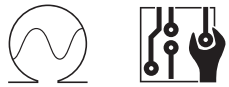
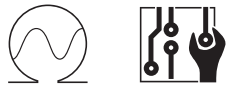


Technische Universität München
Department of Electrical Engineering and Information Technology
Institute for Electronic Design Automation

Performance Analysis and System Prototyping Automation on ZedBoard FPGA Board

Master Thesis

M. Benjamin Bordes



Technische Universität München
Department of Electrical Engineering and Information Technology
Institute for Electronic Design Automation

Performance Analysis and System Prototyping Automation on ZedBoard FPGA Board

Master Thesis

M. Benjamin Bordes

Supervisor :	M.Sc. Munish Jassi
Supervising Professor :	Prof. Dr. Ulf Schlichtmann
Topic issued :	01.03.2014
Date of submission :	10.11.2014

M. Benjamin Bordes
Knorrstrasse 2
80807 Munich

Abstract

Designing a System-on-Chip (SoC) made of many processing elements is a complex task. Increasing complexities of today's SoC have made the system architects to increasingly reuse the intellectual properties (IP) in system design. These IPs are either available from the previous projects or are provided by the third party IP suppliers as IP-libraries. When developing a software application for a target SoC, every sub-task can be executed by a processor as software implementation or by a dedicated hardware acceleration module. Determining which sub-task should be accelerated involves a significant exploration effort slowing the system development. Moreover, a non-optimal integration of multiple accelerators can increase the communication load on the associated bus, hence leading to a potential bus bottleneck. Shortening the time required between a system description and its optimal implementation is a key to support design reuse. This Thesis contributes to the establishment of an automated design exploration and performance evaluation framework based on the Xilinx tool-chain. This Thesis includes three major contributions. The first task is to automate the hardware and software design flow using the Xilinx tool-chain to avoid the manual interventions. The second task is to address the potential communication congestion arising from the hardware acceleration of tasks by monitoring the AXI bus system. A traffic generator and monitor cores are used to stress the bus system up to its limit and estimate the maximum sustainable traffic on the bus. The bus load estimation provides a prior criterion for the decision to add an additional hardware accelerator module on the bus or not. Finally, the Design Space Exploration (DSE) based on an IP-library for a targeted video-processing application is done. DSE involves the exhaustive search of all feasible architectures which can be achieved by using a chosen IP-library. The system architectures hence obtained vary based upon the computation control and the handling of data communication. In this DSE, thirteen different configurations with a varying number of hardware accelerators integrated in either pipelined or non-pipelined are obtained. Further, all these architectures are prototyped on an FPGA and analyzed to observe the trade-off between design performances and resource utilizations. The CPU load, bus utilization, area utilization and the power consumption of the chip are chosen as design metrics. Mapping the sub-tasks on hardware considerably increases the FPGA area utilization and the data traffic, while pipelining relieves it by some extent but with a penalty of less control on the data flow. The increase in the area utilization results in a higher chip power consumption. From a pure software to a pure hardware implementation, the sustainable frame rate increases from 3 to more than 100 with a growth in the area utilization by 39%. For the pipelined architecture, the frame rate reaches 300 with an 29% area utilization increase.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Overview

Since the conception of the first two microcontrollers - the Texas Instrument TMS 100 and Intel 4004 in 1971 - requirements of computational systems have driven the microcontroller's complexity and performance to the top. But a processor is not an optimal solution for the tasks that could be parallelize in hardware. An FPGA on the contrary can implement dedicated hardware modules that accelerate a system. Moreover, as it can be programmed to realize any custom logic function, it represents an opportunity to prototype a system where some processing tasks are accelerated. Today's System-on-Chip (SoC) leverage processors and FPGA fabrics to offer cutting edge system design prototyping capabilities in a single chip. A SoC includes one or several hard processor cores, an FPGA, a memory system, numerous interconnects and fast Input/Output (I/O) connectivities. Processors can either run an embedded Operating System (OS) or work bare metal. The large available area inside today's FPGAs allows implementing dozens or hundreds of functional blocks. Moreover those modules tend to be highly flexible and to offer numerous optimization parameters.

Many semiconductor companies supply a large variety of SoC platforms and hardware module descriptions which can be implemented on the FPGA. Those descriptions - namely Intellectual Property (IP) - are integrated as design libraries inside design software. Alongside those proprietary and platform optimized IP blocks, other companies propose third party IP modules from which a powerful SoC can also be designed. The availability of already designed and tested system blocks has triggered the emergence of the "Design and Reuse" methodology where the same design description is reused over multiple projects. As a result, significant time is saved during the system's design phase, but at the cost of a higher block interfacing complexity. Therefore more and more IP are used in today's digital systems and this tendency is expected to carry on, according to the International Technology Roadmap for Semiconductors 2011 study (ITRS 2011). The percentage of design reuse in modern system already achieved 54% in 2011 and is foreseen to reach 92% by 2020, calling for research towards IP integration tools.

The two main FPGA providers - Xilinx and Altera - represented in 2012 respectively 47% and 41% of the total FPGA market (Tech411 2013). Both companies have a long time experience with programmable logic devices, a large portfolio and community users, as well as established software tool chains. In March 2011 Xilinx released the first SoC backed up by a dual ARM Cortex-A9. In July 2012 the electronic manufacturer Avnet Electronics launched the ZedBoard platform featuring a Xilinx Zynq and offering an affordable SoC for prototyping purposes. Once a chip is selected, the main concern for SoC designers is to yield the maximum performance for the given application. The overall implementation should take advantage of

the task execution acceleration inside the FPGA logic and the good handling of unpredictability by the processor. While the hardware accelerators focuses on the processing intensive tasks, the processor can concentrate on some other tasks, including the configuration and control of the accelerators. The selection between a hardware or software platform for an algorithm implementation is based on a process called Hardware/Software Co-design. It involves comparing the performances of the different hardware mapped tasks with their software mapped counterparts.

The GRammar-based IP Integration Platform (GRIP) is a project in development at the institute for Electronic Design Automation at the Technische Universität München targeting digital platform development. It aims to facilitate digital system design by providing an automated IP integration and DSE engine. It is designed as an iterative system architecture exploration process, based on a set of IP-integration rules. Those rules based on generative graph grammars contain the required information to determine the design space associated with the design. Figure 1.1 presents the functioning of the GRIP. It only requires two main inputs: a high level description of the system in the IP-XACT format and a set of design space exploration rules. The GRIP is able to generate an optimized IP module from a C/C++ code. The performance of this module implemented inside the FPGA can then be compared with its equivalent C/C++ code running on the processor. From a pure software implementation of a complex processing flow, the GRIP takes step by step decision on the mapping of the tasks to reach the best possible system configuration. The ZedBoard was selected as initial target platform for the GRIP framework.

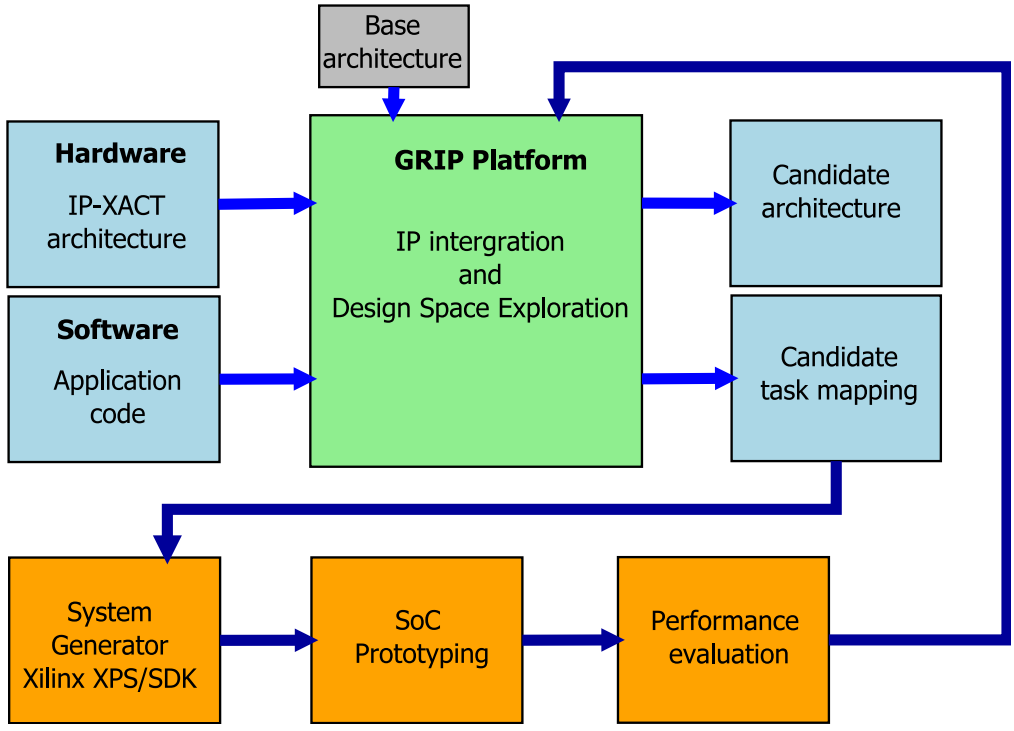


Figure 1.1.: The GRIP platform. The focus of this Thesis is represented in orange.

IP-XACT is an XML established standard (IEEE-1685) providing language- and vendor-neutral IP descriptions. It provides a high-level XML description of a system and its components, which eases verification and design via information extraction. Due to its high-level of abstraction, the effort for a designer to describe his target system with IP-XACT is much lower than that would be a traditional fine description. The provided high-level model of the system is converted by the GRIP into a lower-level description that can be used as input for code generation. After code generation, the system can be implemented on a target platform where a performance evaluation of the design can be carried out. The results of this analysis are then used by the GRIP to consider and explore both design and parametric alternatives. Automatically and iteratively performed, this flow cut down user intervention traditionally involved with DSE. This Master Thesis contributes to a subset of the aforementioned tasks. Firstly, the conversion from IP-XACT to a lower-level description is out of the scope of this work. Secondly, considering that the potential parameters and architecture modifications of a system are dependent on the given application, the work presented here relates to the identification of system modifications that could improve system performances for a desired image processing application. This Thesis proposes a first scope for the design space exploration but doesn't implement the decision-making engine. Finally, the chosen video-processing application reuses the work presented in (Razzaq 2013).

1.2. Video-processing Application Case Study

Embedded computer vision systems have long became part of the everyday life. Parking assistance technology, augmented reality or airport surveillance systems are just few examples out of many. It is a high performance demanding sector coerced by the same factor as most embedded systems: cost, power and size. For this reason, the low-cost Zynq Z-7020 is a perfect solution for low-end embedded system implementing computer vision algorithms. While simple processing functions require only a few operation per pixel, more complex video-processing are often algorithmic-intensive. In order to analyze and propose optimal system architectures for video-processing applications, a computer vision system made of three common image-processing functions is selected. Those functions are selected based on their different complexity and broad utilization in a large variety of applications. The chosen image processing primitives are the grayscale conversion, erosion and edge detection.

- Grayscale conversion is a low complexity and single pixel operation where the Red, Green and Blue color values representing the colored pixel are averaged into a single grayscale intensity value.

- Erosion is a morphological operator which erodes the boundaries of groups of pixels inside an image. It applies a 2D 3x3 mask on the image and replaces the value of the center pixel with the minimum value of the neighborhood. It involves a higher complexity than the grayscale conversion.

- Sobel filter is an edge detection algorithm based on two 2D 3x3 masks which compute the gradients in the horizontal and vertical directions of every neighborhood in the image.

The center pixel is then replaced by a combination of the two computed gradients. The processing complexity of this function lies in between the two previous ones.

The processing functions are performed successively on the same input frame to model a standard multiple-steps processing application. A camera provides the input data to the system where it is written to the external memory by the CPU. Then the input frame (Figure 1.2a) is successively passed through the Sobel filter (Figure 1.2b), Erosion operator (Figure 1.2c) and Grayscale conversion (Figure 1.2d) before being written back to the memory. Every function is either performed in software by the CPU or in hardware by a dedicated hardware accelerator. The processed image is then displayed on an external HDMI screen. The system is shown in Figure 1.3.

(a) The original frame from the camera

(b) The frame after the Sobel edge-detection

(c) The frame after the image erosion

(d) The frame after the grayscale conversion

Figure 1.2.: Frame processing

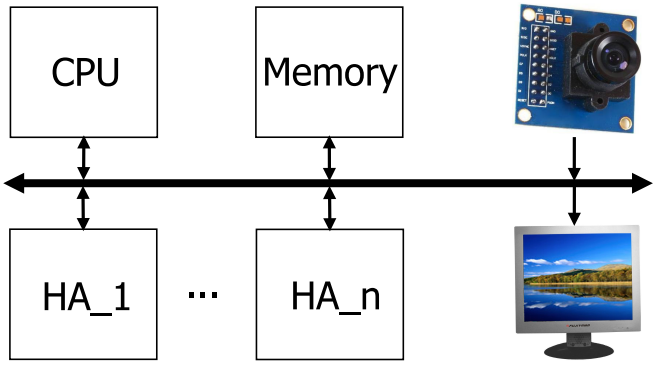


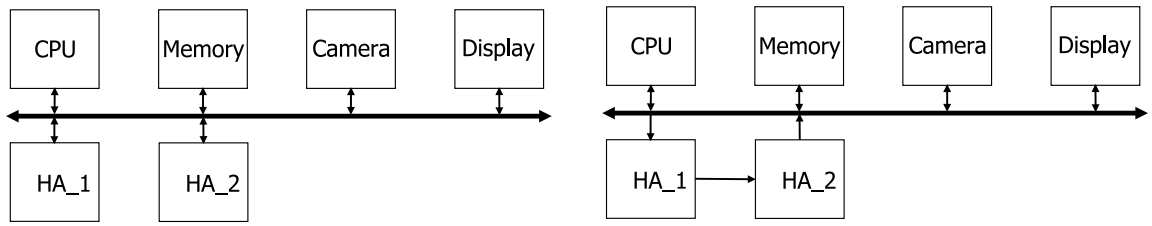
Figure 1.3.: System overview

1.3. Problem Description

Designing a SoC imposes a considerable exploration time to implement, monitor and reach an acceptable performance-utilization tradeoff inside the final system. Reducing this time would boost productivity and improve products time-to-market. The main time-limiting factor is the amount of user intervention required in tools along with the design. However digital system design relies on Graphical User Interface (GUI)-based software that are built on user intervention. It helps new users to easily get started with the program but it suffers from a poor return when it comes to repetitive tasks and minor system modifications. To overcome this issue, software generally also offer a command prompt interface which can be used for automation purposes. Automatizing the designer tasks associated with the Design Space Exploration (DSE) using scripts would then be a valuable improvement.

For image processing applications, IP blocks usually process a complete frame of data that was previously stored in memory. The input frame originates from a camera module acting as a data source. The processed frame is written to the memory, from which a monitor may display it or other processing steps may be performed. Accordingly, the basic solution when multiple processing modules are used is to feed each of them from the memory and store the processed frame back to the memory. The camera, the memory and the processing devices are connected over a group of buses through which the data exchanges occur. However, if a large number of IPs are connected on the same bus and transfer a high amount of data, congestion issues might occur. Thus, reaching a certain point, adding an extra device on a bus is suboptimal. Estimating how many cores can be connected on the same bus would help to determine when an extra bus is required.

A design alternative for data exchange between cores is to plug IPs one after the other so that the output of the first core is used as input for the next one, and so on. This design strategy, called "pipelining", reduces the amount of traffic on the bus and represent a design alternative. Figure 1.4 illustrates the two different system architectures. From the system in Figure 1.4a, the two Hardware Accelerators (HA) are pipelined, generating the system in Figure 1.4b.



(a) The original system without pipeline

(b) The modified system using two pipelined modules

Figure 1.4.: Pipelining two hardware accelerators

This leads us to consider two sources of design exploration:

- Different Hardware/Software task-mappings.
- The pipelining of hardware accelerators.

Automatizing the complete flow of designing, synthesizing and evaluating system architectures would alleviate the designer tasks and support the increase in design reuse.

1.4. Outlines

The main task of this Master Thesis is to identify and set up a high level system design methodology and analysis based on Xilinx SoC design tools so that complete system architectures can be synthesized and analyzed without any user intervention. Chapter 2 outlines relevant information regarding the target ZedBoard and communication requirements inside modern SoC platforms. The first goal is to organize a fully-automated system synthesis and interface among relevant Xilinx tools, leveraging existing support between them. This will be presented in chapter 3. The second objective of this work addresses the performance evaluation of the design. The selected performance parameters are the processor and interconnect load, the FPGA resource utilization and the power consumption. Estimating the bus load is of particular importance for FPGA acceleration because it determines if an extra hardware accelerator can be added on a bus, or not. This will be developed in chapter 4. The third goal is to apply the developed flow to an image-processing system based on three processing steps, involving a Sobel Edge detection filter, Erode filter and Grayscale conversion. While the different image-processing algorithms are subject to HW/SW Codesign, power optimization techniques are also considered. In order to select the best system architecture candidate, the performance criteria are monitored and compared over all the potential architectures. This will be presented in chapter 5.

2. Background

This chapter firstly presents the Zynq device and ZedBoard platform. Then it gives an overview of today's SoC platform main components. It finally describes data communication within the framework of the AXI standardized interface.

