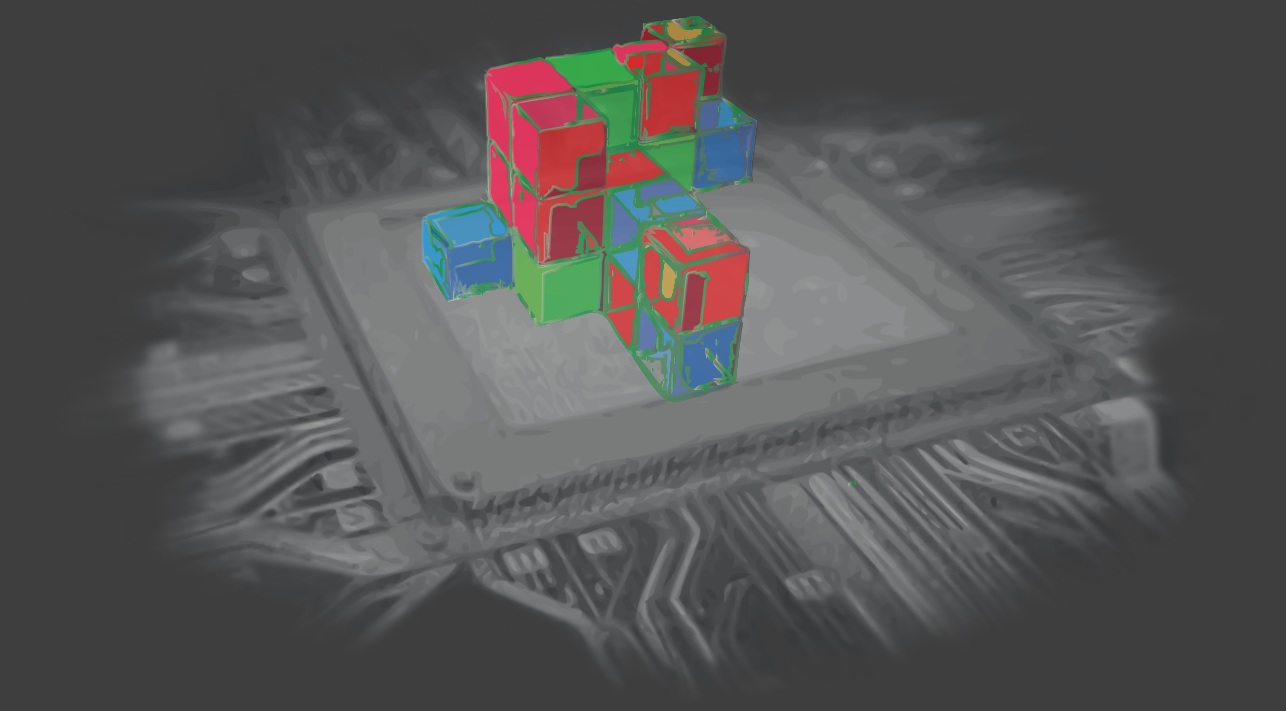
Tesis Doctoral

Teresa Cervero Garcia

Las Palmas de Gran Canaria, Julio 2013



Dynamically reconfigurable architectures for video coding and hyperspectral imaging systems

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

D. PEDRO PÉREZ CARBALLO SECRETARIO DEL INSTITUTO UNIVERSITARIO DE MICROELECTRÓNICA APLICADA DE LA UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA,

CERTIFICA,

Que el Consejo de Doctores del Departamento en su sesión de fecha 26 de julio de 2013 tomó el acuerdo de dar el consentimiento para su tramitación, a la tesis doctoral titulada "***Dynamically reconfigurable architectures for video coding and hyperspectral imaging systems***" presentada por la doctoranda Dña. Teresa G. Cervero García y dirigida por los Doctores D. Roberto Sarmiento Rodríguez, y D. Sebastián López Suárez.

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 veintiséis de julio de dos mil trece.

P.P. Carballo

INSTITUTO UNIVERSITARIO DE MICROELECTRÓNICA APLICADA

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

Instituto: INSTITUTO UNIVERSITARIO DE MICROELECTRÓNICA APLICADA

Programa de doctorado: INGENIERÍA DE TELECOMUNICACIÓN AVANZADA

Título de la Tesis

DYNAMICALLY RECONFIGURABLE ARCHITECTURES FOR VIDEO CODING AND HYPERSPECTRAL IMAGING SYSTEMS

Tesis Doctoral presentada por Dña. TERESA GLORIA CERVERO GARCÍA

Dirigida por el Dr. D. ROBERTO SARMIENTO RODRÍGUEZ

Codirigida por el Dr. D. SEBASTIÁN LÓPEZ SUÁREZ

El Director/a, (firma)	**El Codirector/a** (firma)	**la Doctoranda,** (firma)

Las Palmas de Gran Canaria, a 31 de Julio de 2013

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

DIVISIÓN DE DISEÑO DE SISTEMAS INTEGRADOS

TESIS DOCTORAL

Dynamically reconfigurable architectures for video coding and hyperspectral imaging systems

Teresa Gloria Cervero García

Las Palmas de Gran Canaria, julio de 2013

A mi familia, a Roberto, a mis directores de tesis, al "DSI Research lab", y a todos los que han creído en mí.

Abstract

FPGA-based embedded systems are gaining relevance for implementing a wide range of applications. Part of this success is due to their balanced compromise between performance and flexibility, but also because of their capability for exploiting the dynamic reconfigurability. Some of the most remarkable advantages of the dynamic reconfigurability are: power/size/cost reduction; hardware reusability; obsolescence avoidance and application portability.

The conservative deployment of the dynamic reconfigurability in FPGAs is centered on swapping one module by another, or by an improved version of itself. Moving beyond this strategy, it is possible to concentrate the dynamic reconfigurability strengths for dynamically adjusting the performance of a design. This issue could be very convenient in those FPGA-based embedded systems running on variable environments, since it would permit the variation of the number of processing elements involved in the execution of the application at run time. The ability of varying the number of resources, by increasing or decreasing its number, is known as scalability.

With the aim of contributing to the dynamic reconfigurability, this Thesis is focused on exploring the strengths and weaknesses of developing scalable designs, in which the scalability level might be adjusted and managed at run time. First of all, in order to study the possible positive or negative effects derived from adapting the performance of an application dynamically, this Thesis proposes a set of scalable hardware architectures for two high performance but very different applications: video coding and hyperspectral linear unmixing. However, despite the fact that these hardware solutions might be scaled dynamically, they do not incorporate any mechanism for being reconfigured by themselves. In fact, the reconfiguration management must be supervised by another mechanism. Therefore, with the objective of alleviating this fact, this Thesis also proposes a flexible module for controlling, managing and checking the dynamic reconfigurability procedure. As a whole, these contributions (the scalable designs and the reconfigurability mechanism) are focused on filling the gap in the development of FPGA-based context-aware embedded systems.

Resumen

Los sistemas empotrados basados en FPGAs están cobrando mayor relevancia en un amplio número de aplicaciones. Parte del éxito de las FPGAs se debe al equilibrio que ofrecen entre flexibilidad y rendimiento, pero también a su capacidad para ser reconfiguradas dinámicamente. Entre algunas de las ventajas más relevantes de la reconfiguración dinámica destacan: la reducción de potencia,

tamaño y coste de los dispositivos, así como el incremento de la reutilización del hardware, evitar la obsolescencia e incrementar la portabilidad de las aplicaciones.

El modo más tradicional de utilizar la reconfiguración dinámica en FPGAs se basa en la sustitución de módulos, y/o en la incorporación de versiones mejoradas de los mismos. Sin embargo, es posible ir más allá y aprovechar las ventajas que ofrece la reconfiguración dinámica para ajustar el rendimiento de un diseño. Esto podría ser muy útil para aquellos sistemas empotrados que han de trabajar en entornos variables, ya que permitiría variar el número de elementos de proceso dedicados a la ejecución de la aplicación. La habilidad de variar el número de recursos de un diseño, a través del aumento o descenso de recursos dedicados, se conoce como escalabilidad.

Con el objetivo de aportar mejoras en el campo de la reconfiguración dinámica, esta Tesis Doctoral se centra en explorar las ventajas e inconvenientes que ofrece el desarrollo de diseños escalables, en los que el nivel de escalabilidad se pueda adaptar y gestionar en tiempo de ejecución. Primeramente, con el fin de estudiar los posibles efectos positivos o negativos derivados de la adaptación dinámica del rendimiento de una aplicación, esta Tesis propone un conjunto de arquitecturas hardware escalables para dos aplicaciones de alto rendimiento, bien distintas entre sí: codificación de vídeo y desmezclado hiperespectral lineal. Sin embargo, y a pesar del hecho de que estas soluciones hardware puedan ser escaladas dinámicamente, por sí mismas no incorporan ningún mecanismo para ser reconfiguradas por sí mismas. De hecho, el control del proceso de reconfiguración ha de ser supervisado por otro mecanismo. Por lo tanto, con el objetivo de solventar este problema, esta Tesis también propone un módulo flexible para controlar, gestionar y comprobar el proceso de reconfiguración dinámica. De manera global, este conjunto de aportaciones (los diseños escalables y el mecanismo de reconfiguración) persiguen contribuir a llenar un vacío existente en el desarrollo de sistemas empotrados autónomos basados en FPGAs.

Acknoledgments

This Thesis work has been developed with the support of the Spanish National Program for Research Staff Formation (Programa de Formación de Personal Investigador, FPI). The applicant was awarded with a four years scholarship with reference BES-2009-018706 for the development of this thesis work. The Spanish Ministry of Science and Innovation (Ministerio de Ciencia e Innovación) is gratefully acknowledged for this support. This work also has been developed as part

of the DR SIMON Project, with ID TEC2008-06846-C02, and DREAMS Project, referenced as TEC2011-28666-C04. Finally, we would like to acknowledge the Centro de Electrónica Industrial (CEI-UPM), the Configurable Computing Lab (CCM Lab – VT), and the ARCO (UCLM) which each one kindly hosted a research stay, in order to collaborate in the development of this thesis work.

Caminante, son tus huellas
el camino y nada más;
Caminante, no hay camino,
se hace camino al andar.
Al andar se hace el camino,
y al volver la vista atrás
se ve la senda que nunca
se ha de volver a pisar.
Caminante no hay camino
sino estelas en la mar.

Antonio Machado 1875 - 1939

Index

Abstract i

Resumen iii

Acknowledgment v

Index of contents ix

List of figures ..... **xv**

List of tables ..... **xxi**

1 Introduction **1**

1.1 Outline *2*

1.2 Exposing the problem *2*

1.3 Motivation of research *4*

1.3.1 Scalability exploration in FPGAs 6

1.3.2 High performance computing applications 6

1.3.2.1 The deblocking filter algorithm 7

1.3.2.2 Hyperspectral linear unmixing 8

1.4 Research goals *9*

1.5 Organization of this document *12*

2 Dynamic reconfigurability **15**

2.1 Outline *16*

2.2 Reconfigurable architectures for data and computationally intensive applications *17*

2.3 Overview of the FPGA structure *22*

2.3.1 Traditional design flow 24

2.3.2 Bitstream structure 26

2.3.2.1 Configuration frame addressing 28

2.3.3 Reconfiguration interfaces 29

2.4 Management of scalable and dynamically reconfigurable hardware ... 30

2.5 Conclusions *31*

3 Exploring scalability for video coding applications: H.264/AVC and SVC **33**

3.1 Outline *34*

3.2 Exposing the problem *34*

3.3 The deblocking filter algorithm *37*

3.3.1 Deblocking filter constrained behavior 38

3.3.2 Filtering process: Boundary Strength and filter units 40
3.3.3 A detailed analysis of the deblocking filter data dependences 43

3.4 Proposed MB-level parallelization strategy 45
3.4.1 MB-level parallelization patterns comparison 49

3.5 Proposed scalable deblocking filter architecture 51
3.5.1 Basic architectural description 51
3.5.1.1 Modules description 53
3.5.2 Architectural behavior 59
3.5.3 MB reading sequence and allocation strategy 63

3.6 Implementation and results 67
3.6.1 State-of-the-art comparison 72

3.7 Conclusion and further research 78

4 Exploiting the scalability for hyperspectral image processing: linear unmixing 79

4.1 Outline 80

4.2 Exposing the problem 80
4.2.1 Fundamentals of hyperspectral imaging 83
4.2.2 Linear unmixing of hyperspectral imaging 84
4.1.1.1 Endmembers calculation and dimensional reduction 86
4.1.1.2 Endmembers extraction 86
4.1.1.3 Abundances calculation 87

4.2 Endmember extraction algorithms 87
4.1.1 Pixel Purity Index (PPI) 88
4.1.1 N-FINDR 89
4.1.2 Vertex Component Analysis (VCA) 89
4.1.1 Modified Vertex Component Analysis (MVCA) 91

4.2 FPGA-based MVCA scalable architectures 94
4.4.1 Reference architecture 94
4.4.1.1 The U_GENERATOR module 97
4.4.1.2 The F_GENERATOR module 98
4.4.1.3 The IMAGE PROJECTION module 100
4.4.2 Scalable IMAGE PROJECTION module 101

4.4.2.1 Scaling the number of spectral components of a pixel simultaneously processed (SpectSA_MVCA) 104
4.4.2.2 Scaling the number of pixels simultaneously processed (PixelSA_MVCA) 107

4.5 Comparisons and results 110
4.5.1 Endmember extraction accuracy 110
4.5.2 FPGA implementations 113

4.6 Conclusions and further research 125

5 Dynamic Resource Manager 127

5.1 Outline 128

5.2 Exposing the problem 128

5.3 State-of-the-art on reconfiguration management 130

5.4 A new dynamic reconfigurability paradigm for embedded systems 132
5.4.1 Designs specifications imposed by the DRM 133
5.4.1.1 Flexibility 133
5.4.1.2 Modularity 134
5.4.1.3 Scalability 135
5.4.1.4 Reusability 135
5.4.1.5 Reallocation 135
5.4.2 Scheduling and management challenges 136

5.5 Proposed solution: The Dynamic Resource Manager 139
5.5.1 Overview 139
5.5.2 Detailed work 141
5.5.2.1 The Scheduler 141
5.5.2.2 Allocation strategies 144
5.5.2.3 Main functionalities 147
5.5.3 Reconfiguration Engine (RE) 153
5.5.3.1 Reconfiguration Controller 154
5.5.3.2 Factory 155
5.5.4 Characterization of the Reconfigurable Region 155

5.6 A case study and results 156
5.6.1 DRM behavioral simulation 156

5.6.1.1 A co-simulation platform based on SystemC 160
5.6.1.2 DRM interfaces 162
5.6.1.3 Simulation benchmarking 163
5.6.2 FPGA-based embedded system 167
5.6.2.1 Dynamically scalable hyperspectral linear unmixing application 167
5.6.2.2 Embedded system framework and benchmarking 169
5.7 Conclusion 176
6 Conclusions 179
6.1 Conclusions 180
6.2 Further research 182
7 References 185
A. Sinopsis en español 209
A.1 Introducción 210
A.2 Soluciones arquitecturales escalables dinámicamente 211
A.2.1 Deblocking Filter 212
A.2.2 Extracción de endmembers 214
A.3 Gestión del proceso de reconfiguración dinámica 217
A.4 Conclusiones 218
B. Publications 221
B.1 National and international conferences 223
B.2 Journals and book chapters 224

List of figures

Figure 2.1 Reconfigurable hardware architectures according to their role on a more complex system 19

Figure 2.2 Structure of an FPGA 24

Figure 2.3 Programmable local and global routing tracks on a mesh topology 24

Figure 2.4 Virtex-5 LX110T layout separated in clock regions 27

Figure 2.5 Columns distribution of every section of the FPGA, and the structure of every column in frames 28

Figure 2.6 FPGA architecture 30

Figure 3.1 Vertical and horizontal edges of a MB 39

Figure 3.2 Proposed filtering order within a MB 39

Figure 3.3 Pixels notation during the horizontal and vertical filtering..... 40

Figure 3.4 Boundary Strength's calculation 41

Figure 3.5 Bottom-up dependences at data-level; a) Horizontal filtering at LOP and block-level; b) Vertical filtering al LOP and block-level; c)Data dependences at block and MB-level 44

Figure 3.6 Deblocking filter data dependences at MB-level 44

Figure 3.7 MB dependences according to the direction of the filtering execution 46

Figure 3.8 Proposed wavefront pattern in which the filtering execution among rows is synchronized 47

Figure 3.9 Proposed wavefront pattern filtering a SQCIF (8×6MBs) frame with six PEs.. 48

Figure 3.10 Defragmentation of a full image according to the number of PEs..... 48

Figure 3.11 High-level schematic of the proposed DF architecture..... 52

Figure 3.12 Coarse-grained and modular DF architecture..... 52

Figure 3.13 Functional Unit's architecture..... 57

Figure 3.14 Extended Macroblock structure..... 59

Figure 3.15 Three scalability levels of the proposed DF architecture; a) 1×1; b) 2×1; c) 3×2 61

Figure 3.16 Filtering process of a SQCIF image in stripes, including null MBs at the beginning 63

Figure 3.17 Time scheduling 66

Figure 3.18 Semifiltered MBs..... 66

Figure 3.19 Floorplanning of one PE with its bus macros on top and bottom..... 68

Figure 3.20 Area efficiency surface for several m×n array configurations 69

Figure 3.21 Number of PEs versus the clock frequency according to different video formats 70

Figure 4.1 Hyperspectral cube 83

Figure 4.2 Composition of a mixed pixel; a) 4x4 pixels' scene; b) mixed pixel based on three endmembers (pure pixels) 85

Figure 4.3 Hyperspectral image processing 86

Figure 4.4 Linear unmixing algorithm's model: simplex with three endmembers 88

Figure 4.5 PPI model in a two-dimensional space 89

Figure 4.6 VCA algorithm's representation 90

Figure 4.7 VCA pseudo-code 91

Figure 4.8 MVCA pseudo-code 94

Figure 4.9 General view of the Reference MVCA architecture 96

Figure 4.10 General view of the PROJECTIONS architecture 98

Figure 4.11 General view of the F_GENERATOR 100

Figure 4.12 IMAGE PROJECTION overview 101

Figure 4.13 Processing Element (PE); a) Main structure of a PE; b) Equivalence between the PE and the reference architecture 103

Figure 4.14 Resources of the SpectSA_MVCA according to the scalability level; a) 1 PE; b) 2 PEs; c) 4 PEs 105

Figure 4.15 Modular structure for processing several components of one pixel 106

Figure 4.16 Scalable IMAGE PROJECTION design for processing several components of the same pixel in parallel 107

Figure 4.17 Scalable IMAGE PROJECTION design for processing several pixels simultaneously 108

Figure 4.18 Scalable IMAGE PROJECTION design for processing several pixels in parallel without a comparator tree 109

Figure 4.19 Modular structure for processing several pixels at the same time 110

Figure 4.20 Cuprite image 112

Figure 4.21 Endmember extraction accuracy results 113

Figure 4.22 IMAGE PROJECTION module occupancy in SpectSA_MVCA and PixelSA_MVCA architectures 115

Figure 4.23 Synthesis result for extracting 5 endmembers 116

Figure 4.24 Synthesis results for extracting 10 endmembers 116

Figure 4.25 Synthesis results for extracting 15 endmembers 117

Figure 5.1 Desirable characteristics for dynamic and partial reconfigurable designs ... 136

Figure 5.2 Enough space in the RR for reconfiguring the system 137

Figure 5.3 Enough resources but not space in the RR for reconfiguring the system 138

Figure 5.4 There is neither space nor resources in the RR for reconfiguring the system 139

Figure 5.5 Dynamic Resource Manager structure 141

Figure 5.6 Main responsibilities of the Scheduler 143

Figure 5.7 Reconfiguration request flow 146

Figure 5.8 Behavioral flow of the Scheduler in order to include a reconfigurable module in the RR 151

Figure 5.9 Data structures of the Scheduler 152

Figure 5.10 Possible evolution of the reconfigurable region (RR) at run time 156

Figure 5.11 Traditional method of embedded system design 157

Figure 5.12 Hardware-Software co-design process 158

Figure 5.13 Co-simulation platform structure 160

Figure 5.14 Benchmarking for testing the Scheduler behavior 164

Figure 5.15 Block diagrams of the application; a) Scalable PixelSA_MVCA design; b) Reconfigurable PixelSA_MVCA 168

Figure 5.16 System structure of the MVCA 171

Figure 5.17 Reconfigurable Region structure and reconfigurable modules distribution 173

Figure 5.18 Embedded system structure 174

Figure 5.19 Winner endmembers' indexes for the 36×36 hyperspectral image 176

Figure A.1 Arquitectura escalable propuesta del DF; a) Estructura básica 1×1; b) 1×2; c)2×3..... 213

Figure A.2 Configuración 2x3 siguiendo el patrón wavefront mejorado propuesto..... 214

Figure A.3 Estructura modular de la arquitectura del MVCA..... 215

Figure A.4 Arquitectura escalable ScalableSA_MVCA..... 216

Figure A.5 Arquitectura escalable PixelSA_MVCA 217

Figure A.6 Estructura jerárquica de control para la gestión de la reconfiguración dinámica..... 218

List of tables

Table 2.I Characterization of several technologies 17

Table 2.II Characterization of several reconfigurable hardware architectures..... 21

Table 3.I Filtering operations according to the boundary strength values 42

Table 3.II Number of MB_{cycles} for filtering a whole frame by using different techniques. 50

Table 3.III MB-level parallelization comparison for filtering a full image in terms of MB_{cycles} 51

Table 3.IV Task distribution of each element 62

Table 3.V Synthesis results on a Xilinx V5-LX110T 67

Table 3.VI FPGA-based DF state-of-the-art comparison 75

Table 3.VII Standard cell-based DF state-of-the-art comparison 76

Table 3.VIII Throughput and efficiency for different DF hardware approaches 77

Table 4.I Synthesis results on the FPGA Virtex XC5VSX95t 119

Table 4.II Behavioral simulation with a synthetic image (36×36 pixels and 5 endmembers) 121

Table 4.III Behavioral simulation with Cuprite image (250×191 pixels and 14 endmembers) 121

Table 4.IV Figure of merit for a synthetic image (5 endmembers) 123

Table 4.V Figure of merit for Cuprite hyperspectral image (14 endmembers) 124

Table 5.I Communication interfaces between the microprocessor and the Scheduler. 162

Table 5.II Interfaces between the Scheduler and the Reconfiguration Engine 163

Table 5.III Testbench with simple reconfigurable modules 165

Table 5.IV Testbench with scalable modules 166

Table 5.V Synthesis results of a whole MVCA architecture with 1PE onto a Xilinx V5-LX110T 169

Table 5.VI Post Place and Route summary on Xilinx V5-LX110T 175

Acronym

ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
BRAM	Block Random Access Memory
BS	Boundary Strength
CAD	Computer Aided Design
CD-ROM	Compact Disc- Read Only Memory

CAD	Computer Aided Design
CGRA	Coarse Grained Reconfigurable Array
CIF	Common Intermediate Format
CLB	Configurable Logic Block
CRC	Correction eRror Code
DF	Deblocking Filter
DLP	Data Level Parallelism
DMA	Direct Memory Access
DPR	Dynamic and Partial Reconfiguration
DREAMS	Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems
DRM	Dynamic Resource Manager
DR SIMON	Dynamic Reconfigurability for Scalability In Multimedia Oriented Networks
DSP	Digital Signal Processor
EM	Extended Macroblock
FCLSU	Full Constrained Linear Spectral Unmixing
FGRA	Fine Grained Reconfigurable Array
FPGA	Field Programmable Gate Array
FU	Functional Unit
FM	Figure of Merit
GPP	General Purpose Processor
GPU	Graphic Processing Unit
HDL	Hardware Description Language
HDTV	High Definition TeleVision
HLS	High Level Synthesis
HWICAP	HardWare Internal Configuration Access Port
HySIME	Hyperspectral Signal subspace Identification by Minimum Error
IC	Input Controller
ICAP	Internal Configuration Access Port
ILP	Instruction Level Parallelism
IM	Input Memory

IP	Intellectual Property
ISE	Integrated Software Environment
LOP	Line Of Pixel
LSE	Least Square Error
LSU	Linear Spectral Unmixing
LUT	Look-Up Table
MB	Macroblock
MNF	Maximum Noise Function
MPMC	Multi Port Memory Controller
MVCA	Modified Vertex Component Analysis
NAPC	Noise Adjusted Principal Components
NCD	Native Circuit Description
NGD	Native Generic Database
NPI	Native Port Interface
OC	Output Controller
OM	Output Memory
OPB	On-chip Peripheral Bus
PCA	Principal Component Analysis
PCB	Printed Circuit Board
PE	Processing Element
PLB	Processor Local Bus
QCIF	Quarter Common Intermediate Format
RISC	Reduced Instruction Set Computer
RE	Reconfiguration Engine
ROM	Read Only Memory
RU	Reconfigurable Unit
RR	Reconfigurable Region
RTL	Register Transfer Level
SNR	Signal-to-Noise Ratio
SoC	System-On-Chip
SQCIF	Sub-Quarter Common Intermediate Format
SVC	Scalable Video Coding

TLP	Task Level Parallelism
UCF	User Constrain File
UHDTV	Ultra High Definition TeleVision
VLIW	Very Long Instruction Word
VCA	Vertex Component Analysis

Chapter 1

Introduction

This chapter presents the most significant strengths and weaknesses associated to the dynamic reconfigurability. Based on that, the motivations as well as the goals of this Thesis are outlined. In the end, the organization of this work is disclosed.

1.1 OUTLINE

The technological evolution experienced in the last decades has motivated that smart electronic devices showed up in the market. That is the main reason why consumers and vendors claim for more functional and versatile devices. These demands require increasing parameters such as performance, flexibility, portability, connectivity and efficiency, but also combining them under the same device. On the other side, the development industry prioritizes aspects as the non-recursive costs, area and power savings. From the industry and the academia, the efforts for solving this dichotomy between flexibility and performance have been focused on proposing new designing techniques and methodologies that promote the development of balanced and efficient solutions. Thus, despite the fact that ASIC devices are still being the primary target technology in the market, such as [HR12], [Mcc13] and [UBM13] studies demonstrate, their hegemony is based on developing rigid and efficient designs. Therefore, they are too strict for implementing portable, flexible and efficient embedded systems in order to fulfill with users' expectations. A popular alternative is using reconfigurable hardware, in which the most common technology is the Field Programmable Gate Array (FPGA) [WIN10]. The reconfigurable technology might change the functionality of a hardware design over time more than once by customizing the logic and connections at run time. Therefore, the exploitation of the benefits offered by the FPGA devices facilitates the development of more powerful embedded systems capable of fulfilling with the high-demanding expectations of consumers, vendors, but also the data-intensive applications [INS12].

1.2 EXPOSING THE PROBLEM

As a consequence of the normalization and standardization processes, carried out along the last years, the dominance of computing systems in the market is being replaced by a new market of smart embedded systems, mobile devices and large-scale data centers [DBB+13]. The clearest example of this fact is reflected by the growth of mobile computing devices, evidenced by the spread of tablets, mobile devices and smart phones. At the same time that software

and hardware devices have improved their characteristics, the transmission supports have also been improved since the wireless networks have grown everywhere, and the up and downlinks bandwidths as well as the transmission data transfers rates have augmented significantly in a short period of time [ITU11]. These circumstances have motivated that multimedia field has become one of the most active in the industry, due to its relevance in the consumer market [KM12][Jak13]. However, these enhancements introduce certain level of complexity referred to the performance or functionality adaptation of devices to the demands of their running applications under environmental variations, such as frequently happens in wireless networks, or the Internet. In addition, the capability of providing a real time response to those dynamic changes, it is being day by day a determinant aspect for many systems. Unfortunately, traditional hardware/software co-design methodologies and tools do not facilitate the combination of both solutions (embedded systems performance with run time hardware adaptability capabilities) [INS13], since the hardware designs tend to be designed using ASICs. Consequently, in all those situations in which the running constraints are relaxed, compared to those ones imposed during the early designing and development stages, many of the available resources remain idle and underused. Accordingly, the combination of all these issues (adaptability and real time constraints) opens the window to a new framework, which requires a review of traditional concepts related to embedded computing systems on chip (SoCs), but also of the relationship and the interaction between these systems, the users and the environments.

Thus, Field Programmable Field Arrays (FPGAs) have gained popularity along these recent years in numerous sectors of the market [HO10] [Boa12]. Part of this success is due to their combination of the best features of pure software solutions, like traditional General Purpose Processors (GPPs), and hardware solutions, like gate arrays, under the same device. The balanced tradeoff between flexibility and performance [KTR08], and their reconfiguration capability, become FPGAs into good candidates for the development of embedded SoCs. In addition, some of these market devices offer the possibility of a

dynamic reconfigurability, which means modifying their configuration whereas the rest of the system remains working.

Moving beyond the conservative exploitation of the dynamic reconfigurability in FPGAs, based on the substitution of one module by another, or even swapping it by an improved version of itself; it is possible to take advantage of the dynamic reconfigurability strengths for adjusting the system functionality or modifying the number of processing elements involved in the execution of the running application. More specifically, the capability of incrementing or decrementing some of the properties (physical, structural or behavioral) of a module is known as scalability.

Unfortunately, the dynamic reconfigurability is still an immature technology, in which there are certain methodological and technological deficiencies that make its use on manufactured products still poor [UBM13].

1.3 MOTIVATION OF RESEARCH

As soon as the technology and the fabrication processes have evolved, the complexity of hardware devices has increased rapidly by including higher number of resources on the same die [XWP10]. A very clear example of this fact is the explosion of more dense and powerful devices in the market, such as modern FPGAs [GBI11].

Nowadays, one of the main trends goes through combining programmable logic technology, such as an FPGA, with traditional processing elements, like a microprocessor, which works well for a wide range of complex embedded systems. Some commercial solutions that follow this strategy are Intel Atom [ATOM13], Microsemi SmartFusion [MSF13], Xilinx's Zynq [XZY13], and Altera SoC FPGA [ASOC13]. One of the causes why the FPGAs are attractive is the right mix of performance, flexibility, and price. A primary benefit of FPGAs for processing is their reconfigurability that offers a mechanism for hardware upgrades and product differentiation, which extend product life in a world of evolving interfaces and standards. In addition, this feature remains much more adaptable to design changes than the processor-only or ASIC approach [INS13].

Furthermore, the parallel nature of FPGAs allows that multiple tasks operate in a truly concurrent fashion using dedicated processing elements. In this sense, by using the dynamic reconfigurability is possible to explore the benefits of the scalability in a hardware design. This means, allowing for varying the number of hardware resources performing simultaneously in the same FPGA. Moreover, in order to maximize the benefits of merging the processor and FPGA, designers must address several considerations. Thus, apart from the computational requirements of these applications, the final systems must be able to fulfill the expectations of both the users and the industry, where the most relevant are listed below:

1. Flexibility: it allows the implementation of multiple and diverse designs and functionalities. Even more, it also means supporting different kinds of solutions for overcoming the same problem.
2. Portability: this characteristic is related to the fact that one solution might migrate to a different technology without having to redesign it completely from the scratch.
3. Adaptability: it facilitates the adjustment of the behavior or the performance of the system, in order to accomplish with the proposed tasks, according to the environmental fluctuations.
4. Multitasking: this feature allows that several functions, tasks and/or applications run onto the same device, but in all the cases achieving a tradeoff between performance and flexibility.
5. Efficiency: in hardware solutions, this term is intimately related to the clock frequency, the memory data bandwidth, the usability of the available resources, and power and silicon savings.
6. Autonomy and independency: these properties introduce certain degree of intelligence to the device, in the sense that it might be able to operate without any other external and complex system. A clear example of applications that should exploit these characteristics are all those that are executed on satellites or even those located on places with difficult accesses.

1.3.1 Scalability exploration in FPGAs

The future of high performance computing is likely to rely on the ability to efficiently exploit huge amounts of parallelism. In this sense, there exist different parallelization strategies; such as Instruction Level (ILP) [RF93], Task Level (TLP) [GP95] and Data Level Parallelism (DLP) [PGT07]. The first two parallelization strategies are widely used on systems based on General Purpose Processor (GPP) units. Nevertheless, the TLP and DLP strategies are most extended on Graphical Processing Units (GPUs) or reconfigurable devices. The goal of using these parallelization strategies lies on distributing the computation in space rather than in time all over the device. This policy allows to increase the number of parallel processing elements that performs operations concurrently and, as a result, the computation of the overall implemented application is accelerated.

Taking advantage of these parallelization policies more flexible approaches are possible by adjusting the level of parallelism according to the necessities of the system; in other words, scaling the solution. Thus, the flexibility of the system is enormously increased, since this adjustment increases the hardware reusability, the adaptability and the efficiency. Some of examples of high-parallelized and scalable solutions can be found in [OTR+10], [BBD09] and [EBS+11].

1.3.2 High performance computing applications

All high performance computing applications are characterized by their huge amount of information to process, their long latency for processing the data, but also by the necessity of ensuring consistent and reliable results. As it was previously mentioned, one of the best ways to face these challenges is to provide highly parallelized solutions, in which the exploitation of the scalability might play a relevant role on the consecution of this issue. In this sense, by combining the flexibility and the reconfigurability features of the FPGAs, together with the powerful of the TLP or/and the DLP strategies, it is possible to accelerate and adapt the performance of high performance computing applications.

With the objective of covering diverse applications for demonstrating this affirmation, this Thesis has selected two well-differentiated application domains. One of them is focused on video and imaging applications, more specifically, one of the latest video coding standards, the H.264/AVC [ITU-T07] and its extension the Scalable Video Coding standard [ITU-T09] (SVC). The second choice is related to space applications, more precisely those that benefit from processing hyperspectral images captured by a remote sensing sensor [PBB+09]. This kind of image is used for many applications such as Earth observation [HAA+12], environmental studies [TL11], and geology [MWR+12], among many others.

1.3.2.1 The deblocking filter algorithm

The H.264/AVC standard allows reducing the transmission rates up to 50% and 35% compared to MPEG-2 and MPEG-4 standards [KA03][OBL+04]. However, the better performance of the H.264/AVC and the SVC standards, the higher complexity they demand. Hence, the current trend to deal with this emerging complexity goes through executing several tasks of the encoding/decoding loop simultaneously, or parallelizing the execution of those tasks with higher computational cost, in terms of the complexity of their operations or the amount of data that they have to process [Por05].

As it has been demonstrated in different studies, like [HJK+03], [SMW07], [WDG+10] and [SCL+11], the deblocking filter algorithm represents one of the most time consuming, complex and critical tasks in the encoding and decoding loop for both the H.264/AVC and the SVC standards. Therefore, this is a perfect candidate to be implemented in hardware, such as a large amount of publications demonstrates [HCH03], [CCH06], [TVM09], [MBT+11], [SFZ12] and [LZZ+12].

A common approach to cope with stringent computational requirements in an energy efficient manner is custom hardware acceleration by means of application specific integrated circuits (ASICs). Despite the fact that these devices are much more efficient than general purpose processors, they are inflexible. Video systems are preferably adaptable because the application or the characteristics

of the application can be modified and are often not established or known beforehand. An interesting option to reconcile the conflicting requirements of computational power and flexibility is using FPGAs. The research community is investigating the benefits of using this kind of reconfigurable hardware. In some cases, the scalability or the parallelization relies upon a whole H.264/AVC encoder or decoder, such as in [RSK12] and [Eec07] works. In other cases, the scalability is directly applied over the deblocking filter, like occurs in [KHL10] and [VCK10].

1.3.2.2 Hyperspectral linear unmixing

According to the trends of the spatial industry [Boa12], the use of FPGAs in current missions is increasing due to their size, high function densities and fast working speeds, and reduced power consumption [FLG13]. Thus, many applications related to these issues have been implemented on FPGAs [LVG+13], in order to get a compromise between performance and costs.

Into the wide range of purposes of those space missions, hyperspectral remote sensing applications are gaining relevance due to they make possible to analyze surfaces remotely. These kinds of applications present two main challenges. First of all, the amount of information collected by the hyperspectral remote sensor is huge [SD11], like in the case of the AVIRIS [AVI07]. Then, the spatial resolution of the sensor is not able to distinguish distinct materials when they are close each other. This fact causes the appearance of mixed pixels. In order to identify and separate those mixed pixels into their components, spectral linear unmixing algorithms [BPD+12] have gained in popularity. Furthermore, some of them have been implemented in FPGAs, like is the case of [SJR10], [DP11], [GMR+12], [MZM+13] and [VDL+13].

However, all these solutions do not maximize the benefits of using the dynamic reconfigurability, and suffer from some of the following issues:

- **Modularity**: A whole design should be separated in several modules according to their primary task, since this policy allows an easier upgradeability and reliability for future improvements.

- **Scalability**: The exploitation of the scalability feature facilitates hardware reusability, but also area and power savings.
- **Homogeneity**: The behavior and connectivity of all the modules of a design must remain unaltered, and be always the same, independently of the scalability level demanded in the system.
- **Dynamic and partial reconfiguration:** Combining traditional dynamically reconfigurable designs, with the concepts of the scalability and partial dynamic reconfigurability (DPR), it would be possible to modify the number of processing elements (increasing or decreasing their number) that are running in parallel.
- **Run time reconfigurability management:** Despite the fact that the reconfiguration on an FPGA might be done statically and/or dynamically by using external interfaces; context-aware or/and self-adapting embedded systems require an internal reconfiguration process for loading the required configurations without the intervention of any external control. In this way, there are several internal reconfiguration engines, such as [CLF08], [AM09], [FFC+11], and [KOM11]. However, the control and management of the dynamic reconfigurability process is not completely solved yet.

1.4 RESEARCH GOALS

The development of this research attends to different interests, but all of them focused on contributing to relieve some of the lacks in the dynamic reconfigurability. Thus, the main goal behind this research is to offer solutions in order to facilitate the development of autonomous and flexible FPGA-based embedded systems on chip (SoC); but at the same time doing and efficient, dynamic and intelligent use of the available hardware resources at run time. In this sense, it is necessary to introduce the hardware reconfigurability feature as the base for being able to focus the solutions toward adaptable and scalable environments with real time constraints within consistent, generic and flexible

working scenarios. With the goal of achieving this purpose, this research work is focused on reaching the following objectives:

- Contextualize this work by highlighting the strengths and weaknesses of using reconfigurable hardware for implementing high performance computing applications. In this way, it is possible to establish the fundamentals in which the rest of this research work lies on.
- Provide a set of hardware architectures characterized by their modularity, flexibility, and scalability for performing data and computationally intensive applications. The approaches belonging to this set of solutions do neither necessarily participate nor exploit from the dynamic reconfigurability feature of the FPGAs. The expected contributions of these architectural proposals are related to the characteristics that high performance applications should fulfill before they are able to benefit from the dynamic reconfigurability.
    - In this sense, as a first step, a review and an analysis of the state-of-the-art of the architectural solutions related to the selected application should be established. At this point is important to pay attention to the parallelization strategies exploited in the collected designs, but also the technique applied for improving the final performance of the system.
    - Select the most time consuming or intensive tasks, within the selected applications, in order to accelerate their execution by taking advantage of the benefits of using reconfigurable hardware designs.
    - The next step is to study different ways to parallelize the selected application, considering that the solution must be homogeneous and scalable.
    - Create (design and develop) a specific hardware solution for each one of the selected applications capable for adapting the number of hardware resources dynamically used at run time.

This feature is very important in order to adapt the final performance of the solution to different situations. On the one hand, the scalability has to be exploited on a deblocking filter algorithm, as part of a video codec standard application. On the other hand, another but different data intensive application has to be analyzed, a linear unmixing algorithm as part of a hyperspectral image processing application.

- Explore the scheduling and management tasks related to the reconfigurability in order to adapt the performance and/or functionality of dynamically reconfigurable embedded systems at run time.
- Identify the most general and relevant situations that might occur during the execution of a dynamically reconfigurable system. Based on this information, determine the necessary specifications and procedures that are required for controlling the hardware reconfigurability process in order to prevent the system from failures.
- Design a scheduling unit responsible for receiving reconfiguration requests from the rest of the system, and acting in consequence in order to provide a successful reconfiguration process. In this sense, this unit has to be able to organizing the information involved in the process, analyzing different strategies to use, and deciding the most efficient in order to determine how to proceed with the reconfigurable hardware.
- Integrate the scheduling unit as part of a more complex entity, a dynamic resource manager (DRM), in order to complete all the hardware reconfiguration process in self-adapting or context-aware FPGA-based embedded SoCs.
- Demonstrate the viability of all these goals by joining them all together under the same system. Thus, a dynamically reconfigurable FPGA-based embedded system will be proposed, in which one of the scalable designs previously mentioned will run. Moreover, the scalability adaptations of the application will be controlled by the DRM.

1.5 ORGANIZATION OF THIS DOCUMENT

The present document is structured into six chapters, including this introductory one dedicated to present the problematic and the motivations for developing this PhD. Every chapter is dedicated to a specific contribution of this PhD research work.

Chapter 2: **Dynamic reconfigurability**

This chapter covers basic concepts regarding the reconfigurability, in an attempt to offer a clue about the sense of using this feature/technique. This will answer a series of easy questions such as why, where and how the reconfigurability might be used. In the beginning, the evolution of the dynamic reconfigurability is reviewed, starting from its origins to the present days. Thus, the final objective of this chapter is to contextualize the present research work, by highlighting the relevance of taking advantage of the dynamic reconfigurability, but also exposing the weaknesses and lacks of this field that make difficult moving this feature from the academia to the market.

Chapter 3: Exploiting scalability for video coding applications: H.264/AVC and SVC

Within the set of data-intensive applications, the latest video codecs standards play a relevant role into the development of modern multimedia applications. The most extended video standards are the H.264/AVC and its extension, the SVC. Both of them are very intensive in computation, since they need to perform many operations repeatedly and, in most of the cases, complex ones. Their importance might be shown through an overview of the state-of-the-art in this field. Thus, one of the goals of this chapter is to review the previous proposals on this area, and then presenting the proposed work. The main trend is moving toward generic solutions, which are capable of solving the same kind of problems, but in this case with different dimensions by adjusting some settings parameters. In other words, the behavior or the performance of the solution should be scaled. In consequence, this chapter presents a scalable hardware solution for the *deblocking filter* algorithm. This element has been

selected due to it is the most computational intensive block within the aforementioned decoders.

Chapter 4: Exploiting the scalability for hyperspectral image processing: linear unmixing

This chapter explores the viability of the scalability for processing a different high performance computing application than the one studied in the previous chapter. First of all, a review of the state-of-the-art in this kind of algorithms is presented in order to contextualize the proposed work. Then, as a part of the research work of this Thesis, two different scalable approaches are developed for a linear unmixing algorithm (more specifically the Modified Vertex Component Analysis (MVCA) algorithm), in charge of extracting endmembers from a hyperspectral image. The scalability is determinant for saving resources, silicon area, cost and gaining in flexibility in to the embedded SoCs design. Whether those scalability changes want to be applied at run time, it is mandatory to use the dynamic reconfigurability.

Chapter 5: Dynamic Resource Manager

Due to the fact that the scalability is a necessary condition for exploiting the dynamic reconfigurability, but not sufficient, this chapter outlines the requirements that one hardware embedded design should fulfill in order to maximize the benefits of the dynamic reconfigurability. However, controlling the dynamic reconfiguration process is not trivial, and its complexity is ever growing when context-aware embedded SoCs are considered. With the goal of contributing to overcome this challenge, and after reviewing some of the proposals present in the state-of-the-art, this chapter contributes by proposing a dynamic resource manager. This element is responsible for scheduling the tasks involved in the reconfiguration process, and ensuring their correct execution in the silicon. It is a complementary element of the whole architecture focused on organizing the hardware reconfigurability. In addition, the behavior of the presented contribution is independent from the intrinsic characteristics of all the reconfigurable applications implemented in the embedded SoC, though it has to store certain information of them.

Chapter 6: Conclusions and future work

Finally, the collection of the contributions provided in this PhD, and their relevance into the dynamic reconfigurability field, are summarized. At the end of this document, further research works are proposed, which might complement and enhance some of the aspects developed in this PhD.

Chapter 2

Dynamic reconfigurability

This chapter covers basic concepts regarding the reconfigurability in order to offer a clue about what is the sense of using this feature in embedded systems. Thus, the final objective of this chapter is to contextualize the present research work, and also expose the weaknesses and lacks of this field that make difficult moving this feature from the academia to the market solutions.

2.1 OUTLINE

The impressive ascent in relevance of the high performance computing world in our society is indisputable. In a short period of time, this sector has quickly progressed and, in order to cope with their stringent demands, different hardware devices have been developed. However, not every target device is apt for fulfilling the demands associated to many of these high performance applications. Unlike traditional solutions, dynamically reconfigurable architectures appear as ideal candidates to deal with these necessities.

Some of the most relevant advantages of using the dynamic reconfigurability might be summarized as follow:

1. First, the fact that reconfigurable devices might be repeatedly configured allows reusing the same die for different matters, reducing costs and minimizing the time-to-market.
2. The reconfigurability by its own allows to modify a system, including future releases in a post-fabric stage. Moreover, the dynamic reconfigurability accelerates the reconfigurable hardware adjustment by updating the system at run-time, in many cases supporting real time constraints.
3. Another advantage directly related to the dynamic reconfigurability includes the possibility of implementing diverse functionalities in the same device, and swapping among them according to the system requirements.

Within the wide range of hardware choices, and according to the information shown in Table 2.I, the reconfigurable hardware balances performance, power consumption, flexibility, design time and final costs. Therefore, although Application Specific Integrated Circuits (ASICs) and General Purpose Processors (GPPs) represent fully functional systems, they are not appropriated to assume the stringent demands of computationally intensive applications, in terms of flexibility, performance and time-to-market. Another implementing device widely used nowadays is the Graphic Processing Unit (GPUs).

For these reasons, the research activity has developed useful reconfigurable hardware systems and platforms capable of supporting more powerful applications along these years, not only focusing on FPGAs but also on custom and semi-custom reconfigurable architectures.

Table 2.I Characterization of several technologies

Device	Performance	Cost	Power consumption	Flexibility	Design (NRE)
ASIC	High	High	Low	Low	High
DSP	Medium	Medium	Medium	High	Medium
GPP	Low	Low	Medium	High	Low
GPU	High	High	High	Medium	Medium
Reconfigurable HW	Medium	Medium	Medium	Medium	Medium

2.2 RECONFIGURABLE ARCHITECTURES FOR DATA AND COMPUTATIONALLY INTENSIVE APPLICATIONS

Within reconfigurable hardware devices, and according to the granularity aspect, it is possible to differentiate two types of devices. These are Coarse-Grained and Fine-Grained Reconfigurable Arrays, CGRAs [TSV07] and FGRAs [TSS07] respectively. The former are usually designed as custom or semi-custom devices, whereas in the FGRA group the Field Programmable Gate Arrays (FPGAs) are the most extended and commercially available devices.

The granularity of a reconfigurable fabric reflects the size of the smallest block unit of which a device is made, based on its data width and computational capability. According to this definition, the granularity is broadly divided into two categories: fine and coarse-grained. The former uses basic logic blocks with a data width of small number of bits; whereas the coarse-grained architectures consist of more complex and bigger block units working at word-level. When a fine-grained approach is used, it is possible to manipulate bitwise, so every single bit can be separately routed/used. A coarse-grained design, in contrast, usually does not allow bitwise manipulation, being word or sub-word manipulation the more general approach. Traditionally, the FPGAs are defined as fine-grain devices, whereas on the other side there are the CGRAs. In short, CGRAs can be seen as statically or dynamically reconfigurable coarse-grained FPGAs, with direct interconnections between processing elements (PEs) that need to be pro-

grammed explicitly. Bit-level programmability is less efficient than the coarse-grain one in terms of routing area and configuration overhead. However, fine-grain devices are well suited for application at bit or irregular sized data manipulation. On the contrary, coarse-grain devices reduces the configuration time, as well as they decrement placement and routing complexity tasks. Despite the fact that FPGA-design is faster and easier than the CGRA, the research community is focusing their efforts on both directions.

From a system level point of view, and following the classification proposed in [VK09], these reconfigurable architectures (CGRAs and FPGAs) might be classified, according to their role into a more complex system, as: external processing unit, co-processor, reconfigurable functional unit or embedded processor. A graphical structure of these four cases is represented in Figure 2.1

1. External processing unit: The system is separated into two cores. On the one hand, the processor is responsible for running software tasks and controlling the whole system. On the other hand, the reconfigurable architecture is separated from the processor and communicated with it by using the I/O facilities of the processor. Therefore, the reconfigurable array acts as a peripheral to the processor. This kind of scheme is suitable for all those cases in which the communication between both cores is not continually needed.

2. Co-processor: This structure shares the same memories between the processor and the reconfigurable architecture, but also they have the same connection with the rest of the system.

3. Reconfigurable Unit: This system is similar to the co-processor one, but with the exception that in this case the reconfigurable architecture belongs to the processor. Therefore, it is a functional unit in the system.

4. Embedded processor: On the contrary than in the reconfigurable unit system, in this case the processor is embedded as part of the reconfigurable architecture. Here, the processor might be a soft or a hard-wire core within the reconfigurable architecture. This system-level system is a trend in nowadays FPGAs, since this pattern makes them more coarse-grained.

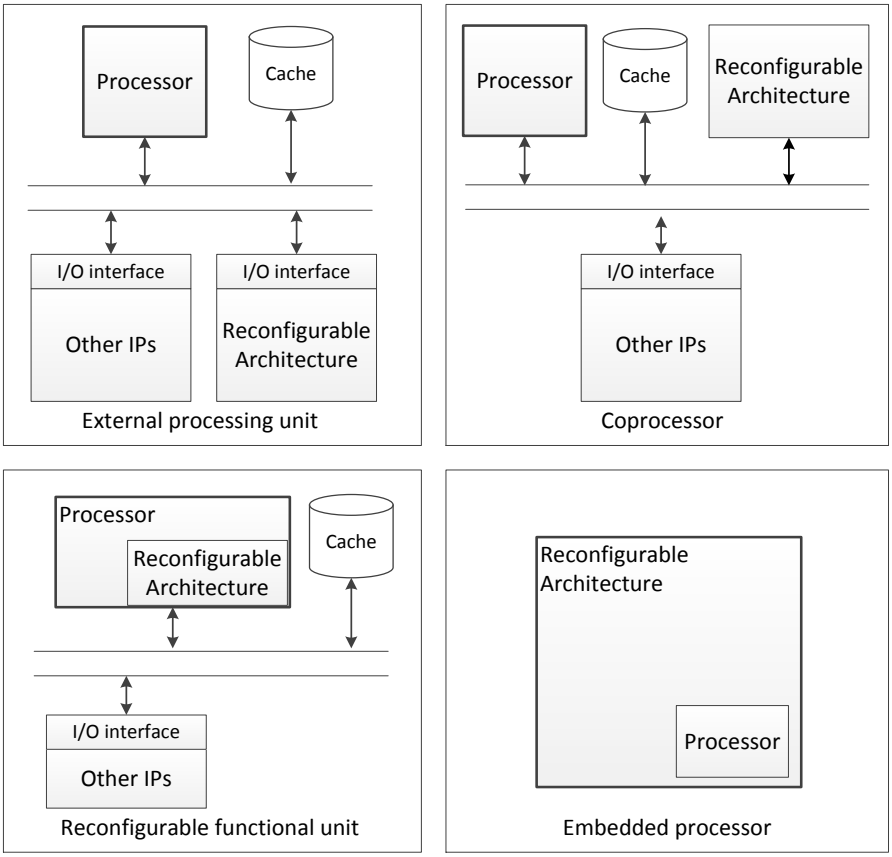


Figure 2.1 Reconfigurable hardware architectures according to their role on a more complex system

On the other hand, according to the reconfigurable hardware design itself, three different categories of reconfigurable designs might be established:

1. Array of functional units (FUs): These architectures are created from chains of interconnected *functional units*, which are replicated to create more complex structures, such as 2D-arrays. This kind of structure is totally dependent on some external resources. For example, it is managed and controlled by a host processor situated off-the-chip, which also supervises the reconfiguration process. Its data or configuration contexts are also stored on external memories. Regarding to this, it is very important to design an efficient communication channel between the host and the array of FUs, in order to accelerate data transfers among them. Consequently, an array of functional units is a useful and flexible system when the running application meets two important conditions: it

is very intensive in computation and, simultaneously, it is quite independent of the processor without constant interruptions to the processor.

2. Coprocessor or hybrid architectures: The design complexity is something higher than on the previous category, because the constraints are higher too. The whole system is formed by two main parts: a processor core, which is usually a RISC or a VLIW processor, and a reconfigurable hardware core. Both cores are often tightly coupled and the control of the whole system is handled and managed by the processor core. The communication protocol and the interconnection network among processor-reconfigurable hardware and memory-reconfigurable hardware are critical decisions, since a bad selection could determine a slow data transfer. In this case, some internal memories are incorporated to the architectural structure. Reconfigurable hardware core has associated some configuration caches, while the data are stored into SRAM-memories. Generally, these structures allow certain level of autonomy, operating without an external intervention.

3. Array of processors: These systems are formed by a set of hard or soft interconnected processors. The power of this architectural structure is obtained at the cost of an important increment of the design complexity. All these processors may operate independently in parallel one to the other, executing different tasks. Although they also may combine their resources to execute the same functions. The major difficulties associated to these systems are the management of the data dependences among processors running in parallel and the assignment of the instructions to its proper processor. It is difficult to design these architectural structures because there are a lot of variables implicated into the correct execution of the processors. This architectural structure is useful for implementing parallel applications with low data dependences.

Regarding the inherent properties of the reconfigurable hardware solutions, these ones are almost ideal candidates for implementing data intensive and high performance computing applications. In this sense, the state-of-the-art of the reconfigurable hardware is full of CGRA or FPGA solutions for accelerating these kinds of applications. Briefly, Table 2.II summarizes the research activity in this field during the last decades.

