ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση με γλώσσα περιγραφής υλικού VHDL του πρωτοκόλλου συμπίεσης εικόνας JPEG-2000 σε πλατφόρμα Xilinx Virtex-5

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ζήσης Παρασκευάς Χ. Πούλος

Επιβλέπων: Δημήτριος Σούντρης

Αν. Καθηγητής ΕΜΠ

Αθήνα, Απρίλιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση με γλώσσα περιγραφής υλικού VHDL του πρωτοκόλλου συμπίεσης εικόνας JPEG-2000 σε πλατφόρμα Xilinx Virtex-5

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ζήσης Παρασκευάς Χ. Πούλος

Επιβλέπων : Δημήτριος Σούντρης

Αν. Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 8η Απριλίου 2011.

....	....	....
Δημήτριος Σούντρης	Κιαμάλ Πεκμεστζή	Γεώργιος Οικονομάκος

Αθήνα, Απρίλιος 2011

....

Ζήσης Παρασκευάς Χ. Πούλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Abstract

The purpose of the present diploma thesis is the co-design and implementation of a Digital Signal Processing (DSP) application on a Xilinx Virtex-5 platform. The DSP application that was selected for this purpose was the JPEG2000 image compression standard. The whole procedure is presented here split in two major parts. In the first part, the methodology that led to a specific hardware/software partitioning strategy is presented, including specification analysis and profiling of *JasPer*, an open-source software-based implementation of the JPEG2000 codec. A detailed set of timing profiles is presented for the JasPer code. Analysis of these profiles led to the decision of selecting the Inverse Discrete Wavelet Transform for implementation in hardware. Additionally, the first part contains a description of the hardware architecture that was implemented in VHDL and the respective simulation results that followed. Furthermore, the first part includes a presentation of the Xilinx EDK tool set and the JPEG200 co-design architecture. The last chapter of the first part presents the implementation results. In the second part, a step-by-step guide is presented, which allows one to follow all the basic and essential steps in order to integrate the developed VHDL design into a larger System-on-Chip and implement it on a Xilinx Virtex-5 development platform. The system incorporates a MicroBlaze processor and was designed and implemented using the set of tools included in the Embedded Development Kit (EDK), which is provided by Xilinx.

KeyWords

Embedded Systems, FPGA, DSP, JPEG2000, Hardware/Software partitioning, Discrete Wavelet Transform, Xilinx EDK, Xilinx Virtex-5

Table of Contents

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής θα ήθελα να ευχαριστήσω κατά κύριο λόγο τον καθηγητή κ. Δημήτριο Σούντρη για την ευκαιρία που μου έδωσε να δουλέψω σε ένα αντικείμενο που με ενδιαφέρει ιδιαίτερα και για τις σημαντικές συμβουλές του καθ' όλη τη διάρκεια της διπλωματικής εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τον καθηγητή κ. Κ. Πεκμεστζή για την παροχή της πλατφόρμας ανάπτυξης και τον λέκτορα κ. Γ. Οικονομάκο για τις πολύ σημαντικές συμβουλές του. Θα ήθελα να ευχαριστήσω, επίσης, όλο το προσωπικό του εργαστηρίου για τις χρήσιμες συμβουλές τους πάνω στην εργασία μου και ιδιαίτερα τους Κ. Σιώζιο, Δ. Διαμαντόπουλο και Δ. Μπεκιάρη. Τέλος θεωρώ χρέος μου να ευχαριστήσω την οικογένειά μου και τους φίλους μου για την ψυχολογική αλλά και υλική υποστήριξη τους.

Part 1

Chapter 1

Embedded Systems

1.1 Definition

Embedded systems are specialized, application-specific computing devices that are not deployed as general purpose computers. An embedded system is preprogrammed to perform a narrow range of functions with minimal end user or operator intervention, usually with real-time computing constraints. These systems are components of a larger complete device, often including hardware and mechanical parts.

1.2 Some Examples

The domain of embedded systems is thriving. From telecommunications to aerospace and medical electronic equipment, embedded systems span all aspects of modern life and there are many examples of their use. Some examples of devices that incorporate one or more embedded systems are given below:

- Personal Digital Assistants (PDAs)
- MP3 players, digital cameras, DVD players, mobile phones, printers
- Videogame consoles

- Microwave ovens, washing machines
- Avionics, inertial guidance systems, GPS receivers
- Automotive systems, traction control systems, ABS
- Medical equipment, PET, SPECT, CT, MRI

1.3 Characteristics and Requirements

Embedded computing is in many ways much more demanding than programming a PC or a workstation. Of course, functionality is important in both general-purpose computing and embedded computing, but embedded applications must meet many other constraints as well. Some major characteristics and requirements of embedded computing applications are:

- *Real time:* Many embedded systems have to perform in real time and data need to be ready on time. In some cases, failure to meet a deadline is unsafe and can even endanger lives. In other cases, missing a deadline doesn't create safety problems but does create unhappy customers.
- *Performance:* The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost. Performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.
- *Cost:* Cost typically has two major components: **manufacturing cost** includes the cost of components and assembly; and **nonrecurring engineering (NRE)** costs include the personnel and other costs of designing the system.
- *Power consumption:* Power, of course, is important in battery-powered systems and is often important in other applications as well. Minimizing heat

production is also a major issue and one that is closely associated with power management.

- *Reliability:* Embedded systems often reside in machines that are expected to run continuously for years without errors and in some cases recover by themselves if an error occurs. Therefore the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.

- *Upgradeability:* The hardware platform may be used over several product generations or for several different versions of a product in the same generation, with few or no changes. However, we want to be able to add features by changing software. Therefore it is of major importance to design a system able to provide the required performance for software not yet developed.

- *Physical size and weight:* The physical aspects of the final system can vary greatly depending upon the application. A handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.

- *Complexity and user interfacing:* The operations performed by the microprocessor may be highly sophisticated (i.e. complicated filtering functions etc.). Furthermore, these systems are frequently used to control complex user interfaces that may include multiple menus and many options (i.e. GPS navigation systems).

1.4 Challenges and the Concept of Co-design

Embedded systems incorporate Hardware (HW) and Software (SW) parts which affect the design process itself resulting in a HW/SW co-design flow. Mixed HW/SW systems are not new. What has considerably grown in recent years is the trend toward methodologies that concurrently apply design techniques from different areas to develop mixed digital systems. When hardware is tuned to its software applications, and vice versa, during the design process, it is impossible to exploit the capabilities of such heterogeneous systems. HW/SW Co-design is the system design process that combines the hardware and software perspectives from the earliest stages to exploit design flexibility and efficient allocation of functions. The concurrent design of hardware and software has shown to be advantageous as long as HW and SW are considered as a whole instead of independent entities. Although benefits of hardware and software working together are evident, complex systems design involving both HW and SW is a non-trivial task due to the interaction of different kinds of system philosophies.

Today the electronic market demands high-performance and low-cost products. Both performance and cost are essential to commercial competitiveness. Thus, the chip industry has faced two major challenges in order to satisfy the consumer needs: the increase in system complexity and the reduction in design times. High functionality on a single chip and reduced time-to-market are goals that can be achieved through co-design methodologies.

1.5 Design Tasks

Various design tasks have to be considered for the implementation of an embedded system. Some of the most typical tasks for the HW/SW co-design are listed in the following:

- *Design space exploration:* Design Space Exploration (DSE) refers to the process of investigating implementation variants regarding their optimal solution. In the case of multiple objectives like minimization of time, area, and power not only a single optimal solution exists.

- *HW/SW partitioning:* HW/SW partitioning can in general be described as the mapping of the interconnected functional objects that constitute the behavior of the algorithm onto a chosen architecture model.

- *Platform based design:* Platform based design focuses on a specific application domain. The platform embodies the hardware architecture, embedded software architecture, and design methodologies for IP authoring and integration. Derivative designs may be rapidly implemented from a single platform that has a fixed and a variable part.

- *Verification:* Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the condition imposed at the start of the phase. This correctness can be verified by simulation or formal methods like for example equivalence-check.

- *Rapid prototyping*: Rapid Prototyping describes the fast development of a working entity to prove that a new theory could really be applied and to have a first impression of the development effort for turning it into a product. Due to the high complexity of modern systems prototyping has become nearly as challenging as designing the product itself.

Chapter 2

DSP and FPGAs

2.1. General Information

Digital Signal Processing (DSP) is closely related to the domain of Embedded Systems. It is used in a very wide range of applications from high-definition TV, mobile telephony, digital audio, multimedia, digital cameras, radar, sonar detectors, biomedical imaging, speech recognition, to name but a few. The topic has been driven by the application requirements which have only been possible to realize because of development in silicon chip technology. Developing both programmable DSP chips and dedicated system-on-chip (SoC) solutions for these applications, has been an active area of research and development over the past three decades. Indeed, a class of dedicated microprocessors has evolved particularly targeted at DSP, namely DSP microprocessors or DSPμs.

The increasing costs of silicon technology have put considerable pressure on developing dedicated SoC solutions and means that the technology will be used increasingly for high-volume or specialist markets. An alternative is to use microprocessor style solutions such as microcontrollers, microprocessors and DSP micros, but in some cases, these offerings do not match well to the speed, area and power consumption requirements of many DSP applications. More recently, the field-programmable gate array (FPGA) has been proposed as a hardware technology for DSP systems as they offer the capability to develop the most suitable circuit architecture for the computational, memory and power requirements of the application in a similar way to SoC systems. This has removed the preconception that FPGAs are only used as 'glue logic' platform and more realistically shows that FPGAs are a collection of system components with which the user can create a DSP system.

2.2. Field-programmable Gate Arrays

FPGAs emerged as simple 'glue logic' technology, providing programmable connectivity between major components where the programmability was based on either antifuse, EPROM or SRAM technologies. This approach allows design errors which had only been recognized at this late stage of development to be corrected, possibly by simply reprogramming the FPGA thereby allowing the interconnectivity of the components to be changed as required. Whilst this approach introduced additional delays due to the programmable interconnect, it avoids a costly and time-consuming board redesign and considerably reduced the design risks.

Like many other industries in the area of electronics, the creation and growth in the market has been driven by Moore's law (1965), depicted in Figure 2.1. Moore's law shows that the number of transistors has been doubling every 18 months. This massive growth has led to the creation of a number of markets and is the driving force between the markets of many electronic products such as mobile telephony, digital TV etc. This is because not only have the number of transistors increased dramatically, but the costs have not increased, thus reducing the cost per transistor at every technology advance. Under these ideal conditions the FPGA market has grown from nothing in just over 20 years to playing a major role in the IC industry with a market judged to be of the order of US$ 4.0 billion.

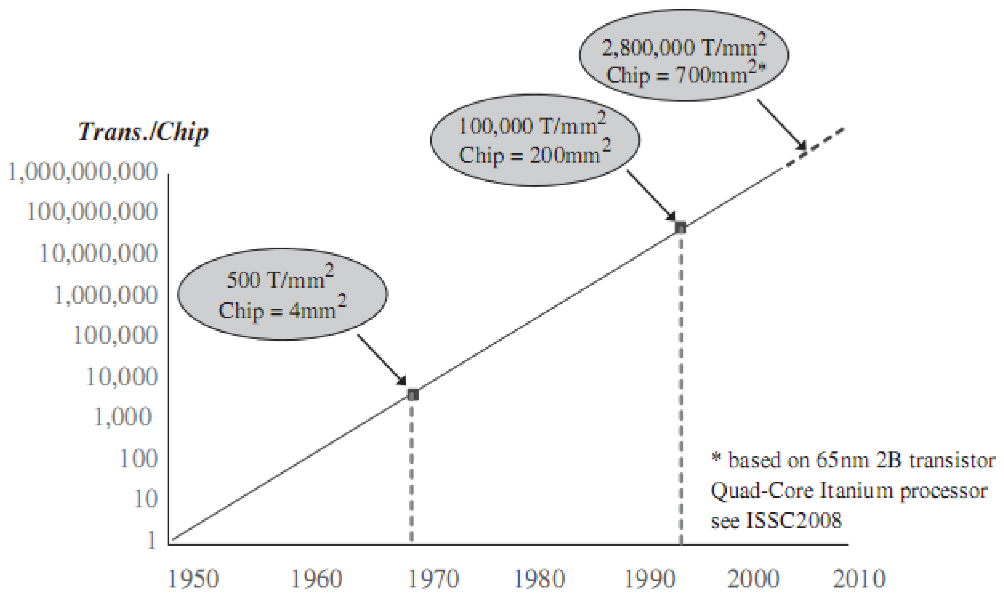


Figure 2.1 Moore's law (1965)

2.2.1 Short history of FPGAs

The FPGA concept emerged in 1985 with the XC2064™ FPGA family from Xilinx. At the same time, a company called Altera were also developing a programmable device, later to become EP1200 device which was the first high-density programmable logic device. Altera's technology was manufactured using 3-μm CMOS erasable programmable read-only-memory (EPROM) technology and required ultraviolet light to erase the programming whereas Xilinx's technology was based on conventional static RAM technology and required an EPROM to sore the programming. The co-founder of Xilinx, Ross Freeman argued that with continuously improving silicon technology, transistors were going to increasingly get cheaper and could be used to offer programmability. This is was the start of an FPGA market which was then populated by quite a number of vendors, including Xilinx, Altera, Actel, Lattice, Crosspoint, Algotronix, Prizm, Plessey, Toshiba, Motorola, and IBM. The market has now grown considerably and Gartner Dataquest indicated a market size growth to 4.5 billion in 2006, 5.2 billion in 2007 and 6.3 billion in 2008. There have been many changes in the market, including a severe rationalization of technologies with many vendors such as Crosspoint, Algotronix, Prizm, Plessey, Toshiba, Motorola, and IBM disappearing from the market and a reduction in the number of FPGA families as well as the emergence of SRAM technology as the dominant technology largely due to cost. The market is now dominated by Xilinx and Altera and more importantly, the FPGA has grown from being a simple glue logic component to representing a complete System on Programmable Chip (SoPC) comprising on-board physical processors, soft processor, dedicated DSP hardware, memory and high-speed I/O. In the 1990s, ASIC was still seen for the key mass market areas where really high performance and energy considerations were seen as key drivers such as mobile communications. Thus graphs comparing performance metrics for FPGA, ASIC and processor were generated and used by vendors to indicate design choices.