2.1. Zynq Functional Blocks

The ZedBoard development platform shown in Figure 2.1 features a powerful Xilinx Zynq Z-7020 Soc including a dual ARM Cortex-A9 core and an Artix-7 Xilinx FPGA. The processor is traditionally referred to as the Processing System (PS) while the FPGA is the Processing Logic (PL). The Artix-7 is Xilinx most power efficient FPGA from its recent 7 Series FPGA family. The two other family members - the Kintex-7 and Virtex-7 - offer higher performances at the cost of higher power requirements. The Zynq benefits from the TSMC 28 nm high-k metal gate (HKMG), high performance and low-power process to provide considerable performances in machine vision, network, automotive and medical sectors (Xilinx 2013b). Its performances originate in the specific clustering of various functional modules.

The ARM Cortex-A9 is a popular high performance and power constrained processor based on the ARMv7-A Cortex architecture. It supports the ARM, Thumb2 and Jazelle Instruction Set Architecture (ISA). It includes an optimized first level of Cache and an optional level 2. Inside the Zynq, each ARM core benefits from a 32 KB Data and 32 KB Instruction 4-way set-associative L1 and a 512 KB unified 8-way set-associative L2 cache. Its memory coherency is taken care of by a Snoop Control Unit (SCU). Optionally a Floating-Point-Unit (FPU) backed

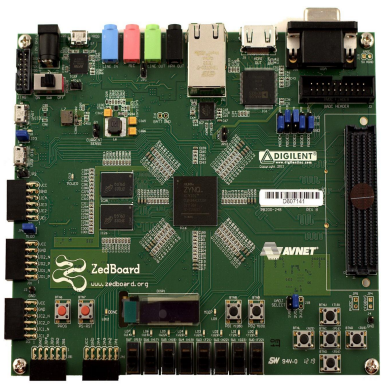


Figure 2.1.: The AVNET ZedBoard SoC platform

by a NEON Data Engine can be implemented to operate with floating-point instructions and benefit from the NEON advanced SIMD ISA.

A Xilinx FPGA is made of a columnar arrangement of functional blocks which connect to each other through a routing network. At the heart of it, the configurable logic blocks (CLB) are the main logic resources for implementing both sequential and combinatorial circuits inside an FPGA. Hardware DSP, memory and mixed-signal solutions are also integrated as dedicated columns into the chip. A CLB is made of two slices arranged in two separated column, as depicted in Figure 2.2. Arithmetic carry logics connect slices vertically for computation purposes. As represented in Figure 2.3, a slice integrates four 6-input Look-Up Tables (LUT), their associated 8 Flip-Flops (FF) and multiplexers (Xilinx 2014a).

The aforementioned Xilinx new FPGA are built upon the 7 series CLB which re-uses what has proven to provide high-performance in the previous Virtex-6 FPGA family:

- Real 6-input LUT (LUT6) Technology.
- Dual 5-input LUT (LUT5) option.
- Distributed Memory and Shift Register Logic (SRL) capability.
- Dedicated high-speed carry logic to improve the performance of arithmetic functions.
- Wide multiplexers for efficient utilization.

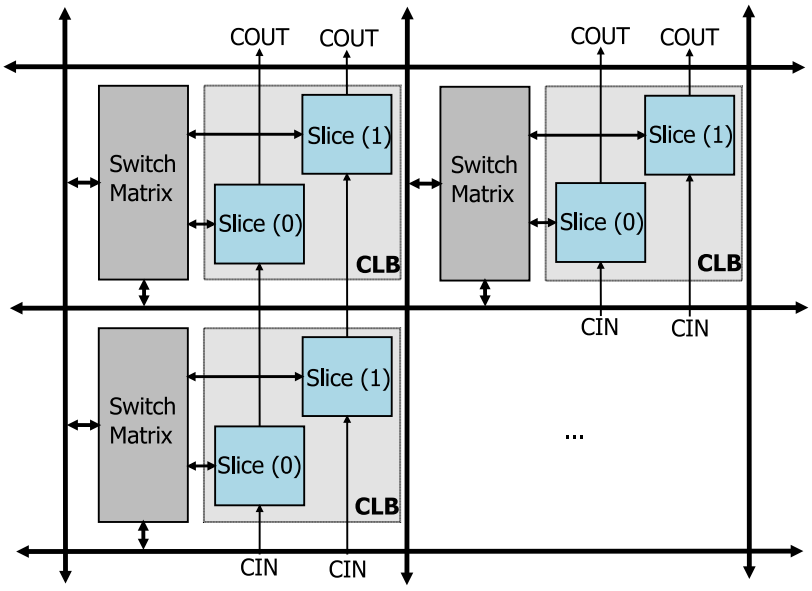


Figure 2.2.: CLBs and slices interconnection within Xilinx FPGA

A LUT can either be configured as a single LUT6 or two LUT5. When two independent LUT5 are used, the availability of two dedicated registers per LUT allows for registering the output inside the same slice. It reduces the spanning of slice utilization. Slices are separated between two categories:

- SLICEL.
- SLICEM.

While a SLICEL only provides logic, arithmetic and ROM memory functions, any LUT inside a SLICEM can also be used to store data - then is namely used as distributed RAM - or to shift data. In these purposes it can be configured as a distributed 64-bit wide RAM, 32-bit wide shift register (SRL32) or as two 16-bit wide shift registers (SRL16). In a distributed RAM the write port is synchronous whereas the read port is asynchronous. Turning the read port into a synchronous port comes at the cost of a flip-flop in the same slice. If lager amount of data or delays are required, then multiple LUTs inside a SLICEM can be combined. SLICEM account for around one third of the total amount of slices. A CLB can only contain a SLICEL and a SLICEM or two SLICEL.

A LUT can be configured to implement a 64x1 ROM memory. Optionally up to four FFs per slice can be configured as level-sensitive latches. Xilinx recommends benefiting from the

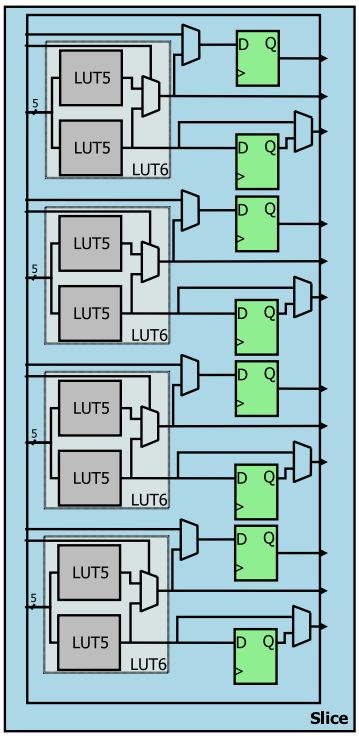


Figure 2.3.: The inner structure of a slice

abundance of FF to improve design performance in implementing pipelines. FFs inside a slice are not totally independent because they share the same control signals clock (CLK), clock enable (CE) and set/reset (SR). It is recommended to foster the implementation of small storage arrays and shift registers inside LUTs rather than FFs to minimize the power consumption and optimize the resource utilization. In this purpose the amount of control signals should be kept minimal, so that multiple FFs inside the same slice can be used for different purposes. The multiple design algorithms implemented in the Xilinx tool chain optimize the use of the FPGA resources according to a user configurable implementation run time or strategy. However the routing process also has the possibility to intentionally increase the area utilization to optimize the delay. Indeed some slices are commonly traversed to optimize the routing path between other slices.

Besides distributed RAM capabilities, an FPGA also integrates block RAM. While distributed RAM efficiently store very small arrays, block RAM are more adapted for larger arrays. A distributed RAM is for example the best choice for memories consisting at most of 64-bits. Indeed a distributed RAM has better performances and smaller power consumption than a block RAM but is available in smaller amount. The Xilinx 7 Series FPGA block RAM is a dual (Write and Read) synchronous port with a memory storage per block of up to 36 Kb of data and independent data width for the two ports. It can also be used as two independent 18Kb blocks or configured into a simple dual-port (SDP) mode to double the data width.

Xilinx 7 Series FPGAs integrate DSP dedicated hardware slices to efficiently implement digital signal processing (DSP) application. DSP slices are grouped by two inside a tile, whose height matches with a block RAM's or five CLBs'. Figure 2.4 presents the simplified structure of a DSP48E1. Those DSP48E1 flexible functional blocks accelerate, improve and reduce the power consumption of a digital signal processing application through the support of many functions. It includes barrel shifter, multiply, multiply accumulate, multiply add, three-input add, wide-bus multiplexing, magnitude comparator, bitwise logic functions, pattern detection and wide counter features. Those functions are possible via the integration of hardware 25 x 18 two's-complement multipliers, 48-bit wide accumulators, power-saving pre-adders, single-instruction-multiple-data (SIMD) arithmetic units and pattern detectors inside the slices.

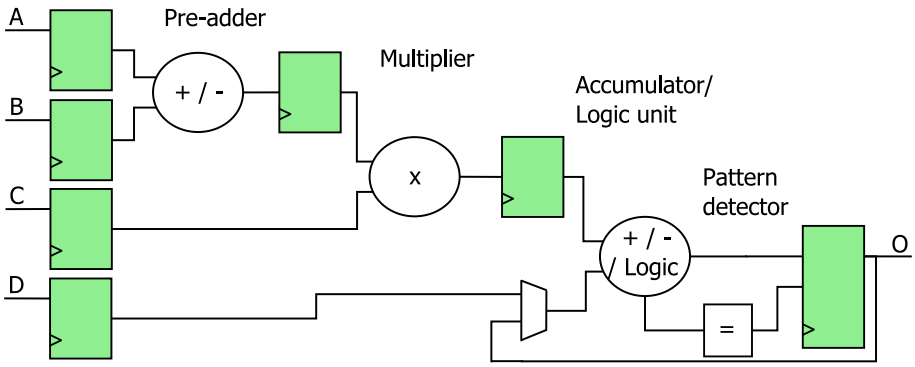


Figure 2.4.: Simplified view of a Digital Signal Processor inside the Zynq device

The CLG 484 selected package for the ZedBoard offers up to 53200 Look-Up Tables (LUTs), 106400 Flip-Flops, 560 KB (140*36Kb) of block RAM, an on-chip boot ROM, a 256 KB On-Chip RAM Memory (OCM) and 220 (18*25 MACCs) DSP slices. It also integrates multiple I/O peripherals and interfaces including USB, CAN, SPI, UARTS, I2C and two 12-bit wide Analog-to-Digital-Converters (ADC).

2.2. The Zynq Memory Architecture

ZedBoard applications are backed by a large external 512 MB Double Data Rate 3rd generation (DDR3) RAM memory. The Access to this external memory has a higher latency compare to other memories in the system. It is able to transfer data at twice the rate using both rising and falling edges. Thus, clocked at 533 MHz it reaches a data rate of 1066 Mb/s. DDR3 memories have an average 30% reduced power consumption compared to the previous DDR2 generation due to a reduced supply voltage (1.5V compared to 1.8V). This DDR RAM is organized as a collection of memory pages or "rows" which can be accessed by 8 registers called memory banks. Each bank dedicates to one memory access and opens the required page. If a bank already contains an open page and is ready to service a request on the same page, then the access is faster. This is known as a page-hit. Its contrary is a page-miss. The Zynq external memory is accessed via a multi-ported DDR memory controller. The Multi-Ported Memory Controller (MPMC) is a combination of an AXI memory controller core and an AXI interconnect which provides a fast and shared interface to a common memory. In the case of the Zynq, the MPMC is already configured and the number of connected slaves is fixed. As presented in Figure 2.5 the DDR controller includes four AXI slave ports to provide a shared access to the memory for the PL and the PS:

- Two 64-bit wide ports for the PL access.
- One 64-bit wide port dedicated for the CPU(s) via the L2 cache controller.
- One 64-bit wide for the central interconnect to which other AXI masters connect.

Each of the two MPMC ports accessible from the PL is shared by a pair of AXI High Performance (HP) ports. The AXI HP0 and HP1 ports, respectively HP2 and HP3 ports, can only access the memory through the slave port S3, respectively S2. Those HP ports integrate buffering FIFO to smooth high latency transfers. The bandwidth for a given AXI_HP port depends on two factors:

- Its frequency.
- The occupied bandwidth on the other port from the same pair.

Considering an AXI_HP port clocked at 143 MHz, the maximum (read + write) data bandwidth is 1144 MBytes per second (MB/s). While the 32-bits wide 1066 Mb/s DDR3 memory has a total (read + write) available bandwidth of 4264 MB/s, the 64-bits wide 356 MHz memory

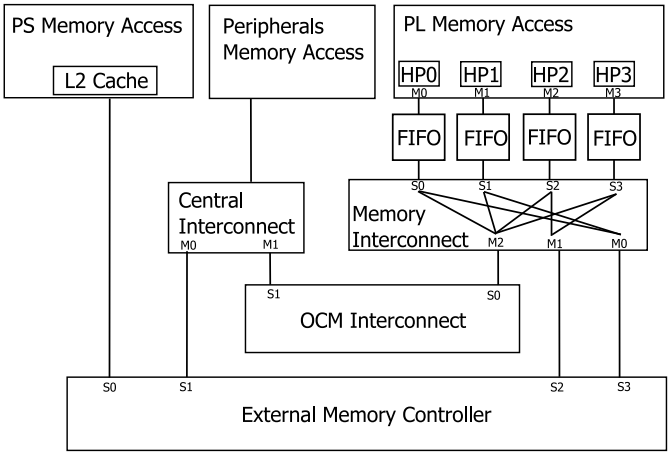


Figure 2.5.: Memory connectivity among the Zynq device

interconnect has a total (read + write) bandwidth of 5696 MB/s. Thus the four AXI HP ports aggregated bandwidth is limited to the same value. It means that the full utilization of the external memory by the PL is possible. So the maximum aggregated bandwidth for two HP ports of a different pair of ports accessing the external memory is limited to 4264 MB/s. A single HP port is not able to fully utilize the DDR memory.

The MPMC optimizes memory accesses via two features:

- A transaction scheduler to optimize the data bandwidth and latency.
- An optional re-ordering engine to maximize the efficiency of memory access.

The re-ordering engine reorders reads to minimize page misses and maximize accesses to open rows. In addition, it groups reads and writes together to minimize the bus turnaround time. General memory accesses are expected to achieve more than 75% efficiency. While random read and write reach 80% average efficiency, continuous read and write reach 90% efficiency. Moreover, moderate length burst accesses don't heavily damaged the efficiency of external DDR accesses. Especially the efficiency remains around 90% with a burst length in the 4 to 16 range. Table 2.1 summarizes the theoretical maximum data bandwidths.

Interface	Bus Width (bits)	Clock (MHz)	Read Bandwidth (MB/s)	Write Bandwidth (MB/s)	Number of Interfaces	Total Bandwidth (MB/s)
AXI_HP Port	64	143	1144	1144	4	9152
Memory Interconnect	64	356	2848	2848	1	5696
external Memory	32	533	4264	4264	1	4264

Table 2.1.: Zynq theoretical bandwidth

2.3. The AXI Protocol

Concurrently to the release of the Spartan-6 and Virtex-6 in 2009 targeting an easy interface with ARM cores, Xilinx introduced in 2010 its IP plug-and-play strategy to accelerate productivity and boost design reuse. It relies on the AMBA AXI4 interconnect protocol jointly developed with ARM. Due to the new largely adopted standard interface, the interoperability of IP cores has never been so high. Therefore the effort to build complex system made of numerous IP cores has been substantially reduced. The AXI4 is the second version of the Advanced eXtensible Interface (AXI) protocol, named AXI3 and initially introduced in 2003 by ARM to supplement his AMBA protocol family released in 1996. The AXI4 interface is available in three different versions:

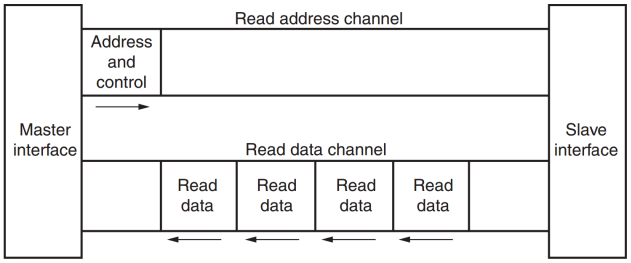
- AXI4 for high-performance communication.
- AXI4-Lite dedicated for simpler and lower throughput requirements than AXI4 such as simple core configuration for example.
- AXI4-Stream for high-speed streaming data communication.

Any transaction according to the AXI3, AXI4 or AXI4-Lite protocols contains besides the data to be transferred, a target address within the system memory space. This makes them "memory-mapped" protocols as opposed to "data-streaming" protocols where the data to be transmitted is simply streamed in and out without any address consideration. The AXI4-Stream protocol is based on this data-streaming principle.

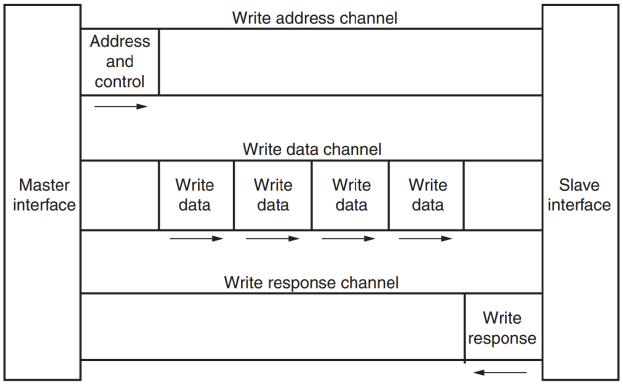
A memory-mapped transaction involves five different channels, enabling separate address, control and data phases:

- Read Address (RA) Channel.
- Write Address (WA) Channel.
- Read Data (R) Channel.
- Write Data (W) Channel.
- Write Response (B) Channel.

Read and write data information are exchanged according to Figure 2.6. The AXI protocol supports simultaneous transactions in both directions, as well as variable transfer sizes. In addition to burst based transactions, outstanding transactions and out of order completion are also supported. While the AXI3 only supports burst transactions up to 16 data transfers, the AXI4 supports up to 256 data transfers. The AXI-Lite exclusively supports one data transfer per transaction. The data width is configurable between 32 and 256 bits, except for the AXI4-Lite where only 32-bits is supported.