Table 2.II Characterization of several reconfigurable hardware architectures

Ref.	Name	Tech.	Hardware design level	Granularity	Reconfigurability	Topology	App.
[RTF+00]	**RAMP**	N/A	Array of FUs	Coarse	Partial dynamically	2-D Mesh	Multimedia
[SCM00] [BG99]	**PipeRench**	500nm	Hybrid arch.	Coarse	Partial dynamically	Hierarchical	Stream-based
[BG01]	**DreAM**	350nm	Array of FUs	Coarse	Partial dynamically	Hierarchical	Mobile signal processing
[BEM+03]	**XPP**	90nm	Array of FUs	Coarse	Partial dynamically	2-D Mesh	Multimedia
[HSM03]	**Montium**	130nm	Hybrid arch.	Coarse	Fully dynamically	1-D Mesh	Multimedia
[KAD03]	**ARDOISE**	Atmel AT40K40	Hybrid arch.	Fine	Partial dynamically	1-D Mesh	Image processing
[EFX+04] [CFF+99]	**RaPiD**	180nm	Array of FUs	Coarse	Fully dynamically	1-D Bus	Data intensive
[BLM+04]	**RAW**	180nm	Array of processors	Coarse	Static	2-D dynamic network	Data intensive
[KRL+06]	**3D-SoftChip**	180nm	Array of processors	Coarse	Fully dynamically	2-D Mesh	Signal processing
[LCB+06]	**XiRISC**	130nm	Hybrid arch.	Fine	Fully dynamically	2-D config.	Multimedia
[PNK+06]	**MorphoSys**	130nm	Hybrid arch.	Coarse	Fully dynamically	2-D Mesh	Data intensive
[SBB06]	**QUKU**	Xilinx V4	Hybrid arch.	Fine	Fully dynamically	2-D Mesh	Data intensive
[HCE07]	**FLEXWAFE**	Xilinx V2 -Pro	Array of processors	Fine	Fully dynamically	1-D Mesh	Stream-based
[KBW+07]	**ECA**	90nm	Array of FUs	Coarse	Partial dynamically	NoCs	Multimedia
[LPC07]	**MORA**	90nm	Array of FUs	Coarse	Fully dynamically	Hierarchical	Multimedia
[SWS05]	**DAPDNA-2**	130nm	Hybrid arch.	Coarse	Partial dynamically	2-D	Stream-based
[BGN08] [BCR+06]	**Butter**	Altera Stratix-II	Array of FUs	Coarse	Fully dynamically	configurable	Signal processing

Ref.	Name	Tech.	Hardware design level	Granularity	Reconfigurability	Topology	App.
[PSD08]	**DART**	130nm	Hybrid arch.	Medium	Fully dynamically	Hierarchical	Multimedia
[MSV+08]	**ADRES**	90nm	Hybrid arch.	Coarse	Fully dynamically	2-D Mesh	Data intensive
[LH09]	**SmartCell**	130nm	Array of FUs	Coarse	Partial dynamically	Hierarchical	Data intensive
[HSC+10]	**Sonic-on-chip**	Altera Flex 10k50	Array of processors	Coarse	Fully dynamically	1-D Bus	Image processing
[PMB11]	**SYSCORE**	90nm	Array of FUs	Coarse	Static	2-D Mesh	Biosignal processing
[AA12]	**BiLRC**	90nm	Array of FUs	Coarse	Static	2-D Mesh	Image processing
[PDM12]	**ReMORPH**	Xilinx Spartan 6	Hybrid arch.	Coarse	Partial dynamically	2-D Mesh	Stream-based

The CGRA usage has been limited due to the lack of commercial CGRA circuits. Thus, some works such as [FVM+11] and [PLS+11] propose a virtual and dynamic CGRA implemented on top of an FPGA. In this way, it is possible to use commercial-off-the-shelf FPGA devices combined with the advantages of customized CGRAs [PLS+11].

Despite the fact that several vendors provide a variety of FPGA devices, not all of all them support the dynamic reconfigurability [DH03]. Xilinx FPGAs have been selected for developing this research work, since modern families, such as the Virtex-5, permit exploiting the dynamic reconfiguration, but also because Xilinx is the most sold in the market [UBM13] and provides a set of support tools [XIL12].

2.3 OVERVIEW OF THE FPGA STRUCTURE

An FPGA is a flexible device designed to be configured after manufacturing and composed by programmable blocks of different types, which are distributed all over the die in columns. This 2D array of blocks includes general Configurable Logic Blocks (CLBs), block RAM memory (BRAM) and multipliers or Digital Signal Processor blocks (DSPs), surrounded by a programmable routing fabric that al-

lows blocks to be programmable interconnected. There are also programmable Input/Output Blocks (IOBs) that connect the chip to external devices, like Figure 2.2 shows. CLBs and DSPs, similar to a processor's Arithmetic Logic Unit (ALU), can be programmed to perform arithmetic and logic operations like add, multiply, subtract, compare, etc. Unlike a processor, in which the architecture of the ALU is fixed and designed in a general-purpose manner to execute various operations, the CLBs can be programmed in such a way that it can provide functionality as simple as that of a transistor or as complex as that of a microprocessor depending on the operations needed by the application. In addition, routing paths in FPGAs consists of horizontal and vertical channel tracks of varying lengths that can be interconnected via electrically programmable switches. Depending of the length of these wires, Figure 2.3 depicts, the FPGA distinguishes between local tracks (those ones that connect adjacent elements), and global tracks (those wires used for providing an indirect connectivity between far neighboring elements).

Advancements in silicon, software, and the design of specific and efficient intellectual property (IPs) cores have proven Xilinx FPGAs to be good solutions for accelerating applications on high performance embedded computers [XWP10]. High performance embedded computers are computers that are incorporated in equipment or an appliance to perform specific compute and data intensive tasks. Some industries demand using these kinds of systems, such as defense, communication, medical imaging and financial services. Due to the interest of this Ph.D. has been focused on particular intensive data applications for embedded systems, the proposed solutions have been implemented on a Xilinx Virtex-5 FPGA. Therefore, the rest of this section is specifically focused on describing in detail several aspects of this particular device.

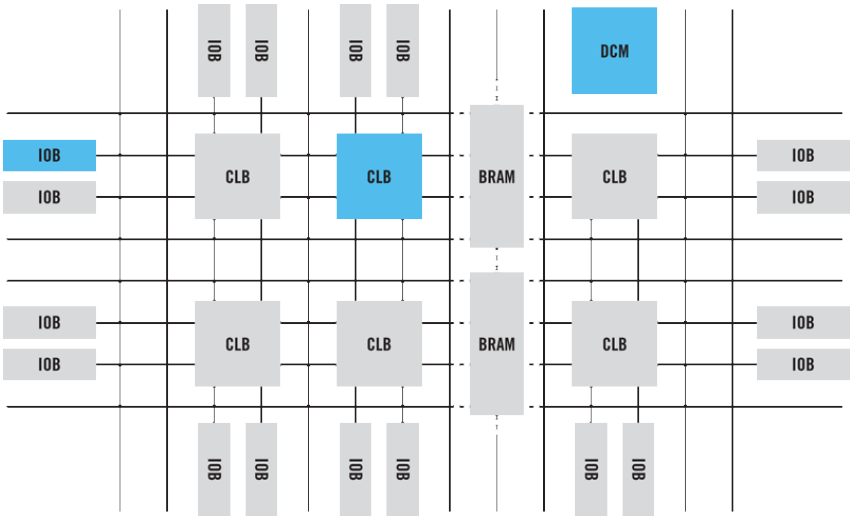


Figure 2.2 Structure of an FPGA

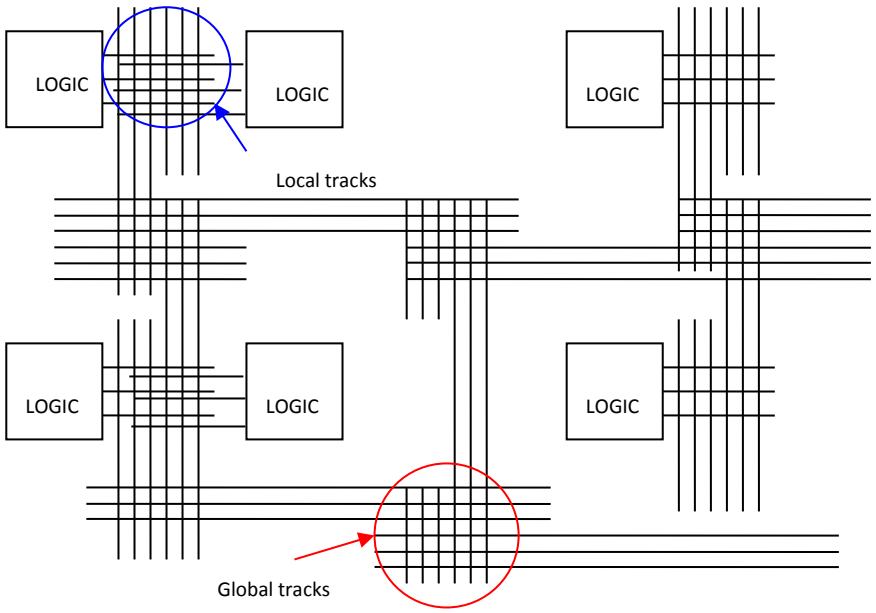


Figure 2.3 Programmable local and global routing tracks on a mesh topology

2.3.1 Traditional design flow

One of the most important advantages of using FPGAs compared to other technologies is the fact that there are several Computer Aided Design (CAD) tools that facilitate the design flow. These tools are usually provided by design

automation companies or even by the academia. A generic design flow of an FPGA includes these steps [PM11]:

- System design: the designer has to decide what portion of his functionality has to be implemented on the FPGA and how to integrate that functionality with the rest of the system.
- I/O integration with the rest of the system: I/O streams of the FPGA are integrated with the rest of the system onto a physical Printed Circuit Board (PCB).
- Design description: designer describes design functionality either by using schematic editors or by using one of the various Hardware Description Languages (HDL), like Verilog or VHDL.
- Synthesis: once the design has been defined, the CAD tools are used to implement the design on a given FPGA. Synthesis includes generic optimization, slack optimizations and power optimizations followed by placement and routing. The implementation includes partition, place and route. The output of the design implementation phase is a bitstream file.
- Design verification: the design is loaded into a simulator in order to check whether the behavior of the design is correct. Then, the design is loaded onto the target FPGA and tested in real environment.

More in detail, the FPGA vendor tools are constituted by the following stages:

- Synthesis: generates files describing the design terms of generic digital logic elements. After synthesis stage the XST tool writes the generic netlist in an *ngc* file.
- Translation: it takes synthesized modules (ngc) together with physical macros and User Constrain Files (ucf), and generates a single Native Generic Database (ngd) file. This operation is performed by the Xilinx NGCBuild utility.

- Technology mapping: it transforms a technology-independent netlist into one that only contains logic cells supported by the targeted FPGA architecture. Xilinx map utility reads the ngc file and maps the contained netlist, generating a Native Circuit Description (ncd) file.
- Packing: it is the process of grouping several LUTs and registers into one logic block.
- Placement and routing: it determines all the positions of the logic blocks onto the silicon of the specific FPGA and constructs paths that connect elements among them.
- Bitstream generation: the fully routed design is converted into a structure of bits (bitstream) and downloaded to the FPGA.

2.3.2 Bitstream structure

Unlike other technologies, which implement hardware directly into silicon, any errors in the final FPGA-based product can be easily corrected by simply reprogramming the device. A bitstream is a chain of zeros and ones responsible for changing the connectivity and functionality of the logic of the circuitry; that is, configuring the device. The contents of the logic block are programmed to control its functionality, while the routing switches are programmed to perform the desired connections. This sequence of data might affect to the whole FPGA (like in a static reconfiguration), or/and just to a region of it (dynamic reconfiguration).

The very first Xilinx FPGAs were completely fine-grained devices; but the complexity of these elements has been growing day by day due to the combination of fine-grain blocks units together with medium or coarse-grain blocks, such as DSPs or Microprocessors. However, independently of the complexity of these resources, the configuration process of all of them follows the same scheme [UG191]. Thus, all these elements are divided into smaller configuration units. This policy allows increasing the configuration flexibility of these resources, but at the same time the reconfiguration process becomes more complex and slow-

er. These small configuration units are known as *configuration frames*. From the Virtex-4 family ahead, a frame is as height as a clock region, and as wide as a configuration column (Figure 2.4). The number of columns of frames depends on the resources distribution within the FPGA. In addition, these frames are responsible for configuring the CLBs, BRAMs, DSPs and IOBs, but they also are in charge of configuring the central row of a clock region, in which the main clock is located (GCLK).

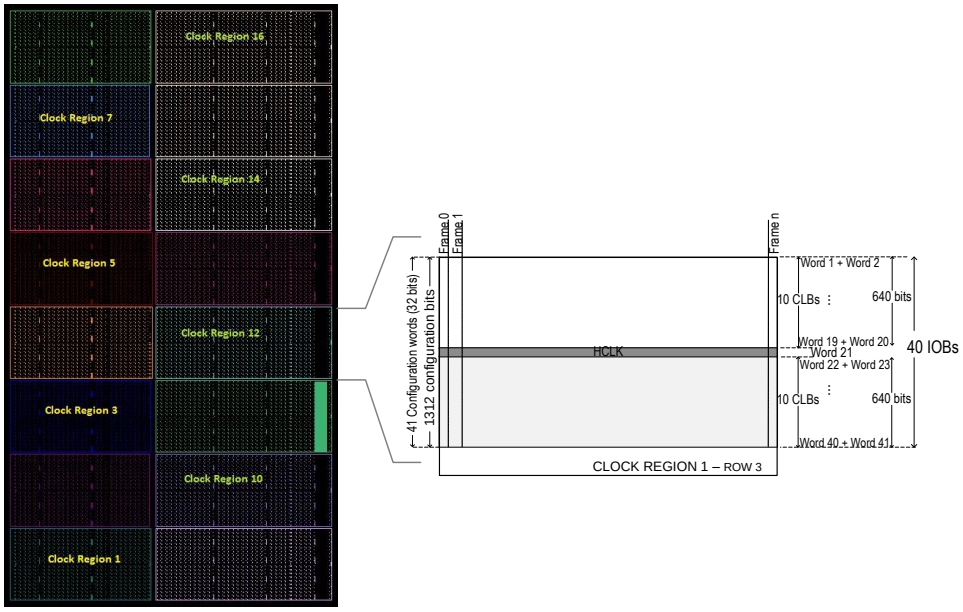


Figure 2.4 Virtex-5 LX110T layout separated in clock regions

The format of the bitstream follows a pattern structured in three sections: header, body and tail. The header collects synchronization information, and configuration commands. The body is the core of the bitstream, since it contains the configuration information of the resources available in the FPGA (configuration frames). Then, the tail of the bitstream stores the CRC (Correction eRror Code) of the whole bitstream.

In order to specify the correspondence between a configuration frame, and a physical location in the silicon, the frames' address is stored in a series of in-

ternal resources specifically focused on the reconfigurability. Within all these registers, the most relevant ones are:

FAR (Frame Address Register): this register stores the physical address within the FPGA where the configuration frame should be loaded.

FDRI (Frame Data Register Input): It is a writing register, which configures the addressing data of the FAR register.

FDRO (Frame Data Register Output): this is an only read register that allows getting the configuration frames information (those ones that are already loaded in the FPGA), starting from the FAR address.

2.3.2.1 Configuration frame addressing

The number of configuration bits necessary for programming an FPGA depends on the family and model of the target FPGA. All the Xilinx Virtex-5 FPGAs define a configuration frame with 41 words of 32-bits length each, which is translated into 1321 configuration bits per frame. More in detail, the configuration address, specified in the aforementioned FAR register is separated into 6 sections, everyone with different and specific information. The less 21 significant bits [0:20] are reserved for determining the exact position of the configuration frame within the surface of the FPGA. The following three bits [21:23] identify what kind of resource it is going to be configured. Xilinx identify three different blocks: connectivity and configuration blocks, BRAM content, and special interconnection blocks. This process is detailed in Figure 2.5.

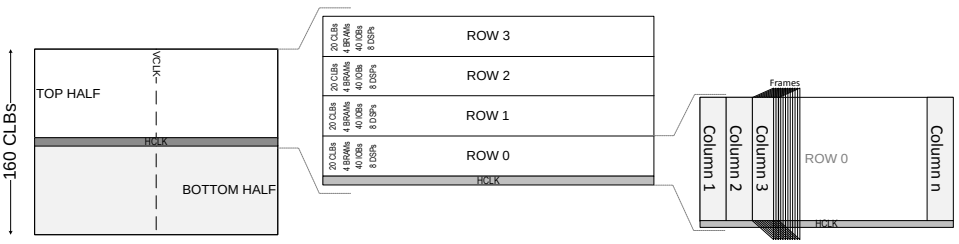


Figure 2.5 Columns distribution of every section of the FPGA, and the structure of every column in frames

2.3.3 Reconfiguration interfaces

An important aspect related to reconfigurable embedded systems is the method they use for loading partial bitstreams on to the device when a new configuration is requested. Actually, Xilinx provides two methods for reconfiguring a device; one of them is external and the other one is internal [XAP138]. The former group offers three different alternatives for loading a bitstream: one is using the Serial configuration port, other through the popular JTAG, and the last one is the SelectMap port. On the other hand, within the internal methodologies there is just one choice known as ICAP port (Internal Configuration Access Port). This port has to be combined with an embedded microcontroller or a state machine in order to control its behavior. However, within all these choices only the last one, the ICAP port, suits the necessities of autonomous embedded systems, since it is the one that permits exploiting the advantages of dynamic and partial reconfiguration [SBB+06] [BHH+07]. Despite the fact that Xilinx provides an IP for using the ICAP port, several research works propose improved versions of it in which the performance and/or the operation frequency are enhanced, such as [SDK09], [CMN+09], and [OMP+10] works. All these works have in common the fact that they wrap the ICAP IP, and then manipulate and manage the configuration bitstreams and the internal registers.

The reconfiguration process using the parallel ICAP interface makes uses of a control driver (HWICAP), which is provided by Xilinx. This self-reconfiguration process requires a control unit that handles the accesses to the configuration memory. Thus, traditional strategies use a microprocessor for performing this role. Then, the communication between the microprocessor and the peripherals of the system is done using the OPB (On-chip Peripheral Bus), though the microprocessor itself uses the PLB (Processor Local Bus) for accessing to the internal memory or controlling the static modules of the running application. A general scheme of a complete system is depicted in Figure 2.6.

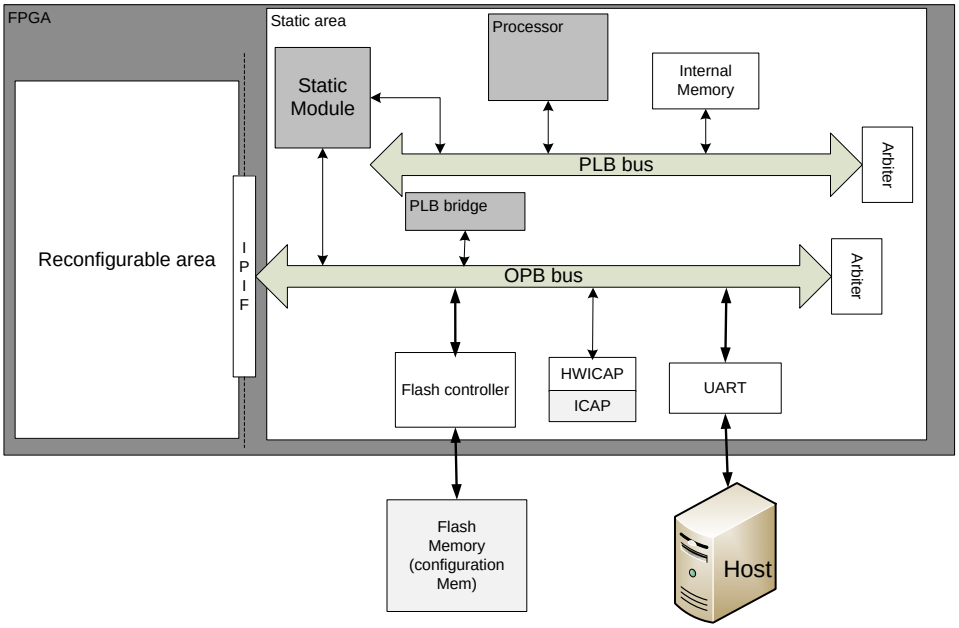


Figure 2.6 FPGA architecture

2.4 MANAGEMENT OF SCALABLE AND DYNAMICALLY RECONFIGURABLE HARDWARE

Both the dynamic/static reconfigurability operations and the data transfers need to be controlled. These tasks can be done statically or dynamically. The former is similar to the way in which operations from static code are scheduled and issued on a processor, whereas dynamically management is similar to the way in which out-of-order processors issue instructions when their operands become available. Any of these strategies might be developed for controlling the tasks previously mentioned.

The dynamic reconfigurability opens the window to the design of autonomous embedded systems, where the swapping between elements or their modification might be managed automatically, according to predetermined criteria established by the designer/developer.

This reconfigurability paradigm, in which several scalable designs are able to compete by hardware resources, is affordable from different perspectives:

- From a system point of view: at high level it is necessary to develop a set of mechanisms that facilitate a seamless execution of the reconfiguration procedure onto the FPGA, according to a set of environment variables or configuration parameters.
- From the design point of view: at middle-level it is deserved the development of scalable and reusable designs that allow the adaptation of the computational workload among different processing elements depending on the environmental requirements. This means, designing regular and homogeneous solutions in terms of their structure and behavior.
- From the implementation perspective: at low-level it is necessary to explore methodologies, mechanisms and tools able to alleviate current restrictions of the commercial tools focused on dynamic reconfigurability.

Dynamically scheduled reconfigurable architectures can deliver higher performance than statically scheduled ones for control-intensive code with unpredictable behavior.

2.5 CONCLUSIONS

The interest of the research community on reconfigurable architectures has considerably grown along this last decade. Nowadays, the vast design space makes difficult to find optimum reconfigurable architectures because this process involves satisfying many trade-offs in choosing values for each parameter. Some of the most remarkable advantages of the dynamic reconfiguration might be summarized as: power/size/cost reduction; hardware reusability; obsolescence avoidance and application portability. However, an intelligent system is needed to manage the reconfiguration process itself in order to save power and meet timing constrains in real time systems. Unfortunately, the exploration of partial reconfiguration for a design requires significant knowledge on the targeted device from the designers and developers side.

Chapter 3

Exploring scalability for video coding applications: H.264/AVC and SVC

This chapter describes the designing and development of a scalable architecture for the DF algorithm, which is part of the H.264/AVC and the SVC standards. This proposal differs from the common DF solutions of the state-of-the-art into two aspects. First, the modularity of the design has been conceived to take advantage of the scalability. Second, this hardware solution follows a novel data parallelization technique. Both features allow to adjust the performance of the DF according to the system requirements.

3.1 OUTLINE

Deblocking filter is one of the most complex functional cores of the H.264/AVC and SVC codecs. Its computational cost is heavily dependent on the video profile and the selected scalability levels. Moreover, its performance is typically constrained by data dependences. For this reason, developers have been focused on designing faster architectures by taking advantage of hardware possibilities and parallelization techniques. This PhD thesis proposes a novel scalable deblocking filter architecture which can be easily adapted to different video configurations, thanks to its modular and regular structure. The scalability property avoids redesigning the whole architecture, in case the environment or the configuration settings change. Therefore, design productivity is increased, but also savings in terms of area and power are achieved by using only the logical resources needed by each application. Furthermore, this approach relies on a novel macroblock level parallelization strategy which reduces the amount of clock cycles needed to filter a video frame, against traditional strategies. The proposed architecture is completely flexible, since the parallelism might be adapted according to the application requirements. Implementation results in an FPGA Virtex-5 demonstrate the performance benefits of this flexible solution compared to some rigid state-of-the-art deblocking filter approaches.

3.2 EXPOSING THE PROBLEM

Nowadays, the H.264/AVC video coding standard is one of the most widely spread codecs for multimedia applications. This fact is due to its overwhelming features compared to its predecessors, such as a better rate-distortion performance. Unfortunately, the strengths of this codec come at the price of increasing the complexity of the operations, as well as the computation. According to [WDG+10], [SCL+11], [HJK+03] and [SMW07], one of the most time consuming tasks in H.264/AVC and SVC codec standards is the deblocking filter (DF) process. Its main task is to reduce blocking artifacts, appearing in this kind of video standards as a consequence of the data processing performed in previous functional blocks of the system. The DF achieves a visual quality improvement of the

reconstructed image by smoothing the borders between objects within an image.

Many DF state-of-the-art proposals have been designed with the aim of reducing the number of memory accesses and/or accelerating the execution by applying new techniques and filtering patterns. Independently whether the solutions are implemented in hardware or software, they are characterized by their lack of parallelism at data-level. The underlying fact, that it is the responsible of this inefficiency; it is the way in which these hardware approaches tend to process the data, commonly known as raster-scan pattern where the data units are read and filtered in ascendant order and one by one. Despite the fact that this technique reduces the control complexity, at the same time it restricts the acceleration possibilities.

The idea of taking advantage of the macroblock-level parallelism has attracted a considerable interest within the research community because it allows parallelizing time-consuming tasks onto different processors. Nonetheless, the development of effective designs capable of exploiting the benefits of using parallel processors is not trivial. In this sense, there have been several software proposals motivated due to the fact that these kinds of solutions requires less design time than hardware designs. These solutions are mainly focused on multiprocessor structures, where different processors are dedicated to execute different tasks, or even to run the same tasks but processing different data. In any of these cases, the level of parallelism is increased by means of modelling the processing pattern to a wavefront pattern, instead of a raster scan one. Some examples of these structures have been presented in [TJG09], [AJM+09] and [WKK10], in which a full H.264/AVC decoder has been implemented in a multiprocessor architecture. However, hardware-based architectures have associated some characteristics such as high performance, high operating frequency and lower energy consumption in comparison to the software ones, which favour the development of parallel hardware designs. On the one hand, there exist many hardware DF solutions focused on improving specific aspects of the DF algorithm execution. Sometimes, authors work on reducing the num-

ber of memory accesses (e.g., [WL06], [LLL07], [LCC10] and [CJC+12]), increasing the clock frequency (e.g., [JLY+10] and [KT10]), reducing or simplifying the number of operations (e.g., [LYC+10] and [KGT+12]), or processing several data simultaneously (e.g., [VJG09], [KCC10] and [ZYD+12]). In most of the cases, these approaches are rigid, since they have been designed for performing in the worst case of well-known and invariable conditions. A direct consequence of these kinds of rigid solutions is the impossibility of getting high efficiency rates when the environmental settings are relaxed. Thus, the design of parallel hardware-based architectures is being very popular, and they have been implemented on FPGAs or customized platforms (ASICs or SoCs), being [YDZ11] and [SZC+11] a proof of this.

This work introduces a novel macroblock (MB) level parallelization of the DF algorithm which exploits the benefits of a wavefront pattern. This technique has been mapped on a coarse grain hardware architecture based on a modular two dimensional structure, in which all modules are connected with its immediate neighbor. As a consequence, several modules might work simultaneously, like in a multiprocessor system, but they overcome the lack of efficiency from the general purpose microprocessor cores and the power inefficiency of GPUs. Furthermore, data transactions between elements can profit of the regularity, modularity and local communications of this kind of structures. As a result, the communication schemes, the distributed memory accesses, and the synchronization and control tasks are simplified compared to other parallel solutions. The scalability characteristic of the proposed DF architecture permits to vary the number of hardware resources by adding or removing elements from the FPGA easily. Moreover, an appropriate data delivery policy and a data distribution strategy play a relevant role in the overall solution, since they are responsible of adapting the workload to a specific configuration. In this sense, they must be adaptable but also general enough to support different levels of scalability without negatively impact on the final performance.

In order to exploit the benefits of the suggested wavefront strategy, the proposed DF solution has been developed as a modular design, with a regular

structure and behavior. This solution makes easy to distribute the workload into several units, minimizing the control and synchronization tasks as will be explained ahead. This architecture is able to exploit benefits of the proposed enhanced wavefront strategy. Therefore, the architectural requirements are closely related to the parallelization analysis offered above. For the sake of clarity, these are summarized here:

- MB-level parallelism by following an improved wavefront pattern proposed.
- Reduced number of accesses to external memory.

This solution includes one important feature and one requirement, both closely related to the proposed MB-level parallelization method and the proposed modular design. The architecture is flexible enough for modifying the level of parallelization freely, according to the environmental demands. The important feature is the scalability, being the solution adaptable to different video formats and scenarios. However it requires that both, the upper and the left neighbors, have to be ready before processing each MB to fulfill data dependences.

Since the proposed architecture filters several MBs in parallel; it can be considered a coarse grained solution. In addition, for a low complexity solution, achieving scalability imposes regularity and modularity as characteristics. In fact, the objective is to adapt the performance of the architecture just by changing the number of homogeneous basic modules working in parallel. Consequently, the proposed DF may be arranged as a two dimensional array of processing elements with a mesh topology. Furthermore, instead of relying on a centralized control module, which should be completely redesigned for each possible size of the array, control logic has been distributed among modules.

3.3 THE DEBLOCKING FILTER ALGORITHM

Video coding standards typically lose information during the encoding process, since it is the best way to reach high compression rates. Due to this fact, and also because of the distortion added by other functional blocks during the

decoding process itself, the appearance of the reconstructed image tends to be rough. This effect is known as blocking effect, since the edges between objects in the image look like blocks. Regarding to this, the target of the DF is to smooth the image by reducing blocking distortion, which improves the visual appearance of the decoded pictures. More in detail, this section is focused on the DF algorithm that complains with the H.264/AVC standard. Its main concepts are also valid for the latest standards such as the SVC and the HEVC.

The decoding procedure of one image brings a huge amount of information that need to be processed. With the idea of simplifying the operations as much as possible to the codec, the image is split into smaller portions. These partitions are subdivided in several levels, with lower amount of information. In this sense, into a H.264/AVC compression system an image is divided into slices, then into macroblocks (MBs), blocks and finally into line of pixels (LOP). A MB is composed by a matrix of 16×16 pixels of luminance, and two smaller matrices of chrominance (red and blue) of 8×8 pixels each one. These matrixes are arranged in groups of 4×4 pixels, named blocks and numbered from zero to 23 (16 blocks correspond to the luminance information, and 4 more for each chrominance). Finally, each strip of 4 pixels in a block is named line of pixels, where each pixel is numbered from zero to three ($pixel_0$ – $pixel_3$) according to the order in which it is processed.

3.3.1 Deblocking filter constrained behavior

The DF is a MB-based filter algorithm, since it takes a MB as a reference to arrange data. Then, the edges of all MBs contained in an image are processed, except all the external borders of it. In this context, a left edge limits the beginning of every block in a MB. As it is shown in Figure 3.1, in a MB we can distinguish eight vertical (V_0 – V_7) and eight horizontal edges (H_0 – H_7), four of them correspond to the luminance information, and the other four belong to the chrominances.

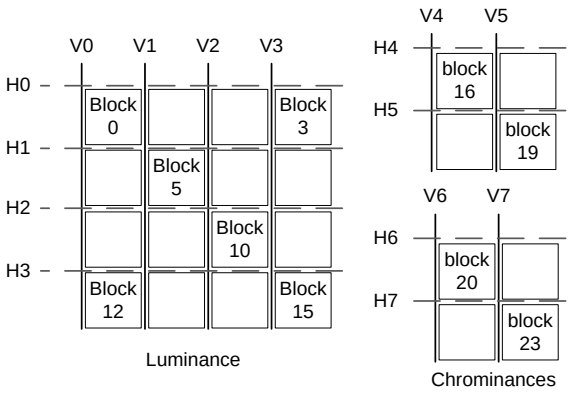


Figure 3.1 Vertical and horizontal edges of a MB

The H.264/AVC video coding standard obligates to process the vertical edges of a MB before the horizontal ones, with the condition of respecting the data dependences all the time. The particularity of these data dependences are explained further in this section and in detail in section 3.4. However, the standard does not determine in which order these edges, and consequently the involved blocks, should be filtered. This freedom permits generating filtering orders, in which the blocks of a MB are processed in a different order form the standard pattern. Because of this fact, in this work we have selected the pattern shown in Figure 3.2, which implies that every MB will be sequentially filtered, starting processing blocks horizontally from left to right, row by row (vertical edges filtering), and then vertically from top to bottom, column by column (horizontal edges filtering). This pattern simplifies the control of the filtering unit itself.

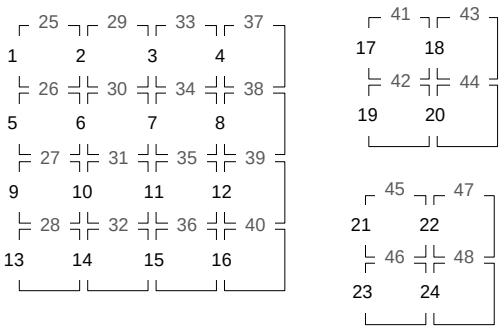


Figure 3.2 Proposed filtering order within a MB

Independently of the filtering pattern selected, the standard imposes strong dependences between the current data unit to be filtered and its adjacent left and top neighbors. In order to facilitate the identification of the current data and the neighboring data, it is generally accepted referring to them as q_n and p_n respectively. Therefore, as Figure 3.3 shows, blocks located on the edges V_0, V_4 and V_6 (see Figure 3.1), need their left neighbor in order to be filtered. The same happens with blocks located on the edges H_0, H_4 and H_6 (see Figure 3.1), i.e., they need their upward blocks.

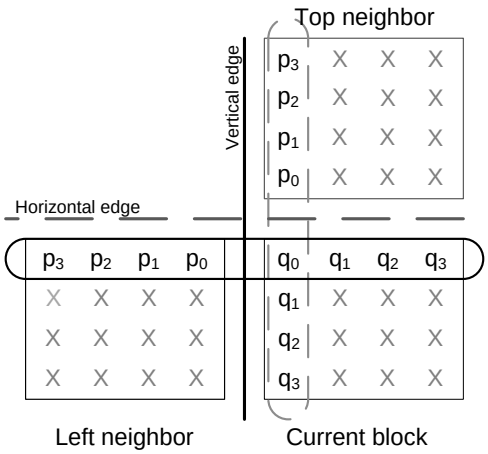


Figure 3.3 Pixels notation during the horizontal and vertical filtering

3.3.2 Filtering process: Boundary Strength and filter units

Regarding to the filtering process itself, the DF is a highly adaptive algorithm since its filtering operations vary according to several parameters. The adjustment is performed by two elements into two stages. The first one is the Boundary Strength unit (BS), and the next one is the Filter unit. The former calculates the filtering strength, which determines the number of pixels that should be processed by the latter.

The BS always returns a value of the strength between zero and four. Zero means that data goes through the filter unaltered, and four means the strongest filtering. This value is calculated based on the configuration characteristics of

the input blocks (the current one and its neighbors), as it is depicted in Figure 3.4.

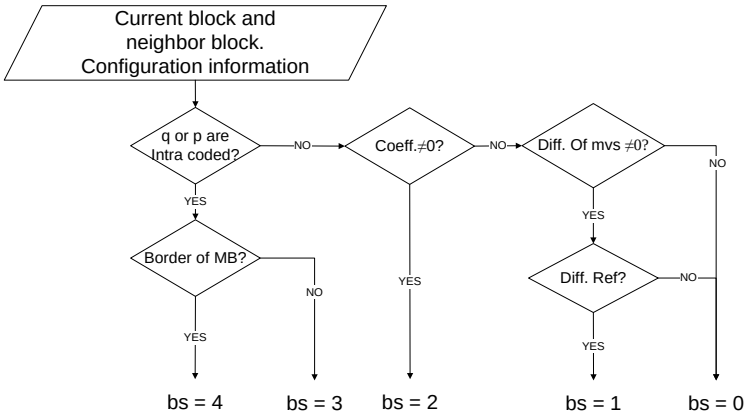


Figure 3.4 Boundary Strength's calculation

The Filter unit receives as inputs the BS value, the input blocks (the current one, denoted by q_n, and its neighbor, represented by p_n) and three parameters used as thresholds (α, β and cl). The two first parameters (α, β*)* are used as thresholds in the standard, and they mainly rely on the quantization parameter (QP). The later parameter, *cl,* is used for avoiding the undesired blurring effect, and its values depend on the QP, but also from the strength value. Before the input blocks are processed, the filter evaluates the following expressions:

$$|q_0 - p_0| < \alpha$$
$$|p_1 - p_0| < \beta$$
$$|q_1 - q_0| < \beta$$

Whether the inputs do not fulfil with the expressions above, the input blocks will pass through the filter, without considering the BS value. In any other case, the filter performs logic operations with the pixels, according to the BS value. The operations are detailed in Table 3.I.

Table 3.I Filtering operations according to the boundary strength values

Boundary Strength values	Filtering operations
$\boldsymbol{BS = 0}$	$q_0' = q_0$; $q_1' = q_1$; $q_2' = q_2$; $q_3' = q_3$ $p_0' = p_0$; $p_1' = p_1$; $p_2' = p_2$; $p_3' = p_3$
$\boldsymbol{1 \leq BS \leq 3}$	$q_0' = q_0 - \Delta_0$; $\Delta_0 = \min(MAX(-c_0, \Delta_{0i}), cl)$ $p_0' = p_0 + \Delta_0$; $\Delta_{0i} = [4+4\cdot(q_0 - p_0)+(p_1 - q_1)] >> 3$
	If $\lvert q_2 - q_0 \rvert < \beta$ then: $q_1' = q_1 + \Delta_{q1}$; $\Delta_{q1} = \min(MAX(-cl, \Delta_{q1i}), cl)$ $\Delta_{q1i} = (q_2 - 2\cdot q_1 + ((p_0+q_0+1) >> 1)) >> 1$
	If $\lvert p_2 - p_0 \rvert < \beta$ then: $p_1' = p_1 + \Delta_{p1}$; $\Delta_{p1} = \min(MAX(-cl, \Delta_{p1i}), cl)$ $\Delta_{p1i} = (p_2 - 2\cdot p_1 + ((p_0 + q_0 + 1) >> 1)) >> 1$
$\boldsymbol{BS = 4}$	If $((\lvert q_2 - q_0 \rvert < \beta)$ & $(\lvert p_0 - q_0 \rvert < (\alpha >> 2) + 2)$ & *luminance*) then: $q_0' = (q_2 + 2\cdot q_1 + 2\cdot p_0 + p_1 + 4) >> 3$ $q_1' = (q_2 + q_1 + q_0 + p_0 + 2) >> 2$ $q_2' = (2\cdot q_3 + 3\cdot q_2 + q_1 + q_0 + p_0 + 4) >> 3$
	If $((\lvert p_2 - p_0 \rvert < \beta)$ & $(\lvert p_0 - q_0 \rvert < (\alpha >> 2) + 2)$ & *luminance*) then: $p_0' = (p_2 + 2\cdot p_1 + 2\cdot q_0 + q_1 + 4) >> 3$ $p_1' = (p_2 + p_1 + p_0 + q_0 + 2) >> 2$ $p_2' = (2\cdot p_3 + 3\cdot p_2 + p_1 + p_0 + q_0 + 4) >> 3$
	If $((\lvert q_2 - q_0 \rvert > \beta)$ & $(\lvert p_0 - q_0 \rvert > (\alpha >> 2) + 2)$ & *luminance*) then: $q_0' = (2\cdot q_1 + q_0 + p_1 + 2) >> 2$
	If $((\lvert p_2 - p_0 \rvert > \beta)$ & $(\lvert p_0 - q_0 \rvert > (\alpha >> 2) + 2)$or *chrominance*) then: $p_0' = (2\cdot p_1 + p_0 + q_1 + 2) >> 2$

3.3.3 A detailed analysis of the deblocking filter data dependences

Before starting to analyze different parallelization techniques, it is crucial to understand the DF data dependences. In fact, this knowledge is the key to maximize the efficiency of any strategy. Even more, the importance of this issue lies on the fact that it constraints the level of parallelism that can be achieved.

According to the DF algorithm, and considering different data granularity (LOP, block or MB) the data dependences might be analyzed from a bottom-up or a top-down perspective indistinctly. Following the former sequence, Figure 3.3 clearly depicts the dependences between the current LOP ($q_0 – q_3$) and its neighbors ($p_0 – p_3$). The horizontal filtering acts over vertical edges, and the vertical one over horizontal edges. However, before starting the vertical filtering of the current LOP, it is mandatory to complete processing of all the horizontal current LOPs, since these operations transform the values of every pixel. Moving to an upper level, the set of the current LOPs conforms a block. At block-level the data dependences remains the same. These situations are perfectly depicted in Figure 3.5.a and Figure 3.5.b. That is, the current block needs information from its left block neighbor for performing the horizontal filtering, and from its top neighbor in order to complete the vertical filtering. At the same time, the whole MB is formed by blocks, so the external borders of the current MB need information from its neighbor MBs, as shown in Figure 3.5.c.

Whether an image is divided into its MBs, the dependences among MBs follow the same pattern as the one shown in Figure 3.6. For instance, MB_7 needs information from its left neighbor (MB_6) in order to be filtered horizontally, and information from its top neighbor (MB_1) to complete its vertical filtering. Even more, before MB_7 receives information from MB_1, the upper right (MB_2) has to be filtered, at least horizontally since this action modifies the right column of blocks of MB_1.

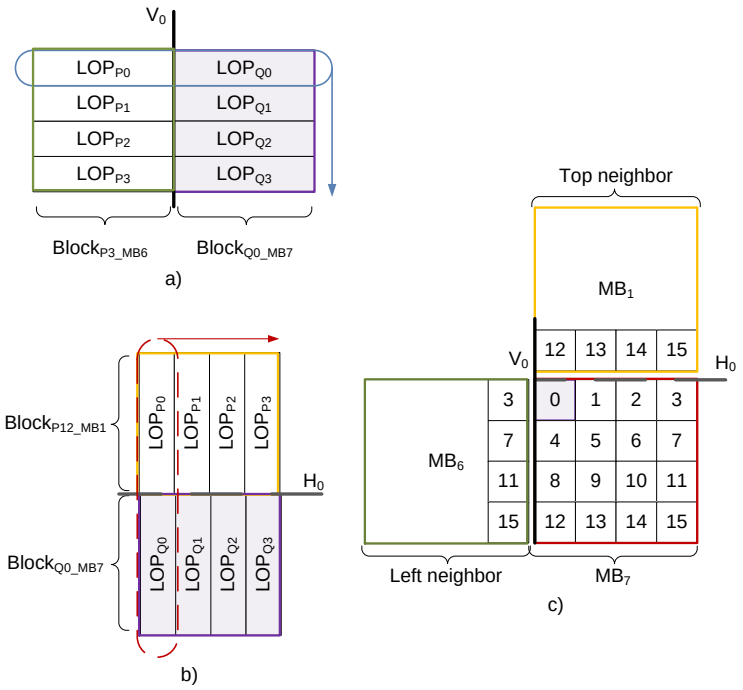


Figure 3.5 Bottom-up dependences at data-level; a) Horizontal filtering at LOP and block-level; b) Vertical filtering al LOP and block-level; c)Data dependences at block and MB-level

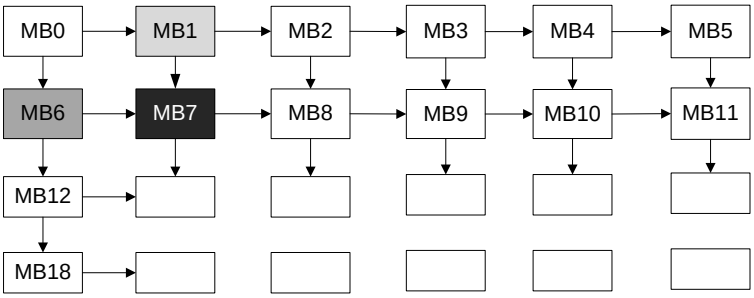


Figure 3.6 Deblocking filter data dependences at MB-level

These data dependences are completely independent from the architecture. That means that any software or hardware DF implementation must respect them to be H.264/AVC compliant.

3.4 PROPOSED MB-LEVEL PARALLELIZATION STRATEGY

The DF might be parallelized at different levels. From the data unit point of view, it is possible to distinguish a LOP-level, a block-level or a MB-level parallelization strategies. However, based on several fundamentals, this work has been focused on exploiting the strengths of processing at MB-level. First of all, according to the overall DF behavior, but also considering the H.264/AVC structure, the DF is a MB-based filter algorithm. Secondly, working at block or at LOP level the level of parallelization is constrained due to the data dependences within the MB. However, in the case that the parallelization is exploited at MB-level, there exists a higher degree of freedom, since there is no limit to the number of MBs that might be processed in parallel as long as the data dependences among MBs are respected. This is possible by using different processing patterns beyond the traditional and sequential raster-scan.

Into the DF state-of-the-art there are significant and recent works that proposes different MB-level parallelization strategies, such as [WYC09], [PHC+11] and [SBK+09]. All of them allow processing several MBs simultaneously by issuing the workload onto several processing elements (PEs). The difference among these proposals lies on how the data are distributed among the available PEs. Thus, the limited error propagation, presented in [WYC09], divides each image frame into as many rectangles as PEs are there in the system. Then, every PE filters all the MBs of a rectangle by following a raster-scan order. Finally, the borders between the rectangles need to be filtered before considering that the frame is completed.

Another MB-level solution has been presented in [PHC+11]. This proposal is efficient just in case the boundary strength values move between one and three (normal filtering), within a MB. That criterion permits exploiting two levels of parallelism. At lower level, several pixels in a MB might be filtered simultaneously, due to the specific relationships among them during the filtering operations. At higher level (MB-level) several MBs might be processed simultaneously following the explained procedure.

On the other hand, [YDZ11] and [SBK+09] make use of a wavefront parallelization strategy, widely exploited on multiprocessor systems. This processing pattern allows to use several PEs, although they do not start processing at the same time in order to respect the data dependences. A PE has to remain idle until its previous neighbor completes filtering two MBs. After that, the PE begins processing.

Finally, [SZC+11] proposes a MB-level parallelization that permits using multiple PEs. Every PE is responsible for filtering a full column of MBs, as part of an image. When the vertical filtering of the image is completed, the same pattern is followed but in the horizontal way. Unfortunately, this technique is not compatible with the H.264/AVC standard.

Going deeper into the data dependences, the filtering direction fixes the relationship between MBs and PEs, as Figure 3.7 depicts. The grey arrows represent the internal MB relationship into each PE (the vertical filtering must start after the horizontal one finishes). The black arrows show the inter-dependences between two consecutive MBs located in the same row of the image. Here, the upper vertical neighbor is not required until the vertical filtering operations start processing the current MB. Then, the dotted vertical arrows highlight the data dependences between two consecutive rows. In order to establish a generic measurement for evaluating the DF execution, the MB_{cycle} parameter has been introduced in this PhD thesis in order to represent the time required to process a whole MB by a PE.

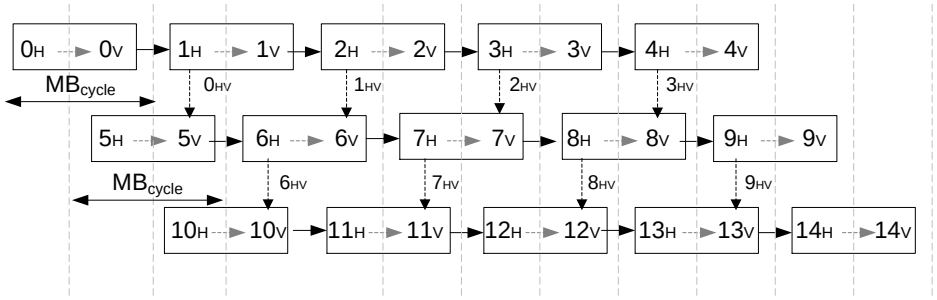


Figure 3.7 MB dependences according to the direction of the filtering execution

From these information is easy to conclude that the freedom to parallelize the DF algorithm is always limited by data dependences. Then, independently of the execution strategy followed by DF implementation, in terms of the number of PEs involved into the filtering process (sequential whether one PE is performing, and parallel whether more than one PE is enabled), the final solution must fulfil with the aforementioned restrictions.

However, in parallel systems all the controlling tasks are complex, mainly due to the amount of independent elements that are running simultaneously. This situation is represented in Figure 3.7, where not all the PEs remain in the same state by attending to the filtering execution direction (even rows are synchronized among them, and the same happens between the odd rows). With the objective of simplifying the whole synchronization, the easiest decision is to move every row half a MB_{cycle} to the right. This new execution pattern obligates to all the PEs to be at the same state all the time. The resultant MB-level parallelization scheme is represented in Figure 3.8, which facilitates control tasks and the exploitation of the parallelization and the modularity of the DF. However, in order to guarantee the data dependences, it has been necessary to take some decisions that directly impact on the DF design. The most important one relies on separating horizontal and vertical filtering stages by following a sequential MB filtering. Therefore, this strategy allows starting filtering MB_7 vertically after MB_1 has been horizontally filtered.

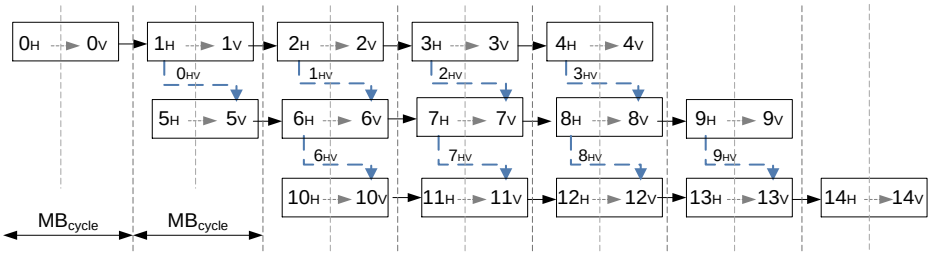


Figure 3.8 Proposed wavefront pattern in which the filtering execution among rows is synchronized

As a result, this MB-level parallelization saves MB_{cycles} with respect to traditional wavefront strategies since allows a higher parallelism by reducing the

delay between two consecutive rows, as Figure 3.9 shows. That means that consecutive PEs might start processing after one MB_{cycle}, instead of waiting two MB_{cycles}.

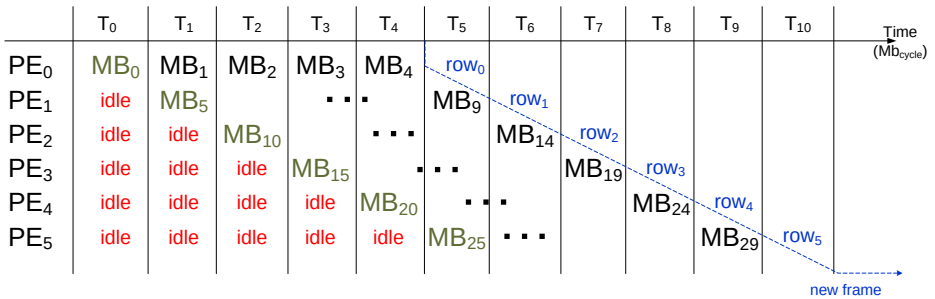


Figure 3.9 Proposed wavefront pattern filtering a SQCIF (8×6MBs) frame with six PEs

The figure above considers that the number of PEs were equal to the height of the image in MBs. Nevertheless, this assumption is rarely true and it is a very ideal case of use. More realistic scenarios limit the number of PEs to a few of them, such that the number of PEs is lower than the number of MBs in an image column. This consideration varies the way that the rows of MBs are arranged among the PEs in order to be processed, by dividing the image into horizontal stripes, as height as the number of PEs are available. Then into each stripe, every row is assigned to one specific PE. Despite the fact that Figure 3.10 divides the image according to the proposed wavefront pattern, the same happens with the traditional wavefront, though considering the time slot of every PE.

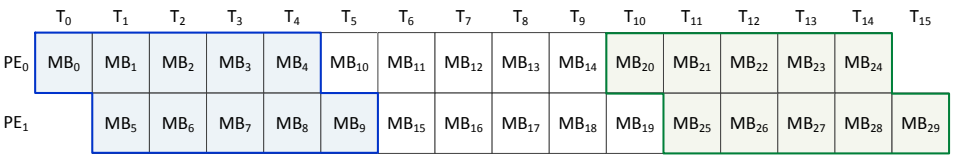


Figure 3.10 Defragmentation of a full image according to the number of PEs

Observing Figure 3.10 we can extract two ideas. First of all, the number of MB_{cycles} has been increased compared with the ideal case in which the number of PEs is higher or equal to the height of an image in MBs, but it is still lower than using a raster-scan. The second one is related to the proportional relation-

ship between the height of the image and the number of PEs. In case they are multiples, the utilization of the PEs is very efficient; otherwise the number of MB_{cycles} necessary for processing the whole image increases, since the system has to wait until the last enabled PE finishes its processing whereas the rest of PEs remain idle.

3.4.1 MB-level parallelization patterns comparison

This subsection establishes a comparison between the raster-scan, the traditional wavefront [Kun88] [Sir13] [AMS+02], the error propagation strategy [PHC+11] and the proposed wavefront patterns. The comparison among these strategies highlights the gain of using one method over the other, expressing the results in MB_{cycles}. More in detail, the equations collected in Table 3.II express how to calculate the total number of MB_{cycles} in all the possible situations, with each of these patterns.

In order to simplify the nomenclature of the expressions, the size of the image has been represented as frameWidthMB×frameHeightMB, where frameWidthMB corresponds to the width of the image in MBs, and frameHeightMB refers to the height. The floor function maps the largest previous integer of its bracket expression. The mod function is the residual value after the division between frameHeightMB and nPE.

According to this metric, any wavefront methodology (the traditional one and the proposed one) is always faster than the raster-scan pattern; and the proposed wavefront is better than the traditional one, such that equation (2) shows:

$$\boldsymbol{Proposal-wavefront} = \begin{cases} (\boldsymbol{nPE}-\mathbf{1});\ If\ frameheightMB\ mod\ nPE\ = 0 \\ ((\boldsymbol{frameHeightMB\ mod\ nPE})-\mathbf{1});\ Otherwise \end{cases} \quad (2)$$

Attending to (2), both wavefront methodologies (the traditional and the proposed one) spend the same number of MB_{cycles} for processing an image when the result from the *mod* function is equal to one. Regarding the improved wavefront pattern and the error propagation methodology, the proposed wavefront gains $[\boldsymbol{frameWidthMB}-(\boldsymbol{nPE}-\mathbf{1})]$ MB_{cycles}.

