7-5-2012

Dynamically reconfigurable management of energy, performance, and accuracy applied to digital signal, image, and video Processing Applications

Daniel Rolando Llamocca Obregon

Daniel Rolando Llamocca Obregon
Candidate

Electrical and Computer Engineering
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

MARIOS PATTICHIS , Chairperson

CHRISTOS CHRISTODOULOU

JAMES C. LYKE

RON LUMIA

Dynamically Reconfigurable Management of Energy, Performance, and Accuracy applied to Digital Signal, Image, and Video Processing Applications

by

DANIEL LLAMOCCA

B.Sc., Electrical Engineering, Pontificia Universidad Católica del Perú, 2002
M.Sc., Computer Engineering, University of New Mexico, 2008
PhD., Computer Engineering, University of New Mexico, 2012

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico
Albuquerque, New Mexico

May, 2012

DEDICATION

This dissertation is dedicated to my parents, Daniel and Arcelia. Besides their guidance and affection, they provided me with all the tools and opportunities that made from me who I am ...

Acknowledgments

I would like to express my deepest and sincerest gratitude to my advisor, Dr. Marios Pattichis, for introducing me to the fascinating world of research and for his stimulating and invaluable help during all these years. I am deeply grateful for his patience, encouragement, invaluable advice, and countless hours of work.

Special thanks to all of my friends and ECE, whose friendship and support made the days of work more bearable.

Dynamically Reconfigurable Management of Energy, Performance, and Accuracy applied to Digital Signal, Image, and Video Processing Applications

by

Daniel Llamocca

B.Sc., Electrical Engineering, Pontificia Universidad Católica del Perú, 2002

M.Sc., Computer Engineering, University of New Mexico, 2008

PhD., Engineering, University of New Mexico, 2012

Abstract

There is strong interest in the development of dynamically reconfigurable systems that can meet real-time constraints in energy/power-performance-accuracy (EPA/PPA). In this dissertation, I introduce a framework for implementing dynamically reconfigurable digital signal, image, and video processing systems.

The basic idea is to first generate a collection of Pareto-optimal realizations in the EPA/PPA space. Dynamic EPA/PPA management is then achieved by selecting the

Pareto-optimal implementations that can meet the real-time constraints. The systems are then demonstrated using Dynamic Partial Reconfiguration (DPR) and dynamic frequency control on FPGAs.

The framework is demonstrated on: i) a dynamic pixel processor, i) a dynamically reconfigurable 1-D digital filtering architecture, and iii) a dynamically reconfigurable 2-D separable digital filtering system.

Efficient implementations of the pixel processor are based on the use of look-up tables and local-multiplexes to minimize FPGA resources. For the pixel-processor, different realizations are generated based on the number of input bits, the number of cores, the number of output bits, and the frequency of operation. For each parameters combination, there is a different pixel-processor realization. Pareto-optimal realizations are selected based on measurements of energy per frame, PSNR accuracy, and performance in terms of frames per second. Dynamic EPA/PPA management is demonstrated for a sequential list of real-time constraints by selecting optimal realizations and implementing using DPR and dynamic frequency control.

Efficient FPGA implementations for the 1-D and 2-D FIR filters are based on the use a distributed arithmetic technique. Different realizations are generated by varying the number of coefficients, coefficient bitwidth, and output bitwidth. Pareto-optimal realizations are selected in the EPA space. Dynamic EPA management is demonstrated on the application of real-time EPA constraints on a digital video.

The results suggest that the general framework can be applied to a variety of digital signal, image, and video processing systems. It is based on the use of offline-processing that is used to determine the Pareto-optimal realizations. Real-time constraints are met by

selecting Pareto-optimal realizations pre-loaded in memory that are then implemented efficiently using DPR and/or dynamic frequency control.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

There is a strong interest in developing effective methods that can provide hardware systems that respond to run-time constraints on energy/power, performance, and accuracy. For example, it is interesting to consider scalable solutions that can deliver different performances based on energy constraints. Here, a low-energy solution will be needed when there is a requirement for long-time operation. On the other hand, a high-performance solution is often considered when there are no power (or energy) constraints.

Effective run-time management of hardware resources can be effectively handled through the use of Dynamic Partial Reconfiguration (DPR). DPR technology, currently available on Field Programmable Gate Arrays (FPGAs), enables the run-time allocation and de-allocation of hardware resources by modifying or switching off portions of the FPGA while the rest remains intact, continuing its operation. In addition to modifying resources, FPGAs with Digital Clock Managers (DCMs) also allow for real time modification of the operating frequency. These two technologies enable the development of dynamically reconfigurable systems that can meet constraints in power/energy, performance, and accuracy.

We consider digital Signal, Image, and Video Processing systems that are characterized in terms of their requirements on energy/power, performance, and precision. The goal of the dynamically reconfigurable system is to select an optimal

architecture that satisfies time-varying energy/power, performance, and accuracy (EPA/PPA) constraints. Thus, the process of determining an optimal solution is defined in terms of multi-objective optimization, with the goal of reducing energy/power consumption, while maximizing performance and accuracy, subject to time-varying EPA/PPA constraints.

The process of controlling Energy/Power, Performance, and Accuracy at run-time is referred as Dynamic Energy/Power-Performance-Accuracy (DEPA/DPPA) management. As an example of DPPA management, consider a simple example. Suppose that a video processing system is assigned the task of delivering performance at 30 frames per second (fps) on limited battery life that will also need to operate for at least 10 hours. If we can meet the performance and power requirements, we can then select the system realization with the highest accuracy. Then, after one hour, suppose that a fast moving target is observed. This will likely change the requirements to an increased frame rate. Now, suppose that we are asked to deliver performance at 100 fps at some minimum level of accuracy. This will certainly increase the minimum power requirements. In this case, we will select the hardware realization that has the lowest power requirements while meeting the performance (≥100 fps) and accuracy constraints. Thus, we see that DPPA management is especially important for video systems for which PPA requirements can vary over time.

Dynamic EPA/PPA management is more effective when applied to hardware architectures that are efficient in terms of resource consumption. We demonstrate dynamic EPA/PPA management on two resource-effective architectures associated with real-time video processing (that demands significant processing requirements).

The first application is the development of the *dynamic pixel processor*. Single-pixel operations include the implementation of functions that perform gamma correction, contrast enhancement, histogram equalization, histogram shaping, thresholding, Huffman table encoding, and quantization. Here, after computing an appropriate function, each output pixel only depends on the corresponding input pixel.

The second application is the development of the dynamic *2-D FIR filtering system*. Here, the focus on 2-D FIR filtering comes from the large number of possible applications. The list of applications includes image and video denoising, linear image and video enhancement, image restoration, edge detection, face recognition, etc.

1.2 Thesis statement

The main objective of this PhD dissertation is the development of a dynamic energy/power-performance-accuracy management approach for digital signal, image, and video processing architectures. This is possible by the use of Dynamic Partial Reconfiguration (DPR) and Dynamic Frequency Control on FPGAs. The dynamically reconfigurable architectures are evaluated in terms of energy/power-performance-accuracy trade-offs. In addition, the architectures presented in this work use techniques that minimize the amount of computational resources and make intensive use of DPR.

In particular, the research is focused on the development of a dynamic pixel processor, and a dynamic 2-D FIR filtering system. The energy/power-performance-accuracy (EPA/PPA) spaces for both the pixel processor and the 2-D FIR filter are explored. Moreover, the optimal realizations (in the multi-objective sense) are extracted from the

EPA/PPA space. The optimal realizations are then used in a dynamic management system to meet real-time varying constraints in the EPA/PPA spaces.

1.3 Innovations and Contributions

A list of the primary innovations and contributions includes:

- Development of fully-parameterized hardware cores for signal, image, and video processing applications. The architectures are implemented with techniques that minimize the amount of computing resources and take advantage of Dynamic Partial Reconfiguration.
- Characterization of the optimal (in the multi-objective sense) hardware realizations from the EPA/PPA space for the architectures presented.
- A new framework for dynamic energy/power, performance, and accuracy (EPA/PPA) management based on a multi-objective optimization approach that guarantees low energy, high accuracy, and high performance. The framework is applicable to a wide array of signal, image, and video processing architectures.
- Development of hardware systems that support dynamic energy/power, performance, and accuracy management that meet real-time EPA/PPA constraints. On hardware, dynamic EPA/PPA management is based on the run-time control of hardware resources and frequency of operation.

1.4 Organization

This dissertation is organized into six chapters. In what follows, a summary of each chapter is provided.

Chapter 2 presents the dynamic pixel processor architecture and its corresponding dynamic energy/power-performance-accuracy management. The material presented in this chapter has been submitted for publication:

D. Llamocca and Marios Pattichis, "A dynamically Reconfigurable Pixel Processor system based on Power/Energy-Performance-Accuracy Optimization", in review, *IEEE Transactions on Circuits and Systems for Video Technology*.

The next two chapters deal with the details of the hardware implementation of a 2D FIR separable filtering system, with the ultimate goal of presenting the dynamic EPA/PPA management of the system in chapter 5.

In Chapter 3, a detailed description of a 1D FIR filter architecture is presented along with an efficient approach for dynamically modifying the filter parameters. The material presented in this chapter has been published in:

D. Llamocca, M. Pattichis, and G. Alonzo Vera, "Partial Reconfigurable FIR Filtering system using Distributed Arithmetic", *International Journal of Reconfigurable Computing*, vol. 2010, Article ID 357978, 14 pages, 2010.

Chapter 4 presents the 2D separable FIR filter implementation based on dynamic partial reconfiguration. By varying the number of coefficients and frame size, a limited version of the energy-accuracy space for 2D filter realizations is shown, and a comparison of the embedded system results with a GPU implementation is provided. The material presented in this chapter has been published in:

D. Llamocca, C. Carranza, and Marios Pattichis, "Separable FIR filtering in FPGA and GPU implementations: energy, performance, and accuracy considerations", in

Proceedings of the IEEE International Conference on Field Programmable Logic and Applications FPL'2011, Chania, Greece, Sept. 2011.

Chapter 5 develops the dynamic energy-performance-accuracy management for the 2D separable FIR filter. The material presented in this chapter is to be submitted to:

D. Llamocca and Marios Pattichis, "Dynamic Energy, Performance, and Accuracy Optimization and Management for Separable 2-D FIR Filtering for Digital Video" to be submitted to *IEEE Transactions on Image Processing*.

Chapter 6 presents conclusions, future work, and scope of the dissertation. Additionally the document has three appendices that include: i) a brief description of the VHDL code, ii) a discussion of the reliability of reconfiguring (whether fully or partial) the FPGA, and iii) a list of publications related with this dissertation.

Chapter 2

A Dynamically Reconfigurable Pixel Processor System based on Power/Energy-Performance-Accuracy Optimization

Abstract

We introduce a dynamically reconfiguration framework for implementing single-pixel operations. The system relies on a multi-objective optimization scheme that generates Pareto-optimal implementations in the Power/Energy-Performance-Accuracy (PPA/EPA) spaces. The Pareto-optimal implementations and their PPA/EPA values are stored in DDR-SDRAM and can be chosen dynamically to meet time-varying constraints.

Results are shown in terms of power, accuracy (PSNR) of the resulting image, and performance in frames per second (fps). Dynamic PPA/EPA management is implemented using Dynamic Partial Reconfiguration (DPR) and dynamic frequency control.

Index Terms—Dynamic Partial Reconfiguration, Field-programmable gate-array (FPGA), LUT-based architectures.

2.1 Introduction

There is a strong interest in developing effective methods that can provide hardware systems that respond to run-time constraints on power and performance. For example, it is interesting to consider scalable solutions that can deliver different performances based on energy constraints. Here, a low-energy solution will be needed when there is a

requirement for long-time operation. On the other hand, a high-performance solution is often considered when there are no power (or energy) constraints.

Effective run-time management of hardware resources can be effectively handled through the use of Dynamic Partial Reconfiguration (DPR). DPR technology, currently available on FPGAs, enables the run-time allocation and de-allocation of hardware resources without requiring system restart. In addition to modifying resources, FPGAs with Digital Clock Managers (DCMs) also allow for real time modification of the operating frequency.

Given the significant processing requirements associated with real-time video processing, it is interesting to consider applications associated with digital video. Here, we are primarily concerned with common single-pixel operations [1]. Single-pixel operations include the implementation of functions that perform gamma correction, contrast enhancement, histogram equalization, histogram shaping, thresholding, Huffman table encoding, and quantization. Here, after computing an appropriate function, each output pixel only depends on the corresponding input pixel. For example, in gamma correction, the output pixels are given by $O = \alpha \times I^{\gamma}$, where I denotes the image intensity of the input pixel for suitable values of α, and γ . Similarly, in histogram equalization, a mapping is first computed between the input and output pixel. $I\_eq = HistEq(\cdot)$. Here, $HistEq(\cdot)$ is a single-pixel operation.

To compare among different single-pixel realizations, we consider power requirements, performance in terms of frame rates, and accuracy (PPA). Then, the goal of the dynamically reconfigurable pixel processor is to select an optimal architecture that satisfies time-varying PPA constraints. Thus, the process of determining an optimal

solution is defined in terms of multi-objective optimization, with the goal of reducing power consumption, while maximizing performance and accuracy, subject to time-varying PPA constraints.

We refer to the process of controlling Power, Performance and Accuracy at run-time as Dynamic Power-Performance-Accuracy (DPPA) management. As an example of DPPA management, we consider a simple example. Suppose that a video processing system is assigned the task of delivering performance at 30 frames per second (fps) on limited battery life that will also need to operate for at least 100 hours. If we can meet the performance and energy requirements, we can then select the system realization with the highest accuracy. Then, after one hour, suppose that a fast moving target is observed. This will likely change the requirements to an increased frame rate. Now, suppose that we are asked to deliver performance at 100 fps at some minimum level of accuracy. This will certainly increase the minimum power requirements. In this case, we will select the hardware realization that has the lowest power requirements while meeting the performance (≥100 fps) and accuracy constraints. Thus, we see that DPPA management is especially important for video systems for which PPA requirements can vary over time (also see motivation in [2]).

DPPA management for audio and video processing had been suggested in earlier works (e.g. [3], [4]). In [3], [4], the authors suggested that DPR could be used for management of power and accuracy. More recently, we have the implementation of DPPA management using DPR in networking [5], dynamic arithmetic [6], DCT implementation [7]. In these papers, DPR was used to trade-off between power and

performance requirements. In [6], the authors considered trade-offs between power, performance, and precision.

To achieve DPPA management, a space of different realizations is generated. We use the term PPA space for the different realizations that we can generate. Then, we determine optimal realizations in the multi-objective sense. In other words, we determine the Pareto optimal front of all realizations [8]. To generate the PPA space, we produce a parameterized architecture based on the input bit-width, output bit-width, the number of cores, and the frequency of operation. An early version of a single-pixel architecture that allowed switching between function was presented in [9].

In terms of application, we are primarily interested in cases where the dynamic reconfiguration rate is relatively low. We do not expect run-time constraints to change faster than once a second. In this case, the DPR overhead is not significant. On the other hand, we note that the reduction of DPR overhead is an area of active research (e.g. [10], [11], [7], [12]).

The proposed DPPA management system is based on a bottom-up approach. First, we develop an efficient architecture for implementing single-pixel operations. Then, we parameterize the hardware description and vary the parameters to generate the PPA space. Third, we use multi-objective optimization to determine the Pareto-optimal realizations. The Pareto-optimal realizations are then stored in memory. DPPA management selects among Pareto-optimal realizations to meet time-varying constraints.

The main contributions of this work include: i) an optimization framework for dynamic PPA management of the pixel processor, ii) the development of a fully-

customizable intellectual property (IP) core in VHDL, and iii) a method to dynamically reconfigure via DPR and run-time reconfiguration of the operating frequency.

The rest of the work is organized as follows. Section 2.2 presents background and related work. Section 2.3 details the internal architecture and parameterization of the pixel processor. Section 2.4 explains how the pixel processor can switch among single-pixel realizations, and modify its frequency of operations at run-time. Section 2.5 details the multi-objective optimization framework. Section 2.6 presents the experimental setup. Then, the results are presented in Section 2.7. Finally, Section 2.8 lists the conclusions.

2.2 Background and Related work

We begin with a summary of related work on the implementation of image processing systems based on Dynamic Partial Reconfiguration (DPR). In [7], the authors presented a design that dynamically reconfigures among Discrete Cosine Transform (DCT) modules of different sizes. The different DCT configurations were studied in terms of power, throughput, and (standard) image quality metrics. A dynamic systolic array accelerator for Kalman and Wavelet filters was presented in [13]. In [14], the authors presented a fingerprint image processing hardware whose stages (segmentation, normalization, smoothing, etc) are multiplexed in time via DPR. The 3D Haar Wavelet Transform (HWT) was implemented by dynamically reconfiguring a 1D HWT core thrice in [15]. A JPEG2000 decoder where the blocks are dynamically swapped is shown in [16]. In [17], an efficient 1D FIR Filtering system that combined the Distributed Arithmetic (DA) technique with DPR was presented.

There has also been some related work on the implementation of single-pixel processing functions, which usually entails two methods: LUT-based and custom hardware. In earlier work [9], we presented a basic architecture for single-pixel functions (8-bit input, 8-bit output) that stores the output pixel values in look-up tables (LUTs). The system could be dynamically reconfigured to perform arbitrary single-pixel functions. In [18], the authors presented custom architectures for 3 single-pixel functions (8-bit input, 8-bit output). A Xilinx® core for implementing gamma correction is described in [19]. In this implementation, the architecture is based on BlockRAMs whose contents can be modified on-demand. An ALTERA® LUT-based core allows for the run-time modification of LUT contents via a special interface [20]. These approaches do not address issues associated with run-time modifications of the input/output bit-widths or the frequency of operation.

A custom architecture for precise gamma correction is presented in [21]. In [22], the authors presented a contrast enhancement hardware that self-adjusts based on the histogram of the current frame. [23] presents a histogram equalization architecture. [24] performs image enhancement using a Successive Mean Quantization Transform. These architectures lack the versatility of the LUT-based approaches, but they can require far fewer resources for large input pixel bit-widths.

The trade-offs between power, performance, and accuracy for different architectures have also been investigated in the literature. Early work dealt with one or two of these properties at a time. In [25], the authors analyzed the precision requirements of a subset of recursive algorithms. In [3], the authors proposed the use of reconfiguration based on perceptual limits and the non-uniformity of video content in order to dynamically manage

power consumption, over which accuracy and performance depend on. Another example of power and precision trade-off is in [4], where the impact of numerical precision on power consumption is studied for audio processing applications. In [6], an application in dynamic arithmetic is presented where arithmetic cores are measured in terms of their power, performance, and precision requirements. Here, the use of DPR was shown to provide a low-energy example where the use of dynamic dual-fixed arithmetic cores was shown to perform as well as double floating point in an example from Linear Algebra. In [7], the authors presented a configuration manager that can dynamically adapt DCTs of different sizes based on PPA considerations.

To the best of our knowledge, no previous work has explored the Power-Performance-Accuracy space using a multi-objective optimization approach as proposed here. As it will be demonstrated by example, this approach offers some unique advantages, in that it allows for both joint and separable optimization based on a range of criteria and constraints.

This work seeks to extend prior research in the area of single-pixel operations for image enhancement by utilizing the LUT-based hardware presented in [9] and developing a fully-parameterized architecture that make use of DPR and dynamic frequency control to control the PPA space. In addition, we propose a multi-objective optimization framework to derive a set of optimal pixel processor realizations over which we can dynamically reconfigure to meet PPA constraints.

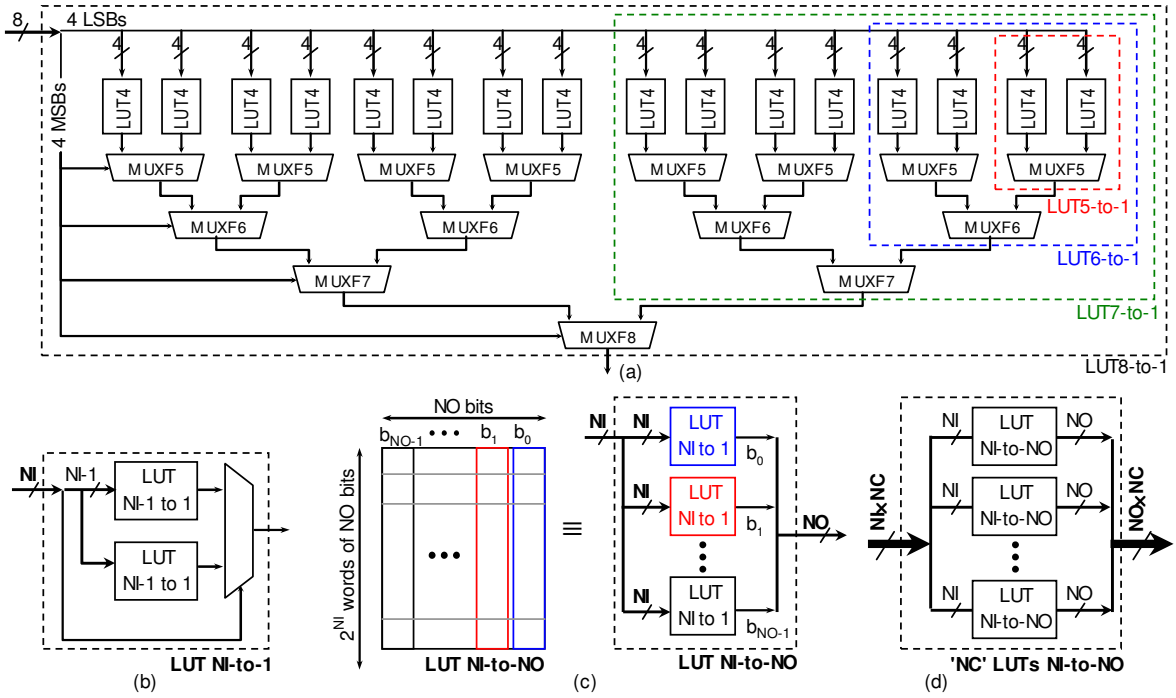

Figure 2.1. Pixel processor architecture. (a) Virtex-4 LUT8-to-1 implementation. Note the recursive implementation with specific CLB primitives (LUT4., MUXF5/6/7/8), LUT4 ≡ LUT4-to-1. (b) Recursive implementation of a NI-to-1 LUT. (c) Implementation of a NI-to-NO LUT. (d) Pixel processor core.

2.3 Pixel Processor Architecture

The pixel processor architecture is detailed here. An earlier version of this architecture appears in [9]. Here, a fully-parameterized hardware architecture based on efficient mapping of LUTs is presented.

2.3.1 Implementation of an NI-to-NO Look-Up Table (LUT)

1) LUT NI-to-1: This module uses NI input bits and one output bit. Xilinx® FPGAs contain hardwired L-to-1 LUT primitives with L = 4 (Virtex-II Pro, Virtex-4), and L = 6 (Virtex-5, Virtex-6). LUTs with higher number of input bits are built by combining the basic LUT primitives with multiplexers. Xilinx® devices let us instantiate primitives of optimized NI-to-1 LUTs for NI up to 8 [9]. Figure 2.1(a) shows the implementation of a LUT8-to-1. LUTs with NI > 8 are implemented by recursively combining two 'NI-1-to-

1' LUTs with a multiplexer, as in Fig. 1(b). The hardware complexity grows exponentially as NI increases, and thus there is a point at which a NI-to-1 LUT becomes unfeasible.

2) LUT NI-to-NO: Figure 2.1(c) depicts how a LUT NI-to-NO is built based on 'NO' LUTs NI-to-1. Each LUT NI-to-1 implements a column of the LUT NI-to-NO.

2.3.2 Pixel Processor Architecture

The pixel processor architecture core is depicted in Fig. 2.1(d). It consists of a collection of 'NC' NI-to-NO LUTs. It provides the following parameters:

- NC: Number of single-pixel processor cores.
- NI: Number of input bits of each single-pixel processor (or the number of bits of the input pixel).
- NO: Number of output bits of each single-pixel processor (or the number of bits of the output pixel).
- LUT contents: provided in a text file. These values specify a unique single-pixel function (e.g. gamma correction, contrast stretching, etc).

Depending on the application, the LUT contents of each core can be identical or different. In addition, there might be applications in which NI and NO need to be different for each single-pixel processor core. However, for the vast majority of applications, NI and NO remain constant for all the cores.

2.4 Dynamic Frequency Control and Reconfiguration of the Pixel Processor

The pixel processor core parameters can be modified at run-time via Dynamic Partial Reconfiguration (DPR). This technology allows us to dynamically allocate resources as needed by particular applications. For the pixel processor, DPR allows the modification of the single-pixel function by re-using the same resources. To reduce resources, DPR allows us to reduce the number of input and/or output bits at the expense of degraded accuracy. To increase performance, we can use DPR to increase the number of cores. In addition, the frequency of operation can be dynamically modified by controlling the Digital Clock Manager (DCM). The DCM feature provides us with the ability to directly control power and performance. The combination of DPR and dynamic frequency control allows us to switch between different realizations.

2.4.1 Dynamic Reconfiguration of the Pixel Processor:

DPR allows us to control NC (number of cores), NI (number of input bits), NO (number of output bits), and the LUT contents. A design where only the LUT contents can be dynamically altered was presented in [9]. Fig. 2.2(a) depicts the block diagram of an embedded system that allows for DPR.

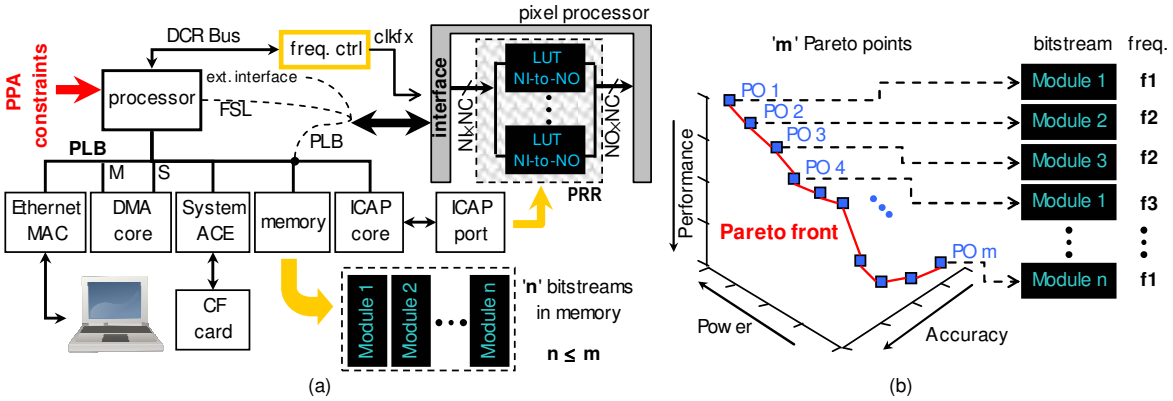

Figure 2.2. Embedded system over which we can perform DPPA management. (a) Embedded system that supports DPR and frequency control. The memory holds 'n' unique bitstreams that are needed for the Pareto front. The pixel processor can be connected to any interface (b) An example of a Pareto front with 'm' Pareto points (note that m ≤ n). A Pareto point is a unique combination of a bitstream and frequency.

The processor can be hard-core (PowerPC) or soft-core (e.g., MicroBlaze, ARM). A Compact Flash (CF) stores partial bitstreams and input frames. The memory stores data needed at run-time (e.g. input video frames, processed video frames, and partial bitstreams). For fast data processing, a Direct Memory Access (DMA) controller can be used. The Ethernet core lets us get new partial bitstreams or video frames from a PC and to send processed video frames to the PC. It is also an interface for throughput measurements and system status. The Ethernet connection lets us perform DPR from the PC (and possibly a remote location).

To perform DPR, the Partial Reconfiguration Region (PRR) must be defined. In this case, the PRR is composed of 'NC' NI-to-NO LUTs and it is dynamically reconfigured via the internal configuration access port (ICAP), which is driven by the ICAP controller. The pixel processor I/O interface depends on the application (e.g., PLB interface, FSL interface as in [9], external, etc). The frequency control core, connected to the processor via the Device Control Register (DCR) bus, provides a clock to the pixel processor.

2.4.2 Dynamic Reconfiguration of Frequency:

Digital Clock Managers (DCMs) inside FPGAs provide a wide range of clock management features [26]. Virtex-4 and Virtex-5 FPGAs use DCM, whereas Virtex-6 devices use a Mixed-Mode Clock Manager (MMCM).

The Dynamic Reconfiguration Port (DRP) of the DCM is used to dynamically adjust the frequency without reloading a new bitstream to the FPGA. The DRP uses register based control of the DCM frequency and phase.

Fig. 2.3 depicts the architecture for dynamic frequency control. The Xilinx® DCM primitive is named 'DCM_ADV' (MMCM_ADV for Virtex-6 FPGAs). We connect the

DCM to the Device Control Register (DCR) bus by means of a DCR Slave interface. The processor becomes the DCR Master. The specific architecture of the DCR Slave interface varies as different FPGA families provide a slightly different approach to load the M and D values. The frequency is dynamically controlled by modifying the ratio of M to D (see in Fig. 2.3).

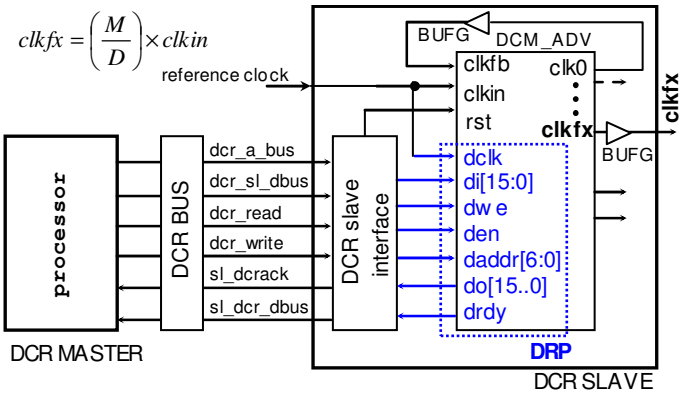

Figure 2.3. Frequency control via the DCR Bus interface. The Dynamic Reconfiguration Port of the DCM is shown in blue.

2.5 Optimization Framework for the Pixel Processor

The goal is to create a system that can select optimal realizations based on PPA constraints. The optimization is carried out for a specific single-pixel function. In this section, we detail: i) how a complete set of pixel processors is generated; ii) the manner in which Power, Performance, and Accuracy are measured; iii) how the optimal pixel processors are generated from the complete set; and iv) how we adjust pixel processor parameters and/or frequency based on PPA constraints.

2.5.1 Generation of the set of single-pixel processors

The space of pixel processor realizations is generated by modifying the pixel processor parameters (NI, NO, and NC), and the frequency of operation. The LUT contents depend on NI and NO. The selection of the parameters and/or frequency combinations depends on the application.

The Power-Performance-Accuracy (PPA) space consists of the measurements for every single pixel processor realization. The generation of the PPA space is a lengthy process, and it is done offline.

2.5.2 Performance measurements

Performance can be measured as the number of pixels (or bytes) processed per second, or by frames per second. The bytes per second processed is determined by the pixel size. Input pixels are usually 8-bit wide.

We are interested in measuring the performance of the IP shown in Fig. 2.2. The aim is to provide results from the IP angle, i.e. assuming that at every clock cycle NI×NC bits can be processed and NO×NC bits can be released. The performance of the entire embedded system depends on many factors (cache size, processor instruction execution, bus type and usage, etc.) that are subject to change. Here, the embedded system is just a generic test-bed.

The IP can process NI×NC bits and output NO×NC bits per clock cycle. Then, the number of bits it can process per unit of time is given by:

$$Performance\,(Mbps) = NI \times NC \times f\,(MHz) \tag{2.1}$$

For digital video processing, performance is measured in terms of frames per second (fps) given by:

$$fps = \frac{10^6}{Tframe(us)}, \quad Tframe(us) = \frac{1}{f\,(MHz)} \times \frac{frame\ size}{NC} \tag{2.2}$$

Note that ‘fps’ does not take into account the number of bits of the input pixels (NI)

2.5.3 Power measurements

In this subsection, we detail the IP power consumption measurement. The IP (shown in Fig. 2.1) is the sole component included in the Partial Reconfiguration Region (PRR).

Table 2.1 provides a concise description of different power types that need to be considered. The device static power depends on the environment, the size of the device, and the device family. For all practical purposes, it is assumed to be constant. Since it does not depend on the IP implementation, we report it separately in Table 2.1 for the XC4VFX60 device.

Table 2.1: Different types of power consumed at each rail for an FPGA. For the XC4VFX60 Virtex-4, the device static power is 0.44W (at 25ºC).

Static	Drawn by the device when it is powered up, configured with user logic, and there is no switching activity.	
	Device static	Consumed by the device when it is powered up and without programming the user logic.
	Design static	Consumed by the user logic when the device is programmed and without any switching activity.
Dynamic	It is the fluctuating power as the design runs; it is generated by the switching user logic and routing.	

In terms of comparing among different cases, we will only consider the sum of the dynamic and design static power (see Table 2.1). In order to estimate this power consumption, we use the FPGA power supply rails: (i) internal supply rail voltage *VCCINT* with current *ICCINT*, and (ii) auxiliary supply rail voltage *VCCAUX* with current *ICCAUX*. Here, we will not consider the output supply power since it is only associated with the power consumed by the external pins. Thus, the IP power is given by:

$$Power\ IP = VCCINT \times ICCINTp + VCCAUX \times ICCAUXp \qquad (2.3)$$

where the currents are given by:

$$\begin{aligned} ICCINTp &= ICCINT - ICCINTQ \\ ICCAUXp &= ICCAUX - ICCAUXQ \end{aligned} \tag{2.4}$$

and *ICCINTQ*, *ICCAUXQ* are defined as the device static supply currents (of their respective voltage rails).