The FPGA evolution is summarized in Table 2.1. It indicates three different eras of evolution of the FPGA. The age of *invention* where FPGAs started to emerge and were being used as system components. The age of *expansion* is where the FPGA started to approach the problem size and thus design complexity was key. The final evolution stage is described as the period of *accumulation* where FPGA started to incorporate processors and high-speed interconnection.

Period	Age	Comments
1984–1991	Invention	Technology is limited, FPGAs are much smaller than the application problem size
		Design automation is secondary
		Architecture efficiency is key
1992–1999	Expansion	FPGA size approaches the problem size
		Ease-of-design becomes critical
2000–2007	Accumulation	FPGAs are larger than the typical problem size
		Logic capacity limited by I/O bandwidth

Table 2.1 Three ages of FPGAs

2.2.2 Challenges of FPGAs

The emergence of the FPGA as a DSP platform was accelerated by the application of distributed arithmetic (DA) techniques (Goslin 1995, Meyer-Baese 2001). DA allowed efficient FPGA implementations to be realized using the LUT-based/adder constructs of FPGA blocks and allowed considerable performance gains to be gleaned for some DSP transforms such as fixed coefficient filtering and transform functions suach as Fast Fourier Transform (FFT). Whilst these techniques demonstrated that FPGAs could produce highly effective solutions for DSP applications, the concept of squeezing the last aspect of performance out of the FPGA hardware and more importantly, spending several person months for the creation of such innovative designs, meant that there was a growing gap in the scope offered by current FPGA technology and the designer's ability to develop efficient solutions using modern tools. This was similar to the 'design productivity gap' (ITRS 1999) identified in the ASIC industry where it was viewed that ASIC design capability was only growing at 25% whereas Moore's law growth was 60%. This is proved by even more recent data during the 2007 ITRS roadmap (Figure 2.2).

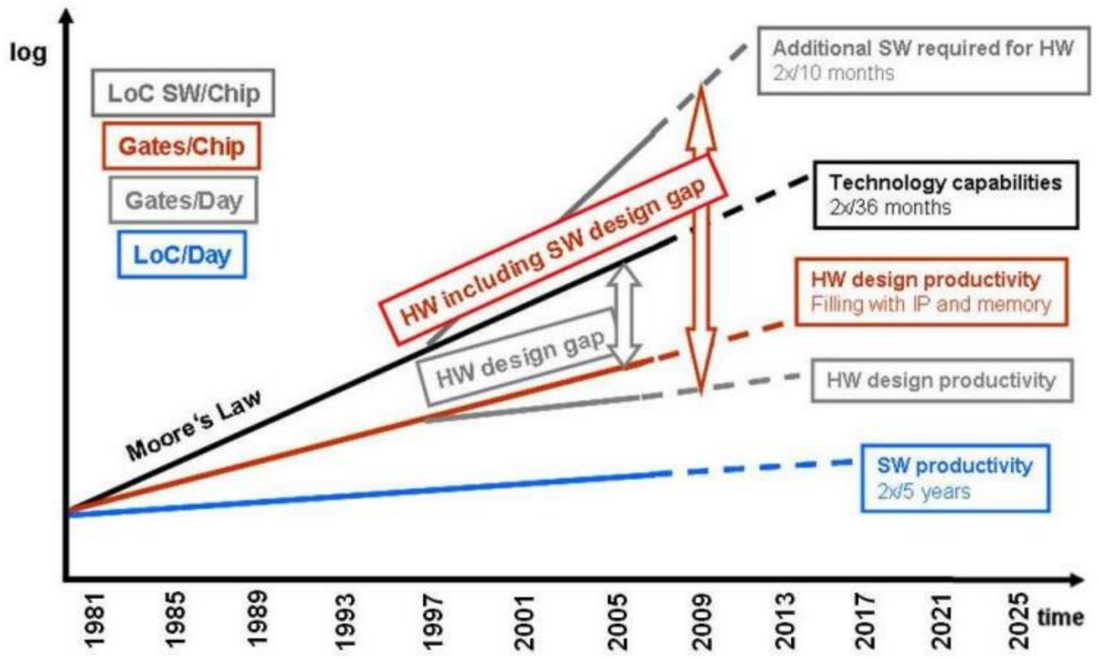


Figure 2.2 The design productivity gap (ITRS 2007)

The problem is not as severe in FPGA implementation, because sub-micrometre design issues are missing. However, a number of key issues exist and include:

- *Design languages.* Currently hardware description languages such as VHDL and Verilog and their respective synthesis flows are well established. However, users are now looking at FPGAs with the recent increase in complexity resulting in the integration of both fixed and programmable microprocessors cores as a complete system, and looking for design representations that more clearly represent system description. Therefore, there is an increased EDA focus on using C as a design language.

- *Understanding how to map DSP functionality into FPGA.* Some of the aspects are relatively basic in this area, such as multiplications, additions and delays being mapped onto on-board multipliers, adder and registers and RAM components respectively. However, the understanding of floating-point versus fixed-point, word length optimization, algorithmic transformation cost functions for FPGA and impact of routing delay are issues that must be considered at a system level and can be much harder to deal with at this level.

- *Development and use of IP cores.* With the absence of quick and reliable solutions to the design language and synthesis issues, the IP market in SoC implementation has emerged to fill the gap and allow rapid prototyping of hardware. Soft cores are particularly attractive as design functionality can be captured using HDLs and efficiently translated into the FPGA technology of choice in a highly efficient manner by conventional synthesis tools. In addition, processor cores have been developed which allow dedicated functionality to be added. The attraction of these approaches are that they allow application specific functionality to be quickly created as the platform is largely fixed.

- *Design flow.* Most of the design flow capability is based around developing FPGA functionality from some form of higher-level description, mostly for complex functions. The reality now is that FPGA technology is evolving at such a rate that systems comprising FPGAs and processors are starting to emerge as a SoC platform or indeed, FPGAs as a single SoC platform as they have on-board hard and soft processors, high-speed communications and programmable resource, and this can be viewed as a complete system. Conventionally, software flows have been more advanced for processors and even multiple processors as the architecture is fixed. Whilst tools have developed for hardware platforms such as FPGAs, there is a definite need for software for flows for heterogeneous platforms, i.e. those that involve both processors and FPGAs.

2.3 DSP System Basics

There is an increasing need to process, interpret and comprehend information, including numerous industrial, military, and consumer applications. Many of these involve speech, music, images or video, or may support communication systems through error detection and correction, and cryptography algorithms. This involves real-time processing of a considerable amount of different types of content at a series of sampling rates ranging from single Hz as in biomedical applications, right up to tens of MHz as in image processing applications. In a lot of cases, the aim is to process the data to enhance part of the signal, such as edge detection in image processing or eliminating interference such as jamming signals in radar applications, or removing erroneous input, as in the case of echo or noise cancellation in telephony. Other DSP algorithms are essential in capturing, storing and transmitting data, audio, images and video; compression techniques have been used successfully in digital broadcasting and telecommunications. Over the years, a lot of the need for such processing has been standardized, as illustrated by Figure 2.3 which gives an illustration of the algorithms required in a range of applications. In communications, the need to provide efficient transmission using orthogonal frequency division multiplexing (OFDM) has emphasized the need for circuits for performing the FFT. In image compression, the evolution initially of the joint photographic experts group (JPEG) and then the motion picture experts group (MPEG), led to the development of the JPEG and MPEG standards respectively; these standards involve a number of core DSP algorithms, specifically DCT and motion estimation and compensation. The appeal of processing signals digitally was recognized quite some time ago as digital hardware is generally superior and more reliable than its analogue counterpart;

analogue hardware can be prone to ageing and can give uncertain performance in production. DSP on the other hand, gives a guaranteed accuracy and essentially perfect reproducibility (Rabiner and Gold 1975). The main proliferation of DSP has been driven by the availability of increasingly cheap hardware, allowing the technology to be easily interfaced to computer technology, and in many cases, to be implemented on the same computers. The need for many of the applications mentioned in Figure 2.3 has driven the need for increasingly complex DSP systems which in turn has seen the growth of the research area involved in developing efficient implementation of some DSP algorithms.

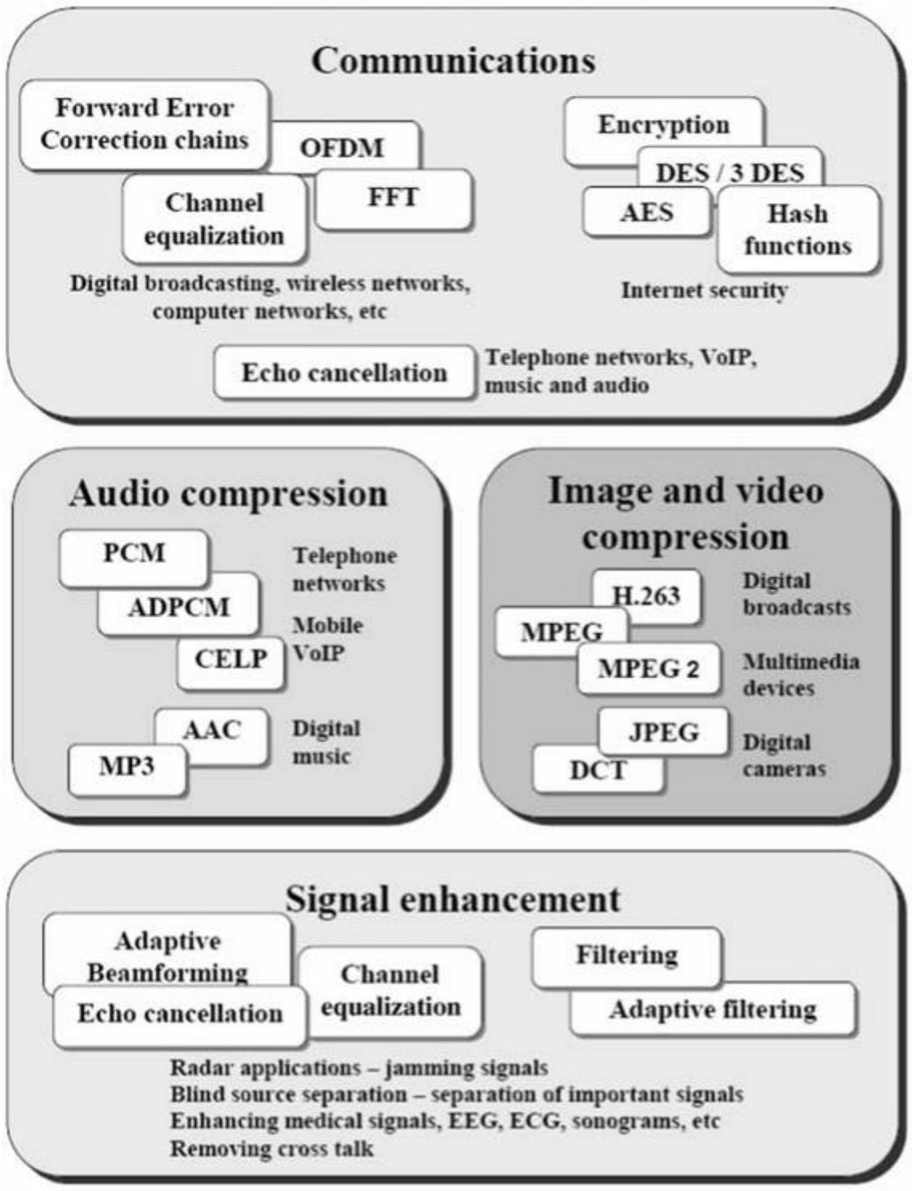


Figure 2.3 Some DSP applications

2.3.1 DSP System Definitions

The basic realization of DSP systems given in Figure 2.2, shows how a signal is digitized using an analogue-to-digital (A/D) converter, processed in a DSP system before being converted back to an analogue signal. The digitised signal is obtained as shown in Figure 2.4 where an analogue signal is converted into a pulse of signals and then quantized to a series of numbers. The input stream of numbers in digital format to the DSP system is typically labelled x(n) and the output is given as y(n). The original analogue signal can be derived from a range of source such as voice, music, medical or radio signal, a radar pulse or an image. Obviously, the representation of the data is a key aspect and this is considered in the next chapter. A wide range of signal processing can be carried out, as illustrated in Figure 2.3, as digitizing the signal opens up a wide domain of possibilities as to how the data can be manipulated, stored or transmitted.