(a) Read data exchange according to the AXI protocol



(b) Write data exchange according to the AXI protocol

Figure 2.6.: The AXI data protocol (ARM 2013)

The AXI specification provides an easy and powerful solution to interface two IP cores by defining the communication between a single master and a single slave. To enable a multi-master and multi-slave communication, the IP cores are connected to an AXI-compliant interconnect which routes transactions back and forth between all the connected communication nodes. Figure 2.7 presents the inner structure of an AXI Interconnect. It is made of two hemispheres - interfacing respectively to the masters or to the slaves - connected over a traffic routing crossbar. Both hemispheres contain optional functional units which are generated based on the system requirements. Those functional units are responsible for an automatic protocol, clock-rate and data-width matching connection over the crossbar. A user selectable register pipelining and data path buffering capability completes the breadth of its interfacing functions.

Register slices can be independently introduced in the channels to improve system timing at the cost of one latency cycle. For the AXI4 protocol, the default setting inserts a 2-deep FIFO buffer on the W and R channels and a 1-stage pipeline register on the AW, AR and B channels of the master interface. The slave interface is not optimized. Datapath buffering improves the system throughput where the data rate of the interconnect differs from the data rate of an attached master or slave. This is case if the data-width or clock-rate needs to adjust with the interconnect settings. The insertion of FIFO buffers in the W, R, AW and AR channels is

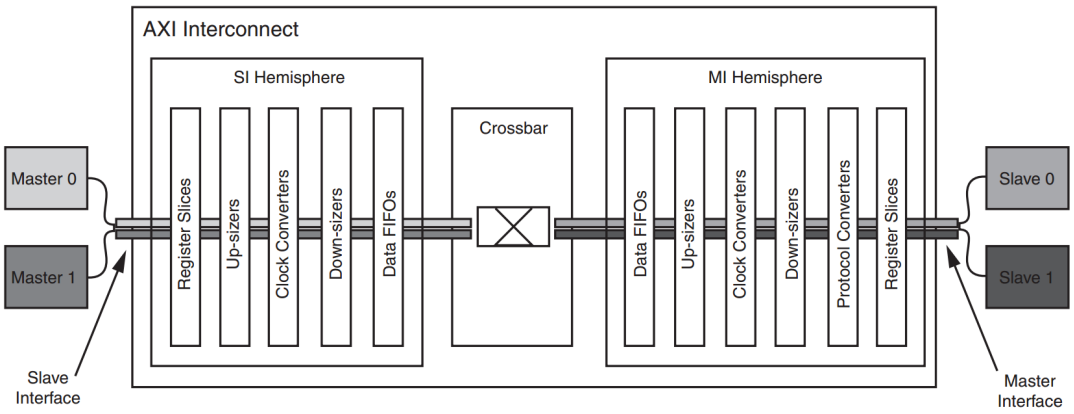


Figure 2.7.: The inner structure of a Xilinx AXI interconnect (Xilinx 2012c)

also fully-independent. For the W and R channels, the depth is configurable either as 32-deep LUT-RAM or 512-deep block-RAM. The width is automatically set to match the native data width of the transactions. On the AW and AR channels the FIFO buffer is 32-deep only.

2.4. Summary

In this chapter, the arrangement of functional blocks in modern SoC and the inner structure of FPGAs were introduced. It reveals to offer a high flexibility in the implementation of complex applications. Computational tasks can be distributed among CPU and FPGA logic to benefit from the best of the both worlds. Then the AXI protocol was introduced to better understand how communication means among on-chip devices have evolved to cope with a high level of integration.

3. Automating System Design and Optimization

This chapter firstly describes the main steps associated with the design of a Xilinx SoC using the GUI mode. It presents the multiple tools that are used and the different I/O files linked to them, as well as their location in the project hierarchy. Then, it explains how this flow can be automatized, leveraging some of the auto-generated files. Finally, it identifies interesting optimization options associated with the synthesis and mapping operations and describes how they can be integrated in the automated flow.

3.1. GUI based SoC Design

Xilinx SoC design on Zynq is supported by two software: ISE and VIVADO. The Integrated Software Environment (ISE) is Xilinx original software development product which is made of numerous tools. Released in 2012, Vivado is targeting future SoC release. Most of digital designer have been using ISE for previous devices generation, which explains why this flow is still popular today and why Xilinx has been maintaining it so far. ISE mainly relies on two tools:

- The Xilinx Platform Studio (XPS) for hardware design.
- The Software Development Kit (SDK) for software design and debug. SDK is Xilinx Integrated Design Environment (IDE) based on Eclipse 3.8.2 and CDT 8.1.2.

Both tools are part of ISE and are called from it upon request or automatically according to progress of the current project. Figure 3.1 gives an overview of the ISE design flow.

An XPS blank project starts with the specification of a Xilinx device: its architecture, device size, package and speed grade are selected among a list. If the Zynq platform is selected, the tool displays a four sub-windows based workspace whose main one on the top right part of the screen is made up of four tabs: Zynq, Bus interfaces, Ports and Addresses. The first one is exclusively dedicated to the configuration of the PS, whereas the other three are used for the PL configuration and the interfacing of the PS with the PL. In the Zynq tab the type of memory access, the clocks settings and the I/O peripherals connectivity are accessible. While the Bus Interfaces tab provides control on the system-interconnections, the Ports tab provides control over clocks and reset signals. Finally, the addresses tab is used for the address mapping of IP blocks used to interface with the CPU(s).

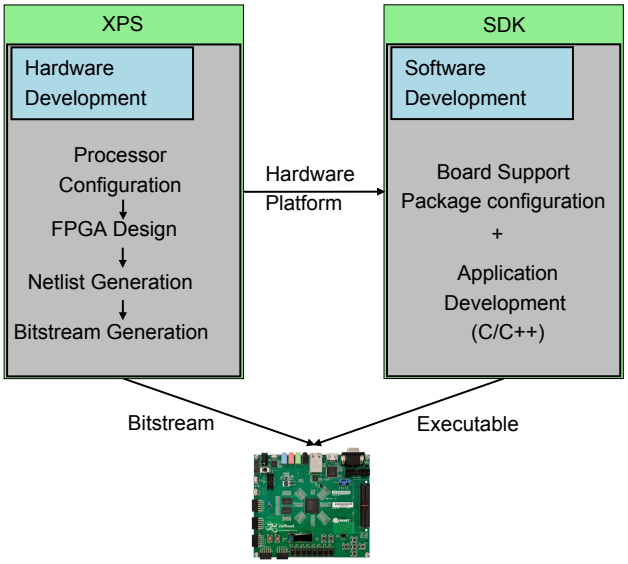


Figure 3.1.: The Xilinx ISE SoC design flow

The designer adds the required IPs to his project from the IP catalog or from a user repository, using a drag-and-drop feature. Every module has numerous degrees of parametrization through which the programmer optimizes the behavior of the system for the target application. This requires navigating among the multiple tabs for every module. When the complete hardware system is designed and the bitstream (BIT) file generated, the Xilinx Software Development Kit (SDK) can b invoked from XPS or standalone. The SDK requires three distinct projects:

- The "Hardware Platform" is a representation of the hardware system configuration needed for the software development. It is automatically included in the workspace.

- The "Application" is the software project to run on the platform. It is manually imported in the workspace by the designer.

- The "Software Platform", also known as a Board Support Package (BSP), acts as an interface between the hardware description in the Hardware Platform and the software project. It needs to be selected by the used among a list. A BSP is made of libraries, device drivers, file systems and interrupt handlers associated with the embedded processor system and its peripherals which constitute the first layer of the application stack. The default BSP known as "standalone" is a semi-hosted, and single-threaded environment providing only the simplest functionality such as input/output communication and access to the processor hardware features. More complex and OS-like ones are also available.

Afterwards the designer creates a build process configuration to store the compiler and linker settings. Then, once the compilation and linking of the application into an Executable Linked Format (ELF) has completed, the BIT and ELF files are respectively downloaded to the FPGA and the CPU from SDK. The system is now prototyped onto the FPGA.

3.2. Command Prompt based SoC Design

3.2.1. Flow Execution

Most of the GUI functionality is also accessible using the command prompt window available in both XPS and SDK or directly inside a bash shell. So the goal is to automatize the complete flow with a set of scripts. However, in both cases it is admitted that the project was first created using the GUI and that commands are later used on it. This means that all necessary initialization files and the project directories respect the tools requirements because they've been auto-generated. So the first task is to create the project with the proper settings. For this purpose, Xilinx Microprocessor Project (XMP) file is used to create an XPS project with the following TCL command:

```
1 # The XPS design script, part 1 #
2 xload xmp system.xmp
```

It is called inside the XPS tool in a no-window mode in order to initialize the internal tool database. If it is run in a folder which only contains the XMP file then it just creates empty data, etc, pcores and __xps folders and the system.log file. However, if the pcores folder is already present and filled with the user-defined module descriptions used in the MHS, then the project creation completes and those created folder contain the required files. At this moment the project is created and the command prompt interface can be used in a shell terminal.

Tools can be invoked using their name, followed by a set of options and input files (Xilinx 2013a). Figure 3.2 presents the implementation flow inside the XPS. The most important files associated with the design flow are presented in the section 3.2.2

- The Platform Generator (Platgen) tool is responsible for compiling the high-level description of the embedded processor system into an HDL netlist. This netlist integrates all the pcores instances described in the MHS file and incorporates every signal required to ensure a proper connection between them.

- The Xilinx Synthesis Technology (XST) tool is responsible for synthesizing each instantiated module -including the top level- and create Xilinx-specific netlist NGC files out of it. It performs the code syntax check, an analysis of the design hierarchy and functionality, as well as optimizing the design for the selected device architecture. The multiple NGC files are compiled into a single top-module NGC file before being passed to the next tool.

- The NGC file is then used in conjunction with a Block RAM Memory Map (BMM) and an User Constraints File (UCF) to generate a Xilinx Native Generic Database (NGD) via the NGD Builder (NGDBuild) . The NGD file contains besides the original system hierarchy from the NGC file, a logical description of the design in Xilinx primitives. The BMM file unifies the memory locations of individual block RAM into a contiguous logical data space. Once this conversion is done, the design is ready to be mapped on

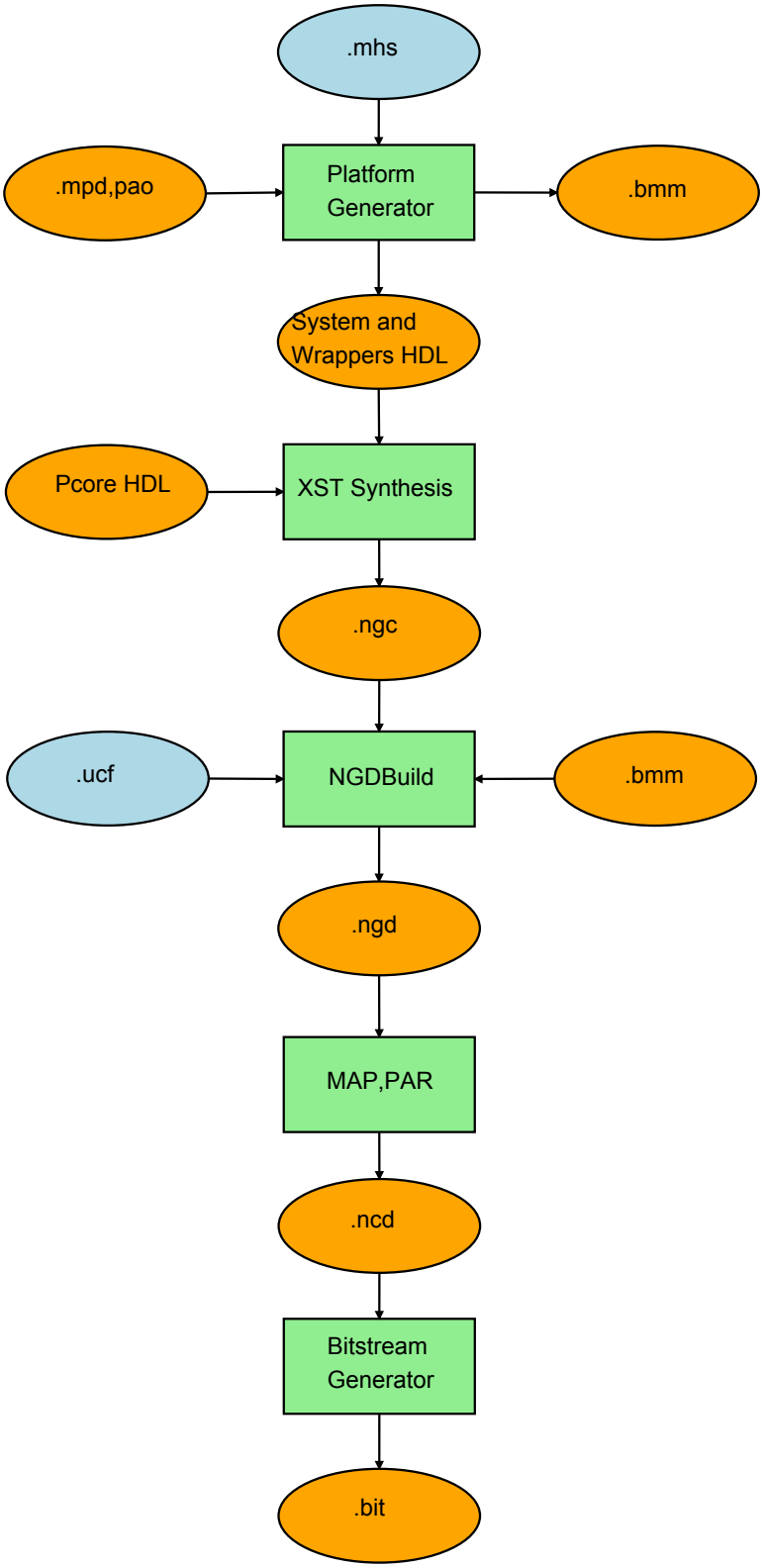


Figure 3.2.: The Xilinx XPS design flow

the platform.

- The Mapping (MAP) operation first checks the NGD file against logical design rule violation. It then maps the entire design to the target platform components (Slices, I/O cells, DSP, etc.) while partially using the information provided in the UCF file. Remaining constraints are stored in a Physical Constraints File (PCF) for later use. MAP produces a Native Circuit Description (NCD) file that defines the physical representation of the design based on the FPGA resources.

- The Placement and Routing (PAR) operation places and routes the previously mapped design described in the intermediate NCD file generated during the MAP process. It outputs a placed and routed NCD file required for the bitstream generation. PAR can be executed along two different approaches:

  – A timing-driven execution based on the timing constraints contained in the UCF file.

  – A cost-based execution ignoring any timing constraints and relying on cost-tables where factors such as available routing resources, constraints and length of connection are assigned weighted values.

  The MAP and PAR tools are grouped together in Figure 3.2 because the placement of the design can also be performed upstream during the MAP process. In this case, PAR only performs the routing; the MAP run time increases while the PAR one reduces. The extra information produced by the placement onto the FPGA increases the size of the NCD file generated.

- Once placed and routed the NCD completely defines the internal logic and interconnections of the FPGA. It is converted into the BIT file by the Bitstream Generator (BitGen).

Instead of calling the various tools successively, a Xilinx project contains an XPS related makefile system.make to automatically run the exact same commands that would be run when a button is clicked on the XPS GUI. It is a simple text file including several lists of commands, each one associated with a keyword. If one of the keywords is given as an option to the make command then the associated commands are executed. Makefiles are useful to describe the building of targets such as a library or an executable file. In that respect they integrate dependency rules which ensure the validity of the compiled targets. The earlier Xilinx project hierarchy only integrated a single makefile from which the software libraries generation, the BSP and ELF compilation were also accessible. However since version 13.4, two supplementary dedicated makefiles are used to compile the BSP and the software application in the SDK workspace. They will be discussed later. The XPS system.make makefile works in pairs with a system_incl.make where all the tools I/O files are listed and some default tool options are also written. Other tools options are defined in .opt files in the __xps project folder. The XPS makefile is automatically generated during the creation of a project and is re-generated every time the GUI is opened, because tools I/O may change. Because of the project creation in the no-window mode, the system.make and system_incl.make were generated in the project folder.

By itself the system.make does not contain any project-dependent information such as a path, so it could be used in different projects. But the system_incl.make is project-dependent and thus needs to be generated independently for every project. Among the available keyword in the system.make file, the following are of importance for us:

- bits: generate the bitstream after running implementation tools.
- exporttosdk: export the hardware description files and bitstream to SDK.
- clean: deletes all generated files.
- bitsclean: deletes the .bit, .ncd and bmm files from the project folder.

The makefile is used according to the following code snippet. The XPS project creation and the makefile calls form the *XPS design script*, which belongs to the set of four scripts presented in this chapter to automatize the system code generation.

```
# The XPS design script, part 2 #
make -f system.make bitsclean
make -f system.make bits
make -f system.make exporttosdk
```

The synthesis process has a large contribution to the total system generation time. Indeed, each and every module making the complete system is synthesized if the synthesis folder in the project directory is empty. However while exploring successive system architectures, some of the modules are not updated. Thus after the first system architecture was explored, some of the modules are already synthesized and they can be re-used. The synthesis folder from the first architecture can then be copied on the new architecture project directory. Only updated cores, or new cores will undergo the synthesis process. For the system modifications presented in the following, it reduced the overall synthesis time from more than an hour to less than thirty minutes. Using the "bitsclean" option rather than the "clean" option in the makefile call provides the advantage to preserve the synthesis folder in the project directory.

