ejemplo en la generación de BRAM). Sin embargo, hemos encontrado problemas no resueltos en la proyección de los resultados hasta la implementación física. Para el caso de los ASICs existen trabajos que muestran la influencia de las decisiones tomadas en SAN en la capacidad de realización de las interconexiones del diseño. Como hemos indicado anteriormente estos problemas deben ser estudiados para encontrar soluciones que se puedan generalizar al menos en determinados dominios de aplicaciones.

b) En segundo lugar, otro de los problemas que es necesario abordar es encontrar alternativas que faciliten la reutilización de los IPs a nivel de comportamiento entre entornos de SAN. Con la ayuda del preprocesador de C y algunos símbolos estándares se permite compatibilizar los modelos. El trabajo aquí consiste en definir una metodología, al estilo del RMM [162] que facilite la reutilización de los IPs descritos a nivel de comportamiento.

El flujo de diseño propuesto utiliza Cadence CtoS como herramienta de síntesis de alto nivel. Durante la fase de cierre de este trabajo Cadence ha anunciado un nuevo entorno de síntesis denominado Stratus. Se plantea la idoneidad de integrar la nueva herramienta en el flujo de síntesis establecido, redefiniendo el punto de partida desde modelos algo más abstractos y manteniendo los puntos intermedios al nivel TLM y al nivel RTL. Esta línea de trabajo supone establecer un enlace con metodologías ESL que facilitan la implementación del sistema completo. Se pretende en este caso establecer las estrategias de síntesis necesarias y evaluarlas usando los parámetros PPA.

Igualmente se pretende validar la estrategia de síntesis para aplicaciones de aceleración *hardware* utilizando SoCs o MPSoC programables, como por ejemplo Xilinx Zynq. Esta línea de trabajo tiene un doble objetivo de investigación y desarrollo. Por una parte se establece una metodología de trabajo y de experiencias de uso de interés en ingeniería de diseño electrónico y por otro permite tener una plataforma de desarrollo que conduce a un rápido prototipado de algoritmos de diverso tipo, facilita su validación en *hardware*. En una primera aproximación se pretende usar esta ampliación del flujo descrito para crear aceleradores para el procesamiento de paquetes en una red de datos.

El desarrollo de las técnicas de SAN facilita la utilización virtual de las FPGAs en un entorno que permite recibir distintos diseños en una misma FPGA física. Aquí es necesario establecer mecanismos de configuración parcial y la posibilidad de configurar la FPGA desde el procesador empotrado, por ejemplo en un dispositivo Zynq. Se compararán diferentes estrategias de síntesis *off-line* y los mecanismos de configuración necesarios. El objetivo final es evaluar sus ventajas en un entorno de aceleración *hardware*.

Por último, tal como se ha indicado en esta tesis, existen alternativas de SAN basadas en otros modelos de uso, como son el caso de Altera OpenCL o Xilinx SDAccel, donde el modelo de referencia es el usado para la programación de GPUs. Para el desarrollo de determinados tipos de aceleradores puede obtenerse ventajas si se logra una mutua compatibilidad entre OpenCL y nuevas versiones de TLM. Este es un camino análogo al que se está siguiendo en computación de altas prestaciones con la combinación y compatibilidad del modelo de comunicación *Shared Memory* con el de *Message Passing*, o en otras palabras OpenMP[163] y MPI (MPI-3)[164]. La concurrencia *hardware* es por naturaleza del tipo MPI. La inclusión en MPI-3 de recursos OpenMP es una buena meta que insinúa el camino entre OpenCL y SystemC.

Referencias

[1] Stanford University, «Solving Big Questions Require Big Computation», 2014. [En línea]. Disponible en: http://news.stanford.edu/features/2014/computing/. [Accedido: 15-oct-2015].

[2] Gartner, «Gartner Says 4.9 Billion Connected “Things” Will Be in Use in 2015», 2014. [En línea]. Disponible en: http://www.gartner.com/newsroom/id/2905717. [Accedido: 20-may-2015].

[3] HIPEAC, «HIPEAC: European Network of Excellence on High Performance and Embedded Architecture and Compilation», 2015. [En línea]. Disponible en: https://www.hipeac.net/. [Accedido: 15-oct-2015].

[4] K. Wakabayashi, «C-based behavioral synthesis and verification analysis on industrial design examples», en *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, 2004, vol. Yokohama, pp. 344-348.

[5] The International Technology Roadmap for Semiconductors, «International Technology Roadmap for Semiconductors, 2011 Edition, System Drivers», *International Technology Roadmap for Semiconductors*, 2011. [En línea]. Disponible en: http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf. [Accedido: 09-jul-2015].