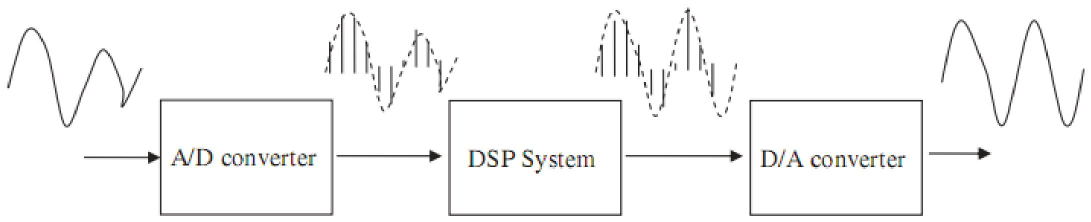


Figure 2.4 Basic DSP system

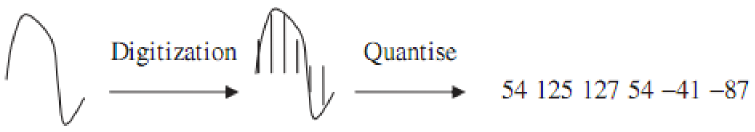


Figure 2.5 Digitization

A number of different DSP functions can be carried out either in the time domain, such as filtering, or operations in the frequency domain by performing an FFT (Rabiner and Gold 1975). The DCT forms the central mechanism for JPEG image compression which is also the foundation for the MPEG standards. This algorithm enables the components within the image that are invisible to the naked eye to be identified by converting the spatial image into the frequency domain. They can then be removed using quantization in the MPEG standard without a discernible degradation in the overall image quality. By increasing the amount of data removed, greater reduction in file size is achievable at a cost in image quality. Wavelet transforms offer both time domain and frequency domain information and have roles, not only in applications for image compression, but also for extraction of key information from signals and for noise cancellation. One such example is in extracting key features from medical signals such as the EEG.

Chapter 3

The Platform

Overview

In this chapter we present the platform on which the JPEG2000 compression standard will be implemented. First, a basic outline of Xilinx FPGA technologies is presented and then the chapter focuses on the Virtex-5 FPGA family.

3.1 Xilinx FPGA Technologies

The first FPGA was the Xilinx XC2000 family developed in 1982. The basic concept was to have programmable cells, connected to programmable fabric which in turn were fed by programmable I/O as illustrated by Figure 3.1. This differentiated Xilinx FPGAs from the early Altera devices which were PLD-based; thus the Altera FPGAs did not possess the same high levels of programmable interconnect. The architecture comprised cells called logic cells or LCs. The interconnect was programmable and was based on the 6-transistor SRAM cell given in Figure 3.2. By locating the cell at interconnections, it could then provide flexible routing by allowing horizontal-to-horizontal, vertical-to-vertical, vertical-to-horizontal and horizontal-to-vertical routing, to be achieved. The I/O cell had a number of configurations that allowed pins to be configured as input, output and bidirectional, with a number of interface modes.

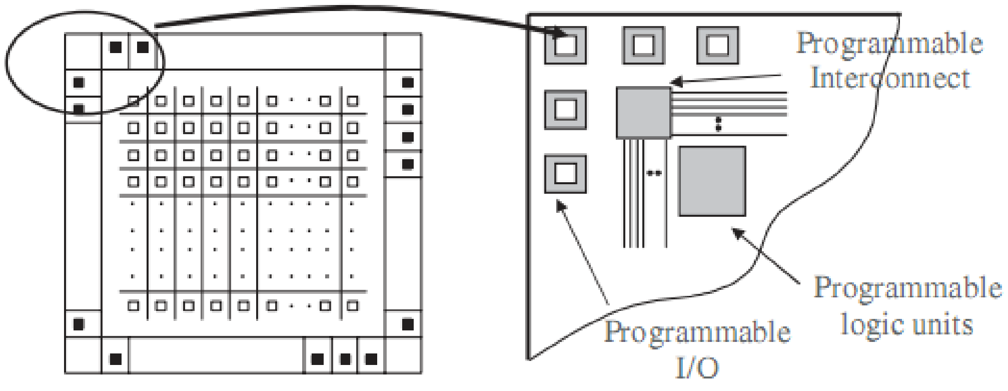


Figure 3.1 Early Xilinx FPGA technology

At this stage, FPGAs were viewed as glue logic devices with Moore's law providing a continual expansion in terms of logic density and speed. The device architecture continued largely unchanged from the XC2000 right up to the XC4000; for example,

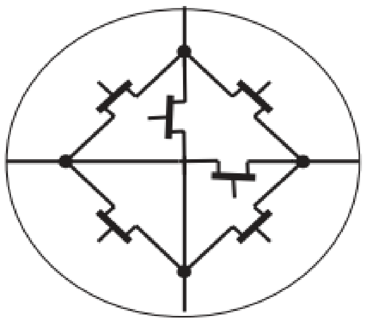


Figure 3.2 Xilinx FPGA SRAM Interconnect

the same LUT table size was used. The main evolution was the inclusion of the fast adder where manufacturers observed that, by including an additional multiplexer in the LE cell, a fast adder implementation could be achieved by mapping some of the logic into the fast carry adder logic, and some into the LUT. The principle is illustrated for the Virtex™ FPGA device in Figure 3.3. At this stage, the device was still being considered as glue logic for larger systems, but the addition of the fast adder logic started to open up the possibility of implementing a limited range of DSP systems, particularly those where multiplicative properties were required, but which did not require the full range of multiplicands. This formed the basis for a lot of early FPGA-based DSP implementation techniques.

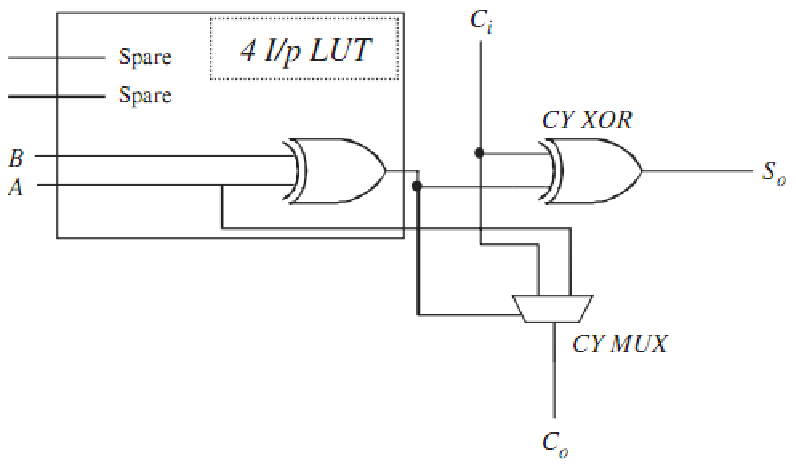


Figure 3.3 Adder implementation on Xilinx Virtex™ FPGA slice

At that time, a lot of FPGA products manufacturers faded away and there began a period defined as *accumulation* where FPGAs started to accumulate more complex components, starting with on-board dedicated multipliers, which appeared in the first Xilinx Virtex™ FPGA family (Figure 4.4), Power-PC blocks and gigabit transceivers with the Xilinx Virtex™ -II pro and Ethernet MAC with the Virtex™ -4. It can be seen from Figure 3.4, that the Xilinx FPGA was now becoming increasingly like a SoC with the main aim of the programmability to allow the connection together of complex processing blocks with the LCs used to implement basic logic functionality.

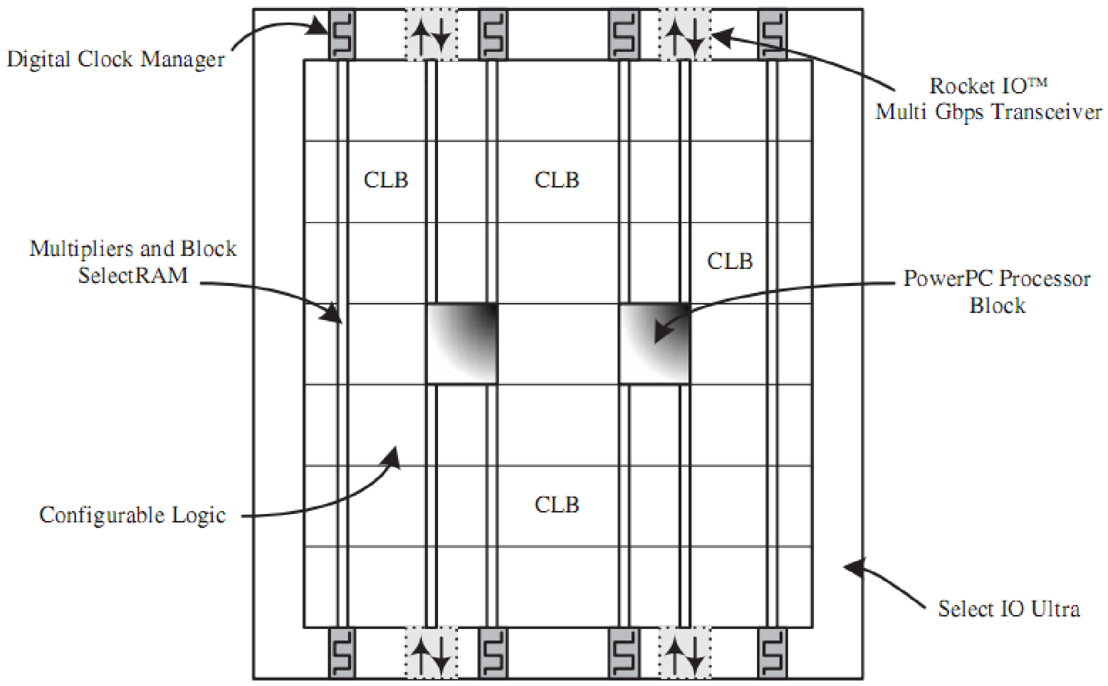


Figure 3.4 Virtex™-II Pro FPGA architecture overview

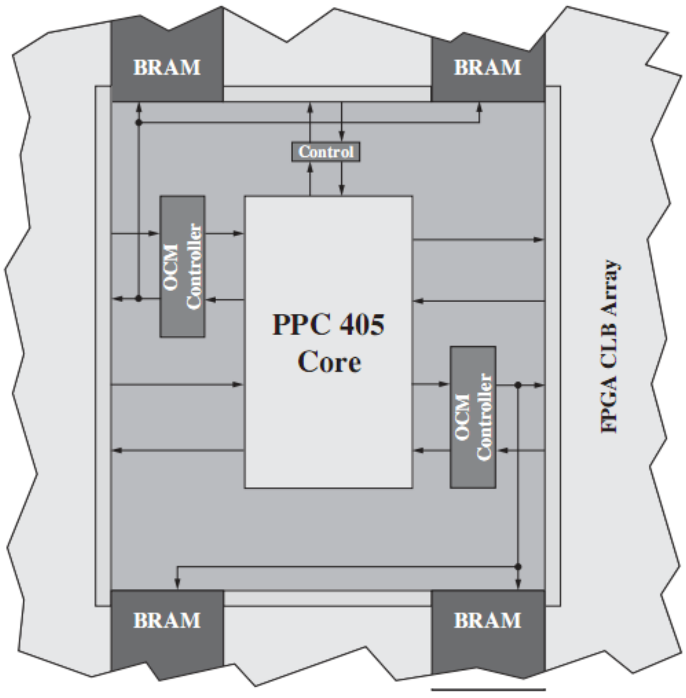


Figure 3.5 Power PC block architecture

The fabric now comprised the standard series of LCs, allowing functions to be connected as before, but now complex processing blocks such as 18-bit multipliers and PowerPC processors (Figure 3.5), were becoming commonplace. The concept of platform FPGA was now being used to describe recent FPGA devices to reflect this trend.

3.1.1 Xilinx Virtex™ -5 FPGA Technologies

The Virtex™ -5 comes in a variety of flavours, namely the LX which has been optimized for high-performance logic, the LXT which has been optimized for high-performance logic with low-power serial connectivity, and the SXT which has been optimized for DSP and memory-intensive applications with low-power serial connectivity. The Xilinx Virtex™ -5 family has a two speed-grade performance gain and is able to be clocked at 550MHz. It has a number of on-board IP blocks and a number of DSP48E slices which give a maximum of 352 GMACS performance. It also provides up to 600 pins, giving an I/O of 1.25Gbps LVDS and, if required, RocketIO GTP transceivers which deliver between 100Mbps and 3.2Gbps of serial connectivity. It also includes hardened PCI Express endpoint blocks and Tri-mode Ethernet MACs.

3.1.1.1 Virtex™ -5 Configurable Logic Block

The logic implementation in the Xilinx device is contained within configurable logic blocks or CLBs. Each CLB is connected to a switch matrix for access to the general routing matrix as shown in Figure 3.6 and contains a pair of slices which are organized into columns, each with an independent carry chain. For each CLB, slices in the bottom of the CLB are labeled as SLICE(0), and slices in the top of the CLB, are labelled as SLICE(1) and so on. Every slice contains four logic-function generators (or LUTs), four storage elements, wide-function multiplexers, and carry logic and so can be considered to contain four of the logic cell logic as given in Figure 3.7. In addition to this, some slices, called SLICEM, support two additional functions: storing data using distributed RAM and shifting data with 32-bit registers.

The basic logic cell configuration comprises a logic resource, a 6-input LUT connected to a single flip-flop, via a number of multiplexers, together with a circuit for performing fast addition. The basic logic cell has been designed to cope with the implementation of combinational and sequential logic implementations, along with some simple DSP circuits that use an adder.