Table 3.II Number of MB_{cycles} for filtering a whole frame by using different techniques

Technique	Case of use		Formula (Total number of MB_{cycles})
Raster-scan	Always		frameWidthMB×frameHeightMB
Traditional wavefront	nPE = frameHeightMB		$\text{frameWidthMB} + [2 \times (\text{nPE} - 1)]$
	nPE<frameHeightMB and (nPE>1)	frameHeight mod nPE=0	$\text{frameWidthMB} \times \text{floor}\left(\frac{\text{frameHeightMB}}{\text{nPE}}\right) + [2 \times (\text{nPE} - 1)]$
		Otherwise	$\text{frameWidthMB} \times \text{floor}\left(\frac{\text{frameHeightMB}}{\text{nPE}}\right) + \text{frameWidthMB} + \left[2 \times \left((\text{frameHeightMB} \bmod \text{nPE}) - 1\right)\right]$
Error prop-agation	nPE=frameHeightMB		$2 \times \text{frameWidthMBs}$
	nPE<frameHeightMB and (nPE>1)	frameHeight mod nPE=0	$\text{frameWidthMB} \times \text{floor}\left(\frac{\text{frameHeightMB}}{\text{nPE}}\right) + \text{frameWidthMB}$
		Otherwise	N/A
Proposed wavefront	nPE=frameHeightMB		$\text{frameWidthMB} + (\text{nPE} - 1)$
	nPE<frameHeightMB and (nPE>1)	frameHeight mod nPE=0	$\text{frameWidthMB} \times \text{floor}\left(\frac{\text{frameHeightMB}}{\text{nPE}}\right) + (\text{nPE} - 1)$
		Otherwise	$\text{frameWidthMB} \times \text{floor}\left(\frac{\text{frameHeightMB}}{\text{nPE}}\right) + \text{frameWidthMB} + [(\text{frameHeightMB} \bmod \text{nPE}) - 1]$

Table 3.III shows a numerical comparison among these methodologies, by processing different images with a different number of PEs. The results demonstrate the direct relationship between the size of the image and the number of PEs. In case of using five PEs, and being processing a 4CIF image both wavefront patterns, the traditional one and the proposed one, requires the same number of MB_{cycles} to complete the image. In case of the error propagation strategy, these values are not available since the documentation does not explain how the workload is rearranged. However, the improvement of using the proposed wavefront is higher when several images are processed one after another, just

in case the system does not restrict what PE starts filtering the MB_0 of any image. This fact allowed accumulating the gain in a factor of [*number_of_images* × *((frameHeightMB mod nPE) -1)*]. Consequently, it is possible to determine the best number of PEs for each input format.

Table 3.III MB-level parallelization comparison for filtering a full image in terms of MB_{cycles}

Parallelization strategy	Equations for calculating the total number of MB_{cycles} to process one image	Image (frameWidthMB × frameHeightMB)		
		QCIF (11×9)	CIF (22×18)	4CIF (44×36)
Raster-scan	Only one nPE all the time	99	396	1584
Traditional wavefront	If nPE = frameHeightMB	27	56	114
	if (frameHeightMB mod nPE) =0 & (nPE = 3)	37	136	532
	if (frameHeightMB mod nPE) ≠ 0 & (nPE = 5)	28	92	352
Error propagation	If nPE = frameHeightMB	18	36	72
	if (frameHeightMB mod nPE) =0 & (nPE = 3)	44	154	572
	if (frameHeightMB mod nPE) ≠ 0 & (nPE = 5)	N/A	N/A	N/A
Proposed wavefront	If nPE = frameHeightMB	19	39	79
	If (frameHeightMB mod nPE) = 0 &(nPE = 3)	35	134	530
	If (frameHeightMB mod nPE) ≠ 0 & (nPE =5)	25	90	352

These comparisons are referred to the processing tasks itself; i.e., just considering the PE executions. They neither consider the initial data loading nor the final data downloading. This criterion tries to isolate the MB-level parallelization strategies from the final architectural implementation of the DF.

3.5 PROPOSED SCALABLE DEBLOCKING FILTER ARCHITECTURE

This section describes all the actors involved into the proposed scalable architecture, in order to introduce then the behavior of the whole system, including also the scalability aspects.

3.5.1 Basic architectural description

In order to process a video sequence following one or other parallelization strategy, first of all the information must be read from memory. The proposed architecture interacts with the rest of the system through two data memories, shown in Figure 3.11. Therefore, the DF reads the unfiltered MBs, necessary for

removing the block artifacts, from an external memory (Shared Input Memory). This memory is shared with other functional blocks involved into the H.264/AVC encoding/decoding process. Once the MBs are fully filtered, the DF architecture writes them into another external memory (Reconstructed Output Memory), overwriting the information corresponding to the previous frame's MBs.

Taking a look into the proposed DF architecture, there are several modules in which all of them perform different and specific tasks: controlling, data distribution and processing. Such as Figure 3.12 shows, there exist five modules referred as IC, IM, PE, OM and OC that compose the whole design. These ones are deeply explained along this section.

The proposed DF has been characterized as coarse-grain architecture from a data-level perspective, since the data units consist on a bunch of pixels. However, this feature does not avoid providing a flexible and a modular design.

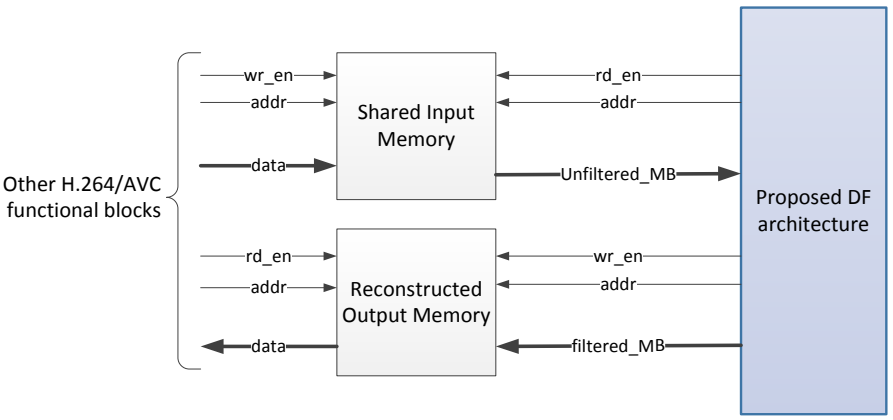


Figure 3.11 High-level schematic of the proposed DF architecture

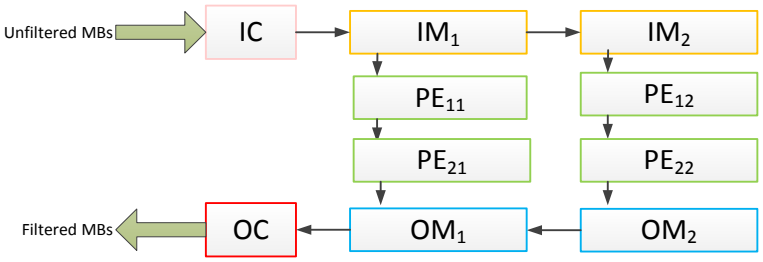


Figure 3.12 Coarse-grained and modular DF architecture

Whether the final design is going to implement the proposed wavefront strategy, in the sense that more than one MB might be filtered simultaneously; then, the PE in charge of the DF has to have a regular behavior and communications patterns all the time. Consequently, the PE always performs its tasks in the same way and it is always interconnected to its neighbors following the same structure. This implies that PEs must spend the same time for processing any kind of MB, independently of the MB type or the filtering strength applied. Moreover, the designed PE of the proposed DF architecture implements a sequential filtering approach, which obligates to perform horizontal filtering operations firstly, and after them, the vertical ones. In this particular case, the PE contains only one filter unit, such that the PE starts processing luminance components, and continues with the chrominances one after another. This restriction makes easier to respect the data dependences between MBs.

In turn, data dependences entail a specific allocation strategy for MBs. The position of a MB within a video frame determines the specific processing element where is going to be filtered. Moreover, the mapping policy is compatible with the architecture scalability, since data dependences are respected for any possible DF size. In order to share partially filtered MBs among processing elements, specific communication channels have been included throughout the entire array. Finally, in all the architectures designed with a mesh topology, the number of memory accesses is inherently reduced. Against the GPU-based solutions, processing elements exchange directly semifiltered data corresponding to shared neighbors, without requiring external memory accesses.

3.5.1.1 Modules description

Firstly, the Input and Output Controllers (IC and OC), responsible of data exchange with the rest of the system, are analyzed. Afterwards, elements belonging to the communication and the data arrangement structure are detailed; they are the Input Memories (IM) and the Output Memories (OM). Finally, the *PE,* responsible of carrying out filtering tasks, is described.

3.5.1.1.1 Input and Output Modules

These two modules act as an interface between the architecture itself, and other external elements within the video encoder/decoder.

3.5.1.1.1.1 Input Controller

The Input Controller (IC) reads from the External Memory the MBs to be processed by the DF. This task requires generating the reading sequence, by defining the order in which MBs are sent to the architecture, according to data dependences. Specific logic has been implemented inside the IC with this purpose. Basically, it consists of a set of counters, comparators, and arithmetic logic in charge of generating counting boundaries. Reading address for each MB also depends on the initial offset within the memory.

A suitable reading sequence depends upon the size of the architecture. Hence, the IC logic is able to adapt the generated sequence, just by changing some generic configuration parameters. In addition, this module introduces an *ID* number in every MB, acting as a header. This value facilitates the identification of every MB, in terms of its position into a video frame.

3.5.1.1.1.2 Output controller

Once MBs have been filtered, the Output Controller (OC) sends these MBs to an external output buffer. However, as a difference with respect the IC, this module does not have to generate any writing sequence to access the external memory. It reads the ID set in every MB, and keeps the order. Memory interfaces of both Input and Output Controllers are implemented using a bus-based approach. Actually, both modules might be joined into one unique module.

3.5.1.1.2 Data Transmission Modules

A communication structure has been designed in order to make feasible the parallel operation of several PEs at the same time. These elements feed to each PE with the appropriate MB that has to be processed and send it back to the external memory once it has been processed.

3.5.1.1.2.1 Input Memories

The Input Memory (IM) is a module in charge of transmitting unfiltered MBs from the Input Controller to each column of PEs. Therefore, the IMs never access to the external memory. Actually, the IC reads the unfiltered MBs from the external memory and sends those data to the DF architecture. Thus, the first IM receives the unfiltered MBs and follows a push-forwarding policy until all the IMs are full. That means the IM_1 does not start filling its data buffer until all its right neighboring IMs have filled their FIFOs. Once all the IMs' FIFOs are completed, the unfiltered MBs are transmitted to their corresponding columns of PEs placed below them. The mechanism responsible of deciding the MBs routing through the array is explained ahead.

The main component of each IM is a FIFO memory. This memory has to be dimensioned, so that it is able to store all of the MBs that need to be processed simultaneously by the column of PEs below. Therefore, despite the inherent scalability of the architecture, IMs have to be oversized to the maximum possible size of the array. Finally, this module also includes control logic to synchronize all the PEs.

3.5.1.1.2.2 Output Memories

Output Memories (OM) receive filtered MBs from each column of PEs, and transmits them to the Output Controller. Thus, these modules also include FIFO memories, but in this case they are to store processed MBs that are subsequently transmitted to its immediate left OM, or to the OC. Unfortunately, all the memories inside this module also suffer from the over dimensioning problem explained above.

3.5.1.1.3 Processing Element

One of the targets of this architecture has been simplifying the control and the synchronization issues, and also keeping independent different tasks. Some of these characteristics have been explained along this section, through the exploration of the previous modules. However, now it is the turn to the PE, which is the core of this architecture since it is responsible of executing the DF

algorithm. Thus, the PE is composed by two modules, a router and a functional unit. The former, despite its name is not a common router, but its main task is also to distribute data. The latter is in charge of processing data according to the DF algorithm explained at the beginning of this chapter.

3.5.1.1.3.1 Router

The main interface between a PE and all its neighboring modules includes an element named router. This is an intermediate element between the data transmission modules (IMs and OMs) and the processing core of the architecture, the functional unit. Its main tasks involve the reception and transmission of unfiltered and filtered MBs, respectively. During the reception of unfiltered MBs, the router keeps MBs that will be processed by its attached functional unit, and transmit the others to the subsequent routers placed into the same row. Memory elements have not been included inside these elements, since the input data is directly passed through its functional unit. Also, during filtered MB sending process, data are directly read by the routers and stored in internal FU memories. The block in charge of generating the Reading sequence is the Input Controller, described above.

3.5.1.1.3.2 Functional Unit

A Functional Unit (FU) is the processing core of the proposed architecture, since it filters MBs. Hence, each FU behaves as a DF unit itself. Its structure, shown in Figure 3.13, consists in several elements of different nature.

Within the whole FU, the most important elements are the BS unit, in charge of determining the filtering strength, and the Filtering unit itself, which is able to process two LOPs at a time, one of them corresponds to the current LOP and the other one is its neighbor). In addition, two transposers have been included to prepare blocks in advance for the next filtering step (from the horizontal to the vertical, and vice-versa). The following memories have also been included in each proposed FU.

1. Q1/Q2: These memories store unfiltered MBs received from the router, as well as the intermediate results of the current MB between the horizontal and the vertical filtering stages.

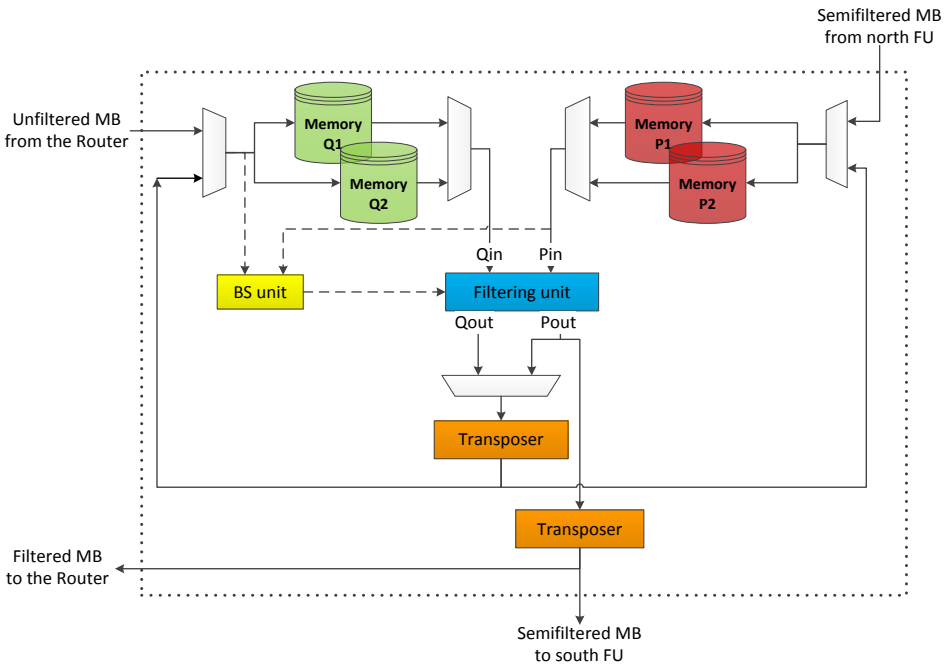


Figure 3.13 Functional Unit's architecture

2. P1/P2: These memories store the vertical semifiltered neighbor MB received from the previous FU, and the horizontal neighbor that has been filtered by the same FU one MB_{cycle} before.

The use of a pair of data memories of each type comes from the necessity of storing and loading data simultaneously every MB_{cycle}. That means that the filter unit loads data from Memory Q1 and Memory P1 in $MB_{cycle}(n)$, and from Memory Q2 and Memory P2 every $MB_{cycle}(n+1)$.

As for the behavior of the proposed FU, it operates in two separate and consecutive phases, splitting into horizontal and vertical filtering stages. This procedure satisfies the restrictions derived from the parallelism and data dependence analysis. Moreover, another advantage offered by the use of such a highly regular and modular architecture as the one proposed in this work, is the possibility

of swapping the PE by any other that achieves better performance or executes a different filtering algorithm.

3.5.1.1.4 Data formats and output memory accesses

In order to keep the advantages that the modularity and the flexibility of this architecture provide, it is important to reduce the number of memory accesses as much as possible. In this way, as it is natural, the higher number of PEs, the higher data requirements. This fact might generate bottlenecks that reduce the overall performance. However, within this architecture the data transmitted through the array is arranged according to a new concept named Extended Macroblock (EM). In the scope of this work, an EM is a data unit which includes the Lines of Pixels (LOPs) to be filtered, but also information required during filtering process. In this way, several independent memory accesses are avoided, since all necessary information is requested at a time. The full EM is composed of 32-bits words, which includes motion vectors and quantization parameter values, as shown in Figure 3.14. This strategy, together with the connectivity and the behavior of the whole architecture, reduces the number of accesses to the external memory.

Regarding semifiltered information shared between FUs, it is not necessary to transmit the full EM. Only the motion vectors and quantization values corresponding to the lower edge of a MB are transmitted, reducing the information sent through the array of FUs.

However, all filtered LOP values are transmitted, since the lower FU is the one that finish filtering the MB, and also the one that sends the full filtered MB to the Output Memory. Therefore, MB life cycle inside the DF is as follows. Each MB arrives to a certain FU, in $MB_{cycle}(n-1)$. In that FU it is filtered, firstly horizontally together with its left neighbor, and then vertically, with the upper one. Then, this semifiltered MB remains in this FU, acting like a left neighbor during the horizontal filtering of the next MB, during $MB_{cycle}(n)$. At the same time, a MB is being filtered, and the result is transmitted to the next FU. There, it is involved during vertical filtering of the element below.

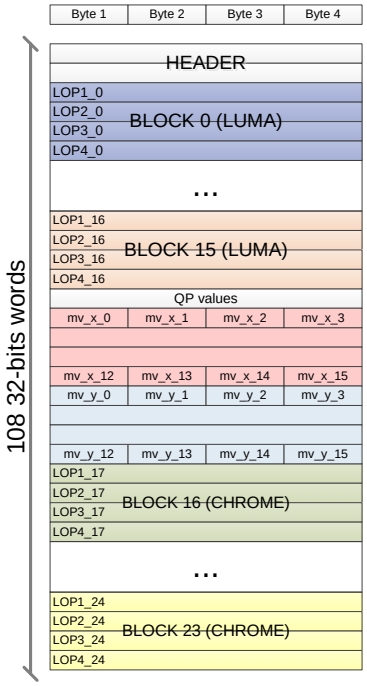


Figure 3.14 Extended Macroblock structure

Finally, it is send completely filtered to the OM, during $MB_{cycle}(n+1)$. In addition, semifiltered values are transmitted point-to-point from an FU to the following one, while in case of an unfiltered data loading the process does not finish until a new MB arrives to the last FU. Compared with state-of-the-art approaches, the number of external accesses is reduced since semifiltered information is directly transmitted from neighboring FUs, instead of having to read it from the external memory.

3.5.2 Architectural behavior

Hardware DF approaches based on a raster-scan pattern are not easily scalable; first of all because of the data dependences, which constraint the level of parallelism. Even, if this was possible, it is hard to isolate the processing element as a regular unit, and replicate it on the device like a multicore structure in order to parallelize the system execution. Another inconvenient is that many hardware DF proposals are rigid, completely static with an invariable structure and characteristics; independently of the final device or environment in which

they work due to they have been developed to perform efficiently under specific settings, i.e. the worst possible case. This fact motivates that many resources remain idle when the environmental conditions are more favourable (smaller frame rate, base layer bitstream, small video sequence, and so on). A direct consequence of this situation is a reduction of the overall efficiency of the system, and it also imposes an unnecessary waste of power consumption and silicon. As an alternative, some research works are focused on exploiting the MB-level parallelism implicit into the DF algorithm; i.e. the parallel strategies explained before are present within the DF state-of-the-art.

Thanks to the implemented MB distribution strategy and the modularity of the proposed design, it is possible to consider the scalability as another degree of freedom within the wide range of flexibility provided by the proposed architecture.

Considering these features, a sequence of different scalabilities for the proposed DF architecture is depicted in Figure 3.15. This representation shows all of the elements of this modular structure, but also how all of them are replicated according to the grown direction of the processing array (m×n). In this way, Figure 3.15.a represents the smallest configuration of this DF architecture, composed by one module of each type. This configuration behaves as a traditional raster-scan approach with only one filter unit. Then, with the increment into the number of columns in the array, from 1×1 to 2×1 as Figure 3.15.b depicts, the architecture is obligated to replicate all the modules, except the input and output control modules (IC and OC respectively). Furthermore, in any circumstance both modules, IC and OC, never are scaled. Both of them remain static. On the other hand, in case the array increases its number of rows, as Figure 3.15.c shows, only the PE is replicated, whereas the output modules (OM and OC) are just moved down to let enough space to the newest PEs. As it can see, the proposed DF architecture is a combination of different modules, not only processing elements, in charge of other tasks, like data arrangement and distribution, or communication with the rest of the system. Data transmission channels have been also included in Figure 3.15. The external communication

with the rest of the system is restricted to two one-directional connections and some control lines throughout the IC and OC, while each module into the architecture connected exclusively with its next neighbors.

In addition, its main feature is the capability to be part of more complex 1D or 2D structures of m×n FUs. This is due to the inclusion of previously defined redundant memory schemes, as well as the suitable control logic.

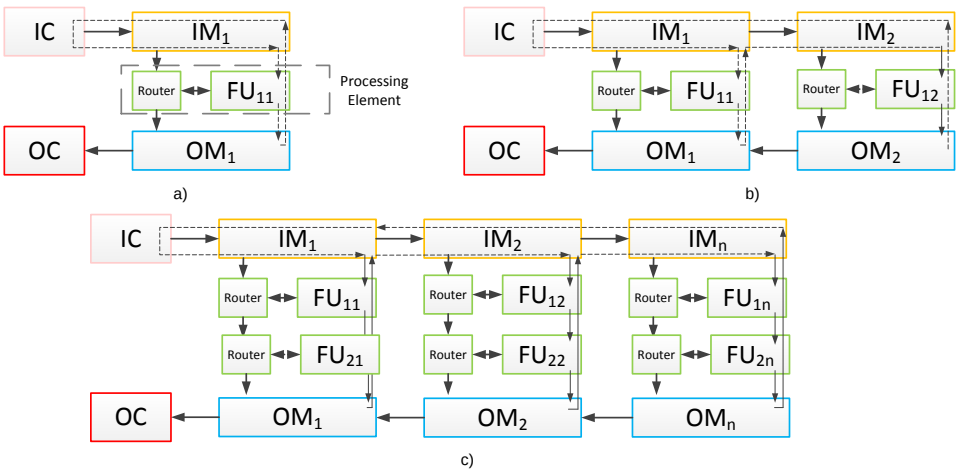


Figure 3.15 Three scalability levels of the proposed DF architecture; a) 1×1; b) 2×1; c) 3×2

Independently of the implemented dimension of the whole system in terms of rows and columns of modules, from now on configuration, all PEs are synchronized and executing the same task at the same time. That is, all of them always stay in the same execution state. This synchronization allows distinguish two execution stages: Phase H associated to the horizontal filtering, and Phase V regarding vertical filtering. However these stages involves not only to the PEs, but also to the rest of the modules present into the architecture, keeping the synchronization of the whole system. Hence, each IM retains the first *n* received MBs, transmitting the followings to the subsequent module. On router's side, during Phase V, each router keeps only the first received MB, transmitting the followings to the rows below. During Phase H, each router transmits filtered MB from its FU only when its neighbor above has already transmitted its data. The same policy has been implemented in OMs. Each one transmits its filtered MBs to the IC only when previous OM is empty.

Implemented data transmission policy, together with the reading sequence adaptation to each array size, allows a seamless architecture scaling process. In fact, all these modules can be reused without redesigning them, for any architectural configuration. More in detail, Table 3.IV details the tasks distribution for each module, structuring the tasks in phases. Thus, during Phase H, the IM receives and stores MBs, which come from the OC and will be filtered during the following MB_{cycle}. In addition, Routers send the MB filtered during previous MB_{cycle} to the OM. Then, and still in Phase H, OM store those MBs. Afterwards, in Phase V, unfiltered MBs, previously stored in Phase H in the IMs, are transmitted to the routers, and from them, to their attached Functional Units. At this point, OMs transmit filtered MBs to the external memory.

Table 3.IV Task distribution of each element

Filtering	Elements of the Architecture			
	IM	**Router**	**FU**	**OM**
Phase H	Receiving $MB_{(t+1)}$ from IC	Sending Filtered $MB_{(t-1)}$	Filtering H $MB_{(t)}$	Receiving Filtered $MB_{(t-1)}$
Phase V	Transmitting $MB_{(t+1)}$ to Routers	Receiving $MB_{(t+1)}$	Filtering V $MB_{(t)}$	Transmitting Filtered $MB_{(t-1)}$

Data transmission policy through the array has been also carefully designed considering scalability restrictions. To share this semifiltered information, point to point connections between FUs have been included. In addition, bypassing signals have been included from one column to the following one, through both Input and Output Memories. Therefore, each element exchanges control signals only with its next neighbors. Following this policy, a centralized control module is unnecessary, which avoids redesigning the control in case of architectural changes.

The modular structure of this architectural proposal, together with its internal data distribution strategy through the array and the regularity of each kind of module, make possible the exploitation of the scalability. This opens the window to a higher degree of flexibility, reusability and adaptability by means of varying the number of FUs which are concurrently processing according to the environmental scenario.

3.5.3 MB reading sequence and allocation strategy

Dependences among MBs restrict the amount of them that can be processed concurrently into the starting stage, and also define which MBs can be processed simultaneously along the time. In fact, only MBs with the results of its upper and left neighbors available can be filtered concurrently. Due to the enhanced wavefront scheme proposed in this work, even those that share semi-filtered MBs between its horizontal and vertical filtering steps might start performing, such as it is depicted in Figure 3.16. In this figure three PEs have been considered, and those MBs included in the same diagonal dotted bar, such as MB_4, MB_{11} and MB_{18} will be processed simultaneously.

Figure 3.16 Filtering process of a SQCIF image in stripes, including null MBs at the beginning

However, the proposed pattern only requires that left neighbor is available, while upper neighbor can be transmitted between horizontal and vertical filtering stages. This is possible by imposing certain restrictions to the design. The following enumerated limitations are directly deduced by analyzing the proposed parallelization.

1. The PE always behaves in the same way, independently of its position into the array.
2. The filtering approach has to be sequential (the horizontal filtering is executed before the vertical one) into the FU.

3. All the FUs should synchronize and execute the same kinds of filtering operations and tasks.

4. A full row of MBs is always filtered by the same PE. This constraint reduces the number of memory accesses and the data transactions, since the left neighbor is already stored in the FU.

In the case of the last element in each column, the subsequent PE corresponds with the first of the following column. Bypassing signals have been included from one column to the following one, through both Input and Output Memories. In addition, the first MB in a video frame (MB_0) is always filtered in the first PE (PE_0), independent to the enabled architectural configuration.

At the beginning and at the end of the frame, it is necessary to include a sequence of null MBs, to fill the PEs until they receive the current MB. This procedure avoids that some PEs start processing before the others. Then, after receiving this kind of MB, a PE remains inactive its filtering tasks, until it receives a new valid MB. Actually, a null M*B* is a specific sequence of simple control information without pixels data. Despite the fact that the inclusion of these null MBs reduces the final performance, since all the PEs must be active, this limitation has been set for the sake of simplicity of the control logic.

The algorithm parallelization and the allocation strategies entail reading sequence of MBs from the external memory. The initial MB of the first strip, it is the first being requested to the external memory, followed by subsequent MBs included in the same strip. Afterwards, the next strips are read using the same procedure. The length of each diagonal strip corresponds to the total number of FUs in the architecture that is m×n. When the number of rows within a video frame is larger than the number of PEs in the architecture, the image is split in disjoint regions, as Figure 3.16 depicts. Each of those regions is sent subsequently to the processing array. Then, each region of rows is processed in order, starting from Region 1 until the last one, as it was explained in previous sections. Furthermore, a FIFO memory has been included in the Input Controller to store semifiltered data, in order to deal with data dependences between border elements included in those disjoint regions. In this context, the semi-

filtered MB term refers to an MB that has not been filtered for all of its neighbors yet. In the case of the purely wavefront approach, in order to filter a certain MB, it is required that its semifiltered neighbors had been sent in advance to the FU, before it starts working.

Finally, an overview of the architectural behavior of a 2×1 array configuration is represented in Figure 3.17, where the time scheduling of a 5×4 MBs frame is processed by the system, but considering the behavior of the aforementioned components (IC, OC, IM, OM, PE). According to the notation used in Figure 3.17, 0_H and 0_V represents the horizontal and vertical filtering stages of the MB_0, respectively. Furthermore, green yellow and blue backgrounds highlight those MBs that belong to the top and left edges of the frame. In the case of the left MBs (blue background; MB_0, MB_5, MB_{10}, MB_{15}), they do not require left neighbors during their horizontal filtering processes. In the same way, the top MBs (yellow backgrounds; MB_0-MB_4) lack of top neighbors during the vertical filtering procedure. However, when the PE_1 starts processing the second, third or higher strip of MBs, it needs the top neighbors that have been previously processed by others PEs. It is in these moments when the semifiltered MBs gain relevance (represented in Figure 3.17 with the lightest colour). These special MBs move through the array following a predefined pattern, as Figure 3.18 depicts. Due to the fact that a semifiltered MB contains less information than an extended MB, it might be transmitted along all the components of the array easily and quickly. As Figure 3.18.a shows, in a 2×1configuration all the semifiltered MBs transmitted by FU_{21} must be received by FU_{11}. However, in order to keep a regular structure, the FUs must behave in the same way, independently of their position into the array. Thus, these MBs are sent to the OM, and then pushed up to the FUs, IM, IC, until coming into the FU_{11}. Similarly, Figure 3.18.b represents how the semifiltered MBs are moved through a 2×2 array configuration. In the case of filtering the frame shown in Figure 3.17, the FU_{11} and the FU_{21} filter the first two rows of MBs, whereas the FU_{21} the third and the FU_{22} the fourth one. Therefore, the FU_{21} will need the semifiltered MBs generated by

FU_{21}, such as MB_5-MB_9, in order to proceed with the vertical filtering of MB_{10}-MB_{14}.

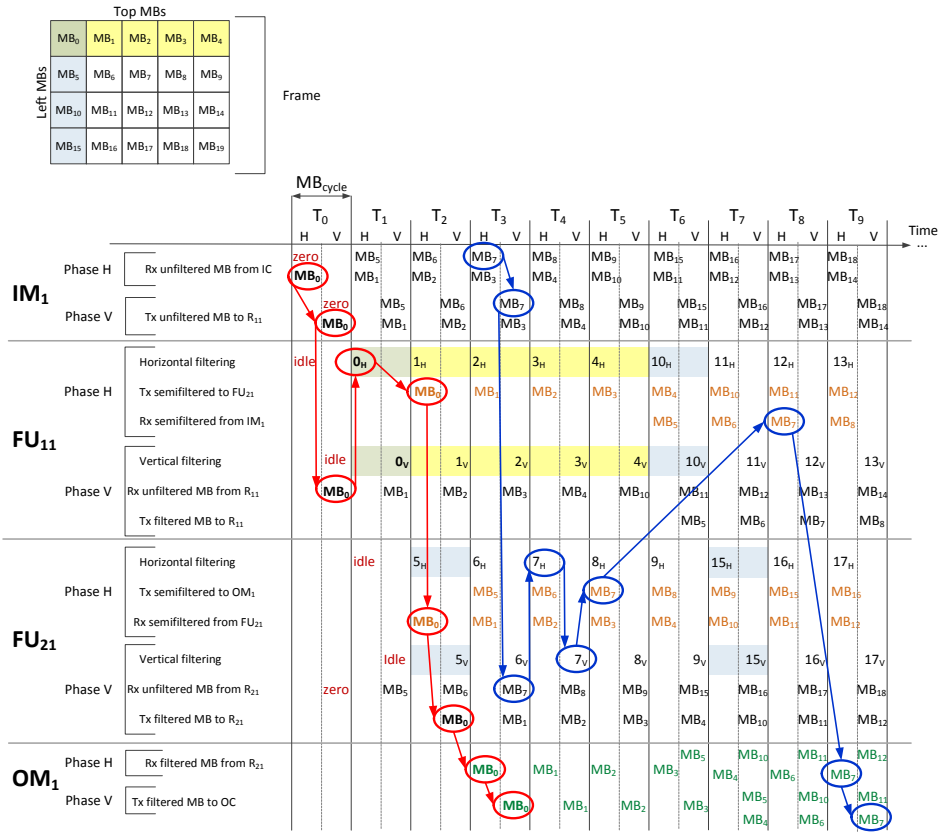


Figure 3.17 Time scheduling

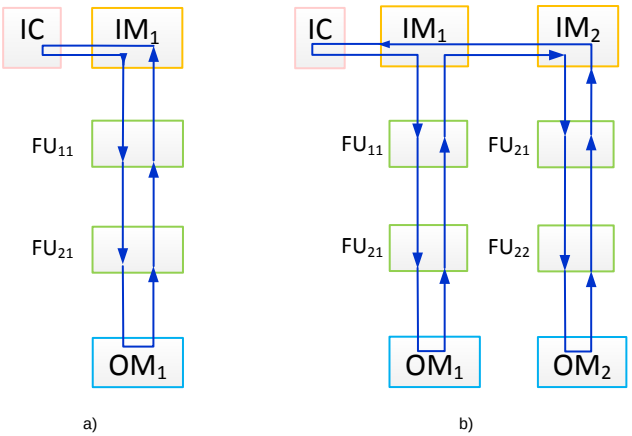


Figure 3.18 Semifiltered MBs

3.6 IMPLEMENTATION AND RESULTS

Once the fundamentals of our DF architecture have been explained, this section is focused on analyzing the performance of the proposed hardware design, and establishing a comparison with other DF state-of-the-art solutions in order to determine the goodness of our proposal.

The architecture has been described at RTL level, using hardware description languages and synthesizing the code on a Xilinx Virtex-5 FPGA (V5-LX110T), using ISE 13.3 Xilinx tools. Synthesis results shown in Table 3.V help to figure out the silicon area of every module, and also the number of logic resources for implementing different configurations. In the V5-LX110T is possible to implement a 3×3 array, because although the resources in term registers and LUTS are enough for 4×4 array, there is a shortage of the Block RAMs. However, as we will see later, a 3×3 array is appropriate for processing UHDTV video. As was expected, the PE module is the most demanding in terms of resources, requiring up to 78% of the FPGA. Attending to these, it is noticed that the resource overhead introduced by the communication (IM and OM), and the control (IC and OC) modules is small compared with the contribution of the PE.

Table 3.V Synthesis results on a Xilinx V5-LX110T

Resources	Modules of the proposed DF architecture						Configurations on the proposed architecture		
	IC+OC	IM	OM	PE	Total	PE/Total (%)	1×1	2×2	3×3
Slices Reg.	399	183	99	2363	3044	77.62%	3196	10493	22510
Slices LUTs	590	148	161	2710	3609	75.09%	3609	11477	24622
BRAM 36Kb	8	2	4	12	26	50%	24	64	128

In fact, control and distribution modules together suppose about 25% over the total synthesis results. The resources mismatching between the architectural configuration 1×1, and the total sum of all the modules individually is due to the inclusion of bus-macros to interconnect one module to another, avoiding using routing paths. These bus macros are dedicated and unidirectional lines for transmitting data, and they place around the module is fixed. Thus, Figure 3.19 depicts a screenshot with the floorplanning of one PE with its associated bus

macros. In this case, the bus macros have been highlighted in yellow on top and bottom of the figure, whereas the logic resources have been represented in green.

Despite the fact that the latest Xilinx design flow, from the v12 release onward, eliminates the use of fixed bus macros, in this design they are necessary. This is because the newest Xilinx solution does not provide enough flexibility to our proposed design and it is unsuitable for this kind of scalable architecture.

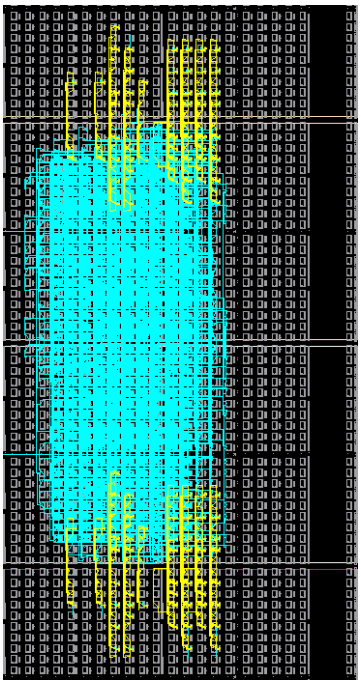


Figure 3.19 Floorplanning of one PE with its bus macros on top and bottom

This architecture is characterized by the ratio between the constant area occupied by the control module (IC and OC together) and the area demanded by the scalable modules, which varies according to different configurations. This ratio gives information about how the scalability impacts on the whole system area. The way to measure this is by comparing the logic resources of control module replicated m×n times faced to an m×n configuration of the proposed architecture. The experiment evaluates a cluster of configurations, starting from 1×1 and finishing with a 16×16. Results are shown in Figure 3.20, where the X on the bottom right and Y on the bottom left axes represent the configuration pa-

rameters m and n, respectively. On the other side, the Z axis denotes all the values of the ratio. In Figure 3.20 the colours of the surfaces represent different values, moving from the poorest values represented in blue (1×1 configuration) that it is equivalent to a 0.162, to the dark red (16×16) equivalent to a 0.216. Other configurations like the 2×2, 3×3, 4×4 reaches the following values 0.195, 0.205, and 0.208 respectively. As conclusions, the scalability is got at an affordable price.

On the one hand, the behavioral simulation results confirm that the number of clock cycles that a simple FU requires to complete one MB filtering ascends to 240, whereas after the post place and route stage the working frequency reported by the tool is 124 MHz. This frequency is more than enough to process most of the video standards having only a few PEs in the system.

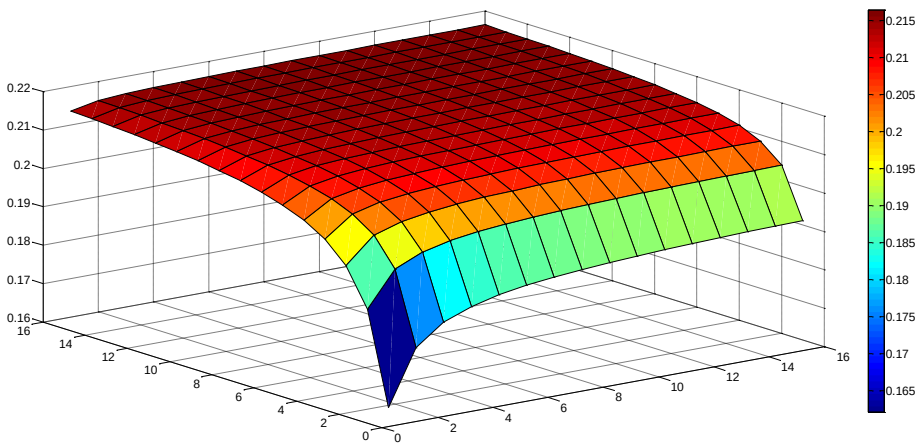


Figure 3.20 Area efficiency surface for several m×n array configurations

In this sense, Figure 3.21 represents the relationship between the number of PEs in the system and the frequency required to process eight traditional video formats for multimedia applications (SQCIF, QCIF, CIF, 4CIF, 16CIF, HDTV@720, HDTV@1080p and UHDTV), being real time complaint (30 frames per second). The required frequency is calculated by following the equation (3):

$$\textbf{frequency} = \mathrm{MB}_{\mathrm{cycles}}\ \text{per frame} \cdot \mathrm{FPs} \cdot \text{clock cycles}_{\mathrm{MB}} \quad (3)$$

The dotted red lines showed on Figure 3.21 in the last three windows (HDTV@720, HDTV@1080p and UHDTV) are the top limit for the architecture.

Results demonstrate that the proposed architecture is capable of processing all the video formats with just one PE, with the exception of an UHDTV that demands at least 8 PEs. In this extreme case is important to use, at least, eight PEs otherwise the real time is not guaranteed. In any case, this limit is taking in account that the architecture should be compliant with:

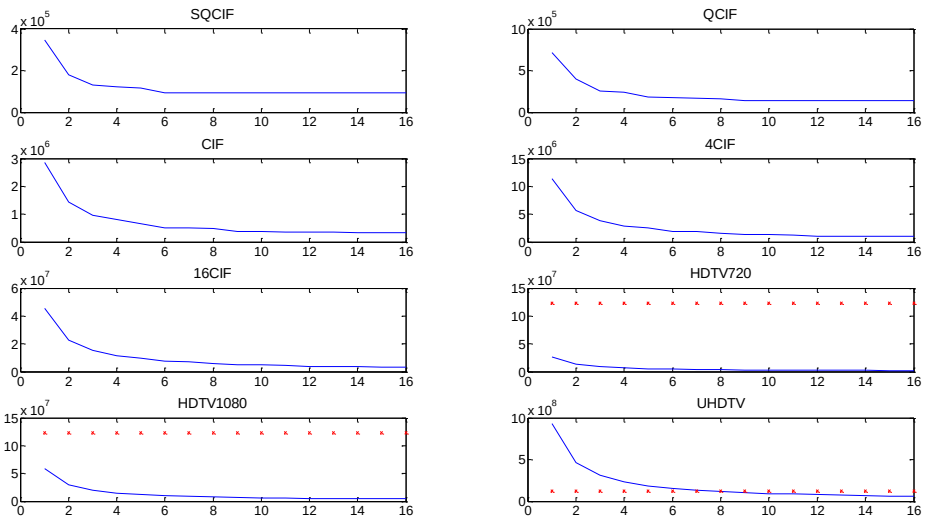


Figure 3.21 Number of PEs versus the clock frequency according to different video formats

1. Filtering order constraint. The latest video standards obligate to fulfil with a determined filtering order at MB-level, in the sense that any MB has to be filtered horizontally firstly, and then vertically.

2. Filtering approach. The implemented structure of the FU is the simplest one, in terms of the number of filter units, since there is only one unit involved. As a consequence, the control and the whole design of the FU are easier to implement, but at cost of decreasing the final performance due to the MB processing is completely sequential. That means the luminance has to be filtered first, and then is the turn to the chrominances components one after another.

3. Synchronization pattern. Independently of the filtering strength applied on a block, every filtering operation spends the same number of clock cycles. This fact ensures the proper synchronization among all the FUs of the processing array. Thus, the horizontal filtering requires 120 clock cycles, and the vertical filtering other 120 cycles. These numbers include filtering operations, but also data distribution and transposition operations.

Once the number of PEs is higher than the height of the image in MBs, the minimum clock frequency necessary to process a real time video sequence is the same than the case when both parameters (number of PEs and image height) are equal. This fact makes that, for a SQCIF format does not make sense using more than six PEs, because the maximum number of MB-level in parallel is fixed by the actual height of the image in MBs. An excess on the number of PEs is translated into a waste of resources, and consequently an inefficient configuration.

These results demonstrate the benefits of exploiting the enhanced wavefront strategy over a scalable application like ours. Furthermore, these results highlight the powerful of designing a modular and scalable architecture. In this particular example, the case study has been focused on the DF algorithm, but the designed framework moves forward and it could be reused for solving different nature problems.

The modular structure of this architecture allows the reuse of all the elements to implement other kinds of tasks, changing the design of the processing core (FU). For doing this, it would be necessary to adapt the FU (or redesign it), and adjust the synchronization and control policies regarding the speed of the data transfers and rearrange the data distribution among the modules. In addition, the input and output control modules (IC and OC respectively) could be reused as an interface between the proper architecture and the rest of the system, but also the reconfigurable input and output memories within (IM and OM respectively) might be reused in order to distribute the data through the array of FUs. As for the routers, these elements may be reused depending on the characteristics or necessities of the task to implement.

The proposed scalable DF architecture has been implemented in a final embedded system in where two on-chip shared memories have been included in order to supply data to the system. The DF directly demands MBs to the input shared memory through the IC module, at a rate of 64-bit word per clock cycle. Once the data is into the DF architecture, the unfiltered MBs pass through the corresponding elements at the same rate (64-bit word per clock cycle). The input shared memory has been dimensioned to be able to store up to 75 MBs, which is above the amount of MBs to be concurrently filtered by the largest DF structure, sized 8×8 that requires 64MBs in order to start processing data. This on-chip memory avoids accesses to external memories, and the DF is never stalled, even in the case of using large configuration structures.

Previously to the filtering stage, in which PEs process the unfiltered MBs, those data must be stored in the IMs in an initial loading stage. In this sense, a latency of m×n×54 clock cycles is therefore required in order to full the FIFOs of all the IMs. Due to the inherent parallel nature of the proposed DF design, the loading stage of IMs, with the first MBs of a new frame, and the filtering stage of the last MBs of the previous one are overlapped. Since a MB_{cycle} corresponds to 240 clock cycles, until four MBs might be loaded to the IMs without any cost, reducing the inter-frame latency penalty. In addition, when the DF was integrated as part of an embedded system using the aforementioned FPGA, the final working frequency was set to 124 MHz, without any behavioral degradation.

3.6.1 State-of-the-art comparison

Until now, the proposed architecture has been based on an absolute parameter (MB_{cycle}). However, this is an architectural parameter, which does not give specific information about how fast is the design after its implementation. In addition, an accurate comparison with other proposals demands information related to an implementation parameters. In this sense, the maximum clock frequency and the number of clock cycles, which are necessary for filtering a full MB by FU, are parameters that characterize accurately the real performance of a specific design.

However, authors publish different aspects of their designs and there are no common criteria. This disparity of criteria makes difficult to compare different approaches in terms of area, power and throughput. Nevertheless, Table 3.VI collects relevant information from different DF implementations belonged to the state-of-the-art since 2006 until nowadays, which describes the main features and results of these solutions. The third column (N filters per PE) shows the number of filter units contained in every PE, according to the DF design. The difference between MB-level parallelization and Scalable design is based on the internal design of the PE. For example, authors in [LAM09] define a DF design in which the PE might be replicated in order to process several LOPs simultaneously, but at the end every MB is processed sequentially following a typical raster-scan pattern. Otherwise, [CM07] is a DF solution based on PEs that process MBs and might be replicated allowing parallel processing. As a result, we can determine that an approach that exploits a MB-level parallelization is scalable but not the opposite. Thus, a scalable design is not necessarily parallelized at MB-level. The last three columns collect performance information completely dependent on the technology and the design. N clock cycles per MB represents the total number of clock cycles necessary to process a complete MB, and Frequency is the maximum processing speed that these designs are able to process. Finally, the last column, Platform, is the vendor and family of FPGAs selected to implement the DF designs.

The analysis of the information collected in Table 3.VI shows that just the approach [Ern07], apart from our proposal, is focused on exploiting a MB-level parallelization; and this one together [CM07] and [LAM09] are scalable at LOP-level. Within the range of frequencies reached by these solutions varying from 42 MHz to 170.95 MHz, our proposal is in the average values. The same happens evaluating the number of clock cycles per MB, since into the diversity of values moving from 53 clock cycles until numbers higher than 600 cycles, our proposal needs 240 clock cycles. In general, the FU of our scalable DF is not the fastest processing unit into the state-of-the-art, neither the slowest, but it has the possibility of being scalable.

Despite of the fact that [ERN07] is another scalable solution that takes advantage of the MB-level parallelization, its control is completely centralized and its scalability requires several modifications in the design every time that a new PE is added. On the contrary, the distributed control and the homogeneity of the modules of our proposal transform the scaling process into an easy task.

Table 3.VII shows some DF approaches of the state-of-the-art implemented using ASIC technology. Although this technology is more specific and optimized than any FPGA design, the scalable proposed DF is better than the expected results when it is directly compared to the ones with only one filter in their designs. Furthermore, all these approaches follow a raster-scan pattern in order to filter a full video frame, which limits the parallelization-level in a short future. However, it is important to highlight that our FU is susceptible to be improved by means of adopting some memory management techniques, improving the datapath, pipelining, etc. Moreover, it could be possible to explore more in detail the inter-parallelism and the techniques used by all these solutions, but that is not the purpose of this work.

Table 3.VI and Table 3.VII describe the main characteristics of many DF approaches belonged to the recent state-of-the-art; however the establishment of a fair comparison among all of them is still complicated. Moreover, another way to analyze the strengths and weaknesses of all of this collection of DF approaches is to use the number of frames per second that they are able to process, when an HDTV@1080p image is processed.

Another interesting comparison measure is the efficiency (*EF*) [TP09], since it permits to figure out the level of resource utilization. Then, the higher efficiency the better. This parameter is defined following the expression (4):

$$EF = \frac{1000}{Kgates \cdot Clock\ cycles\ per\ MB} \qquad (4)$$

A complete comparison might be seen on Table 3.VIII, where different kinds of works have been considered, including both FPGA and ASIC-based approaches. The last three columns show generic measurements such as the throughput (macroblocks processed in a microsecond - MBs/µs), the throughput

per area (macroblocks per second and area – MBs/(Kgates·second)) and the efficiency.

Table 3.VI FPGA-based DF state-of-the-art comparison

Ref.	filter per PE	MB-level parallelism	Scalable design	clock cycles per MB	Freq MHz	Resources Kgates	FPS HDTV 1080p	Platform
[WL06]	4	No Raster-scan	No	608	42.8	19.59	8.76	Xilinx Virtex-II
[CM07]	1	No Raster-scan	Yes LOP-level	4480	100	1.78	2.78	Xilinx Virtex-E
[ERN07]	4	Yes Traditional wavefront	Yes MB-level	140	43.08	155.57	38.27	Xilinx Virtex-5
[RSB07]	1	No Raster-scan	No	256	135	40.60	65.59	Xilinx Virtex-II Pro
[PH08]	1	No Raster-scan	No	5376	72	21.2	1.67	Xilinx Virtex-II
[CAC09]	4	No Raster-scan	No	53	166.56	24.87	623.46	Altera Statrix-IV
[LAM09]	16	No Raster-scan	No	105	150	161.15	177.68	Altera Statrix-II
[KJ10]	1	No Raster-scan	Yes LOP-level	192	103	233.63	66.72	Xilinx Virtex-4
[SCA10]	4	No Raster-scan	No	260	145.54	13.11	69.62	Altera Statrix-II
[MBT+11]	4	No Raster-scan	No	55-71	170.95	70.37	299.47	Xilinx Virtex-5
Proposal **1×1**	1	Yes Improved wavefront	Yes MB-level	240	124	33.83	64.26	Xilinx Virtex-5
Xilinx Virtex: 180nm; VirtexE-Virtex2-Viertex2P: 130nm; Virtex4: 90nm; Virtex5:65nm Altera StratixII: 90nm; StratixIII: 65nm; StratixIV: 40nm								

Table 3.VII Standard cell-based DF state-of-the-art comparison

Ref.	filter per PE	MB-level paral-lelism	Scalable design	clock cycles per MB	Freq. MHz	Resour. Kgates	FPS HDTV 1080p	Tech.
[LLL07]	1	No Raster-scan	No	236	100	21.1	52.70	0.18µm
[MBT+11]	1	No Raster-scan	No	222-210	200	18.7	112.05	0.18µm
[XC08]	1	No Raster-scan	No	204	200	21.4	121.94	0.18µm
[CCG09]	1	No Raster-scan	No	260	225	38.4	107.63	0.13µm
[MC09]	1	No Raster-scan	No	192	70	12.3	43.35	0.18µm
[Lin09]	2	No Raster-scan	No	100-48	196	22.9*	243.78	0.13µm
[TVM09]	2	No Raster-scan	No	348-228	100	19.9	35.74	0.18µm
[ZZZ+09]	2	No Raster-scan	No	136	200	17.9	182.91	90 nm
[LCC10]	1	No Raster-scan	No	212	100	12.2	58.67	0.18µm
[KT10]	1	No Raster-scan	No	246-226	400	19.2*	202.24	0.18µm
[Che10]	4	No Raster-scan	No	76-48	135	41.6	220.93	0.18µm
[CXL10]	2	No Raster-scan	No	96	150	23.9	194.34	0.13µm
[CCC10]	1	No Raster-scan	No	196-100	200	19.8	220.93	0.18µm
Proposal **1×1**	1	Yes Improved wavefront	Yes MB-level	240	124	33.83	64.26	65 nm
Proposal **2×2**						107.60	257.05	
Without considering SRAMs								

Table 3.VIII Throughput and efficiency for different DF hardware approaches

Ref.	Tech	Family	clock cycles per MB	Freq. MHz	Resources Kgates	Through MBs/µs	Through / Area	EF
[WL06]	FPGA	Xilinx V2	608	42.80	19.59	0.07	3.59	0.08
[LLL07]	ASIC	0.18 µm	236	100	21.10	0.42	20.08	0.20
[LCC10]	ASIC	0.18 µm	212	100	12.20	0.47	38.66	0.39
[JLY+10]	ASIC	0.18 µm	246	400	19.20	1.63	84.69	0.21
[CM07]	FPGA	Xilinx VE	4480	100	1.78	0.02	12.54	0.13
[ERN07]	FPGA	Xilinx V5	140	43.08	155.57	0.31	1.98	0.05
[RSB07]	FPGA	Xilinx V2P	256	135	40.60	0.53	12.99	0.10
[PH08]	FPGA	Xilinx V2	5376	72	21.20	0.01	0.63	0.01
[CAC09]	FPGA	Altera Stratix-III	53	265.67	66.39	3.14	126.34	0.28
[LAM09]	FPGA	Altera Statrix-II	105	150	161.15	1.43	8.86	0.06
[KJ10]	FPGA	Xilinx V4	192	103	233.63	0.54	2.30	0.02
[SCA10]	FPGA	Altera Stratix-IV	260	145.54	13.11	0.56	42.69	0.29
[MBT+11]	FPGA	Xilinx V5	71	170.95	70.37	2.41	34.22	0.20
[CZF+08]	ASIC	0.18 µm	222	200	18.70	0.90	48.18	0.24
[XC08]	ASIC	0.18 µm	204	200	21.40	0.98	45.81	0.23
[CCG09]	ASIC	0.13 µm	260	225	38.40	0.87	22.54	0.10
[MC09]	ASIC	0.18 µm	192	70	12.30	0.36	29.64	0.42
[Lin09]	ASIC	0.13 µm	100	196	22.90	1.96	85.59	0.44
[TVM09]	ASIC	0.18 µm	348	100	19.90	0.29	14.44	0.14
[ZZL+09]	ASIC	0.09 µm	136	200	17.90	1.47	82.16	0.41
[Che10]	ASIC	0.18 µm	76	135	41.60	1.78	42.70	0.32
[CXL10]	ASIC	0.13 µm	96	150	23.90	1.56	65.38	0.44
[CCC10]	ASIC	0.18 µm	196	200	19.80	1.02	51.54	0.26
1×1	FPGA	Xilinx V5	240	124	33.83	0.52	15.27	0.12
2×2	FPGA	Xilinx V5	60	124	107.60	2.07	19.21	0.15
3×3	FPGA	Xilinx V5	27	124	230.83	4.59	19.90	0.16
4×4	FPGA	Xilinx V5	15	124	402.03	8.27	20.56	0.17

3.7 CONCLUSION AND FURTHER RESEARCH

This paper presents a scalable deblocking filter architecture implemented on hardware. It exploits an optimized MB-level wavefront parallelization strategy. This characteristic provides the necessary flexibility to allow an easy reuse of the design among different profiles and scalability levels of the latest video standards video standard. The homogeneous design of its FUs provides the enough flexibility and regularity to be able to explore different levels of scalability, since varying the number of FUs is possible accelerate filtering process of a reconstructed image. This customizable parallelism offers the opportunity to adapt the design to different environments and devices, by changing some generic parameters which impact on the final performance in terms of area and speed.