[6] A. B. Kahng y G. Smith, «A new design cost model for the 2001 ITRS», en *Quality Electronic Design, 2002. Proceedings. International Symposium on*, 2002, pp. 190-193.

[7] The International Technology Roadmap for Semiconductors, «International Technology Roadmap for Semiconductors 2009 (ITRS)», *ITRS Report Release 2009*, 2009. [En línea]. Disponible en: http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Design.pdf. [Accedido: 09-jul-2015].

[8] S. Pasricha y N. Dutt, *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.

[9] J. D. Pérez Pérez, P. Pérez Carballo, y A. Sánchez Peña, «Arquitecturas de comunicación para Sistemas On Chip reconfigurables. Aplicación a la arquitectura ARTEMI», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria, 2007.

[10] T. De Schutter, *Better Software. Faster!: Best Practices in Virtual Prototyping*. Synopsys Press, 2014.

[11] F. Balarin, *Hardware-software co-design of embedded systems: the POLIS approach*. Springer Science & Business Media, 1997.

[12] A. Ferrari y A. Sangiovanni-Vincentelli, «System design: Traditional concepts and new paradigms», en *iccd*, 1999, p. 2.

[13] G. Martin, H. Chang, L. Cooke, M. Hunt, A. McNelly, y L. Todd, «Surviving the SOC Revolution: A Guide to Platform Based Design.» Kluwer Academic Publishers, 1999.

[14] K. Keutzer, J. M. Rabaey, y A. Sangiovanni-Vincentelli, «System-level design: orthogonalization of concerns and platform-based design», *Comput. Des. Integr. Circuits Syst. IEEE Trans.*, vol. 19, n.º 12, pp. 1523-1543, 2000.

[15] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, y A. Sangiovanni-Vincentelli, «Metropolis: An integrated electronic system design environment», *Computer (Long. Beach. Calif).*, vol. 36, n.º 4, pp. 45-52, 2003.

[16] A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone, y M. Sgroi, «System level design paradigms: Platform-based design and communication synthesis», en *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2004, vol. 11, n.º 3, pp. 537-563.

[17] Dept. of Electrical Engineering and Computer Sciences UC Berkeley, «Analysis, Synthesis, Verification of Electronic Systems. Introduction», 2015. [En línea]. Disponible en: http://embedded.eecs.berkeley.edu/Respep/Research/asves/. [Accedido: 03-nov-2015].

[18] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, y J. Gray, «A reconfigurable fabric for accelerating large-scale datacenter services», en *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, 2014, pp. 13-24.

[19] T. P. Morgan, «How Microsoft Is Using FPGAs To Speed Up Bing Search», *Enterprise Tech*, 2014. [En línea]. Disponible en: http://www.enterprisetech.com/2014/09/03/microsoft-using-fpgas-speed-bing-search/. [Accedido: 01-oct-2015].

[20] A. B. Kahng, M. Luo, G.-J. Nam, S. Nath, D. Z. Pan, y G. Robins, «Toward Metrics of Design Automation Research Impact», en *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 263-270.

[21] F. Ghenassia, Ed., *Transaction Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[22] S. Dart, «Behavioral IP reuse methodology», 2005. [En línea]. Disponible en: http://www.techdesignforums.com/practice/technique/behavioral-ip-reuse-methodology/. [Accedido: 02-jul-2014].

[23] V. Reyes, T. Bautista, G. Marrero, P. P. Carballo, y W. Kruijtzer, «CASSE: A system-level modeling and design-space exploration tool for multiprocessor systems-on-chip», en *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, 2004, pp. 476-483.

[24] Z. J. Jia, T. Bautista, A. Núnez, A. D. Pimentel, y M. Thompson, «A system-level infrastructure for multidimensional MP-SoC design space co-exploration», *ACM Trans. Embed. Comput. Syst.*, vol. 13, n.º 1s, p. 27, 2013.

[25] H.-Y. Liu, M. Petracca, y L. P. Carloni, «Compositional system-level design exploration with planning of high-level synthesis», en *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, pp. 641-646.

[26] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, y Z. Zhang, «High-Level Synthesis for FPGAs: From Prototyping to Deployment», *IEEE Trans. Comput. Des. Integr. Circuits*

Syst., vol. 30, pp. 473-491, 2011.

[27] D. D. Gajski, N. D. Dutt, A. C. H. Wu, y S. Y. L. Lin, *High—Level Synthesis: Introduction to Chip and System Design*. Springer Science & Business Media, 2012.