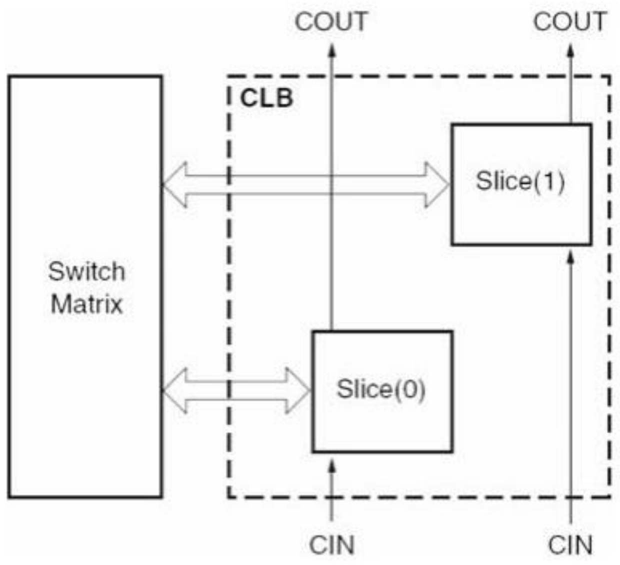


Figure 3.6 CLB Slices

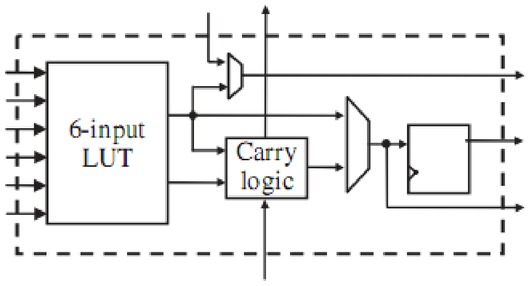


Figure 3.7 Logic cell functionality

The basic combination of LUT plus register has stayed with the Xilinx architecture, and has now been extended from a 4-input LUT in the Xilinx XC4000 series and Virtex™-5 series FPGA family to a 6-input LUT; this is a reflection of improving technology as governed by Moore's law. It is now argued in Xilinx Inc. (2007a) that a 6-input rather than a 4-input LUT which went all the way back to the study by Rose et al. (1990), now provides a better return on silicon area utilization for the critical path needed within the design. The combination of LUTs, flip-flops (FFs), and special functions such as carry chains and dedicated multiplexers, together with the ways by which these elements are connected, has been termed ExpressFabric technology.

The CLB can implement the following: a pure logic function by using the 6-input LUT logic and using the multiplexers to bypass the register; a single register using the multiplexers to feed data directly into and out of the register; and sequential logic circuits using the LUTs feeding into the registers. Scope is also provided to create larger combinational and sequential circuits, using the multiplexers to create large LUTs and registers. One special feature of the 6-input LUT is that it has two outputs. This allow the LUT to implement two arbitrarily defined, five-input Boolean functions, as long as these two functions share common inputs (Figure 3.8). This is an attempt to provide better utilization of the LUT resource when the number of inputs is smaller than six. This concept also allows the logic cell to implement a full adder, as shown in Figure 3.3 whilst at the same time, using the additional inputs and outputs to realize a 4-input LUT for some other function. This provides better utilization of the hardware in many DSP applications, where otherwise LUTs would be wasted to just provide a single gate implementation for an adder.

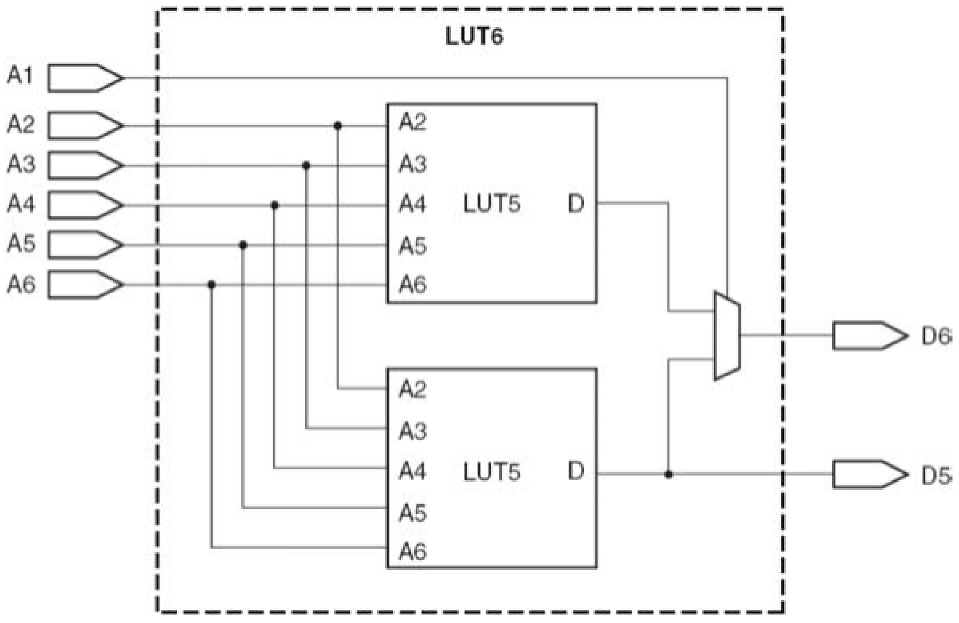


Figure 3.8 Arrangement of slices within the CLB

In this technology, the register resource is very flexible, allowing a wide range of storage possibilities ranging from edge-triggered D-type flip-flops to level-sensitive latches, all with a variety of synchronous and asynchronous inputs for clocks, clock enables, set/reset. The D input can be driven directly from a number of sources, including the LUT output, other D-type flip-flops and external inputs.

One of the advantages of the larger LUT in the Xilinx Virtex™-5 device is that it provides larger distributed RAM blocks and SRL chains. A sample of the various distributed memory configurations is given in Table 3.1 which gives the number of LUTs needed to create the various memory configurations listed. The distributed RAM modules have synchronous write resources, and can be made to have a synchronous read by using the flip-flop of the same slice. By decreasing the clock-to out delay, this will improve the critical path, but adds an additional clock cycle latency.

	No. of memory locations		
Memory type	32	64	128
Single-port	1 (1-bit)	1 (1-bit)	2 (1-bit)
Dual-port	2 (1-bit)	2 (1-bit)	4 (1-bit)
Quad-port	4 (2-bit)	4 (2-bit)	
Simple dual-port	4 (6-bit)	4 (3-bit)	

Table 3.1 Number of LUTs for various memory configurations

A number of memory configurations have been listed. For the single-port configuration, a common address port is used for synchronous writes and asynchronous reads. For the dual-port configuration, the distributed RAM has one port for synchronous writes and asynchronous reads, which is connected to one function generator and another port for asynchronous reads, which is connected to a second function generator. In simple dual-port configuration, there is no read from the write port. In the quad-port configurations, the concept is expanded by creating three ports for asynchronous reads, and three function generators plus one port for synchronous writes and asynchronous reads, giving a total of four functional generators.

The consideration of larger memory blocks is considered in the next section, but the combination of smaller distributed RAM, along with larger RAM blocks, provides the same memory hierarchy concept that was purported by the Altera FPGA, admittedly in different proportions. The LUT can also provide a ROM capability, and as Chapter 6 will illustrate, the development of programmable shift registers. The Virtex™-5 function generators and associated multiplexers some of which were highlighted in Figure 4.7, can implement one 4:1 multiplexers using one LUT, one 8:1 multiplexers using two LUTs etc.

3.1.1.2 Virtex™ -5 Memory Organization

In addition to distributed RAM, the Virtex™-5 device has a large number of 36kB block RAMs, each of which contain two independently controlled, 18 kB RAMs. The total memory configuration is given in Table 3.2. The 18 kB RAMs have been implemented in such a way, that the blocks can be configured to act as one 36 kB block RAM without the use of programmable interconnect. Block RAMs are placed in columns and can be cascaded to create deeper and wider RAM blocks. Each 18 kB block RAM, dual-port memory consists of an 18 kB storage area and two completely independent access ports along with other circuitry to allow the full expected RAM functionality to be achieved (Figure 3.9).

Memory type	Bits/block	No. of blocks	Suggested usage
Distributed RAM/slice	1024	14720	Shift registers, small FIFO buffers, filter FIFO buffers, filter delay lines
36 kbit Block RAM	18000	488	Multi-rate FIFO

Table 3.2 Virtex™-5 memory types and usage

The full definition in terms of access pins is given below, and represents a standard RAM configuration.

- A *clock* for each 18 kB block RAM which can be configured to have rising or falling edge. All input and output ports are referenced to the clock.
- An *enable* signal to control the read, write, and set/reset functionality of the port with an inactive enable pin, implying that the memory keeps the previous state.
- An additional enable signal called the *byte-wide write enable* signal which controls the writing and reading of the RAM in conjunction with the enable signal.

- *The register enable* pin which controls the optional output register.
- *The set/reset* pin which forces the data output latches to contain a set value.
- *The address bus* which selects the memory cells for read or write; its data bit width is decided by the size of RAM function chosen.

In latch mode, the read address is registered on the read port, and the stored data is loaded into the output latches after the RAM access time. When using the output register, the read operation will take one extra latency cycle. The write operation is also a single clock-edge operation with the write address being registered on the write port, and the data input is stored in memory. The additional circuitry highlighted in Figure 3.9, shows how inverted clock can be supported along with a registered output. The contents of the RAM can be initialized using the INIT parameter and can be indicated from the HDL source code.

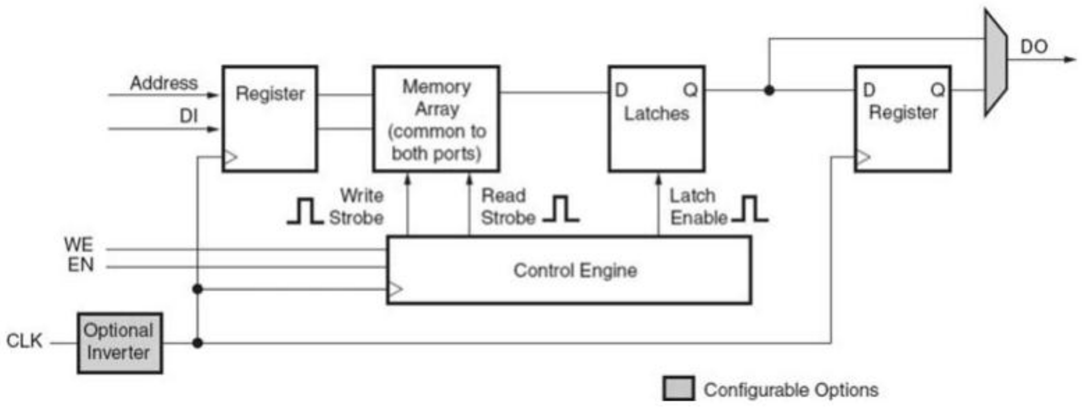


Figure 3.9 Block RAM logic diagram

Data width	Memory depth
1 (cascade)	32768 (65 536)
2	16384
4	8192
9	4096
18	2048
36	1024
72	512

Table 3.3 Memory sizes for Xilinx Virtex™-5 block RAM

The RAM provides a number of options for RAM configuration, some of which are listed in Table 3.3; the table shows how bit data width is traded off for memory depth, i.e. number of memory locations.

Dedicated logic has also been included in the block RAM enables, to allow the creation of synchronous or asynchronous FIFOs; these are important in some high-level design approaches, as will be seen later. This dedicated logic avoids use of the slower programmable CLB logic and routing resource, and generates the necessary hardware for the pointer write and read generation along with the setting of the various flags associated with FIFOs. A number of FIFO sizes can be inferred, including 8KX4, 4KX4, 4KX9, 2KX9, 2KX18, 1KX18, 1KX36, 512X36 and 512X72.

3.1.1.3 Virtex™-5 DSP Processing Resource

In addition to the scalable adders in the CLBs, the Virtex™-5 also provides a dedicated DSP processing block called DSP48E. The Virtex™-5 can have up to 640 DSP48E slices which are located at various positions in the FPGA, and supports many independent functions including multiply, MAC, multiply add, three-input add, barrel shifting, wide-bus multiplexing, magnitude comparator, bit-wise logic functions, pattern detect, and wide counter. The architecture also allows the multiple DSP48E slices to be connected together to form a wider range of DSP functions, such as DSP filters, correlators and frequency domain functions.

A simplified version of the DSP48E processing block is given in Figure 4.10. The basic architecture of the DSP48E block is a multiply–accumulate core, which is a very useful engine for many DSP computations. However, in addition to the basic MAC function, the DSP48E block also allows a number of other modes of operation, as summarized below:

- 25-bit x 18-bit multiplication which can be pipelined
- 96-bit accumulation or addition or subtracters (across two DSP48E slices)
- triple and limited quad addition/subtraction
- dedicated bitwise logic operations
- arithmetic support for overflow/underflow

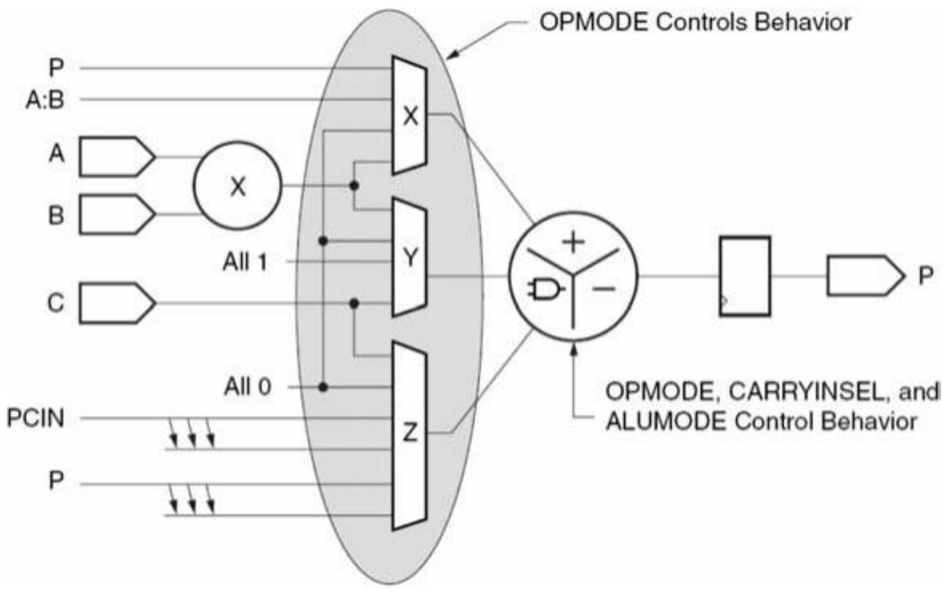


Figure 3.10 DSP processing blocks called DSP48E

Each DSP48E slice has a 25-bit X18-bit multiplier which is fed from two multiplexers; the multiplexers accept a 30-bit A input and a 18-bit B input either from the switching matrix or from the DSP48E directly below. These can be stored in registers (not shown in Figure 3.10) before being fed to the multiplier. Just before multiplication, the A signal is split and only 25 bits of the signal are fed to the multiplier. A fast multiplier technique is employed which produces an equivalent 43-bit two's complement result in the form of two partial products, which are then sign-extended to 48 bits in the X multiplexer and Y multiplexer respectively before being fed into three input adder/subtracter for final summation.