Measuring power directly (e.g. [6]) requires custom-built boards that provide access to the voltage rails themselves. Instead, we can accurately estimate power consumption using software tools that would be widely applicable to all devices. For the purposes of this work, the Xilinx Power Analyzer (XPA) is employed for these measurements (at 25ºC). XPA provides an accurate estimate based on simulated switching activity of the place-and-routed circuit and exact utilization statistics. [27].

We also consider power consumption during dynamic partial reconfiguration. Unfortunately, there is no software tool available that can provide an estimate of this power consumption. In [6], through hardware measurements, it was determined that during DPR, the only supply current that increases is *ICCAUX* (Virtex-II Pro and Virtex-4). Thus, the DPR power can be estimated using:

$$Rec.\ Power = VCCAUX \times ICCAUX(increase) \tag{2.5}$$

From [6], we have that *ICCAUX* increases by 170 mA and 25 mA for the Virtex-II Pro (XC2VP30) and Virtex-4 (XC4VFX12) respectively. Assuming that these dynamic current measurements remain the same within the same device family, we can use these values in (2.5) and (2.6).

Furthermore, for our application in digital image and video, we will report energy consumption in terms of energy spent for processing a single frame:

$$Energy\ \ per\ \ frame(uJ) = Power \times Tframe(us) \tag{2.6}$$

For completeness, the frame size will also be reported along with the energy spent per frame (see (2.2)).

2.5.4 Accuracy measurements

The accuracy is measured using the peak signal-to-noise ratio (PSNR). This is given by:

$$PSNR(dB) = 10 \times log_{10}\left(\frac{MAXValue^2}{MSE}\right) \tag{2.7}$$

where the MSE is the mean squared error between the pixel processor output and the result using double floating-point arithmetic.

2.5.5 Generation of optimal Pixel Processor realizations

Based on the power, performance, and accuracy (PPA) measurements, we can select optimal pixel processor realizations. Here, we define a pixel processor realization to be optimal in the Pareto (multi-objective) sense [8]. A pixel realization is considered to be Pareto optimal if we cannot improve on its Power-Performance-Accuracy measurements without decreasing on at least one of them. We will next provide an example.

The goal is to minimize power, maximize performance, and maximize accuracy. For a given set of pixel processors, we want to find a subset of realizations whose results cannot be improved by any other realization for all three (PPA). The collection of all Pareto-optimal points forms a Pareto front (see Fig. 2.4). In Fig. 2.4, we are plotting realizations as points against power, and the negatives of performance and accuracy. Thus, optimal realizations appear lower-left in 2-D (see Fig. 2.4(a)). The idea is then extended to 3-D in Fig. 2.4(b).

We can also extend the example for satisfying multi-objective constraints. The idea is demonstrated in Fig. 2.4(c) and 2.4(d). Independent constraints appear as lines in 2-D and

planes in 3-D. Optimal realizations are then selected among the Pareto-optimal points that also satisfy the constraints. Dynamic constraint satisfaction only requires that we select Pareto optimal points when the constraints change. In the next section, we provide more details on the hardware implementation of this idea.

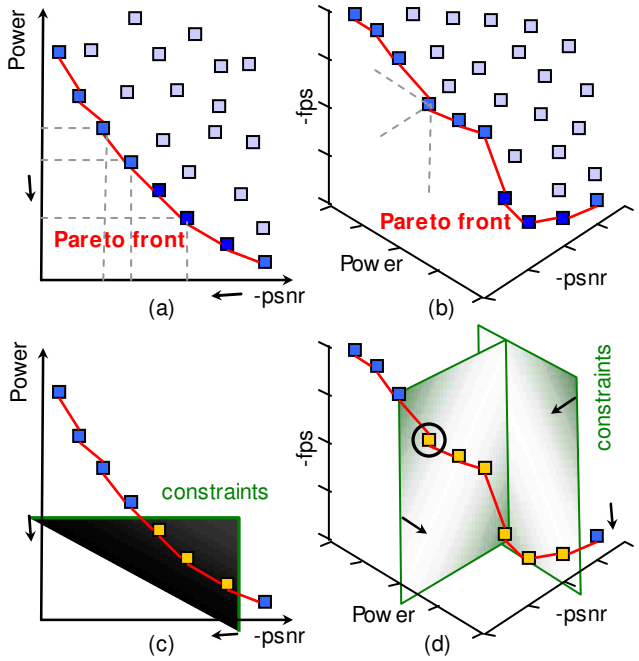

Figure 2.4. Multi-objective optimization of the PPA space. (a) 2-D Pareto Front. (b) 3-D Pareto Front. (c) Two constraints applied to the 2-D Pareto front. (d) Two constraints applied to the 3-D Pareto front. The circled point is the realization with the min. power consumption.

2.5.6 Dynamic PPA Management based on DPR and dynamic frequency control

In hardware, Pareto-optimal realizations are represented by their associated partial bitstreams, frequency of operation, and PPA measurements. The realizations and associated parameters are stored in memory. We demonstrate the basic DPPA management framework in Fig. 2.2.

Dynamic PPA management is based on selecting a single realization that satisfies the dynamic constraints. An example of a single set of constraints is shown in Fig. 2.4(c) for two constraints (PA). Here, the Pareto-optimal points are plotted in yellow. Note that we

are interested in coming up with a single realization. When only PPA constraints are given, we can have more than one solution. Thus, it makes sense to consider the case where we are minimizing one objective while imposing constraints on the other two. We next consider an example to demonstrate the idea.

Without loss of generality, suppose that we want the minimum power realization (Ri) subject to minimum accuracy and performance requirements. In this case, we want to solve:

$$\begin{aligned} &\min_{Ri} Power(Ri), \\ &subject\ to: \quad Accuracy(Ri) \geq 50dB \\ &\qquad\qquad\ \ Performance(Ri) \geq 30\,fps \end{aligned} \tag{2.8}$$

In this case, the Pareto-optimal points that satisfy the constraints are shown in yellow in Fig. 2.4(d). The realization that also minimizes power is circled. This is the optimal realization that is selected for DPR and/or dynamic frequency control. Note that if we also want the optimal solution to satisfy a power constraint, we can simply check whether the minimum power solution meets this constraint.

The implementation of Ri comes with specific values for the pixel processor parameters and frequency of operation.

Fig. 2.2(a) shows an embedded system that can modify the pixel processor parameters and the frequency of operation. Thus, each realization is represented in terms of its unique combination of partial bitstream and frequency of operation. Also, the Pareto-optimal front can contain bitstreams that are associated with more than one frequency. Fig. 2.2(b) illustrates how the user moves dynamically along the Pareto front via DPR and/or dynamic frequency control of the DCM.

2.6 Experimental Setup

In this section, we provide specific details of the platform and the scenarios under which we test the pixel processor. We also provide details on the PLB interface and frequency control in Section 2.6.2.

2.6.1 Platform testing scheme

Fig. 2.2(a) showed a generic embedded system. Here, the pixel processor system was implemented on the ML410 Xilinx® Development Board that houses a XC4VFX60-11FF1152 Virtex-4 FPGA. The PowerPC processor is selected and it is clocked at 300 MHz, with peripherals running at 100 MHz. Here, we note that the PowerPC has internal data and instruction caches that are used for data and instruction fetches from memory (64 MB DDR-SDRAM). The pixel processor IP is connected to the PLB Bus. The ICAP core used is provided by Xilinx®. The embedded system serves as a validating platform from which we extract the processed images.

2.6.2 PLB Interface and frequency control

The pixel processor is connected to the PLB bus (32-bit PLB Slave Burst interface). For the DMA core, we are using the Xilinx® Central Direct Memory Access (DMA) core with a PLB interface that supports burst transfers. The 32-bit PLB transaction requires NI×NC ≤ 32 and NO×NC ≤ 32 for optimal bus usage.

The frequency control core is shown in Fig. 2.3. The core acts as a slave to the DCR bus. The reference clock ‘clkin’ is the PLB clock (100 MHz). Thus, $clkfx = (M/D) \times 100MHz$.

Special care must be taken when varying ‘clkfx’. For Virtex-4 devices, ‘clkfx’ is limited to 32-210 MHz in the ‘low frequency mode’, and to 210-350 MHz in the ‘high

frequency mode' [28]. Switching frequency modes requires a sequence of reads/writes on the DCM dynamic reconfiguration port (DRP). To minimize overhead required for implementing clock speeds above 'PLB_clock', we have constrained 'clkfx' to be lower or equal than 'PLB_clock'. Thus, any M/D combination has to yield a 'clkfx' in the range of 32-100 MHz.

In Fig. 2.5, we show the Pixel processor implemented as a peripheral to the PLB. We handle the difference between the PLB and the Pixel Processor clocks by using input and output FIFOs and separate clock regions ('PLB_clock' and 'clkfx'). FSMs are used to control the signals for each clock region. We clock the pixel processor coress (the PRR) at 'clkfx', while the rest is clocked at 'PLB_clk'. Modifying the 'PLB_clk' directly (i.e. 'PLB_clk = clkfx') is undesirable since other peripherals (e.g. SystemACE, Ethernet core) use the 'PLB_clk' value as a parameter, requiring the dynamic reconfiguration of these IPs each time we modify 'clkfx'.

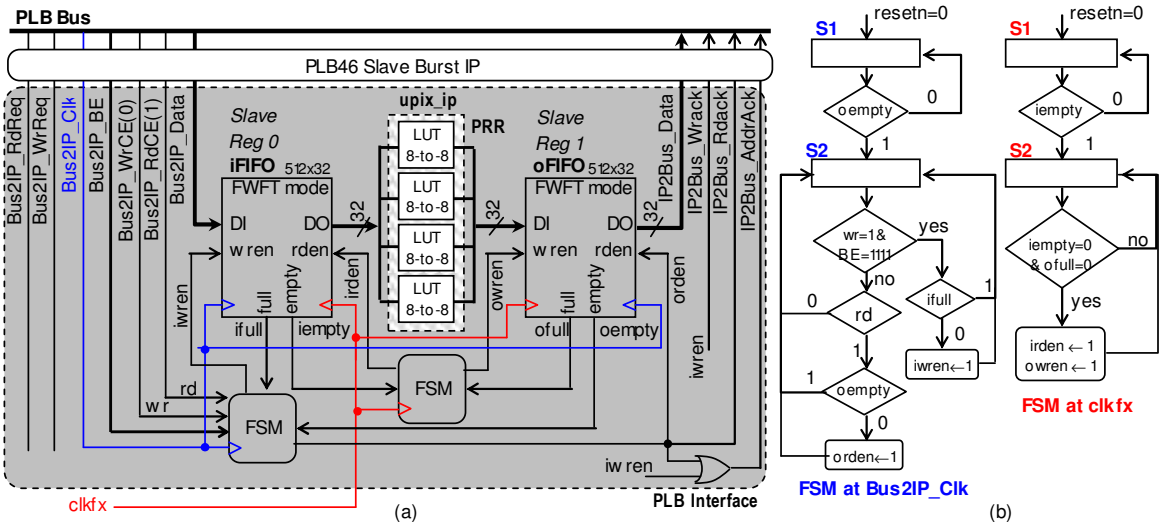

Figure 2.5. (a) Pixel Processor Slave PLB interface. In the figure, NI=NO=8, NC=4. The PRR can change as long as NI×NC ≤ 32 and NO×NC ≤ 32. (b) State Machines for each clock region

2.6.3 Selection of pixel processor parameters and frequency of operation for the generation of the set of pixel processors

Typical image and video formats are limited to a maximum of 12 bits per sample (for each color channel). Thus, we work with 12-bit and 8-bit images. For reducing the input bitwidth, we simply select the most significant bits. To maintain high accuracy in the results, we require the number of outputs bits to be equal or above the number of input bits (NO≥NI). For the adjustable frequency of operation 'clkfx', we select five different frequencies (MHz): 100.00 (M=2,D=2), 66.66 (M=2,D=3), 50.00 (M=2,D=4), 40.00 (M=2,D=5), and 33.33 (M=2,D=6).

Three different testing scenarios are considered for the Pixel Processor: (i) 32-bit I/O constrained implementations, (ii) 8/12 input constrained implementations, and (iii) fixed-frequency constrained implementations. The parameters for each scenario are summarized in Tables 2.2, 2.3, and 2.4.

1) 32-bit I/O constrained implementations: Here, the pixel processor is implemented in the 32-bit embedded system of Subsection 2.6.1. The selection of the parameters NI, NO, NC (number of cores) depends upon the resource availability and the constraints NI×NC≤32 and NO×NC≤32. Table 2.2 shows the combination of parameters chosen for both 12-bit (NI: 12→5) and 8-bit images (NI: 8→5). In this case, we consider the power and resource measurements for implementing both the LUT-cores and the PLB interface.

2) 8/12 bit input constrained implementations: In this case, NI is either 8 or 12. We do not restrict NO and NC (except for NO≥NI). NC can be as high as the FPGA device can allow. Table 2.3 lists the parameters and frequency combinations. Power and resources measurements only consider the implementation of the LUT cores (NC NI-to-NO LUTs).

3) Fixed-frequency constrained implementations: This case is similar to the previous (8/12 bit input) case. However, the frequency is fixed and we allow the input bitwidth (NI) to vary. Table 2.4 lists the possible combinations. Power and resources measurements only consider the implementation of the LUT cores (NC NI-to-NO LUTs).

Table 2.2: 32-bit I/O pixel processor constrained implementations. 8-bit images: upper side of table. 12-bit images: entire table. Each case is tested for 5 different frequencies: 100, 66.66, 50, 40, and 33.33 MHz

NI	**NO (NC)**							
5	5(4)	6 4)	7(4)	8(4)	9(2)	10(2)	11(2)	12(2)
6		6(4)	7(4)	8(4)	9(2)	10(2)	11(2)	12(2)
7			7(4)	8(4)	9(2)	10(2)	11(2)	12(2)
8				8(4)	9(2)	10(2)	11(2)	12(2)
9	9(2)	10(2)	11(2)	12(2)	13(2)	14(2)	15(2)	16(2)
10		10(2)	11(2)	12(2)	13(2)	14(2)	15(2)	16(2)
11			11(2)	12(2)	13(2)	14(2)	15(2)	16(2)
12				12(2)	13(2)	14(2)	15(2)	16(2)

Table 2.3: Pixel Processor Implementations for 8/12 bit input images unrestricted by I/O bitwidth. Each implementation is tested for 5 different frequencies: 100.00, 66.66, 50.00, 40.00, and 33.33 MHz.

Image	**NI**	**NC**	**NO**				
8-bit	8	2 4 6 8 10	8	9	10	11	12
12-bit	12	2 4 6 8	12	13	14	15	16

Table 2.4: Pixel Processor implementations restricted at 100 MHz with unrestricted I/O bitwidths.

Image	**NI**	**NO**								**NC**
8-bit	5	5	6	7	8	9	10	11	12	2
	6		6	7	8	9	10	11	12	4
	7			7	8	9	10	11	12	6
	8				8	9	10	11	12	8
										10
12-bit	9	9	10	11	12	13	14	15	16	2
	10		10	11	12	13	14	15	16	4
	11			11	12	13	14	15	16	6
	12				12	13	14	15	16	8

2.7 Results and Analysis

This section details hardware resource utilization, optimization of the PPA space for the 3 scenarios of Section 2.6.3, and a discussion of the results. For demonstrating the results for 8-bit images, we use the '*lena*' image of size 640x480. For 12-bit images, we use the '*oilp*' image of size 512x512. Here, we note that the problem of assessing the accuracy of the results is closely related to the problem of video quality assessment [1]. In the proposed setup, we expect the users to dynamically adjust the accuracy constraints to meet their expectations. An example is provided in Section 2.7.4.

Fig. 2.6 shows some output results for image '*oilp*', shown along with the selected I/O bit-widths and their accuracy. Note that the result images for *NI=8,12* are nearly identical to those of the double floating point case. For *NI=5*, there are some clear artifacts in the lower right portion of the image; they appear for PSNR levels around 50dB.

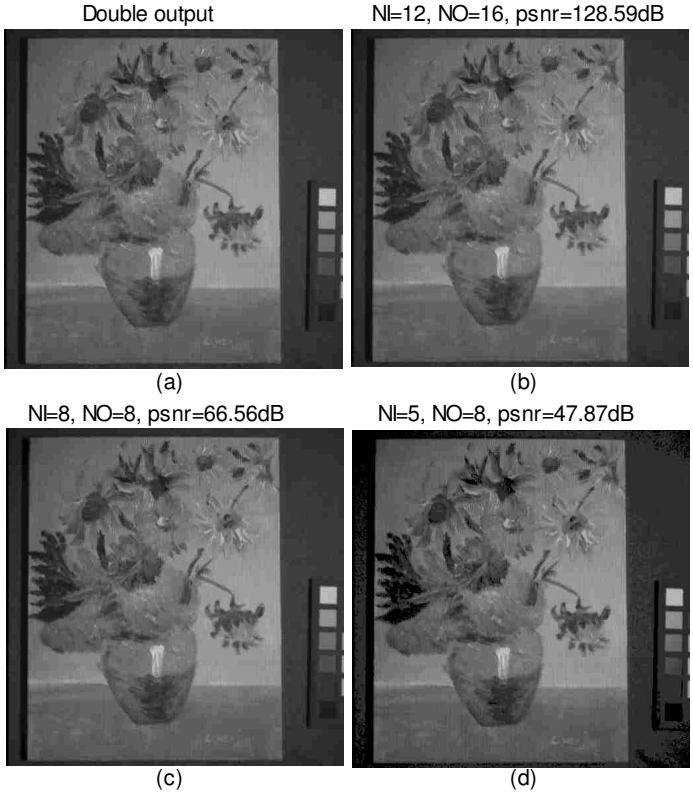

Figure 2.6. Output 'oilp' image results for various input/output cases.
Accuracy results are shown as well (simulated gamma correction for γ=0.5)

While the accuracy values do depend on specific images, we did not see the Pareto front to vary significantly from image to image. Clearly, the PSNR accuracy of the single pixel functions depends on the histogram of the image (see histograms for '*lena*' and '*oilp*' in Fig. 2.7).

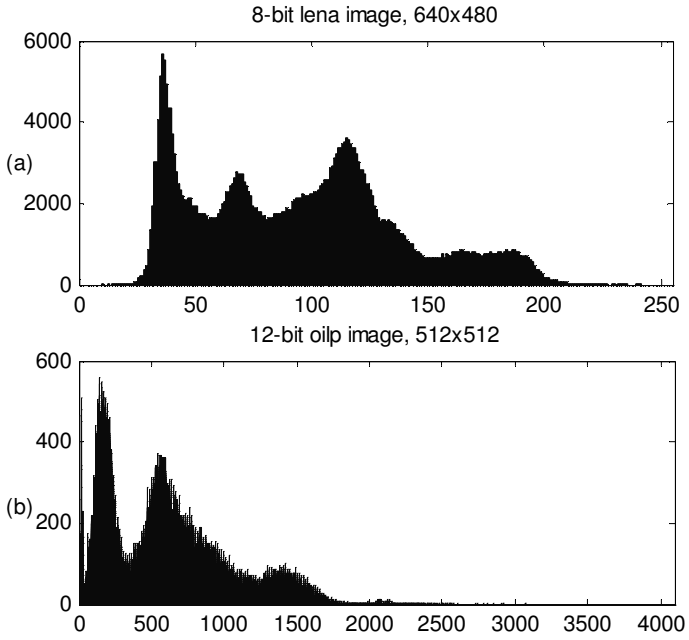

Figure 2.7. Histograms for both 8-bit 'lena' and 12-bit 'oilp' images

2.7.1 Embedded System results for 32-bit I/O constrained implementations

In this case, we report on the pixel processor implementations described in Table 2.2. In terms of the embedded system implementation, we consider the 8-bit and 12-bit systems separately. In each case, we define the PRR region to be sufficiently large for implementing the largest possible realization. Note that this constraint does not imply that the power consumption will be the same for all 8-bit or 12-bit implementations. Here, note that power consumption is a function of the utilized resources (not allocated resources).

For the 8-bit system, the PRR occupies a tightly packed area of 16×22=352 Slices with

a bitstream size of 47194 bytes. In the case of the 12-bit system, the PRR occupies 36×128=4608 Slices with a bitstream size of 449015 bytes.

A summary of resource utilization results are given in Table 2.5. Note that the largest pixel processor in the 8-bit case (4 LUT8tot8) occupies 320 Slices (91% of the allocated space for the PRR), and in the 12-bit case (2 LUT12to16) it occupies 4318 Slices (93% of the allocated PRR Slices).

Table 2.5: Embedded Pixel Processor. Resource utilization (XCV4FX60). Case I: 8-bit inputs: NI=NO=8, NC=4. Case II: 12-bitinputs: NI=12,NO=16,NC=2

Module	Slice	(%)	FF	(%)	LUT	%
Static Region	5308	21%	5519	11%	6517	13%
PRR (Case I)	320	1%	0	0%	576	1%
PRR (Case II)	4318	17%	0	0%	8576	17%
Overall (Case I)	5628	22%	5519	11%	7093	14%
Overall (Case II)	9626	38%	5519	11%	15093	30%

The average processing speed resulted in 352.85 Mbps, i.e. a video of size 640x480 can be processed at 143 fps.

Recall that in our applications, we expect that the dynamic reconfiguration rate will be small (in the order of seconds). A reconfiguration speed of 16.28 MB/s is obtained by using the Xilinx® ICAP core, resulting in 2.89 ms and 27.58 ms of reconfiguration time for the 8-bit and 12-bit cases respectively. On the other hand, a dynamic rate of 295.4 MB/s reported in [10] has been achieved using a custom-built DPR controller. However, note that additional hardware overhead is required for achieving faster rates. During reconfiguration, power increase was estimated to be 62.5 mW [6].

2.7.2 Pixel Processor IP resource utilization

We demonstrate the resource scalability of the approach in Fig. 2.8. Here, the results are independent of the clock frequency. Instead, the resource consumption is a function of

NI, NO, and NC. The hardware resource utilization, based on the number of NI-to-NO LUT cores (NC) is shown in Fig. 2.8 for the cases listed in Table 2.4.

Resource requirements (slices) are clearly clustered for each NI and they grow exponentially as NI increases (e.g. 15,30,63,136 for NI=NO=5,6,7,8 in Fig. 2.8(a)). The amount of resources increases linearly with the number of cores (NC). When we increase the number of output bits (NO), the amount of resources also increases linearly with a less steep slope (e.g. 306 to 547 for NO=9 to 16, NI=9, in Fig. 2.8(b)). The case with NI=12, NO=16, NC=8 requires the largest amount of resources that we have tested in the XC4VFX60 FPGA. Furthermore, for the given device, based on the diversity of the testing, it did not make sense to consider cases for NI>12.

It is also worth noting that resource consumption does not vary significantly with the optimized architecture shown in Fig. 2.1. Thus, we have directed the Xilinx® ISE synthesizer to implement the LUTs without optimizing for the LUT contents, allowing us to effectively swap functions as needed.

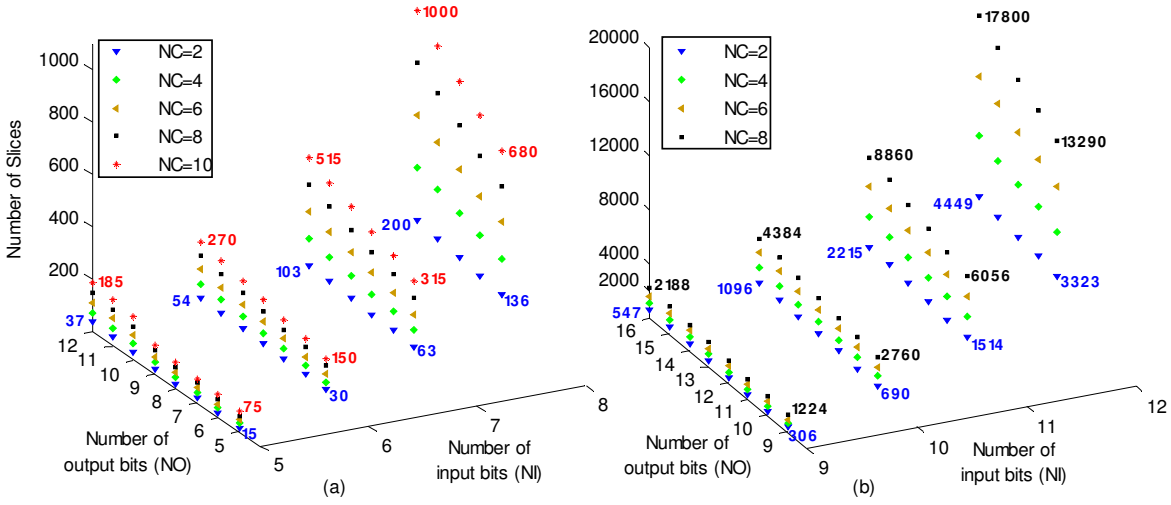

Figure 2.8. Pixel Processor IP resource (slices) utilization as NI, NO, NC vary. Device: XC4VFX60 (25280 Slices). Note the exponential resource growth as NI increases. (a) 8-bit system. (b) 12-bit system

2.7.3 Multi-objective Optimization of the PPA space

We present the results from multi-objective (PPA) optimization of the pixel processor for the 3 scenarios of Section 2.6.3. For comparison, we use gamma correction (γ=0.5) which represents a non-linear function that is used for image and video display (see example in Fig. 2.6).

In Fig. 2.9(a), we show the PPA space and the Pareto front for the 32-bit I/O constrained implementations (12-bit image, NI: 12→5). The Pareto front appears to lie on a piecewise planar surface that includes 43% of all possible realizations. They also cover a wide range of the PPA space, suggesting that the approach is effective in generating a wide range of options. Maximum accuracy of 128.6 dB is achieved at 245 fps and power of 156.7 mW. Maximum performance is achieved at 1526 fps with an accuracy of 48.14 dB and power of 48.15 mW. As shown in Fig. 2.9(b), performance increases with frequency and the number of cores. In addition, it is important to note that the accuracy depends on NI and NO in Fig. 2.9(a).

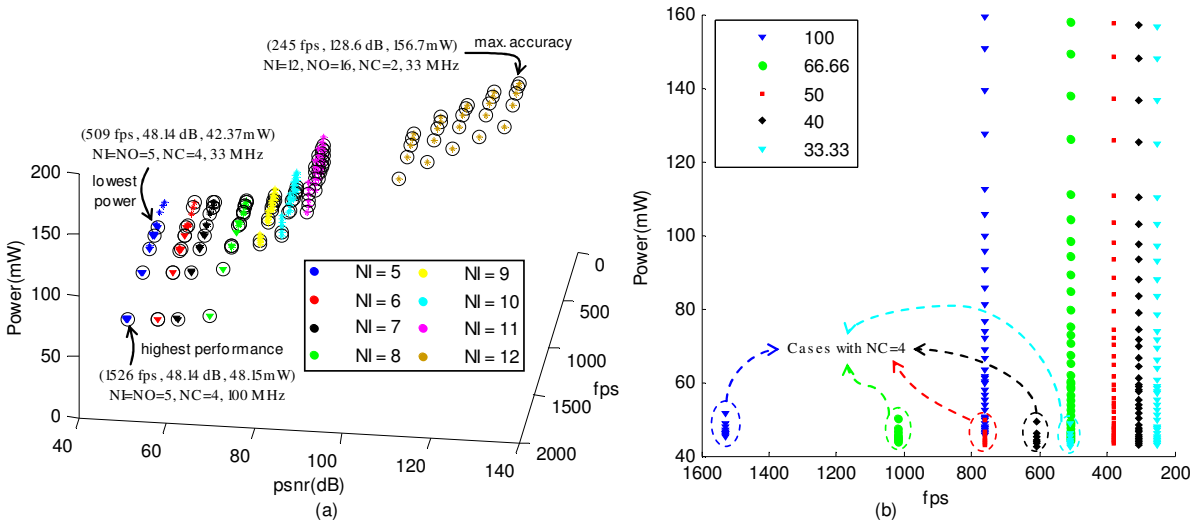

Figure 2.9. 32-bit I/O constrained implementations for 12-bit images. (a) PPA Results and Pareto Front (circled points). (b) Power and performance dependence on frequency. Note that the cases with NC=4 are circled, the rest are the cases with NC=2.

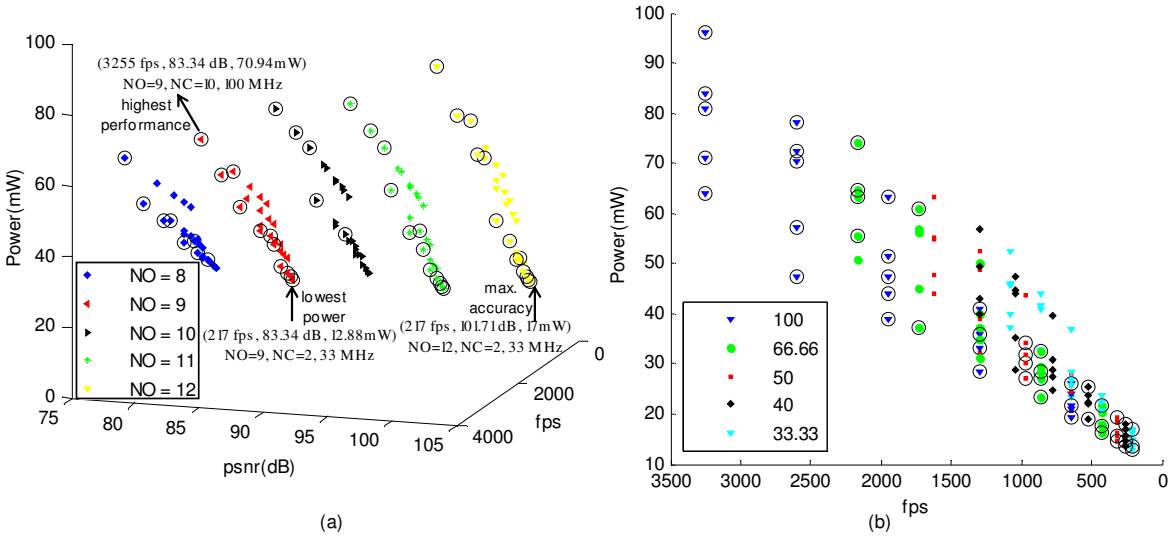

Figure 2.10. 8-bit input constrained implementations. (a) PPA Results and Pareto Front (circled points). (b) Frequency effect on power and performance

Fig. 2.10(a) shows the PPA space, the Pareto front for the 8-bit input constrained implementations (upper half of Table 2.3), and the extreme implementations (max. accuracy, min. power, max. performance). The Pareto optimal points are easily clustered as a function of NO. Fig. 2.10(b) shows how power and performance depend on frequency within an NO cluster. A similar trend occurs with an increase in NC. Unlike the I/O constrained case of Fig. 2.9, the effect of frequency on power is more noticeable in Fig. 2.10(b) because it only depends on the LUT cores. It is also interesting to note that the Pareto front includes 40% of the PPA space. As before, this suggests that parameters variation worked well in that it generated a relatively large number of optimal points. We also note that the 12-bit input constrained implementations gave similar trends as the 8-bit case. Thus, this case is not repeated here.

Instead of PPA optimization, we also considered optimization with respect to Energy, Performance, and Accuracy (EPA). Here, we computed the energy required to process a single video frame. In this case, for both the 32-bit I/O and the 8-bit input constrained

implementations, the 3-D Pareto front lied completely at the maximum frequency of 100 MHz. This implies that frequency variation did not work for EPA optimization. In other words, better implementations were obtained by varying the NI, NO, and NC parameters. This motivates the last scenario that considers the case for fixed-frequency.

Fig. 2.11 shows the EPA space for fixed-frequency (100 MHz) constrained implementations for 12-bit input images (NI:12→9). In Fig. 2.11(b), we can see that performance clusters are defined in terms of the number of cores (NC). Most of the Pareto optimal points occur for 8 cores. For fewer cores, we have three optimal cases: (i) NI=10, NO=11, NC=6, (ii) NI=NO=11, NC=6, (iii) NI=11, NO=12, NC=4. In Fig. 2.11(a), we can see that as NO decreases (for fixed NI and NC), the energy per frame and the accuracy decrease. As expected, the performance (fps) is only affected by NC. So, for NC=8, the Pareto front is defined in terms of NI and NO. Table 2.6 shows the 17 points (out of 104) that make up the 3D Pareto front.