[28] V. Reyes Suárez y T. Bautista Delgado, «Métodos y herramientas para el diseño a nivel de sistema de plataformas multiprocesador heterogéneas para multimedia», Biblioteca General ULPGC, Colecciones, Tesis, Las Palmas de Gran Canaria, 2008.

[29] Z. J. Jia Li, T. Bautista Delgado, y A. Núñez Ordóñez, «System level design space exploration for MPSoC : methods, algorithms and new infrastructure», Biblioteca General ULPGC, Colecciones, Tesis, Las Palmas de Gran Canaria, 2010.

[30] Z. J. Jia, A. Núñez, T. Bautista, y A. D. Pimentel, «A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC», *Microprocess. Microsyst.*, vol. 38, n.º 1, pp. 9-21, feb. 2014.

[31] Z. J. Jia, T. Bautista, A. Nunez, C. Guerra, y M. Hernandez, «Design Space Exploration and Performance Analysis for the Modular Design of CVS in a Heterogeneous MPSoC», en *Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on*, 2008, pp. 193-198.

[32] Z. J. Jia, A. D. Pimentel, M. Thompson, T. Bautista, A. Nunez, y A. Núñez, «NASA: A generic infrastructure for system-level MP-SoC design space exploration», en *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*, 2010, pp. 41-50.

[33] T. Bautista, G. Marrero, P. P. Carballo, y A. Nunez, *Rapid-prototyping of high-performance RISC cores with VHDL*. 1997.

[34] P. Coussy, D. D. Gajski, M. Meredith, y A. Takach, «An introduction to high-level synthesis», *IEEE Des. Test Comput.*, vol. 26, n.º 4, pp. 8-17, 2009.

[35] D. D. Gajski y R. H. Kuhn, «Guest Editors' Introduction: New VLSI Tools», *Computer (Long. Beach. Calif).*, vol. 16, n.º 12, pp. 11-14, dic. 1983.

[36] L. Scheffer, L. Lavagno, y G. Martin, *EDA for IC system design, verification, and testing*. CRC Press, 2006.

[37] G. Martin y G. Smith, «High-level synthesis: Past, present, and future», *IEEE Des. Test Comput.*, n.º 4, pp. 18-25, 2009.

[38] A. M. P. Coussy, *High-Level Synthesis from Algorithm to Digital Circuit*. Springer, 2008.

[39] L. W. Nagel y D. O. Pederson, *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California, 1973.

[40] D. K. Lynn, «Computer Aided Design for Large-Scale Integrated Circuits», *Computer*, vol. 5, n.º 3. pp. 36-45, 1972.

[41] R. K. Brayton, *Logic minimization algorithms for VLSI synthesis*, vol. 2. Springer Science & Business Media, 1984.

[42] M. R. Barbacci, «Instruction set processor specifications (ISPS): The notation and its applications», *Comput. IEEE Trans.*, vol. 100, n.º 1, pp. 24-40, 1981.

[43] P. Pérez Carballo y A. Núñez Ordóñez, «Diseño, simulación y experimentación de microprocesadores en MICRO. Aplicación a un microprocesador a stack y a un microprocesador a registros», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria, 1987.

[44] M. R. Barbacci, «A comparison of register transfer languages for describing computers and digital systems», *IEEE Trans. Comput.*, n.º 2, pp. 137-150, 1975.

[45] C. G. Bell y A. Newell, *Computer structures: Readings and examples*, vol. 2. McGraw-Hill New York, 1971.

[46] C. R. Clare, *Designing logic systems using state machines*. McGraw-Hill, 1973.

[47] R. W. Hartenstein, K. Lemmert, y A. Wodtko, «KARL-III Language Reference, second edition», University of Kaiserslautern, Kaiserslautern, 1986.

[48] H. De Man, J. Rabaey, J. Vanhoof, G. Goossens, P. Six, y L. Claesen, «CATHEDRAL-II-a computer-aided synthesis system for digital signal processing VLSI systems», *Comput. Eng. J.*, vol. 5, n.º 2, pp. 55-66, 1988.

[49] J. Kunkel, H. Meyr, y G. Ascheid, «COSSAP: communication system simulation and analysis package'», en *2nd IEEE Workshop CAMAD of Commun. Links and Networks*, 1988.

[50] J. Mermet, *Fundamentals and Standards in Hardware Description Languages*, vol. 249. Springer Science & Business Media, 2012.

[51] A. C. Verschueren, «An object oriented design and simulation system for VLSI», *Microprocess. Microprogramming*, vol. 30, n.º 1, pp. 241-246, 1990.

[52] C.-M. Chu, M. Potkonjak, M. Thaler, y J. Rabaey, «HYPER: An interactive synthesis environment for high performance real time applications», *Proc. Int. Conf. Comput. Des.*, pp. 432-435, 1989.