Further research is focus on improving the proposed design in several aspects. One goal is to reduce the number of internal memories, without degrading the behavior or increasing the complexity of control tasks. Furthermore, we would like to explore the possibility to extend this architecture to other functional blocks into the SVC decoder.

Chapter 4

Exploiting the scalability for hyperspectral image processing: linear unmixing

Hyperspectral imaging instruments capture and collect hundreds of different wavelength data. As a result, tons of information must be stored and processed. This characteristic makes this application very suitable for being implemented as a scalable system. This chapter explores the viability of the scalability for an endmember extraction algorithm. The scalability is determinant for saving resources, silicon area, cost and gaining in flexibility as part of the embedded SoCs design. Whether those scalability changes want to be applied at run time, it is mandatory to use the dynamic reconfigurability.

4.1 OUTLINE

Hyperspectral imaging is an emerging and fast growing area in remote sensing. The process of the data acquisition by a hyperspectral sensor results in tons of information that must be stored and processed efficiently, reaching a balanced compromise between flexibility and high performance at the same time. On one hand, modern FPGAs are perfect hardware candidates to fulfill with these purposes. On the other hand, the exploitation of the scalability on hardware architectures actively contributes to increase two main aspects: parallelization and reusability. However, this is an unexplored research area within hyperspectral image processing. In this context, this chapter presents three different scalable architectures, implemented on an FPGA, which perform the Modified Vertex Component Analysis (MVCA) algorithm, as part of the hyperspectral linear unmixing processing chain. As a consequence, not only high performance but flexible and reusable designs have been designed. Moreover, final results demonstrate the influence of the implemented parallelization methodology over the final performance, but also how this one varies according to the enabled scalability level.

4.2 EXPOSING THE PROBLEM

The human being curiosity has motivated the development of a global industry based on the Earth's surface observation, but remotely from the space. This technology has permitted a better understanding about environmental issues. However, there exist several difficulties regarding the treatment of these kinds of images due to the large amount of information collected during the image capturing process, especially for the case of hyperspectral remote sensors. Furthermore, the next generation of remote sensing hyperspectral sensors will allow capturing more hyperspectral cubes per second with much more information per cube. At this respect, the European Space Agency (ESA) has already flagged up in 2011 that "*data rates and data volumes produced by payloads continue to increase, while the available downlink bandwidth to ground station is comparatively stable"* [Tra11].

Hyperspectral sensors collect image data simultaneously in dozens or hundreds of narrow, adjacent spectral bands. These measurements make possible to derive a continuous spectrum for each image cell. The resulting data volume, instead of being a 2-dimensional image, is a 3-dimensional one (or hyperspectral cube) with two spatial and one spectral dimensions. The high spectral resolutions of these kinds of images demands fast processing solutions, in order to perform a more efficient exploitation of hyperspectral data sets in various applications. Consequently, designing solutions able to take advantage of the ever increasing dimensionality of sensed hyperspectral images for real time applications has gained a significant relevance during the last decade.

For the particular case of Earth Observation satellites, these on-board systems should at least accomplish the following three mandatory characteristics. First, they must allow high computational performance, since all the state-of-the-art algorithms for compressing and/or processing a hyperspectral image have a huge associated computational burden. Second, they should have a compact size and reduced weight and power consumption, due to the inherent nature of remote sensing satellites. Last, but not least, they must be resistant to damages or malfunctions caused by ionizing radiation, present in the harsh environment of outer space. Furthermore, it would be highly desirable that these high performance on-board processing systems could also show a high degree of flexibility. In this way, they could be adapted to varying mission needs, faults, and/or to the requirements of processing algorithms and standards in a future.

Over the last few years, reconfigurable hardware solutions such as FPGAs have been consolidated as the standard choice for on-board remote sensing processing [LVG+13]. Their success lies on their smaller size and weight compared with traditional cluster-based systems, as well as to their lower power dissipation figures when they are compared with GPUs. Furthermore, the increasing number of FPGAs with tolerance to ionizing radiation in space turns these devices into robust hardware solutions. Because of these reasons, the present work has been focused on FPGAs.

On the other hand, in all those cases in which the number of resources is limited and there are no choices for increasing its amount, or even for replacing the damaged ones, the scalability is a desirable characteristic to be exploited as part of a flexible solution. Furthermore, designing scalable approaches is a useful technique for parallelizing its execution. In this sense, the combination of hardware design techniques, together with the FPGAs' nature facilitates the adaptation of those designs to different constrained scenarios, avoiding redesigning the whole system.

The large amount of information collected in a hyperspectral image makes necessary to create diverse kinds of processing tasks, in which the spectral unmixing plays an important role. The lack of spatial resolution of the sensor provokes that a single pixel in an image might contain several substances. At this point, the spectral unmixing algorithms estimate the fraction of the pixel area covered by each material present in the scene. A classical methodology for analyzing hyperspectral images is based on linear unmixing models. These ones assume that all the hyperspectral pixels are composed by linear combinations of certain spectral information contained in the image. Despite the fact that these models are not the most accurate ones within the unmixing area, since they obviate the spatial information, they are the most widely used due to their computational lightness and clear conceptual meaning. Within the wide spectra of linear unmixing algorithms, the vertex component analysis (VCA) [NB05] is one of the most successful and popular since it reaches better results than others. This algorithm is very intensive in computation, and this is the reason because it is a well-suited candidate to be accelerated on hardware. Furthermore, the modified virtual component analysis (MVCA) algorithm [LHC+12a] behaves very similar than the VCA algorithm, simplifying some of the VCA operations. This work develops hardware acceleration on FPGAs, and scalable designs for improving the performance execution and the resources' reusability. More in detail, this chapter explores, proposes and analyses several scalable hardware solutions in order to overcome the computational complexity of the MVCA algorithm. This will be the base for developing a reconfigurable solution for a linear unmixing algorithm that will be presented in the next chapter.

4.2.1 Fundamentals of hyperspectral imaging

A hyperspectral image is an image in which one point (pixel) is described by many values. Traditional images are composed by sequences of pixels described by only one value of intensity (such as always happens in a grey-scale image), or by three different colours components (like in a RGB image, traditionally used on modern screens or displays). However, in hyperspectral images every spot is described by a complete array of spectral values. All these values correspond to the light contribution detected in that point of the picture but measured in different and tiny spectral bands. Typically, a hyperspectral sensor works in a limited number of spectral bands. They might vary from dozen to hundreds, where the spectral range is not usually constrained to the visible spectrum, being possible to include infrared and ultraviolet measurements.

These kinds of images are depicted as a cube, formally named as hypercube or hyperspectral cube, composed by overlapped images corresponding to the same surface but in different spectrum (*x, y* coordinates with many spectral bands, λ). Hence, every layer is a 2D image observed in a particular wavelength. This cube might be sized as width (samples, *x*) by height (lines, *y*) by length (bands, λ), as Figure 4.1 shows.

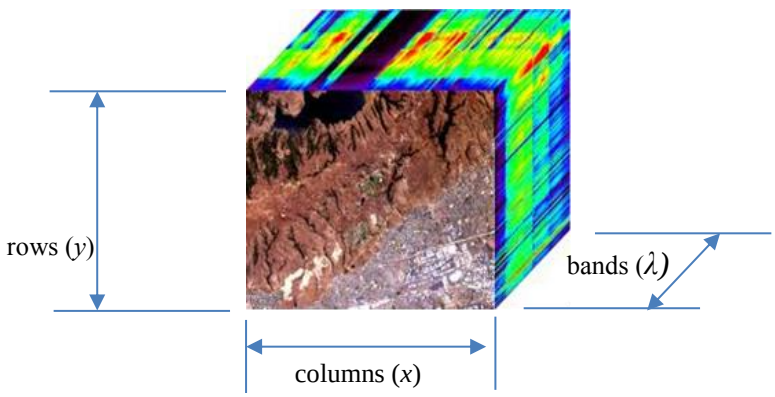


Figure 4.1 Hyperspectral cube

Mathematically, the hypercube can be represented by a matrix, as the expression (1) shows:

$$Y = \begin{bmatrix} (x_1, y_1, \lambda_1) & \cdots & (x_m, y_n, \lambda_1) \\ \vdots & \ddots & \vdots \\ (x_1, y_1, \lambda_N) & \cdots & (x_m, y_n, \lambda_N) \end{bmatrix} = [r_1, r_2 \dots r_R] \qquad (1)$$

where the index *m* is the number of pixels on the *x* dimension (columns), the index *n* is the number of pixels on the *y* dimension (rows), and *N* is the number of spectral bands. The hyperspectral image contains $R = m \times n$ pixels of N bands each and r_R represents a pixel in all N bands. The separation in wavelengths of these bands is determined by the current sensor technology. The lowest bands' distance, and hence the highest spectral resolution, the better results are expected by the application. Nevertheless, higher resolutions mean more information to be processed, and consequently the data processing demands are considerably increased.

4.2.2 Linear unmixing of hyperspectral imaging

The nature of a hyperspectral image is determined, among others, by two main factors. One of them is the number of bands considered (or spectral resolution), and the other one is the purity of every pixel (or sample). In general, it is possible to classify every pixel in one of the following categories: a pure pixel, or a mixed pixel. A pure pixel contains information just for one specific material, whereas a mixed pixel reveals information from several materials. According to this, a mixed pixel might be considered as a combination, in different proportions, of different materials. Every material should be pure, and it is named as endmember, and the quantity in which every material contributes to the final pixel is known as abundance. This idea is graphically represented in Figure 4.2.a, in which a mixed pixel is formed by a combination of three different materials, every one contributing in different proportions. More in detail, Figure 4.2.b represents the spectral signatures of the ground, tree and field pure pixels. The spectral signature of a pixel corresponds with its radiance information in all the considered spectral bands, and it characterizes to each observed object. Then, a spectral signature might be considered as an *N*-dimensional vector, where *N* is the number of spectral bands, and it can be used as a fingerprint for identification purposes.

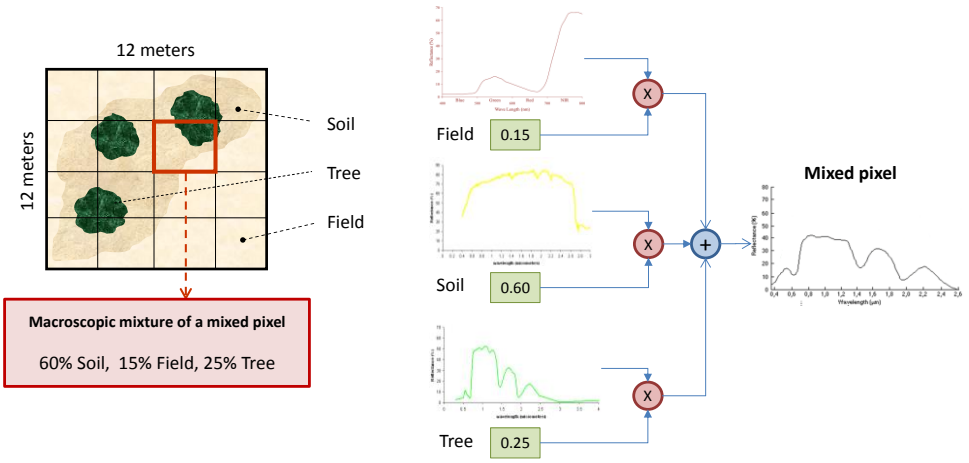


Figure 4.2 Composition of a mixed pixel; a) 4x4 pixels' scene; b) mixed pixel based on three endmembers (pure pixels)

Depending on the mixing scales at each pixel and the geometry of the scene, the observed mixture is either linear or nonlinear. Linear mixing holds when the mixing scale is macroscopic and the incident light interact with just one material. Despite the linear mixing/unmixing models simplicity, they reach an acceptable approximation of the light scattering mechanisms in many real scenarios. Mathematically, a hyperspectral image might be represented as a matrix with N bands and R pixels ($Y \in \mathbb{R}^{N \times R} / \ Y = [r_1, r_2, \dots, r_R]$), in which every column of the matrix is the spectral signature of the pixel y_i, and the rows are the hyperspectral image. Then, a mixed pixel according to the linear unmixed model might be represented as equation (2):

$$r = \sum_{i=1}^{i=p} a_i \cdot e_i + \varepsilon \qquad (2)$$

Thus, each captured pixel ($r = [r_1, r_2, \dots, r_N]^R$) in a hyperspectral image can be represented as the linear combination of a finite set of pure pixels, or endmembers (e_i), weighted by an abundance factor (a_i). The total number of endmembers of an image is p, and ε represents a source of additive noise introduced by the acquisition process. In order to do this, spectral linear unmixing process consists of three main stages, as Figure 4.3 depicts: 1) the estimation of the number of endmembers (p) present in the hyperspectral image, 2) the de-

termination of those endmembers, and 3) the estimation of the corresponding abundances for each pixel.

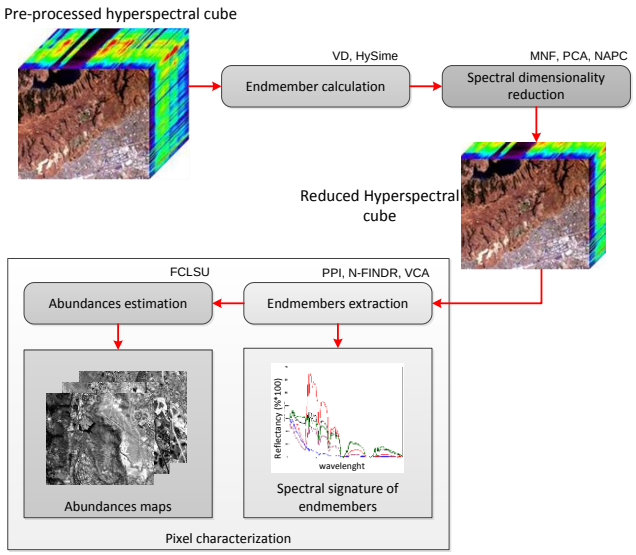


Figure 4.3 Hyperspectral image processing

4.1.1.1 Endmembers calculation and dimensional reduction

The number of endmembers present in a given scene is often much smaller than the number of available spectral bands. Therefore, spectral vectors generally lie on a lower-dimensional (linear) subspace. Some well-known signal subspace identification algorithms are Virtual Dimensionality (VD) [CD04], and Hyperspectral signal Subspace identification by minimum error (HySime) [BN08] for the first stage shown in Figure 4.3. Then the Maximum Noise Fraction (MNF) [Gor00], the Noise Adjusted Principal Components (NAPC) [Rog94] and/or the Principal Component Analysis (PCA) [Jol02] are algorithms focused on the spectral dimensionality reduction.

4.1.1.2 Endmembers extraction

There exists a wide collection of endmembers extraction algorithms, since they are the core of the hyperspectral imaging analysis. It is in this stage where the pure pixels are selected according to the image information. Despite the fact that in this field there are several types of algorithms, the most popular ones are the geometric approaches because of the simplicity of the concept. Some of

them require previously a dimensionality reduction, using Principal Component Analysis (PCA) or others.

4.1.1.3 Abundances calculation

This step consists of estimating the influence of every endmember in the composition of every pixel of the hyperspectral image. This stage normally uses Least Square Error (LSE) methods in order to derive fractional abundances. Furthermore, Fully Constrained Linear Spectral Unmixing (FCLSU) [SMP+10] is an extended algorithm used for the abundances estimation, which imposes two constraints to the abundances. Firstly, all the abundances has to be non-negative ($a_i \geq 0 \; \forall \; i = 1..p$); and the second constraint obligates that the sum of abundances for a given pixel is equal to one $\left(\sum_{i=1}^{i=p} a_i = 1\right)$.

4.2 ENDMEMBER EXTRACTION ALGORITHMS

Geometrical approaches exploit the fact that linearly mixed vectors are in a simplex set or in a positive cone, since assume the endmembers are the most different pixels in the entire image [BP11]. This means that in a graphical representation, where each pixel is represented in a space form by N axis (each axis corresponding to a spectral band) the endmembers are the most extreme pixels (Figure 4.4). The pattern conformed by the edge pixels (endmembers- e_1, e_2 and e_3), which enclose to all the rest of pixels of the image, is known as simplex. Significant algorithms within this category are: Pixel Purity Index (PPI) [Boa94], N-FINDR [Win99], and the Vertex Component Analysis (VCA) [NB05] $[r_1, r_2, \ldots, r_R]$ algorithms.

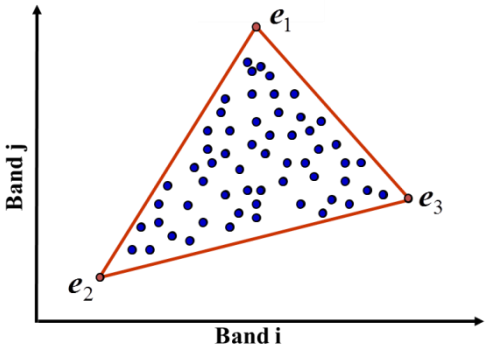


Figure 4.4 Linear unmixing algorithm's model: simplex with three endmembers

4.1.1 Pixel Purity Index (PPI)

Pixel Purity Index (PPI) might use the minimum noise fraction transform [BKG95] as an optative pre-processing step to reduce dimensionality and to improve the signal-to-noise ratio (SNR). The algorithm begins calculating the pixel purity score for each point in the image cube. In order to do this, the algorithm generates a large number of random vectors, named as skewers. Then, every spectral vector (pixel) of the image is projected onto skewers. The next step is to identify and store the extreme points of the projection onto each skewer direction (Figure 4.5), and increment the accumulative counter corresponding to that pixel. The process is repeated many times, and at the end the pixels with highest scores are declared as pure.

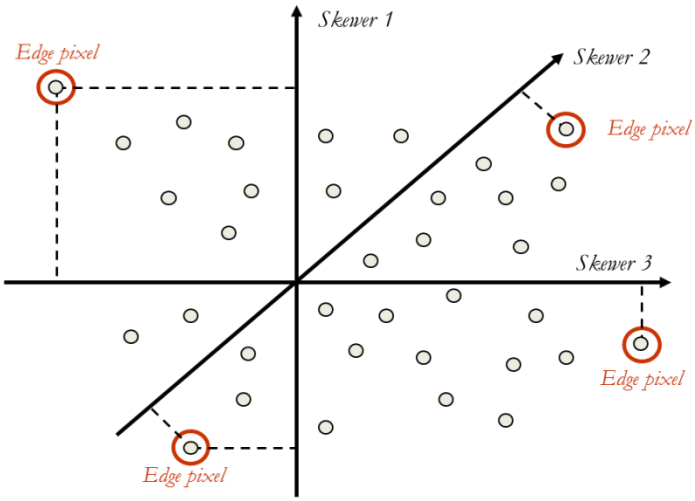


Figure 4.5 PPI model in a two-dimensional space

4.1.1 N–FINDR

This algorithm is based on the fact that in p-1 spectral dimensions, the p-volume defined by a simplex formed by the purest pixels is larger than any other volume defined by any other combination of pixels. Therefore, this method finds the set of pixels that maximizes the volume of the simplex, potentially inscribed within the dataset. In order to refine the initial estimated volume, a trial volume is calculated for every pixel in each endmember position by replacing that endmember and recalculating the volume. If the replacement results in a higher volume, then the pixel replaces the endmember. This procedure is repeated until there are no more endmembers replacements.

4.1.2 Vertex Component Analysis (VCA)

As PPI and N-FINDR algorithms, Vertex Component Analysis also assumes the presence of pure pixels in the data. VCA basically consists in finding the *p* more extreme pixels of the scene. The number of endmembers *p* is known in advance and the spectral matrix $Y = [r_1, r_2, \ldots, r_R]$ has been dimensionally reduced to *p* spectral bands, using a dimensional reduction algorithm; for instance, Principal Component Analysis (PCA) [Jol02].

Generally, VCA iteratively projects data onto a direction orthogonal to the subspace spanned by the endmembers already determined. The new endmember signature corresponds to the extreme of the projection. As a result, VCA models the data using a positive cone (simplex S_p), whose projection onto a properly chose hyperplane is another simplex whose vertices are the final endmembers (Figure 4.6). After projecting the data onto the selected hyperplane, the VCA projects all image pixels to a random direction, and it uses the pixel with the largest projection as the first endmember.

As a first step of the VCA process, the algorithm sets a random endmember e_0. This step initializes a p×p auxiliary matrix which stores the estimated endmember signatures ($E = [e_1, 0, .., 0]^p$ in Figure 4.7 – step 1). The other endmembers are identified in sequence by iteratively projecting the data onto a direction orthogonal to the subspace spanned by the endmembers already determined (given by a vector named *f*). The most extreme pixel in this projection is the new endmember (e_1). Then, the process is repeated until the p endmembers have been extracted.

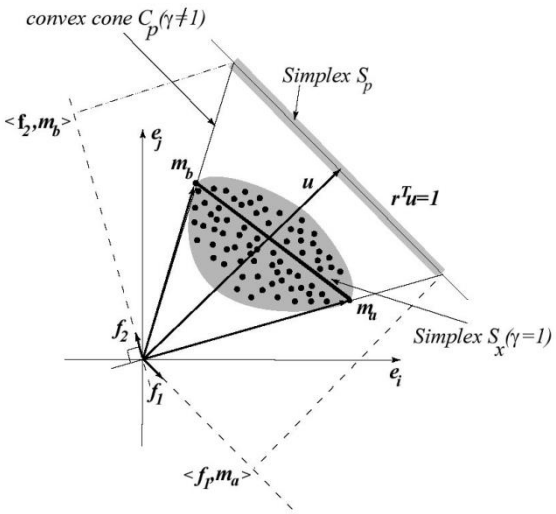


Figure 4.6 VCA algorithm's representation

The whole procedure is described in the commented pseudo-code shown in Figure 4.7.

VCA ALGORITHM COMMENTED PSEUDOCODE

(comments begin with %)

INPUTS: p, Y = [r_1, r_2, ..., r_R] *% Y is composed by R hyperspectral pixels of p frequency bands each*

1: E = [e_0 | 0 | ... | 0 |]; *% e_0 = [0, ..., 0, 1]^T and E is a p×p auxiliary matrix*

2: for i = 1 to p do

3: f = ((I - EE$^+$)w)/ (||(I - EE$^+$)w||); *% f is a random vector orthonormal to the subspace spanned by the columns of E*

4: v = f^{T}Y; *% Y is projected onto the direction indicated by f*

5: index = arg max$_{index = 1, ..., R}$ |v[:,index]|; *% the projection extreme is found*

6: E[:,i] = Y[:,index]; *% endmembers are updated*

7: end for

OUTPUT: E = [e_1, e_2, ..., e_p]

Figure 4.7 VCA pseudo-code

The loop is composed of four main operations (Figure 4.7 – step 2):

Step 3: A vector *f* orthonormal to current endmember is generated (I is the identity matrix, E^+ is the pseudo-inverse matrix of E and w is a *p×1* zero-mean random Gaussian vector).

Step 4: Spectral matrix Y is projected onto vector *f.*

Step 5: Find the new endmember as the extreme of the projection of the matrix Yover *f*.

Step 6: The new endmember is added to the matrix E.

The algorithm finishes processing once all the all endmembers are determined $\left(E = \left[e_1, \ldots, e_p\right]\right)$.

4.1.1 Modified Vertex Component Analysis (MVCA)

Despite the fact that the VCA performs much better than PPI and/or comparable to N-FINDR [NB05]; it is, although effective, a high computational demanding algorithm. Among all the tasks that comprise the VCA algorithm, two of them highlight for being the most intensive in computation. The first one is the pseudo-inverse computation (E^+) in the generation of vector *f,* and the other is the projection of the hyperspectral image onto the direction pointed by the vector f $(v = f^T \cdot Y)$. As an alternative, but with lower computational complexity, the MVCA algorithm is able to reproduce the results provided by the VCA. Furthermore, the MVCA algorithm is able to obtain the same levels of performance with simpler operations and a negligible amount of flops when compared

to the VCA algorithm [LHC+12a]. These improvements come by introducing certain modifications into the processes by which vector f is calculated and by which the hyperspectral image is projected onto the direction pointed by vector *f.* Regarding the first one (the calculation of vector *f*), three modifications are introduced:

1. The norm of vector f is not forced to be equal to the unity. Consequently, the operations performed in the VCA algorithm in order to normalize vector f are skipped in the proposed MVCA algorithm. This modification does not change the endmembers obtained by the VCA algorithm, since a different norm varies the values contained in matrix v (computed at Figure 4.7 - step 4), but it does not alter the position (given by the variable *index* calculated at step 5) that indicates the projection extreme within this matrix.
2. As far as the underlying reason for generating a random w vector is only to get a not null projection in the first iteration, MVCA fixes the vector w to $[1_1, 1_2, \dots, 1_p]$. This light modification allows to reduce the computational cost of the MVCA algorithm and to avoid the need of a hardware-based random number generator.
3. The third modification introduced into the vector f calculation is the most important one, in terms of computational cost savings and ease of implementation. This modification is based on changing the mechanism adopted in the VCA algorithm for the calculation of a vector orthogonal to the subspace spanned by the endmembers that have been already determined. In particular, f is computed by first obtaining an orthogonal set of i vectors, $U = [u_1, u_2, \dots, u_i]$, from the set $E = [e_1, e_2, \dots, e_i]$ defined by the i endmembers that have been already computed. This is achieved by applying the Gram–Schmidt orthogonalization algorithm, which guarantees the set U spans the same i -dimensional subspace as E:

$$u_k = e_k - \sum_{j=1}^{j=k-1} proj(e_k, u_j);\ \{k = 1, \dots, i\ and\ u_1 = e_1\} \quad (2)$$

where *proj* stands for the projection operator, defined as (3):

$$proj(e_k, u_j) = \frac{<e_k, u_j>}{<u_j, u_j>} u_j; where < x, y > = \sum_{z=1}^{z=p} x_z \cdot y_z \qquad (3)$$

As far as *U* spans the same *i*-dimensional subspace of *E*, an additional vector u_{i+1} is also orthogonal to all the vectors included in *E* and *U*, avoiding the computation of the pseudo-inverse of matrix *E.* Vector u_{i+1} is computed by following the procedure stated at equation (4):

$$f = w - \sum_{j=1}^{i} proj(w, u_j) \qquad (4)$$

Once *f* has been computed, the hyperspectral image Y must be projected onto the direction indicated by this vector. In order to further reduce the computational complexity of the endmember extraction process, this projection is performed in the MVCA algorithm using integer rather than floating point arithmetic. This criterion is based on the idea that this modification should not alter the position of the projection extreme (although the value of the projection itself will definitively change). Moreover, as it will be demonstrated in this paper, the adoption of integer arithmetic for computing the indicated orthogonal projection leads to faster and less resource demanding designs.

Regarding the MVCA pseudo-code represented in Figure 4.8 is important to highlight two aspects related to its main loop (step 6). In the beginning of the first iteration (i = 1), no endmembers have been already computed and therefore, the first column of *U* is initialized to e_0 at step 7 of the MVCA algorithm. In the second iteration, the first endmember has been already calculated and the next column of *U* is initialized with this endmember (the second columns of *E* and *U* contain the first *valid* endmember and the first *valid* component of the U set respectively). As a result, the projections indicated in equation (2) should only be computed (steps 8-11) from the third to the *p*-th iteration of the main loop. In addition, it is also remarkable the inclusion of the *proj_acc* vector in order to reuse previously computed projections in the calculation of vector *f* (equation (4)). In this sense, and as far as *U* does not contain any *valid* vector in the first iteration, *proj_acc* is reinitialized at step 15 during the first iteration.

MVCA ALGORITHM COMMENTED PSEUDOCODE

(comments begin with %)

INPUTS: p, Y = [r_1, r_2, ..., r_R]; *% Y is composed by R hyperspectral pixels of p frequency bands each*

1: E = [e_0 | 0 | ... | 0 |]; *% $e_0 = [0, \ldots, 0, 1]^T$ and E is a p×(p+1) auxiliary matrix*

2: U = [0 | 0 | ... | 0 |]; *% U is a p×p auxiliary matrix*

3: w = [1, ..., 1]; *% w is a p×1 vector*

4: proj_acc = [0, ..., 0]; *% facc is a p×1 vector used for saving operations when computing f*

5: Y_{int} = round2int (Y); *% Y_{int} is the integer version of Y where the minimum integer value is equal to 1*

6: for i = 1 to p do *% main loop*

7: U[:,i] = E[:,i]; *% U is initialized with the endmember computed in the last iteration*

8: for j = 3 to i do *% the computation of U is completed within this loop*

9: $\mathbf{proj}(\mathbf{e_k}, \mathbf{u_{j-1}}) = \frac{\mathbf{E[:,i]^T U[:,j-1]}}{\mathbf{U[:,j-1]^T U[:,j-1]}} \mathbf{U[:,j-1]}$**;** *% the projection is computed according to* $proj(e_k, u_j) = \frac{<e_k, u_j>}{<u_j, u_j>}; where < x, y >= \sum_{z=1}^{p} x_z \cdot y_z$

10: $\boldsymbol{U[:,i] = U[:,i] - proj(ek, uj-1)}$**;** *% the i-th column of U is updated*

11: ***end for j*** *% The computation of U according to equation* $u_k = e_k - \sum_{j=1}^{k-1} proj(e_k, u_j); \{k = 1 \; to \; i \; and \; u_1 = e_1\}$ *is finished for the current iteration of the main loop*

12: $\mathbf{proj}(\mathbf{w}, \mathbf{u_i}) = \frac{\mathbf{w^T U[:,i]}}{\mathbf{U[:,i]^T U[:,i]}} \mathbf{U[:,i]}$**;** *% the projection is computed*

13: proj_acc = proj_acc + proj(w,u_i); *% the projection is saved for the next iterations at the accumulator*

14: f = w – proj_acc; *% f is a vector orthogonal to the subspace spanned by the columns of E that is computed according to equation* $f = w - \sum_{l=1}^{i} proj(w, u_l)$

15: if (i == 1) then proj_acc = [0, ..., 0]; *% reset of the accumulator for the first iteration*

16: v = (round2int(f^T))Y_{int}; *% Y_{int} is projected onto the direction indicated by f_{int}*

17: index = arg $\max_{index = 1, \ldots, R}$ |v[:;index]|; *% the projection extreme is found*

18: E[:,i+1] = Y[:,index]; *% endmembers are updated*

19: end for i

OUTPUT: E = [e_1, e_2, ..., e_p];

Figure 4.8 MVCA pseudo-code

4.2 FPGA-BASED MVCA SCALABLE ARCHITECTURES

This section describes the two scalable architectures proposed for the implementation of the MVCA algorithm on an FPGA. Both are based on the previously developed reference architecture [LHC+12a]. Due to the fact that these three architectures share most of their kernels, a general view of the architecture modules is first introduced. Then, the details of each computing kernel are disclosed, highlighting the differences between the two scalable architectures and the reference one.

4.4.1 Reference architecture

The original architecture in this work, which computes the endmembers of a hyperspectral image according to the MVCA algorithm, is outlined in Figure 4.9. As it is seen from this figure, the MVCA architecture distinguishes two kinds of kernels or modules: three of them dedicated to computing tasks, and two extra modules devoted to format conversion operations. The computing kernels

are named U_GENERATOR (this module includes a sub-module named PROJECTIONS), F_GENERATOR, and IMAGE PROJECTION. On the other hand, the format conversion modules are named `int2fp` and `shift_exp`.

The architecture also incorporates an input memory where the hyperspectral pixels to be processed are stored as 32-bit integer values. This double-port memory (one read port and one write port) has been implemented by means of the internal embedded block RAM resources present in Xilinx FPGAs. In particular, we have taken advantage of the Xilinx Core generator tool [XCOR] that generates and delivers parameterizable cores optimized for Xilinx FPGAs. Actually, we have made use the Xilinx LogiCORE IP Block Memory Generator [XMEM], included in the Xilinx Core generator tool. This Memory IP uses embedded block memory primitives in Xilinx FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. Because of the MVCA algorithm operates at the projection stage with integer versions of the single-precision floating point numbers, contained in the originally captured hyperspectral image, we will assume compliant with the IEEE 754-2008 standard [IEE08].

Above every computing module, in Figure 4.9, a set of numbers has been included. These numbers corresponds to the numbered steps indicated in the MVCA pseudo-code (Figure 8). According to this information, the U_GENERATOR module computes the orthogonal set of i vectors, $U = \{u_1, u_2, \dots, u_i\}$, from the set $E = \{e_1, e_2, \dots, e_i\}$ defined by the i endmembers that have been already computed. The F_GENERATOR module computes vector f from the set $U = \{u_1, u_2, \dots, u_i\}$, whereas the IMAGE PROJECTION module projects the integer version of the hyperspectral image stored in the input memory onto the direction indicated by the vector *f*. In this way, the *index* signal determines the position that gives the maximum projection.

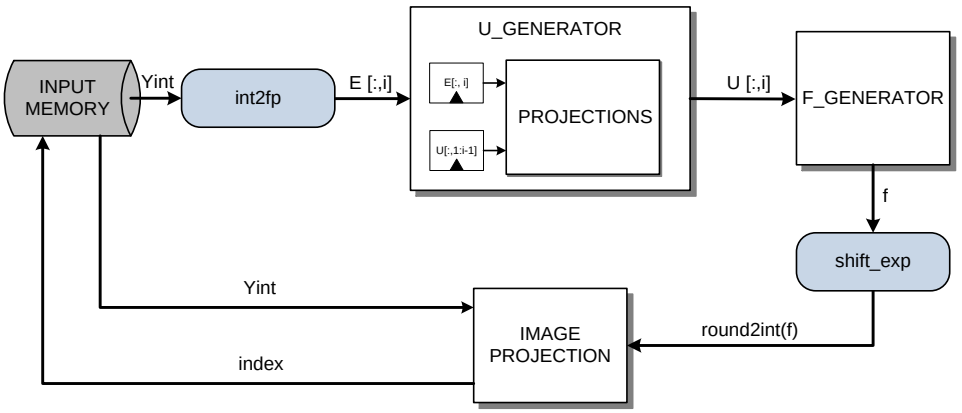


Figure 4.9 General view of the Reference MVCA architecture

Finally, the format conversion modules (`int2fp` and `shift_exp`) have been added to the architecture in order to assure two issues. First of all, the computation of the vector *f* is performed by using floating point arithmetic (`int2fp`); and secondly, the projection of the image onto the direction pointed by this vector is performed using integer arithmetic (`shift_exp`), as it is demanded by the MVCA algorithm. At this point, it has to be mentioned that as far as the reflectance values of hyperspectral images are normally between 0 and 1, a simple rounding mechanism to the nearest integer would not be efficient at all, since the converted values would be 0 or 1. Due to this reason, the floating to integer conversion performed by the `shift_exp` module must be accomplished in two steps. In a first step the exponents of all the floating point numbers to be converted are shifted in a way such that the minimum exponent becomes equal or greater than 127. This process gives back a floating point number with an absolute value equal or greater than 1 (for practical proof, interested readers are referred to [XFPO]). In the next step, and the last one, the previously floating point values are converted into integers.

As a last remark, it is worth to mention that, for the sake of clarity, we have decided to include only the main data signals between modules, skipping the multiple control signals that govern the global behavior of the architecture.

4.4.1.1 The U_GENERATOR module

This module is in charge of calculating the set of i vectors, $U = \{u_1, u_2, \dots, u_i\}$, from the set $E = \{e_1, e_2, \dots, e_i\}$ defined by the endmembers that have been already computed, i.e. steps 7 to 11 in the pseudo-code of the MVCA algorithm (Figure 4.8). In order to accomplish this task, U_GENERATOR counts with a sub-module named PROJECTIONS. This sub-module efficiently computes the coefficient $coeff\_proj(i, j-1) = \frac{\mathrm{E[:,i]^T U[:,j-1]}}{\mathrm{U[:,j-1]^T U[:,j-1]}}$ in every loop iteration placed at step 8.

This calculation is carried out in a double-loop manner, where the inner loop is repeated *p* times in order to compute the previously mentioned term, and the outer loop corresponds with the loop indicated at step 8. In particular, the PROJECTIONS sub-module has one register and one circular buffer where the endmember that was computed during the last iteration of the i loop placed at step 6 and the $(i-1)$ vectors already computed from the set *U* are stored, respectively. Every clock cycle, the data in the register is right-shifted while the data in the buffer is cyclically permuted. This procedure allows obtaining the term $E[:, i]^T \; U = [:, j-1]$ in *p* clock cycles by means of a multiply-and-accumulate hardware structure whose inputs are the values in the most right positions of the aforementioned register and circular buffer. After *p* clock cycles, $coeff\_proj(i, j-1)$ is obtained by dividing the accumulated result by $U = [:, j-1]^T \; U[:, j-1]$. This is exemplified in Figure 4.10, where the hardware core of the PROJECTIONS sub-module is depicted, showing the data movements in both the register and the circular buffer for different values of i, j and *iter*, being the last index controlling the number of the inner loop iterations. As it observed from Figure 4.10.a and Figure 4.10.b, the terms $E[:, i]^T \; U = [:, j-1]$ and $U = [:, j-1]^T \; U[:, j-1]$. are progressively computed with the increments of *iter* (clock cycles), being the results stored in the accumulative registers named acc_EU and acc_U, respectively. In order to guarantee the correctness of the results, both accumulative registers must be initialized to zero at the start of a new iteration of the outer loop, i.e. when the value of j changes. Once *iter* = p (Figure 4.10.c), the coefficient $coeff\_proj(i, j-1)$ is

readily calculated by dividing the content of acc_EU by the content of acc_U. The result of this division is stored in a register that is enabled only when the results accumulated at acc_EU and acc_U are definitive. i.e., when *iter* = p.

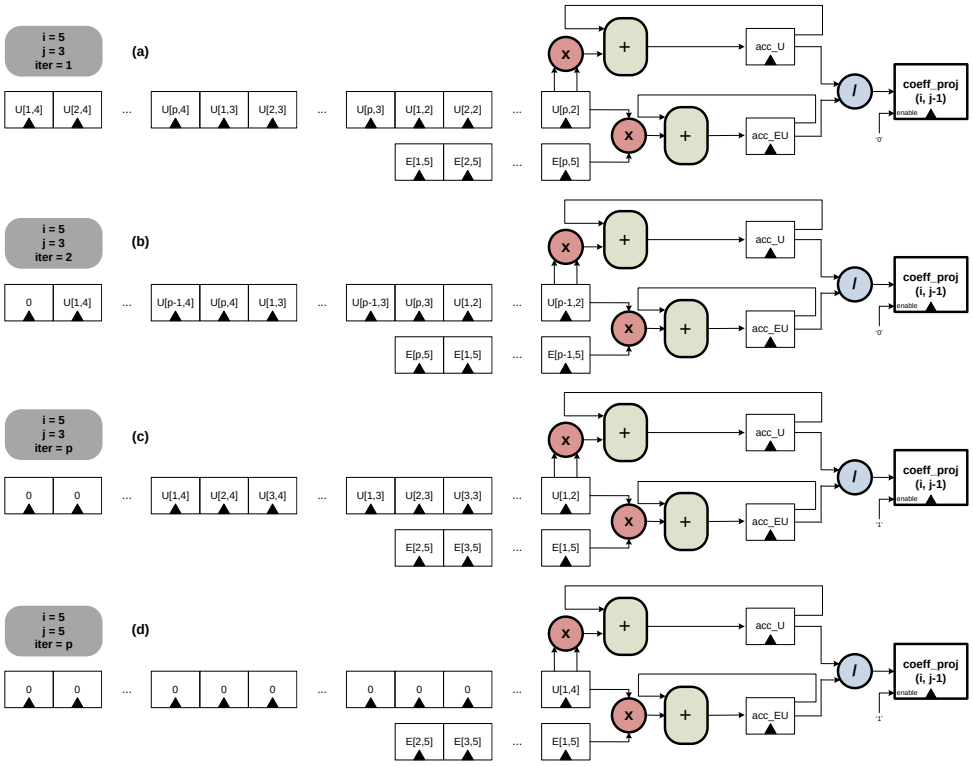


Figure 4.10 General view of the PROJECTIONS architecture

Once a coefficient $coeff\_proj(i, j-1)$ has been computed by the PROJECTIONS sub-module, the U_GENERATOR module has to update $U[:, i]$ according to the equation placed at step 10 of the MVCA pseudo-code. Finally, when the loop initiated at step 8 of the MVCA pseudo-code ends, the U_GENERATOR module must store the definitive $U[:, i]$ in order to be used it in the following iteration of the main MVCA loop (step 6 of the pseudo-code and Figure 4.10.d).

4.4.1.2 The F_GENERATOR module

This module is the responsible of computing the vector named f, which is orthogonal to the subspace spanned by the endmembers already computed. In

order to reduce the computational cost associated to this process, *f* is computed according to steps 12 to 15 in the MVCA pseudo-code, which wisely prevent from re-computing the whole summation expressed at equation (4) for each value of i. This is thanks to the inclusion of the $proj\_acc$ vector, which allows reutilizing previously computed projections in the calculation of vector *f*.

The architecture proposed for this module has been summarized at Figure 4.11, where again for seeking of clarity; we have not included any of the control signals that command the module like the enable signals of the different registers, or the reset signal of the $proj\_acc$ register, just to name the most important ones.

The computation of the coefficient $coeff\_proj(w,i) = \frac{w^T \cdot U[:,i]}{U[:,i]^T \cdot U[:,i]}$ has been outlined in the circuit at the top part of Figure 9, which allows its calculation in *p* clock cycles by circularly displacing the data in the register U with each clock cycle. At this point, it is important to highlight that by fixing the vector w to *[1,1,…,1]* ^{T} in the MVCA algorithm, we are not only bypassing the generation of a *p*×1 random vector for the computation of each of the *p* endmembers to be extracted from the target image, but we are also avoiding the use of a floating-point multiplier for computing $w^T \cdot U[:,i]$, as far as this expression is equivalent to the summation of the p components of $U[:,i]$ for the case of this work ($w = [1,1,\dots,1]^T$). Once $coeff\_proj(w,i)$ has been computed, the estimation of the vector *f* becomes straightforward with the rest of the hardware included in the architecture of the F_GENERATOR module depicted at Figure 4.11, considering that the data at $proj\_acc$ and *f* registers also experiment circular and synchronized displacements with each clock cycle.

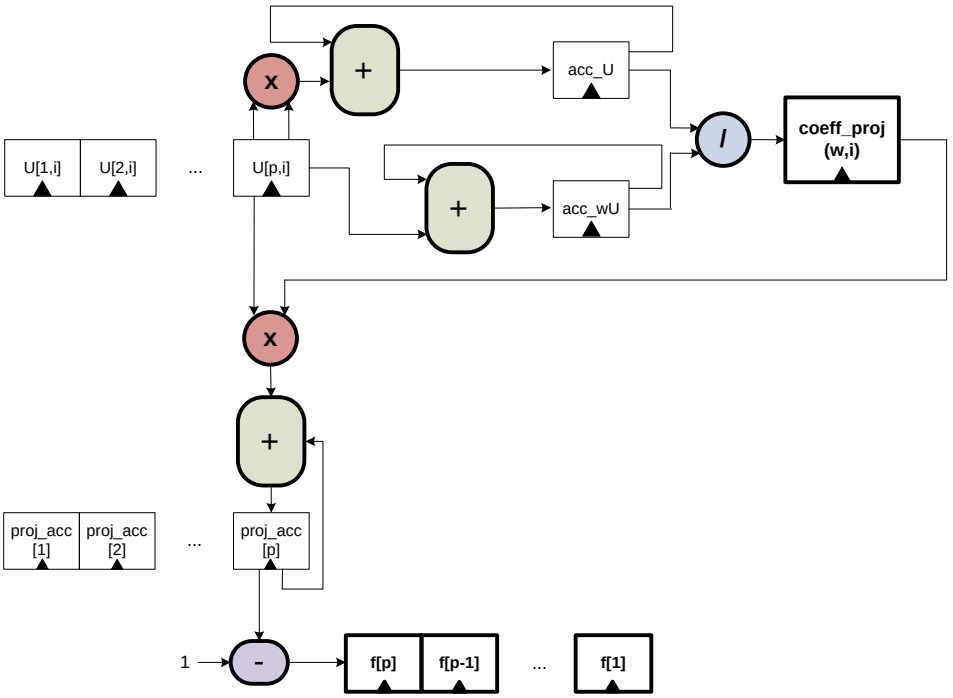


Figure 4.11 General view of the F_GENERATOR

4.4.1.3 The IMAGE PROJECTION module

This module projects the hyperspectral image *Y* onto the direction pointed by the vector *f*. As the spatial and spectral dimensions of the input hyperspectral image are usually large, the IMAGE PROJECTION module has been designed in order to compute the aforesaid projection with a high degree of parallelism. More specifically, as it is summarized in Figure 4.12, the spectral bands of each pixel in the hyperspectral image are concurrently multiplied by the components of the vector *f*. Then, the results obtained feed a parallel adder tree that computes the projection in $NADD = \lceil \log_2 p \rceil$ adding stages, where $\lceil\ \rceil$ represents the ceil function. For ease of design, the first adding stage (the one on top of Figure 4.12) is forced to have a number of adders equal $2^{(NADD-1)}$, being the unused adders in this stage filled with zeros. The rest of adding stages have a number of adders equal to half of the adders of its predecessor stage.

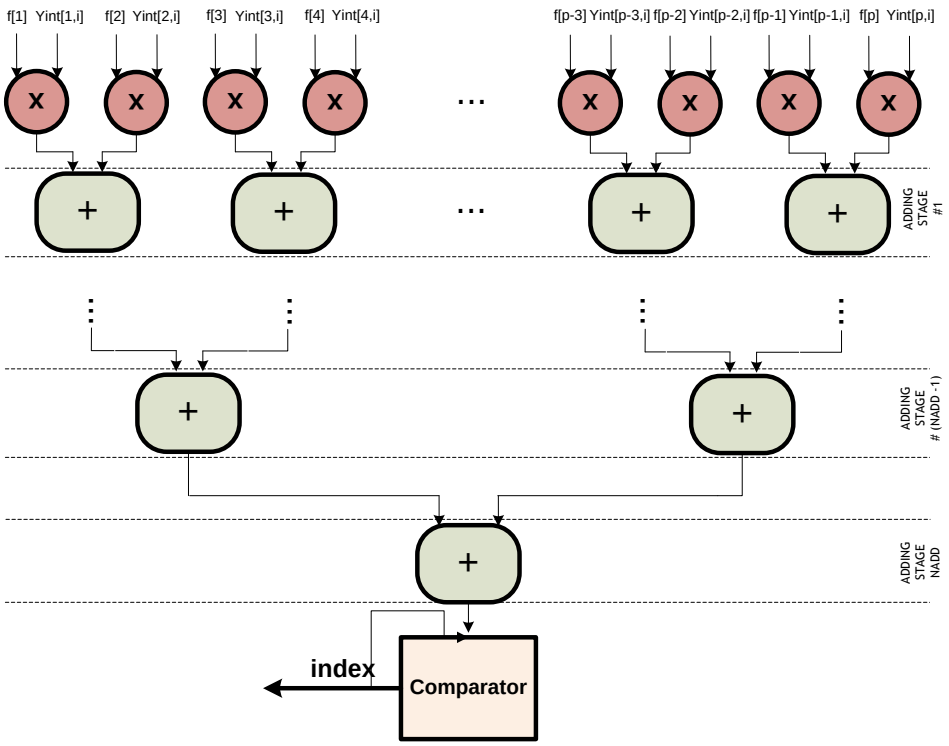


Figure 4.12 IMAGE PROJECTION overview

Once a projection is computed, the comparator allocated at the bottom of Figure 4.12 determines whether the computed projection is bigger than the maximum value provisionally stored (initially set to zero), updating in this case the maximum and its position (*index*). Moreover, the proposed IMAGE PROJECTION module works in a highly pipelined fashion, which means that with each clock cycle, a new hyperspectral pixel is loaded and processed, being all the projections calculated on a cycle-by-cycle basis.

4.4.2 Scalable IMAGE PROJECTION module

The MVCA architecture performs very well in terms of area and frequency achieved, being capable of real time processing of hyperspectral images [LHC+12b]. However, as the number of endmembers (p) is different for different images, a specific implementation for each sensor is required. This is due to the fact that the IMAGE PROJECTION module, in charge of performing a multiplication

between the hyperspectral image Y_{int}, and the vector *f*, requires *p* multipliers (see Figure 4.12). Moreover, as the number of endmembers is relatively high in some images, the area occupied by this module might do unfeasible an implementation over an FPGA.

In this work, a scalable solution for the IMAGE PROJECTION module is presented. Furthermore, different scenarios for scalability of this module are studied and based on that, two alternatives for scaling the IMAGE PROJECTION module computation are proposed. As it was mentioned before, the IMAGE PROJECTION module consists on the iterative calculation of the multiplication between the transposed vector f and the hyperspectral pixel matrix Y_{int}. Then, among the resultant matrix (C) an endmember is extracted (equation 5) as the highest value.

$$[f_1 \quad f_2 \dots \quad f_p] \times \begin{bmatrix} y_{11} & y_{12} \dots & y_{1R} \\ y_{21} & y_{22} \dots & y_{2R} \\ y_{p1} & y_{p2} \dots & y_{pR} \end{bmatrix} = [C_1 \quad C_2 \dots \quad C_R] \qquad (5)$$

$$C_k = \sum_{i=1}^{i=p} f_i \times y_{ij} \,; \{j = 1..R\} \qquad (6)$$

In equation (6), y_{ij} represents the spectral band *i* of the pixel *j*. In this case, the hyperspectral image contains *R* pixels of *p* bands, since it has been spectrally reduced. The matrix multiplication opens the window to explore scalability in two different ways.

1. On the one hand, it might be possible to process several spectral components of the same pixel at the same time (vector *f* by one column of Y_{int}). Hence, it is possible to scale how fast C_k (equation 6) is calculated, by means of adjusting the number of $f_i \times y_{ij}$ operations performed in parallel. For instance, assuming that *p* is equal to 6, it would be possible to use two multipliers and use them repeatedly (three times) until get the C_k value. As a consequence, by following this methodology, the corresponding $C_1, C_2, .. C_R$ values are processed sequentially, one after another.
2. Another alternative might be to process several spectral components of different pixels at the same time. The difference of this procedure,

with respect the previous one, lies on the fact that in this case different $C_k$values might be processed, band by band, by different multipliers.

Both approaches are flexible, since they might permit several levels of scalability, in terms of allowing the adjustment of the number of computations that are performed in parallel. In order to unify the solutions as much as possible, and establish a fair comparison between both alternatives, their processing core is the same, named processing element (PE). The main structure of this PE consists in a multiplier and an accumulator, as Figure 4.13.a shows. The selection of these elements lies on two reasons. The first one is that they are the minimum number of elements required in order to execute a matrix multiplication, but also because this structure is general enough for implementing any one of aforementioned scalable alternatives. Moreover, this PE might be scaled by means of increasing or decreasing its number, according to the level of parallelization demanded in the image projection module, without a numerical impact on the final result. Such as Figure 4.13.b highlights, there exist similitudes between the proposed PE and the structure of the IMAGE PROJECTION module presented into the previous subsection.

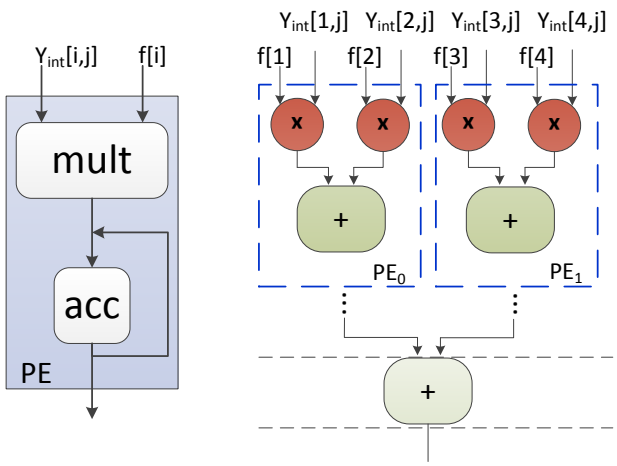


Figure 4.13 Processing Element (PE); a) Main structure of a PE; b) Equivalence between the PE and the reference architecture

4.4.2.1 Scaling the number of spectral components of a pixel simultaneously processed (SpectSA_MVCA)

The original hardware architecture (section 1.4.1.3) obligates to compute all C_k coefficients simultaneously. Hence, as the dimension of the vector f is equal to the number of endmembers (p), p multipliers are required (see Figure 4.12). The main goal behind this scalable design, for now on referred as SpectSA_MVCA, is to offer a more flexible architectural solution, compared to the previous one. In this sense, this proposal offers the freedom of adjusting the number of PEs dedicated to perform $f_i \times y_{ij}$ operations. However, due to the strategy followed by processing every pixel, in which all the PEs compute the same pixel, although different spectral components, the scalability level of this design is limited by p. Consequently, the number of PEs might vary between 1 and p. It is worthy to mention that the design demands a comparator as the last stage of the computation process. The comparator is responsible for calculating the index (memory address) of the maximum projection obtained during the matrix multiplication, once all the pixels have been processed.