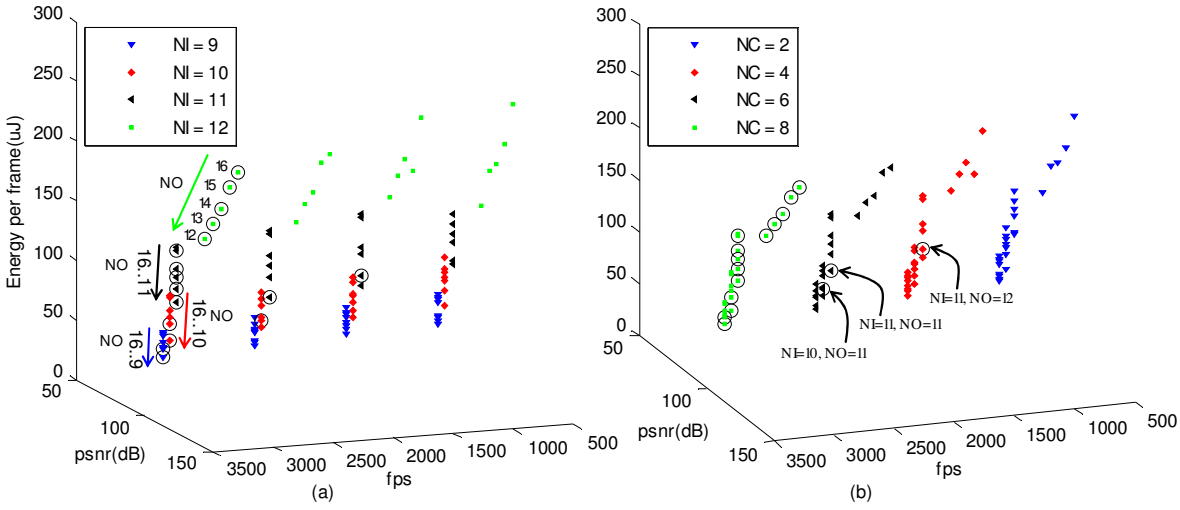

Figure 2.11. Fixed-frequency (100 MHz) constrained implementations (12-bit image). EPA Results and Pareto Front (circled points)

Table 2.6: Fixed-frequency (100 MHz) constrained implementations: Pareto Optimal points (12-bit image)

NI	NO	NC	psnr(dB)	fps	Energy per frame (uJ)
9	9	8	73.1611	3051.7578	29.4850
9	11	8	73.1667	3051.7578	36.0877
10	10	8	77.9215	3051.7578	46.6695
10	11	6	78.0665	2288.8184	57.8377
10	11	8	78.0665	3051.7578	59.8193
11	11	6	83.8819	2288.8184	81.0746
11	11	8	83.8819	3051.7578	81.6185
11	12	4	83.9695	1525.8789	92.4202
11	12	8	83.9695	3051.7578	93.7453
11	13	8	83.9751	3051.7578	102.4708
11	14	8	83.9875	3051.7578	110.5668
11	15	8	83.9922	3051.7578	125.3556
12	12	8	104.7546	3051.7578	146.9356
12	13	8	110.8823	3051.7578	163.8397
12	14	8	116.6600	3051.7578	179.2773
12	15	8	122.6959	3051.7578	201.4623
12	16	8	128.5966	3051.7578	217.2102

2.7.4 Dynamic PPA and EPA management optimization

Given the Pareto-optimal implementations, we are now ready to provide results on dynamic PPA management (DPPA). DPPA management allows us to provide optimized solutions based on time-varying constraints. The basic idea is to select Pareto-optimal implementations that satisfy the constraints and then implement them using DPR and dynamic frequency control (see Section 2.5.6).

First, note that performance constraints are relatively easy to meet since frame rates are always above 300 fps. This motivates the simplification of PPA management to the case of Power-Accuracy management. In this case, for 32-bit I/O and 8-bit input constrained implementations, the Pareto front is obtained for the lowest frequency of 33 MHz. For the fixed-frequency constrained implementation, the Pareto front is obtained for implementations with NC=2.

For finite energy applications (e.g. battery-operated), we are very interested in dynamic EPA management. As mentioned in Section 2.7.3, the Pareto front for EPA

optimization occurs for the maximum frequency of 100 MHz. Also, in this case, performance constraints are still easy to meet. Thus, we switch to optimization in the Energy-Accuracy space as shown in Fig. 2.12(a) for fixed-frequency constrained implementations.

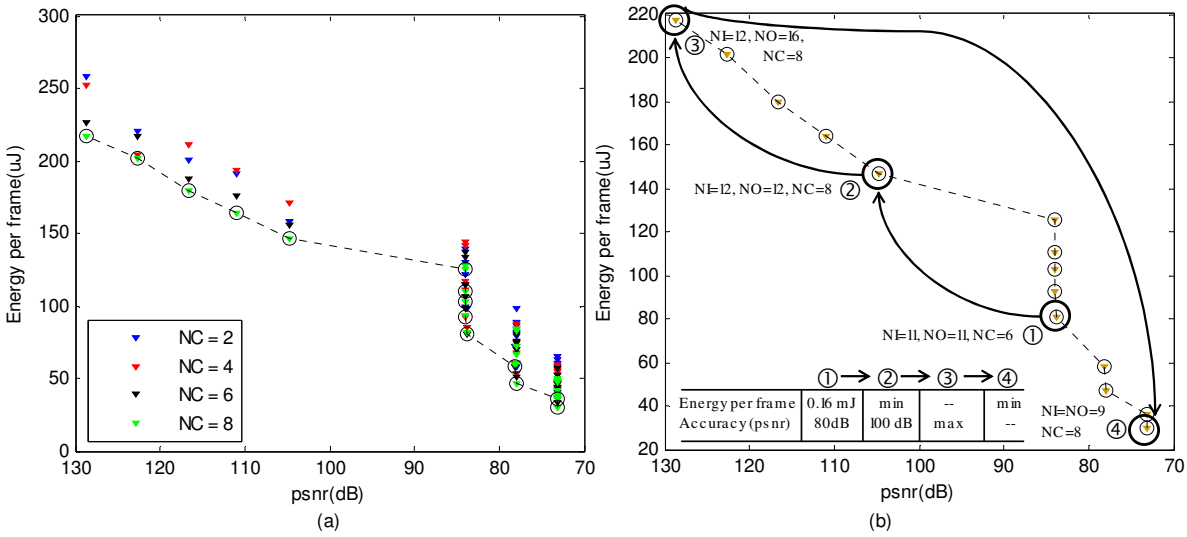

Figure 2.12. Fixed-frequency (100 MHz) constrained implementations (12-bit image): (a) Energy-Precision results and Pareto front (dotted line). (b). Pareto Front and Dynamic control example with 4 points.

To demonstrate dynamic Energy-Accuracy management, we consider an example with time-varying constraints. Sequentially, we list the dynamic constrained and unconstrained optimization requirements as follows:

1. Require Accuracy≥80dB and Energy≤0.16mJ per frame.
2. Minimize Energy subject to Accuracy≥100dB.
3. Maximize Accuracy.
4. Minimize Energy consumption.

The resulting Dynamic Energy-Accuracy management is demonstrated in Fig. 2.12(b). First, we choose the implementation with NI=NO=11, NC=6 that meets the constraints while also minimizing energy consumption (see point '1'). The rest of the constraints are

met by the realizations marked as ‘2’, ‘3’, and ‘4’ in Fig. 12(b). The full DPR implementation details are given in Section 2.5.6.

In the proposed solution, we do recognize that we cannot have ‘hard’ accuracy constraints. Clearly, accuracy varies from frame to frame. The recommendation is simple. The user can dynamically increase or decrease the accuracy constraint based on whether his or her expectations are met. In the example, the user specifies an increase in accuracy from 80 dB (constraint # 1) to 100 dB (constraint # 2).

2.7.5 Comparison with other pixel processor implementations

Table 2.7 provides a comparison between the 8-bit input/8-bit output core and similar implementations found in the literature. Clearly, this comparison does not capture the rich number of implementations described here. However, it provides a reference point that can be used to measure the effectiveness of the basic LUT implementation.

The closest implementation to ours is the Xilinx® core [19]. In [19], the use of 3 BRAM18 resources can be considered as an expensive option. The implementations of [18] and [24] are custom static architectures whose resource consumption exceeds ours.

The implementation in [21] is a 12-bit input/8-bit output function that only uses 146 Slices and 1 DSP Slice. This is a custom-built implementation of the gamma-correction function alone and cannot be generalized to other single-pixel processors. While our 8-bit input/8-bit output implementation requires fewer resources, our 12-bit input/12-bit output case requires significantly more resources at 1662 Slices. In Fig. 2.6(c), we show that the reduction of the 12-bit input to an 8-bit input for use with the proposed approach of Table 2.7 can give satisfactory results. In other words, there are no visible artifacts between the proposed approach of Fig. 2.6(c) and the double floating point implementation of Fig. 2.6

(a). However, in general, we do not recommend the use of the LUT-approach for input bitwidth above 12 bits.

Table 2.7: Comparison against other single-pixel architectures (1 core). For comparing to [21], we note that the proposed 12-bit input/12-bit output core requires 1662 slices.

	Proposed	[19]	[21]	[18]	[24]
Function type	Programmable	Programmable	Precise gamma correction	Histogram equalization	Successive Mean Quantization Transformation
Implementation	LUT-approach	LUT-approach	Custom hardware	Custom hardware	Custom hardware
Test case	8-bit input 8-bit output	8-bit input 8-bit output	12-bit input 8-bit output	8-bit input 8-bit output	8-bit input 8-bit output
Device	Virtex-4	Virtex-5	Virtex-4	Virtex-II Pro	Actel APA600
Resources	128 LUT4, 16 FFs 68 Slices	57 LUT5, 57 FFs 3 BRAM18	146 Slices 1 DSP Slice	269 LUT4, 172 FFs, 16 BRAM 1 MULT18x18	2123 Cells 48 BRAM
Max. Frequency (IP)	229.358 MHz	324 MHz	378 MHz	200 MHz	-

2.8 Conclusions

We have presented a framework for generating Pareto-optimal PPA/EPA implementations based on PPA/EPA constraints. The framework allows for dynamic PPA and EPA management for single-pixel processing architectures. A dynamic reconfiguration system selects the Pareto-optimal realization that meets time-varying constraints.

The Pareto optimal points are computed offline by considering different clock frequencies, the number of pixel processor cores, and the number of inputs and output bits. The validity of the approach is verified by the fact that over 40% of the considered implementations are found to be Pareto-optimal. Furthermore, the scenarios provide practical implementations for 32-bit I/O, 8/12-bit inputs, and fixed clock frequency.

We also demonstrated dynamic EPA management for 12-bit bitwidths. The proposed

framework was used to show how we can meet dynamic constraints in energy and accuracy. Here, performance constraints were met by the fact that all implementations operated over 300 fps. In general though, when the pixel processor is implemented in a larger system, we expect that performance requirements may have to be added.

Chapter 3

Partial Reconfigurable FIR Filtering system using Distributed Arithmetic

Abstract

Dynamic partial reconfiguration (DPR) allows us to adapt hardware resources to meet time-varying requirements in power, resources or performance. In this chapter, we present two new DPR systems that allow for efficient implementations of 1-D FIR filters on modern FPGA devices. To minimize the required partial reconfiguration region (PRR), both implementations are based on distributed arithmetic. For a smaller required PRR, the first system only allows changes to the filter coefficient values while keeping the rest of the architecture fixed. The second DPR system allows full FIR-filter reconfiguration while requiring a larger PR region. We investigate the proposed system performance in terms of the dynamic reconfiguration rates. At low reconfiguration rates the DPR systems can maintain much higher throughputs. We also present an example that demonstrates that the system can maintain a throughput of 10 Mega-samples per second while fully reconfiguring about seventy times per second.

3.1 Introduction

Dynamically reconfigurable systems offer unique advantages over non-dynamic systems. Dynamic adaptation provides us with the ability to adapt hardware resources to match

real-time varying requirements. The majority of the 1-D FIR filtering literature is dominated by static implementations. Here, we use the term static to refer to both CMOS implementations (e.g. [29-33]) and reconfigurable hardware (non-dynamic) (e.g. [34,35]). Some implementations use the label reconfigurable in the sense of having the capability to load different filter coefficients on demand (e.g. [30-33]). In the context of this work, such implementations are considered static since the underlying hardware is not changed or reconfigured.

For reconfigurable hardware, the most efficient implementations are based on Distributed Arithmetic (DA) [36]. These filters have coefficients fixed or hardwired within the filter's logic. This approach allows fast and efficient implementations while sacrificing some flexibility since coefficients can not be changed at run time. Dynamic partial reconfiguration (DPR) can be used in this scenario to provide the flexibility of coefficients' values changes without having to turn off the device and only re-writing a section of the configuration memory. The efficiency of DPR over the full reconfiguration alternative and the savings in terms of power and resources is a function of the relative size of the portion being reconfigured [37].

We consider a DPR approach that allows us to change the filter's structural configuration and/or the number of taps. The proposed approach provides a level of flexibility that can not be efficiently accomplished with traditional static implementations. In particular, we develop a dynamically reconfigurable DA-based FIR system that uses DPR to adapt the number and value of the coefficients, the filter's symmetry and output truncation scheme. Two systems are presented that allow the flexibility to change all these filter's characteristics: (i) a system that only allows changes

to the coefficients values, and (ii) a system that allows changes to the number and value of the coefficients, the symmetry, and the output truncation scheme.

Previous research on dynamically reconfigurable FIR filters has focused on Multiply-Accumulate based implementations and coarse reconfiguration. The first system described in this work is based on dynamically reconfiguring at a coarse level, i.e. the entire FIR filter. The second system is based on dynamically reconfiguring at the finest possible level, the LUTs that store the coefficients, with a small dynamic reconfiguration area. We have demonstrated a related, LUT-based approach in a dynamically reconfigurable pixel processor [9]. The paper also explores different ways to execute dynamic partial reconfiguration and elaborates on the impact over reconfiguration time overhead of the different approaches.

This work provides an extended version of the conference paper presented in [38]. The work has been extended to provide: (i) extended background information, (ii) more implementation details, (iii) extended methodology, (iv) architectural extensions to allow changes on the filter's internal structure, and (v) new results.

The rest of this chapter is organized as follows: Section 3.2 presents background and related work. Section 3.3 describes the FIR filter core implementation. Section 3.4 introduces the dynamically reconfigurable system. Results and conclusions are presented in section 3.5 and 3.6 respectively.

3.2 Background and related work

Reconfigurable logic has established itself as a popular alternative to implement digital signal processing algorithms [39]. Furthermore, a number of articles have been published

on using DPR to implement different signal processing algorithms [40, 41, 38, 37]. In particular, [42-44] report different approaches for taking advantage of DPR in FIR filter implementations. The capability of reconfiguring a filter at run time is of special interest for applications such as wireless communications and software radio.

Hardware realizations of FIR filters can be divided into constant-coefficients and multiplier-based implementations [42]. In the latter case, DPR is mainly used to change a filter's overall structure [43, 44], or other filter-wide characteristic. At a higher level, DPR is also used to simply change the level of parallelism of an implementation by changing the number of filter cores in an application's critical path. In all these cases, changes are usually initiated from a desire to implement a new filter, based on power or resources considerations, or simply to obtain new functionality. A change in coefficients does not require reconfiguration for this type of filter implementation. Thus, for these cases, DPR has milder constraints in terms of reconfiguration speed and reconfigurable logic partition.

The case of constant-coefficients implementation is considerably more complex since DPR is used to change inner characteristics of the filters (coefficients are not easily isolated within the filter structure). This requires more complex schemas to segment logic into reconfigurable tiles and more efficient reconfiguration mechanism in order to reduce the amount of time it takes to reconfigurable a filter.

DA filters in Xilinx® FPGAs are introduced in [45, 46], where the authors exploit common characteristics between the Xilinx's FPGA architecture and the filter architecture. In [35], the authors present other approaches for flexible FPGA

implementations of FIR filters by combining pipelined multipliers and parallel, distributed arithmetic.

In [42], the authors consider different DPR architectures for extending constant-coefficients approaches to implement adaptive filters. This relatively early study already provides insights on the advantages of using run-time partial reconfiguration to modify a filter's behavior at run-time. The study used an earlier device (currently unavailable) and explored architectures different than DA, which were a natural fit for such device. Their results in terms of performance can not be compared to the results of this work due to the inherent difference between the reconfigurable devices used.

In [44] the authors describe a self-reconfigurable adaptive FIR filter system composed of up to three multiplier-based filter modules. These modules can be reconfigured at run-time by a control manager that uses SystemACE to store and fetch the corresponding partial bitstream. This system only allows a full filter reconfiguration instead of finer reconfiguration schemas such as coefficient-only reconfiguration. In this paper, speed results are not clearly presented. The authors report different reconfiguration overhead times for different filters that apparently occupy the same reconfigurable region in the device. These results are surprising since reconfiguration time overhead depends mainly on the bitstream size, which depends on the size of the partial reconfigurable area, not on the number of resources used within that area. It is also worth mentioning that reconfiguration speeds reported are slower than speeds reported on other DPR papers [10, 6].

In [43] a similar system is described although in this case it is not self-reconfigurable and uses an external PC to perform reconfiguration. Reconfiguration times reported are also considerably slower than other reported methods.

In [47], the authors describe a tool-flow to map applications to a self-reconfiguring application. The authors use a 32-tap MAC based FIR filter as an example. The paper compares the performance of simply reloading coefficients by writing over specific registers and using DPR to reconfigure the whole filter. In this paper, the reconfiguration time overhead is large but dismissed as an acceptable handicap for the paper's goals.

In general, the reconfiguration time overhead is an important factor in the evaluation of systems using DPR. Several approaches exist to deal with the overhead. One approach is to hide it by using efficient hardware scheduling strategies (e.g [48]). A more simplified approach is to select carefully the elements of an architecture that requires reconfiguration for a desired change in functionality (e.g [6,38]). By doing so, one can reduce drastically the size of the partial bitstream used to execute the DPR, thus reducing the reconfiguration time overhead. Finally, there is also the approach of maximizing the access speed to the configuration memory (e.g [10]). Unfortunately this approach has a limit determined by the device. In the case of Virtex-4 FPGAs the maximum speed is 3.2Gbps (32 bit wide bus @100MHz). A combination of the last two approaches is used in this work to deal with reconfiguration time overhead.

This work seeks to extend prior research in this area by primarily focusing on developing, analyzing, and improving DPR systems in terms of the dynamic reconfiguration rate on modern devices. This leads us to consider a DA implementation that allows efficient implementations with small hardware footprints on modern FPGA

devices. Then, we consider a scalable approach where we have two systems: (i) a DPR system that allows for faster dynamic reconfigurations of coefficient values while fixing the number of taps, and (ii) a second DPR system that allows flexibility in the number of taps, the filtering structure, and truncation characteristics while allowing for a slower dynamic reconfiguration rate.

3.3 Stand-alone FIR Filter core implementation

A high performance FIR implementation based on Distributed Arithmetic is described in this section (also see [38]). The approach was coded in VHDL, so as to achieve a level of portability. Specific LUT primitives are employed when the system is compiled in Xilinx® devices. We will consider two dynamic realizations based on this core in Section 3.4.

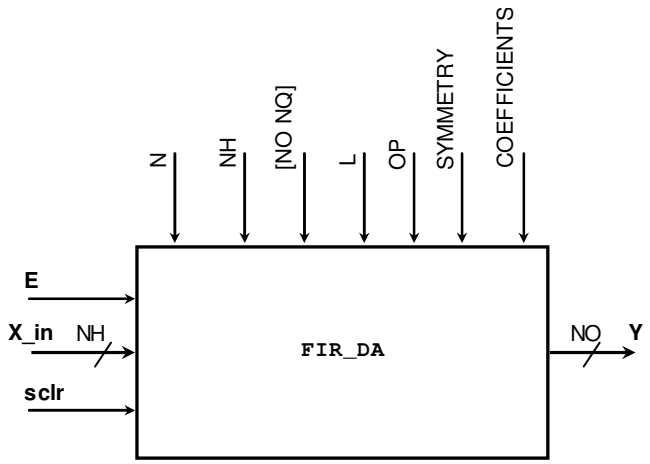

Figure 3.1. Generalized FIR DA Module

3.1. Description

The FIR filter module is shown in Fig. 3.1. It shows the FIR filter module with its inputs, outputs, and parameters. Signal ‘E’ controls the input validity. Clearing the register chain

('sclr' signal) at will is an important requirement when performing filtering on finite size signals.

Two filter implementations are presented in Fig. 3.2. A simplified approach is possible for symmetric filters (see Fig. 3.2) [49]. The more general, non-symmetric case is also presented in Fig. 3.2.

Here, N denotes the number of taps, NH represents the input/coefficients bitwidth, L is the LUT input size (explained in next subsection). We also use OP for controlling the output truncation scheme: (i) LSB Truncation then Saturation, (ii) LSB and MSB Truncation, and (iii) no Truncation. We use the parameter format [NO NQ] to denote the fixed-point output format for NO bits with NQ fractional bits. The filter coefficients are specified in an input text file.

We define $M = \lceil N/2 \rceil$, $sizeI = NH + 1$ for symmetric filters, and $M = N$, $sizeI = NH$ for non-symmetric filters. The inputs/coefficients format is set at [NH NH-1], which restricts values to $[-1,1)$. As a result, the maximum number of output integer and fractional bits results:

$$[2(NH-1)+\lceil log_2(N+1)\rceil+1 \quad 2(NH-1)] \tag{3.1}$$

3.3.2 FIR DA Implementation

The Distributed Arithmetic technique rearranges the input sequence samples (be it x[n] or s[n]) into vectors of length M, which require an array of sizeI M-input LUTs. This becomes prohibitively expensive when M is large. For efficient implementation, we divide the filter into M/L filter blocks [49], as illustrated in Fig. 3.2. Each filter block works on L coefficients requiring sizeI L-input LUTs (each vector of size L goes to one L-input LUT, see Fig. 3.3). Table 3.1 summarizes the resources savings associated with

the filter blocks approach. An advantage of using FIR filter blocks is that it allows for efficient routing while mapping the implementation to the specific LUT primitives found in an FPGA. As shown in [9], the approach is scalable in that can be easily ported to different LUT sizes.

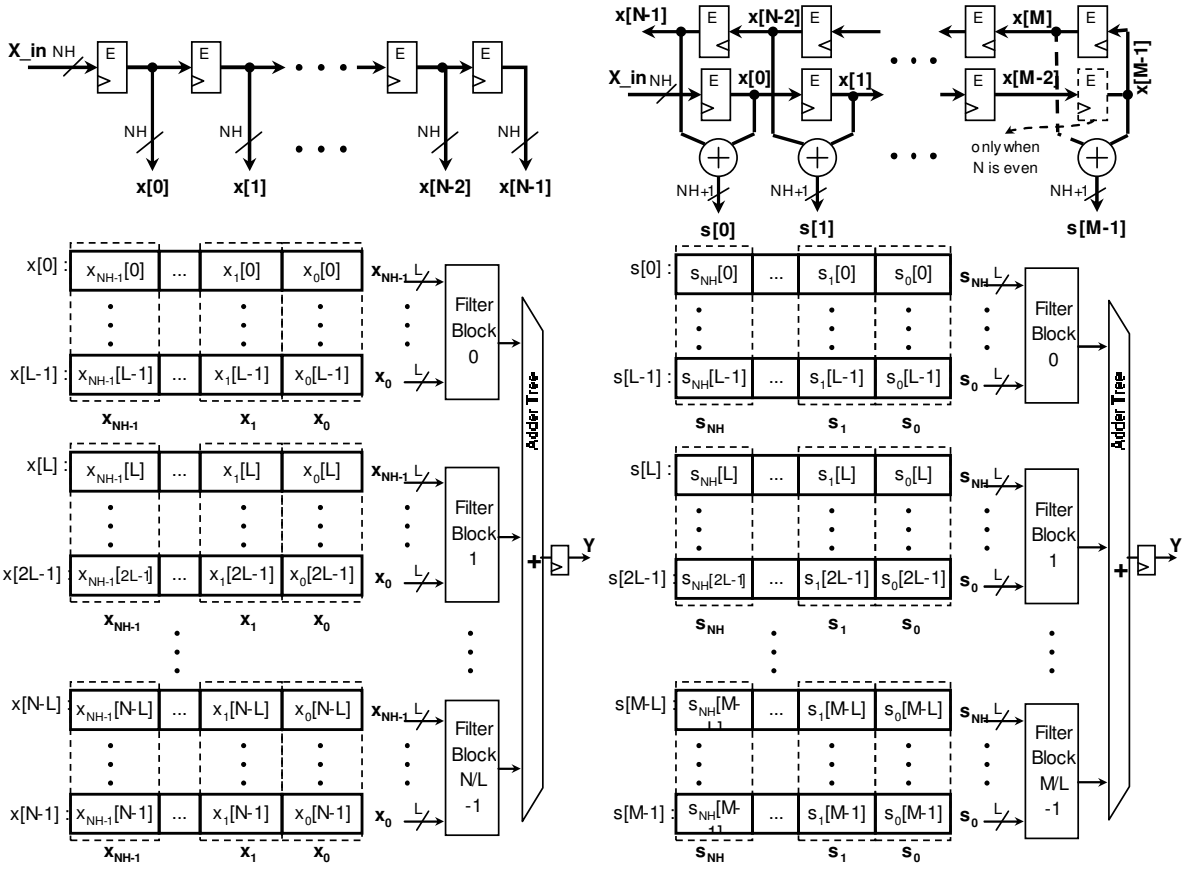

Figure 3.2. High-performance DA implementation based on the underlying LUT input size (L). Non-symmetric Filter (left) Symmetric Filter (right)

Table 3.1: FIR Filter implementation savings due to the use of filter blocks

Implementation	Resource requirements
1 Filter block of size M. LUTs have M inputs	$sizeI \times 2^{M}$ words
M/L filter blocks of size L. LUTs have L inputs	$\text{sizeI} \times 2^{\text{L}} \times \text{M/L}$ words

To demonstrate the savings, we consider a particular example. Using the formulae of Table 3.1, for M=16, L=4, we have significant savings since 2^{16} >> 2^{4}x16/4. It does

require an additional adder tree structure (see Fig. 3.2). However, compared to the savings, the overhead is not significant.

A pipelined implementation of a symmetric filter block example is shown in Figure 3.3. Here, we have the parameters SYMMETRY = YES and NH = 8. It consists of an array of L-input LUTs, an adder tree, shifters, and registers. The number of register levels is given by the following formula:

$$\#\ of\ register\ levels\ in\ Filter\ Block = \lceil log_2(sizeI) \rceil \tag{3.2}$$

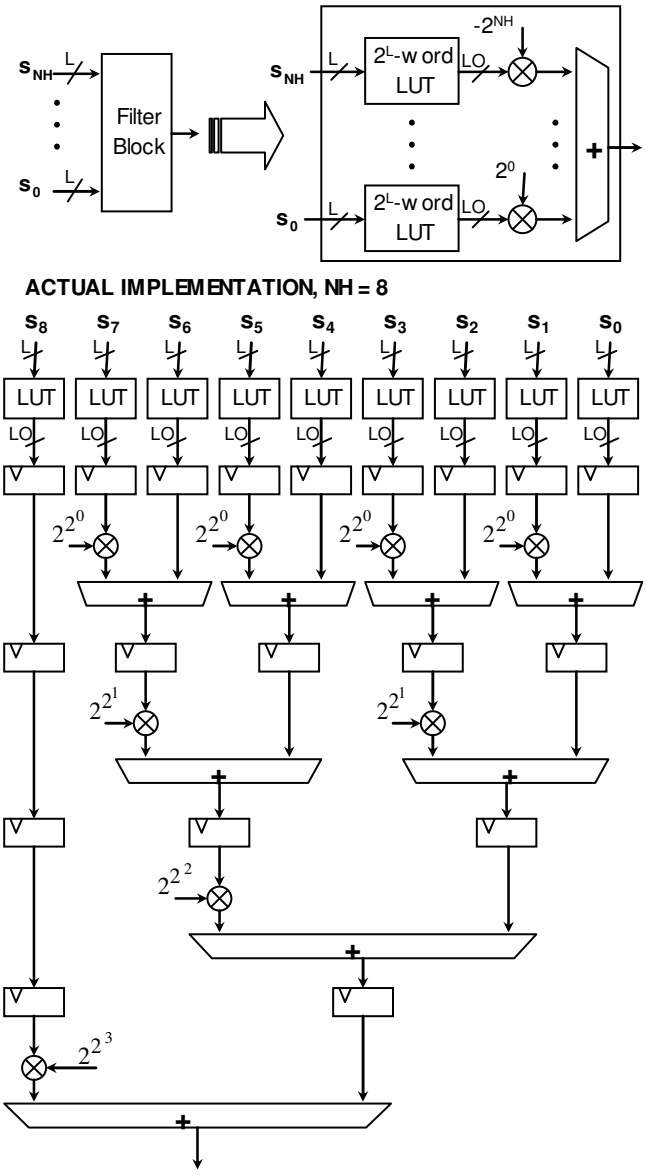

Figure 3.3. Filter Block architecture. SYMMETRY = YES

The L-input LUT sub-blocks are shown in Fig. 3.4. Here the output word size of each L-input LUT is given by $LO = NH + \lceil log_2(L) \rceil$. It also shows its decomposition into LO L-to-1 LUTs, useful for efficient FPGA implementation. Xilinx® FPGA devices contain L-to-1 LUT primitives with L = 4 (Spartan-3, Virtex-II Pro, Virtex-4) and L = 6 (Virtex-5). Thus, L = 4 or L = 6 are optimum values of choice. Moreover, as explained in [9] for Virtex-4, optimal LUT implementations can also be obtained for L = 5, 6, 7, 8.

Fig. 3.5 depicts the internal pipelined architecture of the adder tree that is used for adding the Filter blocks outputs. The result is stored in an output register. The number of register levels of the adder structure is given by:

$$\#\ of\ \ register\ levels\ in\ Filter\ \ Adder\ \ Structure = \lceil log_2(M/L) \rceil \tag{3.3}$$

Since we can quantize the LUT table values (i.e. the summations), rather than the coefficients, this FIR DA Implementation is slightly less sensitive to quantization noise than a normal implementation, with quantized coefficients. The latency of the pipelined system is shown in Fig. 3.6. The latency (input-output delay) is given by $REG\_LEVELS = \lceil log_2(sizeI) \rceil + \lceil log_2(M/L) \rceil + 2$ cycles, where REG_LEVELS is the number of register levels between the input and the output.

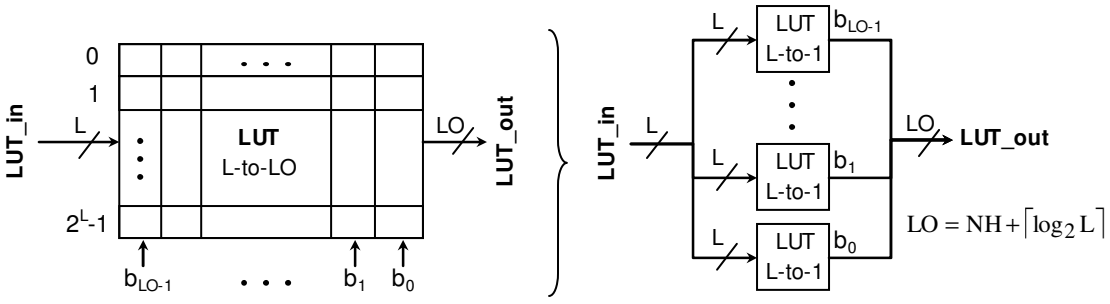

Figure 3.4. Realization of an L-to-LO LUT using LO L-to-1 LUTs

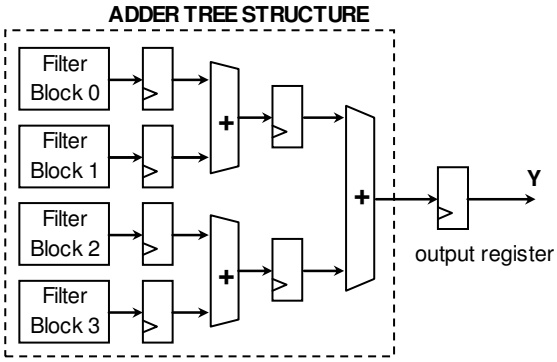

Figure 3.5. Adder tree structure for Filter blocks' outputs. M/L = 4

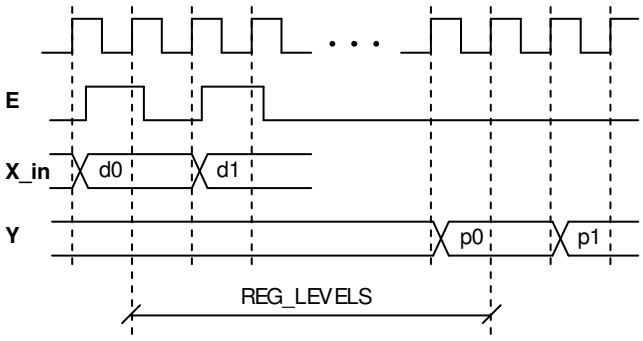

Figure 3.6. Latency measured from the moment 'd0' is input until its correspondent output 'p0' is available

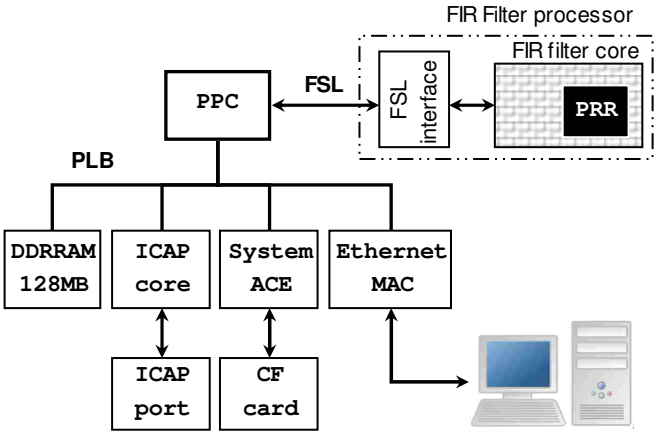

Figure 3.7. System Block Diagram

3.4 Dynamically Reconfigurable FIR Filtering System

The basic FIR filter core is now extended to be dynamically reconfigurable. We allow for the dynamic reconfiguration of both the number and the filter coefficients themselves in an embedded system. The basic system is shown in Fig. 3.7. By means of Dynamic Partial Reconfiguration, a constant coefficient FIR filter is turned into an adaptive FIR filter.

The basic approach requires that we pre-specify the Partial Reconfiguration Region (PRR). Two dynamically reconfigurable realizations are considered:

(1) *Coefficient-only reconfiguration*: The PRR allows modifications to the filter coefficient values, while keeping the rest of the architecture intact.

(2) *Full filter reconfiguration*: The PRR allows modification to the number of coefficients, the coefficient values, and the filter symmetry.

We start by describing the system architecture and FIR filter dataflow, which are not affected by the PRR definition. Then, we explain each of the dynamic realizations by providing a detailing representation of the PRR in the context of the FIR filter architecture.

3.4.1 System Architecture

From Fig. 3.7, we can see that the dynamic FIR core and the PowerPC (PPC) communicate using the high speed FSL Bus. The Partial Reconfiguration Region (PRR) is dynamically reconfigured via the internal configuration access port (ICAP), driven by the ICAP controller core.

The DDRRAM stores volatile data needed at run-time, e.g.: input streams, processed streams and partial bitstreams. At power-up, SystemACE reads a Compact Flash (CF) Card that stores the partial bitstreams and input streams. The processed streams are written back to the DDRRAM. The Ethernet core provides reliable communication with a PC, and allows us to get new partial bitstreams or new input streams, and to send processed streams to the PC for its verification or storage. Also, it serves as an interface for throughput measurements and system status.