During the SDK tool invocation from the GUI, a set of files are automatically imported into the workspace and grouped together to form the hardware platform. The hardware platform configuration is contained in an XML representation which is used as input for the generation of the BSP. Multiple system initialization files ps7_init. (html, tcl, c and h) are also present. This needs to be recreated while using the makefile approach. Once the hardware platform has been exported to SDK using the XPS makefile, it is put into the SDK_Export folder. It then needs to be copied in the SDK_Workspace folder and integrated into the workspace. Then, the BSP is generated with the appguru tool in extracting the information contained in the system.xml file, according to the following code snippet:

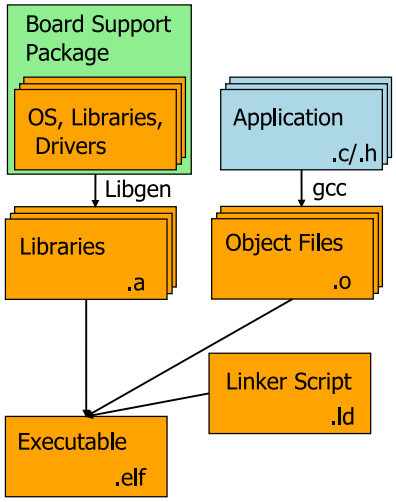


Figure 3.3.: The Xilinx SDK design flow

```
# The SDK design script, part 1 #
appguru \
 -hw $PRJDIR/SDK/SDK_Export/hw/system.xml \
 -app empty_application \
 -pe ps7_cortexa9_0 \
 -lp /nfs/tools/xilinx/ise/14.7/ISE_DS/ISE/data/zynqconfig/ps7_internals/pcores \
 -stdin ps7_uart_1 \
 -stdout ps7_uart_1 \
 -od  $BSPDIR \
 -lp ps7_cortexa9_0
```

This especially generates the associated SDK makefiles and the Microprocessor Software Specification (MSS) file which is then used by the Library Generator (LibGen) tool to build the BSP, according to the following code snippet:

```
# The SDK design script, part 2 #
libgen \
-hw $PRJDIR/SDK/SDK_Export/hw/system.xml \
-pe ps7_cortexa9_0 \
-log libgen.log \
-mhs $PRJDIR/system.mhs \
-lp $APPDIR/lib \
-lp $PRJDIR/pcores \
-p xc7z020clg484-1 \
 system.mss
```

Finally, the generated makefile is used to compile and link the application to generate the ELF. The compilation and linking of the applications into the executable is done by a platform-specific gcc/g++ compiler which is invoked from the makefile via the following command:

```
# The SDK design script, part 3 #
make -f system.make all
```

For the ARM cortex-A9 it runs the *arm-xilinx-eabi-gcc* compiler. After the compilation has ended, the linker combines the compiled applications with some selected libraries and a linker script to produce the executable file in ELF format, according to Figure 3.3. A linker script specifies the memory layout of the target processor and defines the location where each section of the application is placed in memory. The three previous code snippets form the *SDK design script*.

At this stage, the SoC implementation is possible. The bitstream as well as the ps7_init.tcl platform initialization files are available in the hardware platform folder, and the application is in the Debug folder of the application project. As presented in the following, a last script is written to establish the connection to the target and download the bitstream to the FPGA. Then, after connecting to the processor, it is reset and initialized before downloading the ELF. Finally, the application is started into the platform and the connection with the computer is disestablished. It forms the *Download script*.

```
# The Download script #
fpga -debugdevice devicenr 2 -f system.bit
connect arm hw
rst -processor
source ps7_init.tcl
ps7_init
init_user
dow app.elf
run
exit
```

Figure 3.4 depicts the complete script flow diagram. The *Setup script* is used to initialize the path to the input files. Then, the previously introduced scripts are successively started. A top level script is used to run them all.

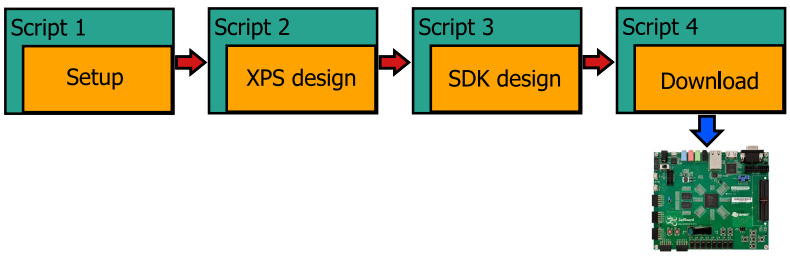


Figure 3.4.: The script based approach

3.2.2. Relevant System Files

- The Xilinx Microprocessor Project (XMP) file contains the Xilinx XPS project settings. It is a 30-line long text file which stores the architecture, device ,package and speed grade information, as well as some default data file paths.

- The Microprocessor Hardware Specification (MHS) file is a high-level description of the system created in XPS which is automatically updated during the system design. It can also be manually updated. This file contains all the relevant information regarding the processor, its peripherals, the instantiated hardware modules and its interconnection via buses. As represented in Figure 3.5, every hardware module is described by three assignment commands:

  – Parameters which set non-default values to specified module parameters.

  – Bus interfaces which define grouping of related interconnecting signals.

  – Ports which define special connectivities such as clock, reset or interrupt signals.

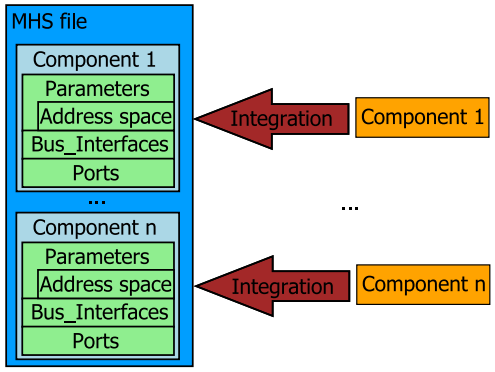


Figure 3.5.: The MHS file

The MHS file is the first input file in the hardware implementation flow and the lower-level system representation which is generated by the GRIP based on the IP-XACT description provided by the designer.

- The User Constraints File (UCF) specifies optional constrains on the logical design, as well as the mapping of I/Os in the device. Likewise the MHS file, the UCF needs to be generated from the GRIP, based on the application and the selected platform.

- The Microprocessor Software Specification (MSS) file is the principal description of the software platform. It contains directives for adapting optional operating systems, libraries and drivers.

- The XML hardware platform configuration is a detailed version of the MHS file. While an AXI bus connection is declared with a single-line bus interface assignment in the MHS file, every field of the AXI interface is declared and assigned a value in the XML file.

3.3. SoC Building Flow Improvement and Design Optimization

Up to this point, Xilinx tools have been used with the default settings associated with the buttons on the GUI. However, the hardware generation flow can be optimized during two main processes:

- Synthesis.
- Mapping.

Those optimizations have an impact on the area utilization, the timing and the power consumption of the design. Their efficiency depends on the considered design but their integration inside the previous flow would ensure the best performance for the system or a second scope of design performance exploration. In order to demonstrate it, the decision is taken to explore a power reduction of the design.

The XST synthesis tool has two available power reduction options:

- The power reduction flag which is used to minimize the number of simultaneously active block RAM ports in a design. Its impact is thus limited by the number of block RAM present in the design. This option is by default disabled as it might also affect the speed and area.
- The register balancing feature which aims at improving the timing performance of the system by enabling a flip-flop retiming algorithm during the synthesis process. It consists in moving flip-flops and latches across logic elements to enhance the maximum clock frequency of the design, as illustrated in Figure 3.6. As a consequence of register retiming, the length of all timing paths in combinatorial logic is balanced. It thus reduces glitching propagation which is dependent on the longest timing paths of the design. The longer the paths, the better the potential outcomes from this option. The more already optimized design, the fewer effects from it.

Changing XST synthesis options from the GUI in ISE is straightforward. The designer selects it from the XST process properties window and the options are then propagated in background files. However, if using the XPS flow, those changes need to be made beforehand. The synthesis phase is part of the command line call snippet below, which auto-generates individual default settings files "module_name.scr" for every module in the design:

```
1 make -f system.make bits
```

The Platgen tool invocation starts by generating the synthesis setting files for every module and immediately proceeds invoking the XST which uses those files. Therefore there is no seamless access of the synthesis options for modification before the tool uses them. A solution would be checking for the generation of the setting files, then pausing the Platgen execution, add the changes and then resume the Platgen execution. A simpler solution is to wait for the completion of the synthesis phases, then add the extra options and re-run the tool. Xilinx recommends writing a small script to replace the desired settings files with other setting files that contained the changes and to integrate a user-defined button in the GUI to run this script. In order to allow an automated system synthesis, the call to the script should be added to the auto-generated makefile. Moreover, instead of replacing the settings files by some others, the changes can be included to the file with an "echo" bash command which eases the process. The synthesis options are defined separately for every module in a .scr settings file in the synthesis folder. It gives freedom on synthesizing all the modules with an option, or for example the top level only. Correspondingly to every .scr file is a .srp report file created which includes a design resource summary and clock domains report. The impact of any XST options on the area and speed is then detectable by comparing two module.srp files. This provides a module-based level of analysis. A second level of analysis is available after the mapping process by comparing two MAP RePort (MRP) files. The area utilization reports available after synthesis are estimated values only, as opposed to the actual values obtained after mapping. So the MRP file provides values that are closer to the prototyped system.

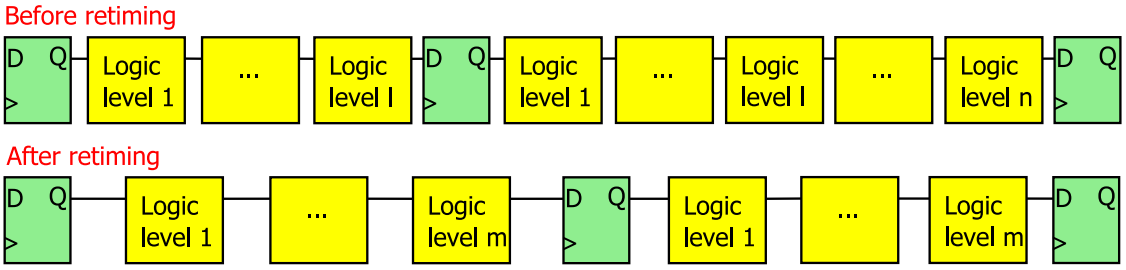
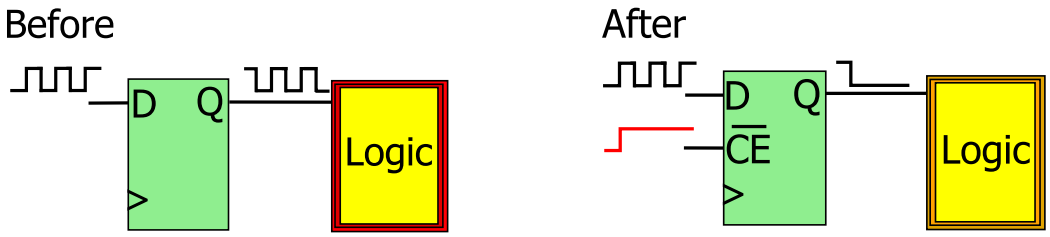


Figure 3.6.: The flip-flops retiming optimization



(a) Before clock gating, the switching activity is high and so is the power consumption

(b) After clock gating, the useless switching activity is reduced and so is the power consumption

Figure 3.7.: Using clock gating to reduce the power consumption

Contrary to the synthesis process, including power optimizations during the MAP process is easy, as it is passed to the tool during its invocation. It requires from the designer to add a "–power" tag and a given effort level, either "on", high" or "xe".

- The "on" tag is used to lower the capacitance of the design. It consists in a load grouping, data grouping and high fanout nets reallocation strategy.
- The "high" tag enables an intelligent clock and data gating algorithm which reduces useless switching activity in the design, as represented in Figure 3.7
- The extra effort "xe" tag activates bot capacitance and gating algorithms presented before. It is expected to provide the best power reduction.

(Meidanis et al. 2011) compares the impact of various optimization settings with Xilinx design tools such as the effort level of the mapping process or the use of the power reduction option for a set of algorithms. The power optimizations leads at best to more than 17% improvement and at worst to a 1% degradation. It also demonstrates that optimizations should be applied on a case by case basis. In order to compare the performance of the different optimization options during the mapping process, without running the complete flow from the beginning, the Xilinx SmartXplorer tool is used. It allows running a set of different implementation (mapping and P&R) strategies in parallel on multiple host machines. It accelerates the optimization of a system with regard to timing and power. The power information is obtained from the Xilinx Power Analyzer tool. All strategies are carried on until the creation of the NCD file which is used for the power analysis. The tool provides a power estimation based on the placed and routed design. To reduce the memory usage, SmartXplorer is configured to only generate the BIT file for the best architecture in regard to the timing properties, which here also matches with the smallest power consumption. However if it was not the case or if not only the best architecture would be required for more investigation, all strategies could be carried on until the bitstream generation. Instead of using the SmartXplorer tool, MAP options can also be inserted in the fast_runtime.opt file in the etc folder or in the system_incl.make file.

3.4. Summary

In this chapter the GUI-based ISE Xilinx SoC design flow was presented. Then its command prompt alternative was analyzed and organized to cut down user intervention. Finally, the introduction of power optimization techniques upgraded the default design flow. Based on the work carried in this chapter, a system can be synthesized, optimized and implemented into the target SoC in a fully-automated fashion.

4. Performances Analysis

In this chapter the performance evaluation inside the GRIP is compared with some available analysis and DSE tools for SoC platform. The sustainability of a high data traffic on the bus system is closely examined to address the automated appending of a device on a bus. Finally, a methodology to determine the CPU load and the chip power consumption is introduced.

4.1. Performance Evaluation Framework

The performance exploration of a complex system can be performed at multiple stages of the design and through multiple exploration strategies. Popular methods include emulation technologies, FPGA/SoC prototyping or simulation. (Moss 2012) presents a Hardware-Software Codesign and Design Space Exploration framework for SoC. System architectures are automatically converted into a SystemC virtual platform which models the hardware and C/C++ software code for the processor. Performance like latency, throughput, memory usage and CPU load are then estimated via a cycle-accurate simulation. FPGA resources utilization is estimated through an optimized Registrer Transfer Level (RTL) generation tool report. Power estimates are obtained from the Xilinx XPower Estimator tool based on the hardware resource estimates. This framework constitutes an interesting DSE tool including a high level of automation. Its main advantage is that a SystemC simulation requires relatively few iterations and a low simulation time to explore multiple design modifications. However it generally demands a complex and user-defined model of the system. On the contrary, emulation techniques and SoC prototyping offer a physical implementation of the system but at the cost of larger exploration time due to a significant synthesis time, even for minor system architecture modifications.

The GRIP targets the acceleration of DSE by automatizing system implementation, and performance exploration via prototyping. The GRIP will then benefit from the physical implementation to monitor the performances on the prototyped system. This should provide a better analysis of the system than via a simulation. To perform the design space exploration, the GRIP relies on graph grammar rules provided by the designer. Those rules are used to analyze and determine the design space that matches with the system description also provided by the designer. Once a proper system architecture inside the design space has been shaped by the tool, it is synthesized and implemented on the SoC. From there, a set of metrics are monitored and the results are provided to the GRIP for further use. The GRIP is indeed able to use metrics results from one configuration to shape other configurations out of it, thus widening the design space. After every system architecture inside the design space has been analyzed, a comparison review between them can be performed. If no system configuration matches the requirements then the designer must provide additional rules to broaden the exploration.

Today's embedded systems need to accommodate to a variety of environments and satisfy always more requirements. More widely, product differentiation has pushed systems integration and consequently the amount of data to transmit to the limits. Indeed with the multiplication of computing units of any kind in a chip, communication has revealed to be a critical factor for system performances. For high quality video-processing applications for example, a strong and consistent communication medium is essential. Nevertheless, bus communication is traditionally only addressed by the consideration of the theoretical maximum bandwidth. The reason lies in the limitation on accessibility or area to integrate monitoring devices. As described in the previous chapter, the AXI standard is tailored for a wide range of applications. However its flexibility also represents a challenge for system analysis and verification. All devices connected on the same AXI bus can request it simultaneously, imposing an unpredictable load on the bus system. So the available bandwidth for a specific module will depend on the other processing elements behavior. Given an AXI interconnect to which some active masters are connected, the remaining available load on the interconnect determines if another device can be connected on the same bus or not. This is the reason why the bus load – define as the number of data Bytes and transaction for a given time frame – is a decisive factor in modern SoC and for FPGA acceleration. This raises two requirements:

- Evaluate the load on a bus.
- Stress the bus up to its limit to determine the maximum load value. Once this maximum load value is known it is used as a reference to gauge the current load state of the bus.

While in a SystemC simulation, data traffics are part of the system model description, prototyping provides a physical access to the transactions. This information is then available, provided an adequate monitor unit on the connection medium. The Xilinx IP library contains an AXI monitor IP (Xilinx 2012d), capable of measuring major performance metrics associated with an AXI transaction. Parametrized during the system design, it gives access to a maximum of 8 physical AXI interfaces and can monitor up to 10 metrics at the same time. Its area cost ranges from 1221 to 6475 FFs and from 992 to 5594 LUTs on a Zynq device running at 200 MHz. It makes it possible to monitor the number of transactions and data for a given time duration such as two successive hardware interrupts or an image frame processing time. The number of elapsed clock cycles associated with the measurement is also monitored so that it can be checked. This monitor provides the desired functionality to monitor the amount of traffic on the Zynq device and will be implemented onto the FPGA.