[53] R. Rudell y A. Sangiovanni-Vincentelli, «Logic synthesis for VLSI design.» University of California, Berkeley, 1989.

[54] G. De Micheli y D. C. Ku, «HERCULES—a system for high-level synthesis», en *Proceedings of the 25th ACM/IEEE Design Automation Conference*, 1988, pp. 483-488.

[55] G. De Micheli, D. Ku, F. Mailhot, y H. Truong, «The Olympus synthesis system», *Des. Test Comput. IEEE*, vol. 7, n.º 5, pp. 37-53, 1990.

[56] D. Genin, P. Hilfinger, J. Rabaey, C. Scheers, y H. De Man, «DSP specification using the Silage language», *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*. pp. 1056-1060 vol.2, 1990.

[57] R. Camposano y W. Rosenstiel, «Synthesizing circuits from behavioural descriptions», *Comput. Des. Integr. Circuits Syst. IEEE Trans.*, vol. 8, n.º 2, pp. 171-180, 1989.

[58] P. G. Paulin y J. P. Knight, «Scheduling and binding algorithms for high-level synthesis», en *Design Automation, 1989. 26th Conference on*, 1989, pp. 1-6.

[59] M. D. Ercegovac y T. Lang, *Digital systems and hardware/firmware algorithms*. John Wiley & Sons, Inc., 1985.

[60] M. Davio, J.-P. Deschamps, y A. Thayse, *Digital systems, with algorithm implementation*. New York, NY, USA: John Wiley & Sons, Inc., 1983.

[61] A. Nuñez, Pedro P. Carballo, y R. Sarmiento, «Some Experiences in Designing an LSI GaAs Processor Using KARL-3», en *Proceedings ABAKUS workshop*, 1988, pp. 1-11.

[62] A. Nunez, R. Sarmiento, y P. P. Carballo, «Some Results in GaAs Processor Design using LSI Integrated-Circuits», *Microprocess. Microprogramming*, vol. 25, n.º 1-5, pp. 127-132, 1989.

[63] P. P. Carballo, R. Sarmiento, y A. Nunez, «Integer and Control Units for a GaAs 32-Bit Risc Processor», *Microprocess. Microprogramming*, vol. 37, n.º 1-5, pp. 105-108, 1993.

[64] A. M. López Alonso, M. Marrero, P. P. Carballo, M. Marrero Martín, y P. Pérez Carballo, «Entorno para la síntesis y generación automática de DSPs», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria, 1998.

[65] D. W. Knapp, *Behavioral Synthesis: Digital System Design using the Synopsys Behavioral Compiler*. Prentice-Hall, Inc., 1996.

[66] J. P. Elliott, *Understanding Behavioral Synthesis: a practical guide to High-Level Design*. Springer Science & Business Media, 2012.

[67] P. Schaumont, S. Vernalde, L. Rijnders, M. Engels, y I. Bolsens, «Synthesis of multi-rate and variable rate circuits for high speed telecommunications applications», en *European Design and Test Conference, 1997. ED&TC 97. Proceedings*, 1997, pp. 542-546.

[68] J. Moraga García, M. Marrero Martín, y P. Pérez Carballo, «Modelado VHDL y síntesis del algoritmo de cifrado IDEA», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria : (sp), 2001.

[69] P. Schaumont, R. Cmar, S. Vernalde, M. Engels, y I. Bolsens, «Hardware reuse at the behavioral level», en *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 784-789.

[70] S. Vernalde, P. Schaumont, y I. Bolsens, «An object oriented programming approach for hardware design», en *VLSI'99 Proceedings*, 1999, p. 68.

[71] C. E. Ruiz Cabrera, P. Pérez Carballo, G. I. Marrero Callicó, C. E. R. Cabrera, P. P. Carballo, y G. I. M. Callicó, «Diseño de un núcleo codificador/decodificador aritmético para JPEG2000 con SystemC», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria, 2003.

[72] D. D. Gajski, S. Abdi, A. Gerstlauer, y G. Schirner, *Embedded system design: modeling, synthesis and verification*. Springer Science & Business Media, 2009.

[73] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, y S. Zhao, *SPECC: Specification Language and Methodology*. Springer Science & Business Media, 2012.

[74] Impulse, «Tools - Impulse Accelerated Technologies», 2015. [En línea]. Disponible en: http://www.impulseaccelerated.com/. [Accedido: 15-oct-2014].

[75] M. Fingeroff, *High-level synthesis Blue Book*. Xlibris Corporation, 2010.