Many fast multipliers work on the concept of using fast carry-save adders to eventually produce a final sum and carry signals, and then using a fast carry ripple to perform the final addition. This final addition is costly, either in terms of speed or if a speed-up technique is employed, then area. By postponing the addition to the ALU stage, a two-stage addition can then be avoided for multiply–accumulation, by performing a three-stage addition to compute the final multiplication output and an addition for the accumulation input in one stage. Once again, for flexibility, the adder/subtracter unit has been extended to function as a arithmetic logic unit (ALU), thereby providing more functionality at little hardware overhead. As the final stage of the conventional multiplication is being performed in the second-stage adder, a three-input addition is required with the third input used to complete the MAC operation if required.

The multiplexers allow a number of additional levels of flexibility to be added. For example, the P input can be used to feed in an input either from another DSP48E block from below using the PCIN in the Z multiplexer or looped back from the current DSP48E block say, for example, if a recursion is being performed using the P input to the Z multiplexer. The multiplier can be bypassed if not required, by using the A:B input which is a concatenation of the two input signals A and B, 25-bit and 18-bit words respectively; this gives a 43-bit word size which is the same as the multiplier output. Provision to initialize the inputs to the ALU to all 0s or all 1s, is also provided. To increase the flexibility of the unit, the adder can also be split into several smaller adders, allowing two 24-bit additions or four 12-bit additions to be performed. This is known as the SIMD mode, as a single operation namely addition, is performed on multiple data, thus giving the SIMD operation. The DSP48E slice also provides a right-wire-shift by 17, allowing the partial product from one DSP48E slice to be shifted to the right and added to the next partial product, computed in an adjacent DSP48E slice. This functionality is useful, when the dedicated multipliers are used as building blocks, in constructing larger multipliers. The diagram in Figure 4.10 is only basic, and does not indicate that other signals are also provided, in addition to the multiply or multiply–accumulate output, P.

From a functional perspective, the synthesis tools will largely hide the detail of how the design functionality is mapped to the FPGA hardware, but it is important to understand that the level of functionality that is available as it determines the design approach the user will adopt. A number of detailed examples are listed in the relevant user guide (Xilinx Inc. 2007c), indicating how performance can be achieved.

3.1.1.4 Clock Networks and PLLs

The Xilinx Virtex™-5 FPGA family can provide a clock frequency of 550MHz. The clock domains in the Virtex™-5 FPGA are organized into six clock management tiles or CMTs, each of which contain two digital clock managers (DCMs) and one PLL. In total, the FPGA has eighteen total clock generators.

A key feature of the Xilinx Virtex™-5 FPGA is the DCM, which provides a wide range of powerful clock management features including a delay-locked loop (DLL);this acts to align the incoming clock to the produced clock as described earlier. It also allows a range of clock frequencies to be produced, including a doubled frequency a range of fractional clock frequencies of the input clock. Coarse (90°, 180° and 270°) fine-grained phase shifting and various types of fine-grained or fractional phase-shifting are supported.

The PLL's main purpose is to act as a frequency synthesizer and to remove jitter from either external or internal clocks, in conjunction with the DCMs. With regard to clock generation, the six PLL output counters are multiplexed into a single clock signal for use as a reference clock to the DCMs. Two output clocks from the PLL can drive the

DCMs; for example, one could drive the first DCM while the other could drive the second DCM. Flexibility is provided to allow the output of each DCM output to be multiplexed into a single clock signal, for use as a reference clock to the PLL, but one DCM can be used as the reference clock to the PLL at any given time.

3.1.1.5 I/O and External Memory Interfaces

Virtex™-5 FPGA supports a number of different I/O standard interfaces termed SelectIOTM drivers and receivers, allowing control of the output strength and slew rate and on-chip termination. As with the Altera FPGA, the I/Os are organized into a bank comprising 40 IOBs which covers a physical area that is 20 CLBs high, and is controlled by a single clock. The Virtex™-5 FPGA also includes digitally controlled impedance (DCI) technology, allowing the output impedance or input termination to be adjusted, and therefore, accurately match the characteristic impedance of the transmission line. The need to effectively terminate PCB trace signals, is becoming an increasing important issue in high-speed circuit implementation, and this approach purports to avoid the need to add termination resistors on the board. A number of standards are supported, including low-voltage transistor–transistor logic (LVTTL), low-voltage complementary metal oxide semiconductor (LVCMOS), peripheral component interface (PCI) including PCIX, PCI33, PCI66, and low-voltage differential signalling (LVDS), to name but a few.

Input serial-to-parallel converters (ISERDES) and output parallel-to-serial converters (OSERDES) are also supported. These allow very fast external I/O data rates such as SDR and DDR, to be fed into the internal FPGA logic which may be running an order of magnitude slower. This is essentially a serial-to-parallel converter with some additional hardware modules that allow reordering of the sequence of the parallel data stream going into the FPGA fabric, and circuitry to handle the strobe-to-FPGA clock domain crossover.

Chapter 4

The Application

Overview

As seen in the second chapter, image and video compression is a major part of the domain of DSP applications. In this chapter some of the basic principles of image compression are introduced. Special focus is given of course in comparing two widely used and closely related compression methods: JPEG and JPEG2000; the last being the application that was implemented for the purposes of this thesis. The JPEG2000 algorithm is described more analytically in order to provide a more in-depth background around this particular standard.

4.1 Introduction

Modern computers employ graphics extensively. Window-based operating systems display the disk's file directory graphically. The progress of many system operations, such as downloading a file, may also be displayed graphically. Many applications provide a graphical user interface (GUI), which makes it easier to use the program and to interpret displayed results. Computer graphics is used in many areas in everyday life to convert many types of complex information to images. Thus, images are important, but they tend to be big! Modern hardware can display many colors, which is why it is common to have a pixel represented internally as a 24-bit number, where the percentages of red, green, and blue occupy 8 bits each. Such a 24-bit pixel can specify one of 224 ≈ 16.78 million colors. As a result, an image at a resolution of 512×512 that consists of such pixels occupies 786,432 bytes. At a resolution of 1024×1024 it becomes four times as big, requiring 3,145,728 bytes. Videos are also commonly used in computers, making for even bigger images. This is why image compression is so important. An important feature of image compression is that it can be lossy. An image, after all, exists for people to look at, so, when it is compressed, it is acceptable to lose image features to which the eye is not sensitive.

4.2 JPEG

If you have ever built a web-page, taken photos with a digital camera or generally worked with digital images, then it's likely you have had contact with JPEG image compression. This compression standard was developed by the Joint Photographic Experts Group, whose "JPEG" abbreviation has become synonymous with the standard itself. JPEG is a sophisticated lossy/lossless compression method for color or grayscale still images (not videos). It does not handle bi-level (black and white) images very well. It also works best on continuous-tone images, where adjacent pixels have similar colors. An important feature of JPEG is its use of many parameters, allowing the user to adjust the amount of the data lost (and thus also the compression ratio) over a very wide range. Often, the eye cannot see any image degradation even at compression factors of 10 or 20. There are two operating modes, lossy (also called baseline) and lossless (which typically produces compression ratios of around 0.5). Most implementations support just the lossy mode.

4.3 Why JPEG2000?

The image compression field is very active, with new approaches, ideas, and techniques being developed and implemented all the time. JPEG is widely used for image compression but is not perfect. The use of the Discrete Cosine Transform (DCT) on 8×8 blocks of pixels results sometimes in a reconstructed image that has a blocky appearance (especially when the JPEG parameters are set for much loss of information). Despite the success of JPEG in the 1990s, a growing number of new applications such as high-resolution imagery, high-fidelity color imaging, multimedia and Internet applications etc., require additional, enhanced functionalities from a compression standard that JPEG cannot satisfy due to some of its inherent shortcomings and design points that were beyond the scope of JPEG when it was developed. This is why the JPEG committee has decided, as early as 1995, to develop a new, wavelet-based standard for the compression of still images, to be known as JPEG 2000 (or JPEG Y2K). Perhaps the most important milestone in the development of JPEG2000 occurred in December 1999, when the JPEG committee met in Maui, Hawaii and approved the first committee draft of Part 1 of the JPEG 2000 standard. At its Rochester meeting in August 2000, the JPEG committee approved the final draft of this International Standard. In December 2000 this draft was finally accepted as a full International Standard by the ISO and ITU-T. This standard specifies the creation of a new image coding system for different types of still images (bilevel, gray level, color, multicomponent), with different characteristics (natural images, scientific, medical, remote sensing, rendered graphics, etc.) allowing different imaging models (client/server, real-time transmission, image library archival, etc.) preferably within a unified system. The standard could be used on a royalty and fee-free basis. This was important for the standard to become widely accepted, in the same manner as the original JPEG is now.

The markets and applications better served by this standard are numerous, from multimedia devices (e.g., digital cameras, PDAs, 3G cell phones, scanners, printers etc.) and client/server communication (the internet), to many other specific applications such as military/surveillance and medical imagery.

In order to have a first understanding of the most eye-catching difference between the JPEG and JPEG2000 compression methods, one can compare the two compressed images in Figure 4.1.

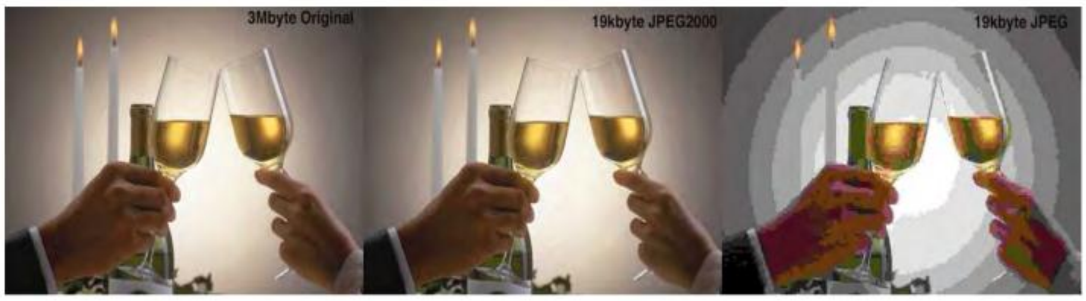


Figure 4.1 JPEG2000 vs. JPEG

Figure 4.1 shows an example of the superior performance of JPEG2000 over JPEG at very high compression ratios. JPEG2000 (middle) shows almost no quality loss from the original image, even at 158:1 compression ratio.

The basic idea of JPEG2000 can be clearly illustrated in Figure 4.2. The idea is to compress an image once and decode the encoded bitstream in many different ways to fulfill various application requirements. This is a general concept not found in JPEG.

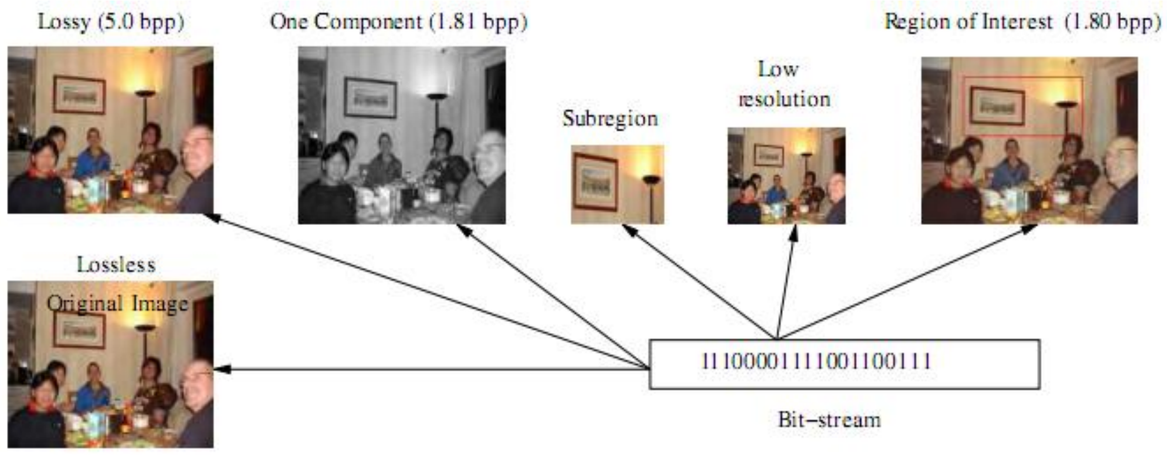


Figure 4.2 JPEG2000 features

In order to become more specific, regarding the advantages of JPEG2000 over JPEG, we must point out some special characteristics of the JPEG2000 standard:

- *Superior compression performance:* At high bit rates, where artifacts become just imperceptible, JPEG2000 has a compression advantage over JPEG by roughly 20% on average. At lower bit-rates, JPEG2000 has a much more significant advantage over certain modes of JPEG. The compression gains over JPEG are attributed to the use of Discrete Wavelet Transform and more sophisticated entropy encoding scheme.
- *Multiple resolution representation:* JPEG2000 provides seamless compression of image components each from 1 to 16 bits per component sample. With tiling, it can handle large image sizes in a single codestream.
- *Progressive transmission:* it provides efficient codestream organizations which are progressive by pixel accuracy or by quality (SNR) and also by resolution and size.
- *Lossless and lossy compression:* JPEG2000 provides both lossless and lossy modes form a single compression architecture with the use of an integer –and thus reversible- wavelet transform.
- *Random codestream access and processing (Region of Interest):* the standard’s codestreams offer several mechanisms to support spatial random access or region of interest access at carrying degrees of granularity.