Moreover, in a system simulation the amount of information exchanged can be easily adjusted to explore various traffic scenario. (ARM 2009) depicts the ARM AMBA CoreLink Verification & Performance Exploration (VPE) tool. It consists in a statistical traffic profile extractor from RTL simulation (VPE monitor/Traffic Profiling Toolkit GUI) and a cycle-accurate SystemVerilog simulator of interconnect transaction requests and responses based on the generated traffic profile (VPE master and slave). Due to its statistical-based approach on traffic generation, it claims to be more effective and faster than traditional RTL-based simulations. Mentor Graphics proposes a similar approach in its Questa Verification platform (Peryer 2013). Both provide an interesting solution for the AXI buses analysis in a simulation environment. Such a solution is however not native in a SoC. The functionality should then be implemented onto the FPGA

for evaluating if the bus communication threshold has been reached.

The AXI Exerciser (Xilinx 2012b) is a high data rate capable AXI traffic generator IP requiring about 3737 FFs and 4937 LUTs on a Zynq device running at 150 MHz. It can be configured to concurrently generate read and write AXI transactions. It has an internal command RAM which stores up to 256 write and 256 read commands. Those commands define the write and read transactions to be issued by the exerciser after it has started. The issuing of transactions stops as soon as the first command with a valid_cmd bit set to zero is met. However, if no command has this valid_cmd bit set to zero, then the list of commands is re-executed until the bus saturates. Consequently the exerciser can be used to determine the maximum load on a bus where it is acting as master. The maximum amount of traffic it is able to generate on a bus gives the maximum load value that should be respected by the accelerators that will be in the future attached to the bus. Once this value is known, determining the remaining load when a first master is already generating some data traffic on the bus should be easy. According to the superposition of the two traffics, this second value is expected to relate to the previous one, minus the traffic due to the first accelerator on the bus.

Consequently, using those two IP cores, the GRIP will integrate the ability to:

- Monitor the traffic on a bus
- Generate a flexible background traffic

It will be used according to the following example: Given a set of two tasks A, and B to implement on the platform, all the different HW/SW mapping configurations will be consider. With the appending of hardware accelerators on the bus, the risk of a traffic congestion rises. Consequently, before that both A and B can be accelerated into the PL and attached to the same bus, a test is performed with the single accelerated task A. While A is active, the traffic on the bus is measured by the monitor core. If the amount of traffic doesn't exceed the maximum value, then A and B can be connected to the same bus and the system is designed. If not, the system investigates a design alternative where an extra bus is used for B. It is assumed in this case that the FPGA still have enough resource to implement the extra bus.

4.2. Bus Load Estimation

The AXI Exerciser and Monitor are implemented in the PL. Accordingly to the methodology introduced, they are respectively used to generate a high traffic on the bus and monitor it. The data traffic is monitored between two consecutive frame interrupts from the camera module (66.6 ms). The AXI Exerciser IP core is the only master connected to the AXI_interconnect, as shown in Figure 4.3. It continuously issues read and write transactions to the system memory. Those transactions are received by the HP2 port which is the only slave attached to the bus. On its master side, the HP2 communicates with the external DDR3 memory. The Exerciser acts as a traffic generator to stress the AXI bus up to the limit of the link. The Monitor is connected at the output of the traffic generator. The transactions are incremental burst based.

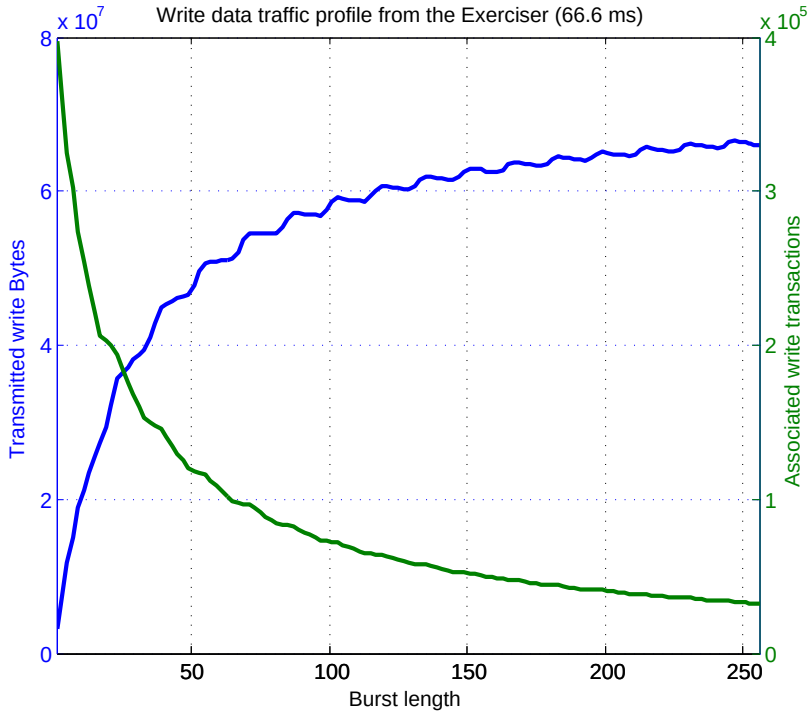


Figure 4.1.: The data traffic profile from the Xilinx AXI exerciser

The length of the burst it issues is configured in real time by the CPU. It is initially set to 1 and is successively increased until 256. Figure 4.1 gives the write data traffic profile from the exerciser for a transaction length between 1 and 256. As the length increases, the number of transmitted Bytes increases rapidly while the number of transactions required to transfer it reduces. Indeed every transaction transfers a higher number of Bytes. Consequently the crossbar's address channel control logic is less requested and more data Bytes are transferred over the bus. So by increasing the transaction burst length from the exerciser it is possible to increase the amount of data Bytes and thus reach the maximum data traffic on the bus. Figure 4.2 depicts the number of write Bytes from the exerciser against the number of write transactions. The measurements - in blue - are approximated with a linear model - in red -. The coefficient of determination is slightly higher than 0.99 . Consequently, the equation of the linear approximation is taken as reference for the maximum write bus load, according to the equation 4.1.

$$\begin{aligned} &Bus\ load = Number\ of\ transmitted\ Bytes + \beta * Number\ of\ transactions \\ &\text{with } \beta = 192.6268 \text{ and } Bus\ load_{MAX} = 72090252.0639 \end{aligned} \tag{4.1}$$

The exerciser is generating a maximum of 66 MB on the bus for the given time frame of 66.6 ms. This corresponds to a bandwidth of 991 MB/s. So the exerciser is reaching more than 86% of the theoretical bus bandwidth for a burst length of 256 (Table 2.1). For a length of 16 however, the maximum available number of Bytes only reaches 25 MB. It means that a group

of hardware accelerators with a burst length of 16 cannot generate an aggregated traffic above this limit.

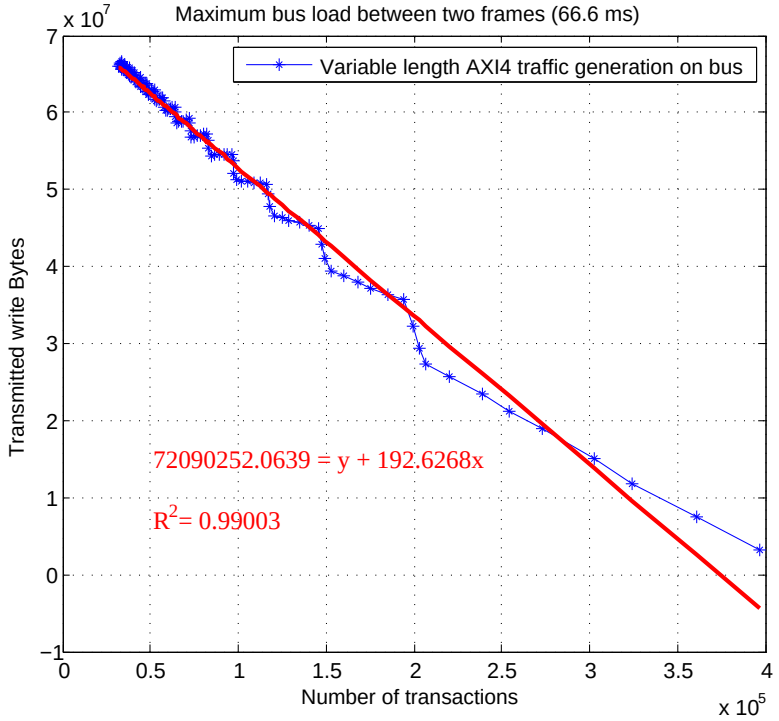
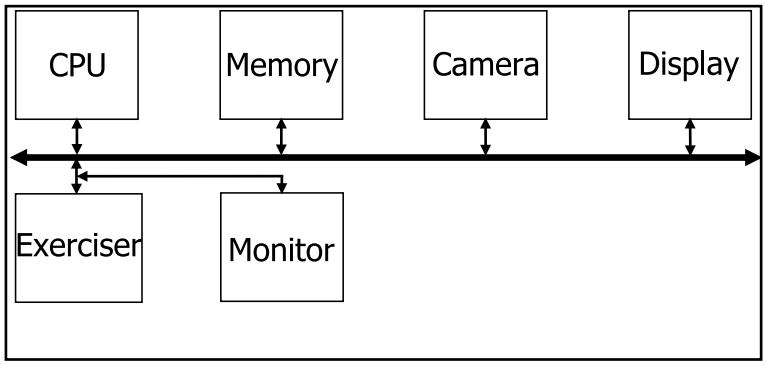
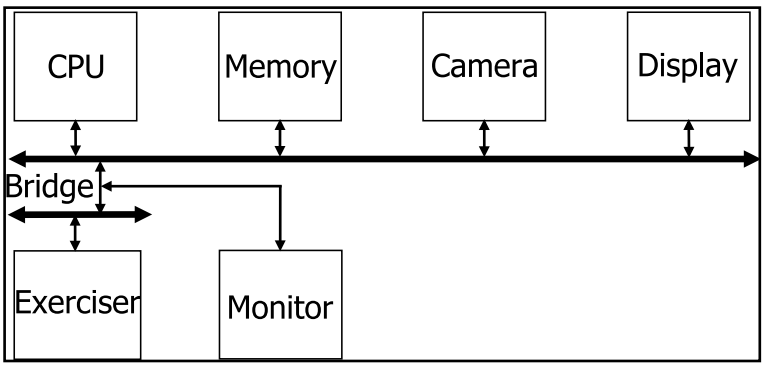


Figure 4.2.: Measure and linear estimation of the Xilinx AXI4 interconnect bus load

In the previous measurement, the AXI monitor was connected to the output of the Exerciser. See Figure 4.3a. Indeed, the monitor can only be attached to an existing Master or Slave interface. However the core can only implement up to 8 monitoring interfaces whereas the maximum number of AXI Master that can be connected on a bus is 16. If they are more than 8 Master on the bus, it is then infeasible to independently monitor them. Moreover, the area cost of using the 8 monitor interfaces would reduce the design possibilities. So monitoring the traffic on the Master side is not adapted. On the Slave side, the HP port is the only connected device. However the HP2 port is built upon the AXI3 protocol which set a burst length limit of 16. It means that a burst transaction with a length higher than 16 will be split into multiple transactions whose maximum length will be 16. Consequently, the number of transactions on the bus can only be monitored at the HP2 ports if the burst length is at most 16. For higher length burst transaction, the value measured at the HP2 port is unpredictable. A solution is to replace the interconnect by two cascaded interconnects which are bridged with an AXI2AXI interconnect. The exerciser is connected to the downstream interconnect while the HP port slave is connected to the upstream interconnect, as shown in Figure 4.3b. The correct number of transactions present on the bus can be monitored on either the master or the slave side of the bridge, regardless of the value of the length, because it respects the AXI4 protocol. Moreover, the addition of the AXI2AXI interconnect has a very minor impact on the area utilization. Indeed, no duplication of the interconnect resources occurs for the two



(a) Reaching the communication limit of the bus via the traffic exerciser. Measuring the value via the traffic monitor



(b) Interleaving a bus bridge for monitoring purpose

Figure 4.3.: Monitoring the maximum bus load

interconnects. The control logic performing the protocol conversion is only present in the upstream interconnect.

4.3. CPU Load Estimation

Every hardware accelerator implemented onto the FPGA is a dedicated module which accelerate a single processing task. On the contrary the CPU focus on a set of tasks and is then constrained by its frequency which defines the maximum of operations it can perform in a second. In the selected application, the CPU is clocked at 667 MHz which sets its maximum load to 667 million cycles of operations. So the risk of overloading the CPU exists and monitoring the CPU load is an effective mean to evaluate the amount of remaining processing capacity of the core. Thus it indicates if more tasks can be mapped in software or must be implemented onto the FPGA. The Cortex-A9 processor integrates a Performance Monitoring Unit (PMU) (ARM 2011) capable of recording up to six of the fifty-height different event notifications of the core. Those events provide a mean to analyze the global performance of the core and its

memory system at run-time. One of those events is the number of clock cycles elapsed since the last reset of its associated counter. So the processor load can be evaluated in software while the code is running on the platform.

The camera device notices the availability of a new frame to the CPU with an interrupt at a frame rate of 15 frames per second. Thus, around every 66 ms, which corresponds to 43.96 million of CPU clock cycles, a new frame is available. If this frame is acknowledged by the CPU, then upon reception of the camera interrupt it writes the frame in the DDR memory. The CPU transfer the frame from the camera module to the memory, through the AXI GP0 bus and the memory interconnect. It takes around 61.5 ms (41.3 million of CPU clock cycles) and represents 6% of the total CPU load. From then on start the processing of the frame and so should the CPU load estimation by resetting the PMU count. The input image then undergoes successively an edge-detection, an erosion and a grayscale conversion. When those three functions have completed, then the value of the PMU count provides the CPU load estimation for the processing of a single frame.

This method only holds if the total processing time doesn't exceed the frame inter-arrival time. Indeed the camera interrupt provides a good indicator on the limiting state of the processing system: if the processing of a frame requires less than the frame inter-arrival time then every interrupt can be acknowledged and the camera rate can be respected. In this case, the CPU load can be estimated in between two consecutive interrupts and the amount of work corresponding to the frame transfer to the memory only accounts once in the measurement. However if the processing requires more time and every frame is acknowledged, then the acknowledgement of several frames contributes to the measured values which is thus wrong. Indeed the number of frames that would account for the total value would depend on the exact overall processing time which is unknown. Based on some processing estimation however, one could determine the number of non-acknowledged frame that is required to accurately estimate the CPU load. Another solution is to use a large inter-frame-acknowledgement time such that here only every tenth frame is acknowledged and written to memory. The frame refresh rate is consequently limited to 1.5 fps. Moreover, because tasks are either mapped to hardware or software, the CPU is not busy during the complete frame processing time. So only the time required by every software processing step should be monitored. They are independently estimated, before being summed to obtain the total aggregated CPU load. The CPU halts while the hardware processing is performed. Each hardware-mapped processing task uses a Video Direct Memory Access (VDMA) module to transfer the frame back and forth between the memory and the hardware accelerator. A VDMA is made of two global channels:

- An ascending channel towards the accelerator which inputs the frame coming from the memory (MM2S).

- A descending channel towards the bus which outputs the processed frame from the accelerator (S2MM).

Both channels have a configurable interrupt which indicates when the processing of the frame starts and when it completes. Moreover the VDMA can be turn on or off by a function call. In this way the processing of a single frame can be ensured.

The methodology applied to estimate the time required by the three processing steps is then as follows:

- For a software-mapped task, the CPU time counter is reset before the call of the processing function and stopped after it finishes, as presented in the following for the Sobel edge-detection software function call:

```
1 init_perfcounters(1, 0);
2 %SOBEL edge-detection in SW
3 EdgeDetection(VIDEO_BASEADDR, SOBEL_VIDEO_BASEADDR, 640, 480,
4                               detailedTiming[currentResolution][H_ACTIVE_TIME]);
5 printf("The SW Sobel proccesing time took %d CPU clock cycles\n\r",
6                               get_cyclecount());
```

- For a hardware-mapped task the time is monitored between the start of the VDMA operation and the processing completion interrupt, as presented in the following for the Sobel edge-detection hardware accelerator:

```
1 init_perfcounters(1, 0);
2 %SOBEL edge-detection in HW
3 config_filterVDMA(XPAR_AXI_VDMA_4_BASEADDR, DMA_MEM_TO_DEV,
4                               VIDEO_BASEADDR);
5 config_filterVDMA(XPAR_AXI_VDMA_4_BASEADDR, DMA_DEV_TO_MEM,
6                               SOBEL_VIDEO_BASEADDR);
7 while (SOBEL_INTR == 0) ;
8 SOBEL_INTR = 0;
9 printf("The HW Sobel proccesing time took %d CPU clock cycles\n\r",
10                              get_cyclecount());
```

4.4. Power Estimation

The diversity of devices today integrated on a single chip imposes strict requirements on immunity to noise and parasitic effects. Moreover power has a prominent incidence on the maximum performance. Consequently a chip is internally powered by multiple power sources. The power driven from each power supply can be decomposed in three sections:

- The device static or leakage power features the power required by the device to operate and be available for programming. It is prominently used to hold the device configuration. The technology process, applied voltage and device junction temperature strongly impact it.

- The design static power features the additional continuous power drawn after configuration and without any activity. The elements dissipating power regardless of transistor activity such as I/O terminations and clock manager contribute to it.

- The design dynamic power features the power dissipated when the system is running. It varies over time with the switching activity α, the clock frequency, the capacitance C and the applied voltage according to the equation 4.2.

$$Dynamic\ Power = \alpha * f_{CLK} * C * V^2 \tag{4.2}$$

The power consumption of a chip can be evaluated according to two major techniques: simulation or on-chip measurement. Simulation requires a precise model of the system, whereas on-chip measurement requires accessibility to the power interface.