[76] M. Meredith, «High-level SystemC synthesis with forte's cynthesizer», en *High-Level Synthesis*, Springer, 2008, pp. 75-97.

[77] D. McGrath, «Celoxica's Agility compiler supporting SystemC 2.1 | EE Times», *EETimes*, 2006. [En línea]. Disponible en: http://www.eetimes.com/document.asp?doc_id=1158763. [Accedido: 27-oct-2015].

[78] H. D. Patel, S. K. Shukla, E. Mednick, y R. S. Nikhil, «A rule-based model of computation for SystemC: integrating SystemC and Bluespec for co-design», en *Proceedings of the Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE'06. Proceedings.*, 2006, pp. 39-48.

[79] Synopsys, «Synphony C Compiler», 2011. [En línea]. Disponible en: https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/SynphonyC-Compiler.aspx. [Accedido: 15-oct-2014].

[80] J. Teubner y L. Woods, «Data Processing on FPGAs», *Synth. Lect. Data Manag.*, vol. 5, n.º 2, pp. 1-118, jun. 2013.

[81] Cadence, «Cadence C-to-Silicon Compiler User Guide.» p. 1080, 2013.

[82] X. Shen, «Using term rewriting systems to design and verify processors», *Micro, IEEE*, vol. 19, n.º 3, pp. 36-46, 1999.

[83] T. Grötker, S. Liao, G. Martin, y S. Swan, *System Design with SystemC*. Boston: Boston: Kluwer Academic Publishers, 2002.

[84] L. Cai y D. Gajski, «Transaction level modeling: an overview», en *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2003, pp. 19-24.

[85] ARTEMI+, «ARTEMI+: Architectures for multi-format, multi-frontend, multimedia portable terminals», *ARTEMI+*, mar-2008. [En línea]. Disponible en: http://www.iuma.ulpgc.es/artemi/index.php?option=content&task=view&id=55&Itemid=58. [Accedido: 09-sep-2014].

[86] M. Thadani, P. P. Carballo, P. Hernandez, G. Marrero, y A. Nunez, «ESL flow for a hardware H.264/AVC decoder using TLM-2.0 and high-level synthesis: quantitative study», *Proc. SPIE*, vol. 7363, p. 73630K-73630K, 2009.

[87] A. Domínguez Hernández, P. Pérez Carballo, y A. Núñez Ordóñez, «Metodología de codiseño HW/SW para un decodificador H.264/AVC», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria : (sp), 2008.

[88] R. Herrera Navarro, P. Pérez Carballo, P. Hernández Fernández, y A. Núñez Ordóñez, «Modelado SystemC, simulación y síntesis de un subsistema para mejora de vídeo», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria : (sp), 2010.

[89] Mark Milligan, «750 engineer survey on HLS verification issues & power reduction»,

2014. [En línea]. Disponible en: http://www.deepchip.com/items/0544-03.html. [Accedido: 01-nov-2014].

[90] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, y D. Stroobandt, «An overview of today's high-level synthesis tools», *Des. Autom. Embed. Syst.*, vol. 16, n.º 3, pp. 31-51, 2012.

[91] Calypto Design Systems, «Catapult ® Synthesis Process , Concept , and Reference Manual.» p. 771, 2014.

[92] I. C. Society, «IEEE Standard for Standard SystemC® Language Reference Manual.» 2012.

[93] Kronos Group, «OpenCL - The open standard for parallel programming of heterogeneous systems», 2015. [En línea]. Disponible en: https://www.khronos.org/opencl/. [Accedido: 20-may-2005].

[94] Altera Inc., «Altera SDK for OpenCL. Programming Guide.» 2015.

[95] B. Jenkins, «OpenCL Overview webcast», 2015. [En línea]. Disponible en: https://www.altera.com/webcasts/opencl-overview/presentation.html?utm_source=Altera&utm_medium=webinar&utm_campaign=OpenCL_15_0&utm_content=NA_OpenCL_Overview_Tutorial. [Accedido: 03-sep-2015].

[96] J. Cong, Y. Fan, G. Han, W. Jiang, y Z. Zhang, «Platform-based behavior-level and system-level synthesis», en *SOC Conference, 2006 IEEE International*, 2006, pp. 199-202.

[97] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, y Z. Zhang, «xpilot: A platform-based behavioral synthesis system», *SRC TechCon*, vol. 5, 2005.

[98] «The LLVM Compiler Infrastructure», 2015. [En línea]. Disponible en: llvm.org. [Accedido: 01-sep-2015].