- *Error resilience:* JPEG2000 is robust to bit errors introduced by noisy communication channels such as wireless. This is accomplished by the inclusion of resynchronization markers, the coding of data in small independent blocks, and the use of special error spotting mechanism within each block.

- *Sequential build-up capability:* JPEG2000 allows for encoding of an image from top to bottom in a sequential fashion. Thus, there is no need to buffer the entire image.

- *Flexible file format:* the existing file formats (JP2 and JPX) allow for handling of color-space information, metadata, and for interactivity in networked applications.

4.4 The JPEG2000 Compression Engine

The JPEG2000 compression engine (encoder and decoder) is illustrated in block diagram form in Figure 4.3.

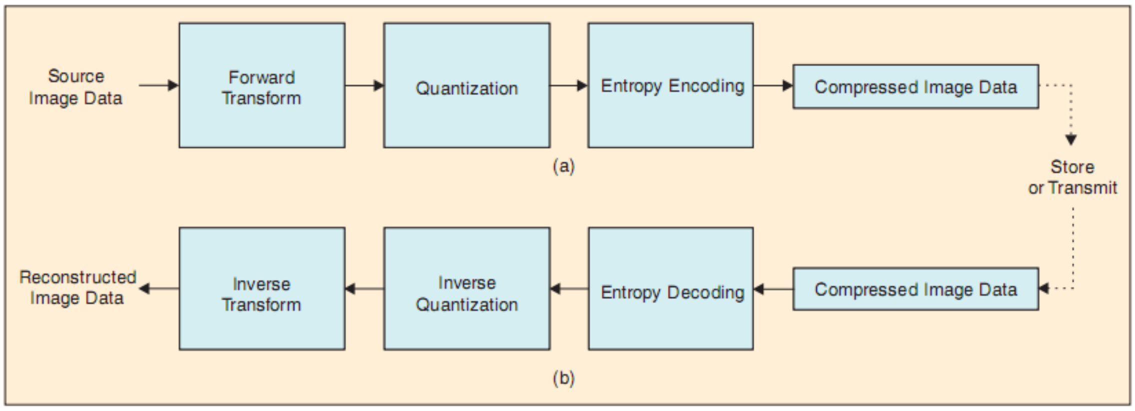


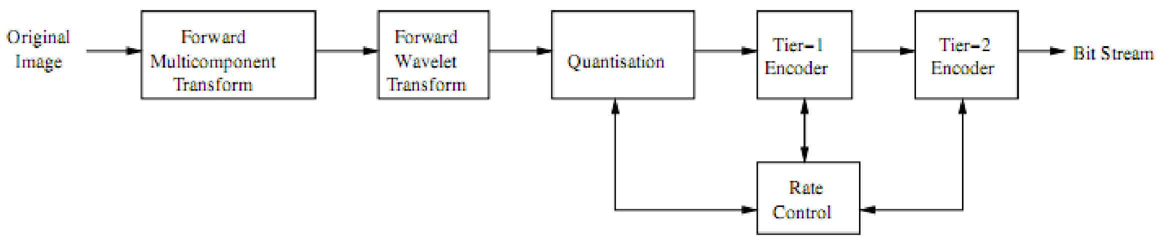
Figure 4.3 General block diagram of the JPEG 2000 (a) encoder and (b) decoder.

At the encoder, the discrete transform is first applied on the source image data. The transform coefficients are then quantized and entropy coded before forming the output code stream(bit stream). The decoder is the reverse of the encoder. The code stream is first entropy decoded, dequantized, and inverse discrete transformed, thus resulting in the reconstructed image data. Although this general block diagram looks like the one for the conventional JPEG, there are radical differences in all of the processes of each block of the diagram. A quick overview of the whole system is as follows:

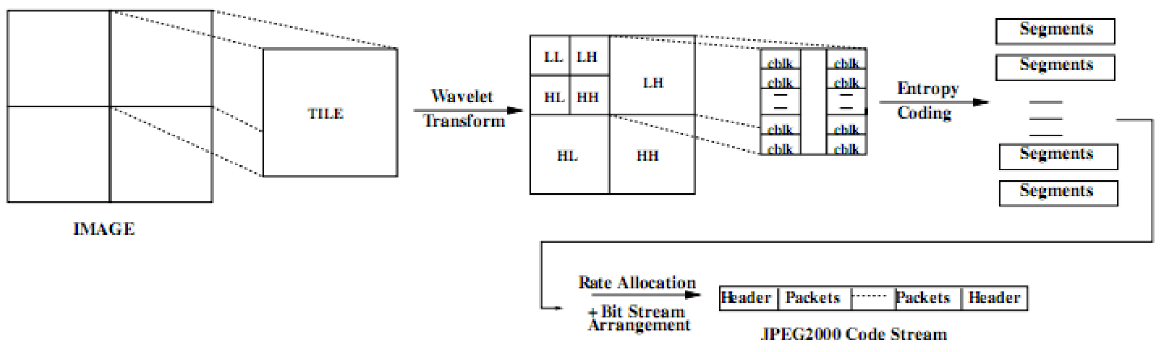
- The source image is decomposed into components.
- The image components are (optionally) decomposed into rectangular tiles. The tile-component is the basic unit of the original or reconstructed image.
- A wavelet transform is applied on each tile. The tile is decomposed into different resolution levels.
- The decomposition levels are made up of subbands of coefficients that describe the frequency characteristics of local areas of the tile components, rather than across the entire image component.
- The bit planes of the coefficients in a code block (i.e., the bits of equal significance across the coefficients in a code block) are entropy coded.
- The encoding can be done in such a way that certain regions of interest can be coded at a higher quality than the background.
- Markers are added to the bit stream to allow for error resilience.
- The code stream has a main header at the beginning that describes the original image and the various decomposition and coding styles that are used to locate, extract, decode and reconstruct the image with the desired resolution, fidelity, region of interest or other characteristics.

4.4.1 Encoder Functionality

Figure 4.4 shows the structure of the JPEG2000 encoder and its related coding steps more analytically. The Forward Transform and Entropy Encoding are split into more basic components.



(a) Blocks in JPEG2000 Encoder



(b) Coding Steps in JPEG2000

Figure 4.4 Structure of JPEG2000 Encoder

Considering an image with multi-component segments, the encoding blocks performs the following functions below:

- **Forward multi-component transform (Inter/Intra-component transform):** refers to the mapping of an image data from the RGB color space to the YCrCb color space. In lossy coding the transform is irreversible (real-to-real), whereas in lossless coding it is reversible (integer-to-integer).
- **Forward Discrete Wavelet Transform (FDWT):** is a transform that analyzes a tile (image) component to decompose it into a number of subbands at different levels of resolution. Two-dimensional DWT (2D DWT) is performed by applying one-dimensional DWT row-wise and then column-wise in each component (Figure 4.4 (b)). The first level of decomposition results to the creation of four subbands LL1, HL1, LH1, and HH1. The low-pass subband (LH1) represents a 2:1 subsample in both vertical and horizontal dimensions, thus a low resolution version

of the original component. The LL1 subband can again be analyzed to produce four subbands LL2, HL2, LH2, and HH2. The higher level of decomposition may continue in a similar fashion. . In lossy coding the transform is irreversible (real-to-real), whereas in lossless coding it is reversible (integer-to-integer). The DWT will be further analyzed later in this thesis.

- **Quantization (Uniform quantizer with Dead-zone):** all the subbands are quantized in lossy compression mode in order to reduce the precision of the subbands to aid in the achieving of compression. The block quantizes transform coefficients with dead-zone scalar quantizer. Dead-zone scalar quantizer with step size b means the width of the central quantization around the origin is 2b. In lossless coding the quantizer steps are forced to 1, thereby no actual quantization takes place.

- **Tier-1 Encoder:** this is an entropy encoding process, particularly a combination of MQ-Coder and EBCOT. MQ-Coder is a form of arithmetic mean coder and EBCOT is a form of bit plane coding, abbreviated as Embedded Block Coding with Optimized Truncation. In tier-1 coding, quantizer indices related to each subband are partitioned into fixed-size code blocks to produce an embedded codestream using bit-plane coding. This algorithm (EBCOT) has benn built to exploit the symmetries and redundancies within and across the bit-planes so as to minimize the statistics to be maintained and minimize the coded bitstream that MQ Coder would generate.

  The bit-plane coding scheme functions as follows. If the precision of the elements in the code-block is p, then the code-block is decomposed into p bit-planes and they are encoded from the most significant bit-plane to the least significant bit-plane sequentially. Each bit-plane is scanned in a particular scan pattern as shown in Figure 4.5. The scan pattern can be divided into sections (or stripes), each with four consecutive rows starting from the first row of a code-block. If the total number of rows of a code-block is not a multiple of 4, all the sections will have four consecutive rows except the very last section.

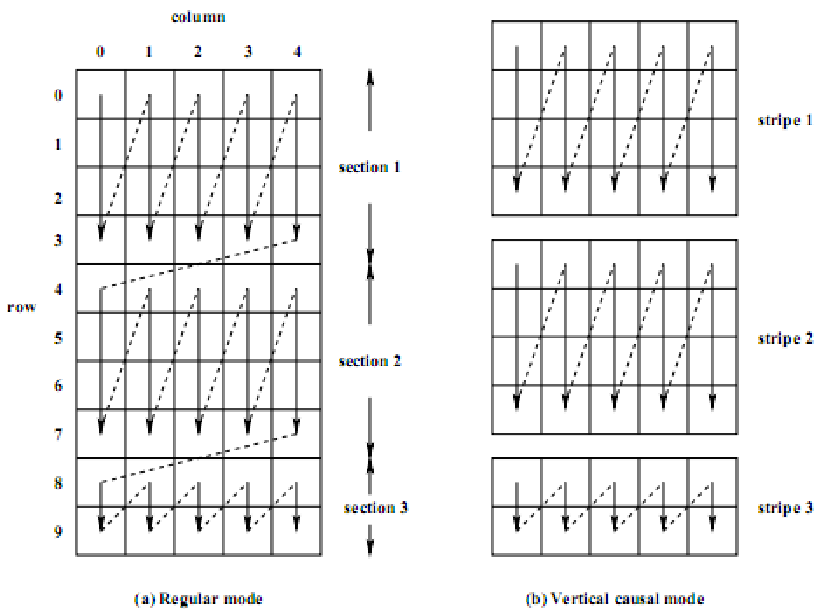


Figure 4.5 Scan pattern of each bit-plane in a code-block:

(a) regular mode;

(b) vertical causal mode

There are three passes per bit plane:

- **significant propagation pass (SPP):** During SPP, a bit is coded if its location is not significant, but at least one of its eight-connected neighbors is significant. By significant, we mean that the bit is most significant bit of the corresponding sample in the code-block.
- **magnitude refinement pass (MRP):** All the bits that have not been coded in SPP and became significant in a previous bit-plane are coded in this pass.
- **cleanup pass (CUP):** All the bits that have not been coded in either SPP or MRP are coded in this pass. CUP also performs a form of run-length coding to efficiently code a string of zeros.

The symbols produced by the bit-plane encoder are coded using an adaptive binary arithmetic coder. Optionally, arithmetic coding can be bypassed for some symbols produced during processing of the less significant bit-planes.

- **Tier-2 Encoder (Bit-stream organization):** Bit plane coding passes are included for each block and the order of appearance of these in the final code stream are encoded along with the actual coding pass of data. Only subsets of the coding passes are included in the bit stream. Rate control is achieved through both the choice of quantizer step sizes and also the selection of the subset of coding passes to include in the final code stream.

4.4.2 Decoder Functionality

Each functional block in the decoder either exactly or approximately inverts the effects of its corresponding block in the encoder. The decoder structure essentially mirrors that of the encoder. Hence, with the exception of rate control, there is a one-to-one correspondence between functional blocks in the encoder and decoder. Figure 4.6 shows the structure of the JPEG2000 decoder, while Figure 4.7 shows its conceptual spatial and bit stream representations.