Fig. 3.8 depicts the interfacing of the FIR filter processor and the PPC for both dynamic realizations. The FIR Filter processor, as shown in Fig. 3.7, consists of the FIR filter core and a control unit that provides interfacing with the 32-bitwide FSL bus. Fig. 3.8 shows a special case when the filter input size is NH = 8 bits. Here, the input is processed sample by sample (one byte at a time). After 32 output samples are computed, they are transmitted through the FSL bus. Other input/output bit-width configurations require different logic and control.

We next provide a description of the different possible modes of operation. First, we note that an FIR Filter with N coefficients and NX input values can output a maximum of NX+N-1 values. The three modes of operation are implemented through a finite state machine as follows:

- *Basic output mode*: The system computes the first NX output values. This mode is useful for finite 1D signals.
- *Symmetric output mode*: The system computes the central NX output samples (i.e., in the range $\lfloor N/2 \rfloor + 1 : NX + \lfloor N/2 \rfloor$). This mode is useful when performing 2D separable convolution on images.

- *Streaming mode*: with infinite number of input samples, i.e. NX = ∞

3.4.2 FIR Filter processor data flow

The FIR Filter processor receives and sends 32 bits at a time via the FSL bus. Due to the FIFO-like nature of the FSL bus [50], the PPC processor sends a data stream to FIFOw to be grabbed by the FIR Filter processor that in turn writes an output data stream on FIFOr to be retrieved by the PPC processor (see Fig. 3.8).

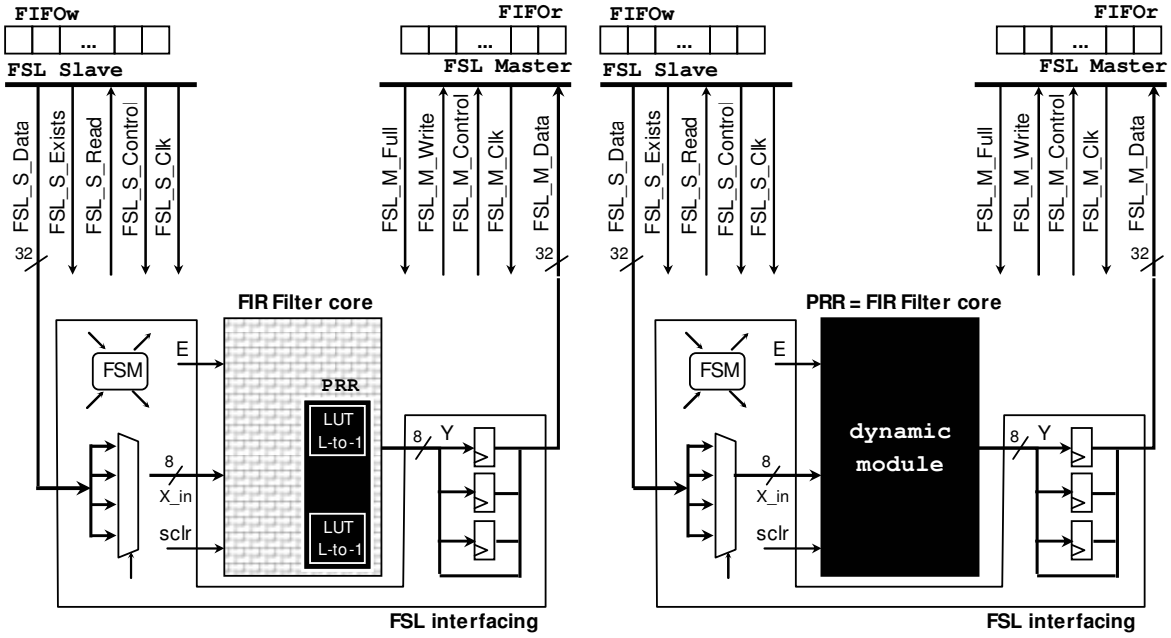

Figure 3.8. Dynamic FIR Filter processor interfacing with FSL. PRR for dynamic reconfiguration of the coefficients (left) and PRR for dynamic reconfiguration of the number of coefficients, their values and symmetry (right)

We optimize FSL bus usage by letting the PPC write a large block of data on FIFOw. The FIR Filter processor then processes the data and writes the results on FIFOr in a pipelined fashion. After reading all data in FIFOr, the PPC writes another large block of data on FIFOw, i.e. the PowerPC is busy only when reading/writing each large block of data. In addition, the FIR filter processor starts reading the next available block of data on

FIFOw right after writing a processed chunk of data on FIFOr. Each FIFO depth has been set to 64 words (32-bit words).

3.4.3 Dynamic Partial Reconfiguration Setup

Fig. 3.8 presents two dynamically reconfigurable systems and the associated PRRs. In the full-filter reconfiguration case, we do not allow any changes to the I/O bit-width. Here, we note that a change to the I/O bit-width would also require a generalized FSL interface to be included in the PRR, further complicating the design. Despite the complexity of doing so, this will be of interest for allowing us to build a dynamic precision system.

The static region is defined by everything else outside the PRR, including FSL interface, FSL circuitry, peripheral controllers, and the FIR filter core static portion (coefficient-only reconfiguration).

All signals between the dynamic region (PRR) and the static part are connected by pre-routed Bus Macros in order to lock the wiring. Also, the PRR I/Os are registered as the reconfiguration guidelines advise [51]. To perform DPR, the partial bitstreams are read from a CF card and stored in DDRRAM. When needed, they are written to the ICAP port. This fairly simple technique is explained in [6].

For throughput measurement purposes, the partial bitstreams and the input set of streams reside on DDRRAM. The streams are sent to the FIR Filter processor, and the output streams are written back to the DDRRAM. This process is repeated with different partial reconfiguration bitstreams loaded at specific rates, so as to get different filter responses and measure performance as the reconfiguration rate varies.

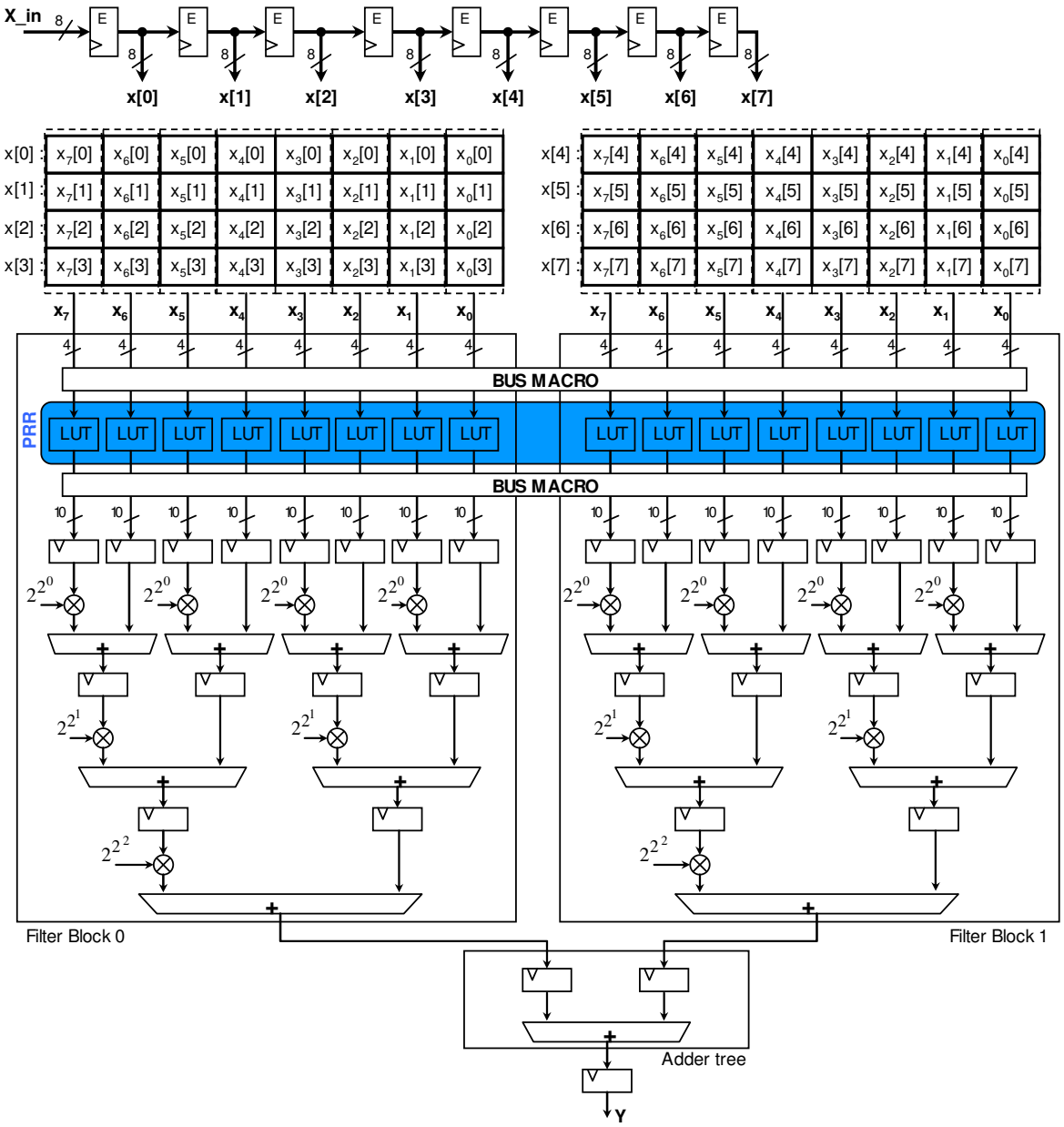

Figure 3.9. FIR filter core where the PRR and Bus Macros can be appreciated. Here, we can modify only the coefficients via the LUTs.

3.4.3.1 Coefficient-only reconfiguration

In this dynamic realization, the dynamic region is made of $(M/L)\times sizeI$ L-to-1 LUTs, resulting in a PRR with $(M/L)\times sizeI\times L$ inputs and $(M/L)\times sizeI\times LO$ outputs. Fig. 3.9 depicts

the PRR along with the Bus Macros when SYMMETRY = NO, NH = 8, N = 8, L = 4. The PRR is depicted in the context of the FIR filter core.

This realization is very useful for applications that only require filter coefficients modification, and it exhibits a smaller reconfiguration time overhead than the full reconfiguration case. Also, since only the LUT values are modified, the routing inside the PRR does not change. This has potential advantages in the area of run-time bitstream generation, as there is no need for run-time place-and-route operation. Fast routing is a very demanding task, and in most cases cannot be performed at run-time [52].

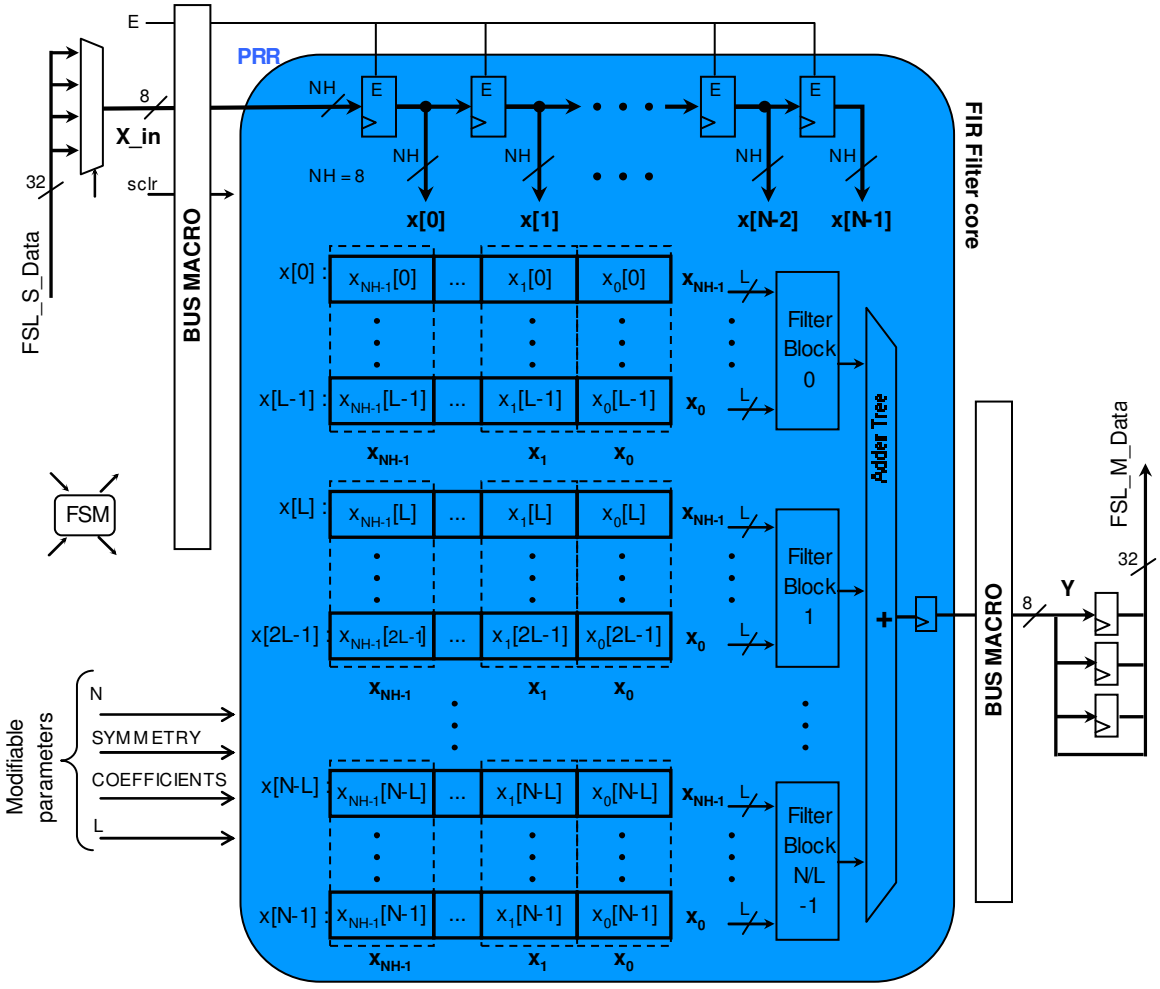

Figure 3.10. FIR filter processor where the PRR is the FIR Filter core. Note the parameters we can modify.

3.4.3.2 Full filter reconfiguration

In this case, the PRR involves the entire FIR filter core. It enables us dynamically modify the coefficients, number of coefficients, symmetry, and LUT input size. Fig. 3.10 depicts the PRR along with the Bus Macros in the context of the FIR Filter processor (with the FSL interface). We can see that the PRR has $NH+2$ inputs and NH outputs

3.5. Results

3.5.1 Stand-Alone FIR Filter core

Fig. 3.11 shows hardware resource utilization as a function of the number of coefficients (N), input bitwidth (NH), and symmetry (dotted lines: non-symmetric filters, solid lines: symmetric ones). Also, we set OP = 0, L = 4. Here, we use the XC4VFX20-11FF672 Virtex-4 device, with 8544 slices.

In addition, for each input bitwidth, we are considering the largest output format attainable (in the range $[-1,1)$). The output format ([NO NQ]) plays a negligible role in resource consumption (a difference of at most 12 slices).

Regarding frequency of operation, the goal of 200 MHz minimum frequency of operation was attained in all cases.

In addition, an error analysis is performed for the same parameters. Fig. 3.12 shows the relative error curves for three cases (input stream = 1024 sinusoid samples). The error metric is:

$$Relative\ error = \left| \frac{ideal\ value - FPGA\ output}{ideal\ value} \right| \tag{3.4}$$

Fig. 3.12 shows that in most cases the relative error is below 5%. The peaks correspond to FPGA values of zero and ideal values close to zero, resulting in a deceptive 100% error.

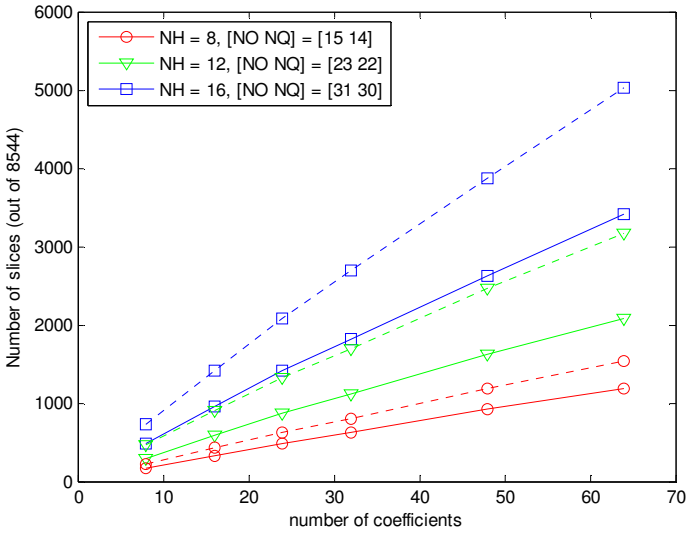

Figure 3.11. Resources vs number of coefficients and input bitwidth. Solid lines represent the symmetric case. Dotted lines represent the non-symmetric case

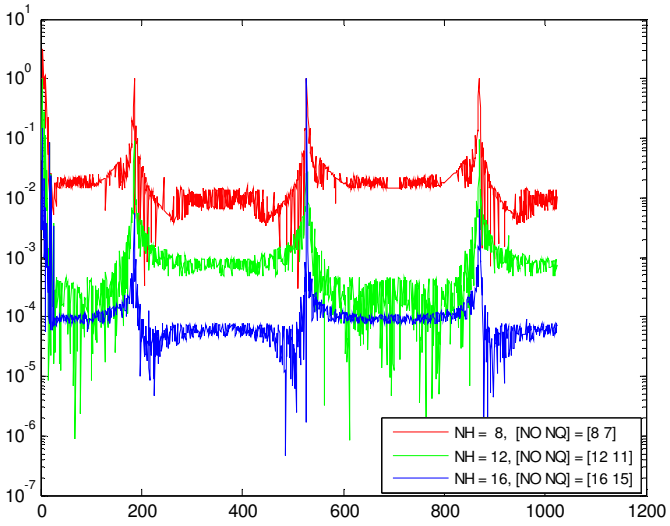

Figure 3.12. Relative error, N = 32. Three bitwidth cases.

3.5.2 Embedded System

Results are shown using the following FIR Filter core parameters: N = 32, NH = 8, [NO NQ] = [8 7], L = 4, OP = 0, SYMMETRY = YES.

The system is implemented on the ML405 Xilinx® Development Board that houses a XC4VFX20-11FF672 Virtex-4 FPGA. The PPC is clocked at 300 MHZ and the peripherals run at 100 MHz. In order to improve performance, the DDRRAM memory space is cached. Also, the dynamic systems are tested in the basic output mode, i.e. only the first NX outputs are considered.

*3.5.2.1 Hardware resource utilizatio*n

Results for this section depend on the specific dynamic realization. Tables 3.2 and 3.3 show hardware resource utilization for two DPR systems: (i) Coefficient-only reconfiguration, and (ii) Full filter reconfiguration. It shows the static region, dynamic region and the entire system resource usage. The module 'PRR interface' is the gluing static logic needed to join the static and dynamic regions.

As expected, the overall resource utilization is about the same. What varies is the static region size, which is larger in the coefficient-only reconfiguration case.

Table 3.2. Hardware Utilization on Virtex-4 XC4VFX20-11FF672 for coefficient-only reconfiguration

Module	FF	(%)	Slice	(%)	LUT	%
PRR	0	0%	180	2%	360	2%
Static Region	5303	31%	6130	72%	8698	51%
PRR interface	1313	8%	786	9%	885	5%
Overall	5203	31%	6310	74%	9058	53%

Table 3.3. Hardware Utilization on Virtex-4 XC4VFX20-11FF672 for full filter reconfiguration

Module	FF	(%)	Slice	(%)	LUT	%
PRR	1324	8%	818	10%	1306	8%
Static Region	4017	24%	5515	65%	8072	47%
PRR interface	6	0%	5	0%	107	1%
Overall	5341	31%	6333	74%	9378	55%

Table 3.4 shows the reconfiguration size and its partial bitstream size. Note that the PRR in the first case is somewhat larger than expected (about 62% of the second case). This can also be appreciated in Fig. 3.13 that shows the dynamic region (PRR) for both realizations, which are functionally the same.

Table 3.4. PRR measures for both dynamic partial reconfiguration system realizations

Dynamic Realization	PRR size (Slices)	Bitstream size (bytes)
1.Coefficient-only reconfiguration	90x6 = 540	43000
2. Full-filter reconfiguration	44x20 = 880	83000

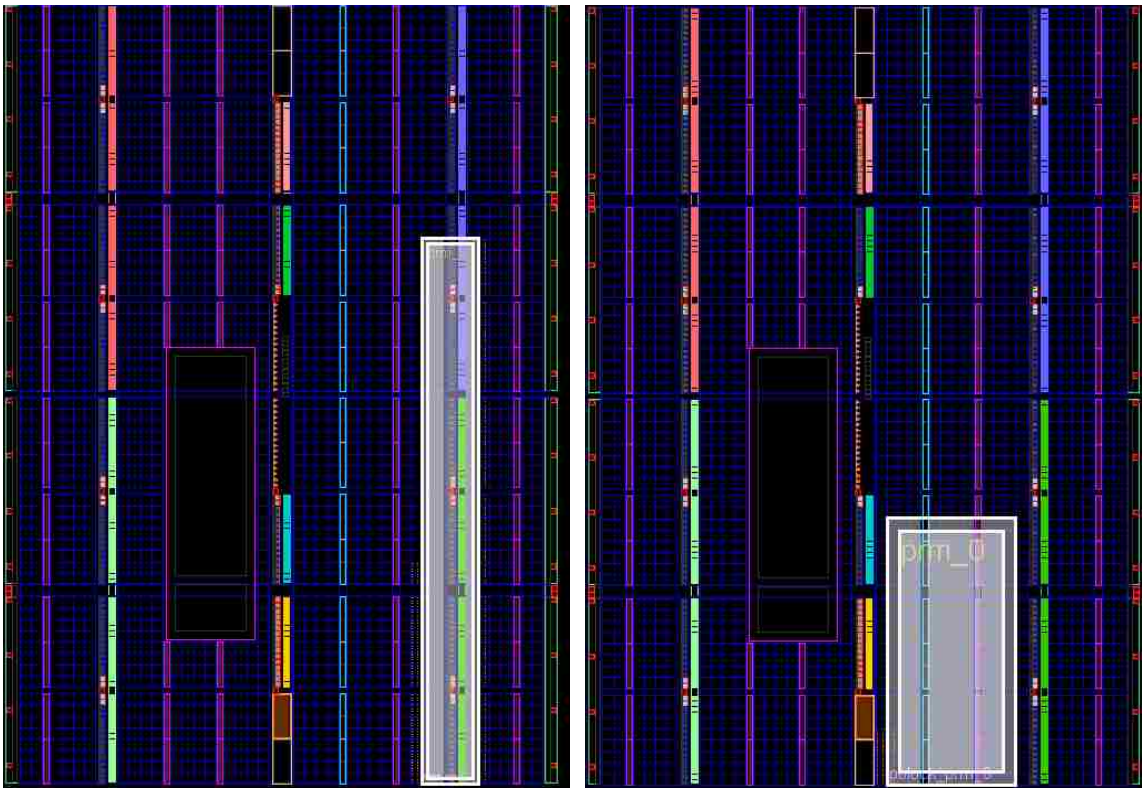

Figure 3.13. Dynamic reconfiguration region for (i) coefficient-only reconfiguration system (left), and (ii) full filter reconfiguration system (right)

The reason for the large PRR in the first case is the large number of required Bus Macros I/Os. In the coefficient-only reconfiguration case, the system needs access to the LUTs (see Section 3.4.3.1). As a result, for the special case shown, we require $(M/L)\times sizeI\times L = 4\times 9\times 4 = 144$ inputs and $(M/L)\times sizeI\times LO = 4\times 9\times 10 = 360$ outputs.

As explained in Section 3.4.3.2, in the second case (full filter reconfiguration), we only need $NH + 2 = 10$ inputs and $NH = 8$ outputs. So, the PRR in the first case is larger than

what it is actually needed for the L-to-1 LUT array, thereby wasting hardware resources in order to accommodate the large number of Bus Macros I/Os.

3.5.2.2 FIR Filter processor performance bounds

The maximum throughput of this particular FIR filter processor (NH = 8) is given by:

$$Max.\,Throughput = \frac{1\ byte}{1\ cycle} = \frac{8\ bits}{10\ ns} = 0.8\,Gbps \tag{3.5}$$

Note that since the system is pipelined, there is an initial setup delay that becomes negligible over time. Actual throughput depends on many factors, such as cache size, PPC instruction execution, and FSL usage. Note that the maximum throughput of (3.5) can not be attained since the PPC can not read and write into the FIFOs at the same time.

3.5.2.3 Reconfiguration Time

Table 3.5 shows the reconfiguration time for 3 scenarios. Both dynamic realizations are included. In our setup, called Scenario 1, we used the Xilinx® ICAP core and obtained a reconfiguration average speed of 3.28 MB/s. The reconfiguration time of Scenario 2 is computed based on the speed results reported in [47]. The dramatic improvement in reconfiguration lies on the use of a custom ICAP controller, DMA access, and burst transfers. Scenario 3 is the maximum theoretical throughput, which for the Virtex-4 is 400 MB/s [6].

Table 3.5. Reconfiguration time for both DPR system Realizations. The 43 KB Bitstream corresponds to coefficient-only reconfiguration case. The 83 kb bitstream corresponds to full-filter reconfiguration

Scenario	Reconfiguration Speed	Reconfiguration Time	
		43 KB bitstream	*83 KB* bitstream
1. Current	3.28 MB/s	13.10 ms	25.30 ms
2. Custom [10]	295.4 MB/s	0.145 ms	0.280 ms
3. Ideal	400 MB/s	0.107 ms	0.207 ms

3.5.2.4 Dynamic performance

We use software timers to measure the elapsed time from the moment we start reading the input stream from DDRRAM until the processed stream is written back on DDRRAM. We are considering sinusoids as the input stimuli. Please refer to Section 3.4.3 for some of the details that will be discussed in this section.

In order to evaluate the dynamic performance of the system, we use a stream of 102400 samples (1 sample = 8 bits). The stream is processed a number of times (100 runs). Within the 100 runs, partial bitstreams are loaded at a specific rate. Each partial bitstream amounts to a different filter response. Note that for the coefficient-only reconfiguration case, we only load a different set of coefficient values.

For the full-reconfiguration case, we switch between a filter with N = 32 coefficients and one with N = 16 coefficients. The PRR size is defined to be sufficiently large so as to allow implementation of the larger filter, i.e. the N = 32 filter case. The filter with N = 16 requires only one fewer latency cycle (Equation 3.3). As a result, the static performance improvement of the smaller filter is not significant.

We report the average throughput over the 100 runs. Here, we define the dynamic reconfiguration rate in terms of the inverse of the number o samples that are being processed prior to a hardware reconfiguration. For better visualization, we report throughput in terms of the number of processed Mega samples per second (MSPS). This corresponds to the inverse of the reconfiguration rate.

Figures 3.14 and 3.15 show the dynamic performance over 100 runs for both dynamic realizations. There are 3 curves that correspond to the 3 scenarios shown in Table 3.5. In the limit, at zero reconfiguration rate, we have static performance. The performance

results converge for the static case. From Fig. 3.14 (coefficient-only reconfiguration), we see that for Scenario 1 (actual measurements), the static performance resulted in 29.25 MSPS. At the maximum reconfiguration rate (one per stream), the dynamic performance was 6.16 MSPS. The other curves (Scenarios 2 and 3) provide performance bounds based on the static performance and reconfiguration speeds of Table3.5.

We can see that the dynamic performance of the full filter reconfiguration case is slightly lower than the coefficient-only reconfiguration. This is due to differences in the PRR size. But as we increase the number of samples before a reconfiguration, or use a Scenario other than the first one, this effect is less noticeable.

As expected, the dynamic performance heavily depends on reconfiguration speed and input stream size. Better reconfiguration speeds offset the reconfiguration time overhead (Scenarios 2 and 3). We have the same effect for smaller dynamic regions. The slower reconfiguration rates due to longer data streams help to offset the reconfiguration overhead as well.

In Table 3.6, we present the full filter reconfiguration system throughput as a function of the time between reconfigurations. It is quite clear from the results that even for the slowest scenario, we can maintain throughputs over ten MSPS while dynamically reconfiguring seventy times per second.

Table 3.6. DPR system throughput (MSPS) as function of delay between reconfigurations for 1-D FIR filtering with Full filter reconfiguration

Scenario	Number of samples between reconfigurations				
	2048K	1024K	409.6K	204.8K	102.4K
	Amount of time between reconfigurations				
	68ms	**34ms**	**13.6ms**	**6.8ms**	**3.4ms**
1. Current	21.9	17.2	10.5	6.3	3.5
2. Custom [10]	29.9	29.8	29.5	28.9	27.8
3. Ideal	30.0	29.9	29.6	29.2	28.3

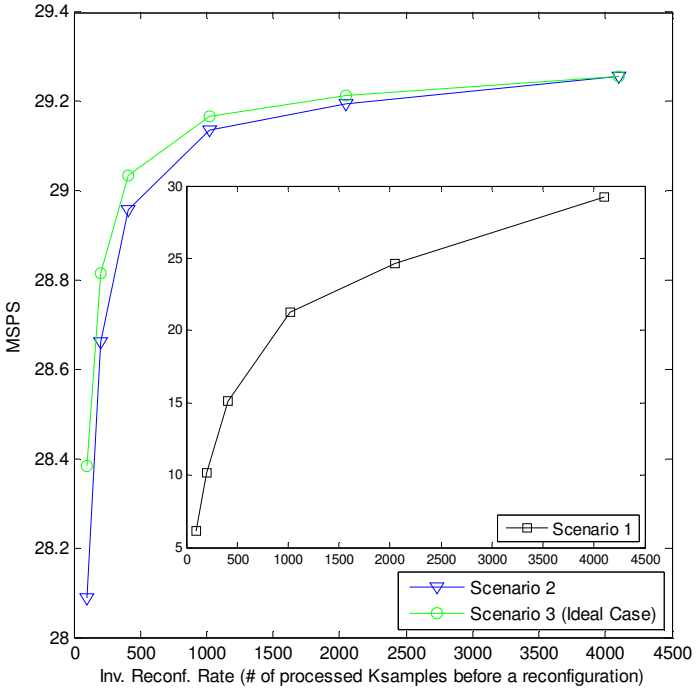

Figure 3.14. DPR system performance for coefficient -only reconfiguration

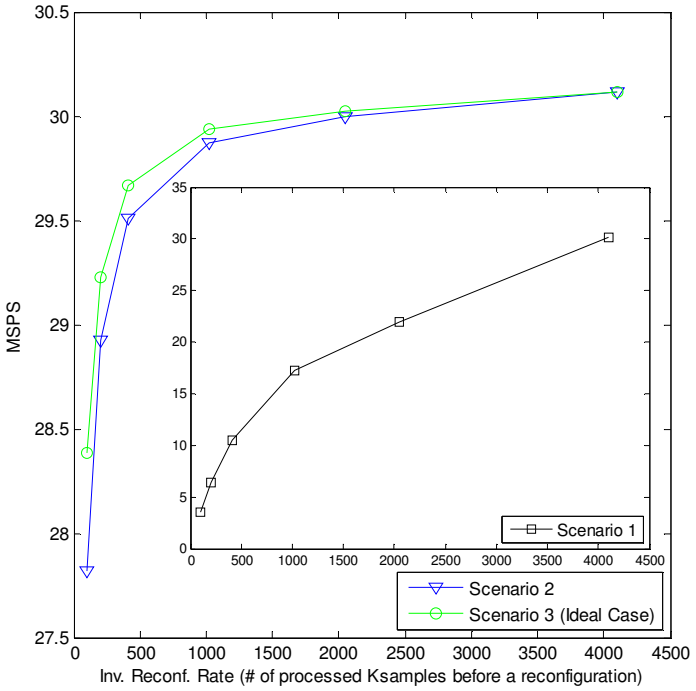

Figure 3.15. DPR system performance for full filter reconfiguration

3.5.3 Experimental results with ECG processing

We present an example application for electrocardiogram (ECG) characterization (R-wave detection). Here, we consider coefficient-only reconfiguration for implementing a 3-channel, 1-D filterbank. We make use of the embedded system detailed in Section 3.5.2. Each channel filter is symmetric, with 32 8-bit coefficients for 8-bit I/O, using truncation (saturation) arithmetic for the outputs. The approach here is to implement a variation of the ECG processing algorithms presented in [53]. ECG signal processing is of great interest for emergency applications, including the detection of cardiac arrhythmias [54] and stenosis assessment for atherosclerotic plaque video analysis [55]. A popular approach based on [53] is to use the outputs of a Wavelet filterbank for ECG analysis. As in Wavelet analysis, we design a dyadic filterbank to cover the entire, discrete frequency space. We have a high-pass filter with a positive frequency pass-band from $\pi/2$ to π, a band-pass filter from $\pi/4$ to $\pi/2$, and a low-pass filter for frequencies up to $\pi/4$. For each channel filter, we consider efficient implementations using 32 8-bit coefficients. The magnitude response of the designed filterbank is shown in Fig. 3.16.