[99] B. Vega, P. P. Carballo, P. Hernández-Fernández, A. Domínguez, y A. Núñez, «TCP/IP Packet Analyzer on a Zynq Platform», en *DSD 2015 - Euromicro Digital Systems Design 2015*, 2015, p. 2.

[100] A. Domínguez, P. Pérez Carballo, y A. Nuñez, «Síntesis de alto nivel basada en Xilinx Vivado para aceleradores hardware», Las Palmas de Gran Canaria, 2014.

[101] Xilinx Inc., «LocalLink Interface Specification», vol. 006, 2005.

[102] S. Swan, Q. Zhu, y X. Li, «Moving to SystemC TLM for design and verification of digital

hardware», *EETimes*, p. 4, may-2013.

[103] P. P. Carballo, O. Espino, R. Neris, P. Hernandez-Fernandez, T. M. Szydzik, y A. Nunez, «Scalable Video Coding Deblocking Filter FPGA and ASIC Implementation Using High-Level Synthesis Methodology», *Digital System Design (DSD), 2013 Euromicro Conference on*. pp. 415-422, 2013.

[104] P. P. Carballo, O. Espino, R. Neris, P. Hernández-Fernández, T. M. Szydzik, y A. Núñez, «Implementation of scalable video coding deblocking filter from high-level SystemC description», en *Proc. SPIE 8764, VLSI Circuits and Systems VI*, 2013, vol. 8764, pp. 876408-876410.

[105] P. Monzón Rodríguez, P. P. Carballo, y P. Hernández-Fernández, «Metodología de de síntesis ESL basada en TLM. Aplicación al diseño de un decodificador de vídeo», Biblioteca General ULPGC, Colecciones, PFC, 2015.

[106] J. Villarreal, A. Park, W. Najjar, y R. Halstead, «Designing modular hardware accelerators in C with ROCCC 2.0», en *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, 2010, pp. 127-134.

[107] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, y T. Czajkowski, «LegUp: high-level synthesis for FPGA-based processor/accelerator systems», en *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011, pp. 33-36.

[108] D. Gajski, T. Austin, y S. Svoboda, «What input-language is the best choice for high level synthesis (HLS)?», en *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, 2010, pp. 857-858.

[109] D. Payne, «Choosing C++ or SystemC for High Level Synthesis», *Semiwiki.com*, 2015. [En línea]. Disponible en: https://www.semiwiki.com/forum/content/4823-choosing-c-systemc-high-level-synthesis.html. [Accedido: 20-oct-2015].

[110] J. Sanguinetti, «High-level synthesis, verification and language | EE Times», *EETimes*, 2010. [En línea]. Disponible en: http://www.eetimes.com/document.asp?doc_id=1276220. [Accedido: 04-abr-2014].

[111] A. Sánchez-Peña, P. P. Carballo, L. García, y A. Núñez, «VIPACES, Verification Interface Primitives for the Development of AXI Compliant Elements and Systems», en *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. 305-312.

[112] A. Sánchez-Peña, P. P. Carballo, y A. Núñez, «Exploring system interconnection architectures with VIPACES: from direct connections to NOCs», en *Microtechnologies for*

the New Millennium, 2007, p. 65900Q-65900Q.

[113] E. Domínguez Quintana y P. Pérez Carballo, «Librería de componentes multinivel TLM-PA para buses de comunicación en chip. Aplicación a AMBA AXI 3», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria, 2012.

[114] J. Cong, «High-Level Synthesis Revisited. Progress and Applications», abr-18d. C. [En línea]. Disponible en: http://cadlab.cs.ucla.edu/~cong/slides/HLS_July_2012.pdf. [Accedido: 16-oct-2015].

[115] Y.-T. Chen, J. Cong, J. Lei, y P. Wei, «A Novel High-Throughput Acceleration Engine for Read Alignment.»

[116] G. Inggs, D. Thomas, y W. Luk, «A heterogeneous computing framework for computational finance», en *Parallel Processing (ICPP), 2013 42nd International Conference on*, 2013, pp. 688-697.

[117] P. V. dos Santos, J. C. Alves, y J. C. Ferreira, «An FPGA Framework for Genetic Algorithms: Solving the Minimum Energy Broadcast Problem», en *Digital System Design (DSD), 2015 Euromicro Conference on*, 2015, pp. 9-16.

[118] O. Arcas-Abella, G. Ndu, N. Sonmez, M. Ghasempour, A. Armejach, J. Navaridas, J. Mawer, A. Cristal, y M. Lujan, «An empirical evaluation of High-Level Synthesis languages and tools for database acceleration», en *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1-8.