Figure 4.14 represents various levels of scalability, from one PE to four. Here the number of PEs is scaled in order to increase the number of operations performed in parallel. As it might be appreciated in Figure 4.14.a, in the case of using only one PE, the internal accumulator of the PE provides the coefficient C_k by itself. However, when two PEs are enabled, the result of the C_k coefficient requires the sum of the partial results calculated by each PE_1 and PE_2 (Figure 4.14.b). The process is even more complex when the number of PEs is higher than 2 (Figure 4.14.c), since the number of adders must be increased in order to combine all the partial sums of every couple of PEs, until complete the calculation of the C_k coefficient.

Independently of the level of the scalability, the overall behavior of the whole IMAGE PROJECTION kernel cannot vary. Therefore, the final design has to be flexible enough for fulfilling with this specification. Considering the requirements of all the configurations depicted on Figure 4.14, it is easy to identify three main tasks. One dedicated to compute values (related to PEs), another

one in charge of obtaining the final value of C_k (related to adders), and the last one responsible for calculating the index of the endmember, after the matrix multiplication (related to the comparator).

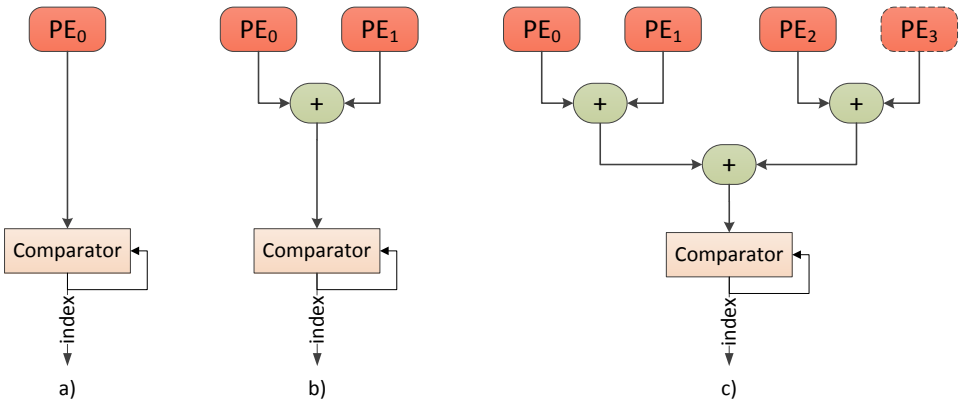


Figure 4.14 Resources of the SpectSA_MVCA according to the scalability level; a) 1 PE; b) 2 PEs; c) 4 PEs

Thus, the differentiation of these tasks allows to describe a modular design, with three elements: a control unit, a scalable PE unit and an adder-tree unit, as it has been outlined in Figure 4.15. The first module is responsible for managing the data (loading and storing data from memory to the rest of the system), synchronizing the PEs, and calculating the output of the IMAGE PROJECTION. The scalable PEs module is the core of the computation. The number of the available PEs should not necessarily be constant; it might vary depending on the desired scalability level (for instance depending on the dimension of the hyperspectral image to be processed, or the time constraints demanded by the system). Finally, an adder-tree is required for collecting and adding all the partial results performed by the PEs.

When the number of PEs increases (it is higher than one), all of them collaborate in order to obtain the projection of one pixel (performing a row by a column multiplication (equation 6)).

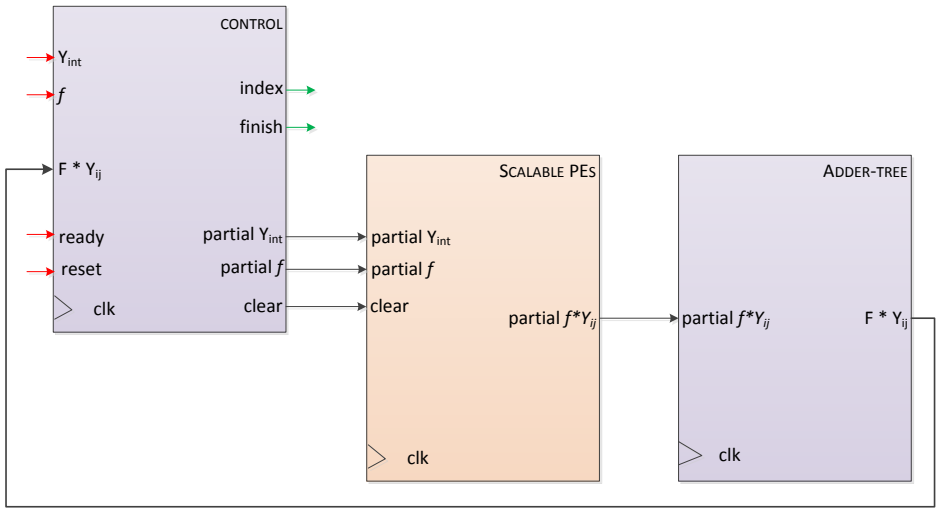


Figure 4.15 Modular structure for processing several components of one pixel

The most effective way of implementing this behavioral strategy, in terms of execution time and complexity, is to synchronize the execution of all the PEs in order to ensure that all of them start processing, and get results at the same time. In the case that the number of PEs is a multiple of the number of spectral bands (*p*) the synchronization and the data loading, from the control unit to the scalable PEs, do not present any complication. Otherwise, during the computation of the last spectral bands of a pixel, some PEs will remain idle (without processing data) until a new pixel is loaded. For example, in case of *p=5* and the level of scalability is two (PEs=2; PE_0 and PE_1), the pixel will be processed in three steps, and then another step will sum their results.

Step 1: $PE_0 = f_1 \times y_{11} = p_{01}$ and $PE_1 = f_2 \times y_{21} = p_{11}$;

Step 2: $PE_0 = p_{01} + f_3 \times y_{31} = p_{02}$ and $PE_1 = p_{11} + f_4 \times y_{41} = p_{12}$;

Step 3: $PE_0 = p_{02} + f_4 \times y_{41} = p_{03}$ and $PE_1 = p_{12}$;

Step 4: $C_1 = p_{03} + p_{12}$;

Step 5: $index_{max} = \max\{index_{max}, C_1\}$;

Step 6: repeat the process from step 1;

Under these conditions, during the last step the result of PE_1 will be a data full of zeros. Although is not an optimum solution, it is based on two reasons. First, when there is more than one PE, all the adders involved in the result generation needs a data input, since they do not contain any control logic. Second, this is the easiest way to keep the system synchronization.

Collecting all the aforementioned considerations (the modularity of the design, the regularity of the system behavior independently from the level of scalability, the flexibility for processing using different number of PEs, and the synchronization issues) the schematic of the hardware SpectSA_MVCA design is represented in Figure 4.16.

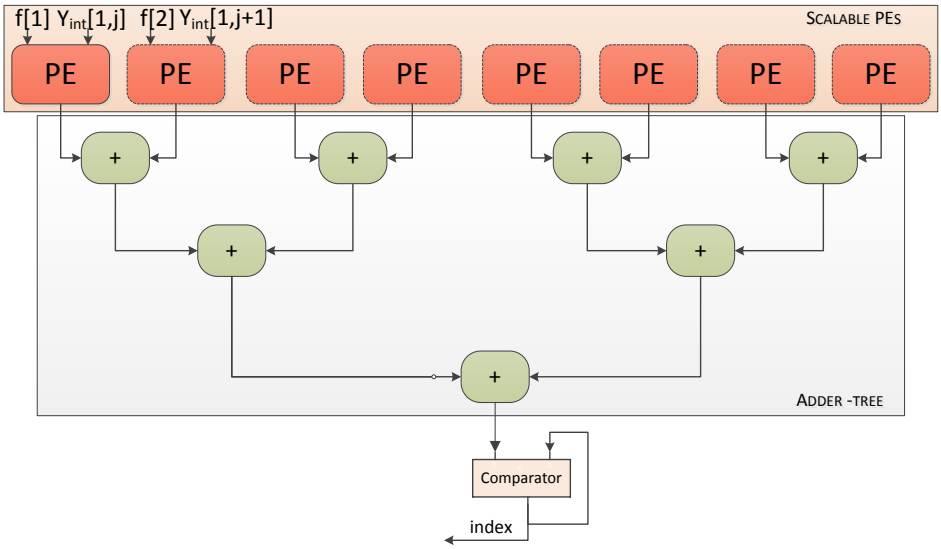


Figure 4.16 Scalable IMAGE PROJECTION design for processing several components of the same pixel in parallel

4.4.2.2 Scaling the number of pixels simultaneously processed (PixelSA_MVCA)

As an alternative to the IMAGE PROJECTION approaches presented before, this section presents PixelSA_MVCA as another alternative for exploiting the benefits of the scalability. On the contrary to the architecture presented before (SpectSA_MVCA), where each C_k is processed sequentially (equation 6), now the solution consists on computing more than one C_k coefficients at the same

time. In other words, performing several rows by columns operations simultaneously ($f_1 \times y_{11}, f_1 \times y_{12} \ldots f_1 \times y_{1p}$). The easiest way to implement this approach is shown in Figure 4.17, in which every PE should be responsible for a specific C_K coefficient. Therefore, each PE has to iterate p times until all the components of a pixel have been processed. Under this scenario, every PE would produce a result at the same time. Then, a comparator tree would be needed in order to identify the maximum value within the set of results provided by the PEs, such as Figure 4.17 depicts. The addition of this comparator tree makes the solution more complex, due to the extra logic, the management and the synchronization issues.

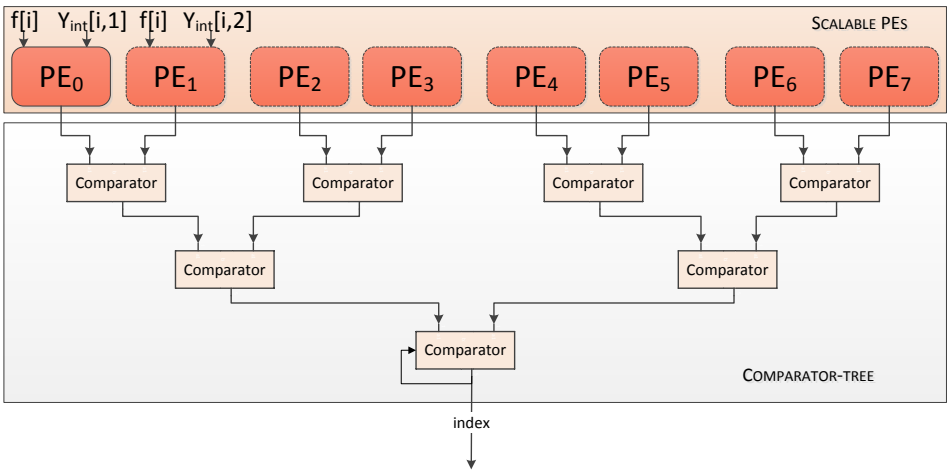


Figure 4.17 Scalable IMAGE PROJECTION design for processing several pixels simultaneously

The way to eliminate the comparator tree from the design would require introduce certain mechanism in order to generate only one C_k coefficient per clock cycle. If this would be possible, then the comparator tree would be reduced to only one comparator, like in the previous IMAGE PROJECTION designs. The cost of implementing this choice goes through adding more complexity to the synchronization process among the PEs. Such as Figure 4.18 shows, the schematic of the `PixelSA_MVCA` design, all the data inputs includes $PE_{i-1}(i = 1..p)$ delays (represented by Δ symbol); being PE_i the position of the PE in the system. As it can be noticed, every PE_i is delayed one clock cycle compared to the previous PE_{i-1} unit. This fact implies that, during the initial PE_{i-1} clock

cycles, not all the PEs are processing data. Once all of them are processing data, all the PEs keep working until the last data is completed.

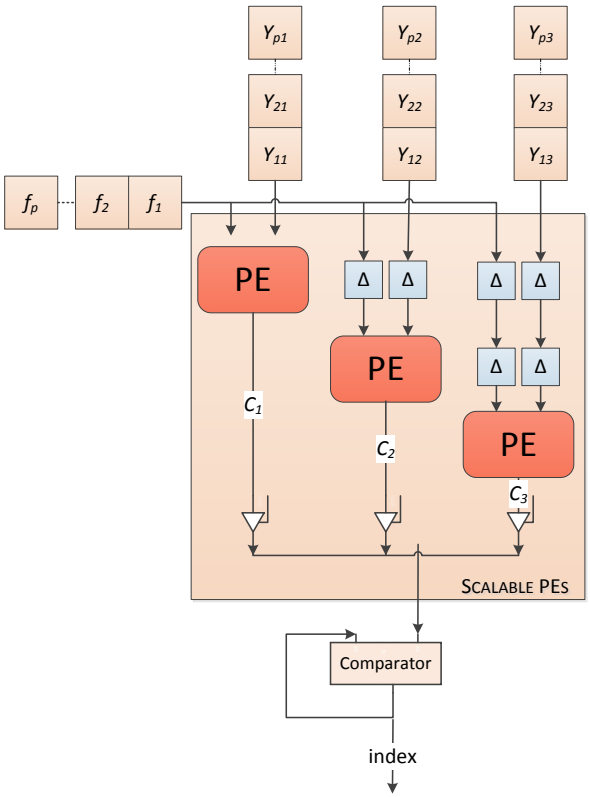


Figure 4.18 Scalable IMAGE PROJECTION design for processing several pixels in parallel without a comparator tree

Regarding the scalability level, this hardware design allows varying the number of PEs between one and *p*. In spite of this fact, it would be possible to increase the scalability higher than *p*, but at cost of introducing a comparator tree in the system. In this sense, this design would have the same drawbacks of the one depicted in Figure 4.16.

As for the hardware implementation of the PixelSA_MVCA, the design respects the same precepts than the SpectSA_MVCA: modularity, scalability, regularity, and distributed control. This approach and the block diagram is represented in Figure 4.19. The comparator but also the inputs delays to the PEs are managed by the control block.

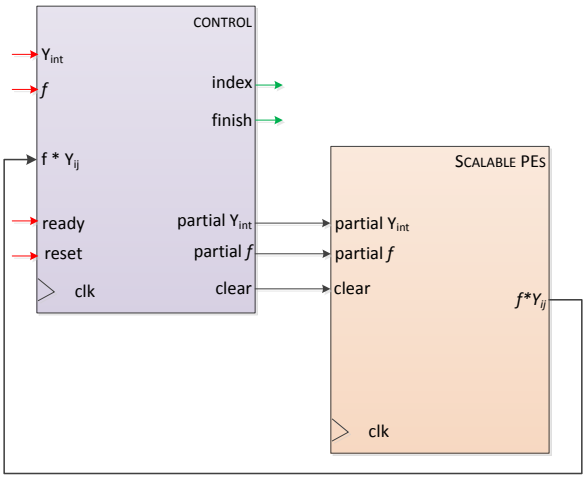


Figure 4.19 Modular structure for processing several pixels at the same time

4.5 COMPARISONS AND RESULTS

The architectures outlined in the last section have been described in Verilog Hardware Description Language (HDL) prior to the implementation and verification onto a XCVSX95T FPGA, belonging to the Virtex-5 family from Xilinx. The Verilog codes can be easily configured in order to process hyperspectral images of any spatial size and/or number of spectral bands, and also can be tailored to adjust the number of processing elements. This section shows the synthesis results for all the three architectures detailed previously, and their performance when they are implemented onto the Virtex-5 FPGA.

4.5.1 Endmember extraction accuracy

Before showing the implementation results, it is necessary to assess the new solution in terms of the endmember extraction accuracy. For this reason, the proposed FPGA implementation of the MVCA algorithm has been validated by using 40 artificially generated hyperspectral images (synthetic images) as well as one real hyperspectral scene captured by the Jet Propulsion Laboratory (JPL) NASA's AVIRIS sensor [GRE+98].

The artificial hyperspectral images used in this work were generated with the `demo_vca` software tool available at [VCAA], which allows creating a hyperspectral image of a spatial size defined by the user from p spectral signatures

selected from the USGS digital spectral library [USGS] that are mixed according to abundance fractions generated with a properly tuned Dirichlet distribution. In addition, a certain amount of Gaussian noise can be added so that the generated image has a signal-to-noise ratio (SNR) value also defined by the user. In particular, we have generated 40 images of 36×36 pixels each with five different values of SNR (10 dB, 20 dB, 30 dB, 40 dB, and 50 dB) and eight different number of endmembers, *p* (3 to 10, both inclusive).

In addition, the well mineralogical understood Cuprite scene [NAC], which has been widely used to validate the accuracy of endmember extraction algorithms, has been also taken into account in this work. This scene was captured by NASA's AVIRIS sensor over the Cuprite mining district in Nevada. Particularly, we have used a 250×191-pixel subset available online in reflectance units after atmospheric correction which comprises 224 spectral bands between 0.4 and 2.5 μm. Prior to the analysis, different bands have been removed due to water absorption and low SNR, leaving a total of 188 reflectance channels to be used in our tests. The number of endmembers present in a real hyperspectral scene like Cuprite is unknown a priori. As far as this number is an input of the MVCA algorithm, it should be calculated prior to the unmixing step. In order to determine it, the Virtual Dimensionality (VD) has been estimated by inspecting the eigenvalues of the sample covariance matrix and the sample correlation matrix. More exactly, the VD was estimated by the Noise Whitened Harsanyi–Farrand–Chang (NWHFC) eigenthresholding method [CD04] using the Neyman–Pearson test with the false-alarm probability set to 10^{-5}, resulting in a total number of 14 different pure materials within the *Cuprite* sub-image already mentioned. This image is depicted in Figure 4.20.

For all these sequences, the results obtained with the MATLAB code of the MVCA algorithm have been compared with the ones obtained by means of the proposed MVCA architecture when mapped onto a Virtex-5 XC5VSX95T FPGA. In all the cases, a perfect match between both results has been obtained which guarantees that the FPGA implementation is functionally equivalent to the original MATLAB MVCA code.

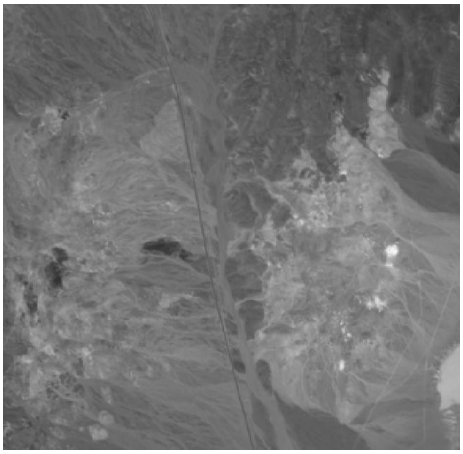


Figure 4.20 Cuprite image

Just for illustrative purposes, and as far as the MVCA is based on VCA algorithm [VCAA], Figure 4.21 shows the maximum and the minimum spectral angles obtained after comparing the endmembers extracted in Cuprite by the VCA algorithm (running on a desktop personal computer) and by the MVCA architecture (running onto the aforementioned FPGA device) against the USGS library [USGS] spectra of the 14 minerals reported in the original VCA paper [LHC+12b].

For obtaining these results, both the VCA software implementation and the MVCA hardware implementation have been applied ten times to the *Cuprite* image with p = 14. Results shown in Figure 4.21 clearly state that the spectral angles between the proposed FPGA implementation of the MVCA algorithm and the VCA algorithm are pretty similar. In general, the former gets reduce the maximum deviations (spectral angles) with respect to the USGS reference signatures. In particular, for the case of the VCA algorithm the average maximum and minimum spectral angles have been of 8.19° and 3.96° respectively, while for the case of the MVCA algorithm the average maximum spectral angle decreases until 7.82° and the average minimum spectral angle only increases up to 4.05°.

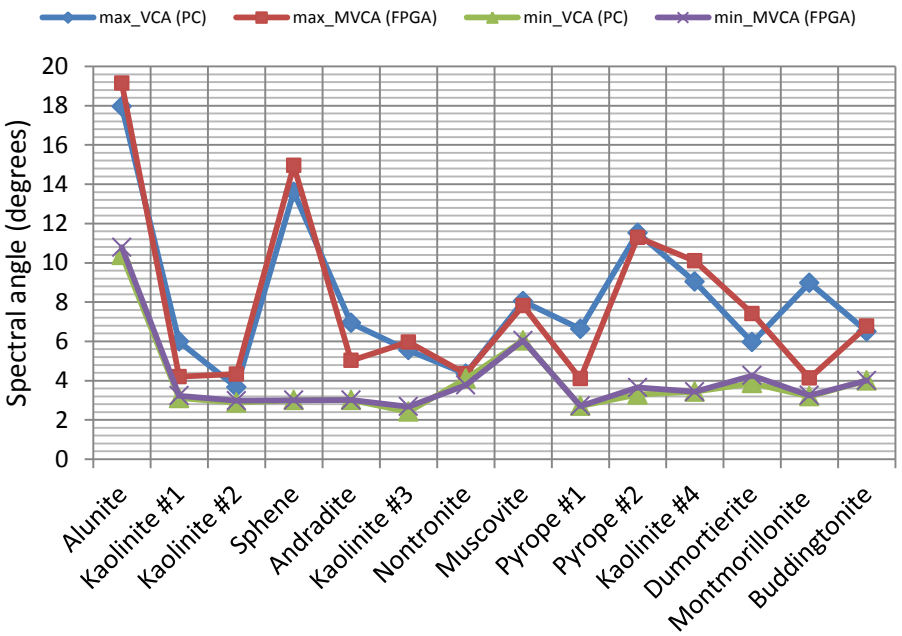


Figure 4.21 Endmember extraction accuracy results

4.5.2 FPGA implementations

This subsection discloses the synthesis results of the proposed MVCA architectures onto the selected FPGA device. These data are expressed in terms of hardware resources and the amount of required time for extracting a finite set of endmembers from a given hyperspectral image, but also for different levels of scalability.

First of all, and seeking for clarity, the three presented architectures are distinguished by their names: `Original_MVCA`, `SpectSA_MVCA` and `PixelSA_MVCA`. The former one refers to the architecture that was presented first (Section 4.4.1).The `SpectSA_MVCA` corresponds to the scalable architecture that parallelizes the number of the spectral components that might be processed simultaneously (Section 4.4.2.1). The latter design, `PixelSA_MVCA` architecture, refers to the scalable architecture that processes several pixels in parallel (Section 4.4.2.2).

These three architectures share some common characteristics among them that should be mentioned before analyzing the results. Related to the reliability of the results, the three options provide the same precision in terms of the endmembers extraction accuracy, independently of the spatial size of the hyperspectral image. Furthermore, regarding their designs, the three architectures are focused on parallelizing the IMAGE PROJECTION module and, therefore, the U_GENERATOR and F_GENERATOR modules are the same for all of them. However, the parallelization strategy used for processing the IMAGE PROJECTION is completely different among these three architectures. Moreover, although the solutions are flexible, since they are able to extract a wide range of endmembers, the scalability is not explored for all of them. Only SpectSA_MVCA and PixelSA_MVCA are capable of adjusting their number of processing elements once the number of endmembers (p) has been fixed. On the contrary, under these circumstances, the number of multipliers Original_MVCA remains fixed by always processing all the spectral bands simultaneously. Due to this fact, Original_MVCA is considered as a static design during the rest of the analysis and comparisons.

With the goal of establishing the advantages and disadvantages of these MVCA architectures, a comparison in terms of hardware resources and performance is presented along the rest of this section. First of all, the amount of hardware resources is analyzed graphically. Thus, Figure 4.22 shows the slice occupancy of the IMAGE PROJECTION module within the whole scalable architectures (SpectSA_MVCA and PixelSA_MVCA), when the number of processing elements varies from 1 to 15. These numbers allow to figure out the following conclusions. On the one hand, due to the fact that the PE element is exactly the same for both architectures (122 slice registers, 80 slice LUTs, 4 DSP48E), the mismatch between their logic resources is because of how their corresponding control units have been designed. On the other hand, the impact of the scalable IMAGE PROJECTION, as part of the rest of the MVCA architecture, might be determined. However, in order to do this, it is necessary to compare these numbers to the global occupancy of these architectures (Figure 4.23-25).

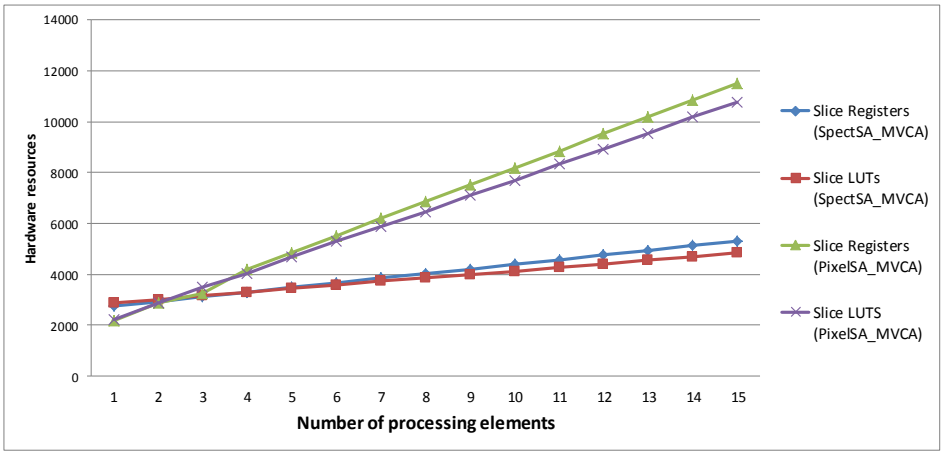


Figure 4.22 IMAGE PROJECTION module occupancy in SpectSA_MVCA and PixelSA_MVCA architectures

The data depicted in Figure 4.22 shows how the hardware requirements of the IMAGE PROJECTION module rise linearly when the number of processing elements increases. As it was previously mentioned, the number of PEs has the same effect in terms of number of resources in both architectures. Therefore, the influence of the adder-tree and the storage requirements vary in the case of the SpectSA_MVCA and PixelSA_MVCA architectures respectively. In the case of the SpectSA_MVCA the adder-tree is always present in the design, and consequently its contribution to the number of resources remains constant when the number of PEs increases. On the other hand, despite the fact that the PixelSA_MVCA does not incorporate an adder-tree, its pixel storage requirements vary with the number of PEs. This is a direct consequence from the scalability process. Whereas the SpectSA_MVCA only has to storage one pixel for all the PEs, the PixelSA_MVCA needs to register one pixel per PE. Therefore, when the number of PEs is higher than three, the image projection module of the PixelSA_MVCA architecture is more demanding in resources than the SpectSA_MVCA.

Then, Figure 4.23-25 represent the number of FPGA slices occupied by the three approaches when a different number of endmembers is extracted. In addition, different levels of scalability have been explored for each chart, with the exception of Original_MVCA that is always characterized by one point (when

the number of its multipliers coincides with the number of endmembers to extract).

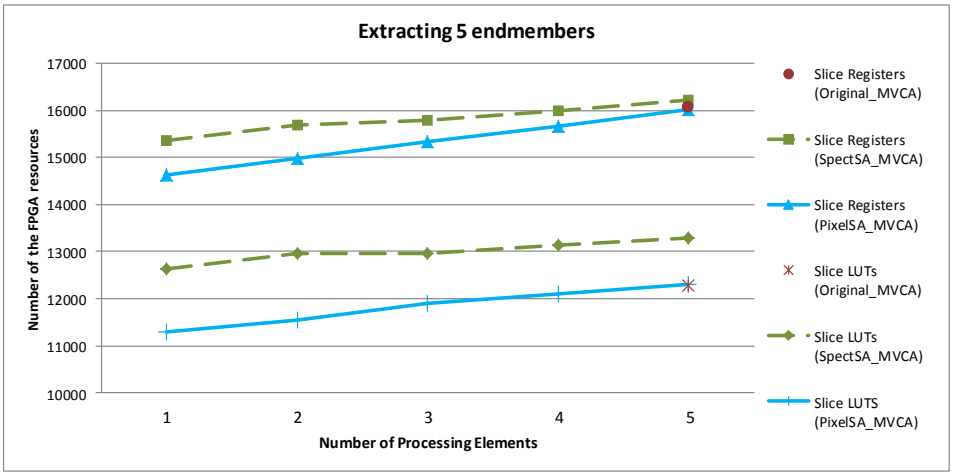


Figure 4.23 Synthesis result for extracting 5 endmembers

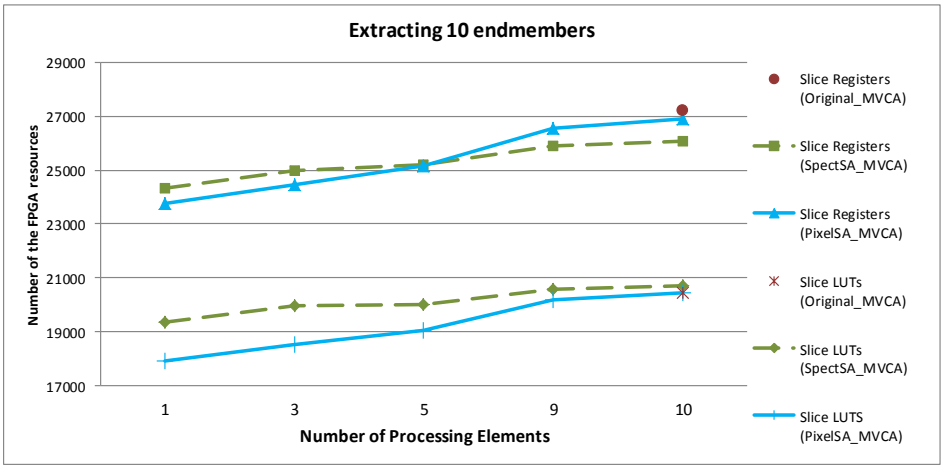


Figure 4.24 Synthesis results for extracting 10 endmembers

In figures (Figure 4.23-25) above the horizontal axis represents the scalability level, which means the number of processing elements enabled for the computation; whereas the vertical axis represents the number of the FPGA resources consumed for implementing the design. These values have been obtained after synthesizing the designs onto a Xilinx Virtex-5 SX95T.

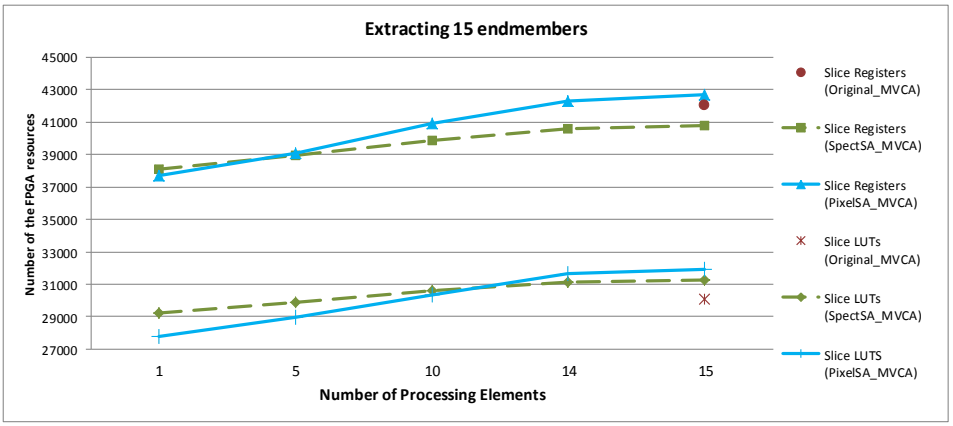


Figure 4.25 Synthesis results for extracting 15 endmembers

Figure 4.23 shows all the scalability choices of the selected architectures, when p=5. However, in the rest of figures (Figure 4.24 and Figure 4.25) where the number of endmembers to extract is 10 and 15, respectively, only a few numbers of configurations are represented. Notice that the Original_MVCA is represented by only one value in all these charts. As it was mentioned before, once the number of endmembers to extract has been determined, the Original_MVCA architecture is considered as a static design, without any possibility of scaling its computation.

Despite the fact that the three architectures share some modules of their designs, the synthesis resources (the number of slice registers and slice look-up-tables (LUTs)) notably vary from one to another. The main reason for this disparity in the numbers is due to way in which the IMAGE PROJECTION module has been designed. As it was expected, there is a direct relationship between the scalability level and the number of consumed resources. Thus, the higher scalability demands higher resources. Surprisingly, according to the graphical results, the modifications introduced into the SpectSA_MVCA and PixelSA_MVCA designs in order to let vary the number of PEs available in the system, do not have a significant impact on hardware. Moreover, in these two scalable architectures, the slope of the curves of hardware resources tends to moderate their slope when the number of PEs is closer to the number of endmembers.

On the other hand, the scalability has different effects on the number of resources for the SpectSA_MVCA and the PixelSA_MVCA architectures. Independently from the scalability level, the SpectSA_MVCA only has to store only one pixel for feeding the PEs and ensuring a correct behavior of the system, but also the adder-tree always occupies the same number of resources. Nonetheless, the PixelSA_MVCA is more sensitive to the scalability variations, since it has to store as many full pixels as the number of PEs there is in the system.

Regarding the percentage of the FPGA resources occupied by these three proposed architectures, the data are analyzed two by two, according to the information collected in Table 4.I. In fact, the comparisons are established between the Original_MVCA (when the number of PEs is equal to *p*) and the SpectSA_MVCA; and between the SpectSA_MVCA and the PixelSA_MVCA (when the number of PEs is *p*). As for the Original_MVCA approach, the number of slice registers varies from the 27% (*p=5*) to the 71% (*p=15*), while the number of slice LUTs ranges between the 20% (*p=5*) and the 51% (*p=15*). Under the same circumstances, and assuming the highest level of scalability the number of slice register for the SpectSA_MVCA varies from the 27% (*p=5*) to the 69% (*p=15*) respectively, whereas the number of slice LUTs are a little higher than the Original_MVCA, varying between the 21% (*p=5*) and the 53% (*p=15*). Then, analyzing the resources utilization of the PixelSA_MVCA, its number of slice registers changes between the 24% (p=5) and the 72% *(p=15)*, while the number of slice LUTs fluctuates between 19% and 54% respectively.

According to the synthesis results, the Original_MVCA architecture might operate with a frequency up to 268.15MHz, while the maximum frequencies achieved for the scalable architectures (SpectSA_MVCA and PixelSA_MVCA) are 244.29MHz and 243.15MHz respectively.

Considering the results collected in Table 4.I, the flexibility offered by both scalable proposed architectures (SpectSA_MVCA and PixelSA_MVCA) does not require a significant amount of extra hardware resources compared to the static reference design (Original_MVCA). Moreover, the reduction in the clock fre-

quency of these scalable designs is a 9.23% lightly lower than the `Original_MVCA` architecture.

Table 4.I Synthesis results on the FPGA Virtex XC5VSX95t

HW DESIGN	Configuration		Synthesis report		
	N. of endmembers	N. of PEs	N. Slice Registers	N. Slice LUTs	Freq. (MHz)
Original_MVCA	**5**	**5**	**16,063 (27%)**	**12,275(20%)**	268.15
	10	**10**	**27,223 (46%)**	**20,438(34%)**	
	15	**15**	**42,023 (71%)**	**30,103(51%)**	
SpectSA_MVCA	**5**	1	15,366 (26%)	12,641 (21%)	244.29
		2	15,687 (27%)	12,949 (22%)	
		3	15,788 (27%)	12,968 (22%)	
		4	16,000 (27%)	13,132 (22%)	
		5	**16,212 (27%)**	**13,297 (22%)**	
	10	1	24,333 (41%)	19,363 (32%)	
		3	24,995 (42%)	19,953 (33%)	
		5	25,179 (43%)	20,018 (33%)	
		9	25,908 (44%)	20,559 (34%)	
		10	**26,089 (44%)**	**20,693 (35%)**	
	15	1	38,098 (64%)	29,242 (49%)	
		5	38,944 (67%)	29,897 (50%)	
		10	39,854 (67%)	30,572 (51%)	
		14	40,578 (68%)	31,108 (52%)	
		15	**40,776 (69%)**	**31,257 (53%)**	
PixelSA_MVCA	**5**	1	14,630 (24%)	11,280 (19%)	243.15
		2	14,978 (25%)	11,554 (19%)	
		3	15,322 (26%)	11,886 (20%)	
		4	15,668 (26%)	12,094 (20%)	
		5	**16,014 (27%)**	**12,300 (20%)**	
	10	1	23,763 (40%)	17,913 (30%)	
		3	24,464 (41%)	18,525 (31%)	
		5	25,166 (42%)	19,036 (32%)	
		9	26,570 (45%)	20,160 (34%)	
		10	**26,921 (45%)**	**20,429 (34%)**	
	15	1	37,692 (64%)	27,796 (47%)	
		5	39,116 (66%)	28,947 (49%)	
		10	40,897 (69%)	30,368 (51%)	
		14	42,320 (71%)	31,640 (53%)	
		15	**42,676 (72%)**	**31,919 (54%)**	

With the purpose of analyzing the performance of the proposed architectures, in terms of the speedup, Table 4.II and Table 4.III collect information regarding the number of clock cycles necessary for extracting one endmember, when several scalabilities are used. More in detail, the number of clock cycles are measured at two points: when the MVCA starts processing (*clock cycles MVCA* column), and when the IMAGE PROJECTION starts processing once the *f* vec-

tor has been calculated by the rest of the MVCA system (*clock cycles IMAGE PROJECTION* column).

Then with these numbers, and considering the maximum clock frequencies of each of the architectures, the time spent in extracting one endmember (*1 endm* column), or several ones might be inferred, according to the following expression (7):

$$time\ (ns) = \frac{1}{frequency\ (MHz)} \cdot cycles_{MVCA} \cdot p \qquad (7)$$

The information shown in Table 4.II has been obtained when 5 endmembers are extracted from a small synthetic hyperspectral image of 36×36 pixels. Then, results collected in Table 4.III correspond with a more realistic simulation, in which 14 endmembers are extracted from the Cuprite image (250×191 pixels).

Independently on which of the three architectures is selected, the generation of the vector *f* requires the same number of clock cycles for the three cases, since the U_GENERATOR and F_GENERATOR modules are the same. More specifically, the generation of the vector *f* requires four clock cycles since the MVCA starts processing. Therefore, the rest of the time is spent in extracting the endmembers, which means computing the matrix multiplication between the vector *f* and the hyperspectral image Y_{int}.

According to the results shown in Table 4.II, the Original_MVCA architecture is the fastest one, when the three architectures operate with 5 PEs. This result is based on the fact that the Original_MVCA requires less number of clock cycles for extracting one endmember, and it reaches the highest clock frequency.

According to these results, it is evident that both scalable architectures (SpectSA_MVCA and PixelSA_MVCA) accelerate the endmembers extraction process whenever their number of PEs increases. However, the scalability does not have the same influence in the speedup of these architectures. In the case of the PixelSA_MVCA, the decrement in the number of clock cycles is directly proportional to the number of PEs involved in the IMAGE PROJECTION task. On the

other hand, when the number of PEs (N_{PE}) is higher than p/2 and lower than p $\left(\frac{p}{2} < N_{PEs} < p\right)$, the SpectSA_MVCA architecture does not experiment any acceleration compared to the situation in which the number of PEs is equal to p/2.

Table 4.II Behavioral simulation with a synthetic image (36×36 pixels and 5 endmembers)

Configuration			Behavioral simulation – 1 endmember extraction					
				Time (ns)		Clock cycles	Post P&R results	
ENDMEMBERS = 5 FRAME = 1,296 pixels	PEs	Freq. MHz	Clock cycles MVCA 1 endm	1 endm	5 endm	IMAGE PROJECTION 1 endm	Freq. MHz	Time (ns)
Original_MVCA	**5**	**268.15**	**1,306**	**4.87**	**24.35**	**1,302**	**178.22**	**36.64**
SpectSA_MVCA	1	244.29	6,514	26.69	133.45	6,510	130.63	249.33
	2		3,922	16.05	80.25	3,918		150.11
	3		2,625	10.74	53.70	2,621		100.47
	4		2,625	10.74	53.70	2,621		100.47
	5		**1,330**	**5.44**	**27.20**	**1,326**		**50.90**
PixelSA_MVCA	1	243.15	6,501	26.73	133.65	6,497	145.36	223.61
	2		3,262	13.41	67.05	3,258		112.20
	3		2,183	8.97	44.85	2,179		75.08
	4		1,644	6.76	33.80	1,640		56.54
	5		**1,321**	**5.43**	**26.95**	**1,317**		**45.43**

Table 4.III Behavioral simulation with Cuprite image (250×191 pixels and 14 endmembers)

Configuration		Simulation analysis – endmembers extraction					
			Synthesis result			Post P&R results	
				Time (ns)			
ENDMEMBERS = 14 FRAME= 47,750 pixels	PEs	Clock cycles MVCA 1 endm	Freq. MHz	1 endm	14 endm	Freq. MHz	Time (ms)
Original_MVCA	**14**	**47,781**	268.15	**178.18**	**2,494.52**	178.22	**3.75**
SpectSA_MVCA	1	668,536	244.29	2,736.64	38,312.96	130.63	71.64
	3	238,786		977.46	13,684.44		25.59
	6	143,286		586.54	8,2111.56		15.35
	9	95,536		391.07	5,474.98		10.23
	11	95,536		391.07	5,474.98		10.23
	13	95,536		391.07	5,474.98		10.23
	14	**47,786**		**195.61**	**2,738.54**		**5.12**
PixelSA_MVCA	1	668,521	243.15	2,749.41	38,491.74	145.36	64.38
	3	222,860		916.55	12,831.70		21.46
	6	111,450		456.48	6,390.72		10.73
	9	74,309		305.60	4,278.40		7.15
	11	60,804		250.06	3,500.84		5.85
	13	51,456		211.62	2,962.68		4.95
	14	**47,784**		**196.52**	**2,751.28**		**4.60**

This is due to the way in which the SpectSA_MVCA processes a pixel, where every PE perform several operations according to the following expres-

sion: $PE_j = \sum_{i=j+1}^{p} f_i \times y_{ki}; \begin{Bmatrix} k = 1..R \\ j = 0..(N_{PE}-1) \\ i = j : N_{PE} : p \end{Bmatrix}$. This behavior, together with the fact that all the PEs start and finish processing at the same time, obligates to all the PEs to execute the same number of operations. Thus, when the number of endmembers (p) is not proportional to the number of PEs, some PEs will remain idle until the rest of the PEs conclude their computation in the last iteration. This situation always happens when the number of PEs is higher than p/2. Another inconvenient of this design occurs when $N_{PE} = p$, since the accumulators belonging to the PEs remain unused during the whole endmembers' extraction process. For this reason, the less and the more efficient scalability cases, for the SpectSA_MVCA, occur when the numbers of PEs are equal to one and *p/2* respectively.

Beyond the synthesis results, post place and route simulations have been run in order to verify the proper execution of these architectures under more accurate circumstances. In this sense, Table 4.III includes the clock frequency after the post place and route stage (*Post P&R results* column), which is reduced in a 33%, 46% and 40% for the Original_MVCA, SpectSA_MVCA and PixelSA_MVCA respectively. The number of clock cycles is the same for the behavioral and the post place and route simulations for the three architectures. Hence, the static architecture is the fastest one compared to the scalable ones when the number of PEs coincides to the number of endmembers. Nevertheless, the lack of flexibility for adjusting the number of PEs of this architecture limits its reusability for running under different environmental conditions. As for the scalable architectures (SpectSA_MVCA and PixelSA_MVCA), both of them achieve similar performance when the number of hardware resources and the speedup are considered. However, the PixelSA_MVCA design permits a higher range of efficient scalability levels, since in all the scenarios a higher number of PEs is translated in less number of clock cycles. Moreover, all the PEs' resources are always utilized in this architecture.

In general, all these numbers make difficult to establish a fair comparison and then, determine which one of these three architectures is more efficient or

more appropriated for an FPGA-based embedded system. In order to simplify the analysis and offering an objective comparative, a figure of merit (FM). This factor considers itself all the previous data, according to the following equations (8) and (9):

$$FM = \frac{Frequency(MHz) \times Efficiency}{100} \quad (8)$$

$$Efficiency = \frac{10^8}{Total_{Resources} \times Clock\ cycles_{1endmember\ extraction}} \quad (9)$$

Considering these new issues, Table 4.IV represents a global comparison between the static and the scalable architectures presented in this chapter, when 5 endmembers are extracted from a synthetic image (36×36 pixels). Then, Table 4.V collects the same information but in the case of extracting 14 endmembers from the Cuprite image.

Table 4.IV Figure of merit for a synthetic image (5 endmembers)

Architecture	NPE	Total Resources	Clock cycles	Efficiency	Freq. (MHz)	FM
Original_MVCA	**5**	**28,338**	**1,306**	**2.70**	**268.15**	**7.25**
SpectSA_MVCA	1	28,007	6,514	0.548	244.29	1.34
	2	28,636	3,922	0.890	244.29	2.18
	3	28,756	2,625	1.32	244.29	3.24
	4	29,132	2,625	1.31	244.29	3.19
	5	**29,509**	**1,330**	**2.55**	**244.29**	**6.22**
PixelSA_MVCA	1	25,910	6,501	5.94	243.15	1.44
	2	26,532	3,262	1.16	243.15	2.81
	3	27,208	2,183	1.68	243.15	4.09
	4	27,762	1,644	2.19	243.15	5.33
	5	**28,314**	**1,321**	**2.67**	**243.15**	**6.50**

According to the *Efficiency* definition, the higher values of efficiency the better. Actually, this means lower numbers of resources (slice registers + slice LUTs) and clock cycles for extracting one endmember. In the case of the *FM*, it follows the same criterion. That is, the *FM* is better when it reaches higher values. This issue responds to the fact that it is interesting high clock frequencies.

With these considerations, and after analyzing the data shown in Table 4.IV, the `Original_MVCA` is the most efficient and the architecture with higher figure of merit, followed by the `PixelSA_MVCA` and then by the `SpectSA_MVCA`.

Table 4.V Figure of merit for Cuprite hyperspectral image (14 endmembers)

Architecture	NPE	Total Resources	Clock cycles	Efficiency	Freq. (MHz)	FM
Original_MVCA	**14**	**62,188**	**47,781**	**3.37E-2**	**268.15**	**9.02 E-2**
SpectSA_MVCA	1	62,807	668,536	2.38E-3	244.29	0.58 E-2
	3	63,361	238,786	6.61E-2	244.29	1.61 E-2
	6	64,269	143,286	1.09E-2	244.29	2.65 E-2
	9	65,016	95,536	1.61E-2	244.29	3.93 E-2
	11	65,532	95,536	1.60E-2	244.29	3.90 E-2
	13	66,049	95,536	1.58E-2	244.29	3.87 E-2
	14	**66,344**	**47,786**	**3.15**E-2	**244.29**	**7.71 E-2**
PixelSA_MVCA	1	60,242	668,521	2.48E-3	243.15	0.60 E-2
	3	61,512	222,860	7.29E-3	243.15	1.77 E-2
	6	63,382	111,450	1.42E-2	243.15	3.44 E-2
	9	68,429	74,309	2.06E-2	243.15	5.00 E-2
	11	66,745	60,804	2.46E-2	243.15	5.99 E-2
	13	68,011	51,456	2.86E-2	243.15	6.95 E-2
	14	**67,620**	**47,784**	**3.09E-2**	**243.15**	**7.53 E-2**

When the Cuprite image is processed, the FM is the same for both scalable architectures in those cases in which the number of PEs varies from 1 to p/2, and when the number of PEs is equal to the number of PEs. In the rest of the scenarios $\left(\frac{p}{2} < PE < p\right)$, the FM keeps constant in the case of the SpectSA-_MVCA architecture, whereas the PixelSA_MVCA enhances its FM.

Considering all these results, the Original_MVCA architecture is the most appropriated solution in the case of having enough resources and running in a static scenario, in which the number of endmembers to extract remains constant. However, in case of the number of resources is a critical factor, the scalable architectures show up as better candidates. Furthermore, when the environmental conditions might vary, it is more interesting and efficient moving toward the PixelSA_MVCA approach, since it offers a higher range of scalability levels where the number of PE might be adjusted between 1 and the number of endmembers to extract.

Finally, it is worth to mention that as the AVIRIS sensor is able to collect 512 hyperspectral pixels in 8.3 ms [GRE+98]. Hence, it might be concluded that the three architectures are suitable for real time hyperspectral endmember extraction systems.

4.6 CONCLUSIONS AND FURTHER RESEARCH

In this chapter, two scalable FPGA-based architectures have been presented as part of the linear spectral unmixing processing chain devoted to analyze hyperspectral images. The architectures are based on the MVCA algorithm, which extracts the endmembers of a specific hyperspectral image. The two scalable approaches have been synthesized onto a XC5VSX95T FPGA from Xilinx. Both cases result in a suitable solution for real time applications according to the acquisition times of the NASA's AVIRIS sensor. Furthermore, the exploitation of the scalability feature within these solutions introduces several strengths to the system, such as modularity (which increases the reusability of all the modules), flexibility (which facilitates changing the overall design configuration easily), and adaptability (which permits adjusting the architectural performance to different requirements).

Chapter 5

Dynamic Resource Manager

The design and development of dynamic and partial reconfigurable FPGA-based embedded systems is almost mandatory when dealing with high performance computing applications under changeable environments. Unfortunately, the management of the reconfiguration process in context-aware embedded SoCs is a complex task. In this sense, this chapter proposes a control and management element responsible for scheduling and ensuring a successful reconfigurability under these scenarios.

5.1 OUTLINE

A costly reconfiguration process and the lack of a reliable management supports have prevented a broader use of the FPGAs. In order to contribute to solve these issues, in this chapter a hardware/software dynamic resource management system is proposed. This solution combines scheduling and placement tasks, providing a complete management flow for supporting dynamically reconfigurable hardware designs. One of the advantages of the proposed model is the capability for running its scheduling and placement tasks in different nodes, as part of a distributed network. The results of our experiments demonstrate that our placement policy, specially designed for reconfigurable systems, achieves good results in terms of reusability and performance, compared to other management approaches.

5.2 EXPOSING THE PROBLEM

In the recent years, the field of embedded Systems-on-Chip (SoCs) has become more demanding and complex. This fact implies improving parameters such as the time-to-market, cost, power consumption, performance and flexibility. As a clear example, high performance computing applications require more powerful and faster devices in order to fulfill with real time constraints. These characteristics have to be combined together with reduced silicon areas, high reusability of the designs, but also a high efficiency of the resources available in the system. Real time performance limitations might be mitigated, however, by taking advantage of the integrated programmable logic of the FPGAs. In this sense, modern FPGAs are gaining wider popularity over GPPs and GPUs [TB10] for implementing applications that can benefit from parallelism, due to their well-balanced tradeoff between flexibility and performance. These devices offer the benefits of hardware determinism and reliability without the drawbacks of ASICs, such as cost and rigidity. With the explosion of embedded devices in the past few decades, many improvements have been made in both the hardware components and software tools. Despite this innovation and growth, traditional embedded system design approaches have evolved slowly, since there is a gap that needs to be filled. Given the increasingly rapid spread of new standards and

protocols as well as the increasing pressure on designers and developers to deliver their products faster and earlier, embedded-system design is due for a disruptive paradigm change. Additionally, the property of being configurable, allowing for loading new configurations and redefining the connectivity in the FPGA fabric, makes possible for designers including future proof designs. Moreover, this feature permits more robust updates and customizations without requiring any substantial hardware modifications in a post-fabric stage. The use of SoC FPGAs for real time applications provides not only integration benefits but also the ability to scale performance as needed.

The capability of some FPGAs to support dynamic reconfiguration [LF09] allows swapping different hardware functions onto the FPGA at run time, while the rest of the system remains operating normally, without being interrupted. This fact facilitates the adaptation of the FPGA to different requirements at run time, making it possible to keep the overall performance mostly stable all the time, or adapting the behavior to more restrictive or relaxed specifications when the system operates in hazardous and uncontrolled environments.

Unfortunately, there is still too much research work to do in embedded systems designs before high performance computing applications can leverage the benefits of this new paradigm. Thus, the complexity of managing dynamically reconfigurable resources is a hot spot, since it creates a significant barrier-to-spread the use of the FPGAs into the market. The solution goes through designing an intelligent mechanism able to manage the reconfiguration process but increasing hardware reusability, saving power and meeting timing constraints. Due to the fact that the reconfiguration management is not a trivial challenge, researchers tend to address the problem by considering scheduling and placement as separated issues. The former is responsible for analyzing, organizing and controlling what, how and where the reconfigurable tasks run. The latter carries out the physical implementation of those reconfigurable tasks into the reconfigurable region. The combination of both kinds of solutions may help to alleviate and overcome the lack of dynamically reconfigurable SoCs management.

In this chapter a *Dynamic Resource Manager* (DRM) system is proposed. It provides a complete flow of scheduling and placement tasks for FPGA-based embedded SoCs exposed to environmental variations. Two main features characterize this solution. The first one is regarding its hierarchical structure, which provides a device independent hardware reconfigurability service and, at the same time, permits uncoupling the scheduling and management tasks as independent but complementary elements of the same DRM. The second one is its capability for managing different designs by offering a flexible 2D structure of the reconfigurable region.

5.3 STATE-OF-THE-ART ON RECONFIGURATION MANAGEMENT

Modern reconfigurable devices such as FPGAs can be reconfigured at run time, and some of them can be even dynamically partially reconfigured. While flexibility opens a window to new paradigms on the reconfigurable computing field, it also introduces new challenges such as reconfiguration management and the efficient reusability of the available reconfigurable resources. Thus, management issues have become a priority for researchers in order to facilitate the development of dynamically reconfigurable embedded systems. In this section, some representative existing approaches published during the last years are reviewed. Many of these research works circumscribe scheduling issues to the physical management of reconfigurable resources. As an example, DreamSim [NAO+12] is a simulation framework for distributed systems in which all the nodes of the network might incorporate partial reconfiguration. The strength of this approach is its capability for defining customized task scheduling policies. However, they are always specified as part of the simulator. The weakness of this solution is the fact that it has been conceived for testing purposes. Therefore, no concrete design has been presented. In addition, the proposed simulations have not been tested with real workloads and realistic scenarios yet.