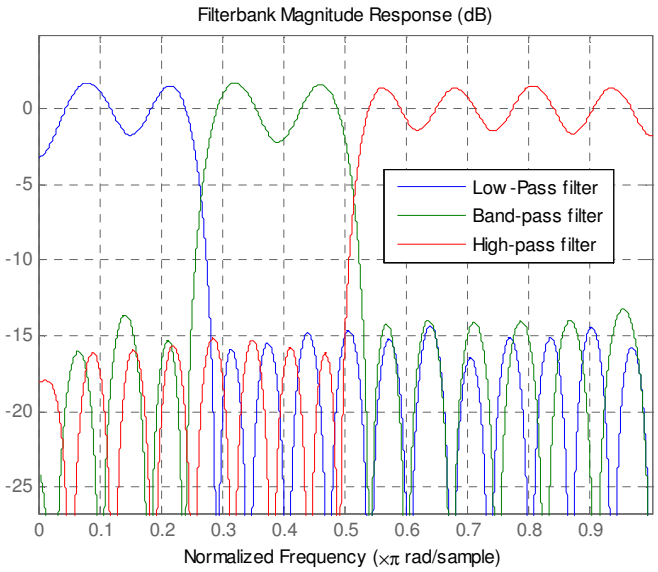

Figure 3.16. Filterbank used for R-wave detection

For testing the implementation, we use the first recording (record 100) from the MIT arrhythmia database [56]. In this record, we have 2 channels with 650K samples sampled at 360 Hz and quantized at 11-bits over a 10mV range. We further quantized the input down to 8 bits, downloaded them to the DDRRAM using the Ethernet core and tested using the procedure outlined in Fig. 3.17.

Based on [53], we implemented a simple R-wave detection algorithm. For detection, we look for thresholds in the outputs. In the example of Figure 3.18, we threshold as follows: Low pass filter ($[-1/32, 1/32]$), band-pass ($[-1/32, 1/32]$), high-pass ($>1/64$). This results in perfect R-wave detection for the first 5 cycles of the second channel (1500 samples). We refer to [53] for more details on how to adjust thresholds in such algorithms for near-perfect results verified over the entire database. The goal is to simply demonstrate the DPR FIR system on real signals.

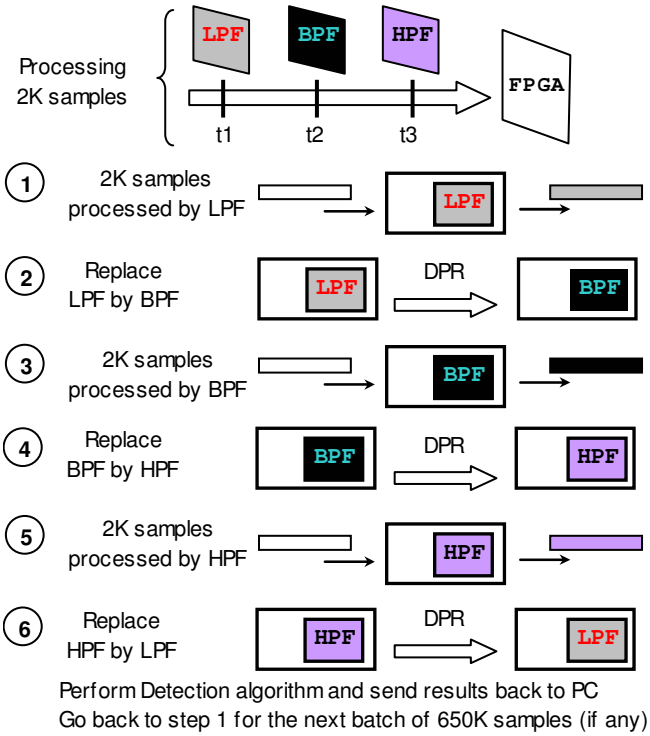

Figure 3.17. Filterbank data processing. LPF: Low-pass filter, BPF: Band-pass filter, HPF: High-pass filter

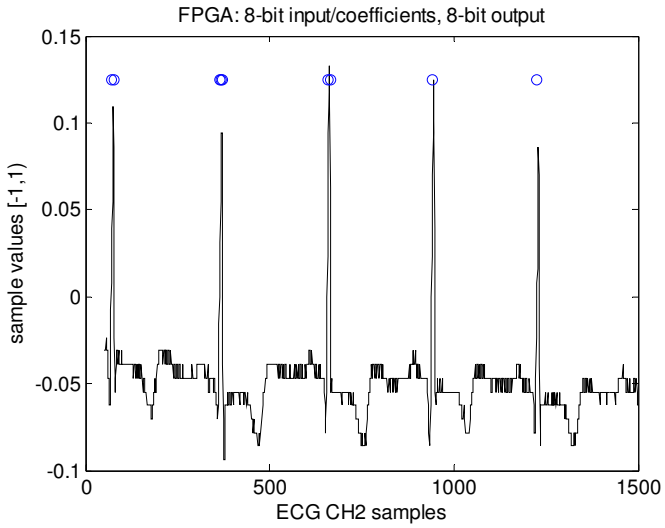

Figure 3.18. Perfect detection of R-waves for the first 5 ECG cycles

The detection algorithm is included in the embedded PowerPC software routines, and the resulting signal is stored into the DDRRAM. We note that performance improves with larger input signals (see Fig. 3.14). The detection algorithm is performed at the end of the operations, and takes about 80 ms. Dynamic reconfiguration of a channel filter requires 13.1 ms. At a sampling rate of 360 Hz, the system allows significant time for implementing real-time detection algorithms and DPR. As a result, the number of samples that are processed prior to reconfiguration can be significantly reduced. By processing every 2000 samples, the processing rate stands at 4.62 MSPS (2000 samples takes 140 ms to process). Thus, after 5.5 seconds spent in acquiring 2000 samples, we get a detection response in 140 ms.

3.5.4 Comparison with other PR systems for FIR filtering

The majority of previously reported work on FIR filtering is based on multiply-and-add approaches [41, 43, 44, 47]. In [41], the authors reported a reconfiguration time of

1.5 ms for changing the coefficients and their number on a Virtex-II (74.7 KB bitstream). In [43, 44], the authors presented a DPR FIR system that only allowed for changes in the number of coefficients. Reconfiguration time for 8794 slices for a 20-tap filter required 700 ms. The filter presented in [47] most closely resembles our FIR filter: 32-taps, 8-bit coefficients, 8-bit input, but with multiply-and-add approach. It required 1985 LUTs for a 13.1 ms reconfiguration time. We can change the entire filter using a 83KB bitstream for a reconfiguration time of 25.3 ms.

As mentioned earlier, for FPGA implementations, the distributed arithmetic presented here is far better suited than these multiply-and-add approaches. DA approaches allow for efficient use of hardware resources. Beyond this, multiply-and-add approaches tend to have fixed input/output characteristics as opposed to the flexible, dynamically reconfigurable arithmetic representations presented here.

The constant-coefficient filter with DPR is mentioned in [42], but the work is more theoretical and the results are non-comparable with ours, as stated in Section 3.2.

3.6 Conclusions

We presented two efficient dynamic partial reconfiguration systems that allow us to implement a wide range of 1-D FIR filters. Requiring a significant smaller partial reconfiguration region, the first system allows changes to the FIR filter coefficients while keeping the rest of the architecture intact. Using a larger partial reconfiguration region, the second system allows full filter reconfiguration. This system can be used to switch between FIR filters based on power, performance, and resources considerations.

For both systems, the required partial reconfiguration region is kept small by using Distributed Arithmetic implementations. System performance is evaluated in terms of the dynamic reconfiguration rate. For a representative example, it is shown that we can process over ten Mega samples per second while dynamically reconfiguring about seventy times per second. The introduction of faster dynamic reconfiguration controllers can lead to much higher throughputs for the same number of reconfigurations per second. Alternatively, we can maintain much higher throughputs at much lower reconfiguration rates.

The results have encouraged us to explore the use of dynamically reconfigurable filtering for digital image and video processing applications. As seen from the results of this work, it is possible to dynamically reconfigure at real-time frame rates. For such applications, the DPR systems can be extended to separate implementations of 2-D dynamically reconfigurable filterbanks.

Chapter 4

Separable FIR Filtering in FPGA and GPU Implementations: Energy, Performance, and Accuracy Considerations

Abstract

Digital video processing requires significant hardware resources to achieve acceptable performance. Digital video processing based on dynamic partial reconfiguration (DPR) allows the designers to control resources based on energy, performance, and accuracy considerations.

In this chapter, we present a dynamically reconfigurable implementation of a 2D FIR filter where the number of coefficients and coefficients values can be varied to control energy, performance, and precision requirements. We also present a high-performance GPU implementation to help understand the trade-offs between these two technologies.

Results using a standard example of 2D Difference of Gaussians (DOG) filter indicate that the DPR implementation can deliver real-time performance with energy per frame consumption that is an order of magnitude less than the GPU. On the other hand, at significantly higher energy consumption levels, the GPU implementation can deliver very high performance.

4.1. Introduction

Hardware implementations of digital video processing methods are of great interest because of the ubiquitous applications. In terms of performance acceleration, many image and video processing algorithms require efficient implementation of 2D FIR filters [57].

Dynamic partial reconfiguration (DPR) allows FPGA designers to explore different implementations based on energy, performance, and accuracy requirements. In addition to DPR, efficient filter implementations are based on the direct use of LUTs [9], the distributed arithmetic technique [17], and separable designs [58,59].

On the other hand, Graphic Processing Units (GPUs) offer high-performance floating point capabilities at significant energy consumption levels [60]. With the introduction of OpenCL and CUDA (Compute Unified Device Architecture), there has been a significant growth of GPU implementations; with [61] and [62] as examples of 2D FIR implementations.

In this work, we are interested in exploring the energy, performance, and accuracy trade-offs between DPR FPGA and the corresponding GPU implementations. Some trade-offs have been explored in [62] and [63]. The goal is to provide recommendations for different implementations based on specific energy and performance requirements.

This work is organized as follows: Section 4.2 describes the embedded filter implementation on the FPGA. Section 4.3 details the GPU filter implementation. Section 4.4 explains the measurement setup for both implementations. Section 4.5 presents the results in terms of Energy, performance, and accuracy. Finally, Section 4.6 summarizes the work.

4.2 2D FIR Filter System on the FPGA

We consider a 2D separable filtering implementation that is an extension of prior work presented in [17] and [58]. Here, we extend prior research to allow DPR of the entire FIR core, its FSL (Fast Simplex Link) bus interface, and the Partial Reconfiguration Region (PRR) control interface.

4.2.1 System Architecture

Figure 4.1 depicts the block diagram of the embedded system. The 1D FIR Filter processor core and the PowerPC (PPC) interact via the Fast Simplex Link (FSL) bus. The PRR is reconfigured via the internal configuration access port (ICAP). The Compact Flash (CF) card holds the partial bitstreams and input data. Bus macros are no longer needed in the Xilinx ISE 12.2 Partial Reconfiguration Tools.

In the context of the embedded system of Fig. 4.1 a 2D separable filter is realized by i) filtering the rows, ii) turning a row filter into a column filter via DPR, and iii) filtering the columns. Figure 4.2 depicts this scheme, where the 2D filter can be modified at run-time by using a different pair of row and column filters.

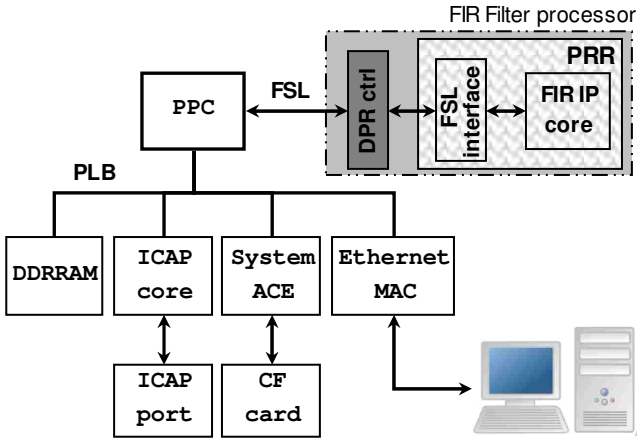

Figure 4.1. System Block Diagram

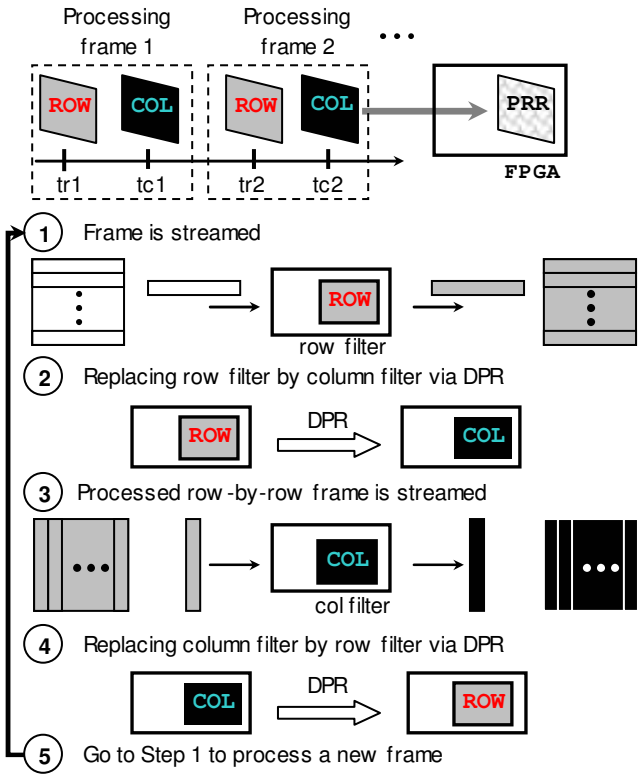

Figure 4.2. 2D separable FIR filter implementation.

The filtered images (row and column-wise) are stored in the DDRRAM. The row-wise filtered image is transposed in the memory before it is streamed to the column filter. The system is implemented in the ML405 Xilinx Dev. Board that houses a XC4VFX20 Virtex-4 FPGA. The PPC is clocked at 300 MHz and the peripherals at 100 MHz.

4.2.2 1D FIR filter core

This fixed-point core is based on the one presented in [17]. We allow for full-reconfiguration, i.e. the entire filter is included in the PRR. Several modifications are introduced:

- A new parameter 'B' allows the specification of the input data bit-width. This is different from the coefficients' bit-width.

- The frame size determines the input length of both the row and column filter. The input length is a parameter to the Finite State Machine (FSM) that controls the FSL interface. As a result, the FSL interface has to be included in the PRR (see Fig. 4.1).
- An interface that disables the PRR outputs during reconfiguration is required since the PRR outputs now include FSL interface signals (shown in Fig. 4.1).

Here, the 2D filter requires 2 bitstreams: one for the row filter and one for the column filter. The PRR must accommodate the largest filter.

4.3 Filter implementation on the GPU

We consider a parallel FIR algorithm implementation in the CUDA environment [64]. Here, parallelism is achieved by a grid that consists of blocks, with each block having a number of threads. All threads within a block are run in parallel from the software perspective. The actual number of blocks that can run in parallel is bounded by the number of streaming multiprocessors (SMs). Here, we can run a single block on each SM. Also, the number of threads that can be run in parallel at each SM is given by the number of CUDA cores inside each SM.

For the purposes of this work, we will report energy and performance measurements on the GPU (termed the device) as opposed to the CPU (termed the host). Here, GPU memory is divided into global memory, shared memory, constant memory, and texture memory.

The algorithm exposes and exploits parallelism of the 2D FIR filter in order to obtain significant speed up gains. It is based on ideas exposed in [61]. Double precision (64 bits)

is utilized. The filter symmetry and its separability are taken advantage of. The algorithm steps are summarized below:

1. The image and the filter kernel are transferred form the host to the device (global memory).
2. The image is then divided into blocks. Each image block is filtered by a thread block by rows.
3. The row-filtered image is also divided into blocks. Each image block is column-filtered by a thread block.
4. The final filtered image is transferred to the host.

To further describe the algorithm, we let the input image to be of size HxW (H rows by W columns) and a filter kernel of size KxK (row and column filter of same length). We refer to [61] for more details on the separable implementation. Performance is achieved based on: i) loop unrolling, ii) storing image blocks in shared memory, and iii) storing the filtering coefficients in constant memory.

Each image block is processed as follows: It is first loaded to the shared memory (with extra $\lfloor K/2 \rfloor$ pixels on both sides for correct filtering). Then, for row filtering, each thread inside a block performs a point-wise multiplication between the row kernel and a row portion of the image; and then adds up each product producing an output pixel. This process continues until the filtered image block is obtained.

Figure 4.3 shows the setup of a thread block for row filtering. Since all thread blocks work concurrently (from the software perspective), we are left with the row-filtering image in the global memory at the end of the previous process. This image (in blocks) is loaded again in shared memory, this time to perform column filtering. A thread block

does not transpose the column-ordered data since the image block is small and it is not worth the effort. Thus, this division of the image in blocks effectively avoids transposing the entire image prior to column filtering.

For row processing, the dimension of the thread block in the x direction must be higher or equal than $\lfloor K/2 \rfloor$ (effective size of the kernel). For the dimension in y direction, any power of 2 is suitable as long as H is its multiple. For column processing, the dimension of the thread block in the y direction must be higher or equal than $\lfloor K/2 \rfloor$. For the dimension in x direction, any power of 2 is suitable as long as W is its multiple.

The device utilized is a NVIDIA GeForce GTX465, with 11x32 CUDA cores running at 607 MHz. There are 11 Streaming Multiprocessors that run at 1.215 GHz, each with 32 CUDA cores. 1 GB of GDDR5 memory is available and runs at 1603 MHz with a bandwidth of 102.6 GB/s. There are 48K bytes of shared memory per block. The maximum power dissipation of the board is 200 W.

The GPUs are tested in a desktop environment with an Intel® Xeon W3520 running at 2.67 GHz, with 6GB of DRAM. The software configuration uses Windows 7 Ultimate (64-bits) with CUDA 3.2.

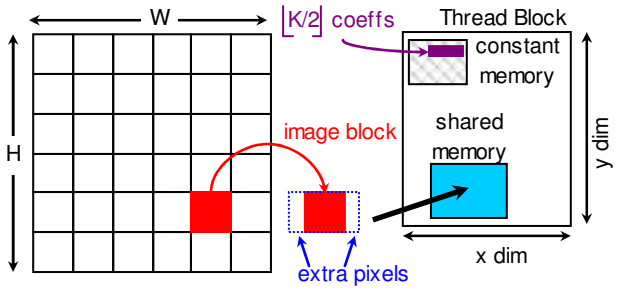

Figure 4.3. Thread block configuration for row filtering

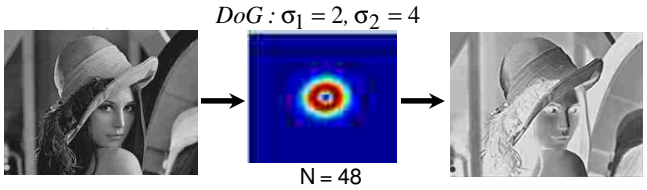

Figure 4.4. Frequency response – ideal filter with N = 48

4.4 Experimental Setup

This section details how the results were obtained. The set of filters for the test are first described. Then we detail how performance, energy, and accuracy were measured.

4.4.1 Set of filters for testing

To demonstrate the system, we consider a popular bandpass filter implementation based on the Difference of Gaussians (DOG) filter with $\sigma_1 = 2, \sigma_2 = 4$ [57]. For comparison, we implement the DOG filter using 48 coefficients and double precision arithmetic precision.

The input image selected is the standard grayscale level (8 bits) image 'Lena'. Fig. 4.4 shows the ideal frequency response of the filter with the input and output images.

We consider 6 filter implementations, each with a different number of coefficients (N = 8, 12, 16, 20, 24, 32). In addition, we consider 3 different frame sizes: 640x480 (VGA), 352x288 (CIF), 176x144 (QCIF), derived from cropped versions of 'Lena' (to preserve the frequencies). This results in 18 filtered images.

In the case of the FPGA implementation, the bit-width of the coefficients is set at 16 bits. The row filter receives 8-bit pixels at the input and outputs 16-bit pixels. The column filter receives and outputs 16-bits pixels, taking advantage of the symmetry of the filter [17]. The system switches to a different 2D filter via DPR. This is realized by reconfiguring a different row filter at step 4 in Figure 4.2. Then, having streamed the

image through the new row filter, we load the respective column filter at step 2. After that, we keep switching between these new row and column filters. As a result, the PRR size is that of largest column filter.

In the case of the GPU implementation, the system is implemented with double floating point numerical precision, although it can be programmed with fixed-point.

4.4.2 Energy, performance, and accuracy measurements

We measure performance in terms of frames per second (fps). In the case of the FPGA implementation, the processing time per frame includes: i) row filtering process, ii) column filtering process, iii) transposing row-filtered image, and iv) PRR reconfiguration (twice). The transposing of the row-filtered image occurs right after the filtering of the rows is completed. Two reconfigurations are needed per frame. Then, the performance (fps) is given by:

$$fps_{FPGA} = 1/\left(t_{rows} + t_{cols} + t_{transpose} + 2 \times t_{reconfig}\right) \tag{4.1}$$

In the case of the GPU implementation, the processing time per frame includes: i) Allocation of memory and data transfer from host to device, ii) Frame processing, and iii) data transfer from device to host. We run the filters 1000 times and get an average quantity of each of these times.

$$fps_{GPU} = 1/\left(t_{alloc+transf(\,h->d\,)} + t_{process} + t_{transf(\,d->h\,)}\right) \tag{4.2}$$

With regard to energy measurements, we consider the energy consumption per frame. In the FPGA case, the power spent by the three Virtex-4 FPGA power sources (VCCINT, VCCAUX, VCCO) is obtained, which amounts to the embedded system power consumption. We use the Xilinx Power Analyzer (XPA) tool that provides a more accurate estimate than the Xilinx Power Estimator (XPE) because it is based on simulated

switching activity of the place-and-routed circuit [27]. Our results are obtained with XPA at 25ºC. We get the power drawn by both the row (P_{row}) and column filter (P_{col}).

Each filter variation amounts to a difference in resource usage, and in turn in different power consumption. However, the filter core is small compared to the rest of the embedded system, so the power difference is not noticeable. As a result, it is more useful to consider the power drawn (both P_{row} and P_{col}) just by the FIR Filter IP core.

The power consumption during reconfiguration is an important quantity since the 2D FIR filter makes intensive use of DPR. Unfortunately, there is no tool available that can provide an estimate of this power consumption. In [6], hardware measurements determined that only the VCCAUX supply current increased during reconfiguration, and it increased by 25 mA for the XC4VFX12 device. This dynamic current does not depend on the device size, so we use this current for the XC4VFX20 device. The reconfiguration power then results:

$$\begin{aligned} P_{reconfig-row} &= P_{row} + (25mA) \times VCCAUX \\ P_{reconfig-col} &= P_{col} + (25mA) \times VCCAUX \end{aligned} \tag{4.3}$$

Note that $P_{reconfig-row}$ is the power during reconfiguration of the row filter into a column filter. $P_{reconfig-col}$ is defined in a analogous fashion.

With the processing times of the row and column filter, and the reconfiguration time, the energy per frame results:

$$\begin{aligned} epf_{FPGA} = P_{row} \times t_{rows} + P_{col} \times t_{cols} + \\ \left(P_{reconfig-row} + P_{reconfig-col}\right) \times t_{reconfig} \end{aligned} \tag{4.4}$$

In the case of the GPU implementation, similarly to [60], the current is measured with the clamp sensor ESI 687 on the power connectors. Both the external power of the GPU

and the power provided to the PCIe bus (20 W max.) are considered. Note that we measure the power consumption of the whole board that includes the GPU, memory, and other components. The average power during the tasks is measured, thus the energy per frame results:

$$epf_{GPU} = P_{average-clamp} \times \left(t_{alloc+transf(h->d)} + t_{process} + t_{transf(d->h)}\right) \quad (4.5)$$

Since the transferring and allocation times can be considered as an offset any GPU implementation has to deal with, we might also be interested in measuring the energy per frame spent only during the processing stage:

$$epf_{GPU} = P_{average-clamp} \times t_{process} \quad (4.6)$$

For accuracy measurements, we define accuracy as the relative error between the FPGA or GPU processed frame and the results using double precision with 48 coefficients. Consequently, we measure accuracy using the PSNR between the FPGA or GPU outputs and the double precision implementation (48 coefficients). Here, note that GPU implementation is also using double precision but with variable number of coefficients. On the other hand, for the FPGA, the error is due to truncation in the number of coefficients and the use of fixed-point arithmetic (16 bits).

4.5 Results

4.5.1 FPGA resource usage and reconfiguration time

The PRR must accommodate the largest filter (column filter with N = 32). Thus, the PRR occupies a tightly packed area of 24×90=2160 Virtex-4 slices with a bitstream size of 183754 bytes. It takes about 25% of the FPGA fabric.

Table 4.1. Embedded FIR Filtering system resource utilization (Virtex-4 XCVFX20-11FF672)

Module	Slice	(%)	FF	(%)	LUT	%
PRR (col filter)	2125	25%	3680	21%	3812	22%
Static Region	4973	58%	5226	31%	5998	35%
Overall	7098	83%	8906	52%	9810	57%

Table 4.1 shows the hardware resource usage of the embedded FIR filtering system of Figure 4.1. It reveals the actual resource usage of the PRR and the static region. Note that the largest column filter (N = 32) occupies 2125 Slices (98% of the PRR Slices).

A reconfiguration speed of 3.28 MB/s is obtained with the Xilinx® ICAP core, resulting in 56.02 ms of reconfiguration time for the given bitstream size.

4.5.2 Running times

In the FPGA case, t_{rows} and t_{cols} are in line with the FSL transfer speed of 226 Mbps reported in [9]. For example, for N = 32, t_{rows} = 10971, 3620, and 905 us for the VGA, CIF, and QCIF frame sizes respectively. The number of coefficients plays a negligible role in the processing time because the FIR filter is a fully pipelined system in which the number of coefficients only increments the register levels, which in turn increases the initial latency of the pipeline (that fades out for an input length larger than the number of coefficients). This effect is usually masked by the bus speed with bus cycles larger than the register levels of the pipeline. System performance is limited by the time spent in transposing the image (about 4152 us, 1453 us, and 379 us for the VGA, CIF, and QCIF frame sizes respectively) and the reconfiguration time (about 56.02 ms).

The reconfiguration time of 56.02 ms achieved with the Xilinx® ICAP controller significantly limits real-time system performance. With the use of the custom-made ICAP controller presented in [10], the reconfiguration time would be 0.622 ms. For a good comparison with the GPU, this reconfiguration time is used instead. Note that the custom

ICAP core has different power requirements than the Xilinx® ICAP core. In practice, we expect the power difference to be negligible since the custom ICAP core is a small (and low-power) circuit.

In the case of the GPU, we found that most of the time is consumed by the allocation of memory and data transfers from/to host to/from device. Table 4.2 shows these times.

Note that $t_{alloc+transf(\,h->d\,)}$ and $t_{transf(\,d->h\,)}$ are about the same for a given frame size. Also, the processing times do vary according to the number of coefficients, unlike in the case of the FPGAs.

Table 4.2. GPU running times (ms). N: number of coefficients

	N	$t_{process}$	$t_{alloc+transf(\,h->d\,)}$	$t_{transf(\,d->h\,)}$
640x480	8	0.4099	2.0	1.9
	12	0.4661	1.9	1.8
	16	0.5096	2.0	1.6
	20	0.5801	1.9	1.8
	24	0.6481	1.9	1.9
	32	0.7777	1.9	1.8
352x288	8	0.2536	1.14	0.86
	12	0.3031	1.12	0.94
	16	0.3486	1.10	0.90
	20	0.3527	1.40	0.75
	24	0.3975	1.73	0.70
	32	0.4610	1.40	0.60
176x144	8	0.1998	0.60	0.30
	12	0.2105	0.75	0.35
	16	0.2371	0.60	0.30
	20	0.2417	0.70	0.30
	24	0.2729	0.80	0.30
	32	0.2853	0.80	0.30

4.5.3 Power measurements

In the case of the FPGA, the power consumption is not dependent upon the frame size. Thus, it makes sense to report the result in terms of energy consumption per frame. Table 4.3 shows that the embedded system's power fluctuations due to the number of coefficients are negligible since only the filter IP core is modified. It is then more

meaningful to consider the power of the FIR Filter core which does vary according to N (number of coefficients).

Device static power does depend exclusively on the device size and operating temperature, called 'device static power' [27]. It is consumed by the device when it is powered up and without programming the user logic. For the XCVFX20 device, it amounts to 166 mW (all 3 voltage rails), at 25 ºC. If the power results are to be meaningful across different devices, this quantity must be considered as an offset that will vary across devices.

In the case of the GPU, we found that on average, it consumes 96.8, 92.5, and 88 Watts for VGA, CIF, and QCIF frame sizes respectively. Variations for different number of coefficients are negligible (around 0.1 W) since the algorithm uses the maximum number of cores regardless of the number of coefficients of the filter. The power fluctuations for different frame sizes are due to the fact that for smaller frame sizes, the GPU is moving data over a longer period of time than when it is processing.

Table 4.3. Embedded system Power consumption (Watts) on the XCVFX20-11FF672 Virtex-4 FPGA

	P_{rows}	P_{cols}	$P_{reconfig-row}$	$P_{reconfig-col}$
Mean	1.2410	1.2472	1.3035	1.3097
Std	0.0059	0.0140	0.0059	0.0140

4.5.4 Energy, Performance, and accuracy results

For comparing energy consumption, we only consider the energy spent by the filtering process. Thus, for the FPGA, we consider the energy consumed by the FIR filter and the ICAP cores. For the GPUs, we will also consider the energy spent during actual video processing (Equation. 4.6).

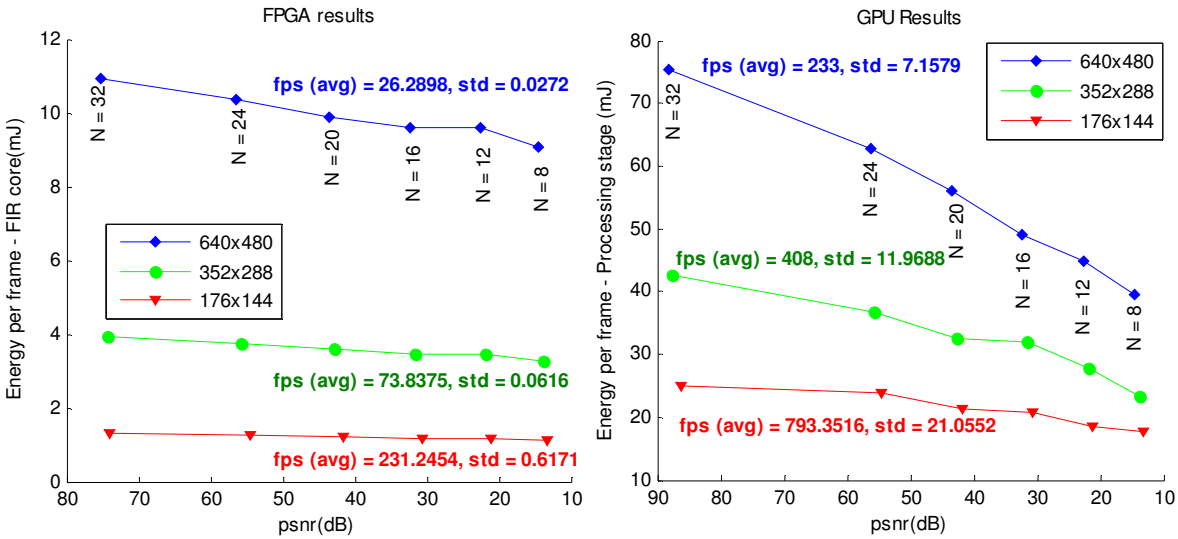

Figure 4.5. Performance, energy, and accuracy results for both FPGA and GPU. N: number of coefficients

Figure 4.5 shows the energy per frame, performance (achieved frames per seconds) and accuracy results. Note that in the case of performance, we report the mean fps with its standard deviation for a given frame size. We observe an energy dependence on the number of coefficients in the FPGA case, although it is more pronounced in the GPU case. In addition, the performance dependence on the number of coefficients is negligible in the FPGA case, but noticeable in the GPU case.

In terms of PSNR (dB), the GPU gives better results due to its use of double precision. However, there is no significant difference at the output except for N = 32. In this case, we have very high PSNR values that exceed 80dB.

In terms of performance, the GPU always prevails due to the massive amount of parallelization achieved in the algorithm coupled with the high operating frequencies. The speed up (GPU over FPGA) is about 9X, 5X, and 3.3X for VGA, CIF, and QCIF frame sizes respectively. For smaller frame sizes, the time consumed in allocations and transfers is closer to the processing times.

In terms of energy per frame, the FPGA implementation is much better than the GPU. The GPU implementation consumes 6, 9, and 19 times more energy than the FPGA's for VGA, CIF, and QCIF frame sizes respectively.

The results suggest that the FPGA implementation provides a low-energy solution at near real-time performance. Here, we refer to frame rates that are over 30 fps as achieving real-time performance. On the other hand, when energy consumption is not an issue, the GPU implementation is superior, delivering much higher performance at slightly better accuracy.

4.6 Conclusions

This work successfully compares energy, performance (in frames per second), and accuracy for both FPGA and GPU implementations. Moreover, these 2 implementations allow the user to modify the 2D FIR Filter at run-time. The results indicate that separable 2D FIR filtering implementations can deliver excellent accuracy for both the FPGAs and the GPUs. However, based on energy consumption, FPGAs are preferred for low-energy applications. On the other hand, GPUs should be considered for high-performance, high-power (high-energy) applications.

Chapter 5

Dynamic Energy, Performance, and Accuracy Optimization and Management for Separable 2-D FIR Filtering for Digital Video

Abstract

In this work, we develop a dynamically reconfigurable 2D FIR filtering system that can meet real-time constraints in Energy, Performance, and Accuracy (EPA). To meet the EPA constraints, we generate a set of Pareto-optimal realizations, described by their EPA values and associated 2D FIR hardware description bitstreams. Dynamic management is achieved by selecting Pareto-optimal realizations that meet the time-varying constraints. For efficient implementation, the Pareto-optimal realizations are stored in DDR-SDRAM.