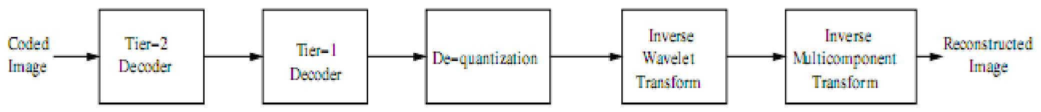


Figure 4.6 Blocks in JPEG2000 Decoder

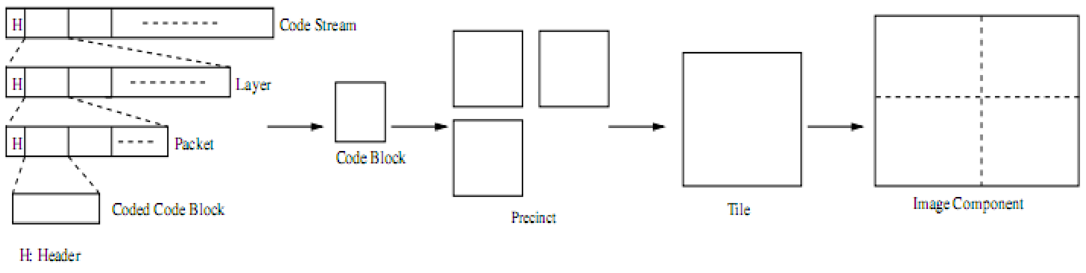


Figure 4.7 Spatial and Bit Stream Representations form Tier-2 to Component Transform blocks

- **Tier-2 decoder:** The bit-plane coding passes for the various code blocks are extracted from the code stream. Except in the lossless case, not all coding passes for all code blocks are guaranteed to be present.

- **Tier-1 Decoder:** The bit-plane coding passes for each of the code blocks is decoded, yielding the reconstructed quantizer indices. In the lossy case, not all of the bit-plane coding passes are typically present in the code stream, in which case the reconstructed quantizer indices are themselves only approximations to the original quantizer indices.

- **Dequantizer:** The quantized transform coefficient values are obtained from the reconstructed quantizer indices. In the case of lossless coding, the transform coefficients are the same as the quantizer indices.

- **Inverse Discrete Wavelet Transform (IDWT):** The inverse transform is applied –if need- to the data for each component.

- **Inverse multicomponent transform:** The inverse multicomponent transform is applied to the image data. If the sample values for a component are unsigned, the original dynamic range is restored by adding bias. In the case of lossy coding, a clipping operation is performed on the sample to ensure that they do not exceed their allowable range.

Chapter 5

HW/SW Partitioning

Overview

The purpose of this chapter is to map the computational parts of the JPEG2000 codec onto the components of the system architecture. First a software-based open-source implementation of JPEG2000 is selected and a timing analysis is performed. The profiling data are then analyzed in order to construct a Hardware/Software partitioning solution for the co-processing architecture.

5.1 Software implementation selection

A number of software implementations of JPEG2000 were assessed in order to determine which would be the most suitable basis for a co-processing system. Three implementations with available source-code were assessed – JJ2000, Kakadu and JasPer. Of these three, JasPer was chosen as best fitting the selection criteria imposed by the project.

5.1.1 Selection Criteria

The selection of a software implementation was based on several selection criteria. The most fundamental of these was availability of source code. Using the software for a co-processing implementation of JPEG2000 requires modification of the software codec and source code access was therefore necessary. The three implementations considered all fulfilled this fundamental prerequisite.

Cost was a consideration in software selection. Preference was given to implementations that were available at low-cost or free of charge.

The architecture of each implementation was assessed, whenever these details were made available. Along with the architecture, consideration was given to the programming language in which the code was written. A related criterion to the architecture was the overall complexity of the implementation. These factors were of importance due to the need to make modifications to allow co-processing. The more complex the internal structure of the software, the greater the difficulty in modifying the code and the greater the scope for potential problems during development.

5.1.2 JJ2000 Implementation

JJ2000 has been developed in a joint effort between Canon Inc., Ecole Polytechnique Fédérale de Lausanne and Ericsson Inc. The source code for the software is made freely available from the project's website. JJ2000 is written in Java. For the purposes of hardware coprocessing, this is a severe limitation. Being Java software, the code runs on a Java virtual machine. This introduces two key problems. Firstly, the software is slower to execute than an implementation that is native to a PC. More importantly, there is the issue of portability. Portability is a major strength of the Java language. However, providing hardware coprocessing for the software by its very nature ties the software to a specific hardware and operating system architecture. Portability of the software between platforms is therefore a less important issue for this thesis. For these reasons, JJ2000 is not a suitable implementation upon which to base a co-processing JPEG2000 system.

5.1.3 Kakadu Implementation

Kakadu is a comprehensive JPEG2000 software toolkit developed by Dr. David Taubman of the University of New South Wales, Australia. The Kakadu software includes a Dynamic Link Libary (DLL) of core routines that provide JPEG2000 compression and decompression. A set of basic utilities that make use of the core DLL is also provided. Kakadu is written in C++ and is presented as a well-designed system that seeks to provide an efficient implementation of the standard. It appears to be a complex framework, with a great deal of functionality included. An individual non-commercial license for Kakadu costs US$100, while a multi-user non-commercial license costs US$500. Commitment to good design and performance are key advantages offered by Kakadu. However, it is the only implementation of the three considered that is available in exchange for payment. Additionally, the substantial array of features it offers is beyond the scope of the thesis and would mostly remain unused.

5.2.4 JasPer Implementation

Image Power Inc. and the University of British Columbia developed the JasPer implementation of JPEG2000. The software's chief architect, Michael Adams, was also involved in the JPEG2000 standardization process. The JasPer code is available free of charge and can be downloaded from the project's web site.

JasPer is written in C code. As a language, C is relatively low-level and is therefore well-suited to interfacing with co-processing hardware. The software is set up as a static library, suitable for compilation on most platforms that support the C language. The project is especially targeted for Windows and Unix-type systems. The JasPer library allows an input image to be compressed using JPEG2000, and for already compressed JPEG2000 files to be decompressed back again. A number of image formats are supported for input images files. In addition to the static library, JasPer includes a basic command-line utility program, which provides user access to the library functionality.

The JasPer software has been designed in a reasonably modular manner, although in places the code is somewhat obscure and lacks commenting. It does not appear to have been as well designed as Kakadu. However, it is more than adequate as a basic program for compressing and decompressing images using the JPEG2000 standard. This fact, combined with being written in C and available free of charge, led to JasPer being selected as the software implementation of JPEG2000 that would be used for the basis of coprocessing work.

The version of JasPer used in this thesis was JasPer 1.500.3. The JasPer manual states that the software consists of about 40,000 lines of C code in total . Development work was carried out on a PC running Windows Vista. The JasPer software was compiled using the Microsoft Visual C++ version 8.0 (MSVC) compiler. The project and workspace files necessary for compilation under MSVC were provided with the JasPer distribution.

5.2 Profiling Background and Motivation

The code for nearly all non-trivial programs is split into multiple functions, often across multiple source code files. Software 'profiling' is a general term for the analysis of which 'sections' of a program are actually executed, the number of times each section is executed and how long each section takes to run. This analysis is produced using dedicated profiling tools.

Modifying JasPer to allow it to take advantage of co-processing requires replacing some of the JasPer code with processing performed by the hardware. The overall speed-up of the hybrid system over the software-only system is maximized when the most computationally intensive modules in JasPer are replaced by the FPGA. Thus the first aim of software profiling was to determine which parts of the JasPer software take the longest time to execute.

The second aim of profiling was to use the profiling data generated in order to decide which routines in the JasPer code were best suited to execution on the FPGA.

Three criteria were used in this decision. The first of these was obviously to select JasPer routines with large execution times. However, not all such software routines make sense to implement on an FPGA. Routines in this category include I/O transfers to and from disk as well as routines that require frequent and random access to large sections of the PC's main memory. For this reason, the second criterion was that the routines selected could indeed benefit from FPGA processing. Thirdly, it was desired that the routines selected were as far as possible a modular section of the overall JasPer compression utility. The motivation for this criterion was to reduce the frequency with which data would need to be transferred to and from the Virtex™-5 FPGA. Less modular sections of the program would require frequent interaction with other parts of JasPer. Consequently, more frequent bus transfers of data would become necessary. Bus transfers should be considered as overhead in the co-processing system, since during the transfer time the FPGA will be unlikely to be performing any computation. This overhead can be minimized by selecting a modular section of JasPer for hardware processing.

5.3 Profiling Strategy

As discussed above, the results of profiling JasPer were used in order to determine which part of the algorithm the hardware processing would implement. It was therefore important that the profiles generated were an accurate representation of JasPer's execution.

It was also important that profiling information be obtained for a broad range of compression scenarios. In this way, it was possible to isolate the computationally intensive parts of the software under a wide variety of input conditions.

5.3.1 Scenarios

The JPEG2000 algorithm's flexibility provides for the user to specify a number of compression options, thus presenting a wide range of compression scenarios. It was desired that profiling would take the most significant of these options into account. Furthermore, different image types, sizes and characteristics were taken into consideration. Table 4.1 lists all the factors that were taken into account.

- ✓ **Lossy vs. Lossless compression**
- ✓ **Image size**
- ✓ **Encoder vs. Decoder**
- ✓ **Grayscale vs. Color Images**

Table 5.1 Profiling scenario factors

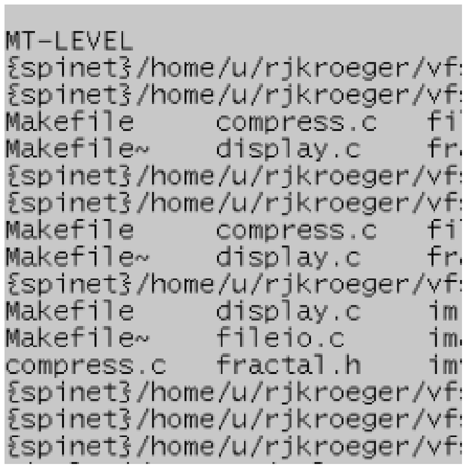
5.3.2 Test Images

In order to perform profiling, four different images were used as input to the JPEG2000 compression engine. These test images were passed to JasPer in Portable Pixel Map/Portable Gray Map (PPM/PGM) format, and are presented in Figure 5.1.

“Lena” (256x256, 8-bit grayscale)

“Airplane” (512x512, 8-bit grayscale)

“Peppers” (512x512, 24-bit color)



```
MT-LEVEL
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
Makefile      compress.c   fi
Makefile~     display.c    fr
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
Makefile      compress.c   fi
Makefile~     display.c    fr
{spinet}/home/u/rjkroeger/vf
Makefile      display.c    im
Makefile~     fileio.c     im
compress.c    fractal.h    im
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
```

“Text” (256x256, 8-bit grayscale)

Figure 5.1 Test Images

5.3.3 Profiling Testcases

In order to estimate as adequately as possible the impact of the previously presented performance factors, 6 different testcase scenarios have been followed. For each testcase scenario, four profiles were generated. In each scenario certain input variables have been held constant, while one or two other variables were modified between individual profiles. The following table (Table 5.2) provides a detailed description for each scenario.

Profile Scenario 1: ***Image sizes vs. other performance factors***	***Constants:*** • *Encoding process* • *Lossy Compression* • *Color Image*	***Variable:*** • ***Image size***
Profile Scenario 2: ***Image sizes vs. other performance factors***	***Constants:*** • *Decoding process* • *Lossy Compression* • *Color Image*	***Variable:*** • *Image size*
Profile Scenario 3: ***Image sizes vs. other performance factors***	***Constants:*** • *Encoding process* • *Lossless Compression* • *Color Image*	***Variable:*** • *Image size*
Profile Scenario 4: ***Image sizes vs. other performance factors***	***Constants:*** • *Decoding process* • *Lossless Compression* • *Grayscale*	***Variable:*** • *Image size*
Profile Scenario 5: ***Image Content vs. other performance factors***	***Constants:*** • *Encoding process* • *Lossless Compression* • *Grayscale(512x512x8bit)*	***Variable:*** • *Image content type (sharp edges, photo, synthetic, text)*
Profile Scenario 6: ***Image Content vs. other performance factors***	***Constants:*** • *Decoding process* • *Lossless Compression* • *Color(512x512)*	***Variable:*** • *Image content type (sharp edges, photo, synthetic, text)*

Table 5.2 Profile Scenarios

5.3.4 Profiling Procedure

Profiling was performed using the profiler distributed with Microsoft Visual Studio 2008 version 9.0 and was carried out from within the MSVC development environment itself. The specifications of the profiling system are shown below in Table 5.3:

Processor	**Intel Core 2 Duo T9400, 2.53 GHz**
System RAM	4 GB
OS	Windows Vista
Profiling Tool	Microsoft Visual Studio 2008 Profiler, version 9.0
Software	*JasPer* 1.900.1

Table 5.3 Profiling System Specifications

5.3.5 Profiling Results

The following tables present the timing profiles for the different scenarios that were mentioned above. It must be specified that those modules and functions of the JPEG2000 encoder/decoder that did not account for over 1% of the total execution time in any of the testcases, are not presented seperately; instead, they are all accumulated into a seperate class named "Others".

Before proceeding to the presentation of the time profiles we remind that Tier-1 Coder consists of 2 basic components: EBCOT (Coefficient Bit Modeling stage) and Arithmetic Entropy Coding (MQ-Coder) as described in Chapter 4 and seen in Figure 5.2. For the purposes of calculating execution times, those 2 components have been evaluated seperately in order to provide more specific results. By simply adding the execution times of these components we get the total execution time for Tier-1 Coding.

Finally, the abbreviations used in the tables are: **DWT:** Discrete Wavelet Transform, **MCT:** Multi-Component Transform, **AEC:** Arithmetic Entropy Coding (MQ-Coder).