[119] O. Knodel y R. G. Spallek, «Computing Framework for Dynamic Integration of Reconfigurable Resources in a Cloud», en *Digital System Design (DSD), 2015 Euromicro Conference on*, 2015, pp. 337-344.

[120] Cadence, «Stratus High-Level Synthesis», 2015. [En línea]. Disponible en: http://www.cadence.com/rl/Resources/datasheets/Stratus_ds.pdf. [Accedido: 15-oct-2015].

[121] M. R. N. Mendes, P. Bizarro, y P. Marques, «A Performance Study of Event Processing Systems», en *Performance Evaluation and Benchmarking SE - 16*, vol. 5895, R. Nambiar y M. Poess, Eds. Springer Berlin Heidelberg, 2009, pp. 221-236.

[122] D. Schneider, «The microsecond market», *Spectrum, IEEE*, vol. 49, n.º 6. pp. 66-81, 2012.

[123] S.-T. Levi, A. K. Agrawala, A. K. A. S. Levi, S.-T. Levi, y A. K. Agrawala, *Real-time System Design*. New York, NY, USA: McGraw-Hill, Inc., 1990.

[124] A. Aly, E. Zeidan, A. Hamed, y F. Salem, «An Antilock-Braking Systems (ABS) Control: A Technical Review», *Intell. Control Autom.*, vol. 2, n.º 3, pp. 186-195, 2011.

[125] G. F. Mauer, «A fuzzy logic controller for an ABS braking system», *Fuzzy Syst. IEEE Trans.*, vol. 3, n.º 4, pp. 381-388, 1995.

[126] J. L. Díaz, D. Barreto, L. García, G. Marrero, P. P. Carballo, y A. Núñez, «Accelerating a MPEG-4 video decoder through custom software/hardware co-design», en *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE)*, 2007, vol. 6590, p. 90013.

[127] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, y H.-A. Jacobsen, «Efficient Event Processing Through Reconfigurable Hardware for Algorithmic Trading», *Proc. VLDB Endow.*, vol. 3, n.º 1-2, pp. 1525-1528, 2010.

[128] P. S. G. M. Gokhale, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Birkhäuser, 2005.

[129] «Algo or Algorithmic Trading Definition - NASDAQ.com», *NASDAQ Web Site*. [En línea]. Disponible en: http://www.nasdaq.com/investing/glossary/a/algo-trading. [Accedido: 30-jul-2014].

[130] «Algorithm Trading Definition | Algorithm Trading Meaning - The Economic Times», *EETimes*. [En línea]. Disponible en: http://economictimes.indiatimes.com/definition/algorithm-trading. [Accedido: 30-jul-2014].

[131] G. Nuti, M. Mirghaemi, P. Treleaven, y C. Yingsaeree, «Algorithmic Trading», *Computer*, vol. 44, n.º 11. pp. 61-69, 2011.

[132] K. A. B. Aly, «Introducing FPGA-Based Acceleration for High-Frequency Trading | EE Times.» [En línea]. Disponible en: http://www.eetimes.com/author.asp?section_id=36&doc_id=1323278. [Accedido: 01-ago-2014].

[133] Edosoft Factory, «ROBOBROKER: Ponga a disposición de sus clientes de banca electrónica el Bróker Online más potente del mercado.» [En línea]. Disponible en: http://www.edosoftfactory.com/images/dossier_rb.pdf. [Accedido: 31-jul-2014].

[134] M. Radetzki y R. Khaligh, «On Construction of Cycle Approximate Bus TLMs», en *Embedded Systems Specification and Design Languages SE - 3*, vol. 10, E. Villar, Ed. Springer Netherlands, 2008, pp. 31-43.

[135] G. Delp, J. Biggs, y S. Jadcherla, «Design & Verification of Low Power SoCs», 2009.

[136] M. Keating, D. Flynn, R. Aitken, A. Gibbons, y K. Shi, *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[137] Forte Design Systems, «Cynthesizer Reference Guide.» p. 1455, 2014.

[138] P. Li, L.-N. Pouchet, y J. Cong, «Throughput Optimization for High-Level Synthesis Using Resource Constraints», en *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*, 2014.

[139] Xilinx Inc., «Vivado Design Suite User Guide. High-Level Synthesis.» Xilinx, Inc, p. 694, 2015.

[140] Xilinx Inc., «Vivado Design Suite», 2015. [En línea]. Disponible en: http://www.xilinx.com/products/design-tools/vivado/index.htm. [Accedido: 08-jul-2015].

[141] A. Sutton, «The Great Divide: Why Next-Generation FPGA Designs will be Hierarchical and Team-Based», Mountain View, CA, 2011.

[142] A. Sutton, «FPGA Design Methods for Fast Turn Around», Mountain View, CA, 2012.