We validate the approach using three different 2D Gaussian filters. Filter realizations are evaluated in terms of the required energy per frame, accuracy of the resulting image, and performance in frames per second. We demonstrate dynamic EPA management using a Difference of Gaussians (DoG) applied to a standar video sequence.

Index Terms—Dynamic Partial Reconfiguration, Field-programmable gate-array (FPGA), Distributed Arithmetic, 2D separable FIR filtering.

5.1 Introduction

The recent introduction of Dynamic Partial Reconfiguration (DPR) provides a framework for managing hardware resources in real-time. In particular, the use of DPR enables the development of dynamically reconfigurable systems that can meet constraints in energy, performance, and accuracy (EPA).

In this work, we are interested in the development of a dynamically reconfigurable 2-D FIR filtering system for digital video processing applications. Here, the focus on 2-D FIR filtering comes from the large number of possible applications. The list of applications includes image and video denoising, linear image and video enhancement, image restoration, edge detection, face recognition, etc [65], [1]. Depending on the application, we can have very different EPA requirements. Furthermore, we can have real-time constraints that are imposed during the execution of a particular application. In what follows, we present an example.

Suppose that we have the use of a 2-D FIR filtering system in a real-time video analysis application. First, suppose that during real-time video acquisition, we determine that there is nothing interesting in the scene. In such a case, we may want to save energy until something interesting occurs. In this case, we may be willing to sacrifice accuracy and performance to allow us to operate longer. In this case, we will want to dynamically reconfigure the 2-D FIR filter to minimize energy consumption. Now, suppose that the real-time video scene changes to something much more interesting. In this case, we want to improve accuracy and performance while willing to sacrifice energy.

The example motivates the development of a management system that can be used to dynamically reconfigure hardware resources to meet real-time constraints in energy, performance, and accuracy. Here, we measure performance in terms of frames per second

(fps) and estimate accuracy in terms of achieved PSNR on a test image. Dynamic management is based on using dynamic partial reconfiguration to implement pre-computed realizations. We are only interested in implementing realizations that are Pareto-optimal in the EPA space (see Fig. 1, [8]). As shown in Fig. 1(b), dynamic EPA management is achieved by swapping among Pareto-optimal realizations that meet or exceed real-time constraints. More specifically, Pareto-optimal realizations will simultaneously minimize energy and maximize accuracy and performance. When multiple realizations meet the constraints, we will pick the one that also minimizes energy consumption.

Energy (or power) and accuracy are intrinsically linked to performance. Dynamic Energy-Performance-Accuracy (EPA) management has been hinted in some earlier work (e.g. [3], [4]). Here, it was suggested that one of these three system properties could be potentially modified via DPR. As the design flow for DPR matured, more work on this regard has appeared (e.g. [5], [66], [7]). Power, performance and accuracy were variables commonly changed using DPR by trading off one by the other. A dynamic arithmetic example for controlling precision in real-time was presented in [66].

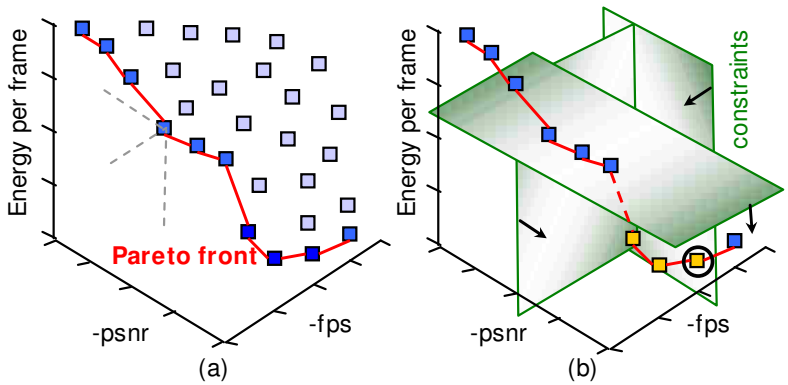

Figure 5.1. Multi-objective optimization of the EPA space.
(a) 3-D Pareto Front. (b) 3 constraints applied to the 3-D Pareto Front. Minimum energy point is circled

For efficient hardware realizations, we will focus on 2D FIR separable filtering. Here, we note that separable filters allow for efficient implementations by means of two 1D FIR filters. Furthermore, note that non-separable 2D filters can be expressed as a sum of separable kernels through the use of Singular Value Decomposition [67]. This technique can be either exact (no error introduced) or inexact (certain approximation error is allowed). Thus, without loss of generality, we focus on separable 2D filters. This separability property allows us to consider a DPR approach that keeps only one filter (row or column) at a time and changes to the other 1D filter when needed.

We presented some related earlier work in [17], [68]. In [17], we presented an efficient 1D FIR Filtering system that combined the Distributed Arithmetic (DA) technique with DPR. In the conference paper in [68], we presented the 2-D FIR Filter. The main contributions of the current work include: i) an optimization framework for dynamic EPA management of 2D FIR filters; ii) a platform to generate the EPA space of 2D FIR filters, iii) an analysis of the behavior of the EPA space of 2D FIR filters when the parameters and the filter types vary, and iv) a demonstration of dynamic EPA management on a standard video sequence.

The proposed system relies on an efficient DPR controller. This is required since the 2D FIR filter is implemented through the use of DPR of 1D FIR filters. For research related to the development of efficient DPR controller, we refer to [10] [7] [11] [12].

The rest of this chapter is organized as follows: Section 5.2 presents background and related work. Section 5.3 provides details on the dynamic video filter implementation. Section 5.4 presents the optimization framework for 2D filters. Section 5.5 presents the experimental setup. Section 5.6 presents the results; and Section 5.7 lists the conclusions.

5.2. Background and related work

Static FPGA image processing examples include MPEG buffer analyzer [69], JPEG decoders [70], JPEG2000 encoders [71], face detection systems [72],[73], reconfigurable embedded systems for real-time vision [74] and ultrasonic imaging [75], 2D Discrete wavelet transform using the residue number system [76], and binary morphology architectures [77].

More recently, we also have DPR implementations of image processing systems. In [7], the authors presented a design that dynamically reconfigures among Discrete Cosine Transform (DCT) modules of different sizes (e.g. 8x8, 5x5, 4x4). The different DCT configurations are studied in terms of power, throughput, and image quality. A dynamic systolic array accelerator for Kalman and Wavelet filters was presented in [13]. In [14], the authors present a fingerprint image processing algorithm whose stages (e.g., segmentation, normalization, smoothing, etc) are time-multiplexed via DPR. A system that can reconfigure among single-pixel operations is presented in [9]. The 3D Haar Wavelet Transform (HWT) was implemented by dynamically reconfiguring a 1D HWT core thrice in [15]. A JPEG2000 decoder where the blocks are dynamically swapped is shown in [16].

For 2D FIR Filtering, the authors in [78] presented a multiply-and-add implementation of separable Gaussian filters. Similarly, 2D separable filter implementations using multiply-and-add approaches are presented in [59]. In [79], the authors present a novel design methodology that decomposes a 2D filter into 2D separable and non-separable

filters, and efficiently allocates the heterogeneous resources (embedded multipliers, LUTs, FFs) on an FPGA.

There is also some related work on the implementation of 2D Filters based on DPR. In related work in [58], a 2D filterbank implementation based on the run-time coefficient-only reconfiguration of a single 1D FIR Filter was presented. In [80], the authors presented a system that switches between a median filter (nonlinear) and an averaging filter via DPR on a custom-built FPGA device (180nm CMOS technology). Similarly, the authors in [81] used DPR management to switch between mean and median filter implementations on a Xilinx® FPGA. In [82], the system dynamically reconfigures a 3x3 2D FIR filter by changing the coefficients. In earlier work in [68], a 2D FIR filter is implemented by dynamically reconfiguring between the row and column filter.

There has also been some earlier research related to Dynamic EPA management. Early work dealt with one or two objectives at a time. For instance, in [25], the authors analyze the precision requirements of a subset of recursive algorithms. In [3], the authors propose the use of reconfiguration to take advantage of perceptual tolerance and the non-uniformity of video content in order to dynamically manage power consumption, over which accuracy and performance depend on. Another example of power and accuracy trade-off is [4], where the impact of numerical precision on power consumption is studied for audio processing applications. In [83], the authors presented a static iterative hardware implementation for particle filters that allowed run-time modification of the number of particles (for trading off accuracy and performance), and the degree of parallelism of some components (for trading off power and performance). This was accomplished by tuning buffer controller parameters and interconnection switches. In

[66], an application in dynamic arithmetic is presented where arithmetic cores are measured in terms of their power, performance, and precision requirements. In [7], the authors presented a configuration manager that can dynamically adapt DCTs of different sizes based on power, performance and accuracy considerations. In earlier work in [68], we presented a comparison of the energy-accuracy space of a 2D FIR Filter for both FPGA with DPR and GPU implementations.

In the current work, we evaluate different realizations based on their Energy-Performance-Accuracy characteristics. Here, each realization comes with its own EPA values. However, we are only interested in realizations that are Pareto-optimal [8]. In other words, we select EPA realizations that cannot be improved upon without sacrificing in at least one of the EPA characteristics (see Fig. 5.1(a)). As discussed earlier, the framework allows us to meet real-time constraints by simply selecting the realization with minimum energy (see Fig. 5.1(b)).

As mentioned earlier, reconfiguration time overhead is a limiting factor in the use of DPR. Techniques to reduce the DPR overhead include improving the access speed to the configuration memory [10], reducing the size of the reconfigurable area [84], and reducing the reconfiguration rate [85]. In [7], the authors improved the configuration memory access speed by compressing the partial bitstreams while they are moved through the slow parts of the system. In [12], the reconfiguration overhead is less of a concern since the approach allows the processor to multi-task will full access to the peripheral bus.

This work seeks to extend prior research in the area of 2D filtering by using the 2D separable FIR filter implementation with DPR presented in [68], and study its EPA space.

Furthermore, we propose a multi-objective framework to derive a set of optimal filter realizations over which we can dynamically reconfigure to meet EPA constraints.

5.3 Video filtering using Dynamic Partial Reconfiguration

This section presents the architectural framework that allows the generation of different 2D FIR filter realizations. The approach is to consider realizations based on the number of coefficients, the coefficient bit-width, and the output bit-width.

5.3.1 Distributed Arithmetic Stand-Alone 1D FIR Filter

We present the 1-D FIR filter core in Fig. 5.2. The core can be used to implement the row or the column filter.

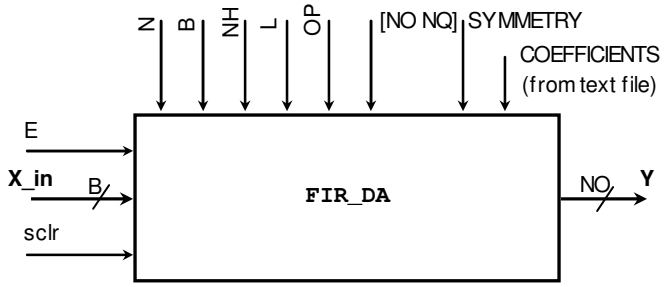

Figure 5.2. FIR Filter Intellectual Property (IP) core. Here, *N* denotes the number of coefficients, *NH* the coefficients' bitwidth, *B* the input bitwidth, *L* the LUT input size (FPGA device dependent), *[NO NQ]* the output fixed-point format, and *OP* the output truncation scheme.

We next consider the largest possible format based on the input and coefficient formats. First, we let *NO* represent the number of output bits with *NQ* fractional bits. The output format is then expressed as *[NO NQ]*. Both symmetric and non-symmetric filters are supported (see Fig. 5.3). Second, we let the fixed-point input format be *[B B-1]* and the coefficients' format be *[NH NH-1]*, so that we normalize the inputs/coefficients to *[-1,1)*. The required largest output format is then given by:

$$\left[NH + B - 2 + \left\lceil \log_2 (N+1) \right\rceil + 1 \quad NH + B - 2\right] \tag{5.1}$$

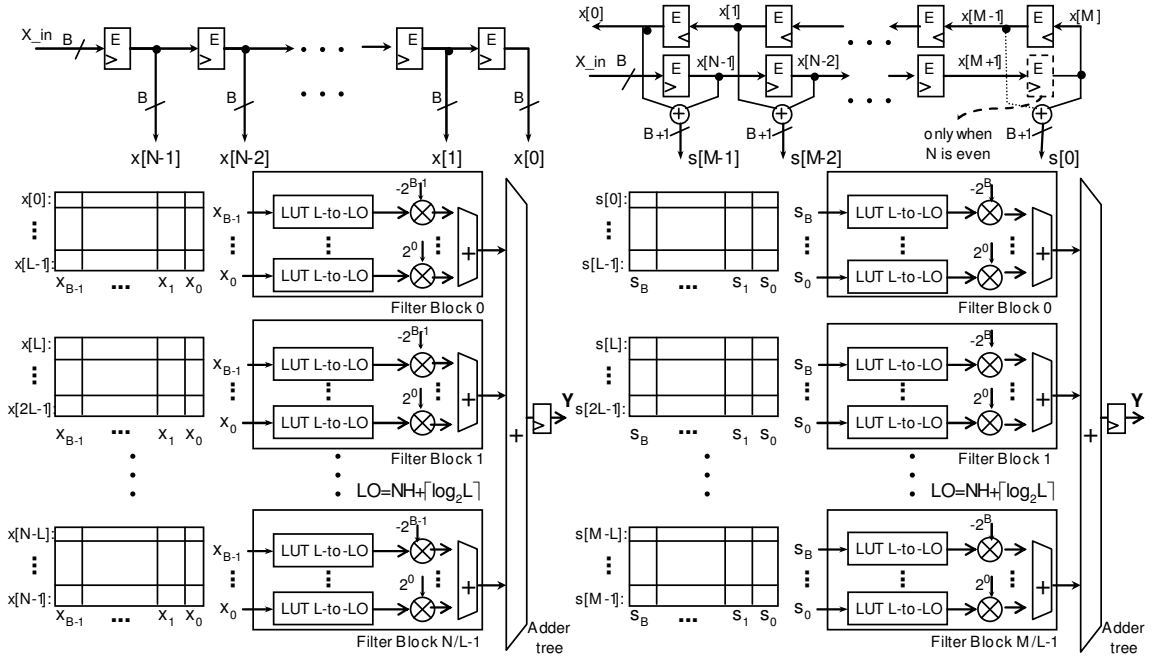

Figure 5.3. High Performance FIR filter implementation. Nonsymmetric filter (left), symmetric filter (right). *B* and *NH* are independent parameters here. Refer to [17] for details on the Filter Block implementation. The L-input, LO-outputs LUT (LUT L-to-LO) is a 2^L-word LUT

Clearly, we do not need to have an output format that exceeds the largest possible. On the other hand, we are interested in investigating the EPA space for formats that do not exceed (5.1). When considering smaller output formats, overflow is avoided by using LSB truncation and saturation (controller by parameter *OP*, see Fig. 5.2).

We refer to [17] for details on how to implement each 1-D FIR filter core using distributed arithmetic. In the extended core considered here, we have expanded the core of [17] to also allow the number of input bits (*B*) to be independent of the coefficients bitwidth (*NH*). Our new 1-D FIR core was summarized in [68]. Here, we want to provide more implementation details for the new core of [68].

The FIR filter latency (register levels between input and output, or I/O delay, in cycles) results in $REG\_LEVELS = \lceil \log_2(sizeI) \rceil + \lceil \log_2 M/L \rceil + 2$ cycles, where (i)

$M = \lceil N/2 \rceil$ and $sizeI = B + 1$ for symmetric filters, and (ii) $M = N$ and $sizeI = B$ for non-symmetric filters. Here, note that *sizeI* is different from [17].

5.3.2 Dynamic 1D FIR Filter Core Architecture

The constant-coefficient filter is turned into an efficient and flexible FIR filter via DPR, as described in [17]. Two dynamic realizations were presented: coefficient-only reconfiguration and Full-filter reconfiguration. We focus on the full-filter reconfiguration case, where the entire filter is included in the Partial Reconfiguration Region (PRR). This allows us to generate many realizations for exploring the EPA space by allowing us to modify all the parameters independently. The PRR has *B+2* inputs and *B* outputs.

Fig. 5.4(a) shows an embedded system that allows for Full-filter reconfiguration. The FIR Filter processor IP and the processor communicate via the 32-bit Fast Simplex Link (FSL) bus. At power-up, the partial bitstreams and input data are stored in Compact Flash (CF). During run-time, we store the input data, the DPR bitstreams of the filter realizations, and the output in the DDR-SDRAM. The Ethernet core allows us to get new partial bitstreams or new input streams from a PC and to send processed streams to the PC. It also provides an interface for throughput measurements and system status.

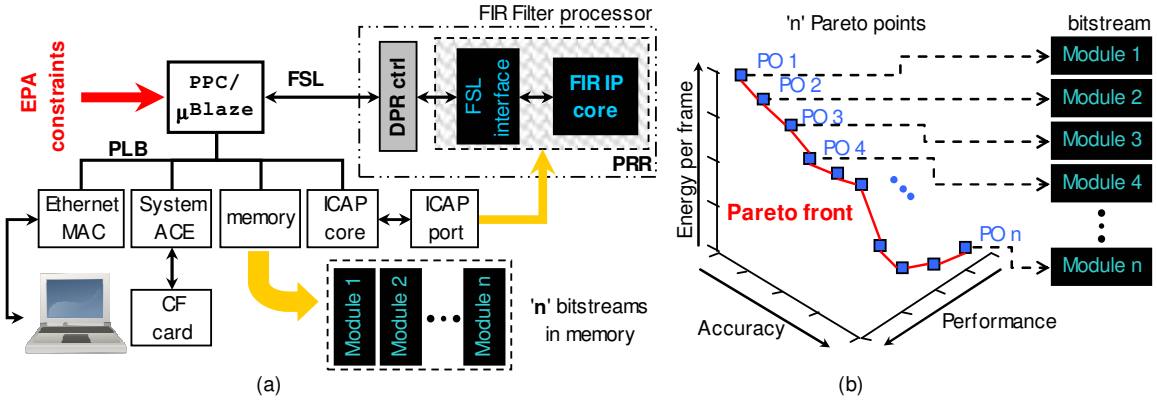

Figure 5.4. Embedded system over which we can apply Dynamic EPA management. (a) Embedded system that supports DPR. The memory holds the 'n' unique bitstreams that are needed for the Pareto front. (b) An example of a Pareto front with 'n' points.

The static region is defined by the logic outside the PRR. The PRR I/Os are registered as the reconfiguration guidelines advise [86]. The partial bitstreams are read from a CF card and stored in memory. To perform DPR, the bitstreams in memory are streamed to the ICAP port [66].

During DPR, we want to allow changes to the I/O bit-width. This is accomplished by including the FSL interface in the PRR. This also allows us to dynamically control the input stream length (*NX*).

Each FIR convolution generates *NX+N-1* values where *N* is the number of coefficients and *NX* denotes the number of input values. The FSL interface offers four output choices for storing the convolution results: (i) basic: first *NX* output samples, (ii) centered: *NX* samples in the range $\lfloor N/2 \rfloor + 1 : NX + \lfloor N/2 \rfloor$, (iii) full: All the *NX+N-1* samples, and (iv) streaming: $NX = \infty$, infinite output samples.

For proper DPR operation, we include a DPR control block that addresses two issues that arise due to the fact that the FSL interface is inside the PRR. First, during DPR, the PRR outputs are disabled so as to avoid erratic FSL control behavior. Second, to avoid erroneous results, the DPR control block resets the PRR flip-flops after each partial reconfiguration. This is needed since the flip-flops are not reset automatically as it is the case for full reconfiguration [86].

Fig. 5.5 shows the dynamic FIR Filter core along with the DPR control block. The FSL bus uses two FIFOs (*FIFOw* and *FIFOr*) to communicate with the FIR filter core.

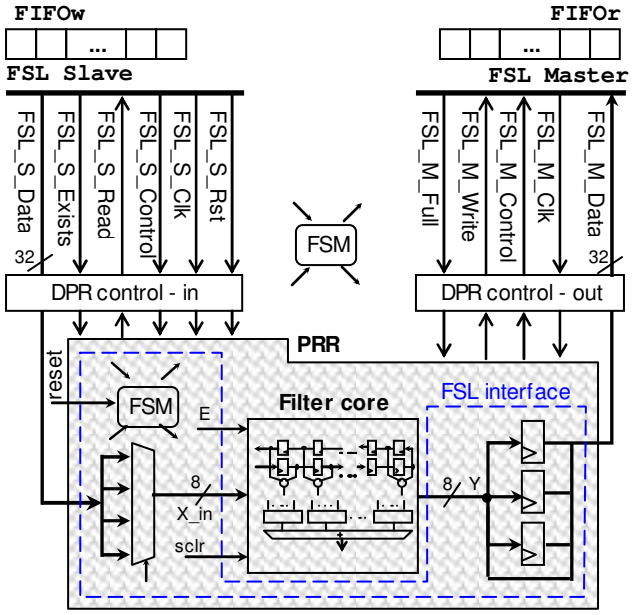

Figure 5.5. PRR that includes the filter core for full-filter reconfiguration and the FSL interface, needed for image filtering . B = 8.

5.3.3 2D separable FIR filtering

In the context of the embedded system of Fig. 5.4(a), a 2D separable FIR filter is represented by 2 bistreams (one for the row and one for the column filter). A 2D filter is implemented by cyclically swapping the row filter with the column filter via DPR [68].

The implementation of the 2D separable FIR filter includes the following considerations:

1) The 2D filtering process usually requires the output image to be of equal size as the input image. As a result, the dynamic FIR Filter IP of Section 5.3.2 needs to be in the centered output mode.

2) The column filter is not necessarily the same as the row filter with coefficients modified. It usually requires other parameters to be modified (e.g. number of coefficients, I/O format). This requires a full filter reconfiguration.

3) The length of the input signal in the row filtering case is different than in the column case (unless the image is square). This means that we have to change *NX* to match the input size.

4) Two reconfigurations are performed per frame. The row filter processes and stores the result in a sequential row-by-row fashion, but the column filter does so in a sequential column-by-column fashion. For the purposes of this work, the row-filtered output images are transposed prior to column filtering.

5.3.4 General Filterbank implementations

The extension of the current framework for implementing general filterbanks is straightforward. To do this, we only need to implement each filter using a 2-D separable approximation (e.g. [67]). Here, a non-separable filter is approximated by a sum of separable filters.

For implementing the full filterbank, we will have to sequentially apply DPR for each filter's row and column bitstream(s). This way we switch among 2D FIR filters. Note that this does not incur in any performance penalty since we are always performing DPR twice per frame. In other words, execution time grows linearly with the number of 2-D separable filters. We refer to [58] for an example based on changing filter coefficients (not full-filter reconfiguration).

5.3.5 Resource considerations

The proposed dynamic DPR approach only requires resources for a single 1-D FIR filter at a time. Thus, this approach can yield significant savings over static implementations of the full 2-D filters. Naturally, this assumes that the DPR controller does not require

significant resources. It is expected that the DPR controller resource consumption will be negligible when compared to the resources needed for larger 2-D filters.

5.4 Optimization framework for Video Filters

This section describes a framework for extracting an optimal set of 2D filter realizations from the EPA space. Then, we provide a framework for selecting optimal realizations that meet dynamic EPA constraints. We detail: i) how we generate a collection of FIR filter possibilities, ii) how we measure energy, performance, and accuracy, iii) how we select the Pareto-optimal filters, and iv) how we perform dynamic EPA management that meets the EPA constraints.

5.4.1 Generation of the set of 2D filters

We want to devise a procedure that allows us to meet energy, performance, and accuracy constraints by considering different filter implementations. We first create a large set of filters from which the optimal set would be extracted. Note that this space is generated offline.

This space of possible 2D FIR filter realizations is generated by varying the parameters discussed in Section 5.3. Here, for each 2D filter, we assume *N, L, NH* to stay the same for both row and column filters, while *B* and *[NO NQ]* can be different. By varying the input stream length *NX*, which is usually different for both the row and column filters, we can explore different frame sizes. The collection of EPA measurements for each filter realization forms the "EPA space" of possibilities.

5.4.2 Performance measurements

The performance of the embedded system depends on many factors (cache size, processor instruction execution, bus usage, etc) that can easily change. For the purposed of this work, the embedded system is just a generic test-bed. To avoid dependence on the embedded system characteristics, we are interested in direct measurements of the dynamic FIR core shown in Fig. 5.5. This is often referred to as the intellectual property (IP) angle. Here, we also assume a continuous streaming of the input signal. In what follows, we explain how we measure *filter performance* based on *filter processing time* and *reconfiguration time*.

1) Filter processing time: The 2-D filter operates on a row-by-row or column-by-column basis. We use the term stream to refer to a single row or column. After each stream is processed, the register chain in the FIR filter is cleared, ready for a new stream. Let the lengths of the input streams be defined as *NXr* and *NXc* for the row filter and column filter respectively. Here, *NXc* is the number of rows, and *NXr* the number of columns of the input video frame. For the time taken, we have:

$$\begin{aligned} t_{rows}(\text{sec}) &= (NXr + \lfloor Nr/2 \rfloor + REG\_LEVELSr) \times NXc \times T_{cycle} \\ t_{cols}(\text{sec}) &= (NXc + \lfloor Nc/2 \rfloor + REG\_LEVELSc) \times NXr \times T_{cycle} \end{aligned} \tag{5.2}$$

where *Nr*, *Nc* represent the number of row and column filter coefficients respectively. *REG_LEVELSr, REG_LEVELSc* denote an initial latency (see Section 5.3.1) and T_{cycle} is the clock period. Here, note that $\lfloor Nr/2 \rfloor$, $\lfloor Nc/2 \rfloor$ cycles are needed to provide centered row/columns convolution outputs.

2) Reconfiguration Time: Based on the PRR bitstream size and the reconfiguration speed, the reconfiguration time is given by:

$$t_{reconfig}(\text{sec}) = \frac{PRR\ size(in\ bytes)}{\text{Rec. } speed(bytes\ per\ \text{sec})} \tag{5.3}$$

The maximum reconfiguration speed is achieved if there is a direct link between the ICAP port and the memory that holds the partial bitstreams (400 MB/s for Virtex-4). If the DPR bitstreams are loaded in the BRAM (local memory inside the FPGA), it is possible to get the maximum reconfiguration speed [84]. However, the size and quantity of partial bitstreams is limited by the available BRAMs. While BRAMs are rather limited in Virtex-4 devices, there are significantly more BRAM resources in the (newer) Virtex-6 devices [87]. This provides the opportunity to build an ICAP controller that is directly attached to BRAM. In what follows, we will assume the maximum reconfiguration speed as reported in [84], [11].

3) Filter Performance: Based on the processing and reconfiguration times, we can define the filter performance. A frame of pixels goes to both the row and column filter and thru two partial reconfigurations. Thus, the performance (in frames per second) is given by:

$$fps = \frac{1}{t_{rows} + t_{cols} + 2 \times t_{reconfig}} \tag{5.4}$$

Please note that Eq. (5.4) only measures the performance of the FIR filter architecture of Fig. 5.3. It does not account for the time needed to transpose the row-filtered image. Here, we note that the transposition time is a function of the image size and the embedded platform over which the system is tested. Thus, it does not depend on the DPR filter architecture. For an embedded system example that includes transposition time, we refer to [68]. Also, for completeness, in the results section, we will report the transposition time.

5.4.3 Energy measurements

In this sub-section, we detail the IP core energy consumption measurement. The IP core consists of the PRR (FIR filter and the FSL interface) and the DPR control block. In the context of our system, energy per frame provides more information than power since the system goes through several stages that draw different amounts of power.

We will report the energy consumption in terms of energy spent for processing a single frame. This is estimated as the sum of the products of the power and processing times of the row filter, the column filter, and the reconfiguration process.

1) Power measurements: Power inside the FPGA is drawn by the following power supply rails: (i) internal supply voltage *VCCINT* with current *ICCINT*, and (ii) auxiliary supply voltage *VCCAUX* with current *ICCAUX*. Here, we will not consider the output supply power since it is only associated with the power drawn by the external pins.

Power at each supply rail is divided into static and dynamic power. The *static power* is drawn by the device when it is powered up, configured with user logic, and with no switching activity. It is divided into: i) *device static power*: drawn by the device when it is powered up and not programmed, and ii) *design static power*: drawn by the user logic when the device is programmed and with no switching activity. The *dynamic power* is the fluctuating power as the design runs; it is generated by the switching user logic and routing [27].

For comparing among different cases, we will only consider the sum of the dynamic and design static power while ignoring the device static power. The device static power depends on the environment, the device size, and the device family. FPGA datasheets provide the device static current as constant values (at 25º C) for each supply rail. For the purposed of this work, for the XC4VFX20 Virtex-4 FPGA, the voltage supply values can

be fixed while the device static current are constant and given by *ICCINTQ=71mA* and *ICCAUXQ=35mA*. The FIR filter core power for row or column filtering is then given by:

$$P_{row/col} = VCCINT \times ICCINTp + VCCAUX \times ICCAUXp \tag{5.5}$$

where the currents are given by:

$$ICCINTp = ICCINT - ICCINTQ$$
$$ICCAUXp = ICCAUX - ICCAUXQ$$

Power measurement amounts to current measurement, assuming minimum fluctuation of the voltage values. Direct power measurement, (e.g., [66]) requires custom-built boards that allows for current measurement on the supply rails. Instead, power consumption can be accurately estimated using software tools that are widely applicable to all devices. To estimate the current measurements (at 25 ºC), we are using the Xilinx Power Analyzer (XPA) that provides an accurate estimate based on simulated switching activity of the place-and-routed circuit and exact utilization statistics [27].

We next consider the power consumption during dynamic partial reconfiguration. Unfortunately, no software tool exists that can provide an estimate of this power consumption. In [66], by direct current measurements, it was determined that the only supply current that increased during DPR was *ICCAUX* (Virtex-II Pro and Virtex-4). The DPR power (power drawn by the user logic and the increase due to DPR) is then estimated by:

$$\begin{aligned} P_{reconfig-row} &= P_{row} + VCCAUX \times ICCAUX(increase) \\ P_{reconfig-col} &= P_{col} + VCCAUX \times ICCAUX(increase) \end{aligned} \tag{5.6}$$

In [66], the authors found that *ICCAUX* increased by 200mA and 25mA for the Virtex-II Pro (XC2VP30) and Virtex-4 (XC4VFX12) respectively. Assuming the

dynamic behavior of ICCAUX only depends on the user logic, we expect that these current values will remain the same within the same device family.

2) Energy per frame: The total energy per frame is the sum of the energy consumed by the following processes: i) row filtering, ii) turning a row filter into a column filter via DPR, iii) column filtering, and iv) turning a column filter into a row filter via DPR. Using the power and the processing times of each process, the energy per frame (in Joules) is given by:

$$\begin{aligned} Energy\ per\ frame = {} & P_{rows} \times t_{rows} + P_{cols} \times t_{cols} \\ & + \left(P_{reconfig-row} + P_{reconfig-col}\right) \times t_{reconfig} \end{aligned} \tag{5.7}$$

5.4.4 Accuracy measurements

We measure the accuracy of the 2-D impulse response and the filtered images using the peak signal-to-noise ratio (PSNR). This is given by:

$$PSNR = 10 * \log_{10}\left(\frac{MAX\ Value^2}{MSE}\right) \tag{5.8}$$

where the MSE is the mean squared error between the fixed-point filter output and the output of the filter implemented with double floating point precision.

5.4.5 Generation of optimal Filter realizations

Based on energy per frame, performance, and accuracy (EPA) measurements, we create the EPA space, from which we extract the optimal realizations. A 2D filter realization is defined to be optimal in the multi-objective (Pareto) sense if we cannot improve on its EPA measurements without decreasing on at least one of them.

The goal is to minimize the energy per frame consumption and to maximize performance and accuracy. For a given EPA space, the collection of all Pareto optimal

realizations forms a Pareto front (see Fig. 5.1(a)). The points are plotted against energy, and the negatives of performance and accuracy.

Fig. 5.1(a) shows the EPA space along with the Pareto front. Independent constraints appear as planes in 3-D. Optimal realizations are then selected among the Pareto optimal points that also satisfy the constraints (see golden points in Fig. 5.1(b)). Dynamic EPA constraints satisfaction only requires that we select Pareto-optimal points when the constraints change. The computation of the Pareto points is straightforward. Here, we are interested in understanding how the FIR core parameters generate Pareto-optimal realizations.

5.4.6 Dynamic EPA management based on DPR

In hardware, a 2D filter Pareto-optimal realization is represented by its two associated bitstreams (row and column filters), and the EPA measurements. The realizations and associated parameters are stored in memory. The dynamic EPA management framework is shown in Fig. 5.4. Fig. 5.4(a) shows an embedded system that can dynamically modify the 2D FIR filter realization. Fig. 5.4(b) illustrates how the system moves dynamically along the Pareto front via DPR.

EPA constraints can be met by selecting solutions along the Pareto front. For certain EPA constraints, we are left with the feasible set (see golden-colored points in Fig. 5.1(b)). Fig. 5.1(b) depicts a case in which a system sets a maximum value for the energy per frame, but requires minimum levels of performance and accuracy. The 2D FIR filter realizations represented by the golden points meet these specifications. The selected 2D FIR filter realization is chosen to be the one that also minimizes the energy consumption.