Moving toward running solutions, the approach introduced in [AAS12] swaps the software (SW) and the hardware (HW) version of the same task at run time. This fact increases system flexibility, since a HW task might be swapped for its SW version, or vice versa, depending on the environmental demands. The

basis of this approach lies on assigning three different states to the partial reconfigurable regions: free, ready or busy. The disadvantage of this solution is the fact that it behaves according to a data flow graph defined at compilation time. As a consequence, the system cannot be adapted to an uncontrolled scenario. As an alternative, [CRG+11] proposes a HW scheduler for managing reconfigurable systems at run time, based on DAGs (Directed Acyclic Graphs). The authors understand the reconfigurable region as a set of independent reconfigurable units (RU), without any relationship between them. These RUs might operate simultaneously, providing a multi-tasking environment. However, this idea of separated RUs might limit the number of solutions since it forces the design of the RUs to the worst reconfigurable module size that can be hosted. On the other hand, despite the fact that the authors evaluated their solution with several experiments, they do not provide the mechanism responsible for manipulating the reconfigurable bitstreams.

[JG11] proposes a framework for managing partial reconfiguration, in multi-application SoCs, named VAPRES. This framework introduces a dynamic resource manager which performs both scheduling and placement tasks. The defined scheduling algorithm follows an offline methodology, since the scheduler behaves according to a present and well-known data flow. This fact might limit its usability under variable environments. Other approaches, such as [KBR+11] and [CRS+11], propose a simulation framework and an automatic design flow for DRMs. In both cases, the designed systems are flexible enough for specifying the scheduling policies and achieving solutions independent from the architecture description. Authors in [KBR+11] provide a simulator for evaluating dynamic reconfiguration scenarios by using the SystemC language. Then [CRS+11] creates a DRM starting from a high level description application. As it can be appreciated here, no solution offers a general DRM solution for multitasking reconfigurable SoCs. On the contrary, they are application oriented, in the sense that they have to describe the applications prior to the development of the DRM.

5.4 A NEW DYNAMIC RECONFIGURABILITY PARADIGM FOR EMBEDDED SYSTEMS

Emerging applications are supposed to respond according to the environment and the user needs. Context-awareness implies that systems and applications integrate intelligence in a seamless way. That means they should include both, hardware and software mechanisms to reason and infer the correct system behaviour. New methodologies, strategies and platforms to support and manage self-adaptation and intelligent response should have to be devised. Dynamic reconfiguration has been during years a promising mechanism intended to provide systems with powerful features such as adaptability, self-healing, dynamic deployment, etc. Unfortunately, current design and management procedures still obligate the designer to develop detailed and cumbersome low-level tasks to accomplish such desirable characteristics. Overcoming challenges of this magnitude supposes a complex concern. In this sense, one of the main goals of this work is to fill the gap between the traditional conception of the dynamic reconfigurability and more complex working scenarios, in which is possible that several designs compete for resources at run-time within the same FPGA. In addition, the increment of the development of embedded systems and the high restrictive demands suggested by the users (in terms of adaptability, power savings, costs, etc.) open the window to a new range of flexible and complex solutions, in which the scalability and the evolvable features of the dynamically reconfigurable designs are the main characters. These changes have to be orchestrated by an entity, as part of the embedded system, capable of attending the reconfiguration request, and acting in consequence in order to provide a successful response.

In addition, the fact that many solutions separate the scheduling and management tasks independently makes the integration of both kinds of solutions together as part of the same embedded system difficult. Furthermore, many of them do not consider running under unpredicted environments, and consequently, they are not allowed to adapt the behavior or performance of embedded systems under dynamic scenarios. In other order of things, any of the ap-

proaches reviewed within the state-of-the-art consider managing scalable and/or evolvable hardware designs.

In this way, the proposed DRM for FPGA-based reconfigurable embedded systems, focused on covering weaknesses of traditional dynamically reconfigurable systems, has been founded on several restrictions and basic specifications that constrain the variations of the considered working scenarios and provide a simple but efficient solution to the DPR management.

5.4.1 Designs specifications imposed by the DRM

The proposed dynamic resource manager has to be able to provide a successful hardware reconfiguration process on an FPGA. However, in order to accomplish with this responsibility, the reconfigurable designs have to fulfill the following specifications: flexibility, modularity, scalability, reusability and reallocation.

5.4.1.1 Flexibility

Partial reconfiguration can be used for different types of designs. The most significant ones are modular and evolvable designs. In the former case, all the modules are indeed designed to share reconfigurable areas. However, evolvable designs tend to use the reconfigurable resources on-demand according to the evolution of the system, without redesigning the whole system. From the managing point of view, these two design methodologies differ in the scheduling strategies that they might use. In most of the cases, partial reconfiguration scheduling of a modular design might be planned in advance, by means of exploring diverse off-line scheduling strategies and considering a well-defined layout of reconfigurable regions, specifically defined for the reconfigurable modules according to their size and interfaces. In the case of modular designs, several granularity levels of area partitioning might be defined. The layout in this case is fixed and the shape and placement of areas depends on the modules that they contain. These modules can be swapped in and out or just can be placed in the corresponding area they have reserved. The communication infra-

structure between areas depends on the data path and the relationship between implemented modules, and usually forms a rigid system.

Nevertheless, in an evolvable design any free area may be used for placing new modules or hardware tasks that were defined after the deployment of the system. These reconfigurable areas are constantly released in response to an evolution of the initial system. This behavior requires more flexible scheduling solutions (on-line scheduling strategies). For evolvable designs, and also in order to obtain more flexibility, a 2D mesh configuration with predefined communication channels can be used for the reconfigurable region layout. Although a predefined communication channel implies a non-flexible structure, flexibility is obtained by the fact that this structure allows to place a module anywhere. This approach implies that all components should have the same interface that might be obtained with some kind of wrappers and therefore enabling the mobility of components over different areas. In this work we define a set of reconfigurable areas in a 2D mesh approach following this strategy. This 2D mesh approach is also extended with the two levels of granularity of macro and micro areas approach valid for evolvable designs. In addition, the system might vary its behavior by loading new configurations at run-time, or it also might vary the performance of a determined functionality.

5.4.1.2 Modularity

This feature should be considered at system but also at design structure. Regarding the system structure, the FPGA has two well-differentiated regions: the static and the reconfigurable ones. The former contains all those parts of the architecture that never change, whereas the reconfigurable region is formed by a bunch of areas in which the reconfigurable modules might be placed. At the design structure, the modularity facilitates the separation of tasks in different units. Thus, every hardware design is divided into a set of functional modules, where some of them might remain unchanged during its execution, while others modules might be swapped or changed at run time. In the end, this characteristic improves the reusability of the designs.

5.4.1.3 Scalability

Whether a module has the particularity of being scalable, this feature must be annotated as part of the characterization, and it should include also the maximum size of the array (one dimensional or 2-dimensional). It is worth recalling that all the instances belonging to the same scalable module must be placed in contiguous areas within the reconfigurable region in order to keep a regular 1D or 2D matrix structure.

5.4.1.4 Reusability

Once a module finishes or stops its execution, it remains allocated on-chip ready to be re-executed until a new module occupies the space. This criterion allows maximizing the reusability of the available resources.

5.4.1.5 Reallocation

A reconfigurable module might be moved to other reconfigurable regions, or to other areas within the same RR, without any inconvenience. This is possible when the number of resources of the new allocation is coincident to those required by the module, but also the organization of those resources has to follow the same pattern that the demanded by the module through its bitstream.

These characteristics might be understood as different reconfigurability cases, such as Figure 5.1 shows. In order to simplify the solution, in the beginning this work has considered only one reconfigurable region (RR) in the FPGA. However, along this work the RR is understood as a flexible structure composed by small reconfigurable areas that might be joined together in order to allocate bigger reconfigurable modules. Therefore, the RR may be seen as a 2-dimensional array of reconfigurable areas.

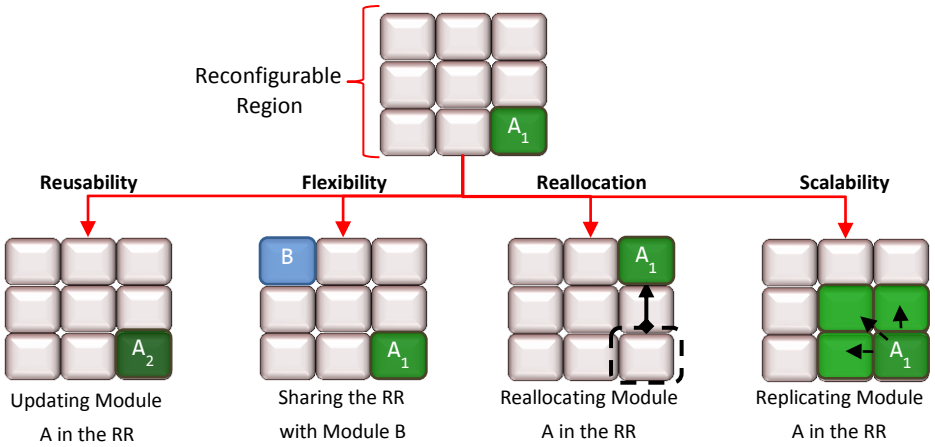


Figure 5.1 Desirable characteristics for dynamic and partial reconfigurable designs

5.4.2 Scheduling and management challenges

Some system and/or environmental fluctuations could influence on the modules' disposition on the reconfigurable region, in order to adapt the behavior and/or performance of the FPGA to those variations. But first of all, it is important to highlight that a reconfigurable module is a component that has been designed for being implemented in reconfigurable areas. Each module is associated with its corresponding configuration bitstream and an univocal identifier. If a hardware copy (replica) of a module is implemented as part of the reconfigurable region, then it is considered as an instance of the module. Each instance has also its corresponding identifier (ID). Therefore, several instances of the same module may coexist in hardware simultaneously, but all of them having different IDs. These replicas of a module might cooperate among them, forming arrays of elements, with the aim of sharing the workload and accelerating the execution. It is also possible that these replicas perform their corresponding functionalities separately independently to each other.

With all these considerations, there exist three main circumstances that might occur along the execution of the system.

1. There is enough space in the RR to achieve a successful reconfiguration. As Figure 5.2 shows, this is the easiest case to manage since the reconfigurable module might be directly allocated, without executing any previous operation in the RR. Thus, the instantiation of the module just requires the specification of the area where it has to be placed.

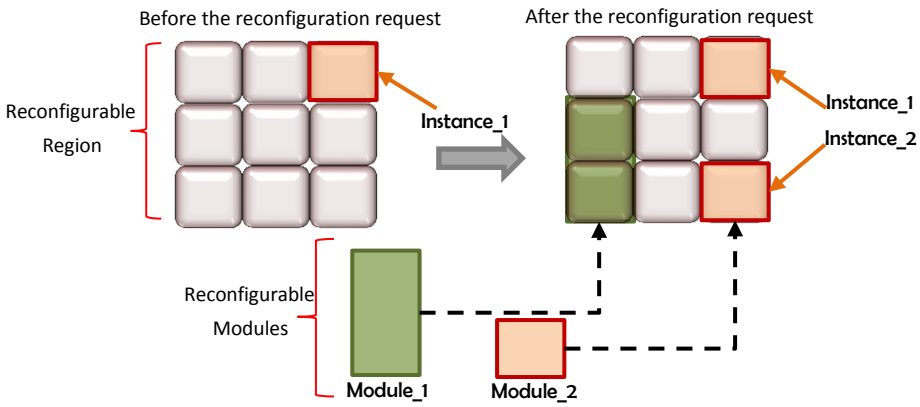


Figure 5.2 Enough space in the RR for reconfiguring the system

2. There are enough resources, but not enough space in the RR for reconfiguring the system. The difference between this case and the previous one is based on the fact that the instantiated modules are distributed all over the RR, keeping idle resources spread through the region. In this circumstance, the amount of idle resources would be enough for placing the requested instance whether all of them were concentrated together. Like Figure 5.3 depicts, in case of instantiating Module_2 in the R.R. it would be necessary to reallocate some of the other modules already instantiated.

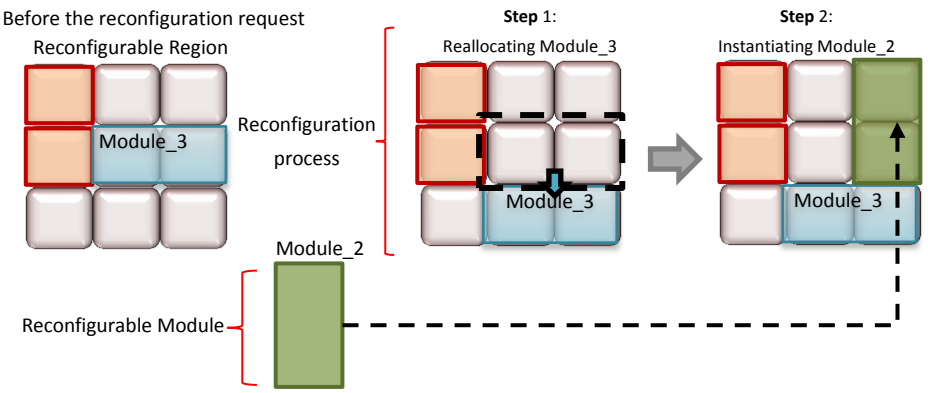


Figure 5.3 Enough resources but not space in the RR for reconfiguring the system

3. There is neither space nor resources in the RR for reconfiguring the system. In case there is not space, and the number of resources is not sufficient to contain the reconfigurable module requested by the system, there are two ways to proceed. The easiest one would be doing nothing. In other words, do not permit or do not attend to the reconfiguration request. However, this would be an inappropriate behavior in the case of the reconfigurable module was important. As an alternative, it would be interesting to provide a solution able to evaluate the importance of the reconfiguration request compared to the modules already instantiated in the RR. Thus, whether the reconfiguration request has a higher priority than some of the modules placed in the RR, then that (or those) modules have to be removed and let space to the incoming reconfigurable module. This context is detailed in Figure 5.4.

Finally, it is important to consider the case in which the incoming module has lower priority than all of the modules instantiated in the R.R. Under this situation the reconfiguration request might not be attended until some of the running modules finish their execution.

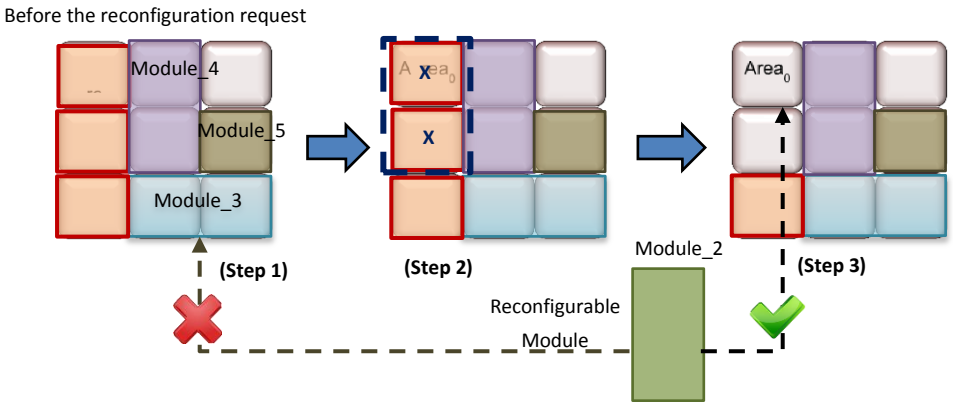


Figure 5.4 There is neither space nor resources in the RR for reconfiguring the system

5.5 PROPOSED SOLUTION: THE DYNAMIC RESOURCE MANAGER

The scheduling of the reconfiguration process is not a simple task. It includes the evaluation of free resources, the decision about which of the already instantiated components can be replaced by another one, the control of timing and latency of the reconfiguration process, or the mobility of components within the reconfigurable region, just to name a few of the aspects that should be considered to perform an efficient scheduling of the reconfiguration process.

In dynamically reconfigurable systems the scheduler and the placer tend to be so intimately related to each other that most of the time it is usually hard to find clear boundaries between them. However, this work proposes a *Dynamic Resource Manager* (DRM) structured in layers, where the scheduler and the placer are perfectly differentiated according to their functionality and implementation. The objective of the DRM is to provide a set of services to perform an efficient scheduling and control of the sequence and processes related to the partial reconfiguration of FPGAs.

5.5.1 Overview

The proposed DRM has been defined by means of a hierarchical structure, based on two main levels. The top-layer is a software Scheduler in charge of the evaluation of the most efficient way to perform a partial reconfiguration re-

quested by the application. At the intermediate layer, the Reconfiguration Engine (RE) is the responsible for the dynamic placement (and deployment) of the reconfigurable components on the reconfigurable grid. Most of these tasks are completely device and technology dependent and, for that reason, the main purpose of the Reconfiguration Engine is to provide a common abstraction, so that the dynamic reconfiguration of the components can be encapsulated as a transparent service to the upper layers.

Another contribution of this work is that the proposed DRM supports the scheduling and reconfiguration of scalable designs by means of replicating or removing multiple instances of the same module.

The proposed DRM system is based on a hardware/software co-design that follows the structure depicted in Figure 5.5. It supports services to dynamically configuring, controlling and monitoring reconfigurable modules. Even more, the DRM can identify common hardware modules and cache their partial bitstreams for later reuse by another application reducing in that way the reconfiguration time overhead.

As Figure 5.5 shows, the software part of the system is composed by a microprocessor, in which sequential software tasks and also the Scheduler are executed. The other layers, (the hardware part of the embedded system), differentiates the static and the reconfigurable regions of the FPGA. The static region implements the Reconfiguration Engine, and also those modules that remain without changes during the system execution.

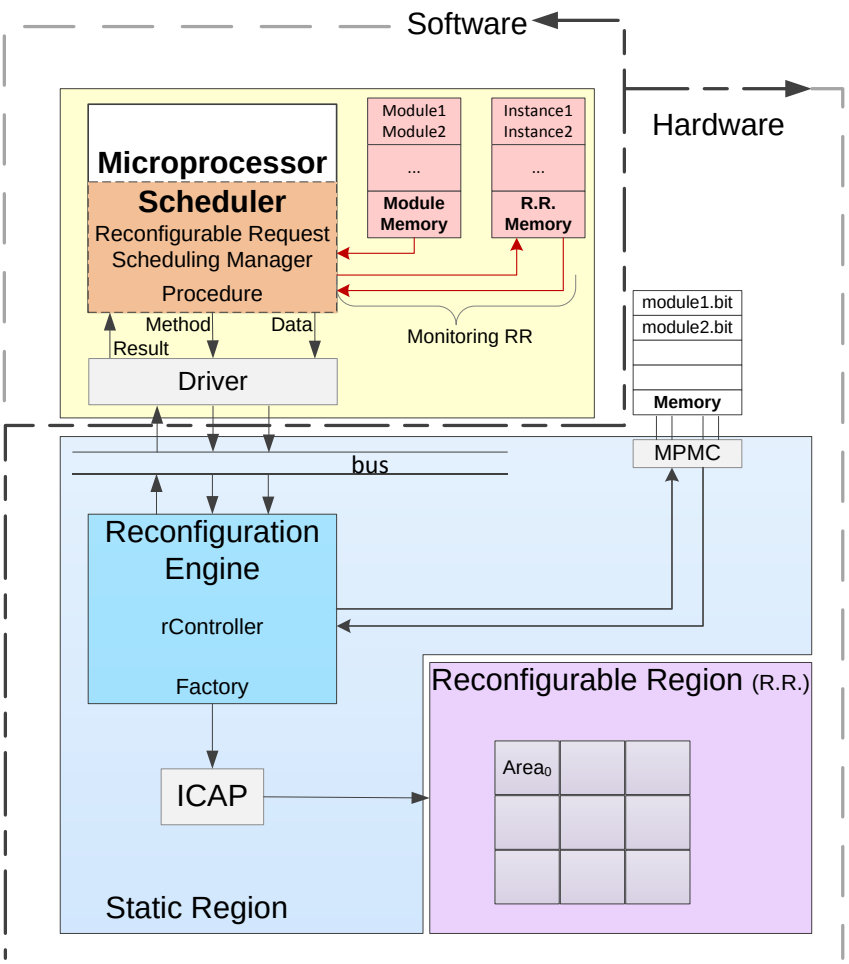


Figure 5.5 Dynamic Resource Manager structure

5.5.2 Detailed work

Along this subsection the proposed DRM is explained in detail, following its hierarchical structure from the top to the bottom; such that the Scheduler is introduced first. Then, the next layer, the bitstream manager referred as Reconfiguration Engine, is presented.

5.5.2.1 The Scheduler

The proposed Scheduler makes use of the basis of on-line scheduling methodologies in order to determine what, where and how reconfigure a hard-

ware module. The complexity of this entity lies on the fact that these decisions have to be made at run-time. Furthermore, the Scheduler respects some behavioral strategies with the idea of keeping a balanced compromise between performance and reconfiguration overhead. As a result, the Scheduler is able to orchestrate all the tasks related to the reconfiguration process in hardware by means of evaluating, analyzing, organizing and verifying the whole sequence of steps to ensure a successful reconfiguration. Seeking for simplicity, the behavioral strategies have been separated into two categories: reconfiguration, and allocation policies. The former collection of strategies has a structural nature, since they fix the actuation procedure to the Scheduler when a reconfiguration request is received. The allocation policies, on the other hand, are directly related to low-level tasks with the aim of maximizing the reusability of the logic resources, minimizing the utilization of the silicon area and reducing the reconfiguration time as much as possible.

5.5.2.1.1 Reconfigurable policies

The Scheduler has been provided with several databases in which different information is accessible in order to make its decisions. Part of this information is loaded at design-time, since it is used for characterizing every module that might be reconfigured at run-time. Other information is loaded and stored by the Scheduler during the execution of the system. This run time data is updated every time that a new modification is included into the reconfiguration region. These databases will be explained more in detail in this section.

In other order of things, the fact that the DRM has been conceived for adapting the behavior/performance of a system obligates to make decisions rapidly. This is a problem with a difficult solution when the DRM ignores how the environment is going to change, since there are many unknown parameters to be considered. This lack of a prior knowledge obligates the DRM to use online strategies for the scheduler. The characteristic of these kinds of strategies is that there is not a pre-loaded sequence of configurations. Therefore, it is not possible to dispose a priori what, where and how a modification has to be performed in the system. In these cases the system is responsible for deciding

when to adapt the RR, while the Scheduler determines where and how the reconfiguration request has to be carried out. Besides this is a dynamic process, the Scheduler structures its behavior into four stages, summarized in Figure 5.6.

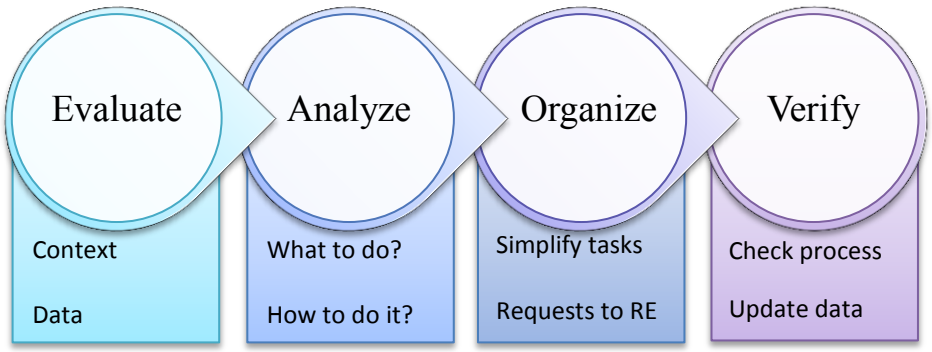


Figure 5.6 Main responsibilities of the Scheduler

The first one is an evaluation stage, where the Scheduler ensures on one hand that the reconfiguration process makes sense and, on the other hand, that it might be executed properly by assessing the context and the available data. On a second stage, the Scheduler analyzes what to do in order to support the reconfiguration demand, but also how to proceed according to the bundle of disposable functionalities implemented on it. Once the action has been selected, the Scheduler decomposes the task in smaller and simpler actions, whether it is necessary. Then, one by one it transmits the orders to the low-level manager (the Reconfiguration Engine) to execute it physically on the Reconfigurable Region. This procedure is repeated until all the basic actions have been completed. If any error occurs during the reconfiguration process, the Scheduler identifies what originates the fault and tries to solve it; otherwise the reconfiguration process fails and the error is reported to the rest of the system, being the previous context recovered. Finally, the last stage is to update all the corresponding information related to the executed reconfiguration. The whole process is represented in Figure 5.7.

As part of the novelties of the proposed DRM, it is important to highlight its capability for combining the traditional dynamically reconfigurability man-

agement together with the scalability of modules. Actually, these improvements are transparent to the Reconfiguration Engine, since the Scheduler is the element within the DRM who manages these cases, following the structure shown in Figure 5.7. Furthermore, in the case of receiving more than one reconfiguration request simultaneously, the Scheduler attends first of all the one with higher priority. This priority might be established by two ways. The direct method is assigning to every reconfigurable module a manual priority at design time. This value will be dependent to the environment and the device in which the system is going to run. The second way for discarding one request against the other is determining the most critical one according to several design and run-time factors, by means of using a cost function that considers all these aspects.

5.5.2.2 Allocation strategies

One of the most complex tasks of the Scheduler is inserting a new module, due to the dynamism of the RR in which many situations might occur along the time. The easiest case, as an exception, is the scenario in which there is enough space in the RR (because the region is empty, or there is little number of modules) for inserting a copy (instance) of the requested module. Here the behavior is simple, easy and fast, since the DRM only has to load the corresponding bitstream. Under other circumstances, the process is more complex and might require several steps.

Therefore, the Scheduler has to decide where to instantiate the module efficiently. For accomplishing with this goal, it evaluates several strategies in an attempt for reusing the logic resources, but also reducing the reconfiguration time as much as possible. In this sense, the Scheduler has to consider whether there is any idle module (of the same type that the requested one) that might be reused by reallocating it and restarting it.

Another alternative is the case where there is not any copy of the module that has been loaded previously in the RR stored in cache; or even if the incoming module has a higher priority than any other of the modules already instantiated in the RR. With these considerations, and using the same terminology than

[NAO+12], the Scheduler behaves according to the following strategies. First of all, the Scheduler tries to reuse any idle instance of the current module already placed in the RR. This strategy is known as Optional Closest-Match Strategy. This behavior is independent from the fact that there is enough space in the RR, since it is faster to reactivate or reallocate a configuration bitstream already loaded than accessing to the external configuration memory and recompose the bitstream. In the case the first strategy fails because all the instances are busy, the second strategy, known as Sufficient-Area Priority Strategy, is followed. The Scheduler looks for a recent copy of the requested module in cache, otherwise it instantiates a new copy in the RR but only if there is enough space in the RR. When all these strategies are not enough for achieving the reconfiguration, the Exact-Match Priority Strategy is used. That means there are neither idle instances nor available areas in the RR. At this point, the Scheduler checks in the RR instances if there are candidates to be removed in favour to the incoming one. The criterion for this decision is based on the priority of every module. Thus, an instance with less priority will be stopped and replaced by the requested one.

At this point it is important to highlight the wide flexibility of the proposed DRM compared to other solutions of the state-of-the-art, since it is capable of managing reconfigurable modules with different sizes that might be placed onto the same areas of the RR. Therefore, the Scheduler has to consider the size of every reconfigurable module before executing any of the aforementioned tasks.

Regarding the scalability feature of some modules, the Scheduler imposes certain restrictions to the increment of the number of their replicas. This growth always has to conform an array structure, that might be one or two-dimensional. In addition, all the replicas have to be placed together one to another. Moreover, in the case that the required expansion of the array cannot be completed due to the lack of space, the Scheduler always tries to select the configuration that includes a higher number of replicas, depending on the RR state.

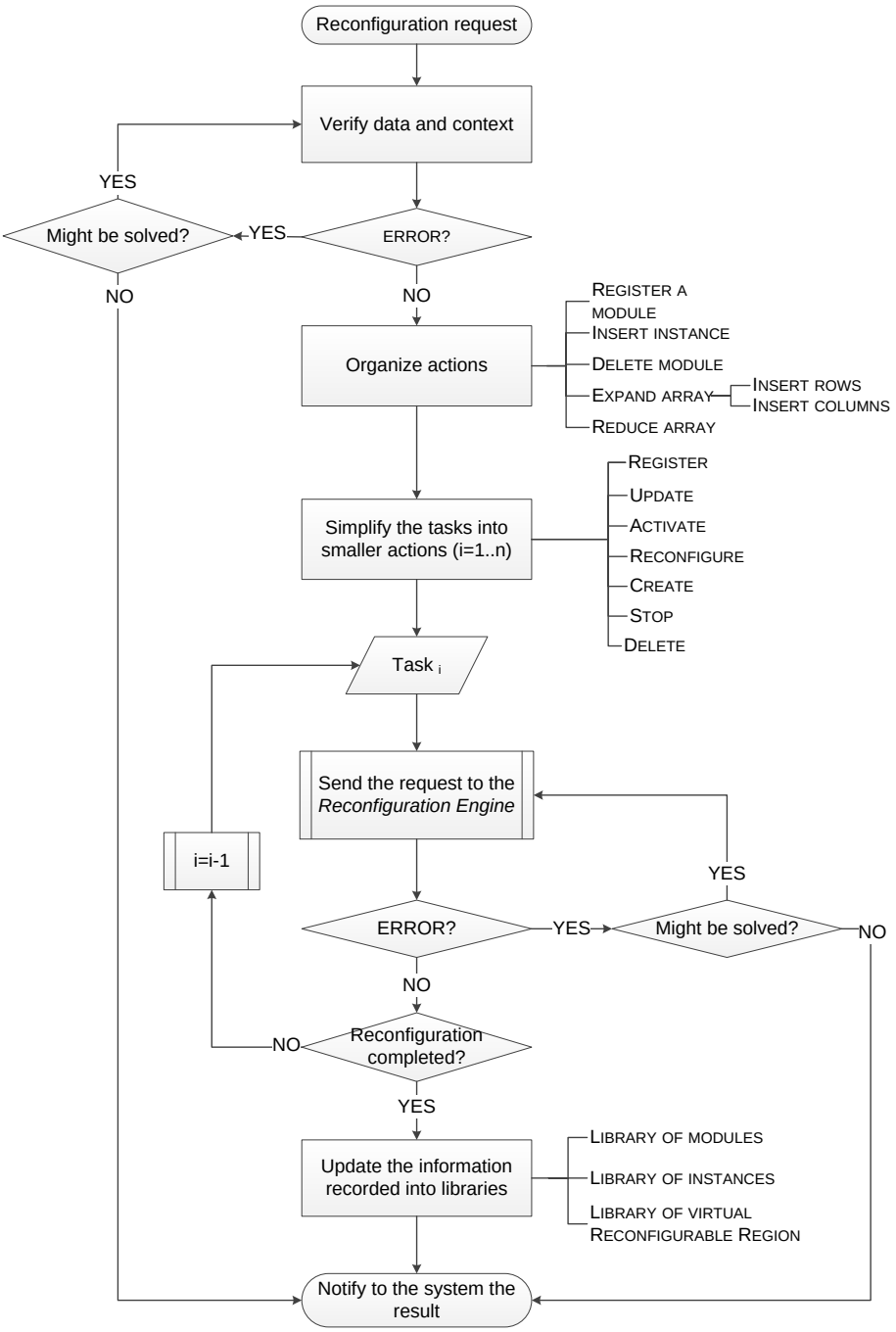


Figure 5.7 Reconfiguration request flow

5.5.2.3 Main functionalities

The DRM receives some instructions related to the reconfigurability through the Scheduler, which interprets those general instructions and operates in consequence by simplifying those ones into basic functions. Therefore, the Scheduler performs all the tasks concerning to the organization, analysis, controlling and updating of all the information related to the hardware reconfigurability. More specifically, it is focused on the evolution of the reconfigurable region. Furthermore, it also informs to the bottom layer (RE) on how to proceed, depending on the necessities of adaptation of the system. With these considerations, and attending to the previous behavioral strategies, the Scheduler provides a collection of functions, separated in top-level and basic functions depending on the abstraction level.

5.5.2.3.1 Top-level functions

These functions are the communication link between the rest of the system and the DRM. When a reconfiguration request is received by the Scheduler, the system also specifies its needs, in terms of which adjustment is required and how. Despite the fact that this is a desirable feature of the Scheduler, the first approach is still limited in intelligence. Accordingly, the system determines which reconfigurable module has to be handled, by using a univocal identifier (ID_MODULE), but also what kind of task is needed. The number of functions is limited to the following ones: register and delete a module, insert an instance, expand an array or reduce an array. The last two functions correspond to the insertion and removal of replicas of a module, and although they would be also implemented using the insert and delete functions, this differentiation accelerates the reconfiguration process, and reduces the number of interruptions from the system to the DRM.

5.5.2.3.2 Basic functions

The Scheduler simplifies the previous functions by using the basic functions described ahead:

Create: This function creates a new instance of a reconfigurable module onto a specific area. At this point, the Scheduler decides the area where the instance will be placed into the RR. The election of the final placement depends on the list of predefined areas in which the specific module might be instantiated; the free or idle reconfigurable areas currently available in the RR; or the priority of the incoming reconfigurable module compared to the currently running ones in the RR.

Register / Delete: These functions are in charge of including or removing a module to the list of reconfigurable modules, respectively. Two possible situations are distinguished. On one hand, the reconfigurable module has to be added to a preloaded design; on the other hand, the reconfigurable module is a new design in its own. The main difference between both is the fact that in the former case it is necessary to know the identifier of the design that it belongs to. However, in the latter case, a new design identifier is automatically assigned.

Activate /Stop: These procedures modify the execution status of a reconfigurable instance. The former (Activate) function starts the execution of a reconfigurable module, whereas the Stop function pauses or concludes the running activity of a module.

Reconfigure: This procedure is responsible for the reuse of instances that were previously used.

Procedures such as Register and Delete are always executed without being combined with any other procedure. On the other side, the rest of procedures might be chained depending on the reconfiguration request, such as Figure 8 depicts.

When an array of modules is scaled, the workflow is the same that the one shown in Figure 5.8, except by the fact that first of all the reconfigurable region is checked in order to ensure whether there are enough available areas, or the current instantiated modules have lower priority than the scalable module. If

one or both conditions are fulfilled, the scalability is executed. The growth of an array might be executed in columns, rows or in both directions.

Complementary to the methodologies and strategies mentioned above, the Scheduler could not accomplish with the reconfigurability without accessing to some information related to the system. Some of these data are collected at design-time, before the system starts running. However, there is another set of data that must be stored and loaded at run-time, since they are obeyed to the variations of the reconfigurable region. With the objective of simplifying the terminology, every chain of data is named as library. Thus, the Scheduler differentiates three types of libraries: library of modules, instances and reconfigurable region (Figure 5.9).

1. Library of modules: This chain of data stores information of every reconfigurable module that might be instantiated in the RR. Therefore, every element of the library characterizes univocally a module through its identifier (ID_MODULE), number of hardware resources (CLBS, DSPS and BRAMS), priority level, how scalable it is (indicating the maximum number of rows and columns it supports), and the memory address where its partial bitstream is stored. Despite the fact that most of the modules are characterized at design time, the Scheduler is flexible enough for allowing to update some registers of a module at run time.

2. Library of instances: This one is created as soon as there is any activity in the RR, since it is responsible for registering what instances are available in the system. Of course, this information has to be updated all the time, in order to provide an accurate and efficient control of the status of the system. Every instance has associated certain kind of information that allows to know which module it is, and whether it is currently running, idle, or whether it has been removed from the RR (although it is still “alive” in cache). In the case that the instance belongs to a scalable structure, this information is also annotated here. Actually, there are as many libraries of instances as modules are placed in the RR. This fact is possible because one module might have several copies of it in the RR. As a consequence, it is

necessary to identify every one of those replicas univocally by using an identifier (ID_INSTANCE), even when all of those copies belong to the same array.

3. Library of reconfigurable areas: This bundle of information virtualizes the reconfigurable region, in order to adapt its state during the execution of the system. In this sense, this library contains certain information preloaded at design time, but other one collected at run time. Regarding the design time, it is mandatory that the Scheduler knows in advance how many reconfigurable regions are there in the system, before the system starts running, but also how they are composed in terms of the number of areas, how those areas are distributed and the final size of the RR. On the other hand, the status of every reconfigurable area is updated at run time.

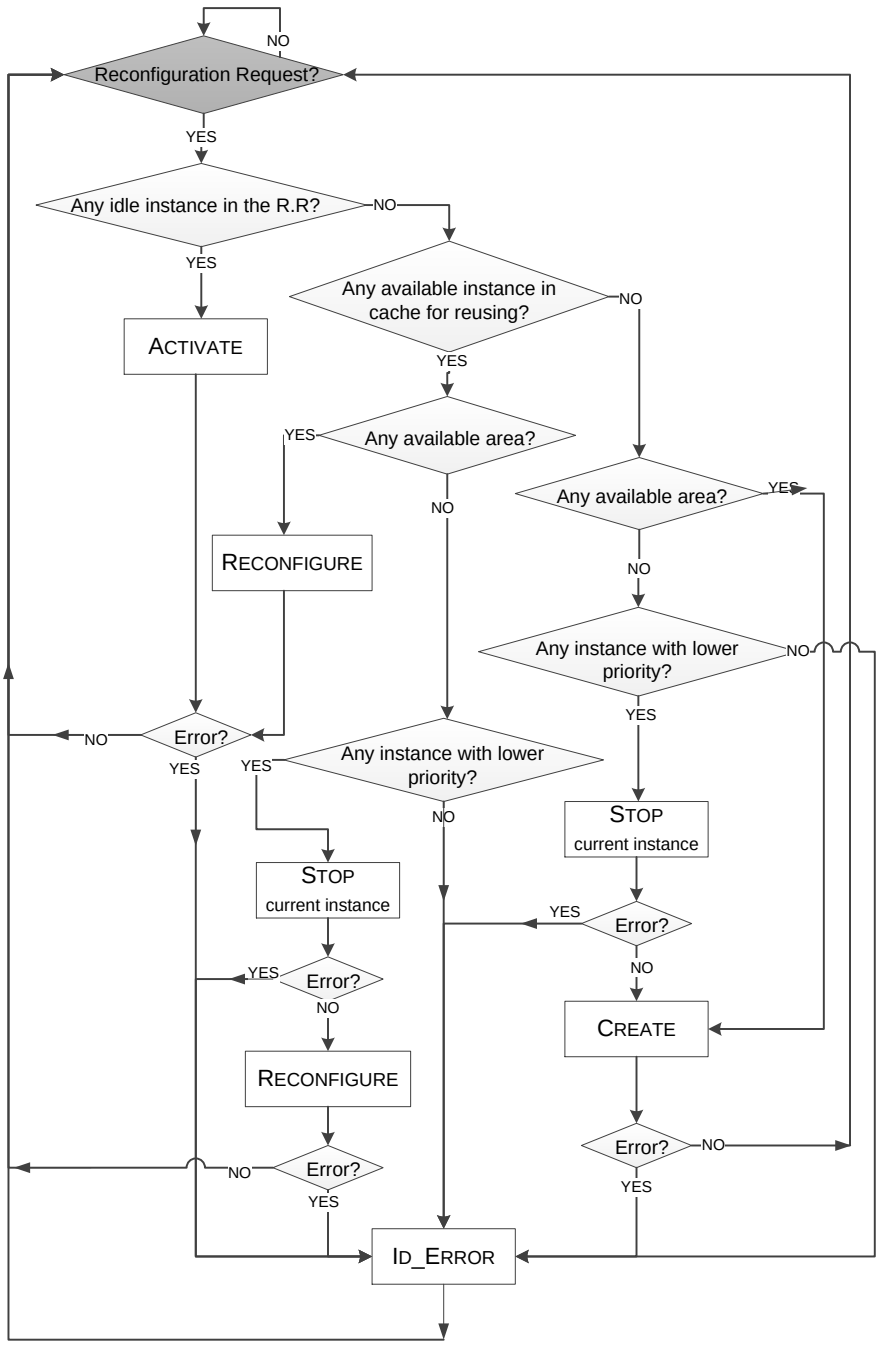


Figure 5.8 Behavioral flow of the Scheduler in order to include a reconfigurable module in the RR

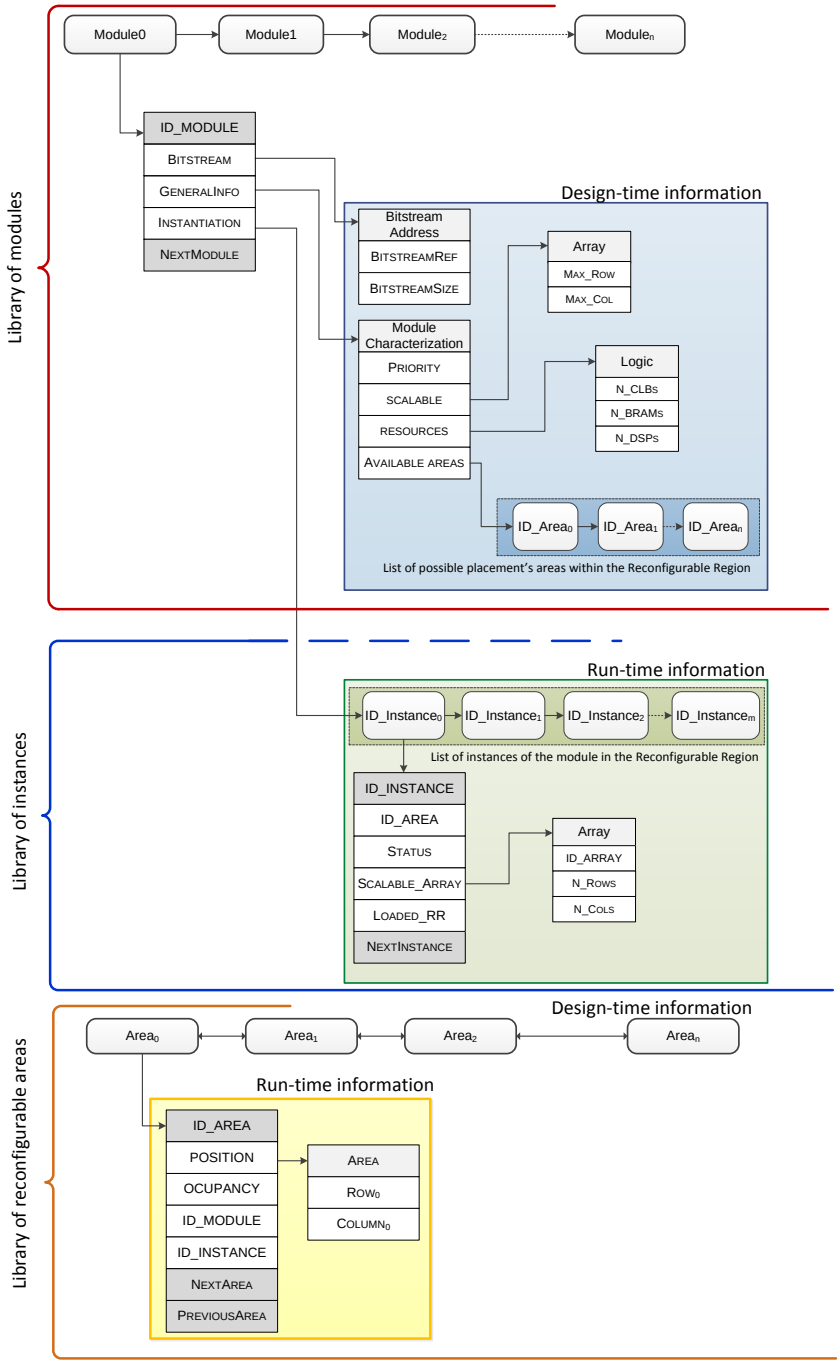


Figure 5.9 Data structures of the Scheduler

The Scheduler prevents the system from failures of the physical implementation during the reconfiguration process, but also from incomplete reconfigurations. Thus, the ID_ERROR signal notifies what has happened during the reconfigurability. In the case that a reconfiguration request cannot be attended correctly or fails, the Scheduler recovers the existing context before the reconfiguration request. In addition, the Scheduler gives back to the system certain information regarding the evolution of the reconfiguration process, which allows to check the process, step by step from the reconfiguration request to the end of the operations. Although, this information does not guarantee a correct behavior of the whole system, it has been implemented for debugging purposes. Moreover, all those modules that do not interfere to the current reconfiguration process are not interrupted and kept running.

5.5.3 Reconfiguration Engine (RE)

The execution of dynamic reconfigurations in FPGAs is typically solved using software approaches, where an embedded processor (such as the Microblaze) is used to transfer the configuration partial bitstream to the configuration memory of the device to the silicon area. As an alternative, the placement layer presented in this paper relies on a specialized hardware component (the Reconfiguration Engine - RE) to carry on with reconfiguration tasks directly related to the bitstream manipulation. This approach offers several advantages, such as it isolates the Scheduler from lower-level tasks and, consequently both elements (the Scheduler and the Reconfiguration Engine) might be independently released, while their interfaces remain the same. In addition, the RE design avoids requiring a processor for managing the reconfiguration process.

More in detail, the Reconfiguration Engine offers a set of reconfiguration services through a simplified interface. Those services include bitstream transference, the start and stop of individual instances, status requests, and location and relocation of components. Another important aspect related to the proposed Reconfiguration Engine is the way in which the reconfigurable region is managed. Traditionally, most of the proposals take a rigid coarse-grained ap-

proach, where the number and location of reconfigurable areas are completely predefined at compile time. Under this scenario it is mandatory that any reconfigurable component comply with certain restrictions, such as the location of the communication ports, size and shape factors.

From the architecture point of view, the Reconfiguration Engine is composed by two modules: the Reconfiguration Controller (rController) and the Factory. It also requires some kind of storage resources, such an external memory (Figure 5.5), although it is not part of the Reconfiguration Engine itself.

5.5.3.1 Reconfiguration Controller

The Reconfiguration Controller orchestrates the placement tasks of the reconfiguration process. It might be understood as the centralized control mechanism of the Reconfiguration Engine. Therefore, it is the access point to the RE from the rest of the system. The communication with the rController is performed through a set of messages that can be issued from a hardware component, embedded software, or even remotely through an Ethernet or a serial link, among others. In addition, it includes an internal table that dynamically stores the state of the reconfigurable region: location of the instances, bitstream references, and the execution status of the instances, or memory references for module persistence issues. Although, it seems that the Scheduler and the Reconfiguration Engine duplicate information, this fact has been implemented for security and controlling reasons, since thanks to this duplicity the DRM might detect an error faster.

The rController receives the reconfiguration commands from the Scheduler and executes the corresponding operations. These operations range from starting or stopping components to be evicted if necessary, saving the status of a reconfigurable component, and sending the reconfiguration request to the Factory. As it was mentioned in the previous subsection, every reconfiguration component has its univocal identifier (ID_MODULE) associated to a partial bitstream. Furthermore, every time a module is instantiated, the rController generates a new identifier (one per instance) (ID_INSTANCE). This identifier is used to

address the invocations to the corresponding module; i.e. a reconfigurable component *filterA* might be replicated several times generating in this way several instances of it such as *filterA_1; filterA_2* and *filterA_3.*

5.5.3.2 Factory

This component deals with the issues related to the transference and manipulation of bitstreams. As the name suggests, it is able to enable new reconfigurable components by means of transferring the corresponding bitstream from the storage memory to the configuration memory of the device by using a dedicated bus. It takes the role of a specialized DMA controller. In the case of Xilinx FPGAs, for example, the Factory moves the data from a Native Part Interface (NPI) interface memory to the corresponding Internal Configuration Access Port (ICAP) controller.

The Factory is also able to manipulate the bitstream on the fly, which means allowing the relocation of that bitstream in different reconfigurable areas. To perform these tasks it is necessary to have all partial bitstreams, associated with every reconfigurable module, previously stored in memory. When the Factory receives orders, it takes the information of the specified module (bitstream) and the area to be placed (ID_AREA). Then, it modifies the corresponding partial bitstream and sends it to the RR through the ICAP. The Factory also supplies the reconfigurable region layout of the system, which permits creating several instances of the same module. Furthermore, the communication with the Factory is asynchronous in order to avoid bus locking, and it takes a callback approach to indicate the completion or error of the operation.

5.5.4 Characterization of the Reconfigurable Region

At the bottom layer of the DRM, the reconfigurable region is configured as a grid of compassable reconfigurable areas. A minimum reconfigurable micro-area unit is defined for the specific system, which can be used separately, or integrated into a larger reconfigurable macro-area at run-time. In this context, the DRM is able to manage this dynamic 2D area model of the FPGA, where the reconfigurable region is structured as a homogeneous grid in which several are-

as might be coupled into bigger areas at run-time. This characteristic of agglutination makes the reconfigurable region more efficient since areas can be adapted to match with reconfigurable modules of different shapes and sizes. (Figure 5.10) As a consequence, over this mesh, bigger areas were defined grouping several of the original areas. These areas are joined together through their communication infrastructure, generating a hierarchical approach able to contain more resource demanding components.

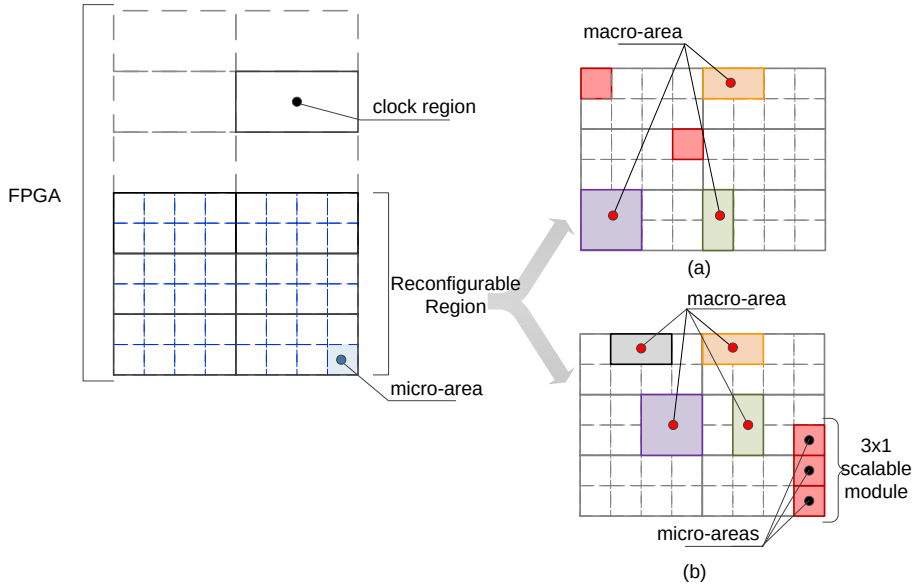


Figure 5.10 Possible evolution of the reconfigurable region (RR) at run time

5.6 A CASE STUDY AND RESULTS

Once a complete description of the DRM has been presented, this section combines all the concepts presented along this Thesis altogether; such that a reconfigurable application is implemented as part of an FPGA-based embedded system, and the proposed DRM supervises the dynamic reconfiguration process.

5.6.1 DRM behavioral simulation

With the increasing demands for embedded products, and the increase in hardware integration, the serial process of traditional methods of embedded development [Dub09], shown in Figure 5.11, gradually reveal many inadequa-

cies, such as a serial design process that increases the overall design cycle, poor flexibility of the final embedded system, difficulties in verifying the entire system due to the lack of a unified HW-SW representation and hence to incompatibilities across the HW/SW boundaries.

In order to combine the HW and SW integration, co-design methods emerge as an alternative beyond the traditional mechanisms. The main characteristic of co-design methods is they accelerate the verification stage, allowing for simulating HW and/or SW units in early stages without waiting until their development stage is concluded. Another difference with respect to traditional design method is that co-design methods of hardware and software emphasized the parallelism and mutual feedback (Figure 5.12).

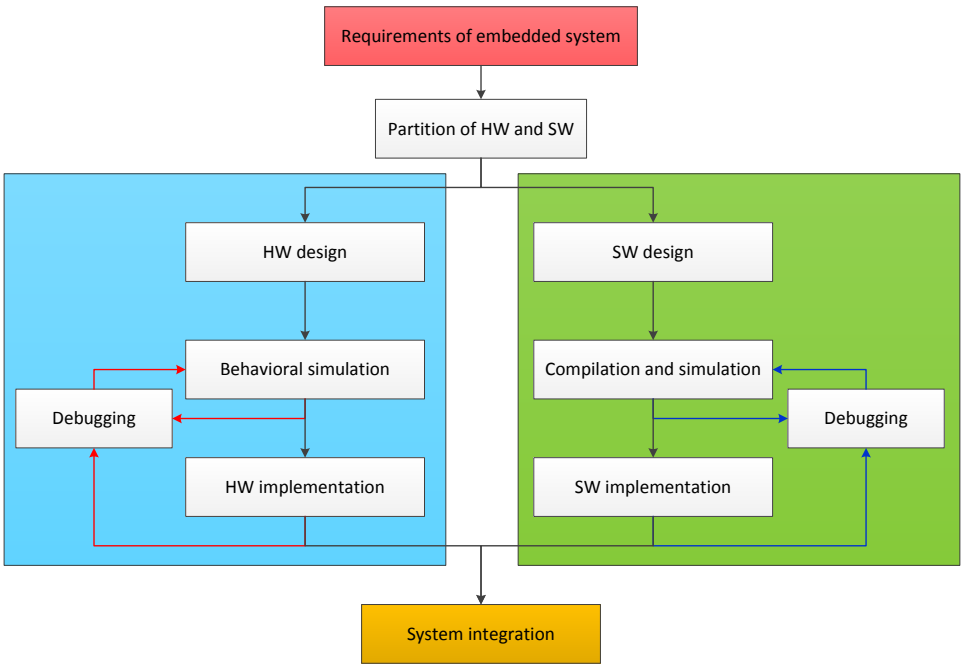


Figure 5.11 Traditional method of embedded system design

The main goal of co-design strategies relies on modelling a system structure, functionality and its temporal constraints in an attempt for accelerating the verification of the HW and SW elements of a system. With this purpose, the

co-design based on SystemC uses a uniform language to describe system functionalities.