4.4.1. Power Estimation from Simulation Tools

Xilinx offers two levels of power investigation for its device inside two different tools: a power estimator and power analyzer.

- The Xilinx Power Estimator (XPE) (Xilinx 2014b) targets the early phases of a design, when most or part of the design is still undefined. The further the progress of the design, the more realistic the provided estimation value.

- The Xilinx Power Analyzer (XPA) (Xilinx 2011) is a post implementation tool, thus leading to more accurate estimation for the power consumption. Indeed it integrates signal trimming, register replication or retiming netlist optimizations and accurate delay propagation calculation.

The XPA is accessible from the "xpwr" command line tool invocation whereas the XPE is a Microsoft Excel macro-enabled XLSM spreadsheet. As already mentioned, all the steps of the performance estimation need to be automated. Thus the XPA has a significant advantage over the XPE. Moreover, the developed GRIP framework requires the design to be completed, giving access to the post implementation design. This makes the XPA tool a more useful candidate compared to the XPE.

The XPA has the following input files:

- Xilinx Power Estimators settings (XPA) file which contains environment and activity data.

- Post P&R design database (NCD) file which contains all logic configuration and routing information. It is available after the Placement and Routing of the design.

- Physical Constraint File (PCF) which specifies the settings for all IOs and logic of the design, as well as activity information regarding specific nets and clocks. It is created during the MAP operation.

- Optionally, a switching activity and static probability file can be included to guide the tool and improve the accuracy of the estimation. Such a file can be obtained from a behavioral or post PAR simulation. The amount of work involved to obtain a simulation file should be balanced with its potential gain in the power estimation. If it is not provided, the XPA uses the clocks constraints information from the PCF file as initial conditions for a computation of the nodes activity. This computation is based on a vectorless estimator that propagates the probability and toggle rates from the known nodes to reach a comprehensive coverage of the design activity.

The quality of the estimation significantly depends on the pertinence of the clock constraints provided, the complexity of the design and the accuracy of the selected probability and toggle rates. (Hong Nguyen et al. 2014) analyses the accuracy of a power estimation with the Xilinx XPA tools for a Sobel filter implementation on a Zynq-7000 AP SoC. The design running at 197.161 MHz consumes 170 mW when the Xilinx simulation tools estimates it around 134 mW reaching almost 23% of error. The use of a design node activity file to guide the power estimation is not mentioned. Therefore the error cannot be fully-appreciated. (Meidanis et al. 2011) analyzes the deviation between the XPA estimations and measurement on a Virtex-II Pro board for various optimization options during the mapping process. The article claims that the XPA provides overestimated results in any case.

Xilinx recommended methodology to address power analysis on the Zynq device via XPA requires some user manual intervention between the XPA and XPE tools. The XPA configuration file needs to be manually generated from the XPE tools after the parameters regarding the PS have been filled out. This represents a single effort to obtain the initial configuration file that can be later automatically updated. Besides the environment information, the file also contains the configuration parameters for the DDR, the PS and the PL.

4.4.2. On-chip Power Measurement

The on-chip power measurement requires sensing every on-board power rail. Zynq devices have not only separate power domains for the PS and the PL, but also more than ten different sources, each dedicated to an internal system block such as the clock management units or the DDR controller. So all of them need to be monitored to evaluate the total chip power consumption. When all currents flowing into the design and associated voltages are known, the computation is easy. The total chip power is the sum of every individual power driven at every source, which is the product of the voltage by the current. Voltage sources can be monitored by the user configurable analog interface (XADC) module and displayed from the ChipScopePro tool. Nevertheless, the current values remain unknown. The most convenient solution is demonstrated in the Xilinx ZC702 SoC whose integrated power regulators integrate digital power controllers offering a dedicated sensing interface for voltage and current. However such power regulators are more expensive and thus don't equip every board. The power regulators used in the ZedBoard platform are not designed to provide access to the current that flow from every power source. Another common solution is to insert a small precision resistor in series between the FPGA and the regulator output to create a small voltage drop

proportional to the current flowing from the source. This would require to modify the board and is thus not considered. However, the ZedBoard is equipped with a special sensing resistor providing designers with a voltage information image of the complete board power dissipation. A 10mΩ , 1W current sense resistor is in series with the 12V input power supply. The power is the measured voltage squared divided by the resistor value. So power estimation via on-chip measurement is also achievable. Although it doesn't allow differentiating for the PS, PL or DDR power consumption, it nonetheless provides a way to compare the system architecture changes and could be easily integrated within the GRIP framework. Consequently, the decision is taken to carry on with on-chip measurement and leave out simulation results.make

4.5. Area Estimation

Once the MAP operation completes, the hardware resources utilization is available in the MRP report file inside the implementation folder. This file can easily be parsed to extract the relevant information regarding the number of LUTs, slice registers or total number of slices used. Besides the total system resource utilization, a module-level utilization investigation is also possible. In the following, the number of total utilized slices is taken as area performance criterion.

4.6. Summary

In this chapter, the problem related to the addition of multiple devices on a same bus was introduced. The use of the Xilinx AXI exerciser as a data traffic generator and the AXI performance monitor as a data traffic analyzer was illustrated. The use of the AXI2AXI connector to relieve the HP port transaction mismatch resulting from the maximum length limitation of the AXI3 protocol was proposed. It finally presented the selected CPU load, FPGA area and power consumption estimation methodologies.

5. Results

In this chapter, the results of the Design Space Exploration are presented in three phases. The exploration is firstly performed from a pure software implementation of the processing flow, until a pure hardware implementation is reached. Several paths, corresponding to different orders of the acceleration of the tasks are analyzed. Secondly, the exploration is also performed from a pure software implementation. But this time the realization of pipelines between adequate Hardware Accelerator (HA) is investigated. Finally the improvement gained with the introduced power optimizations is analyzed for a pure hardware implementation without pipeline.

5.1. Task Acceleration on the FPGA

The different system configurations explored by the GRIP from a pure software implementation are presented in Figures 5.1 and 5.2. The three selected hardware accelerators process frames of size 640*480 with a 32-bits per pixel representation. A frame represents a total of 1.2288 MB of data that is read from the memory, processed and then written back to the memory. The VDMAs attached to the accelerator are configured to transfer a single frame on the S2MM and MM2S channels. So the expected write and read traffic on the bus per accelerator is 1.2288 MB.

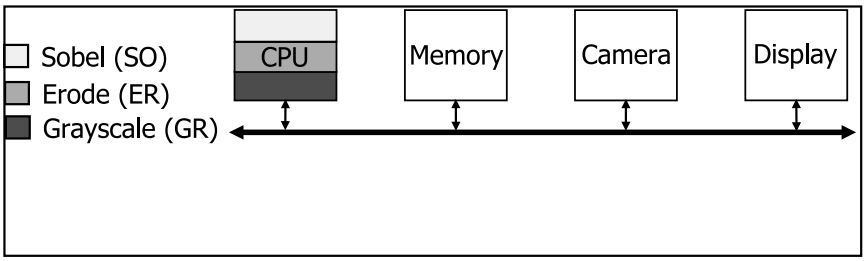
Figure 5.3 presents the variation of the CPU load with the bus load. As expected the bus load increases by steps of 1.2288 MB with the number of HW on the bus. The CPU load required independently by the tasks is as follows:

- The grayscale conversion requires 18.9 million of CPU clock cycles in software.
- The Sobel edge-detection lasts 96.9 million of CPU clock cycles in software.
- The image erosion requires the longest software processing time with 111.6 million of CPU clock cycles in software.

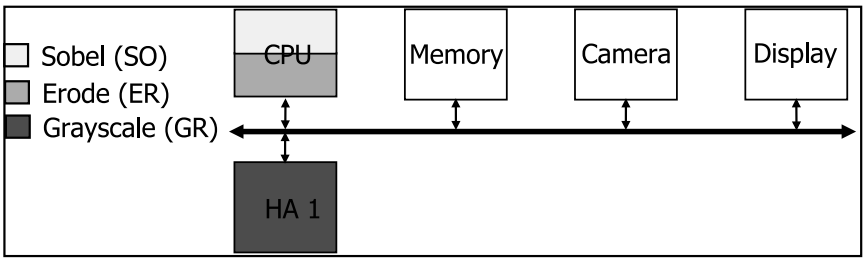
After acceleration onto the FPGA, those tasks last the equivalent of:

- 1.9 million of CPU clock cycles for the grayscale conversion in hardware.
- 2.0 million of CPU clock cycles for the edge-detection and the image erosion in hardware.

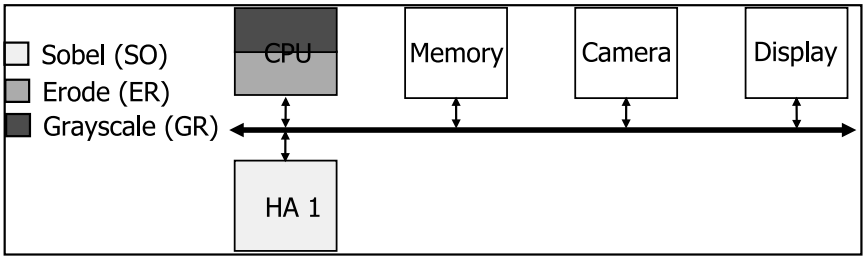
The acceleration factor is respectively about 10, 48 and 56 for the grayscale conversion, the



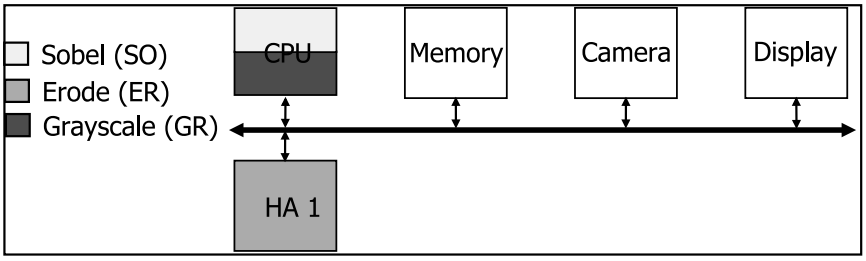
(a) System architecture 0; SW: SO-ER-GR



(b) System architecture 1; SW: SO-ER, HW: GR

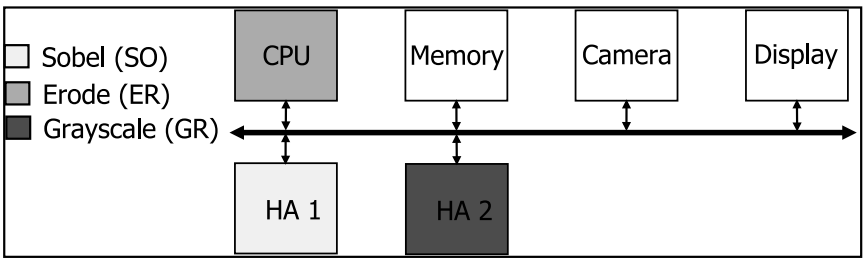


(c) System architecture 2; SW: ER-GR, HW: SO

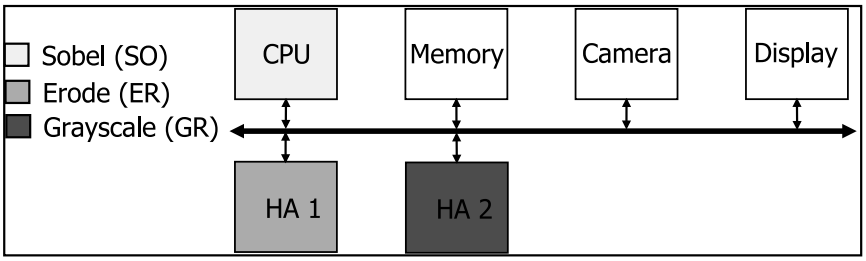


(d) System architecture 3; SW: SO-GR, HW: ER

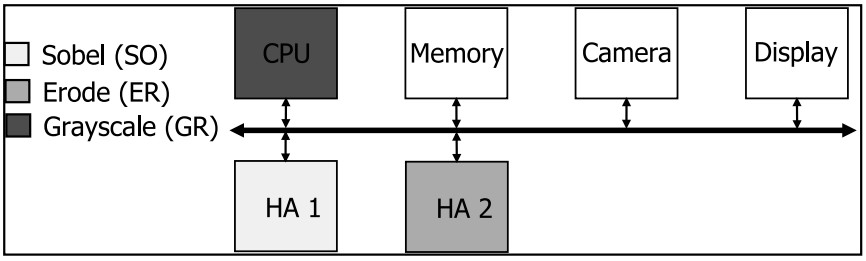
Figure 5.1.: System architectures 0 to 3



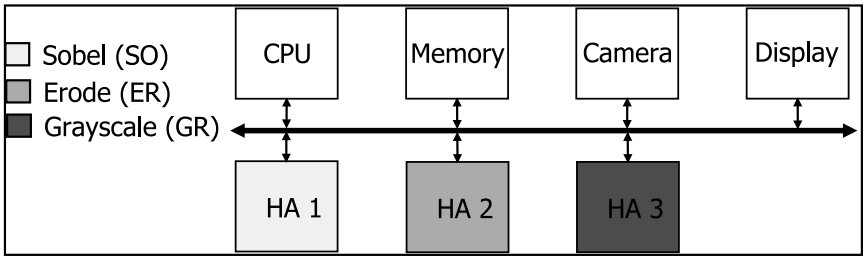
(a) System architecture 4; SW: ER, HW: SO-GR



(b) System architecture 5; SW: SO, HW: ER-GR



(c) System architecture 6 SW: GR, HW: SO-ER



(d) System architecture 7; HW:SO-ER-GR

Figure 5.2.: System architectures 4 to 7

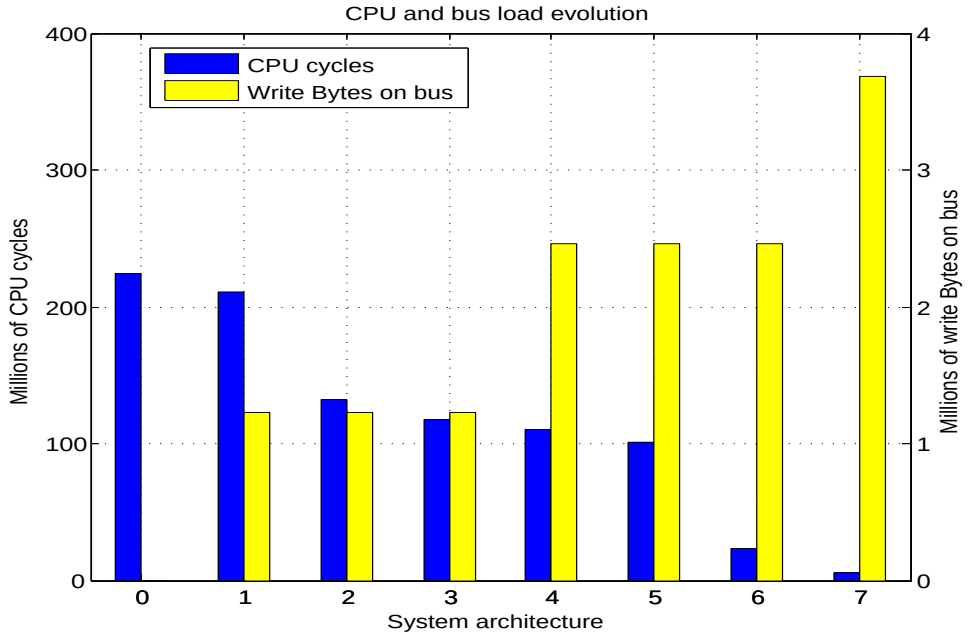


Figure 5.3.: CPU and bus load variation over the architectures 0 to 7

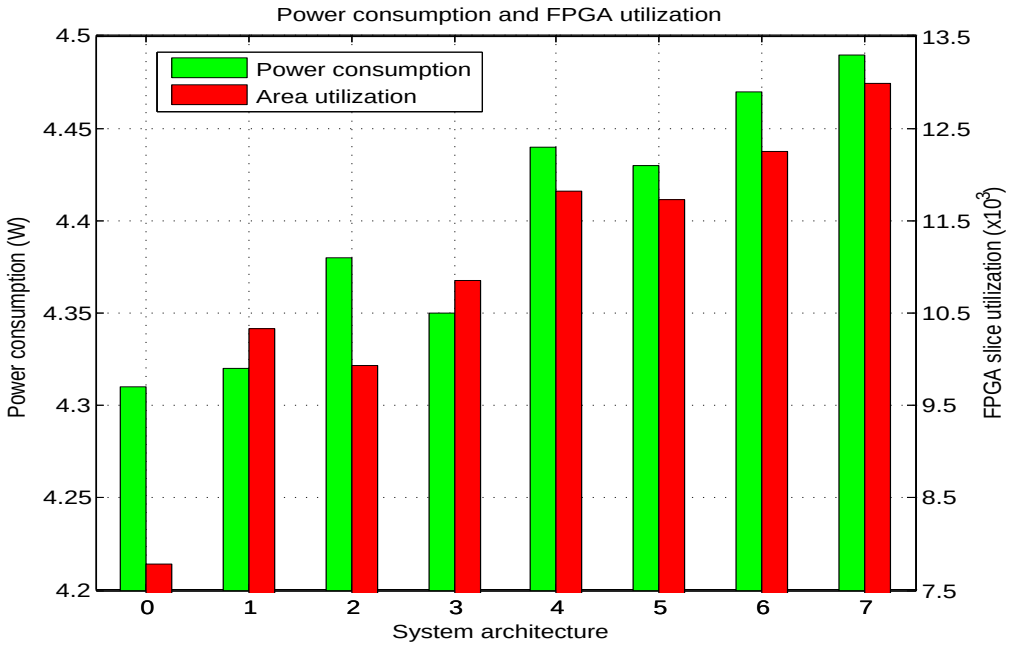


Figure 5.4.: Power consumption and area utilization over the architectures 0 to 7

edge-detection and the image erosion. Those three processing functions are significantly accelerated, thus reaching a high level of parallelization inside the FPGA. Because of the acceleration, the complete processing of a frame can be performed in 6 million of CPU clock cycles, so the sustainable frame rate reaches 111 frame per second (fps). Consequently the camera input frame rate of 15 fps can now be fulfilled. This represents an noticeable improvement compared to the pure software implementation system, where the maximum frame processing rate was less than 3 fps. See Figure 5.13 for a global overview. Figure 5.4 shows a correlation between the increase in the area utilization and the power consumption of the design. The hardware acceleration is globally responsible for a higher power dissipation inside the ZedBoard.