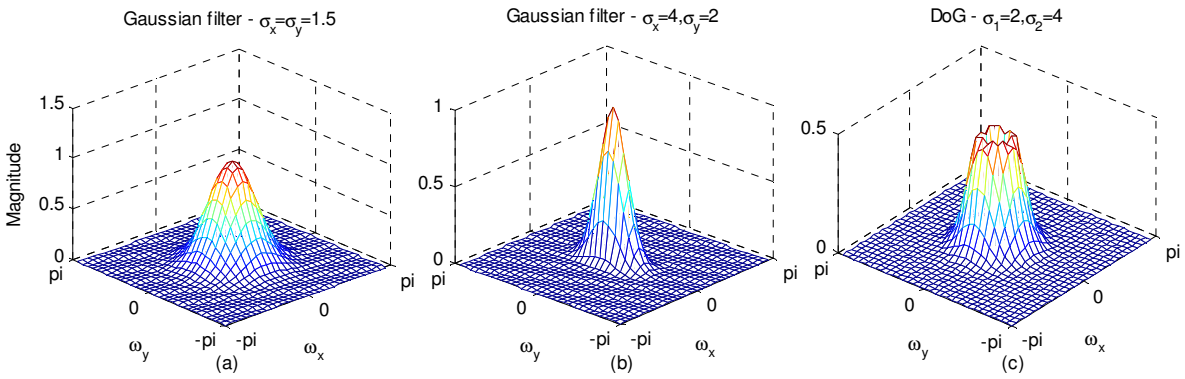

Figure 5.6. Frequency-magnitude responses for (a) Isotropic low-pass Gaussian filter ($\sigma_x=\sigma_y=1.5$) using 24 double-floating point coefficients in the row and column filters. (b) Anisotropic low-pass Gaussian filter ($\sigma_x=4$, $\sigma_y=2$) with 32 coefficients in each direction. (c) Band-pass, Difference of Gaussians (DoG) filter ($\sigma_1=2$, $\sigma_2=4$) with 48 coefficients in each direction.

5.5 Experimental Setup

5.5.1 Generation of the set of 2D separable filters

The FIR core platform is tested using Gaussian filters of different spreads (sigmas) and frequency characteristics. Gaussian filters are selected since they have several applications in image processing (e.g., image restoration, image analysis, and computer vision [65]).

To cover a variety of possibilities, we investigate the performance for: (i) isotropic, low-pass, Gaussian filter with $\sigma=1.5$, (ii) anisotropic, low-pass, Gaussian filter with $\sigma_x=4$, $\sigma_y=2$, and (iii) isotropic, band-pass filter based on a Difference of Gaussians (DoG), with $\sigma_1=2$, $\sigma_2=4$. Please note that all of these filters are 2-D separable and symmetric. Fig. 5.6 shows the magnitude-frequency response of the three filters using double-precision arithmetic.

5.5.2 FIR Filter core parameters

For the filters, we use symmetric implementations, with an arithmetic mode that uses truncation of the LSB, followed by saturation. In all cases, we consider 8-bit input images

(for row filter input bit-width). We also constrain the output to be in the same range as the input $[-1,+1)$ (*NQ=NO-1*), and keep the same number of output bits for the row and column filters. We define *OB* as the 2D filter output bitwidth.

For simplicity, we keep *N*, *L*, *NH* the same for both the row and column filters (see Fig. 5.2 for definitions). Here, for the anisotropic cases, note that accuracy could be improved if we considered separate numbers of coefficients for each dimension (as a function of sigma). For the isotropic cases, the optimal case is to keep the number of coefficients the same as we do here. We summarize the parameter values for *N*, *NH*, *OB*, *NXr*, and *NXc* as shown in Table 5.1.

Table 5.1. Parameters combinations (108) for the set of 2D Filters. The choice of *OB=8,16* is based on the fact that the FSL bus width is 32 bits. We fix the LUT input size (*L*) for a given *N*.

Frame Size (*NXc*×*NXr*)	640x480 (VGA)		352x288 (CIF)		176x144 (QCIF)	
Number of coefficients (*N*) (LUT input size (*L*))	8 (4)	12 (6)	16 (4)	20 (5)	24 (6)	32 (4)
Coefficients bit-width (*NH*)	10		12		16	
Output bit-width (*OB*)	8		16			

5.5.3 Platform testing scheme

The 2D FIR filtering system of Fig. 5.4(a) was implemented on the ML405 Xilinx Development Board that houses a XC4VFX20-11FF672 Virtex-4 FPGA. The selected processor (PowerPC) is clocked at 300 MHz, peripherals run at 100 MHz, and the 128MB DDR-SDRAM memory space is cached. The system was tested with the Xilinx® ICAP core.

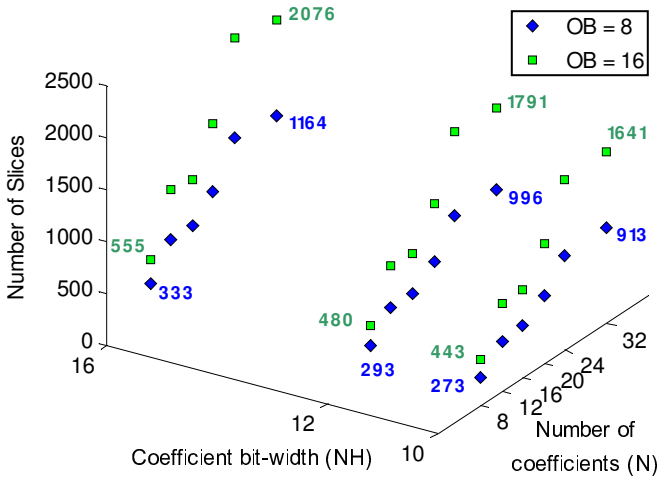

Figure 5.7. Hardware resource utilization for the column filters. For all filter types, the resources depend on the number of symmetric coefficients (N), the coefficient bitwidth(NH), and the output bits(OB). Hardware resources for the row filters are very close to the required resources for the column filters for OB=8 (see text for details).

5.6 Results and analysis

5.6.1 Hardware resource utilization

1) FIR IP Utilization: Fig. 5.7 presents the numbers of slices required by the column filter core (PRR and the DPR control block) as a function of the number of coefficients, coefficient bit-width, and output bits. Refer to Table 5.1 for parameter combinations. Here for *OB=8*, the row filter core resources are only slightly different (by 2-5 slices) than what is shown for the column filter case. Also note that the ISE synthesizer was directed to avoid optimizing the LUT values themselves, thus giving essentially the same resource consumption for the three filter types. Furthermore, since the input size is fixed at 8 bits, for the row filters (only), changing the output bits from 8 to 16 requires minimal additional resources (1-8 slices).

The range of required resources in Fig. 7 varies significantly. For example, the use of 8 10-bit coefficients with 8-bit output bits requires the minimum of 273 slices. The use of 32 16-bit coefficients with 16-bit output bits requires the maximum of 2076 slices. As we

shall see, this strong variation in resource consumption will enable an effective optimization of the EPA space.

2) Size of Reconfigurable Area: The PRR size is set to the largest possible filter realization (be it row or column). This largest realization is given by the column filter with *N=32, NH=16, OB=16*. The PRR occupies a tightly packed area of 24×94=2256 slices for a bitstream size of 183,754 bytes.

3) Embedded system resource utilization: Table 5.2 shows the hardware resource utilization of the embedded FIR Filtering system of Fig. 5.4(a), detailed in Section 5.4 under the parameter setup of Table 5.1. The PRR includes a 1D filter and the FSL interface. The largest realization occupies 2125 Slices (94% of the allocated space for the PRR). This is slightly higher than what Fig. 5.7 reports since the results are obtained by compiling the embedded system. For transposition of the row-filtered image, we have 4152µs, 1453µs, and 379µs for the VGA, CIF, and QCIF frame sizes respectively.

Table 5.3 shows the reconfiguration time for 3 scenarios. In our setup, we used the Xilinx® ICAP core. We obtained an average reconfiguration speed of 16.28 MB/s. Significant improvements can be obtained through the use of custom-built controllers as reported in Table 5.3.

Table 5.2. Embedded FIR Filtering system resource utilization (Virtex-4 XCVFX20-11FF672). Largest column filter: N = 32, NH=16, OB=16

Module	Slice	(%)	FF	(%)	LUT	%
PRR (column filter)	2125	25%	3680	21%	3812	22%
Static Region	4973	58%	5226	31%	5998	35%
Overall	7098	83%	8906	52%	9810	57%

Table 5.3. Reconfiguration time for a 178KB bitstream (XCVFX20-11FF672).

Scenario	Reconfiguration speed	Reconfiguration Time
Current	16.28MB/s	11.28ms
Custom [10]	295.4MB/s	0.622ms
Ideal ([84],[11])	400MB/s	0.459ms

Table 5.4. Implementation Comparisons for 2D FIR Filters

	DPR-DA (proposed)	[78]	[80]	[81]	[82]	[59]
Filter type Separable	NCxNR Yes	7x7 Gaussian Yes	5x5 mean Yes	5x5 mean Yes	3x3 No	NCxNR Yes
Coefficients	Variable at run-time via DPR	Fixed	Fixed	Fixed	Variable at run-time	Can be modified at compilation time
Size	NC, NR variable at run-time	Fixed	Fixed	Fixed	Fixed	NC, NR variable at compilation time
Implementation	DA-based	Multiply-and-add	Multiply-and-add	Multiply-and-add	Multiply-and-add	Multiply-and-add
Test case	16x16 symmetric 8-bit input, 16-bit coeffs, 16-bit output	Symmetric filter 8-bit input	Symmetric filter 8-bit input	Symmetric filter 8-bit input	8-bit input	15x15 symmetric 8-bit input, 12-bit coeffs, 12-bit output
Device	Virtex-4	StratixII	Custom-made FPGA	Virtex-4	Virtex-5	Virtex-II Pro
Bitstream size (DPR designs)	120KB	N/A	28 KB	242 KB	N/A	N/A
Resources	1098 Slices 0 BRAMs 0 DSP48	320 ALUT 7 M4K 16 DSP	246K logic gates	3410 Slices 5590 LUTs 5 BRAM	406 LUTs, 402 FFs, 1 BRAM 9 DSP48E	727 slices 15 BRAMs 16 DSP48
Max. Clock frequency (IP)	202 MHz	264 MHz	-	-	125 MHz	201 MHz
Notes	Column and row filter are swapped via DPR	Implemented with ALTERA DSP Builder	Reported: 36.78 dB (Lena)			Xilinx reference design

4) Comparison with other systems: Table 5.4 compares the proposed approach with related 2D FIR implementations found in the literature. For comparison purposes, we chose the symmetric filter with *N=16, NH=16, OB=16*, requiring a 120KB partial bitstream (if this filter were the largest realization).

A fundamental difference between the proposed 2-D implementation and the ones reported in Table 5.4 is the use of a distributed arithmetic approach as opposed to multiply-and-add based methods. Also, note that another advantage of the proposed approach comes from the fact that we only implement one 1-D filter at a time. The use of separable filtering by other approaches requires the allocation of resources for both the row and column filters at all times.

The closest related implementation is given by the Xilinx reference design [59] shown in the last column of Table 5.4. Here, we note that the proposed approach requires more slices but saves on the use of expensive DSP48 blocks and BRAM resources.

The filter reported by [78] uses an ALTERA device that makes use of Adaptive LUTs (ALUT) which can pack more logic than a Xilinx slice. However, as for the Xilinx case, this implementation makes use of 16 DSPs. Similarly, the proposed implementation does compare favorably against [80], [81], and [82].

5.6.2 Multi-objective optimization of the EPA space

This section summarizes the results for EPA optimization for all filter types. For testing different image sizes (VGA, CIF, QCIF), we use the cropped regions of the 'lena' image as presented in Fig. 5.8. For the DoG filter, refer to Table 5.1 for the different parameter combinations that are considered. For the considered spreads of the low-pass filter cases, the quantized coefficients gave zero (or near-zero) values for $N>24$ (note that $12=3\sigma_{max}$, $\sigma_{max}=4$).

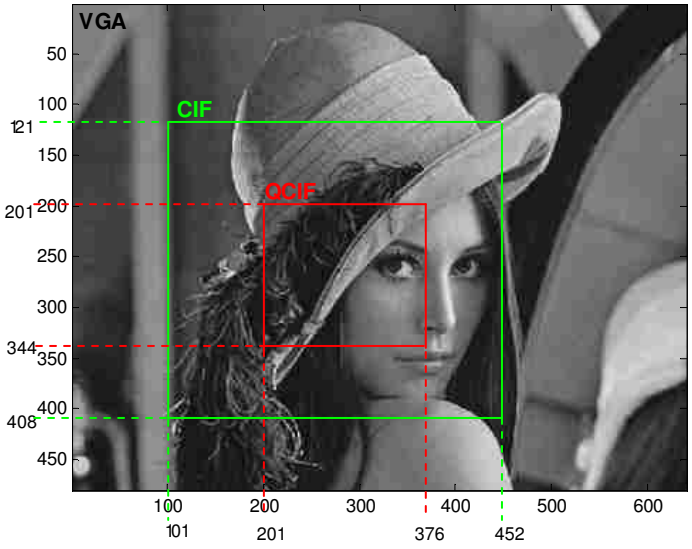

Figure 5.8. Cropped regions for the '*lena*' image. We have the following cropped image sizes: VGA, CIF, and QCIF

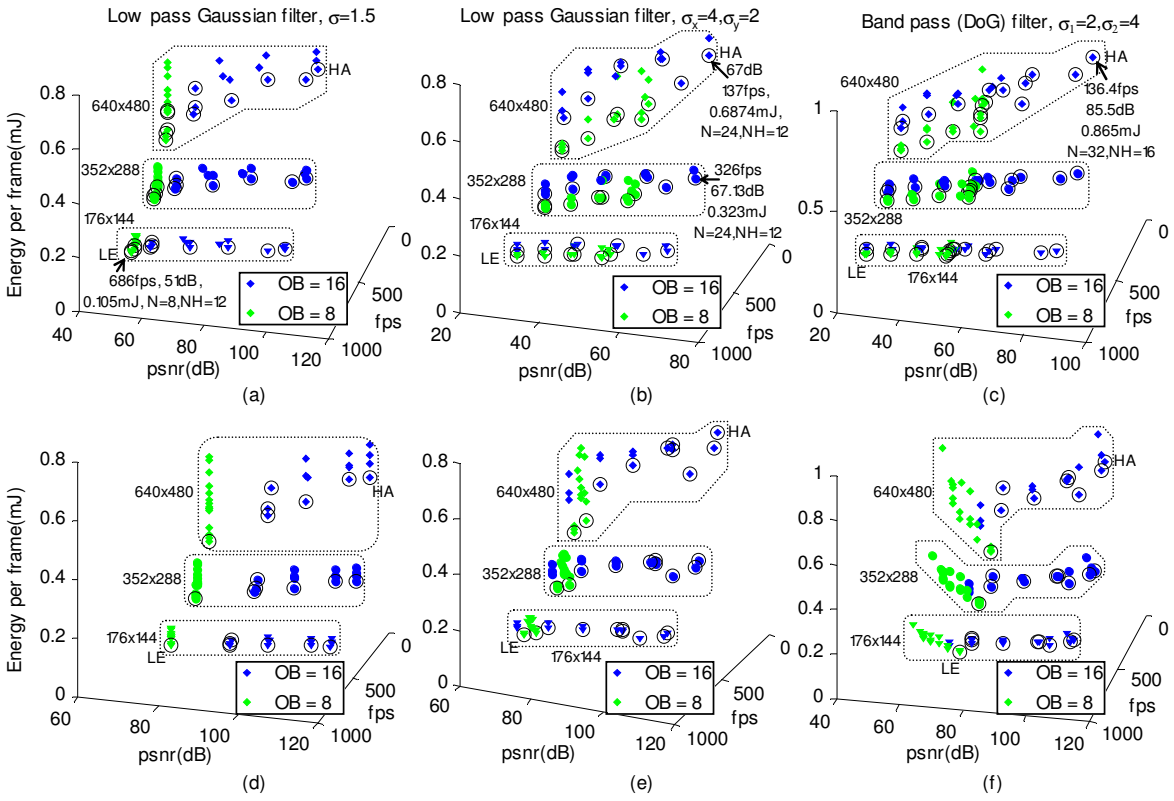

Figure 5.9. Pareto optimal realizations for the three filters and different image sizes. (a) Results for isotropic low-pass filtering. (b) Results for anisotropic low-pass filtering. (c) Results for DoG filtering. (d) Results for impulse response for isotropic low-pass filter. (e) Results for impulse response for anisotropic low-pass filter. (f) Results for impulse response for DoG filter. For (a), (b), (c), refer to Fig. 8 for the input images. Here, 'LE' refers to the lowest energy realization and 'HA' refers to the highest accuracy realization. The Pareto optimal points are circled.

In what follows, we designate Pareto-optimal 2D filter realizations using 'E', 'P', 'A' for energy, performance, and accuracy values. Similarly, we use 'L' for the lowest-possible value and 'H' for the highest. Thus, 'HA' refers to a realization with the highest accuracy. The filter realization with the highest performance is denoted by 'HP'. The filter realization with lowest energy realization is given by 'LE'.

Fig. 5.9 presents the results from EPA space optimization for all filter types and image sizes. Clearly, when using only 8 output bits (*OB=8*), the lowest energy realizations and lowest accuracy results (LE, LA) are obtained. The highest accuracy is achieved by

increasing the number of coefficients, the coefficient bitwidth, and with 16 output bits. As the frame size increases, we also see a significant increase in the required energy per frame. Thus, we are presenting the Pareto-optimal results independently of each frame size.

Figs. 5.10-5.12 show the EPA space and Pareto front for processing the 'lena' image at CIF resolution. For each figure, we show the Pareto-optimal realizations as a function of *N*, *NH*, and *OB* (*OB=8* is grouped in a polygon). Corresponding to Fig. 5.10, Table 5.5 lists the Pareto-optimal realizations and their EPA values for the case of the isotropic low-pass Gaussian filter. In Table 5.5, it is interesting to note that there is not much variation in performance. Performance variations only occur for different frame sizes as demonstrated in Fig. 5.9. Furthermore, with the exception of HP_5, it is also interesting to note that accuracy increases with energy consumption.

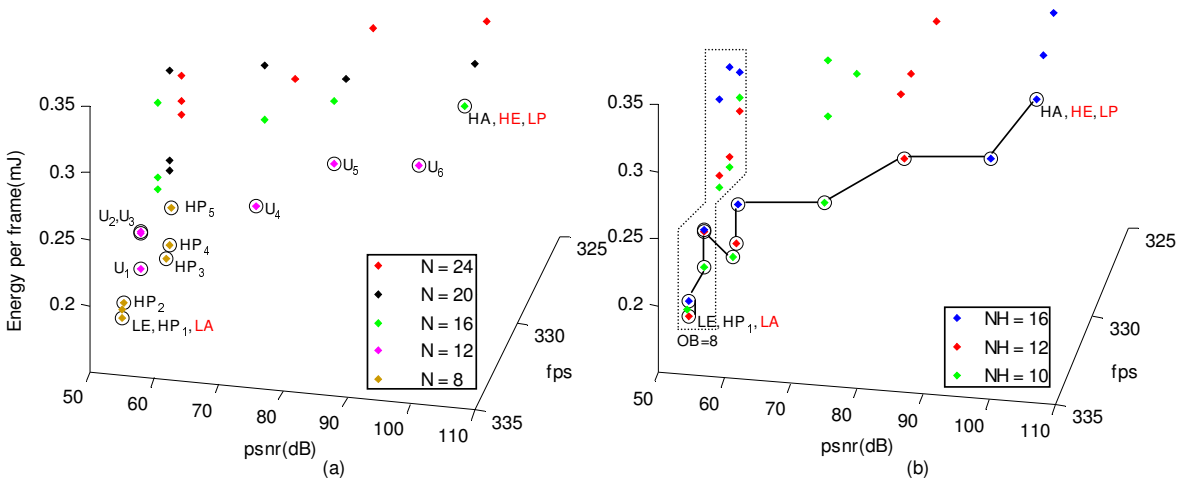

Figure 5.10. Pareto-optimal realizations for the isotropic low pass Gaussian filter (σ=1.5) for CIF resolution: (a) Graph showing dependence on the number of coefficients *N*. (b) Graph showing dependence on the coefficient bitwidth *NH*. Pareto-optimal points are circled. Refer to Table V for EPA measurements and corresponding filter parameters. *OB* refers to the output bitwidth. Refer to text for definitions of 'HP', 'LE', 'HA', 'LP', 'HE', and 'LA'.

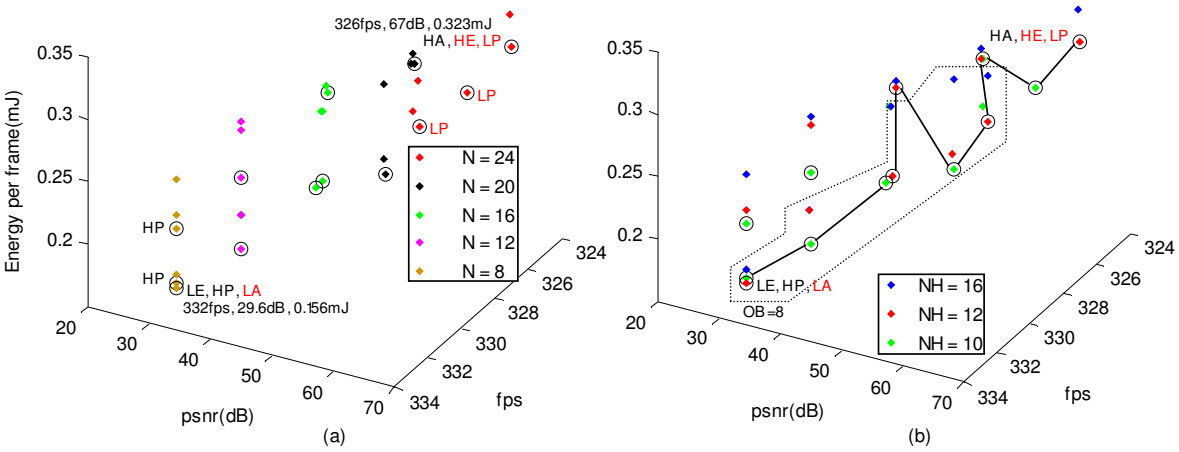

Figure 5.11. Pareto-optimal realizations for anisotropic low-pass Gaussian filter (σ_x=4, σ_y=2) for CIF resolution: (a) Graph showing dependence on the number of coefficients *N*. (b) Graph showing dependence on the coefficient bitwidth *NH*. *OB* refers to the output bitwidth. Refer to text for definitions of 'HP', 'LE', 'HA', 'LP', 'HE', and 'LA'.

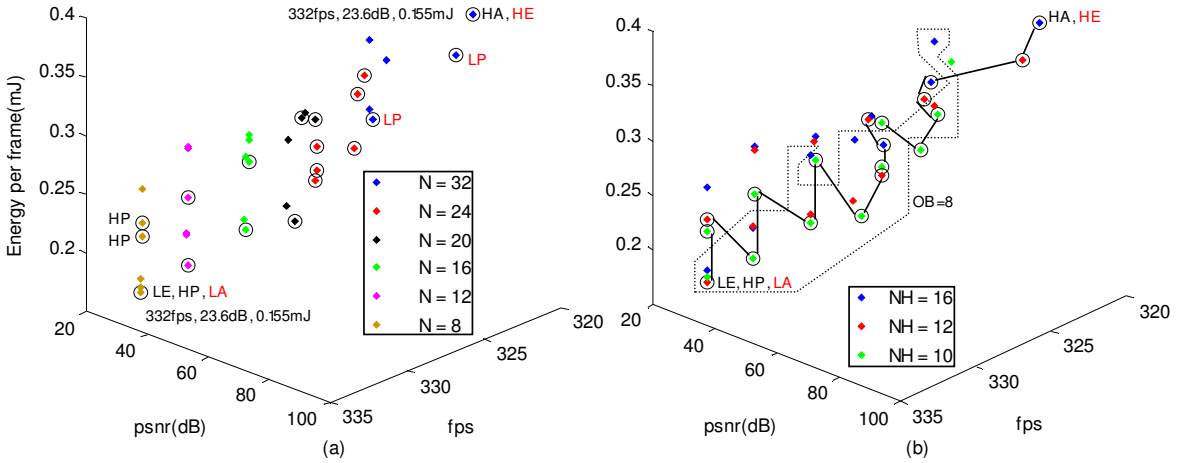

Figure 5.12. Pareto-optimal realizations for DoG filter (σ_1=2, σ_2=4) for CIF resolution: (a) Graph showing dependence on the number of coefficients *N*. (b) Graph showing dependence on coefficient bitwidth *NH*. *OB* refers to the number of output bits. Refer to text for definitions of 'HP', 'LE', 'HA', 'LP', 'HE', and 'LA'.

Even at the largest frame size (VGA), performance results are always over 100 fps. For CIF resolution, performance exceeds 300 fps. Overall, for a fixed frame size, performance does not change significantly. Thus, in what follows, we will restrict the attention to the energy-accuracy space.

Table 5.5. Pareto Optimal realizations for the isotropic, low-pass Gaussian filtering of the lena image at CIF resolution ($\sigma_x=\sigma_y=1.5$). Realizations are sorted by energy consumption. Refer to text for acroynms. $U_{1\text{-}6}$ represent intermediate points.

	N/σ	N	NH	OB	psnr(dB)	Energy per frame (mJ)	fps
LE	5.33	8	12	8	51.2898	0.1548	332.19
HP_2	5.33	8	16	8	51.2974	0.1661	332.19
U_1	8.00	12	10	8	52.0870	0.1736	330.78
U_2	8.00	12	12	8	52.1495	0.2003	330.78
U_3	8.00	12	16	8	52.1500	0.2010	330.78
HP_3	5.33	8	10	16	58.0208	0.2021	332.19
HP_4	5.33	8	12	16	58.6846	0.2124	332.19
U_4	8.00	12	10	16	70.2218	0.2287	330.78
HP_5	5.33	8	16	16	58.7053	0.2414	332.19
U_5	8.00	12	12	16	82.2992	0.2667	330.78
U_6	8.00	12	16	16	95.5084	0.2724	330.78
HA	10.6	16	16	16	100.0664	0.2914	328.69

Table 5.6. Pareto Optimal realizations for DoG filtering of lena at CIF resolution ($\sigma_1=2,\sigma_2=4$). realizations are sorted by energy consumption. Refer to text for acronyms. $U_{1\text{-}11}$ represent intermediate points.

	N	NH	OB	psnr(dB)	Energy per frame (mJ)
LE	8	12	8	23.6345	0.1551
U_1	12	10	8	31.7124	0.1780
U_2	16	10	8	40.2984	0.2062
U_3	20	10	8	49.5758	0.2139
U_4	24	12	8	49.6447	0.2416
U_5	24	10	8	49.8104	0.2508
U_6	24	16	8	49.9967	0.2704
U_7	32	10	8	51.6392	0.2781
U_8	24	10	16	62.3297	0.2807
U_9	24	12	16	63.1992	0.3276
U_{10}	24	16	16	65.8199	0.3464
U_{11}	32	12	16	78.7569	0.3586
HA	32	16	16	84.4371	0.3991

Fig. 5.13 shows the results from energy-accuracy space optimization for all filters. Results refer to filtering the '*lena*' image at CIF resolution. Table 5.6 lists the 2D Pareto-optimal realizations for the DoG filter. Here, it is interesting to note that accuracy increases with energy consumption and the number of output bits.

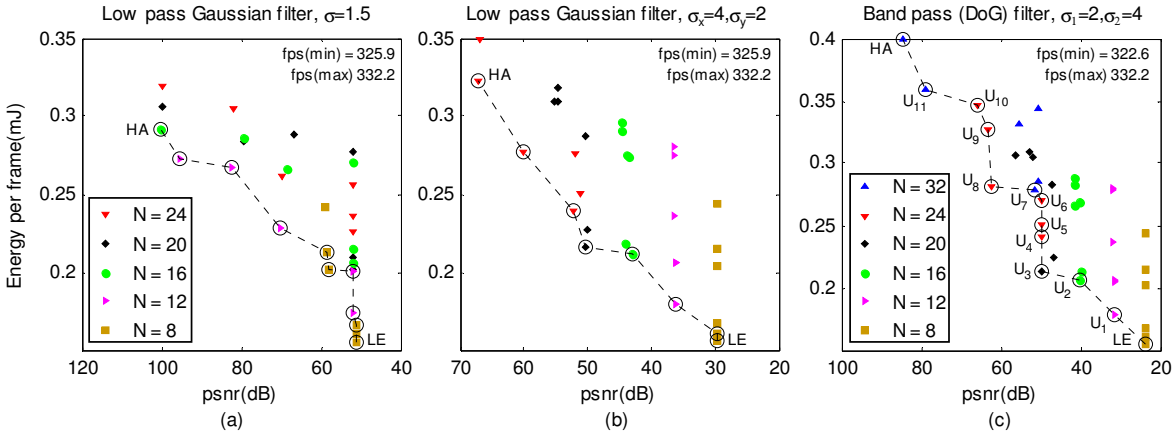

Figure 5.13. 2D Pareto-optimal realizations for Energy-Accuracy space for all filter types at CIF resolution: (a) Isotropic, low-pass filter. (b) Anisotropic, low-pass filter. (c) DoG filter. In all cases, we also summarize performance in terms of the minimum and maximum fps. Here, 'LE' refers to the lowest energy realization, and 'HA' refers to the highest accuracy realization.

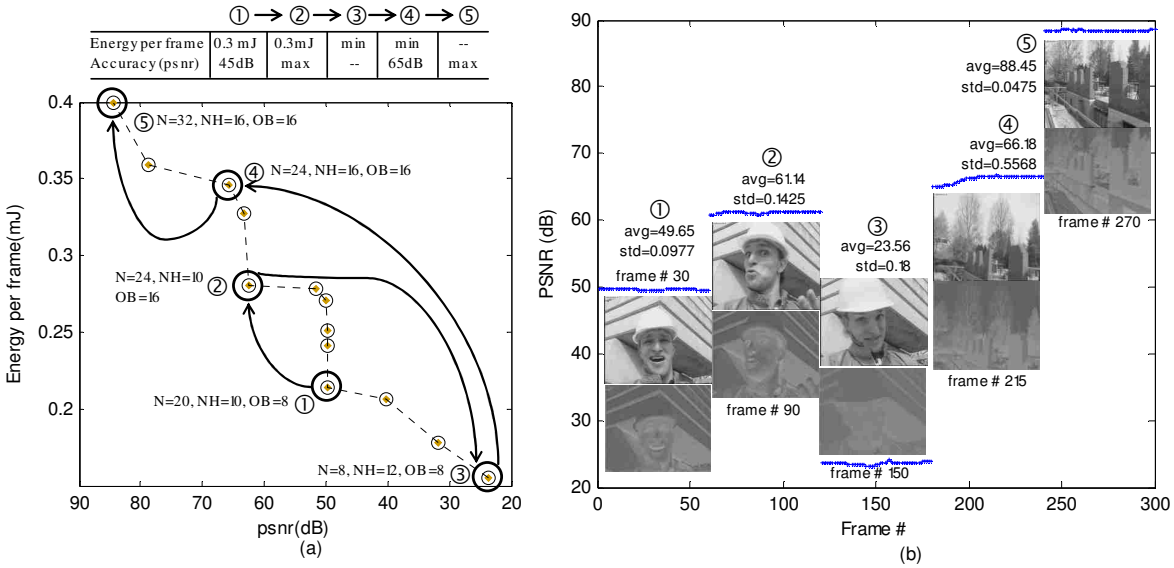

Figure 14. Dynamic EPA management example for DoG filtering of the *foreman* video sequence (CIF resolution): (a) Sequence of 2D FIR realizations that meet dynamic constraints. The dynamic constraints are displayed above the graph. (b) Video filtering accuracy achieved by dynamic EPA management.

5.6.3 Dynamic EPA management optimization

We next demonstrate dynamic EPA management on a video example. In this case, we consider a time-varying sequence of energy-accuracy constraints for the DoG filter as listed on the top of Fig. 5.14(a). The goal of the proposed dynamic management approach is to meet the constraints by using Pareto-optimal realizations listed in Table 5.6.

In the proposed approach, it is assumed that the accuracy results from the '*lena*' image will also work for the '*foreman*' video sequence. More generally, we will need to know that the video images used in estimating accuracy will correspond to the testing cases. As we shall see, this assumption seems to apply here. Nevertheless, even if the assumption does not hold, a user can dynamically adjust the constraints to match expectations. The time-varying constraints are:

1. Require Accuracy≥45 dB and Energy≤0.3mJ per frame.
2. Maximize Accuracy subject to Energy≤0.3mJ per frame.
3. Minimize Energy consumption per frame.
4. Minimize Energy per frame subject to Accuracy≥65dB.
5. Maximize Accuracy.

Here, recall that when different realizations are possible, we select the one with minimum energy (e.g. see point ①). The management of the EPA constraints leads to the 2D FIR filter realizations shown in Fig. 5.14(a). Furthermore, as shown in Fig. 5.14(b), the accuracy constraints are well met.

5.7 Conclusions

We have presented a 2D FIR filtering framework for determining Pareto-optimal realizations in the Energy-Performance-Accuracy (EPA) space. We also demonstrate how the use the Pareto-optimal realizations can be used to meet time-varying EPA constraints. This provided for an effective method for dynamic EPA management.

We presented results over three 2D Gaussian filters. In each case, we provide a collection of Pareto-optimal solutions based on maximizing accuracy and performance, while minimized consumption of energy per frame.

Dynamic EPA management is demonstrated on a standard video sequence. Here, it was clearly demonstrated how energy-accuracy constraints and optimization requirements can be easily met using a pre-computed set of Pareto-optimal realizations. Future work can focus on the automatic generation of time-varying constraints. For example, the detection of a scene change may trigger a requirement for an increase in accuracy. Similarly, when a scene remains the same over a long period of time, we may want to trigger a requirement for a decrease in energy consumption.

Chapter 6

Concluding Remarks, Future Work, and Recommendations

6.1 Concluding remarks

A framework has been presented for the generation of optimal implementations in the Power/Energy, Performance, and Accuracy (PPA/EPA) spaces. The framework allows for dynamic PPA/EPA management for digital signal, image, and video processing applications that can meet real-time PPA/EPA constraints.

The framework was tested on the development of single-pixel processors, 1-D FIR, and 2-D FIR filtering architectures. In addition, dynamic management was performed using Pareto-optimal realizations that can be used to meet time-varying PPA/EPA constraints. This provided for an effective method for dynamic PPA/EPA management.