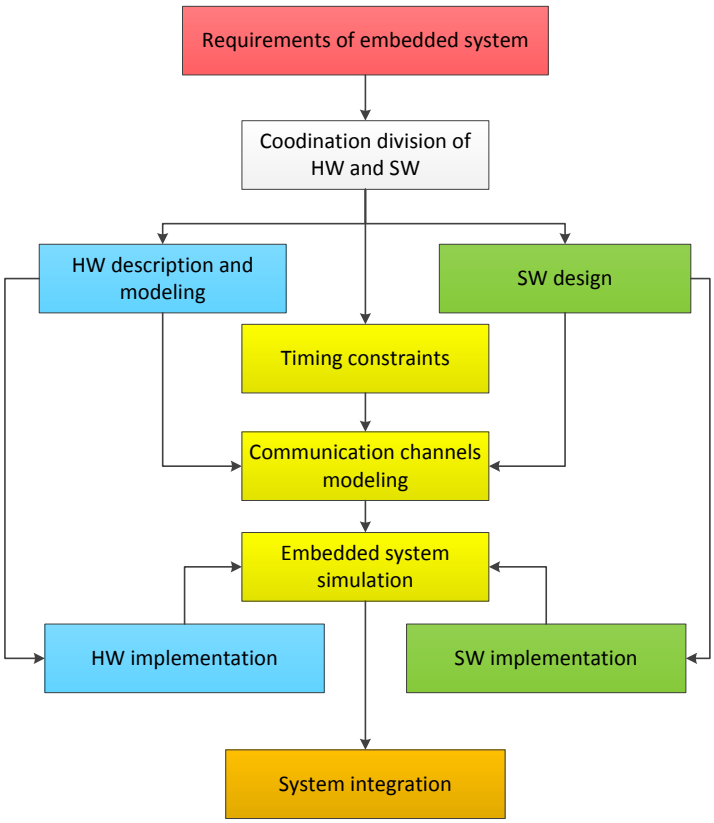


Figure 5.12 Hardware-Software co-design process

SystemC is a system modelling language that offers many data types and concurrent programming structures [Nie98]. It can describe structures of complex systems, and support the description for HW and SW interfaces. One of the most significant capabilities of SystemC is its support for modelling systems at different levels of abstraction. In order to simulate a SystemC program, an event-based simulation library is available so that timing aspects of a co-design may be evaluated for correctness not only of its functionality, but also with respect to the timing properties.

Nowadays, and more critically in a near future, embedded systems will require a certain degree of runtime adaptability in order to cope with unpredicted

and unexpected situations [Tei12]. With the apparition of FPGAs, the adjustment of the system is not constrained to the running software, but also it is available through the adaptation of the reconfigurable hardware. Hence, online techniques for HW/SW co-design, in order to achieve a context-aware embedded system optimization of the partitioning of hardware and software, are desirable.

With all these considerations, in order to provide a solution that fit with the management of dynamically reconfigurable systems, which contemplates scheduling aspects but also the procedure of the partial reconfigurability itself, it is necessary a framework capable of offering a HW/SW co-simulation. Furthermore, this framework should be general enough for modelling the dynamic reconfigurability. In this sense, Arco research group [ARCO] has developed a SystemC framework, as part of the DREAMS project. This platform pursuits for combining the advantages of most of the co-design strategies (providing co-simulation support, and facilitating the interfaces between the boundaries of HW and SW elements), but also being able to incorporate dynamically reconfigurable scenarios, together with their management. More in detail, system level HW/SW co-simulation is a way to give designers feedback on their designs. Then, the interfacing implementation domains offer different alternatives for communicating HW/SW in the form of cooperating circuits and software procedures embedded in the implementation. The simulation platform is capable for allowing hardware/software co-design, but also combining the dynamic reconfigurability on FPGAs together with its management. This simulation framework accelerates the embedded systems design development, since SW development can start immediately before the HW becomes available, preventing unnecessary design iterations, and allowing debugging and modifications in early stages of the process. Thus, modifications are immediately reflected and do not require recoding.

Validating the viability of the DRM management's functionalities goes through integrating it as part of a dynamically reconfigurable FPGA-based embedded system. Considering the hierarchical nature of the proposed DRM, the

Scheduler will be run on the microprocessor, like any other software task, whereas the Reconfiguration Engine will be part of the hardware elements of the system. For all these reasons, this platform has been selected in order to validate the proposed DRM.

5.6.1.1 A co-simulation platform based on SystemC

The objective of this SystemC platform is to provide a support for modelling the DRM behavior (including the Scheduler and the Reconfiguration Engine) and the Dynamically Reconfigurable Region for any applications running on an FPGA. The overview of this simulation platform is shown in Figure 5.13.

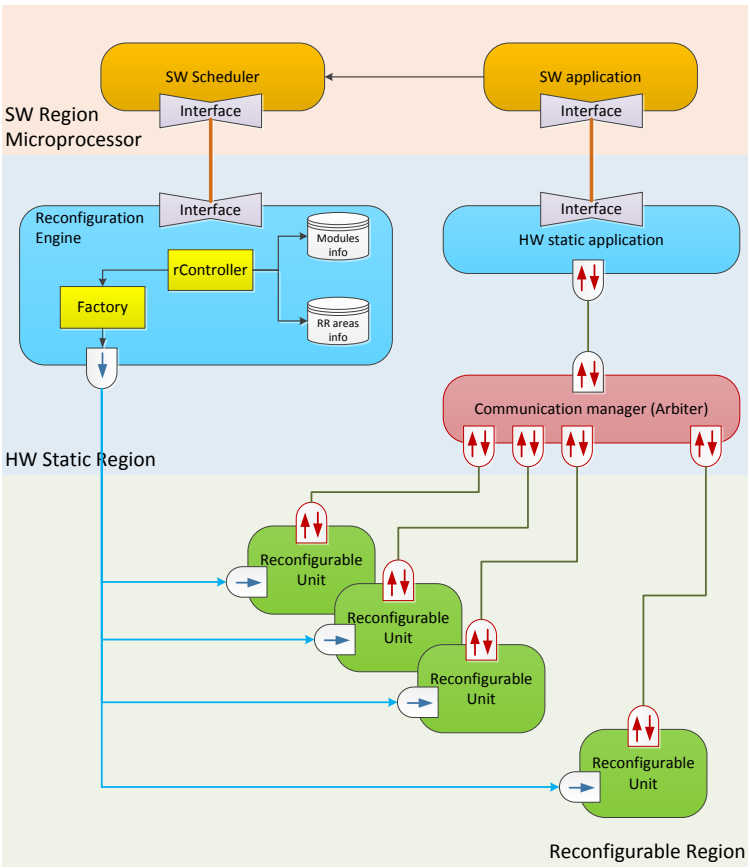


Figure 5.13 Co-simulation platform structure

The framework follows the same hierarchical structure than the DRM, in the sense that the top layer (*SW Region* in Figure 5.13) groups all the SW tasks

that will be executed in a microprocessor (the Scheduler and the rest of software tasks). Then, the middleware (*HW Static Region*) contains the Reconfiguration Engine and all the static hardware blocks of the desired applications. Finally the bottom layer corresponds to the reconfigurable region, where the reconfigurable modules are simulated. However, the limitations of SystemC language obligates to represent the dynamic reconfigurability in a similar way than the dynamic and partial reconfiguration Xilinx' pattern. That means, generating general units (named as *Reconfigurable Units* in Figure 5.13) where each one groups, as part of it, all the functionalities that might be activated on it along the time. Thus, regarding the communication, each reconfigurable unit has to be characterized for the worst possible case.

One of the strengths of this framework is its generality. It permits characterizing multiple FPGAs, and therefore simulating multiple situations, according to the application requirements. As a result, it might provide the basis for determining HW/SW partitions in early stages in the development of an application.

This co-simulation environment is mainly focused on the timing modelling of the final platform, with the aim of providing an accurate response to the input stimuli as an approximation to the real environment. In order to accomplish with these requirements, the co-simulation framework assumes that:

- The DRM unit (scheduling and reconfigurable modules' manipulation) will be responsible for controlling and managing the dynamic reconfigurability.
- As part of the reconfigurable region, reconfigurable modules might be independent among them (being directly connected to the static region through the communication manager), or being communicated with other reconfigurable modules in order to generate more complex structures (simulating the same behavior than a scalable design).
- Managing the connectivity between reconfigurable modules.
- Parameterizing the reconfiguration time of all the reconfigurable modules.

- Simulating real time workload conditions.
- Modifying the data flow at run-time.
- Adaptability to unpredicted environments.
- The communication interfaces between areas (the static and the reconfigurable one) will remain the same during the execution time.
- The swapping task between reconfigurable modules is modelled as a functionality change in the reconfigurable unit.
- The interconnection model has been generalized with the TLM standard.

5.6.1.2 DRM interfaces

The communication interfaces between the different levels of the DRM are carried out through a simple set of signals. The interface between the system and the Scheduler is similar to the one between the Scheduler and the Reconfiguration Engine, since in both cases it is necessary to specify what to do and where. More in detail, Table 5.I contains all the signals involved in the transferences between the system and the Scheduler.

Table 5.I Communication interfaces between the microprocessor and the Scheduler

DRM inputs	Description
NEW_RECONF	Reconfiguration flag request – *interruption*
ID_RECONF	Reconfiguration action identifier – *what to do*
ID_MODULE	Module identifier – *over whom to act*
CONF_SIZE_ARRAY	Desirable array dimension (this is for scalable designs)
DRM outputs	**Description**
STATUS_RECONF	Reconfiguration process status

The system interrupts the DRM when a reconfiguration is requested by enabling NEW_RECONF signal. Then, ID_RECONF and ID_MODULE signals determine what the system demands and related to whom, respectively. An example would be the case in which the system needs to incorporate a new functionality; then it would indicate one of the top-level functions of the Scheduler, and the specific module it wants to include. The last input signal (CONF_SIZE_ARRAY) determines the final size of the scalable module to modify, indicating the number of columns and rows. When the reconfiguration process has concluded, the DRM gives back information to the system about the result of the process by

using the signal STATUS_RECONF. Once the DRM receives a reconfiguration request, the Scheduler starts processing according to the strategies and methodologies described in the previous section. Thus, the Scheduler is responsible for sending specific instructions to the Reconfiguration Engine respecting the interface structure shown in Table 5.II.

Table 5.II Interfaces between the Scheduler and the Reconfiguration Engine

RE inputs		Description
Method	ID_TASK	Identify the function to be performed
Data	ID_MODULE	Identify which module will be used
	ID_INSTANCE	Identify which instance of the module will be used
	ID_AREA	Identify which area will be manipulated
	BITSTREAMREF	Bitstream memory address
Result	ID_ERROR	Report status after the task has been executed

The rController, as part of the Reconfiguration Engine, is the unit in charge of transferring information with the Scheduler, and then the Factory sends the bitstreams to the ICAP port, completing in this way the data flow of the system. The interface of the rController to the rest of the system is composed by three main signals (two inputs and one output). The inputs are *Data* and *Method*. These ones are directly related to the information shown in Table 5.II since the ID_TASK identifier corresponds to the *Method* signal, whereas the *Data* signal agglutinates the ID_MODULE, ID_INSTANCE, and the ID_AREA identifiers. More in detail, *Data* is a 16 bits wide signal whose value depends on the invoked *Method*. As for the output *Return* signal, this one is associated to the ID_ERROR or to the ID_INSTANCE identifiers, depending on the performed operation. Thus, all *Methods* return a value indicating that the task has been completed successfully or with errors. The rController registers the reconfigurable components information in a couple of internal tables.

5.6.1.3 Simulation benchmarking

Once the whole platform has been characterized and the data interfaces fixed, it is time to load the SW Scheduler and verify its behavior under a simulated real environment. This process is organized according to the following steps:

1. Configuring the reconfiguration time, and the execution time of all the functionalities, in order to model a real context.
2. Initializing the data memories of the Scheduler for loading the reconfigurable modules information.
3. Data generation and data flow configuration in the platform.
4. Sending reconfiguration requests (from Test block) to the Scheduler.
5. The Scheduler organizes the reconfiguration process and interacts with the rest of the system through the rController.
6. The Scheduler repeats the previous step until completing the reconfiguration process.

A general structure of the developed benchmark is depicted in Figure 5.14.

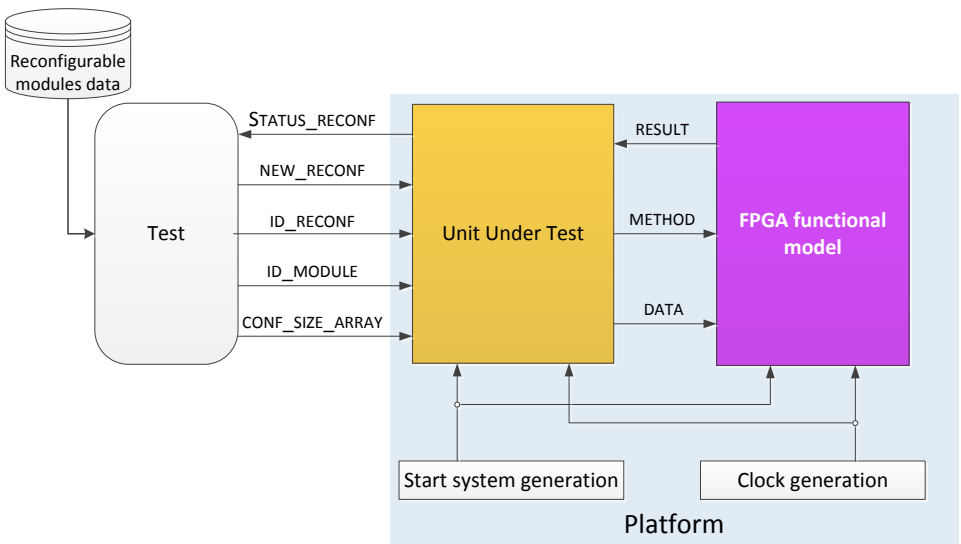


Figure 5.14 Benchmarking for testing the Scheduler behavior

In this particular case, the Unit Under Test (UUT) corresponds to the proposed Scheduler. Then, hardware elements of the embedded system (Reconfiguration Engine, reconfigurable modules and static elements of the application) are transparent to the UUT, and they are modelled and represented by the box named as FPGA functional model.

The behavior of DRM has been tested under different situations, in order to check the Scheduler's decisions and its corresponding responses in cases in which the reconfiguration process might conclude successfully, but also in less favourable situations. Thus, moving from the easiest cases, in which there is enough space in the reconfigurable region for instantiating a new module, the testbenches recreate more complex situations by running one test after another, like if the reconfigurable region was evolving along the time. In this sense, three main test groups are run:

1. Tests related to simple reconfigurable modules. The most significant cases under study have been collected in Table 5.III. First column (*Action*) refers to the reconfiguration request demanded by the system. Second column (*Description*) explains the objective pursued by the reconfiguration, whereas the last column (*Basic function*) describes the main task executed by the Scheduler in order to fulfill with the request.

2. Tests related to scalable reconfigurable modules. Once the Scheduler fills its database with all the information related to the reconfigurable modules, this test (Table 5.IV) checks the behavior of the Scheduler when the main players of the reconfigurability are scalable modules. These scalable modules might conform one dimensional or two dimensional arrays. In the first case all the replicas of the module are placed one after another but always in one direction (horizontally or vertically). Otherwise the array might grow in rows and columns, but once again all the replicas of the module might be placed together, one after another.

3. Tests related to scalable and no scalable modules. Actually, this group of tests combines situations previously tested and described in Table 5.III and Table 5.IV.

Table 5.III Testbench with simple reconfigurable modules

Test	Description		
Action	N	Description	Basic function
Register module	1	First registration of reconfigurable modules with different priorities	Register
	2	Registration of an already registered module	Update

Test		Description		
Action	N	Description		Basic function
Insert instance	3	Insert module in the reconfigurable region and there is enough available areas		
		3.1	The Scheduler looks for reusing an idle instance in the desired area	Activate
		3.2	The Scheduler looks for reusing an instance that was previously used	Reconfigure
		3.3	The Scheduler looks for creating a new instance of the module	Create
	4	Insert a module in the reconfigurable region but there is not available areas since there are other copies of the module already running		ERROR_NO_SPACE
	5	Insert a module in the reconfigurable region but there is not available area because there is another module already instantiated		
		5.1	Requested module has lower priority than the already instantiated	ERROR_NO_SPACE
		5.2	Requested module has higher priority than the already instantiated module	5.2.1 Stop 5.2.2 Create
	6	Insert a module, which occupies more than one micro-area in the reconfigurable region, but those micro-areas are occupied by another modules		
		6.1	All the already instantiated modules have higher priority than the requested module	ERROR_NO_SPACE
		6.2	At least one of the already instantiated modules has higher priority than the requested module	ERROR_NO_SPACE
		6.3	All the already instantiated modules have lower priority than the requested module	6.3.1 Stop 6.3.2 Create
Delete module	7	Remove reconfigurable modules from the system		
		7.1	There are not instances in the reconfigurable region	Delete
		7.2	There is, at least, one instance idle in the reconfigurable region	Delete
		7.3	There is, at least, one instance running in the reconfigurable region	7.3.1 Stop 7.3.2 Delete

Table 5.IV Testbench with scalable modules

Test		Description	
Action	N	Description	Basic function
Create an array	1	Create a new array in the reconfigurable region 3×3 from the scratch (there is not any instance in the reconfigurable region)	Expand_array
Reduce the array	2	Decrease the size of the previous array from 3×3 to 1×2	Reduce_array
Increase the array	3	Include new instances in the array from 1×2 to 4×5	Expand_array
		3.1 Reusing some of the first instances created for the initial array (3×3)	Activate
		3.2 Introducing new instances until complete the requested size (from 3×3 to 4×5)	Create
Create an array	4	Create a new array but some of the required areas are occupied	
		4.1 Requested scalable module has lower priority than the already instantiated modules. Then, the final array	Insert_rows / Insert_columns

Test		Description		
Action	N	Description	Basic function	
		will reach an intermediate size (growing in the direction that gets the highest number of instances)		
	4.2	Requested scalable module has higher priority than the already instantiated modules	4.2.1	Stop
			4.2.2	Expand array
Reduce an array	5	Remove an array from the reconfigurable region (stop all the instances of the array but not removing the scalable module from the system)	Stop	
Delete module	6	Remove a scalable module from the system		
	6.1	There are not instances in the reconfigurable region	Delete	
	6.2	There is, at least one instance in the reconfigurable region	6.2.1	Stop
			6.2.2	Delete

5.6.2 FPGA-based embedded system

This section is focused on the implementation of a dynamically reconfigurable application on FPGA Xilinx V5-LX100T, in which the reconfiguration process is managed by the DRM according to the environmental requirements. In this case, as a simplification, the reconfiguration requests follow a battery of tests that specify to the DRM when and how to reconfigure the running application.

5.6.2.1 Dynamically scalable hyperspectral linear unmixing application

The start point is the architecture referred as `PixelSA_MVCA`, presented in Chapter 4. The exploitation of the dynamic reconfigurability requires modifying the architecture in order to maximize the exploitation of characteristics like the flexibility, and the reallocation of reconfigurable modules. One of the main challenges is the placement and routing of the design on an FPGA, since the communication lines between modules and between the static and the reconfigurable modules have to be fixed independently of the level of the scalability at run time. Furthermore, due to the connectivity lines within an FPGA vary notably between horizontal resources and vertical resources, the designer/developer should ensure the shape of the reconfigurable modules, and also minimize the number of data wires. These are the main aspects that have been considered before modifying the `PixelSA_MVCA` design, such as Figure 5.15

shows. Thus, the scalable version depicted in Figure 5.15.a might vary its number of PEs statically, or even using a total dynamic reconfiguration of the FPGA, since the routing paths have to be adapted every time that a PE is inserted or removed from the reconfigurable region. On the contrary, the architecture shown in Figure 5.15.b is regular in terms of routing, since all the data go through the same channels. This last structure makes easier the exploitation of the dynamic and partial reconfiguration (DPR). Notice that for getting this regularity, two new modules have been incorporated to the architecture: the distributer and the dispenser; both of them routing elements. In addition, the dispenser also includes some basic control logic in order to distribute the input data (coming from the north) between its corresponding PE or its bottom neighboring dispenser.

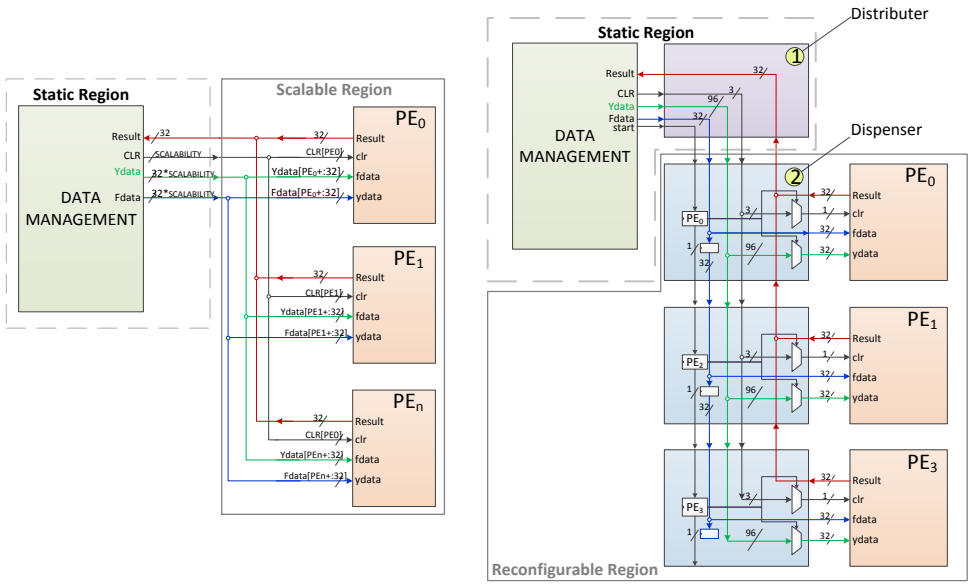


Figure 5.15 Block diagrams of the application; a) `Scalable PixelSA_MVCA` design; b) `Reconfigurable PixelSA_MVCA`

Comparing the dynamically reconfigurable `PixelSA_MVCA` (Figure 5.15.a) with the DPR `PixelSA_MVCA` version (Figure 5.15.b), neither their behavior not their number of clock cycles for extracting an endmember vary between them. The disparities among these implementations lie on the number of hardware

resources and the clock frequency, once they are synthesized onto a Xilinx V5-LX110T FPGA, such as Table 5.V represents.

Table 5.V Synthesis results of a whole MVCA architecture with 1PE onto a Xilinx V5-LX110T

Logic Resources	Scalable PixelSA_MVCA			Reconfigurable PixelSA_MVCA		
	1 PE	2 PE	3 PE	1 PE	2 PE	3 PE
Slice Registers	14,632	14,984	15,320	14,249	14,674	14,978
Slice LUTs	11,282	11,591	11,923	10,993	11,294	11,597
DSP48E	16	20	24	16	20	24
Clock cycles for extracting 1 endmember	6,501	3,264	2,185	6,501	3,264	2,185
Freq. (synthesis / Post P&R)	284.33 MHz / 156.76 MHz			90.63 MHz		
Time for extracting 1 endmember	22.86 ns / 41.47 ns	11.47 ns / 20.82 ns	7.68 ns / 13.93 ns	71.73 ns	36.01 ns	24.10 ns

Attending to the data collected on Table 5.V, the direct consequence of moving from a dynamically scalable design (`Scalable PixelSA_MVCA`) to a dynamically and partial reconfigurable one (`Reconfigurable PixelSA_MVCA`) is the reduction of the clock frequency. Therefore, the endmembers extraction process is slower in the `Reconfigurable PixelSA_MVCA` than the `Scalable PixelSA_MVCA`, despite the fact that both of them require the same number of clock cycles. More specifically, the `Scalable PixelSA_MVCA` is 3.13 times faster than the `Reconfigurable PixelSA_MVCA` for all the scalability levels. Even though, the `Reconfigurable PixelSA_MVCA` architecture might be used under real time constraints for hyperspectral endmember extraction, considering the fact that the AVIRIS sensor is able to collect 512 hyperspectral pixels in 8.3 ms, as it was referred in Chapter 4.

5.6.2.2 Embedded system framework and benchmarking

In order to consider all the aspects presented along the previous chapters, this subsection joins all those concepts (scalability, dynamic reconfigurability and resource management) together, by means of integrating them as part of the same FPGA-based embedded system. Due to the scope of this Thesis is to demonstrate the viability of using scalable DPR designs under unpredicted environments, the demonstrator simplifies the solution as much as possible. As a result, the proposed embedded system integrates the proposed DRM, and the

Reconfigurable PixelSA_MVCA architecture in order to create a dynamic hyperspectral system.

Before integrating the whole system, hardware and software together, it is recommended to verify the correct behavior of all the elements onto the FPGA separately. This is the best way to constrain future fails as much as possible, mainly to the communication interfaces among all the elements and their synchronization. Despite the fact that the Scheduler, and the whole DRM interfaces, has been verified through the co-simulation platform (Section 1.6.1.3), it is turn to verify the behavior of the Reconfigurable PixelSA_MVCA onto the FPGA.

Reconfigurable PixelSA_MVCA integration: A static version of the application is implemented and run. In this stage two executions have been run. The first one used the lowest scalability level of the application (one PE); whereas the second one enables the highest scalability level of the application (three PEs). The comparison between both execution results allows to detect any anomaly in the behavior of the system.

From a higher abstraction level, the proposed benchmarking for verifying the hardware architecture is represented in Figure 5.16. The Reconfigurable PixelSA_MVCA architecture is represented by a black box named HW MVCA, and it is directly connected to a communication controller (FSL_Ctrl), in charge of initializing and activating the MVCA through the scalability, n_endmembers and start signals. Then, once the MVCA extracts the index of the winner pixel the FSL_Ctrl receives the information and forward it to the microprocessor.

The soft embedded microprocessor (Microblaze) of the structure depicted in Figure 5.16 controls the configuration of the system (the scalability level and the number of endmembers to extract), and it sends the results of the MVCA out by using the serial port (UART – Universal Asynchronous Receiver-Transmitter). The simplicity of this design has been the cause why the communication between the microprocessor and the FSL_Ctrl has been implemented

using a Fast Simple Link (FSL bus). Moreover, the communication between the microprocessor and the rest of elements of the system is using the Processor Local Bus (PLB).

In order to ensure an appropriate behavior of the system, the system has been configured as follows: clock frequency to 100 MHz, `n_endmembers` to 5, `scalability` varies between 1 and 3, and the input corresponds with the small synthetic image of 36×36 pixels and 5 endmembers used previously for running simulations.

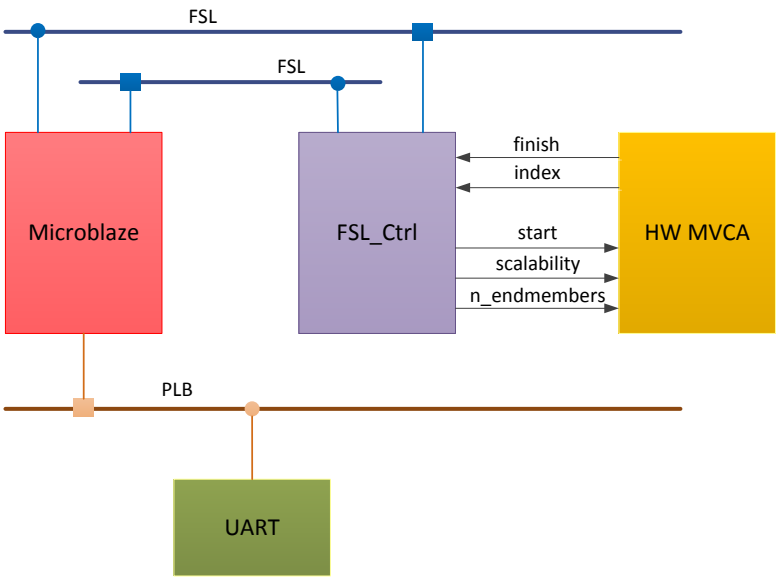


Figure 5.16 System structure of the MVCA

As it was expected, under the three scalability configurations tested, the results (the number of clock cycles required for extracting an endmember, and the winner indexes) are the same when the `Reconfigurable PixelSA_MVCA` is loaded and run onto the FPGA than when it is simulated. Thus, the values collected in Table 5.V are the same for both scenarios.

System integration: Once the behavior of the proposed reconfigurable design has been verified when it runs statically, the next step is to repeat the same but at this time changing its scalability dynamically. That means, first of all, separating the static elements of the `Reconfigurable PixelSA_MVCA` to the

reconfigurable ones and generating their corresponding partial bitstreams. The next step consists of adding complexity to the embedded system in order to incorporate the tools for managing the reconfiguration process of the system. That is, to include the proposed DRM module (the Scheduler and the Reconfiguration Engine).

For every different reconfigurable module of the design, there is to designate a region on the FPGA where it can be placed and routed. This is an easy process with the help of PlanAhead tool [XPAH]. Due to the fact that the proposed experiment will only replicate the same kind of element, it has been considered that the `Reconfigurable PixelSA_MVCA` only has one reconfigurable module (the combination between the dispenser and the PE). Then, it is time to describe the structure of the Reconfigurable Region. The number of areas within the RR, their distribution and size will depend on two main aspects: the logic resources distribution of the selected FPGA (not all the families disseminate the columns of DSPs, CLBs and BRAMs in the same way), and the requirements of the reconfigurable modules in terms of resources. As an example, the proposed architecture uses DSPs in its PEs; therefore their reconfigurable areas (pblock_pr1n in Figure 5.17) must be placed onto the same column of DSPs provided in the V5-LX110T.

Every one of these reconfigurable regions, named as pblock_pr1n in Figure 5.17, is composed by 320 look-up-tables, 320 flip-flops, 50 Slices L-type, 30 Slices M-type and 4 DSPs.

The main differences between the final FPGA-based embedded system, represented in Figure 5.18, and the one used for verifying the architecture statically are the inclusion of the Reconfiguration Engine (depicted as HW RE), the memory controller (MPMC – Multi Port Memory Controller) and the communication bus among them (NPI – Native Port Interface). The RE needs the MPMC in order to load the configuration bitstreams from the configuration memory. Moreover, the ICAP port has been incorporated as part of the RE since it is under control of the latter. As for the Scheduler, the top layer of the DRM, it runs

on the microprocessor (Microblaze) because of it was described in software. Then, the rest of modules are purely hardware.

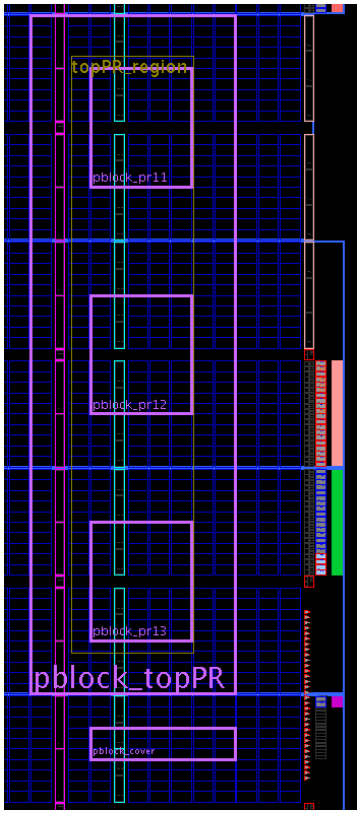


Figure 5.17 Reconfigurable Region structure and reconfigurable modules distribution

More specifically, the Reconfiguration Engine and some parts of the Reconfigurable PixelSA_MVCA design are implemented as static modules in hardware, because they never modify neither their number of elements, nor their performance. However, the number of reconfigurable modules (FU_n) of the Reconfigurable PixelSA_MVCA might vary according to the required scalability level. That is the reason why those units must be instantiated into the dynamically reconfigurable region, which in this case has been structured as a one dimensional matrix of 1×3 micro areas.

Regarding hardware aspects of the proposed system (Figure 5.18), Table 5.VI shows its post place and route summary reported by the Xilinx Platform

Studio tool. The information has been organized in terms of the used flip-flops (FFs) and look-up-tables (LUTs) resources for the most relevant blocks, such as the `Reconfigurable PixelSA_MVCA` (`Static HW MVCA` and `Dynamically Reconfigurable Region`), the Reconfiguration Engine (HW RE) and the serial port module (UART). These numbers are obtained when the clock frequency of the system has been configured to 100MHz.

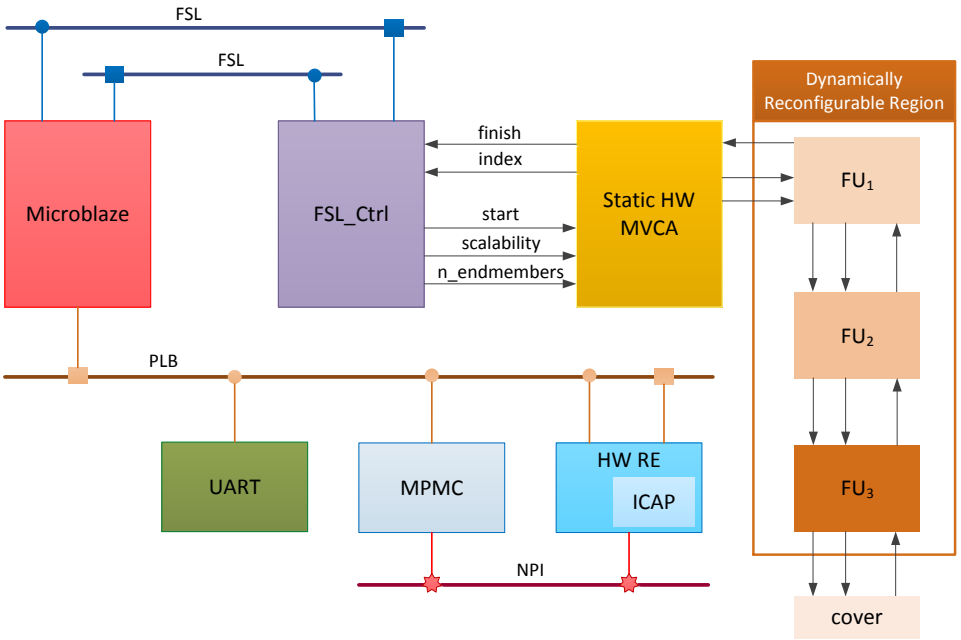


Figure 5.18 Embedded system structure

As a consequence of being static blocks, the Reconfiguration Engine and the UART blocks always occupy the same number of resources, independently of the scalability of the system. The MVCA, and therefore, the whole system require more or less resources depending on the number of FUs configured on the system.

Table 5.VI Post Place and Route summary on Xilinx V5-LX110T

Blocks	FFs used			LUTs used		
	1 FU	2 FU	3 FU	1 FU	2 FU	3 FU
System	20,148	20,514	20,882	15,254	15,554	15,945
MVCA	5,477	5,809	6,143	4,350	4,650	4,951
RE	6,371	6,371	6,371	2,102	2,102	2,102
UART	148	148	148	131	131	131
	60%	60%	61%	43%	44%	45%

According to the information collected in Table 5.VI, the number of hardware resources of the MVCA, RE and UART blocks corresponds to the 60% and 44% of the FFs and LUTs resources of the system, respectively. More in detail, the reconfigurable architecture (MVCA block) assumes, in average, the 28% and the 30% of the flip-flop and LUTs required in the system, respectively. The other 40% of the hardware resources of the system are due to the implementation of the microprocessor and the other internal controllers required for handling the buses.

As for the reconfigurability management, in the beginning the system is loaded with the simplest configuration (one PE). At that point, the DRM has to manage different situations and reconfigure the system. Sometimes, the reconfiguration request demands a higher scalability (two or three PEs). Other times the DRM has to reduce the number of PEs. In order to verify the correct behavior of the system, the `Reconfigurable PixelSA_MVCA` always has to process the same image. Therefore, the extracted endmembers have to be always the same, independently of the number of FUs configured by the DRM. Thus, under these scenarios the indexes of the winner endmembers are shown in the following Figure 5.19 Winner endmembers, which represents a screenshot of the serial port communication.

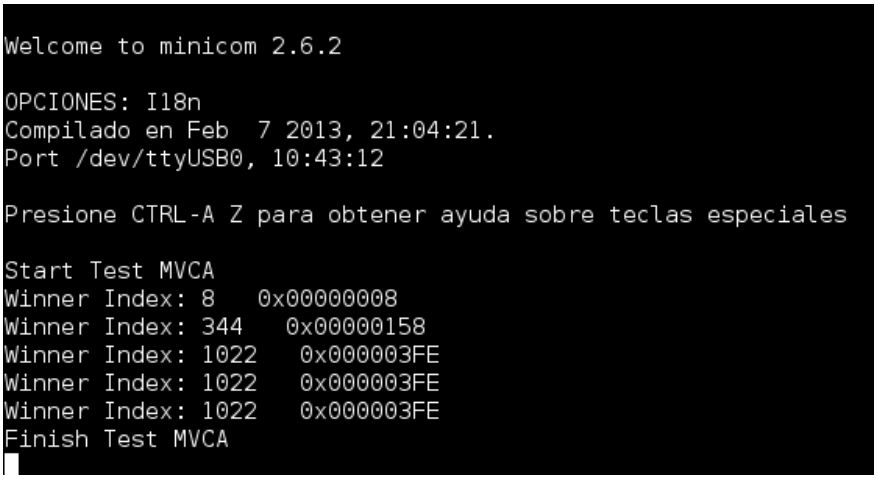


Figure 5.19 Winner endmembers' indexes for the 36×36 hyperspectral image

Concerning the reconfiguration time, the worst possible case is the one in which the whole configuration bitstream has to be loaded on the FPGA. That means placing and routing all the static and the reconfigurable blocks of the whole design. In this situation, the `Reconfigurable PixelSA_MVCA` only includes one functional unit, since this is its basic and smallest configuration. Thus, the highest size of the configuration bitstream is 3,70Mbytes. Then, considering that the throughput of the Reconfiguration Engine is 180Mbits/second, the slowest configuration time is up to 160 ms.

5.7 CONCLUSION

The development of dynamically reconfigurable FPGA-based embedded SoCs is a complex issue. The main reason is due to the absence of mechanisms for controlling the reconfiguration process automatically, without external intervention. In this sense, this chapter presents a dynamic reconfiguration management system (DRM), consisting of a mixed hardware-software framework that transparently controls the reconfiguration process by scheduling and loading partial bitstreams on the reconfigurable region. One of the novelties of this DRM, compared to previous research works, is its capability for handling traditional reconfigurable modules and scalable ones.

As a proof of concepts, an FPGA-based embedded system has been developed, in which the DRM has been integrated in order to lead the reconfigurability of the `Reconfigurable PixelSA_MVCA` architecture. The goal of this demonstrator is to consolidate and validate several aspects presented along this PhD research work:

1. The advantages of designing scalable designs are superior to their disadvantages for dynamic scenarios.

2. It is possible to conceive a new paradigm for dynamic reconfigurability on Xilinx FPGAs, where the reconfigurable region might be more flexible than traditional contexts.

3. It is possible to design autonomous or semi-autonomous embedded systems capable of managing the variations of a scalable design dynamically by means of using the proposed DRM.

Chapter 6

Conclusions

The collection of the contributions provided in this PhD, but also their relevance into the dynamic reconfigurability field, are summarized. Then, at the end of this document, further research works are proposed, which might complement and enhance some of the aspects studied along the development of this PhD work.

6.1 CONCLUSIONS

This PhD research work has reviewed the most significant previous works regarding the dynamic reconfigurability, including different architectural proposals for high performance applications, low-level assembly methodologies, and management solutions for controlling the variations of the system at run-time. This review has demonstrated that there are still some lacks within the dynamic reconfigurability field regarding the tools, methodologies and management.

With the goal of overcoming the aforementioned lacks, this PhD has focused its efforts in several aspects of the dynamic reconfigurability. First of all, a review of the state-of-the-art of reconfigurable architectures on data intensive applications has been analyzed. Then, with the results extracted after the analysis, a set of contributions have been proposed. In this sense, in order to contribute in the design of more flexible systems, capable to adapt their performance to environmental variations, part of this research has been focused on exploiting the advantages of the scalability over several kinds of applications. As a result two different applications have been selected: a deblocking filter algorithm, as part of H.264/AVC and SVC video codecs standards, and the MVCA algorithm, as part of an endmember extraction chain within the hyperspectral imaging analysis. Both applications have several characteristics in common. The most important ones are their high computational cost, in terms of number of operations, and the amount of time they require for finishing processing their data. Furthermore, both algorithms provide a high level of parallelism that is a desirable feature in order to design efficient scalable designs. Despite their common characteristics, the differences between both algorithms make their solutions completely different. Whereas the DF algorithm is based on simple arithmetic operations and its high data dependences, the main operation of the MVCA is the multiplication, in which there is a low dependence among data. Moreover, the data dependences After demonstrating the viability of designing scalable designs for these two high performance computing applications, this feature has been exploited at run time by means of taking advantage of the dynamic recon-

figurability. Therefore, the scalable DF and one of the scalable MVCA solutions have been adapted, and implemented on an FPGA-based embedded system. On the other hand, a dynamic resource manager has been designed with the goal of being responsible for controlling the whole hardware reconfiguration process, as part of context-aware embedded systems. Finally, all these contributions have been combined in order to prove the reliability of all these works. Therefore, one of the proposed scalable designs have been running on the FPGA, and scaled dynamically by the proposed DRM without the intervention of any external device.

As a brief summary, the contributions of this PhD research are the following ones:

- An extended review of reconfigurable architectures focused to execute several computationally intensive applications.

- A novel parallelization strategy for the H.264/AVC deblocking filter. Despite the fact that in the present PhD this improved wavefront has been used for parallelizing macroblocks (MBs) data, it is not constrained to this scenario. Therefore, it might be used in several fields whether the data dependences are respected.

- A novel scalable architecture for executing the deblocking filter of the H.264/AVC and the SVC codecs has been proposed. The structure of this design introduces a high level of flexibility into the DF processing, since the final performance might be adapted to different conditions, just adjusting the number of processing elements. Furthermore, this structure might be reused for executing diverse kinds of tasks that fulfill with specifics restrictions related to the data dependences.

- The strength of the scalability has been used on the hyperspectral imaging field through the design of a couple of scalable designs for processing the MVCA algorithm. The main success of these proposals is based on the fact that is a completely unexplored feature in this research field. The results demon-

strate the viability of these kinds of designs for processing high computational demands on hyperspectral imaging applications.

- New assembly methodologies have been proposed in order to facilitate the reallocation of modules in different regions of an FPGA, as part of a dynamically reconfigurable embedded system.

- Designing a controlling and management mechanism for ensuring a correct dynamically reconfiguration process on embedded systems, based on online scheduling strategies and improved techniques for manipulating bitstreams.

- Proposing a new paradigm for reconfigurable regions in which the reconfigurable elements might compete for logic resources with a higher level of flexibility than traditional concepts. This fact is shown through the design of a 2D array of micro areas that might be joined with its closest neighbors in order to place bigger modules at run-time.

- Developing a demonstrator capable to collect on the same system all the contributions reached in this PhD work.

6.2 FURTHER RESEARCH

This work opens the window to new research lines focused on going deeper into some of the proposed contributions. In this sense, several improvements might be done, some of them directly related to the design and development of scalable architectures; and others related to the management of the reconfiguration process itself.

- Despite the fact that the proposed scalable DF has been implemented and validated on a Virtex-5 board, it would be very interesting to integrate it as part of a whole H.264/AVC encoder or decoder.

- Exploring the scalability benefits on other kind of applications intensive in computation, by following the same principles than the ones used in this PhD work, such as flexibility, modularity, regularity (related to the behavior and connectivity of every module) and parallelization of the operations.

- Regarding the management of the reconfiguration process, it would be very interesting working on including certain level of intelligence to the scheduler, in order to be more independent from the rest of the system, but also from the microprocessor.

  - Designing a more complex cost function which includes implicitly a fourth dimension (considering the other three: number of logic resources, physical area, and priority), such as the reconfiguration time.

  - Being able to collect information regarding the running state of the system, but also from the running environment. Therefore, the scheduler would take decisions about what, when and where to reconfigure the hardware without the microprocessor intervention.

  - Being able to swap modules from hardware to software, or vice versa, according to two factors: the performance of the overall system, and the persistence of the reconfigurable module. In this context, the persistence analyzes the history of the current reconfigurable module during the system execution (how many times has been required, how many times has been running in hardware, etc).

  - The scheduler could manage different kinds of modules regarding the scalability, since a scalable design might require replicating different type of modules when the level of parallelism is to be increased.

References

[AA12] O. Atak, A. Atalar; "BilRC: An Execution Triggered Coarse Grained Reconfigurable Architecture," Very Large Scale Integration (VLSI) Systems, IEEE Trans. on, vol. PP, no.99, pp.1-14, 2012 (Accepted for publication).

[AAS12] A. Al-Wattar, S. Areibi, F. Saffih; "Efficient On-line Hardware/Software Task Scheduling for Dynamic Run-time Reconfigurable Systems," IEEE Int. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp.401-406; 2012.

[AJM+09] A. Azevedo, B. Juurlink, C. Meenderinck, A. Terechko, J. Hoogerrugge, M. Alvarez, A. Ramirez, M. Valero; "A Highly Scalable Parallel Implementation of H.264"*;* Trans. on High-Performance Embedded Architectures and Compilers (HIPEAC), vol. 4, Issue 2, pp. 1 – 25, 2009.

[AM09] Z.E.A. Alaoui Ismaili, A. Moussa; "Self-partial and dynamic reconfiguration implementation for AES using FPGA"; Int. Journal of computer science issues, vol.2, pp.33-40, 2009.

[AMS+02] J. Anvik, S. MacDonald, D. Szafron, J. Schaeffer, S. Bomling, K. Tan; "Generating parallel programs form the wavefront design pattern"; Int. Parallel and Distributed Process Symposium, vol.2, pp. 1-4, 2002.

[ARCO] Grupo de investigación de Arquitectura y Redes de Computación (ARCO). Available at: arco.esi.uclm.es/es.

[ASOC13] Altera Corp., Alteraa SoCs: when architecture matters, available at: www.altera.com/devices/processor/soc-fpga/overview/proc-soc-fpga.html

[ATOM13] Intel Corp., Atom Processor; available at: ark.intel.com/es-es/products/codename/42360

[AVI07] Airbone Visible InfraRed Imaging Spectrometer (AVIRIS), available at: aviris.jpl.nasa.gov

[BBD09] S. Banerjee, E. Bozorgzadeh, N. Dutt; "Exploiting application data-parallelism on dynamically reconfigurable architectures: placement and architectural considerations," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol.17, no.2, pp.234-247, 2009.

[BCR+06] Brunelli, C.; Cinelli, F.; Rossi, D.; Nurmi, J., "A VHDL model and Implementation of a Coarse-Grain Reconfigurable Coprocessor for a RISC Core," *Research in Microelectronics and Electronics 2006, Ph. D.*, pp.229-232, 2006.

[BEM+03] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M Vorbach, M. Weinhardt, "PACT XPP - A self-reconfigurable data processing architecture", Journal of Supercomputing, 26, pp. 167-184, 2003.

[BG01] J. Becker, M. Glesner "A parallel dynamically reconfigurable architecture designed for flexible application-tailored hardware/software systems in future mobile communication" Journal of supercomputing, vol.19, pp. 105-127, 2001.

[BG99] M. Budiu; S. C. Goldstein; "Fast compilation for pipelined reconfigurable fabrics", ACM; pp.195 – 205, 1999.

[BGN08] C. Brunelli, F. Garzia, J. Nurmi, "A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilites", Special Issue, Real-time image Proc., pp. 21-32, 2008.

[BHH+07] J. Becker; M. Hubner; G. Hettich; R. Constapel; J. Eisenmann; J. Luka; "Dynamic and Partial FPGA Exploitation," Proceedings of the IEEE, vol.95, no.2, pp.438-452, 2007.

[BKG95] J. Boardman, F.A. Kruse, R.O. Green; "Mapping target signatures via partial unmixing o AVIRIS data", in Summaries of the V JPL Airbone Earth Science Workshop, vol.1, pp.23-26, 1995.

[BLM+04] M. Bedford, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, "Evaluation of

the RAW microprocessor: an exposed-wire-delay architecture for ILP and Streams", Proc. 31st ISCA, pp. 2-13, 2004.

[BN08] J.M. Bioucas-Dias, J.M.P. Nascimento; "Hyperspectral Subspace Identification", IEEE Trans. on Geoscience and Remote Sensing, vol.46, no.8, pp.2435-2445, 2008.

[Boa12] R. Boada; "Trends and patterns of ASIC and FPGA use in European space missions and impact in technology roadmaps or the European Space Agency"; MS. Thesis developed and evaluated by T.U. Delft and ESA; pp. 1-94, 2012.

[Boa94] J. W. Boardman; "Geometric mixture analysis of imaging spectrometry data", Proc. Int. Geoscience and Remote Sensing Symposium, vol. 4, pp. 2369-2371, 1994.

[BP11] J.M. Bioucas-Dias, A. Plaza; "An overview on hyperspectral unmixing: Geometrical, statistical, and sparse regression based approaches," IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS), pp.1135-1138, 2011.

[BPD+12] J.M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, J. Chanussot; "Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches"; IEEE J. Sel. Top. Appl. Earth Observation Remote Sensing; vol.5, n.2, pp. 354-379, 2012.

[CAC09] G. Correa, L. Agostini, L. Cruz; "A fast FPGA implementation of the inter-layer deblocking filter for H.264/SVC"; Conf. on Telecommunications, pp. 1-4, 2009.

[CCC10] H.-C. Chung, Z.-Y. Chen, P.-C. Chang; "Low Power architecture design and hardware implementations of deblocking filter in H.264/AVC"; IEEE Trans. On Consumer Electronic, vol.57, no.2, pp. 713 – 719, 2010.

[CCG09] C.-A. Chien, H.-C. Chang, J.-I. Guo; "A high throughput deblocking filter design supporting multiple video coding standards"; IEEE Int. Symposium on Circuits and Systems (ISCAS), pp. 2377-2380, 2009.

[CCH06] C.-C. Cheng, T.-S. Chang, K.-H. Lee; "An in-place architecture for

deblocking filter in H.264/AVC"; IEEE Trans. on Circuits and Systems II, vol. 53, no. 7, pp. 530-534, 2006.

[CD04] C.-I. Chang, Q. Du; "Estimation of number of spectrally distinct signal sources in hyperspectral imagery", IEEE Trans. on Geoscience and Remote Sensing, vol.42, no.3, pp. 608-619, 2004.

[CD04] C.-I. Chang, Q. Du; "Estimation of number of spectrally distinct signal sources in hyperspectral imagery"; IEEE Trans. on geoscience and remote sensing, vol. 42, pp. 608-619, 2004.

[CFF+99] D.C. Cronquist, C. Fisher, M. Figueroa, P. Franklin, C. Ebeling; "Architecture design of reconfigurable datapaths"; pp. 1-18; 1999.

[Che10] K.-H. Chen; "48 cycles-per-macroblock deblocking filter accelerator for high-resolution H.264/AVC decoding"; IET. Circuits, Devices & Electronic, vol.4, no.3, pp. 196 – 206, 2010.

[CJC+12] C-A Chien, G-A Jian, H-C Chang, K-H Chen, J-I Guo; "High efficiency data access system architecture for deblocking filter supporting multiple video coding standards"; IEEE Trans. on Consum. Electron., vol. 58, no.2, pp. 670-678, 2012.

[CLF08] E. Cantó, M. López, F. Fons; "Self-reconfiguration of embedded systems mapped on Spartan-3", Reconfigurable communication-centric SoC, pp. 117-124, 2008.

[CM07] V. Ciric, I. Milentijevic; "Area-time tradeoffs in H.264/AVC deblocking filter design for mobile devices"*;* Int. Symposium on Signal Processing and Its Applications, pp. 1-4, 2007.

[CMN+09] S. Corbetta, M. Morandi, M. Novati, M.D. Santambrogio, D. Sciuto, P. Spoletini; "Internal and External Bitstream Relocation for Partial Dynamic Reconfiguration," EEE Trans. on Very Large Scale Integration (VLSI) Systems, I, vol.17, no.11, pp.1650-1654, 2009.

[CRG+11] J.A. Clemente, J. Resano, C. Gonzalez, D. Mozos; "A Hardware Implementation of a Run-Time Scheduler for Reconfigurable Systems," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol.19, no.7, pp.1263-1276, 2011.

[CRS+11] J.A. Clemente, V. Rana, D. Sciuto, I. Beretta, D. Atienza; "A Hybrid Mapping-Scheduling Technique for Dynamically Reconfigurable

Hardware," Int. Conference on Field Programmable Logic and Applications (FPL), pp.177-180, 2011.

[CXL10] X. Chen, W. Xia, X. Lu; "A high-throughput low-power hardware architecture for H.264 deblocking filter"; IEEE Int. Conf. On Computer Engineering and Technology, vol.2, pp. 561 – 565, 2010.

[CZF+08] Q. Chen, W. Zheng, J. Fang, K. Luo, B. Shi, M. Zhang, X. Zhang; "A pipelined hardware architecture of deblocking filter in H.264/AVC"*;* Int. Conf. on Communications and Networking in China, pp. 815-819, 2008.

[DBB+13] M. Duranton, D. Black-Schaffer, K. de Bosschere, J. Maebe; "The HiPEAC vision for advanced computing in horizon 2020", FP7 HiPEAC Network of Excellence, pp.1-48, 2013.

[DH03] S. Donthi, R.L. Haggard: "A survey of dynamically reconfigurable FPGA devices," Proceedings of the 35th Southeastern Symposium on System Theory, pp.422-426, 2003.

[DP11] O. Duran, M. Petrou; "Applicability of robust unconstrained linear unmixing (RULU) to endmember extraction techniques," Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), pp.1-4, 2011.

[Dub09] Dubey, R.; "Introduction to embedded system design using field programmable gate arrays", Ed. Springer ISBN: 978-1-84882-015-9, pp. 1-153, 2009. Available at: www.springer.com

[EBS+11] H. Espeland, P.B. Beskow, H.K. Stensland, P.N. Olsen, S. Kristoffersson, C. Griwodz, P. Halvorsen; "P. P2G. A framework for distributed real-time processing of multimedia data"; Conference on Parallel Processing Workshop, pp. 416-426, 2011.

[Eec07] H. Eeckhaut; "Scalable hardware for scalable video", Thesis work; University of Gent; pp. 1-205; 2007.

[EFX+04] C. Ebeling, C. Fisher, G. Xing, M. Shen, H. Liu, "Implementing an OFDM Receiver on the RaPiD reconfigurable architecture", IEEE Trans. on Computers, issue 11, pp. 1436-1448, 2004.