The performance associated with those configurations are now presented in more details in Figures 5.5, 5.6 and 5.7. The addition of each HA is identified by a dedicated color:

- A blue arrow symbolizes the acceleration of the SObel edge-detection (SO).
- A red arrow symbolizes the acceleration of the image ERosion (ER).
- A green arrow symbolizes the acceleration of the GRayscale conversion (GR).

Every nodes of the graphs represents a design configuration which is identified by the list of the software implemented tasks (SW) it contains. Figure 5.5 depicts the evolution of the CPU and bus load from a pure software to pure hardware implementation. The X-axis indicates the percentage of write data Bytes transmitted over the bus for the processing of a frame, relative to the maximum value of 3.686 MB for this application. The Y-axis indicates the percentage of total CPU clock cycles required for the processing of a frame for the given configuration. The configuration 3 exhibits a lower CPU load than the configuration 1 and 2, for a similar bus load. When a single HA is implemented onto the FPGA, the best solution regarding the CPU and bus load is therefore to accelerate the image erosion function. If two HAs are implemented, the configuration 6 provides the best solution. It is either reached by accelerating first the edge-detection then the image erosion or the other way round. Finally, the pure software implementation, respectively pure hardware implementation is the best design choice if the bus load, respectively CPU load, should be minimized. The design configurations 0, 3, 6 and 7 are the Pareto points for the CPU and bus load.

Figure 5.6 presents the evolution of the CPU load with the area utilization of the system configuration. The X-axis indicates the percentage of utilized slices inside the FPGA. The Y-axis indicates the percentage of total CPU clock cycles required for the processing of a frame. The Pareto points are the configurations 0, 2, 6 and 7. An interesting point to notice is that the acceleration of the image erosion (configuration 3) requires a higher amount of resource than the acceleration of the edge-detection (configuration 2). It shows that the higher acceleration factor is obtained at the cost of a higher logic area utilization inside the PL.

Recalling Equation 4.2, the acceleration of a processing task into the PL increases the dynamic power of the chip. The two reasons for it are:

- The switching activity of the extra FPGA resource required to implement the accelerator.

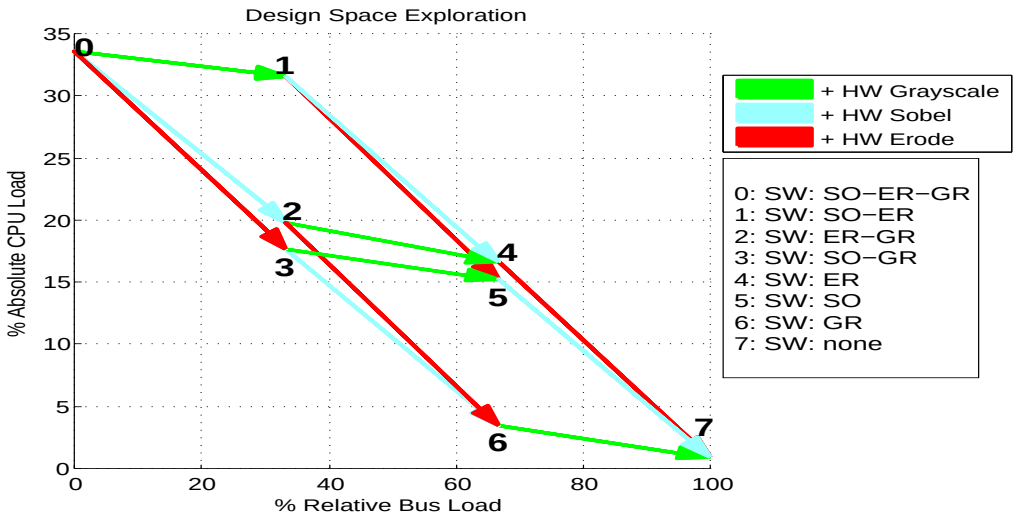


Figure 5.5.: CPU and bus load variation over the architectures 0 to 7

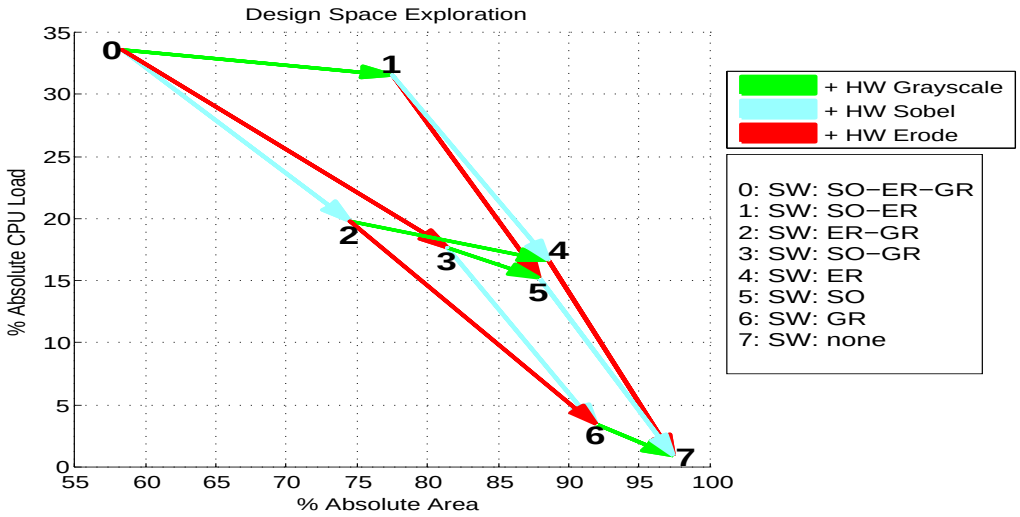


Figure 5.6.: CPU load and area utilization over the architectures 0 to 7

- The additional communication to transfer data between the memory and the hardware modules.

The measurements are performed for the processing of a single frame. So the low additional communication is expected to have a limited impact on the power increase compared to the contribution of the extra FPGA area used for the implementation of the HAs. However, because the frequency as a much higher impact on power consumption than the area does, the FPGA acceleration might reduce the power consumption. Indeed, if the accelerator is running at a lower frequency than the CPU does and the CPU load is substantially reduced due to the acceleration, then a dynamic power reduction can be reached. It means that an FPGA may not only be used to offload a processor from load demanding tasks but also as a power consumption improvement strategy. This turns out to be highly affecting for highly parallel algorithms where the FPGA can run at a significantly lower frequency than the CPU.

Figure 5.7 shows the evolution of the CPU load with the power consumption. The X-axis indicates the total power consumption of the ZedBoard. The Y-axis indicates the percentage of total CPU clock cycles required for the processing of a frame. The pure software implementation has the lowest power dissipation which the successive addition of hardware accelerators deteriorates. The configuration 3 which consumes about 7% more area than the configuration 2 is consuming less power. The reason for it lies in its 5% smaller CPU load. Indeed, while the FPGA is clocked at about 143 MHz, the CPU runs at 667 MHz. So we observe that a moderate decrease in the CPU load shortens the power consumption more than a moderate reduction of the FPGA resource utilization. The same evolution is not observed moving from the configuration 0 to either 1, 2 or 3 where the area cost of the implementation of the first HA is too high. As a matter of fact, the acceleration of the first task requires not only the

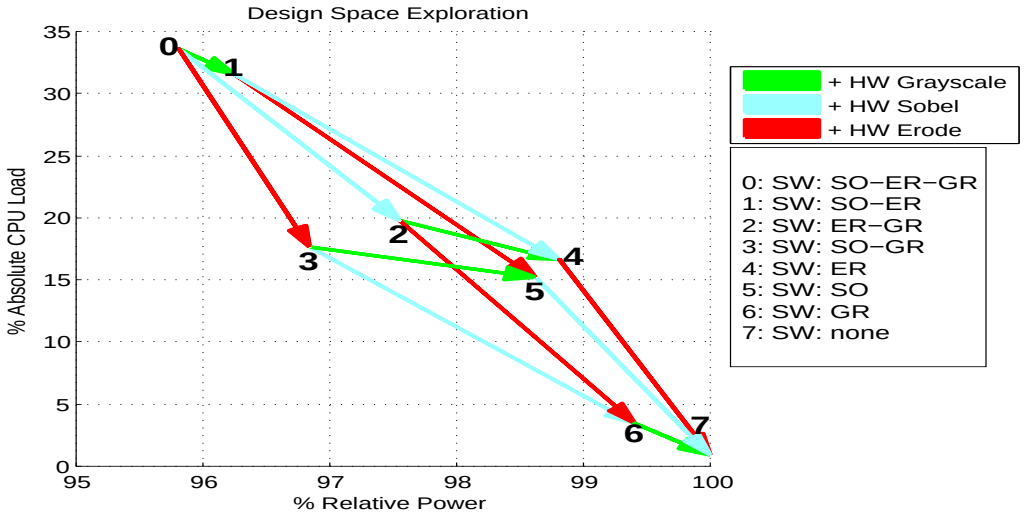


Figure 5.7.: CPU load and power consumption over the architectures 0 to 7

implementation of the dedicated HW module, but also of a dedicated bus and dedicated memory interconnect access inside the Zynq memory architecture. This huge increase in the area utilization counteract with the slight decrease in the power consumption from the CPU due to its reduced load. The configurations 0, 3, 6, 7 are the Pareto points associated with the CPU load and the power consumption.

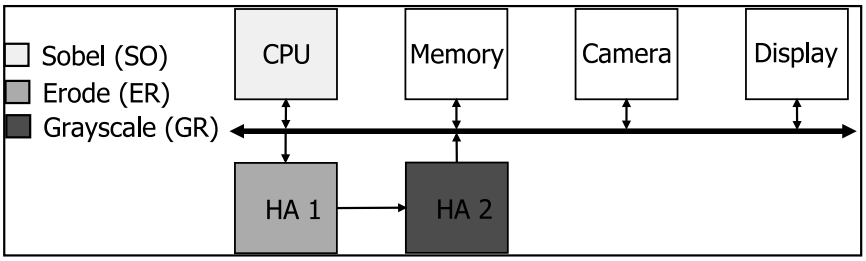
5.2. Improving the Performance with Pipeline

The system configurations related to the introduction of pipeline levels are presented in Figure 5.8. Three pipelines can be realized at most. The first one is made of the Sobel edge-detection and image erosion cores. The second one is made of the image erosion and grayscale conversion cores. The last one is the combination of the previous two. In the first two introduced pipelines, however, the remaining task can either be mapped to the CPU or to the FPGA. Consequently five different configurations integrate at least one pipeline.

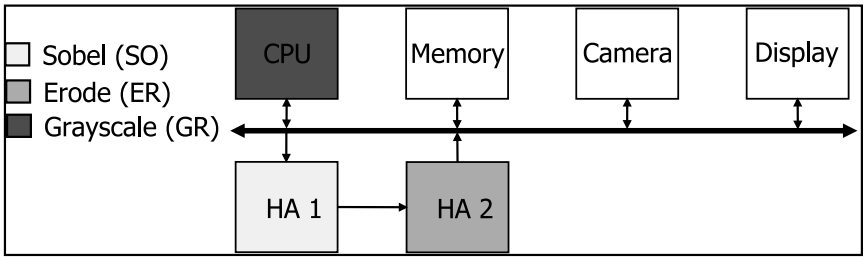
Figure 5.9 depicts the evolution of the CPU load and the bus load when pipeline stages are introduced in the design, compared to the previous non-pipelined configurations. The introduction of a pipeline is represented by a doted line. From configuration 1, where only the grayscale conversion has been accelerated, configuration 8 is designed in accelerating the image erosion and in pipelining the two cores. The bus load remains constants as only one data frame is read and write in memory. The CPU load however reduced considerably. Similarly, configuration 9 is obtained from configuration 2 in pipelining the already accelerated edge-detection with the image erosion. Configuration 3 can lead to both configurations 8 and 9. From configuration 8, the edge-detection is also accelerated. If it is used to add another pipeline level it leads to configuration 10, else 11. If it is integrated in the pipeline, then the bus load is unchanged as the amount of data transferred over the bus matches with the configurations 1, 2, 3, 8 and 9. If it is not integrated into the pipeline, then the bus load doubles. In both cases the CPU load reduces because the three tasks are performed in hardware. Similarly configuration 9 leads to either configuration 10 or 12. In all cases, using pipeline stages reduces the amount of data transfer on the bus. The biggest difference is observed between the configurations 7 and 10.

Figure 5.10 presents the impact of using pipeline levels on the area utilization. Every nodes of the graphs represents a design configuration which is identified by the list of the software tasks (SW) and the eventual level of pipeline (P) it contains. As the number of HA increases, the area utilization also increases. Pipelining the Erode and grayscale conversion HAs reduces the area utilization of around 4%, while pipelining the Sobel and Erode HAs saves about 9% of the total number of slices. The best improvement is obtained when the three HAs are pipelined, which frees 10% of the available slices. The difference of 3% between the theoretical and experimental savings is due to the higher total amount of resources used. Indeed, the larger the FPGA utilization, the tougher to pack resources inside the same slice.

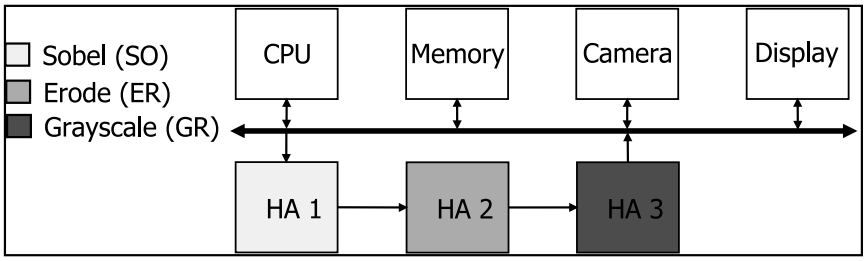
Figures 5.11 shows the performance results for the use of a two-levels pipeline between the Sobel and Erode cores. Figures 5.12 shows the performance results for the use of a three-levels pipeline between the Sobel, Erode and Grayscale cores. In the two cases, the power decrease



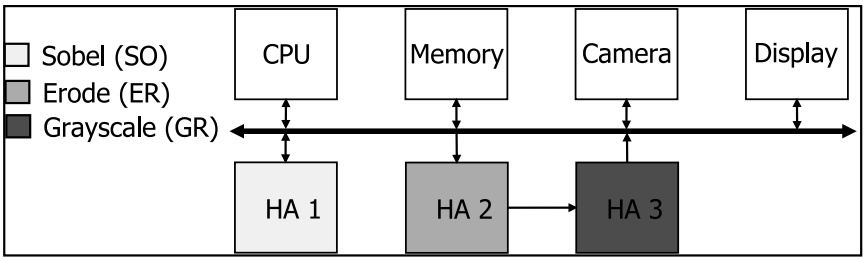
(a) System architecture 8; SW: SO, HW:ER-GR, Pipeline:ER-GR



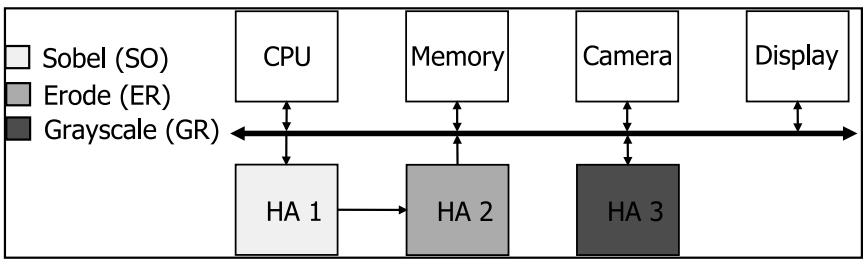
(b) System architecture 9; SW: GR, HW:SO-ER, Pipeline:SO-ER