In the case of the pixel processor core, the Pareto optimal points were generated by considering different number of pixel processor cores, number of inputs bits, number of output bits, and clock frequencies. The validity of the approach was verified by the fact that over 40% of the considered implementations were found to be Pareto-optimal.

As for the 2D FIR filtering system, the Pareto optimal points were generated by considering different numbers of coefficients, coefficient bit-width, and output bitwidth. Results were presented for three different 2D Gaussian filters. The approach worked in the sense that the Pareto-optimal realizations were generated based on combinations of different parameters, i.e. no parameter predominated.

For both the pixel processor and the 2D FIR filter, a collection of Pareto-optimal solutions (computed offline) was provided. These solutions were based on maximizing accuracy and performance, while minimizing consumption of power/energy per frame.

Dynamic PPA/EPA management was demonstrated on a standard video sequence and a standard image. Here, it was clearly demonstrated how power/energy-performance-accuracy constraints and optimization requirements can be easily met using a pre-computed set of Pareto-optimal realizations.

The results suggest that the general framework can be applied to a variety of digital signal, image, and video processing systems. This framework can be greatly improved by the automatic generation of time-varying constraints. For example, the detection of a scene change may trigger a requirement for an increase in accuracy. Similarly, when a scene remains the same over a long period of time, we may want to trigger a requirement for a decrease in power/energy consumption. Ultimately, this framework will lead to exciting new methods that allow for systems to only switch between architectures that are optimal in the multi-objective sense.

6.2 Future Work and Recommendations

In what follows, I provide a set of recommendations for future work:

- The framework for dynamic management presented here has been tested on a medium-sized Virtex-4 FPGA, where the static power consumption was not too much of a problem. Newer high-end FPGA families (e.g., Virtex-5, Virtex-6, Virtex-7, and Kirtex-7) exhibit very high static power consumption even for the smallest device in the family. A way to deal with this issue is to use low-power FPGA families that

support dynamic partial reconfiguration/dynamic frequency control (e.g., Spartan-6, Artix-7). Another option is to consider dynamic reconfiguration with architectures that exhibit large variations in resource consumption. In this case, dynamic power consumption variation will be comparable to the static power consumption. This second approach will still not reduce device static power. Thus, the basic recommendation here is to consider re-implementation on the latest, low-power device (e.g., Artix-7).

- Direct current measurement requires custom-built boards that provide access to the power rails. Newer FPGAs can measure voltage in real-time via the System Monitor, so the power measurement capability depends on the board. Most commercial boards allow for power measurement on only one rail (if any). As a result, throughout this Dissertation, the Xilinx Power Analyzer was used, whose accuracy has been corroborated with direct power measurements available in the ML605 Development Board (the power regulators provide information through I^2C). Thus, we consider the use of power estimation software tools to be the most convenient option that provides decent estimates and allows us to apply the framework to any device.
- It would be interesting to test the approach using different objective functions. Instead of PSNR, we can use SSIM (structural similarity). Instead of frames per second, bandwidth could be more informative. Note that the conversion from frames per second to bandwidth is straightforward. We just need to use the number of frames per second, the frame size, and the I/O bitwidth to determine the I/O bandwidth. Instead of energy/power, we might also want to use hardware resource consumption.

- There are limits to the use of dynamic partial reconfiguration. The basic idea is that DPR makes sense when the dynamic reconfiguration rate overhead is low. Thus, it does not make sense to have high reconfiguration rates. For the architectures shown, we studied the effect of the reconfiguration rate ([9], [17]) and found that reconfiguring made sense for our applications. Therefore, before attempting to apply this proposed framework, one should assess whether the reconfiguration rate makes sense. On the other hand, the reconfiguration overhead is not a problem for dynamic frequency control since it can be accomplished in tens of cycles. Alternatively, dynamic reconfiguration time can be reduced by context switching. Here, dynamic reconfiguration would require two different regions. This allows the system to work with one region while dynamically reconfiguring the other. Clearly, this requires that we have prior specification of the requirements for the dynamic region, additional resources overhead, and a DPR controller that can operate in parallel with the rest of the system.
- FPGA vendors should develop methods to speed up the process of dynamic partial reconfiguration, including: i) increasing the bit-width of the ICAP from 32 to 64 bits, ii) allowing for a dedicated bus that supports data streaming at high frequencies (e.g. 200 MHz), iii) allowing the streaming of compressed partial bitstreams to the ICAP. The reconfiguration overhead can be made negligible by the application of these three techniques, and it would enable a new frontier where DPR becomes a common technique. Naturally the usefulness of this approach assumes that the DPR controller overhead should remain relatively low as compared to the whole design.

- Another recommendation for FPGA vendors is to develop a built-in capability to measure current for all the FPGA power rails. Currently, some Xilinx® FPGAs offer voltage measurement, and current measurement capability depends on the board. This would certainly take dynamic power management a step further.
- The reconfiguration controller presented only accepts external constraints, but does not generate them. Future work will deal with the devising of a reconfiguration controller that is content-based and power-aware (System Monitor). It will reconfigure based on both dynamic hardware sensing and dynamic software constraint generation. Dynamic hardware sensing can be based on real-time measurement of power. Dynamic software constraints can be generated from video scene changes, or a detection of an object of interest, etc.

Appendix A

VHDL code description

A.1 Pixel Processor and 1D FIR filter architectures

The parameterized VHDL code for these architectures is depicted in Fig. A.1. Both cores are fully parameterized architectures that allow for the creation of a large set of hardware realizations. The pixel processor architecture was presented in Fig. 2.1; the file '*LUT_NItoNO.vhd*' implements a NI-to-NO LUT, and the file '*LUT_NIto1*.vhd' implements a NI-to1 LUT. The 1D FIR architecture was presented in Fig. 5.3; the file '*fir_block.vhd*' describes the FIR filter block, and the file '*LUTn.vhd*' describes a LUT with L inputs and LO outputs.

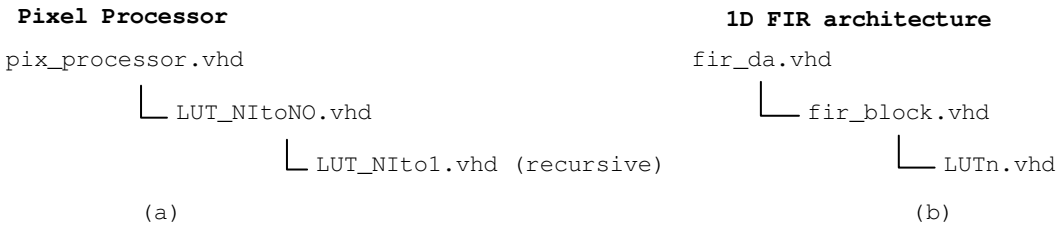

Figure A.1 VHDL code for Pixel Processor and 1D FIR filter

As an example of parameterization, Fig. A.2 shows the entity VHDL declaration of the pixel processor core with all the generic parameters.

```
entity pix_processor is
   generic ( NC: INTEGER:= 4; -- number of cores
             NI: INTEGER:= 8; -- number of input bits per pixel
             NO: INTEGER:= 8; -- number of output bits per pixel
             F: INTEGER:= 1; -- type of function (1..5)
             file_LUT: STRING:= "LUT_values.txt");  -- text file for LUT values
   port ( dyn_in: in std_logic_vector (NC*NI - 1 downto 0);
          dyn_out: out std_logic_vector (NC*NO - 1 downto 0));
end pix_processor;
```

Figure A.2 VHDL code ('entity' declaration) for the Pixel Processor architecture

Recall that the utilization of these cores within an embedded system requires the development of hardware interfaces, which were developed in VHDL. In the case of the pixel processor, the interface was connected to the PLB bus with burst support (see Fig. 2.5). For the 1D FIR filter processor, an interface that connects to the Fast Simplex Link bus, was designed around the 1D FIR filter core (see Fig. 3.8). Finally, for the 2D FIR filter processor, the interface was attached to the Fast Simplex Link Bus. Here, the 2D filter is implemented by cyclic swapping of the row and column filters (see Fig. 4.2)

A.2 Dynamic Frequency Control core

A core that allows the dynamic modification of the frequency of operation was described in Fig. 2.3. The core is made of two VHDL files. The first file ('*dcm_ctrl.vhd*') is a stand-alone DCM control core (not connected to any bus) that allows the for the run-time modification of frequency. The second file ('*dcmctrl_DCRslv.vhd*') is a Device Control Register (DCR) slave interface around the stand-alone DCM control core that receives orders from the embedded processor so as to be able to manage the DCM control core.

Appendix B

Reconfigurability on FPGAs

An FPGA is a programmable device consisting of an array of programmable logic blocks, surrounded by programmable I/O blocks, and a programmable interconnection network.

A function to be implemented in FPGA is partitioned in modules, each of which can be implemented in a logic block. The logic blocks are then connected together using the programmable interconnection. All three basic components of an FPGA (logic blocks, I/O blocks, and interconnection network) can be re-programmed by the user [88].

B.1 Dynamic Partial Reconfiguration

Dynamic Partial Reconfiguration enables the run-time allocation and de-allocation of hardware resources by modifying or switching off portions of the FPGA while the rest remains intact, continuing its operation.

The operating FPGA design is modified by loading a partial configuration file, usually a partial bit file. After a full bit file configures the FPGA (full reconfiguration), partial bit files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfiguration. Fig. B.1 illustrates the idea where the Block A (user-defined reconfigurable region) can be modified by any of the partial bit files (A1.bit, A2.bit, A3.bit, or A4.bit). The static region remains functioning and it is completely unaffected by the loading of a partial bit file [86].

This technology can dramatically extend the capabilities of FPGAs. In addition to potentially reducing size, weight, power, and cost, dynamic partial reconfiguration

enables new types of FPGA designs that provide efficiencies not attainable with conventional design techniques. The main FPGA vendors, ALTERA and Xilinx provide commercial support for this technology.

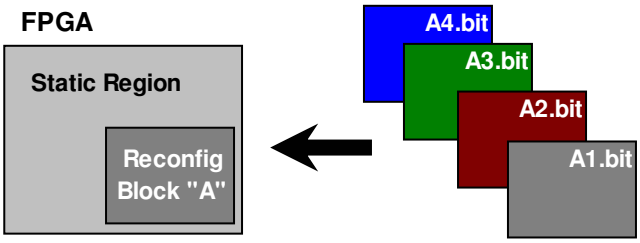

Figure B.1. Basic premise of Partial Reconfiguration ([86])

B.2 Technology that enables reconfiguration (full/partial) of FPGAs

Current ALTERA and Xilinx FPGAs use a memory-based paradigm for computations as well as for the realization of interconnections. Among the programmable technologies available, we can list SRAM, EEPROM, and Flash-based [88]. SRAM devices, the dominate technology for FPGAs, are based on static CMOS memory technology, and are re- and in-system programmable.

Figure B.2. Basic Xilinx SRAM cell [88].

In an SRAM-based FPGA, the states of the logic blocks, I/O blocks, and interconnections, are controlled by the output of SRAM cells (figure B.2). The basic SRAM configuration cell (Fig. B.2) is constructed from two cross-coupled inverters and uses a standard CMOS-process. The major advantage of this technology is that FPGAs can be configured indefinitely. A new connection or function is implemented by a change on the SRAM cells values. Moreover, the device can be reconfigured in-circuit (while it is mounted on the circuit board with the other components) very quickly and on-the-fly (while the device is operating).

A major disadvantage of SRAM programming technology is its large area. It takes at least five transistors to implement an SRAM cell, plus at least one transistor to serve as programmable switch [89]. Furthermore, the device is volatile, i.e. the configuration of the device stored in the SRAM-cells is lost if the power is cut off. Thus, external storage or non-volatile devices such as CPLDs, EPROM or Flash devices, are required to store the configuration and load it into the FPGA-device at power-on.

Appendix C

Related publications

This section lists published work related with the dissertation: A system that reconfigures among single-pixel operations is presented in [C.1]. A FIR filter implemented with distributed arithmetic, whose coefficients can be modified, is presented in [C.2]. A platform that allows for rapid swapping of image processing algorithms is presented in [C.3]. A revamped version of the FIR filter, where it is also possible to modify the entire filter structure is presented in [C.4]. Preliminary results of a 2D separable filterbank are presented in [C.5]. Some ancillary work has also been presented. A dynamically reconfigurable computing model for video processing applications is presented in [C.6]. In addition, [C.7] describes a system that can automatically obtain partial bitstreams at running-time via the Ethernet link. In [C.8], a comparison of the energy-accuracy space of a 2D FIR Filter for both FPGA with DPR and GPU implementations is presented.

[C.1] D. Llamocca, M. Pattichis, and A. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", in *Proceedings of 2009 International Conference on Field Programmable Logic and Applications*, Prague, Czech Republic, Sep. 2009.

[C.2] D. Llamocca, M. Pattichis, and A. Vera, "A dynamically reconfigurable platform for fixed-point FIR filters," in *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig '09)*, pp. 332–337, Cancun, Mexico, Dec. 2009.

[C.3] D. Llamocca, M.S. Pattichis, and G. A. Vera, "A dynamic computing platform for image and video processing applications," in *Proceedings of the 43rd Asilomar Conference on Signals, Systems and Computers*, pp. 327–331, Pacific Grove, CA, USA, Nov. 2009.

[C.4] D. Llamocca, M. Pattichis, and G. Alonzo Vera, “Partial Reconfigurable FIR Filtering system using Distributed Arithmetic”, *International Journal of Reconfigurable Computing*, vol. 2010, Article ID 357978, 14 pages, 2010.

[C.5] D. Llamocca, M. Pattichis, “Real-time dynamically reconfigurable 2-D filterbanks”, in *Proceedings of 2010 IEEE Southwest Symposium on Image Analysis & Interpretation*, Austin, TX, May. 2010.

[C.6] G. A. Vera, D. Llamocca, M. S. Pattichis, and J. Lyke, “A dynamically reconfigurable computing model for video processing applications,” in *Proceedings of the 43rd Asilomar Conference on Signals, Systems and Computers*, pp. 327–331, Pacific Grove, Calif, USA, November 2009.

[C.7] D. Llamocca, M.S. Pattichis, G. A. Vera, and J. Lyke, “Dynamic Partial Reconfiguration through Ethernet Link”, in *Proceedings of the 2010 AIAA Infotech Conference at Aerospace*, Atlanta, GA, USA, April 2010.

[C.8] D. Llamocca, C. Carranza, and M. Pattichis, “Separable FIR Filtering in FPGA and GPU implementations: Energy, Performance, and Accuracy considerations”, in *Proceedings of 2011 International Conference on Field Programmable Logic and Applications FPL’2011*, Chania, Greece, Sep. 2011.

Other publications:

[C.9] A. Vera, D. Llamocca, M. Pattichis, W. Kemp, D. Alexander, and J. Lyke, “Dose Rate Upset Investigations on the Xilinx IV Field Programmable Gate Arrays”, in *Proceedings of the 2007 IEEE Radiation Effects Data Workshop*, Honolulu, HI, Oct. 2007.

[C.10] A. Vera, D. Llamocca, J. Fabula, W. Kemp, R. Marquez, W. Shedd, D. Alexander, “Xilinx Virtex V Field Programmable Gate Array Dose Rate Upset Investigations”, in *Proceedings of the 2008 IEEE Radiation Effects Data Workshop*, Tucson, AZ, Oct. 2008.

[C.11] I. Steinwart, J. Theiler, and D. Llamocca, “Using support vector machines for anomalous change detection”, in *Proceedings of the 2010 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Honolulu, HI, July 2010.

References

[1] Alan Bovik ed., *The Essential Guide to Video Processing*, Academic Press, Elsevier, 2nd Edition, 2009.

[2] A. Laffely, J. Liang, P. Jain, N. Weng, W. Burleson, R. Tessier, "Adaptive System on a Chip (aSoC) for Low Power Signal Processing", in *Proc. of the Asilomar Conference of Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2001.

[3] W. P. Burleson, P. Jain, and S. Venkatraman, "Dynamically parameterized architecture for power-aware video coding: Motion estimation and DCT", in *Proc. 2nd USF Int. Workshop Digital and Computational Video*, 2001, pp. 8-12.

[4] R. Chamberlain, E. Hemmeter, R. Morley, and J. White, "Modeling the power consumption of audio signal processing computations using customized numerical representations", in *Proc. of the 36th Annual Simulation Symposium*, pp..249-255, April 2003.

[5] J. Noguera, I.O. Kennedy, "Power Reduction in Network Equipment through Adaptive Partial Reconfiguration", in *Proceedings of the 2007 International Conference on Field Programmable Logic and Applications (FPL'07)*, pp. 240-245, Amsterdam, The Netherlands, Nov. 2007.

[6] G.A. Vera, "*A dynamic arithmetic architecture: precision, power, and performance considerations*", Ph.D. Dissertation, University of New Mexico, Albuquerque, NM, USA, May 2008.

[7] J. Huang, J. Lee, "A Self-Reconfigurable Platform for Scalable DCT Computation using compressed partial bitstreams and BlockRAM Prefetching", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 19, pp. 1623-1632, Nov. 2009.

[8] S. Boyd and L. Vanderberghe, Convex Optimization. Cambridge, U.K: Cambridge Univ. Press, 2004.

[9] D. Llamocca, M. Pattichis, and A. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", in *Proceedings of 2009 International Conference on Field Programmable Logic and Applications FPL'2009*, Prague, Czech Republic, Sep. 2009.

[10] C. Claus et al., "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput", in *Proceedings of the International Conference on Field Programmable Logic and Applications* (FPL'08), pp.535-538, Heidelberg, Germany, Sept. 2008.

[11] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-Time Partial Reconfiguration Speed Investigation and architectural design space exploration", in *Proceedings of the International Conference on Field Programmable Logic and Applications FPL'2009*, Prague, Czech Republic, Sept. 2009.

[12] J.C. Hoffman and M.S. Pattichis, "A High-Speed Dynamic Partial Reconfiguration Controller using Direct Memory Access through a Multiport Memory Controller and Overclocking with Active Feedback", *International Journal of Reconfigurable Computing*, vol. 2011, Article ID 439072, 10 pages, 2011.

[13] A. Sudarsanam, R. Barnes, J. Carver, R. Kallam, and A. Dasu, "Dynamically reconfigurable systolic array accelerators: A case study with extended Kalman filter and discrete wavelet transform algorithms", *Computers & Digital Techniques, IET*, vol. 2, issue 2, March 2010.

[14] M. Fons, F. Fons, and E. Cantó, "Fingerprint Image processing acceleration through run-time reconfigurable hardware", *IEEE Trans. On Circuits and Systems II: Express Briefs*, vol. 47, No. 12, Dec. 2010.

[15] A. Afandi, A. Abbes, "Efficient reconfigurable architectures for 3D medical image compression", in *Proceedings of International Conference on Field-Programmable Technology*, Sydney, Australia, Dec. 2009.

[16] S. Bouchoux, E.-B. Bourennane, and M. Paindavoine, "Implementation of JPEG2000 arithmetic decoder using dynamic reconfiguration of FPGA", in *Proceedings of the 2004 International Conference on Image Processing ICIP'04*, Singapore, Oct. 2004.

[17] D. Llamocca, M. Pattichis, and G. Alonzo Vera, "Partial Reconfigurable FIR Filtering system using Distributed Arithmetic", *International Journal of Reconfigurable Computing*, vol. 2010, Article ID 357978, 14 pages, 2010.

[18] S. Sowmya, R. Paily, “FPGA Implementation of Image Enhancement Algorihtms”, in Proceedings of the *International Conference on Communications and Signal Processing ICCSP’2011*, Kozhikode, India, Feb. 2011.

[19] LogiCORE IP Gamma Correction (DS719), v3.0 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, September 2010.

[20] Video and Image Processing Suite User Guide, v11.0 ed., Altera Corp, 101 Innovation Drive, San Jose, CA, May 2011

[21] Dong-U Lee, R.C.C. Cheung, J.D. Villasenor, “A flexible architecture for precise gamma correction”, *IEEE Trans. On Very Large Scale Integration (VLSI) Systems*, vol. 15, issue 4, pp.474-478, April 2007.

[22] Wang Bing-jian, Liu Shang-qian, Li Qing, and Zhou Hui-xin, “A real-time contrast enhancement algorithm for infrared images based on plateau histogram”, *Infrared Physics and Technology*, Elsevier, pp. 77-82, July 2005.

[23] A. M. Alsuwailem and S.A. Alshebeili, “A new approach for real-time histogram equalization using FPGA”, in *Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems*, Dec. 2005.

[24] M. Chandrashekar, U. N. Kumar, K. S. Readdy, and K.N. Raju, “FPGA implementation of high speed infrared image enhancement”, *International Journal of Electronic Engineering Research*, vol. 1, no. 3, pp. 1480-1485, 2002.

[25] K. Bondalapati, V.K. Prasanna, “Dynamic precision management for loop computations on reconfigurable architectures”, in *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 1999.

[26] Virtex-4 FPGA User Guide (UG070), v2.6 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, December 2008.

[27] Power Methodology Guide (UG786), v13.1 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, March 2011.

[28] Virtex-4 FPGA Data Sheet: DC and Switching Characteristics (DS302), v3.3 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, June 2008.

[29] M. Hatamian, and G.L. Cash, "A 70 MHz 8 bit x 8 bit parallel pipelined multiplier in 2.5 μm CMOS", *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 505-513, Aug. 1986.

[30] K.G. Smitha, and A. P. Vinod, "A reconfigurable high-speed RNS-FIR channel filter for multi-standard software radio receivers", in *Proceedings of 11th IEEE Singapore International Conference on Communication Systems*, Jan. 2009, pp 1354-1358.

[31] Gallazzi, F., Torelli, G., Malcovati, P., and Ferragina, V., "A digital multistandard reconfigurable FIR filter for wireless applications", *Proceedings of 14th IEEE International Conference on Electronics, Circuits and Systems*, 2007, pp 808-811.

[32] H. Bruce, R. Veljanovski, V. Owall, and J. Singh, "Power optimization of a reconfigurable FIR-filter", *IEEE workshop on Signal Processing Systems*, 2004.

[33] R. Mahesh, and A.P. Vinod, "New reconfigurable architectures for implementing FIR filters with low complexity", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, January 2010, pp 275-288.

[34] C. Chou, S. Mohanakrishman, and J. B. Evans, "FPGA implementation of digital filters", in *Proceedings of Signal Processing Applications Technol.*, Santa Clara, CA, 1993.

[35] T. Do, H. Kropp, C. Reuter, and P. Pirsch. "A Flexible Implementation of High-Performance FIR Filter on Xilinx FPGAs"; *Field-Programmable Logic and Applications. From FPGAs to Computing Paradigm: 8th International Workshop, FPL '98,* 1998, pp. 441-445.

[36] S. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review", *IEEE Transactions on Acoustics, Speech and Signal Processing Magazine*, 4-19, 1989.

[37] A. Vera, D. Llamocca, M. Pattichis, and J. Lyke, "A Dynamically Reconfigurable Computing Model for Video Processing Applications", in *Proceedings of the 2009 Asilomar Conference on Signal, Systems and Computers*, Pacific Grove, CA, Nov. 2009.

[38] D. Llamocca, M. Pattichis, A. Vera, "A Dynamically Reconfigurable Platform for Fixed-Point FIR Filters", in *Proceedings of ReConFig'09*, Cancun, Mexico, Nov. 2009, pp. 332-337.

[39] E. Tan, A. Wahab, and K. Wong, "Programmable DSP systems using FPGA," in *Proceedings of the IEEE conference on Digital Signal Processing Applications*, vol. 2, 1996, pp. 654–658.

[40] R. Sidhu, and V. K. Prasanna, "Efficient Metacomputation Using Self-Reconfiguration", *Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream 12th International Conference, FPL 2002,* Montpellier, France, 2002, pp. 85-92.

[41] Delahaye, J., Palicot, J., Moy, C., and Leray, P., "Partial Reconfiguration of FPGAs for Dynamical Reconfiguration of a Software Radio Platform", in *Proceedings of 2007 Mobile and Wireless Communications Summit*, Budapest, Hungary, July 2007, pp. 1-5.

[42] T. Rissa, R. Uusikartano, and J. Niittylahti, "Adaptive FIR filter architectures for run-time reconfigurable FPGAs", in *Proceedings of 2002 IEEE International Conference on Field-Programmable Technology,* 2002, pp 52-59.

[43] C. Choi, and H. Lee, "A Reconfigurable FIR Filter Design on a Partial Reconfigurable Platform", in *Proceedings of ICCE'06*, Hanoi, Vietnam, Oct. 2006, pp. 352-355.

[44] C. Choi, and H. Lee, "A self-reconfigurable adaptive FIR filter system on Partial Reconfigurable Platform", in *IEICE Transactions in Information and Systems*, Vol. E90-D, No. 12, Dec. 2007, pp. 1932-1938.

[45] Ken Chapman, Xilinx's Application Note 054, "Constant Coefficient Multipliers for the XC4000E", Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Dec. 1996.

[46] "Distributed Arithmetic FIR Filter (DS240)", v9.0 ed., Xilinx Inc., 2100 Logic Drive, San Jose, Ca, 05124, April 2005.

[47] K. Bruneel, F. Abouelella, and D. Stroobandt, "Automatically mapping applications to a self-reconfiguring platform", in *Proceedings of Design, Automation, and Test in Europe,* 2009.

[48] S. Chevobbe, and S. Guyetant, “Reducing Reconfiguration Overheads in Heterogeneous Multicore RSoCs with Predictive Configuration Management”, *International Journal of Reconfigurable Computing*, Volume 2009 (2009), Article ID 390167.

[49] “Implementing FIR Filters in FLEX Devices (AN73)”, v1.01 ed., Altera Corp., 101 Innovation Drive, San Jose, CA, 95134, Feb. 1998.

[50] “Fast Simplex Link (FSL) Bus Product Specification (DS449)”, v2.11a ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Jun. 2007.

[51] “Early Access Partial Reconfiguration User Guide for ISE 9.204i (UG208)”, v1.2 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Sep. 2008.

[52] A. DeHon, R. Huang, and J. Wawrzynek, “Hardware-assisted fast routing”, in *Proc. IEEE FCCM’92*, Apr. 2002.

[53] Sahambi, J.S., Tandon, S.N., and Bhatt, R.K.P., “Using wavelet transforms for ECG characterization. An online digital signal processing system”, IEEE Engineering in Medicine and Biology Magazine, vol. 16, Issue 1, pp. 77-83, 1997.

[54] E. Kyriacou, C. Pattichis, M. Pattichis, A. Jossif, L. Paraskeva, A. Konstantinides, and D. Vogiatzis, "An m-Health Monitoring System for Children with Suspected Arrythmias," in *Procedings of the 29th Annual International Conference of the IEEE EMBS*, 2007, pp. 1794-1797.

[55] A. Panayides, M.S. Pattichis, C.S. Pattichis, C.P. Loizou, M. Pantziaris, A. Pitsillides, "Robust and Efficient Ultrasound Video Coding in Noisy Channels Using H.264," in *Proceedings of the 31st Annual International Conference of the IEEE EMBS,* 2009, pp. 5143-5146.

[56] G.B. Moody, R.G. Mark, "The MIT-BIH Arrhythmia Database on CD-ROM and Software for use with it," Computers in Cardiology, 1990, pp. 185-188.

[57] Alan Bovik ed., Handbook of Image and Video Processing. Academic Press, 1st Edition, May 2000.

[58] D. Llamocca, M. Pattichis, “Real-time dynamically reconfigurable 2-D filterbanks”, in *Proceedings of the 2010 IEEE Southwest Symposium on Image Analysis & Interpretation*, Austin, TX, May. 2010.

[59] "Two-dimensional Linear Filtering (XAPP933) by Robert Turney", v1.1 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Oct. 2007.

[60] S. Collange, D. Defour, A. Tisserand, "Power Consumption of GPUs from a Software Perspective", in Proceedings of the 9th International Conference on Computational Science (ICCS'09), pp.914-923, Springer, 2009.

[61] V. Podlozhnyuk, "Image Convolution with CUDA", NVIDIA, June 2007.

[62] Cope, B., Cheung, P.Y.K., Luk, W., Witt, S., "Have GPUs made FPGAs redundant in the field of video processing?", in *Proceedings of the 2005 IEEE International Conference on Field Programmable Technology*, pp. 111-118, Singapore, Dec. 2005.

[63] Jones, D.H., Powell, A., Bouganis, C.-S., Cheung, P.Y.K., "GPU versus FPGA for High Productivity Computing", in *Proceedings of the International Conference on Field Programmable Logic and Applications FPL'2010*, Milan, Italy, Sep.2010.

[64] CUDA C Programming Guide, NVIDIA, v 3.2, Sept. 2010.

[65] Alan Bovik ed., The Essential Guide to Image Processing, Academic Press, Elsevier, 2nd Edition, 2009.

[66] G. Alonzo Vera, Marios Pattichis, and James Lyke, "A dynamic dual fixed-point arithmetic architecture for FPGAs", *International Journal of Reconfigurable Computing,* vol. 2011, Article ID 518602, 19 pages, 2011.

[67] M.S. Andrews, "Architectures for generalized 2d FIR filtering using separable filter structures", in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1999)*, Phoenix, AZ, March 1999.

[68] D. Llamocca, C. Carranza, M. Pattichis, "Separable FIR filtering in FPGA and GPU implementations: Energy, Performance, and Accuracy considerations", in *Proceedings of 2011 International Conference on Field Programmable Logic and Applications FPL'2011*, Chania, Greece, Sep. 2011.

[69] C. Tanougast, M. Janiaut, Y. Berviller, H. Rabah, S. Weber, and A Bouridane, "An embedded and Programmable System based FPGA for Real Time MPEG Stream Buffer Analysis", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 19, No. 2, Feb. 2009.

[70] K. Mei, N. Zheng, C. Huang, Y. Liu, and Q. Zeng, "VLSI design of a high-speed and area-efficient JPEG2000 decoder", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 17, No. 8, Aug. 2007.

[71] K. Varma, H. Damecharla, A. Bell, J. Carletta, and G. Back, "A fast JPEG2000 encoder that preserves coding efficiency: The split arithmetic encoder", *IEEE Trans. On Circuits and Systems – Part I: Regular Papers*, vol. 55, No. 11, pp. 3711-3722, Dec. 2008.

[72] N. Farrugia, F. Mamalet, S. Roux, F. Yang, and M. Paindavoine, "Fast and Robust Face Detection on a Parallel Optimized Architecture implemented on FPGA", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 19, No. 4, April 2009.

[73] D. Nguyen, D. Halupka, P. Aarabi, and A. Sheikholeslami, "Real-Time Face Detection al lip feature extraction using field programmable gate arrays", *IEEE Trans. On Systems, man, and Cybernetics*, vol. 36, No. 4, Aug. 2006.

[74] T. Komuro, T. Tabata, and M. Ishikawa, "A Reconfigurable embedded system for 1000 f/s Real-Time Vision", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 20, No. 4, April 2010.

[75] E. Oruklu, and J. Saniie, "Dynamically Reconfigurable Architecture design for Ultrasonic Imaging", *IEEE Trans. On Instrumentation and Measurement*, vol. 58, No. 8, Aug. 2009.

[76] Y. Liu, and E. M-K Lai, "Design and Implementation of an RNS-Based 2-D DWT Processor", *IEEE Trans. On Consumer Electronics*, Vol. 50, No. 1, Feb. 2004.

[77] H. Hedberg, F. Kristensen, and V. Öwall, "Low-complexity binary morphology architectures with flat rectangular structural elements", *IEEE Trans. On Circuits and Systems – Part I: Regular Papers*, vol. 55, No. 8, pp. 2216-2225, Aug. 2008.

[78] H. S. Neoh, A. Hazanchuk, "Adaptive edge detection for real-time video processing using FPGAs", in *Proc.GSPx2004 Conference*, 2004.

[79] C.-S. Bouganis, S.-B. Park, G.A. Constantinides, and P.Y.K. Cheung, "Synthesis and optimization of 2D filter designs for heterogeneous FPGAs", *ACM Trans. Reconfigurable Technol. Syst*. Vol. 1, no. 4, p. 24, Jan. 2009.

[80] H. Yang, F. Zhang, J. Lai, and Y. Wang, "Image Filtering using Partially and Dynamically Reconfiguration" in *Proc. 2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology*, Shanghai, China, 2010, pp.2067.

[81] S.U. Bhandari, S. Subbaraman, S.S. Pujari, R. Mahajan, "Real Time Video Processing on FPGA using on the fly Partial Reconfiguration", in *Proceedings of the 2009 International Conference on Signal Processing Systems*, pp. 244, Singapore,, May 2009.

[82] T. Raikovich, Feher, B. "Application of partial reconfiguration of FPGAs in image processing", in *Proceedings of 2010 Conference on Ph.D. Research in Microelectronics and Electronics*, Berlin, Germany, July 2010.

[83] S. Hong, J. Lee, A. Athalye, P. Djuric, W. Cho, "Design Methodology for Domain Specific Parameterizable Particle Filter Realizations", *IEEE Trans. On Circuits and Systems – Part I: Regular Papers*, vol. 54, No. 9, pp. 1987-2000, Sep. 2007.

[84] Y. Hori, A. Satoh, H. Sakane, K. Toda, "Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems", in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'08)*, pp.23-28, Heidelberg, Germany, Sept. 2008.

[85] J. Resano et al., "Runtime minimization of Reconfiguration Overhead in Dynamically Reconfigurable Systems", in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'03)*. Ser. LNCS, vol. 2778. Springer Verlag, Sept. 2003, pp. 585-594.

[86] Partial Reconfiguration User Guide for ISE 12.3 (UG702), Xilinx, San Jose, CA, v12.3 edition, Oct. 2010.

[87] Virtex-6 Family Overview (DS150), Xilinx, San Jose, CA, v2.2 edition, Jan. 2010.

[88] C. Bodda, "Introduction to Reconfigurable Computing", Springer, ISBN 978-1-4020-6088-5, 2007.

[89] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays", *Proceedings of the IEEE*, vol. 81, pp. 1013-1029, July 1993.