[143] Xilinx Inc., «Block in Virtex-5 FPGAs», 2010.

[144] Tim Carsten, «Programming with pcap», 2015. [En línea]. Disponible en: http://www.tcpdump.org/pcap.html. [Accedido: 13-oct-2015].

[145] O. Espino Santana, P. Pérez Carballo, P. Hernández Fernández, R. Carrascosa González, y O. E. Santana, «Acelerador Hardware para procesado de eventos en tiempo real», Biblioteca General ULPGC, Colecciones, PFC, Las Palmas de Gran Canaria, 2012.

[146] Faraday Tech, «Faraday / UMC 65 nm Memory Synchronous High-Density Dual-Port SRAM Compiler with One-Pair Row Redundancy and BIST Test Interface Option», 2009.

[147] K. Zhang, *Embedded Memories for Nano-Scale VLSIs*. Boston, MA: Springer US, 2009.

[148] M. F. Nadeem, S. Wong, G. Kuzmanov, A. Shabbir, M. F. Nadeem, y F. Anjam, «Low-power, high-throughput deblocking filter for H.264/AVC», en *System on Chip (SoC), 2010 International Symposium on*, 2010, pp. 93-98.

[149] N. Ta, J. Youn, H. Kim, J. Choi, y S.-S. Han, «Low-power High-throughput Deblocking Filter Architecture for H.264/AVC», en *Electronic Computer Technology, 2009 International*

Conference on, 2009, pp. 627-631.

[150] M. Parlak y I. Hamzaoglu, «A Low Power Implementation of H.264 Adaptive Deblocking Filter Algorithm», en *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, 2007, pp. 127-133.

[151] G. J. Conklin y S. S. Hemami, «A comparison of temporal scalability techniques», *Circuits Syst. Video Technol. IEEE Trans.*, vol. 9, n.º 6, pp. 909-919, 1999.

[152] M. Narroschke, «Functionalities and costs of scalable video coding for streaming services», en *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*, 2002, vol. 2, pp. 1330-1335 o.2.

[153] M. Blestel y M. Raulet, «Open SVC decoder: a flexible SVC library», en *Proceedings of the international conference on Multimedia*, 2010, vol. Firenze, I, pp. 1463-1466.

[154] F. Pescador, D. Samper, M. J. Garrido, E. Juarez, y M. Blestel, «A DSP based SVC IP STB using open SVC cecoder», en *14th IEEE International Symposium on Consumer Electronics, ISCE 2010*, 2010, vol. Braunschwe.

[155] P. List, A. Joch, J. Lainema, G. Bjontegaard, y M. Karczewicz, «Adaptive deblocking filter», *Circuits Syst. Video Technol. IEEE Trans.*, vol. 13, n.º 7, pp. 614-619, 2003.

[156] UPM/ULPGC, «PccMuTe: PowerConsumption Control in multimedia Terminals.» [En línea]. Disponible en: https://www.citsem.upm.es/index.php/proyectos-es?view=project&task=show&id=39. [Accedido: 09-sep-2014].

[157] A. Núñez, «Informe final del proyecto Control del Consumo en Terminales Multimedia TEC2009-14672-C02-02», 2014.

[158] Texas Instruments Inc., «OMAP35x Applications Processor: Technical Reference Manual», 2012.

[159] A. Otero, E. De La Torre, T. Riesgo, T. Cervero, S. López, G. Callicó, y R. Sarmiento, «Run-time Scalable Architecture for Deblocking Filtering in H. 264/AVC-SVC Video Codecs», en *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, 2011, pp. 369-375.

[160] Y.-L. S. Lin, K. Chao-Yang, K. Hung-Chih, y C. Jian-Wen, *VLSI Design for video coding: H.264/AVC Encoding From Standard Specification to Chip*. Springer, 2010.

[161] D. Kamel, F.-X. Standaert, y D. Flandre, «Scaling trends of the AES S-box low power consumption in 130 and 65 nm CMOS technology nodes», en *Circuits and Systems, 2009.*

ISCAS 2009. IEEE International Symposium on, 2009, pp. 1385-1388.

[162] M. Keating y P. Bricaud, *Reuse methodology manual for system-on-a-chip designs*. Springer, 2002.

[163] OpenMP.org, «OpenMP. The OpenMP® API specification for parallel programming», 2015. [En línea]. Disponible en: http://openmp.org/wp/. [Accedido: 05-nov-2015].

[164] MPI Forum, «MPI Documents.» [En línea]. Disponible en: http://www.mpi-forum.org/docs/docs.html. [Accedido: 05-nov-2015].

ULPGC