(c) System architecture 10; HW:SO-ER-GR, Pipeline:SO-ER-GR



(d) System architecture 11; HW:SO-ER-GR, Pipeline:ER-GR



(e) System architecture 12; HW:SO-ER-GR, Pipeline:SO-ER

Figure 5.8.: System architectures 8 to 12

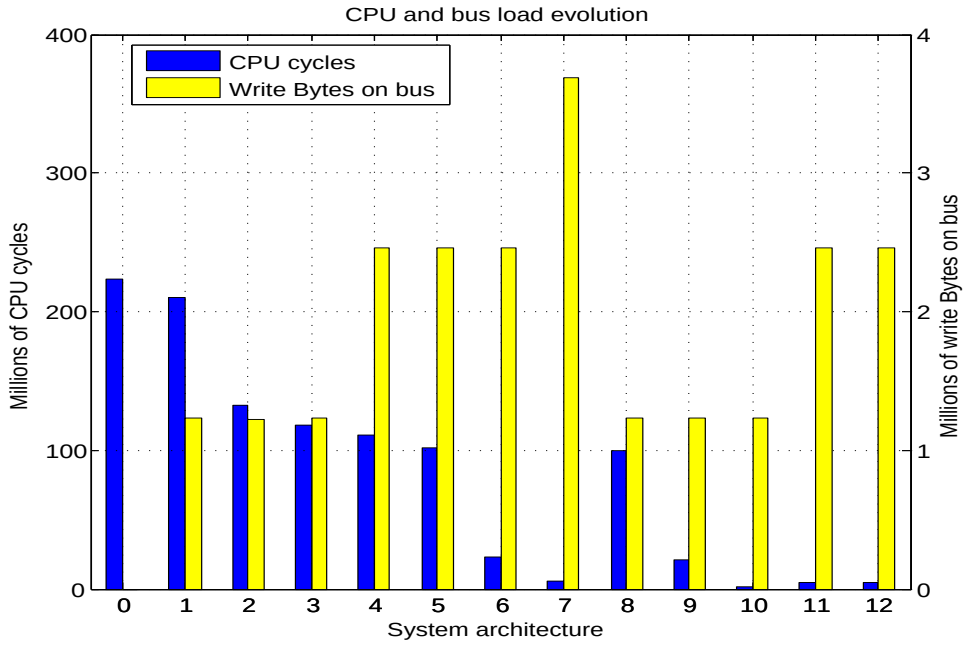


Figure 5.9.: CPU and bus load evolution over the 13 system architectures

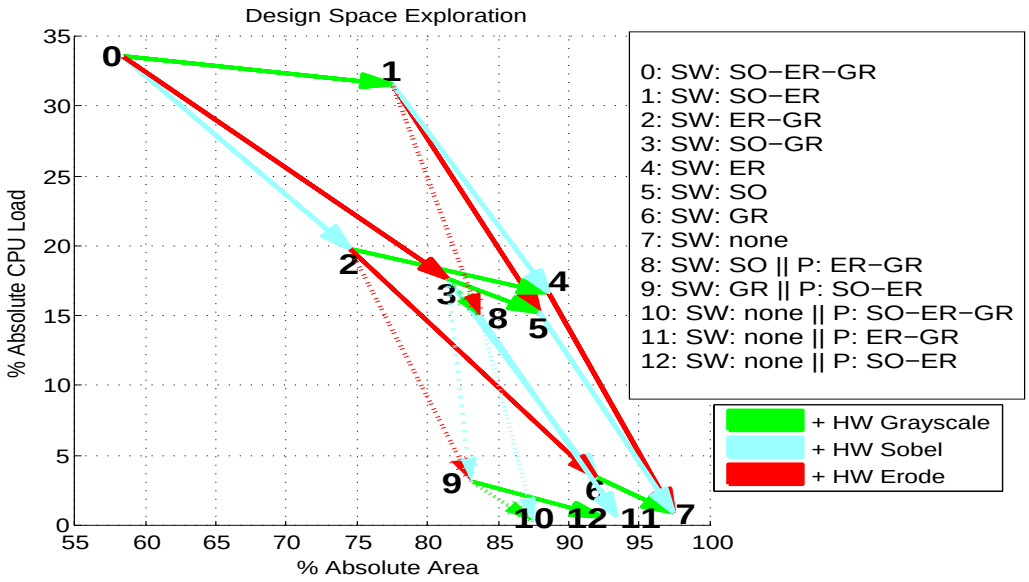
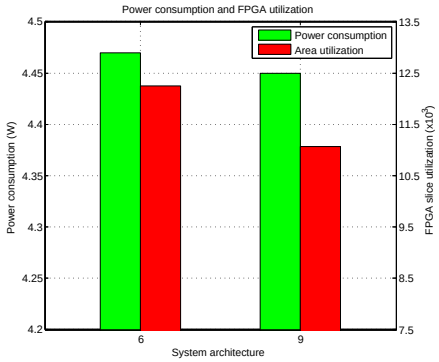
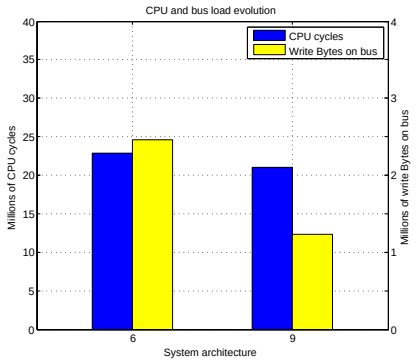


Figure 5.10.: CPU load and area utilization over the system architectures



(a) Power consumption and area utilization reduction



(b) CPU and bus load variation

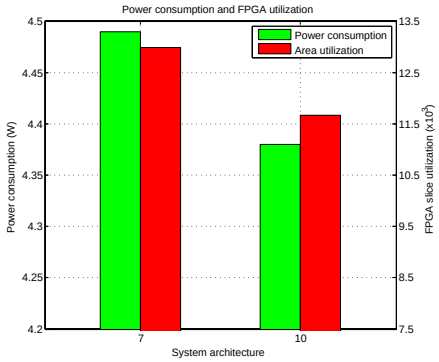
Figure 5.11.: Improvement from pipelining the Sobel and Erode HAs

obtained with the utilisation of pipeline levels originates from a reduction of the other three performance criteria.

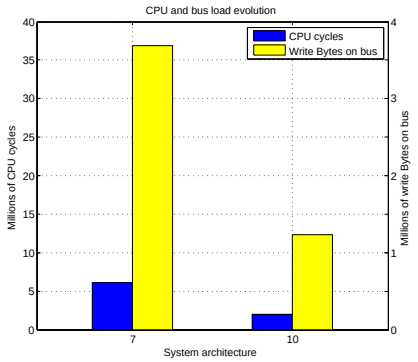
Figure 5.13 presents the sustainable frame rate results for each of the 13 investigated architectures. While a frame rate of more than 100 fps is reached with the acceleration of the three processing tasks inside the PL, the utilization of a three-levels pipeline afford a processing at 300 fps. Considering the low frame rate of 3 fps for the pure software implementation, the fully-accelerated and pipelined design represents a tremendous improvement.

5.3. Impact of Power Optimizations

To complement the conducted design space exploration, the previous system configurations could be further explored by changing design parameters. As presented in Chapter 3, two power optimizations can be applied to a selected design to improve the resource utilization and try shortening the chip power dissipation. The higher the number of modules prone to the optimizations, the higher probability to decrease the power consumption. Therefore the



(a) Power consumption and area utilization reduction



(b) CPU and bus load variation

Figure 5.12.: Improvement from pipelining the Sobel Erode and Grayscale HAs

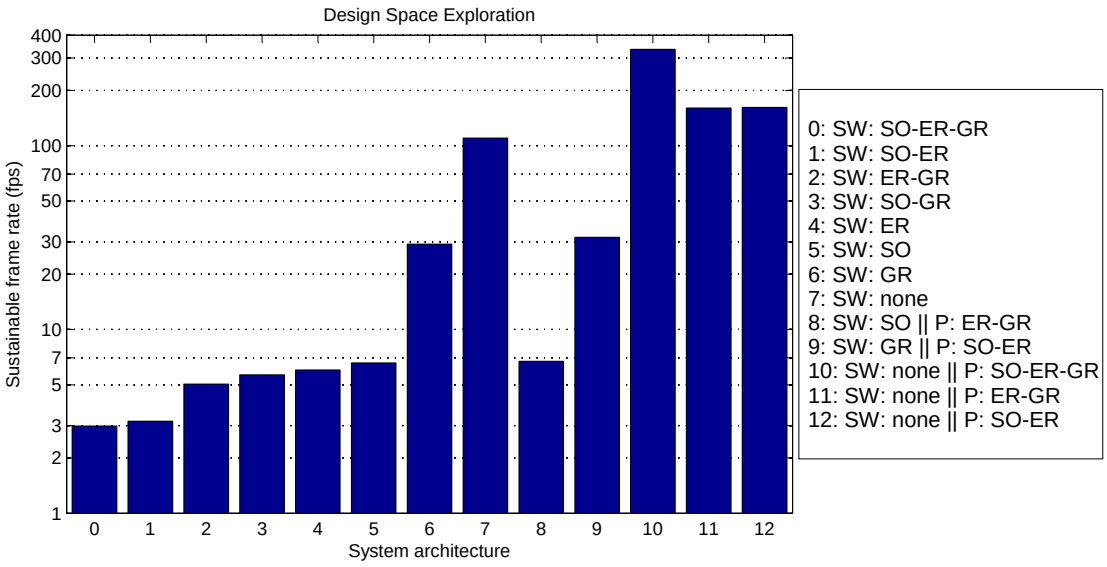


Figure 5.13.: Sustainable frame rate on the system architectures

configuration 7 is selected for optimization. Moreover, as the optimization during the synthesis process can either target the top level only or every modules in the design, the decision is taken to apply the optimization to every modules. Therefore the synthesis option files of every module of the design is updated with the desired options before the synthesis. Regarding the MAP process, the best power reduction is obtained with the "-ol high -power xe -t 4" tag during the MAP tool call and "-ol high" for the PAR tool call. Figure 5.14 presents the area and power gain obtained from it. Those two options does not affect the CPU and bus load. On both synthesis and mapping operations, it produces a better result for the power consumption and area utilization. The MAP optimization reduces the power consumption of about 50 mW and frees more than 70 slices. The XST optimization cut the power consumption of about 60 mW and releases more than 150 slices. However, based on the description of the clock and data gating given in chapter 3, the decrease in the area utilization is not self-evident. Indeed more resources should be used to provide gating capability, especially registers. It thus requires a deeper inspection.

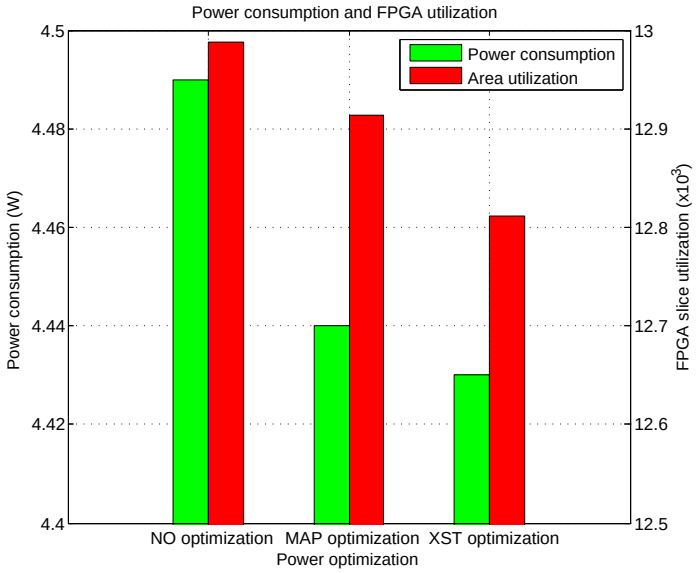


Figure 5.14.: Power consumption and area utilization reduction via optimization

According to (Xilinx 2012a), the improvements obtained through the MAP options are available in a Physical Synthesis Report (PSR) file in the project folder. The statistics information regarding the modification of the initial design mapping are presented in Table 5.1.

Resource type	Number
Slice registers gated	15359
BRAM Ports gated	40
Clock enable net processed	1553
Flops added for Enable Generation	6

Table 5.1.: Power gating algorithm impact on the resource utilization

While comparing the MAP report files for the two system configuration, the power improvement comes at the costs of a slight increase in the number of FF and LUTs used, as presented in Table 5.2.

Resource type	Number
Added FF	28
Added LUT	1278
LUT used as logic	587
LUT used exclusively as route-thrus	691

Table 5.2.: Resource usage increase

However this increase in the number of FFs and LUTs doesn't lead to an increase in the number of occupied slices. Indeed, considering how those resources span over the slices, we note a slight reduction in the total number of utilized slices and a moderate reduction in the number of non-fully utilized slices. It means that a higher packing of the resources inside the slices has been obtained. See Table 5.3.

Resource type	Number
Occupied slices	-74
Slices with an unused LUT	-904

Table 5.3.: Distribution of the logic elements inside the slices

So the better result in the power consumption actually comes from a higher resource utilization. But those resources are packed in a more complex manner into a smaller total amount of slices. This explains the utilized FPGA area decrease in the graph. After the XST optimization, the fully-accelerated system consumes a little less power than the pure software implementation which was the most power efficient system configuration explored. Even if the MAP optimization doesn't reach that level of improvement, it also provides good results. Consequently, a careful selection of the best power optimization design strategies can considerably undercut the power increase associated with FPGA acceleration.

5.4. Summary

In this chapter, a design space exploration has been performed on thirteen different system configurations for a set of four performance metrics. Two supplementary parametrized configurations were also explored. The optimal configurations were identified by the analysis of the CPU and bus load, FPGA resource and power consumption results. As shown in this chapter, the hardware acceleration of complex processing tasks considerably reduces the CPU load at the cost of an increase in the bus communication, area utilization and power consumption. The introduction of pipeline levels diminish simultaneously the resource utilization, the

bus load and the power dissipation of the selected video-processing application. Finally, by means of adequate power design parameters, the power consumption was significantly reduced, counterbalancing the power penalty of the FPGA acceleration.

6. Conclusion and Future Work

6.1. Conclusion

The current trends in System-on-Chip design are a tremendous increase in IP integration and Design & Reuse techniques. Supporting this requires alleviating the user intervention to interface multiple IP cores and explore system design alternatives. After introducing the importance of high-level system models, this Thesis firstly presented how Xilinx tools can be fully-automated to generate and implement a SoC system. Running the complete flow via a command prompt script prevents the user from interacting with every tool graphical user interface. Secondly, the hardware acceleration process automation was considered to avoid potential traffic congestion during system design. The method employed consisted in stressing the AXI communication medium up to its limit via a traffic generator and estimate this maximum value. As a result, the addition of extra hardware accelerators on a bus can be automatically validated until a threshold value is reached. System prototyping was later used to estimate several performance metrics on-chip. The power consumption, the communication and processing load of a video-processing application including a camera and a display were measured. The FPGA area utilization was derived from the generated report files. Those performances were monitored for thirteen different system configurations including different tasks mappings and pipeline levels. The results of the design space exploration were used to identify the multiple tradeoffs in the design of a SoC for a video-processing application.

6.2. Perspectives

Although performed with three video-processing steps only, the design space exploration exhibits striking performance variations. Consequently, now that the principle is validated, a broader scope of analysis could be investigated. Firstly, a more complex application made of a larger amount of IP cores could be selected. To cope with an increased complexity, the use of the second ARM core could secondly be explored, as well as the introduction of an operating system. Finally, setting up the Xilinx power estimation tools would allow supporting the on-chip measurements and differentiating between the PL and PS power increases.

Bibliography

ARM (2009): Traffic Management for Optimizing Media-Intensive SoCs, ARM.
http://www.arm.com/files/pdf/Traffic_Management_for_Optimizing_Media-Intensive_SoCs.pdf

ARM (2011): Technical Reference Manual, Revision: r3p0, ARM.
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388g/DDI0388G_cortex_a9_r3p0_trm.pdf

ARM (2013): AMBA AXI and ACE Protocol Specification, ARM.
https://silver.arm.com/download/download.tm?pv=1377613

Hong Nguyen, T.K, Belleudy, C & Pham, T.V (2014): Power evaluation of Sobel filter on Xilinx platform, IEEE.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6828607

ITRS (2011): International Technology Roadmap for Semiconductors 2011 Edition System Drivers, http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf.

Meidanis, D, Georgopoulos, K & Papaefstathiou, I (2011): FPGA power consumption measurements and estimations under different implementation parameters, IEEE.
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6132694

Moss, Laurent (2012): Rapid Design Exploration on an ESL Framework featuring Hardware-Software Codesign for ARM Processor-based FPGA's, Space Codesign Systems.
http://www.spacecodesign.com/

Peryer, Mark (2013): Caching in on Analysis, Mentor Graphics.
https://s3.amazonaws.com/verificationhorizons.verificationacademy.com/volume-9_issue-3/articles/stream/caching-in-on-analysis_vh-v9-i3.pdf

Razzaq, Umair (2013): Design Space Exploration for Embedded Vision Application on Zynq FPGA using HLS.

Tech411, Source (2013): Top FPGA Companies For 2013.
http://sourcetech411.com/2013/04/top-fpga-companies-for-2013

Xilinx (2011): Power Methodology Guide, Xilinx Inc.

http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/ug786_PowerMethodology.pdf

Xilinx (2012a): Analysis of Power Savings from Intelligent Clock Gating, Xilinx Inc.
http://www.xilinx.com/support/documentation/application_notes/xapp790-7-series-clock-gating.pdf

Xilinx (2012b): LogiCore IP AXI Exerciser v4.00a Product guide, Xilinx Inc.
http://www.xilinx.com/support/documentation/ip_documentation/axi_exerciser/v4_00_a/pg094-axi-exerciser.pdf

Xilinx (2012c): LogiCore IP AXI Interconnect v1.06a, Xilinx Inc.
http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_06_a/ds768_axi_interconnect.pdf

Xilinx (2012d): LogiCORE IP AXI Performance Monitor v3.00a Product Guide, Xilinx Inc.
http://www.xilinx.com/support/documentation/ip_documentation/axi_perf_mon/v3_00_a/pg037_axi_perf_mon.pdf

Xilinx (2013a): Command Line Tools User Guide, Xilinx Inc.
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/devref.pdf

Xilinx (2013b): Lowering Power at 28 nm with Xilinx 7 Series Devices, Xilinx Inc.
http://www.xilinx.com/support/documentation/white_papers/wp389_Lowering_Power_at_28nm.pdf

Xilinx (2014a): 7 Series FPGAs Configurable Logic Block, Xilinx Inc.
http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

Xilinx (2014b): Xilinx Power Estimator User Guide, Xilinx Inc.
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug440-xilinx-power-estimator.pdf