[Ern07] E.G. Ernst; “Architecture design of a scalable adaptive deblocking filter for H.264/AVC”; MSc Dissertation, Rochester, NY, 2007.

[FFC+11] F. Fons, M. Fons, E. Cantó, M. López; “Real-time embedded systems powered by FPGA dynamic partial self-reconfiguration: a case study oriented to biometric recognition applications”, Journal of real-time image processing, special issue, pp. 1-23, 2011.

[FLG12] A. Fernández-León, B. Glass; “ESA strategy: ASICs and FPGAs for space”, ESA TEC-EDM, pp. 1-61, 2013. Available at: indico.cern.ch/getFile.py/access?resId=0&materialId=slides&confId=237564

[FVM+11] R. Ferreira, J.G. Vendramini, L. Mucida, M.M. Pereira, L. Carro; "An FPGA-based heterogeneous coarse-grained dynamically reconfigurable architecture," Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011 Proceedings of the Int. Conference on, pp.195-204, 2011.

[GBI11] GBI Research; “Field programmable gate array (FPGA) market to 2020- Increasing preference for FPGA over application specific integrated circuits (ASICs) will drive demand”; Ref. code: GBISC0012MR; pp. 1-107; July 2011.

[GMR+12] C. González, D. Mozos, J. Resano, A. Plaza; "FPGA Implementation of the N-FINDR Algorithm for Remotely Sensed Hyperspectral Image Analysis," Geoscience and Remote Sensing, IEEE Trans. on , vol.50, no.2, pp.374-388, 2012.

[Gor00] C. Gordon, "A generalization of the maximum noise fraction transform", Geoscience and Remote Sensing, IEEE Trans. on, vol.38, no.1, pp.608-610, 2000.

[GP95] M. Girkar, C.D. Polychronopoulos; “Extracting task-level parallelism”, ACM Trans. On programming languages and systems, vol.17, no.4, pp. 600-634, 1995.

[GRE+98] R.O. Green, M.L. Eastwood, C.M. Sarture, T.G. Chrien, M. Aronsson, B.J. Chippendale, J.A. Faust, et.al.; “Imaging Spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)”, Remote Sensing of Environment, vol.65, Issue 3, pp. 227-248, 1998.

[HAA+12] F. Hilton, R. Armante, T. August, C. Barnet, A. Bouchard, C. Camy-Peyret, V. Capelle, L. Clarisse, C. Clerbaux, P-F Faijan, and coauthors; "Hyperspectral Earth observation from IASI. Five years of accomplishments", Bulletin of the American Meteorological Society, vol.93, no.3, pp. 347-370, 2012.

[HCE07] S. Heitchecker, A. Carmo, R. Ernst, "A high-end real-time digital film processing reconfigurable platform" Journal on Embedded Systems, vol. 2007, pp.1-15, 2007.

[HCH03] Y.W. Huang, T.W. Chen, B.Y. Hsieh, T.C.Wang, T.H. Chang, L.G. Chen. "Architecture design for deblocking filter in H.264/JVT/AVC"; IEEE Int. Conference on Multimedia and Expo, 2003.

[HJK+03] M. Horowitz; A. Joch; F. Kossentini; A. Hallapuro. *H.264/AVC baseline profile decoder complexity analysis*; IEEE Trans. on Circuits and Systems for Video Technology, vol.13, no.7, pp.704-716, 2003.

[HO10] M. Heimlicher, M. Oberholzer; Enclustra GmbH - "FPGA Technology and industry experience"; Guest Lecture in the context of the VLSI III course at D-ITET, ETH Zurich; pp. 1-52; 2010.

[HR12] J. Hasting, C. Rommel; "Strategic insights 2012: Embedded hardware and systems. Market opportunities and forecasts in 2011-2016.", Embedded hardware & systems practice, Executive Brief VDC Research Group Inc.; vol. 5, Track2: embedded processing technologies; pp. 1-11; 2012.

[HSC+10] S. Haynes, J. Stone, P. Cheung, W. Luk, "Video Image Processing with the Sonic Architecture", Configurable computing, pp. 50-57, 2000.

[HSM03] P. Heysters, G. Smit, E. Molenkamp, "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems", The Journal of Supercomputing, nº26, pp. 283-308, 2003.

[IEE08] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754,

2008.

[INS12] National Instruments; Report “Embedded Systems Outlook 2012. Key technologies and methodologies impacting the embedded systems market.”; pp. 1-20; 2012; available at: www.ni.com/embeddedsystems

[INS13] National Instruments; Report “Embedded Systems Outlook 2013. Significant trends, opportunities, and challenges.”; pp. 1-16; 2013; available at: www.ni.com/eso

[ITU11] Int. Telecommunication Union (ITU); “Measuring the information Society. 2011.”Ed. by ITU, pp. 1-174, ISBN: 92-61-13801-2, 2011.

[ITU-T07] ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC; “H.264/AVC Standard Advanced Video Coding for Generic Audiovisual Services”, Version 1: 2003 – Version 7: 2007.

[ITU-T09] ITU-T Rec. H.264 and ISO/IEC 14496-10. *H.264/AVC extension* (Scalable Video Coding - SVC). Advanced Video Coding for Generic Audiovisual Services, Version 8: 2007 – Version 10: 2009.

[Jak13] V. Jakhanwal; Report “Flexible display market to reach nearly 800 million unit shipments by 2020”; HIS Electronics & Media; 2013. Available at: www.isuppli.com/Display-Materials-and-Systems/News/Pages/FlexibleDisplayMarkettoReachNearly800MillionUnitShipmentsby2020.aspx

[JG11] A. Jara-Berrocal, A. Gordon-Ross; "Hardware module reuse and runtime assembly for dynamic management of reconfigurable resources," Int. Conference on Field-Programmable Technology (FPT), pp.1-6, 2011.

[JLY+10] W. Jia, L. Liu, S. Yin, M. Zhu, Z. Wang; “A fast complete deblocking filter on a coarse-grained reconfigurable processor supporting H.264 high profile decoding”; Pacific Conf. on Postgraduate Research on Microelectronics and Electronics (PrimeAsia), pp. 1 – 4, 2010.

[Jol02] I.T. Jolliffe; “Principal Component Analysis”, Springer, 2nd Ed., pp. 1-518, 2002.

[KA03] N. Kamci, Y. Altunbasak; “Performance comparison for the emerg-

ing H.264 video coding standard with the existing standards"; Proc. On the Int. conference on multimedia and expo (ICME), vol.1, pp.345-348, 2003.

[KAD03] L. Kessal, N. Abel, D. Demigny; "Real-time image processing with dynamically reconfigurable architecture"; Elsevier Real-Time Imaging, vol.9, pp. 297-313, 2003.

[KBR+11] M. Kuehnle, A. Brito, C. Roth, K. Dagas, J. Becker; "The Study of a Dynamic Reconfiguration Manager for Systems-on-Chip," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp.13-18, 2011.

[KBW+07] S. Kelem, B. Box, S. Wasson, R. Plunkett, J. Hassoun, C. Philips, "An elemental computing architecture for SD radio", Software Defined Radio Forum Technical Conference, 6 pages, 2007.

[KCC10] W. Kim, K. Cho, K. Chung; "Stage-based Frame-Partitioned Parallelization of H.264/AVC Decoding"; IEEE Trans. on consumer electronics, vol.56, no.2, pp. 1088 – 1096, 2010.

[KGT+12] A. Krutz, A. Glantz, M. Tok, and T. Sikora. *Adaptive global motion temporal filtering;* Picture Coding Simposium (PCS), pp. 289-292, 2012.

[KJ10] R. Khraisha, L. Jooheung Lee; "A scalable H.264/AVC deblocking filter architecture using dynamic partial reconfiguration," IEEE Int. Conference on Acoustics Speech and Signal Processing (ICASSP), 2010, pp.1566-1569, 2010.

[KM12] T. Kelly, M. Minges; "IC4D: Maximizing mobile"; International bank for reconstruction and development, ICT publications; pp. 1-244; DOI: 10.1596/978-0-8213-8991-1, 2012.

[KOM11] M.N. Krifa, B. Ouni, A. Mtibaa; "Exploring the self-reconfiguration of FPGA: design flow, architecture and performance", Int. Journal on computer science and engineering, vol.3, no.4, pp.1713-1720, 2011.

[KRL+06] C. Kim, A. Rassay, S. Lachowicz, M. Lee, K. Eshraghian, "3D-ShoftChip: A novel architecture for next-generation adaptive

computing systems", EURASIP Journal on applied signal processing, vol. 2006, pp. 1-13, 2006.

[KT10] N. Kefalas, G. Theodoridis; "A high throughput pipelined architecture for H.264/AVC deblocking filter"; Int. Conf. on Electronics, Circuits, and Systems (ICECS), pp. 387-391, 2010.

[KTR08] I. Kuon, R. Tessier, J. Rose; "FPGA Architecture: survey and challenges"; Foundations and trends in electronic design automation, vol. 2, n.2, pp. 135-253, 2008.

[Kun88] S.Y. Kung; "Mapping algorithms onto array structures"; VLSI Array Processors, Prentice Hall, pp. 110-188, 1988.

[LAM09] H. Loukil, A.B. Atitallah, N. Masmoudi; "Hardware architecture for H.264/AVC deblocking filter algorithm"; Int. Multi-Conf. On Systems, Signals and Devices, pp. 1 – 6, 2009.

[LCB+06] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, et. al.; "XiSystem: a XiRisc-based SoC with Reconfigurable IO module", IEEE Journal of Solid-state circuits, vol.41, nº1, pp. 88-96, 2006.

[LCC10] Y.-K. Lai, L.-F. Chen, W.-C. Chiou; "A memory interleaving and interlacing architecture for deblocking filter in H.264/AVC"; IEEE Trans. On Consumer Electronics, vol. 56, no.4, pp. 2812 – 2817, 2010.

[LF09] W. Lie, W. Feng-Yan; "Dynamic Partial Reconfiguration in FPGAs," Intelligent Information Technology Application, IITA; Third Int. Symposium on, vol.2, pp.445-448, 2009.

[LH09] C. Liang, X. Huang; "SmartCell: an energy efficient coarse-grained reconfigurable architecture for stream-based applications", EURASIP Journal on Embedded Systems, Hindawi Publishing Corporation, pp. 1-15, 2009.

[LHC+12a] S. López, P. Horstrand, G.M. Callicó, J.F. López, R. Sarmiento; "A low-computational-complexity algorithm for hyperspectral endmember extraction: Modified Vertex Component Analysis";

IEEE Geoscience and Remote Sensing Letters; vol.9, no.3, pp.502-506, 2012.

[LHC+12b] S. López, P. Horstrand, G.M. Callicó, J.F. López, R. Sarmiento; "A Novel Architecture for Hyperspectral Endmember Extraction by Means of the Modified Vertex Component Analysis (MVCA) Algorithm", IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol.5, no.6, pp.1837-1848, 2012.

[Lin09] Y. Lin; "A two-result-per-cycle deblocking filter architecture for QFHD H.264/AVC decoder"; IEEE Trans. On Very Large Scale Integration (VLSI) Systems, vol.17, no.6, pp. 838-843, 2009.

[LLL07] T.-M. Liu, W.-P. Lee, C.-Y. Lee; "An in/post-loop deblocking filter with hybrid filtering schedule"; IEEE Trans. On Circuits and Systems for Video Technology, vol. 17, no.7, pp. 937 – 943, 2007.

[LPC07] M. Lanuzza, S. Perri, P. Cornosello, "MORA: A new coarse-grain reconfigurable array for high throughput multimedia processing", Ed. Springer, book "Embedded computer systems: architectures, modelling, and simulation", pp. 159-168, 2007.

[LVG+13] S. López, T. Vladimirova, C. González, J. Resano, D. Mozos, A. Plaza; "The promise of reconfigurable computing for hyperspectral imaging onboard systems: a review and trends"; Proceedings of the IEEE, vol. 101, n.3, pp. 698-722, 2013.

[LYC+10] T. Liu, E. Yang, R. Cheng, Y. Fu; "CUDA-based H.264/AVC deblocking filtering"; Int. Conf. on audio language and image processing (ICALIP), pp. 1547-1551, 2010.

[LZZ+12] M. Li, J. Zhou, D. Zhou, X. Peng, S. Goto; "De-blocking filter design for HEVC and H.264/AVC", Advances in multimedia information processing, Ed. Springer, Lecture notes in computer science vol.7674, pp. 273-284, 2012.

[MBT+11] K. Messaoudi, E. Bourennane, S. Toumi, G. Ochoa; "Performance comparison of two hardware implementations of the deblocking filter used in H.264 by changing the utilized data width"; Int. Workshop on Systems, Signal Processing and their Applications

(WOSSPA), pp. 55-58, 2011.

[MC09] K.-Y. Min, J.-W. Chong; "A memory efficient architecture of deblocking filter in H.264/AVC using hybrid processing order"; Int. SoC Design Conference (ISOOC), pp. 67-70, 2009.

[Mcc13] S. McCloud; "RTL power reduction & high level synthesis report 2013"; White paper; Caplypto 2013; pp. 1-6; 2013.

[MRA+09] M. Álvarez, A. Ramírez, A. Azevedo, C. Meenderinck, B. Juurlink, M. Valero; "Scalability of Macroblock-level Parallelism for H.264 Decoding"; Int. Conf. on Parallel and Distributed Systems (ICPADS), pp. 236 – 243, 2009.

[MSF13] Microsemi Int., SmartFusion Processor; available at: www.actel.com/products/smartfusion

[MSV+08] B. Mei, B. De Sutter, T. Vander, M. Wouters, S. DuPont, A. Kanstein, "Implementation of a coarse-grained reconfigurable media processor for AVC decoder", Journal of signal processing systems, nº51, pp.225-243, 2008.

[MWR+12] F.D. Van der Meer, H.M.A van der Werff, F.J.A. van Ruitenbeek, C.A. Hecker, W.H. Bakker, M.F. Noomen, M. van der Meijde, et. al.; "Multi-and hyperspectral geologic remote sensing: a review"; Int. Journal of applied Earth observation and geoinformation, vol.14, no.1, pp. 112-128; 2012.

[MZM+13] M. Mehrubeoglu, P.V. Zimba, L.L. McLauchlan, M.Y. Teng; "Spectral unmixing of three-algae mixtures using hyperspectral images," IEEE Sensors Applications Symposium (SAS), pp.98-103, 2013.

[NAC] NASA, "AVIRIS data – Ordering free AVIRIS standard data products". Available at: aviris.jpl.nasa.gov/data/free_data.html

[NAO+12] M.F. Nadeem, I. Ashraf, S.A. Ostadzadeh, S. Wong, K. Bertels; "Task Scheduling in Large-scale Distributed Systems Utilizing Partial Reconfigurable Processing Elements," IEEE 26th Int. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp.79-90, 2012.

[NB05] J.M. Nascimento, J.M. Bioucas; "Vertex component analysis: a fast

algorithm to unmix hyperspectral data", IEEE Trans. on Geoscience and Remote Sensing, vol.43, no.4, pp. 898- 910, 2005.

[Nie98] R. Niemann; “Hardware/software co-design for data flow dominated embedded systems”, Boston: Kluwer Academic Publishers, pp. 262-286, 1998.

[OBL+04] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, T. Pereira, T. Stockhammer, T. Wedi; “Video coding with H.264/AVC: tools, performance and complexity”; IEEE Circuits and Systems magazine, vol.4, no.1, pp. 7-28, 2004.

[OMP+10] A. Otero; A. Morales-Cas; J. Portilla; E. de la Torre; T. Riesgo; "A modular peripheral to support self-reconfiguration in SoCs," 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD), pp.88-95, 2010.

[OTR+10] A. Otero; E. de la Torre; T. Riesgo; Y.E. Krasteva; "Run-Time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs"; Int. Conference on Field Programmable Logic and Applications (FPL), pp.70-76, 2010.

[PBB+09] A. Plaza, J.A. Benediktsson, J.W. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanssot, et. al.; “Recent advances in techniques for hyperspectral image processing”, Elsevier Remote sensing of environment 113, no.1, pp. 110-122; 2009.

[PDM12] K. Paul, C.Dash, M.S. Moghaddam, “A runtime reconfigurable architecture”, 15th Euromicro Conference on Digital System Design (DSD); pp. 26-33; 2012.

[PGT07] R. S. Pai, R. Govindarajan, M.J. Thazhuthaveetil; “Limits of data-level parallelism”, IEEE Int. Conference on high performance computing (HiPC), pp. 1-5, 2007.

[PH08] M. Parlak, I. Hamzaouglu; “Low power H.264 deblocking filter hardware implementations”; IEEE Trans. On consumer electronics, vol. 54, no.2, pp. 808-816, 2008.

[PHC+11] B. Pieters, C.-F. J. Hollemeersch, J. De Cock, P. Lambert, W. De Neve, R.V. De Walle; “Parallel Deblocking Filtering in MPEG-4

AVC/H.264 on Massively Parallel Architecture"; IEEE Trans. On Circuits and Systems for Video Technology, Trans. Letters, vol. 21, no. 1, pp. 96-100, 2011.

[PLS+11] A. Papakonstantinou, Y. Liang, J.A. Stratton, K. Gururaj, D. Chen, W.-M. W. Hwu, J. Cong; "Multilevel granularity parallelism synthesis on FPGAs", IEEE Int. Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 178-185, 2011.

[PM11] H. Parvez; H. Mehrez; "Application-Specific Mesh-based Heterogeneous FPGA Architectures"; Springer Science + Business Media, pp. 9-30, 2011.

[PMB11] K. Patel; S. McGettrick, C.J. Bleakley; "SYSCORE: A Coarse Grained Reconfigurable Array Architecture for Low Energy Biosignal Processing," Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual Int. Symposium on, pp.109-112, 2011.

[PNK+06] H. Parizi, A. Niktash, A. Kamalizad, N. Bagherzadeh, "A reconfigurable architecture for wireless communication systems", Proceedings of the Third Int. conference on Information Technology: New Generations (ITNG'06), pp. 250-255, 2006.

[Por05] R. B. Porter; Book chapter "Chapter 6: Image processing", Reconfigurable Computing. Accelerating computation with field-programmable gate arrays; Published by Springer; pp. 119-139, 2005.

[PSD08] S. Pillement, O. Sentieys, R. David, "DART: A functional-level reconfigurable architecture for a High energy efficiency", EURASIP Journal on Embedded Systems, pp.1-13, 2008.

[RF93] B. R. Rau, J.A. Fisher; "Instruction-level parallel processing: history, overview and perspective"; Journal of Supercomputing, vol.7, no.1, pp. 1-57, 1993.

[Rog94] R.E. Roger; "A faster way to compute the noise-adjusted principal components transform matrix", IEEE Trans. on Geoscience and

Remote Sensing, vol.32, no.6, pp. 1194-1196, 1994.

[RSB07] V.S. Rosa, A.A. Susin, S. Bampi; “An HDTV H.264 deblocking filter in FPGA with RGB video output”; IEEE Int. Conf. On Very Large Scale Integration (VLSI), pp. 308 – 311, 2007.

[RSK12] H. Richter, B. Stabernack, V. Kuhn; "Architectural decomposition of video decoders for many core architectures," Conference on Design and Architectures for Signal and Image Processing (DASIP), pp.1-8, 2012.

[RTF+00] K. Rath, S. Tangirala, P. Friel, P. Balsara, J. Flores, J. Wadley, “Reconfigurable Array Media Processor (RAMP)”, IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 287-288, 2000.

[SBB+06] P. Sedcole, B. Blodget, T. Becker, J. Anderson, P. Lysaght; “Modular dynamic reconfiguration in virtex FPGAs”, IEEE Proceedings on Computers and Digital Techniques, 153(3):157164, 2006.

[SBB06] S. Shukla, N. Bergmann, J. Becker, “QUKU: a two-level reconfigurable architecture” Procee. Of the emerging VLSI tech. and architectures, pp. 1-13, 2006.

[SBK+09] K.-H. Sihn, H. Baik, J.-T. Kim, S. Bae, H.J. Song; “Novel approaches to parallel H.264 decoder on symmetric multicore systems”; IEEE Int. Conf. on Acoustic, Speech and Signal Processing (ICASSP), pp. 2017-2020, 2009.

[SCA10] T. Silva, L. A. Cruz, L. Agostini; “A novel macroblock-level filtering upsampling architecture for H.264/AVC scalable extension”; Proc. of the Symposium on Integrated Circuits and System Design, pp. 163-167, 2010.

[SCL+11] N. Suárez; G.M. Callicó; S. López; J. López; R. Sarmiento; “Performance Analysis of the Scalable Video Coding (SVC) extension of H.264/AVC for constrained scenarios”; Proc. SPIE 8067, VLSI Circuits and Systems V, pp. 1-9, 2011.

[SCM00] H. Schmit, S. Cadambi, M. Moe, “Pipeline Reconfigurable FPGAs”

Journal of VLSI signal processing systems, vol. 24, pp. 129-146, 2000.

[SCS+11] N. Suárez, G.M. Callicó, S. López, J. López, R. Sarmiento; "Performance analysis of the Scalable Video Coding (SVC) extension of H.264/AVC for constrained scenarios"; Proc. Of SPIE Int. Symposium on Microtechnologies for the New Millennium, vol. 8067, 2011.

[SD11] K. Schuckman, J.A. Dutton; "Hyperspectral remote sensing systems"; The Pennsylvania State University, available at: www.e-education.psu.edu/geog883kls/node/463; 2011.

[SDK09] A. Sudarsanam, R. Kallam, A. Dasu; "PRR-PRR dynamic relocation"; IEEE computer architecture letters, vol.8, no.2, pp. 44-47, 2009.

[SFZ12] W. Shen, Y. Fan, X. Zeng; "A 64cycles/MB, luma-chroma parallelized H.264/AVC deblocking filter for 4k x 3k applications"; IEICE Trans. On electronics, vol. E95.C, issue 4, pp. 441-446, 2012.

[Sir13] M. Sir; "Parallel Programming Patterns". Available at: www.cs.uiuc.edu/homes/snir/PPP

[SJR10] X. Song; X. Jiang; X. Rui, "Spectral unmixing using linear unmixing under spatial autocorrelation constraints," IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS), pp.975-978, 2010.

[SMP+10] S. Sánchez, G. Martín, A. Plaza, C.-I. Chang; "GPU Implementation of fully constrained linear spectral unmixing for remotely sensed hyperspectal data exploitation"; Proc. Of SPIE on Satellite data compression, communications and Processing VI, pp. 1-11; 2010.

[SMW07] H. Schwarz; D. Marpe; T. Wiegand; "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard"; Circuits and Systems for Video Technology, IEEE Trans. on , vol.17, no.9, pp.1103,1120, Sept. 2007.

[SWS05] T. Sato, H. Watanable, K. Shiba; "Implementation of dynamically reconfigurable processor DAPDNA-2"; VLSI design, automation and test, VLSI-TISA Int. symposium on, pp. 323-324; 2005.

[SZC+11] H. Su, C. Zhang, J. Chai, Q. Yang; "An Efficient Parallel Deblocking

Filter Based on GPU: Implementation and Optimization"; IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, pp. 280-285, 2011.

[TB10] X. Tian, K. Benkrid; "High-Performance Quasi-Monte Carlo Financial Simulation: FPGA vs. GPP vs. GPU"; ACM Trans. on Reconfigurable Technology and Systems, vol.3, no.4, pp. 1-22; 2010.

[Tei12] J. Teich; "Hardware/software codesign: The past, the present, and predicting the future", Proc. of the IEEE vol. 100, May 13th, pp. 1411-1430, 2012.

[TJG09] E.B. Van der Tol, E.G. Jaspers, R.H. Gelderblom; "Mapping of H.264 decoding on a multiprocessor architecture*";* SPIE Conf. on Image and Video Communications and Processing, vol. 5022, pp. 707-718, 2009.

[TL11] J. Tuominen, T. Lipping; "Detection of environmental change using hyperspectral remote sensing", working report 2011-26 at Olkilouto repository site; pp. 1-56; 2011.

[TP09] T.-H. Tsa, Y.-N. Pan; "High efficient H.264/AVC deblocking filter architecture for real-time QFHD"; IEEE Trans. On Consumer Electronic, vol.55, no.4, pp. 2248 – 2256, 2009.

[Tra11] Trautner, R.; "ESA's roadmap for next generation payload data processors", DASIA Conference 2011. Available at: www.esa.int/TEC/OBDP

[TSS07] K. Tatas, K. Siozios, D. Soudris; "A survey of existing fine-grain reconfigurable architectures and CAD tools"; Chapter 1 of the book Fine- and coarse-grain reconfigurable computing; Springer; pp. 1-85; 2007.

[TSV07] G. Theodoridis, D. Soudris, S. Vassiliadis; "A survey of coarse-grain reconfigurable architectures and CAD tools. Basic definitions, critical design issues and existing coarse-grain reconfigurable systems"; Chapter 2 of the book *Fine- and coarse-grain reconfigurable computing*; Springer; pp. 89-149; 2007.

[TVM09] M. Torabi, A. Vafaee, N. Movahhedinia; "A fast architecture for deblocking filter in H.264/AVC using clock cycles saving process";

Int. Conf. on Multimedia, Signal Processing and Communication Technologies (IMPACT), pp. 324 - 327, 2009.

[UBM13] UBM Electronics; “2013 Embedded market study”; ESC at Design west 2013. Available at: www.slideshare.net/MTKDMI/2013-embedded-market-study-final

[UG191] Xilinx Inc., “Virtex-5 FPGA Configuration User Guide”; UG191. Available at: www.xilinx.com

[USGS] USGS digital spectral library. Available at: speclab.cr.usgs.gov/spectral-lib.html

[VCAA] VCA algorithm MATLAB code for Windows. Available at: www.lx.it.pt/~bioucas/code.htm

[VCK10] S. Vijay, C. Chakrabarti, L.J. Karam; "Parallel deblocking filter for H.264 AVC/SVC," IEEE Workshop on Signal Processing Systems (SIPS), pp.116-121, 2010.

[VJG09] E.B. Van der Tol, E.G. Jaspers, R.H. Gelderblom; “Mapping of H.264 decoding on a multiprocessor architecture”; Proc. SPIE Conf. on Image and Video Communications and Processing, vol. 5022, pp. 707-718, 2009.

[VK09] T. Van As, S. Krijgsman; “Reconfigurable architectures: a survey of design and implementation methods”, Faculty of Electrical Engineering, Mathematics and Computer Science, Delf University, pp. 1-11, 2009.

[VLD+13] A. Barberis, G. Danese, F. Leporati, A. Plaza, E. Torti; "Real-Time Implementation of the Vertex Component Analysis Algorithm on GPUs," Geoscience and Remote Sensing Letters, IEEE , vol.10, no.2, pp.251-255, 2013.

[WDG+10] I. Werda, T. Dammak, T. Grandpierre, M. Ayed, N. Masmoudi; “Real-time H.264/AVC baseline decoder implementation on TMS320C6416”; Journal of Real-Time Image Processing. pp. 1-18, 2010. Springer Berlin.

[WIN10] WinterGreen Research Inc. 2010; Technical Report “Programmable logic IC market shares and forecasts, worldwide, 2010 to 2016.”; pp. 1-287; 2010.

[Win99] M.E. Winter; "N-FINDR: an algorithm for fast autonomous spectral end-member determination in hyperspectral data"; Proc. Of the SPIE conference on Imaging Spectrometry V, pp. 266-275, 1999.

[WKK10] W. Kim, K. Cho, K. Chung; "Stage-based Frame-Partitioned Parallelization of H.264/AVC Decoding"; IEEE Trans. on Consumer Electronics, vol.56, no.2, pp. 1088 – 1096, 2010.

[WL06] T. Warsaw, M. Lukowiak; "Architecture design of an H.264/AVC decoder for real-time FPGA implementation"; Proc. Of Int. Conf. On Application-specific Systems, Architectures and Processors (ASAP), pp. 253 – 256, 2006.

[WYC+09] S.-W. Wang, S.-S. Yang, H.-M. Chen, C.-L. Yang, J.-L. Wu; "A multi-core architecture based parallel framework for H.264/AVC deblocking filters"; Journal of Signal Processing Systems, vol.57, no.2, pp. 195 – 211, Springer, 2008.

[XAP138] Xilinx Inc., Virtex FPGA Series Configuration and Readback"; XAPP138. Available at:

www.xilinx.com/support/documentation/application_notes/xapp138.pdf

[XC08] K. Xu, C.-S. Choy; "A five-stage pipeline, 204 cycles/MB, single-port SRAM-based deblocking filter for H.264/AVC"; IEEE Trans. On Circuits and Systems for Video Technology, vol.18, no.3, pp. 363 – 374, 2008.

[XCOR] Xilinx Inc., Xilinx Core generator tool. Available at: www.xilinx.com/tools/coregen.htm

[XFPO] Xilinx Inc., LogiCORE IP Floating-Point Operator v5.0. Available at: www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf

[XIL12] Xilinx Inc., "Xilinx Corporate Overview". Available at: www.xilinx.com/aboutus/corporate_overview.pdf

[XMEM] Xilinx Inc., Xilinx LogiCORE IP Block Memory Generator. Avaible at: www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen_ds512.pdf

[XPAH] Xilinx Inc., Xilinx PlanAhead design and analysis tool. Available at: www.xilinx.com/tools/planahead.htm

[XWP10] Xilinx Inc.; White Paper – FPGAs, "High Performance Computing using FPGAs"; WP375 (v1.0), pp. 1-15; 2010.

[XZY13] Xilinx Inc., Zynq-7000 all programmable SoC. Available at: www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html

[YDZ11] C. Yan, F. Dai, Y. Zhang; "Parallel deblocking filter for H.264/AVC on the TILERA many-core systems"; Advances in Multimedia Modelling, Lecture Notes in Computer Science, vol. 6523/2011, pp. 51-61, 2011.

[ZYD+12] Y. Zhang, C. Yan, F. Dai, Y. Ma; "Efficient parallel framework for H.264/AVC deblocking filter on many-core platform"; IEEE Trans. on Multimedia, vol. 14, no.3, pp. 510-524, 2012.

[ZZZ+09] J. Zhou, D. Zhou, H. Zhang, Y. Hong, P. Liu, S. Goto; "A 136 cycles/MB, luma-chroma parallelized H.264/AVC deblocking filter for QFHD applications"; IEEE Int. Conf. on Multimedia & Expo (ICME), pp. 1134 – 1137, 2009.

Annex

Annex A

Sinopsis en español

Este capítulo ofrece una visión general del trabajo de investigación realizado en esta Tesis Doctoral (Arquitecturas reconfigurables dinámicamente para sistemas de codificación de vídeo e imágenes hiperespectrales), poniendo especial interés en las contribuciones más significativas en el campo de la reconfiguración dinámica.

A.1 INTRODUCCIÓN

Las mejoras en los procesos productivos, junto con la evolución tecnológica acontecida en las últimas décadas han hecho posible que las FPGAs (Field-Programmable Gate Arrays) hayan incrementado su presencia en el mercado, como sustituto (en determinados casos) a los ASICs como parte de los sistemas empotrados utilizados en numerosos dispositivos de uso diario. Parte de este éxito se debe a su equilibrio entre rendimiento, flexibilidad y coste. Además, la capacidad que poseen algunas FPGAs del mercado para configurarse múltiples veces, incrementa la versatilidad de los sistemas finales que incorporan este tipo de dispositivo, puesto que permite la reutilización de los recursos para ejecutar distintas tareas.

Si el proceso de reconfiguración de las FPGAs se realiza en tiempo de ejecución, sin que ello repercuta sobre el funcionamiento del resto del sistema, se habla de reconfiguración dinámica. Esta característica es muy atractiva para aquellos sistemas empotrados que han de desempeñar sus tareas en entornos variables, o de difícil acceso. La razón de ello es que, haciendo uso de la reconfiguración dinámica, el sistema podría ser capaz de adaptar su funcionalidad y/o rendimiento a las condiciones del entorno de trabajo sin necesidad de la intervención de ningún agente externo. Si bien es cierto que en los últimos ha habido avances en este sentido, tradicionalmente la reconfiguración dinámica se ha orientado con dos objetivos: la adaptación funcional de los sistemas, y la actualización o mejora de una funcionalidad ya existente. Ambos casos se basan en la sustitución de módulos. Entendiendo por módulo un diseño implementado en la FPGA. Sin embargo, existe una alternativa muy atractiva para potenciar la adaptación de los sistemas a las variaciones del entorno, basada en el ajuste de los recursos hardware utilizados en tiempo de ejecución. Esta característica por la cual un diseño ofrece la posibilidad de modificar sus recursos, por medio del aumento o decremento de los elementos de procesamiento, se conoce como escalabilidad. La variación en el nivel de escalabilidad repercute directamente sobre el rendimiento del diseño, ya que este podrá procesar mayor o menor número de datos en paralelo, de acuerdo con los requisitos del entorno, o del

sistema sobre el que se esté ejecutando. Sin embargo, aplicar este tipo de cambios en sistemas empotrados autónomos no es una tarea sencilla. Por un lado, porque en la mayoría de los escenarios dinámicos (que cambian sus propiedades o características a lo largo del tiempo) se desconoce cómo van a evolucionar; y por lo tanto no se puede predecir a priori cuales van a ser sus necesidades. Por otro lado, la gestión de la reconfiguración dinámica sigue suponiendo un reto importante para los diseñadores y desarrolladores, ya que no existen mecanismos o soluciones estándares para hacerle frente.

Con el fin de paliar algunas de las carencias mencionadas en el campo de la reconfiguración dinámica, el trabajo desarrollado en esta Tesis Doctoral tiene por objetivos explorar las bondades de escalabilidad, a través de la reconfiguración dinámica, así como proporcionar un mecanismo de control y gestión del proceso de reconfiguración para sistemas empotrados autónomos basados en FPGAs .

Para hacer frente a la exploración de la escalabilidad se han seleccionado dos aplicaciones complejas y exigentes, en cuanto al número y tipo de operaciones que han de realizar.

Finalmente, las conclusiones acerca de la eficacia y eficiencia de las soluciones propuestas (diseños escalables dinámicamente, y mecanismo de control del proceso de reconfiguración) se establecerán objetivamente, mediante el análisis de los resultados que se deriven de la ejecución de un demostrador que se ha preparado.

A.2 SOLUCIONES ARQUITECTURALES ESCALABLES DINÁMICAMENTE

La consecución de los objetivos mencionados anteriormente requiere que, las soluciones arquitecturales a implementar sobre una FPGA Xilinx Virtex-5 LX110T, cumplan con ciertas especificaciones, entre las que destacan: modularidad, homogeneidad, paralelismo y control distribuido. Por lo tanto, todas las soluciones arquitecturales propuestas cumplirán con estas premisas, independientemente de la aplicación para la que hayan sido diseñadas.

La primera de las soluciones tiene por finalidad diseñar y desarrollar una arquitectura escalable para el *deblocking filter*. Ésta es una de las tareas más costosas, en cuanto al tiempo de ejecución y al tipo de operaciones, que se han de realizar como parte de los procesos de codificación y decodificación de vídeo de los estándares H.264/AVC, y su extensión SVC. La segunda de las soluciones se encarga de la extracción de endmembers, técnica ampliamente utilizada en teledetección para el tratamiento de imágenes hiperespectrales.

A.2.1 Deblocking Filter

La función principal del *deblocking filter* (DF) es la de suavizar los bordes de las imágenes, para reducir la distorsión visual del efecto bloque. Este efecto se origina por el propio procesamiento de los demás bloques que componen los codificadores y/o decodificadores H.264/AVC y SVC.

La solución propuesta se muestra en la Figure A.1. En ella se puede apreciar la modularidad de la arquitectura, en la que se identifican cinco módulos distintos (IC, OC, IM, OM y PE). Estos elementos se pueden agrupar en tres grandes grupos de acuerdo con la función principal que desempeñan: control (IC-OC), almacenamiento y distribución (IM-OM), y procesamiento de datos (PE). Tal y como muestra la Figure A.1.a, el elemento de proceso (PE) está compuesto por dos elementos; el router y la unidad funcional (por sus siglas en inglés, FU). El router es la entrada y salida de datos de la FU que tiene asociada. Es responsable de alimentar a su FU con datos para procesar, así como enviar los datos procesados al sistema. Por su parte, la FU es el núcleo de filtrado de la arquitectura. Se puede decir que este módulo es el DF en sí mismo, mientras que los demás módulos actúan como soporte para que la FU pueda desempeñar su función correctamente.

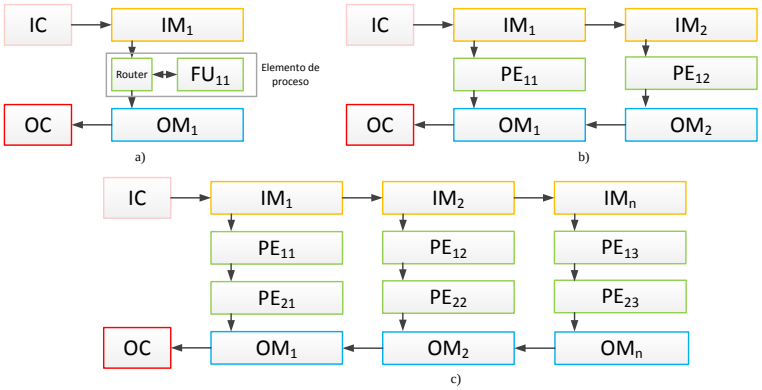


Figure A.1 Arquitectura escalable propuesta del DF; a) Estructura básica 1×1; b) 1×2; c)2×3

En la Figure A.1 se muestra claramente la capacidad de adaptación de la arquitectura propuesta, a través de distintos ejemplos de escalabilidad. La Figura 1.a representa el esquema básico de la arquitectura, es decir el número mínimo de módulos que ha de contener para que pueda operar. Como se puede apreciar en la figura, la escalabilidad se puede apreciar en dos dimensiones. Así, la Figure A.1.b representa el crecimiento de la solución en el incremento del número de columnas; mientras que la Figure A.1.c crece en ambas direcciones (filas y columnas). En cualquier caso, el aumento del nivel de escalabilidad se consigue por la réplica de los elementos básicos, y siempre respetando un mismo patrón. Tal que los IMs siempre conformarán la primera fila de la matriz, y los OMs la última. Por lo tanto, los PEs siempre permanecerán entre estos dos elementos. Como se puede observar, independientemente de la escalabilidad, los elementos de control IC y OC nunca se replican. Esto se debe a que son módulos estáticos, que permanecen siempre ubicados en la misma región de la FPGA con el fin de ser los módulos de comunicación de la arquitectura con el exterior.

Para poder escalar el rendimiento del DF, la solución propuesta paraleliza el procesamiento de los datos a nivel de MB. De este modo cada PE procesa un MB distinto en cada instante. Sin embargo, uno de los retos de este nivel de paralelismo radica acelerar el procesamiento tanto como sea posible, pero además respetar a su vez con dependencias de datos existentes entre los MBs adyacentes dentro de una imagen, según imponen los estándares H.264/AVC y

SVC. Para hacer esto posible, el diseño propuesto sigue un patrón de procesamiento de datos basado en una versión mejorada del *wavefront*, tal y como se representa en la Figure A.2.

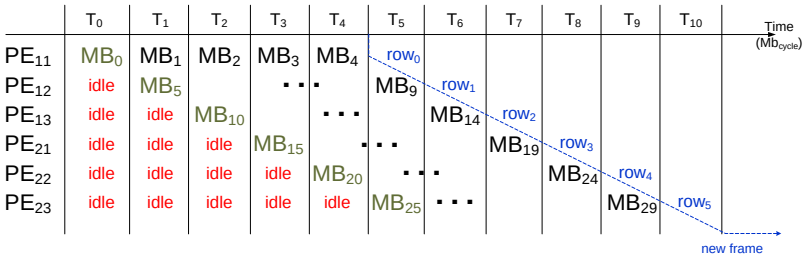


Figure A.2 Configuración 2x3 siguiendo el patrón wavefront mejorado propuesto

Cabe resaltar que el comportamiento de los módulos IM, OM y PE no varía con la escalabilidad. Esta independencia del comportamiento global de la arquitectura con el nivel de escalabilidad se debe, en gran parte, al control distribuido diseñado. Es decir, no existe un único elemento encargado de sincronizar y controlar todas las operaciones permanentemente. Si no que, una vez se inicia el sistema, cada módulo se sincroniza internamente en función de los datos recibidos.

A.2.2 Extracción de endmembers

El estado de la técnica en el campo de la extracción de endmembers engloba diversos algoritmos. Sin embargo, este trabajo ofrece una solución arquitectural escalable a uno sólo, el MVCA. Este algoritmo es reciente, y es una mejora del conocido el VCA.

El algoritmo del MVCA se responsabiliza de extraer los endmembers puros de una imagen hiperespectral. Y para ello, la solución arquitectural propuesta consta de varios módulos (Figure A.3). De entre todos los módulos, el IMAGE PROJECTON es el mejor candidato para aprovechar las ventajas de la escalabilidad. Esto se debe al tipo de operaciones que realiza (multiplicación matricial), la cual permite paralelizar el procesamiento de los datos.

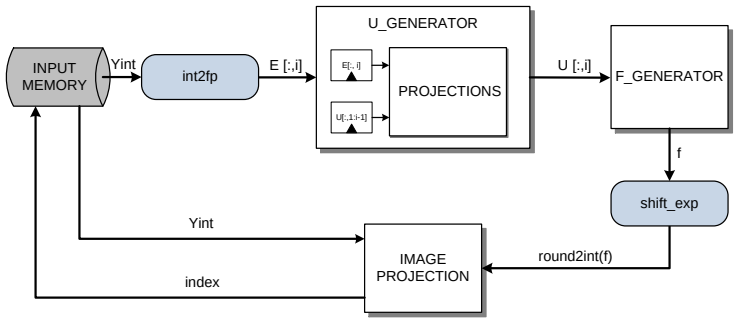


Figure A.3 Estructura modular de la arquitectura del MVCA

Más en detalle, la tarea principal del módulo IMAGE PROJECTION consiste en resolver un producto matricial ($Y_{int} \times f$), y determinar cuál de los productos fila×columna ha obtenido el mayor valor. La matriz Y_{int} está compuesta por los píxeles, con todas y cada una de sus componentes espectrales que lo conforman, de la imagen hiperespectral. Mientras que la matriz f se calcula en los otros módulos de la arquitectura, y representa el conjunto de píxeles puros calculados hasta el momento. De acuerdo con la resolución matemática del producto matricial, existen dos formas de paralelizar el módulo IMAGE PROJECTION, y por lo tanto escalar la solución. De ahí que se hayan propuesto dos soluciones arquitecturales distintas para solucionar un mismo problema.

La primera de ellas se muestra en la Figure A.4. El conjunto de elementos de proceso (PEs) se reparten el procesamiento de todos los elementos involucrados en la multiplicación de una fila ($f[i]$) y una columna ($Y_{int}[i,j]$) del producto matricial. De manera que el nivel máximo de escalabilidad será igual al número de elementos disponibles en una fila. Más concretamente, inicialmente el $PE_1 = f[1] \times Y_{int}[1,j]; PE_2 = f[2] \times Y_{int}[2,j]; \ldots; PE_n = f[n] \times Y_{int}[n,j]$.

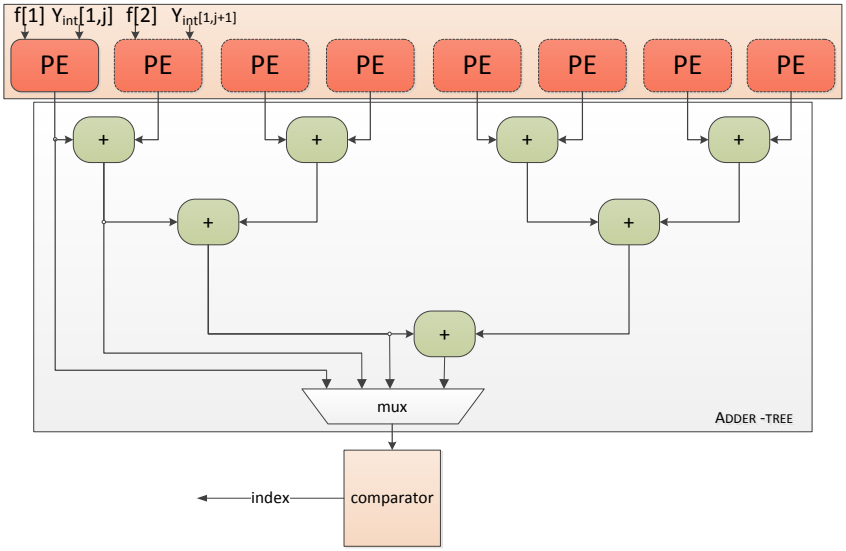


Figure A.4 Arquitectura escalable ScalableSA_MVCA

La segunda de las propuestas, mostrada en la Figure A.5, afronta el producto matricial de diferente forma. En esta caso, cada uno de los PEs disponibles se ha de responsabilizar de la multiplicación de todos y cada uno de los elementos del producto $f[i] \times Y_{int}[i,j]$. Más concretamente:

$$PE_1 = \sum f[i] \times Y_{int}[i,1];\ PE_2 = \sum f[i] \times Y_{int}[i,2]\ ; \ldots ; PE_n = \sum f[i] \times Y_{int}[i,n].$$

Con la finalidad de limitar el diseño a un único comparador se ha impuesto como restricción que los PEs no terminen de procesar simultáneamente. Para conseguir esto, se retrasa un dato el inicio de las operaciones entre PEs consecutivos. De ahí la inclusión de retardos en la Figure A.4 a la entrada de los elementos de proceso.

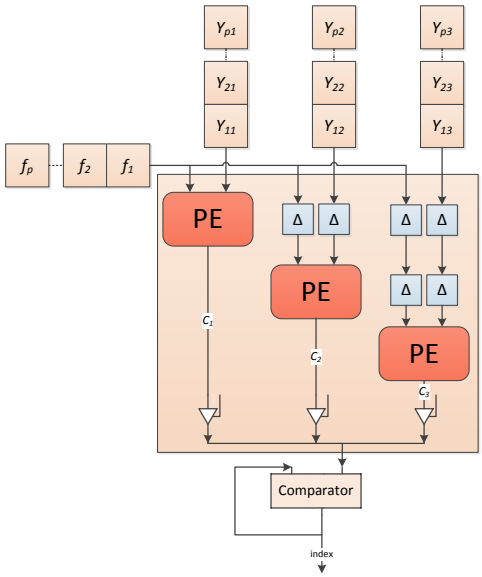


Figure A.5 Arquitectura escalable PixelSA_MVCA

A.3 GESTIÓN DEL PROCESO DE RECONFIGURACIÓN DINÁMICA

La inclusión de soluciones escalables en FPGAs, como parte de sistemas empotrados autónomos, supondría un salto cualitativo y cuantitativo importante para el mercado. Desafortunadamente la reconfiguración dinámica sigue siendo un campo de investigación bastante joven, y por lo tanto existen ciertas carencias que han de cubrirse antes de que llegue a explotarse en el mercado. Uno de los mayores retos que se han de solventar radica en la falta de mecanismos de control y gestión inteligentes, capaces de organizar los cambios de escalabilidad, o de funcionalidad del sistema de acuerdo con las necesidades del entorno en tiempo de ejecución. En este sentido, el trabajo propuesto ha desarrollado un planificador software que, junto con un módulo hardware, permite gestionar la evolución de la región reconfigurable (R.R.) de la FPGA. Para ello, esta solución determina qué tareas y cómo han de desarrollarse atendiendo a las peticiones del microprocesador, tal y como indica la Figure A.6.

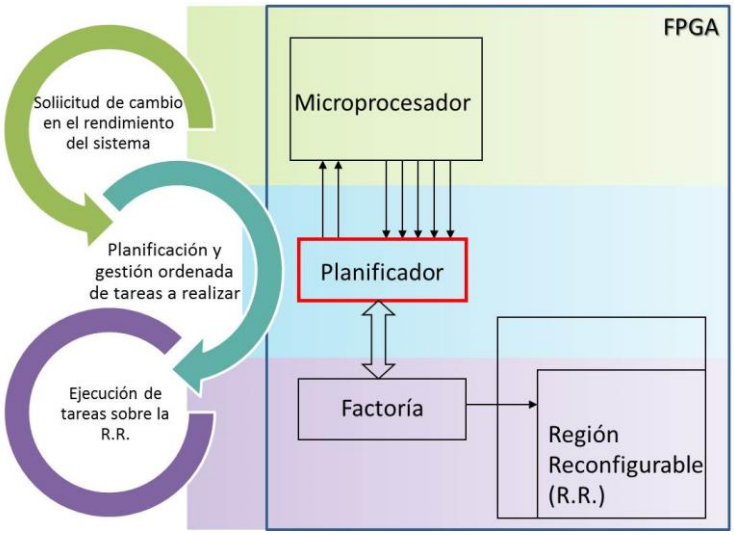


Figure A.6 Estructura jerárquica de control para la gestión de la reconfiguración dinámica

El planificador, desarrollado en software, es el director del proceso de reconfiguración hardware, de tal manera que es responsable de evaluar, analizar y organizar todos y cada uno de los pasos que han de ejecutarse para garantizar el éxito del proceso de reconfiguración. Por su parte, el motor de la reconfiguración (Factoría, según la Figure A.6), se encarga de tareas relacionadas con la manipulación del bitstream de configuración. De hecho, entre sus funciones recaen las tareas de obtener los bitstream, adaptar la información y cargarlo sobre la región reconfigurable.

A.4 CONCLUSIONES

A pesar de las ventajas que ofrece la reconfiguración dinámica, su uso en sistemas empotrados sigue siendo compleja, y en consecuencia poco utilizada en el mercado.

En base al análisis de trabajos previos significativos, dentro del campo de la reconfiguración dinámica, la presente Tesis Doctoral trata de contribuir al estado de la técnica aportando soluciones viables que faciliten la explotación de la reconfiguración dinámica en FPGAs para sistemas empotrados autónomos, en entornos de trabajo variables. Para ello, la investigación de esta Tesis explora dos líneas de trabajo bien diferenciadas. Primero, potenciar la escalabilidad de

los diseños para así, en combinación con la reconfiguración dinámica poder adaptar el rendimiento de un diseño según las demandas del sistema. Seguidamente, aportar un mecanismo de control del proceso de reconfiguración dinámica adecuado para ese tipo de escenarios.

Annex B

Publications

B.1 NATIONAL AND INTERNATIONAL CONFERENCES

[1] T. Cervero, A. Kanstein, S. López, B. De Sutter, R. Sarmiento, J.-Y. Mignolet; "Architectural exploration of the H.264/AVC decoder onto a coarse-grain reconfigurable architecture", Int. Conf. on Design of Circuits and Integrated Systems, pp. 1-6, 2008.

[2] T. Cervero, S. López, R. Sarmiento; "Dynamically reconfigurable architectures for multimedia applications"; Int. Conf. on Design of Circuits and Integrated Systems, pp. 1-6; 2009.

[3] T. Cervero, S. López, G.M. Callicó, F. Tobajas, V. de Armas, J. López, R. Sarmiento; "Survey of reconfigurable architectures for multimedia applications"; Proc. SPIE7363, VLSI Circuits and Systems IV, pp. 1-12; 2009.

[4] T. Cervero, A. Otero, S. López, E. de la Torre, G. Callicó, T. Riesgo, R. Sarmiento; "Framework adaptable y reconfigurable dinámicamente para procesamiento de video: aplicación a la etapa de filtrado adaptativo en sistemas de compresión de video H.264/AVC y SVC"; Jornadas de computación reconfigurable y aplicaciones (JCRA), pp. 127-132, 2011.

[5] T. Cervero, S. López, R. Sarmiento, T. Frangieh, P. Athanas; "Scalable models for autonomous self-assembled reconfigurable systems", Proc. Of the Int. Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 410-415, 2011.

[6] T. Cervero, A. Otero, S. López, E. de la Torre, G. Callicó, R. Sarmiento, T. Riesgo; "A novel scalable deblocking filter architecture for H.264/AVC and SVC video codecs", IEEE Int. Conf. on Multimedia and Expo (ICME), pp. 1-6, 2011.

[7] T. Cervero, S. López, G. Callicó, R. Sarmiento, A. Otero, E. de la Torre, T. Riesgo; "Run-time scalable architecture for deblocking filtering in H.264/AVC – SVC video codecs", Proc. of the Int. Conf. on Field Programmable Logic and Applications (FPL), pp. 369-375, 2011.

[8] T. Cervero, A. Otero, E. de la Torre, S. López, G. Callicó, T. Riesgo, R. Sarmiento; "Scalable 2D architecture for H.264 SVC deblocking filter with reconfiguration capabilities for on-demand adaptation", Proc. SPIE8037,

VLSI Circuits and Systems, pp. 1-10, 2011.

[9] T. Frangieh, R. Streoop, P. Athanas, T. Cervero; “A modular-based assembly framework for autonomous reconfigurable systems”, Proc. Int. Symposium, ARC, pp. 314-319, 2012.

[10] T. Cervero, A. Gómez, S. López, R. Sarmiento, J. Dondo, F. Rincón , J.C. López; “A hierarchical scheduling and management solution for dynamic reconfiguration in FPGA-based embedded systems”, Proc. SPIE8764, VLSI Circuits and Systems VI, pp. 1-9, 2013.

[11] T. Cervero, J. Dondo, A. Gómez, S. López, F. Rincón, R. Sarmiento, J.C. López; “A resource manager for dynamically reconfigurable FPGA-based embedded systems”; Euromicro Conf. on Digital System Design, pp. 1-6, 2013.

B.2 JOURNALS AND BOOK CHAPTERS

[12] Otero, T. Cervero, E. de la Torre, S. López, G.M. Callicó, T. Riesgo, R. Sarmiento; “Run-time scalable architecture for deblocking filtering in H.264/AVC – SVC video codecs”, Book title: Embedded Systems Design with FPGAs, Springer, pp. 173-199, 2013.

[13] T. Cervero, A. Otero, S. López, E. de la Torre, G.M. Callicó, T. Riesgo, R. Sarmiento; “A scalable H.264/AVC deblocking filter architecture”, Journal of Real-Time Image Processing, Springer, January, pp. 1-25, 2013

[14] T. Cervero, S. López, G.M. Callicó, J.F. López, R. Sarmiento; “Scalable architectures for real-time hyperspectral unmixing”, Elsevier Ed. For Microelectronics Journal; pp. 1-23; 2013 (In review)