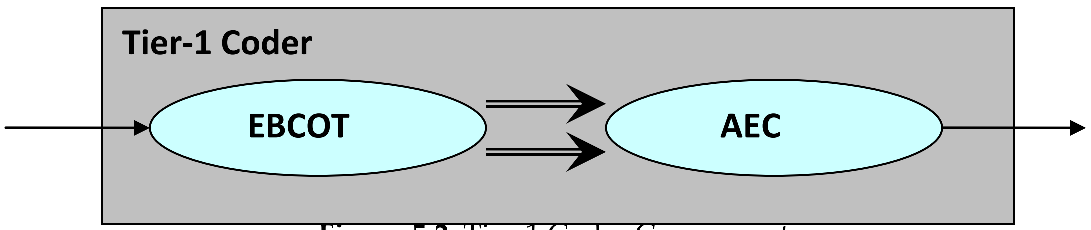


Figure 5.2: Tier-1 Coder Components

Timing Profiles:

Image Size (pixels)	**Total execution time** (ms)	**MCT**	**DWT**	**EBCOT**	**AEC**	**Others**
128x128	34.41	2.8%	26.5%	49.0%	12.3%	9.4%
256x256	90.72	2.6%	25.5%	43.0%	14.9%	14.0%
512x512	292.20	3.9%	27.2%	44.1%	12.0%	12.8%
1024x1024	507.69	3.2%	30.9%	44.3%	11.5%	10.1%

Table 5.4 Time Profile 1 (Lossy 10:1, Encoder, Color)

Image Size (pixels)	**Total execution time** (ms)	**MCT**	**DWT**	**EBCOT**	**AEC**	**Others**
128x128	11.28	2.4%	22.7%	49.5%	15.6%	9.8%
256x256	49.08	2.7%	22.3%	46.5%	16.9%	11.6%
512x512	150.92	2.8%	23.8%	46.0%	15.4%	12.0%
1024x1024	247.81	2.7%	23.1%	45.9%	15.1%	13.2%

Table 5.5 Time Profile 2 (Lossy 10:1, Decoder, Color)

Image Size (pixels)	**Total execution time** (ms)	**MCT**	**DWT**	**EBCOT**	**AEC**	**Others**
128x128	30.17	4.2%	11.8%	57.5%	19.7%	6.8%
256x256	149.22	4.6%	11.9%	56.5%	20.8%	6.2%
512x512	250.71	4.6%	12.5%	56.0%	19.1%	7.8%
1024x1024	484.09	4.5%	13.4%	55.9%	19.8%	6.4%

Table 5.6 Time Profile 3 (Lossless, Encoder, Color)

Image Size (pixels)	**Total execution time** (ms)	**MCT**	**DWT**	**EBCOT**	**AEC**	**Others**
128x128	20.74	0.8%	29.8%	43.1%	15.7%	10.6%
256x256	91.70	0.7%	30.8%	43.7%	15.3%	9.5%
512x512	174.42	0.6%	29.2%	42.0%	16.0%	12.2%
1024x1024	337.12	0.6%	29.0%	41.3%	16.1%	13.0%

Table 5.6 Time Profile 4 (Lossless, Decoder, Grayscale)

Image Size (pixels)	**Total execution time** (ms)	**MCT**	**DWT**	**EBCOT**	**AEC**	**Others**
Photo	221.40	0.7%	22.1%	45.0%	15.4%	16.8%
Synthetic	199.09	0.7%	25.7%	49.1%	13.8%	10.7%
Edges	230.88	0.7%	23.5%	44.5%	15.5%	15.8%
Text	247.57	0.8%	22.8%	43.6%	16.8%	16.0%

Table 5.7 Time Profile 5 (Lossy 10:1, Encoder, Grayscale 512x512x8 bits)

Image Size (pixels)	**Total execution time** (ms)	**MCT**	**DWT**	**EBCOT**	**AEC**	**Others**
Photo	151.59	5.7%	9.9%	54.5%	21.9%	9.0%
Synthetic	144.00	5.3%	10.5%	53.5%	22.0%	8.7%
Edges	152.23	5.4%	10.7%	54.0%	20.9%	9.0%
Text	160.44	4.9%	11.2%	53.9%	21.5%	8.5%

Table 5.8 Time Profile 6 (Lossless, Decoder, Color 512x512)

The percentage of the total execution time that each of the above components consumes, is presented in the following diagrams (Figure 5.3 and Figure 5.4). The diagrams refer to Time Profile 1 (Table 5.4) and Time Profile 6 (Table 5.8) respectively. It is clearly observed that DWT and Tier-1 Coding (EBCOT and AEC) are the most time consuming components, accounting for above 80% of the total execution time in every testcase.

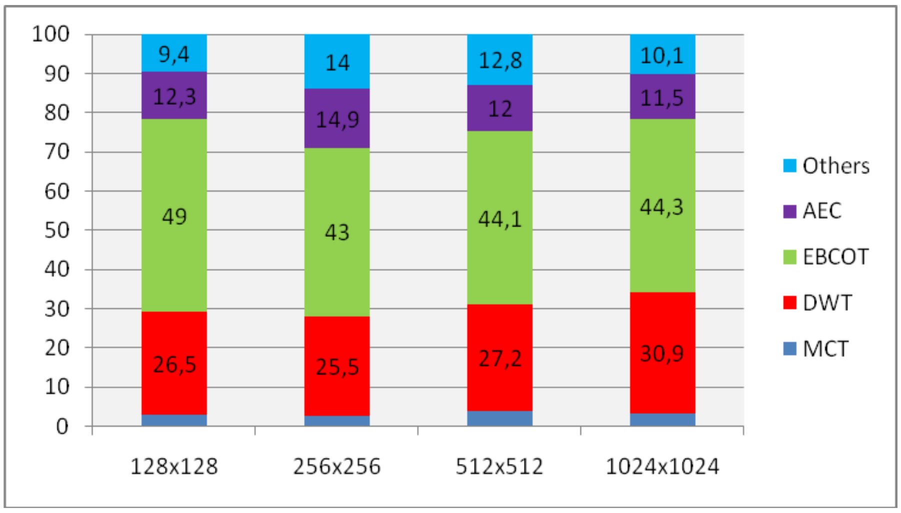


Figure 5.3: Time Profile 1 Execution Percentage Diagram

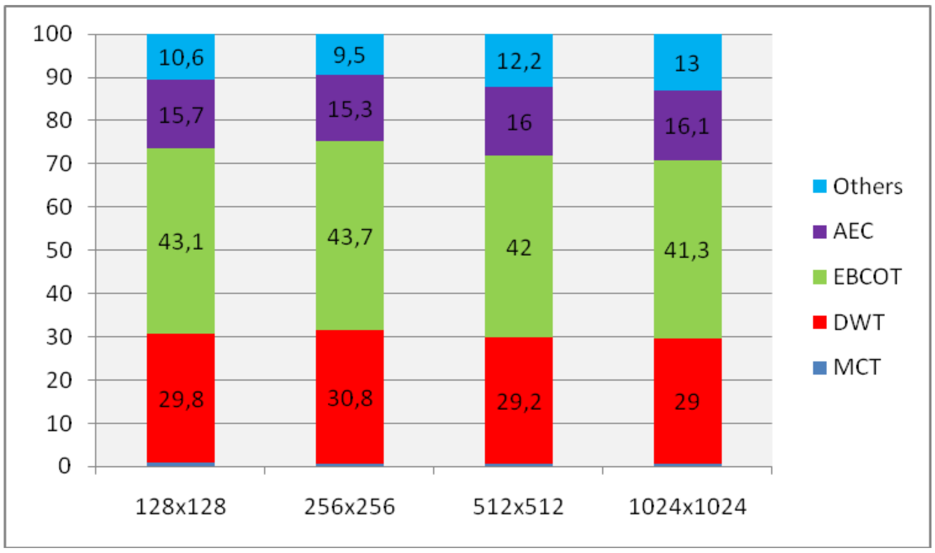


Figure 5.4: Time Profile 4 Exeution Percentage Diagram

5.3.6 Result Analysis and Conclusions

The time profiles that were extracted in the previous step can provide us with a first estimation of what kind of HW/SW partitioning solution would be the most wise to apply in terms of implementing the most time consuming functions in hardware; thus providing a speed-up factor to the whole compression engine. However, finding the most suitable solution is not straighforward. For this reason we have to take into cosideration the three selection criteria that were mentioned in paragraph 5.3. The conclusions one can draw by combining the profiling results and the nature of the algorithms and functions used in the software implementation, are presented in the following paragraphs.

5.3.6.1 Encoder vs. Decoder

The profile sets that were used in order to extract the required time profiles, also provided us with a first major conclusion; it was observed that the compression process (encoder) is more time-consuming in comparison to the decompression process (decoder). More specifically, the time spent in decoding the image is approximately 60% of the total time consumed in encoding the image.

5.3.6.2 Time-significant Routines

The total body of profile data generated clearly shows that there are a number of routines in which JasPer spends most of its execution time. These functions will be said to be 'time-significant'. A list of time-significant routines is given in Table 5.9. This list represents a summary of the overall results from all profile sets.

Category	Encoder	Decoder
Utilities	bitstoint() inttobits()	bitstoint() inttobits()
	pgxwordtoint()	pgxwordtoint()
	jasmalloc()	jasmalloc()
Image Transfer	jas_image_writecmpt()	jas_image_writecmpt()
	jas_image_readcmpt()	jas_image_readcmpt()
EBCOT	jpc_encsigpass()	jpc_decsigpass()
	jpc_encrefpass()	jpc_decrefpass()
	jpc_encclnpass()	jpc_decclnpass()
DWT	jpc_ft_analyze()	jpc_ft_synthesize()
AEC	jpc_mqenc_codelps()	jpc_mqdec_codelps()
	jpc_mqenc_codemps2()	jpc_mqdec_codemps2()

Table 5.9: Time-significant routines in JasPer Encoder and Decoder

5.3.6.3 Assessment of time-significant routines

Each category of time-significant functions was assessed for its suitability for implementation in hardware. As discussed in Section 5.3, it was desirable that if at all possible a modular section of JasPer be implemented on the FPGA.

- *Utility Routines*. The jas_malloc() utility routine performs memory allocation of the PC's main memory. It is not possible to implement such a routine on the FPGA. The other time-significant utility routines in Table 5.9 are very short – they contain very few lines of code. If these routines were implemented on the FPGA, the bus transfer overhead would be so great as to eliminate any potential speedup.
- *Image Transfer Routines*. The jas_image_writecmpt() and jas_image_readcmpt() functions are responsible for moving image component data between a matrix in memory and an I/O stream on disk. They are both I/O intensive functions and consequently are not good candidates for FPGA implementation.
- *Coefficient Bit Modelling Routines (EBCOT)*. The three routines jpc_encsigpass(), jpc_encrefpass() and jpc_encclnpas() perform the three coding passes in the coefficient bit modelling stage of the JPEG2000 encoder algorithm. The same applies in the case of decoding where the routines are those presented in Table 5.9. This section of the algorithm is quite complex. It also requires a degree of random access to code blocks being processed in order to examine the neighbors of the current sample being processed. Thus, although JasPer spends a large proportion of its time in this section, implementing the section's functions on the FPGA would be a considerably complex task.
- *Wavelet Transform Routines (DWT).* The wavelet transform is a modular section of the algorithm and another area where JasPer spends significant time, though not as much time as for the coefficient bit modeling. Implementing the wavelet transform stage on the FPGA would involve the creation of a complex data path. The JasPer implementation involves copying substantial blocks of data in main memory, which suggests that a hardware implementation would also require extensive access to local memory.
- *Arithmetic Entropy Coding Routines (AEC).* Along with coefficient bit modeling and wavelet transform, JasPer spends a great deal of its time in the arithmetic entropy encoding routines. This is especially true when larger images are being compressed. Compared to other stages of the JPEG2000 algorithm, the arithmetic encoding stage is relatively simple. The stage is fed a stream of bits

as input, along with 'context' information to be used in the encoding of those bits. The primary output of the stage is a stream of compressed bytes. In addition to a manageable level of complexity, the arithmetic encoder is very modular. It relies on almost no other part of the JPEG2000 algorithm.

5.3.7 Partitioning Solution

In order to decide which components will be ported to HW, three criteria were used, as stated before in this chapter:

- **Time significance**
- **Modularity**
- **Complexity/Potential speed-up**

Every single one of these criteria can be assumed as a "filter" that promotes or rejects components of the compression engine as candidates for porting to HW. The following table (Table 5.10) demonstrates this methodology according to the previously presented profiling results and the assessment of time significant routines in the previous paragraph.

It has to be underlined that, although the table presents a sequential assessment of the different components according to the above criteria, the modularity and complexity/potential speed-up of every component have been separately assessed. However time significance is the most important factor that overshadows the impact of the other two criteria, therefore it is presented at the top of this hierarchy.

Criteria	MCT	DWT	EBCOT	AEC
Time significance	LOSER 🚫	WINNER ↓	WINNER ↓	LOSER 🚫
Modularity	-	WINNER ↓	WINNER ↓	-
Complexity/Potential speed-up	-	WINNER ↓	LOSER 🚫	-

HARDWARE

Table 5.10

Summarizing, it is clearly seen that the EBCOT and DWT stages are the most time consuming ones in the JPEG2000 compression engine, with AEC following. After assessing each category of time-significant functions, it was decided to port the Discrete Wavelet Transform stage of JPEG2000 in HW. A large amount of JasPer's execution time was spent in the routines of this stage. Furthermore, the DWT's complexity, although being considered high, is not an overhead when taking into account the potential speed up in execution time. Whereas, a solution that would suggest porting the even more complex EBCOT algorithm in HW could not be justified for the purposes of this thesis when taking into account possible execution time speed-ups. The modularity of the wavelet transform was an additional advantage over the other stages considered. Finally, for complexity reasons the Inverse DWT for lossless compression was decided to be implemented in HW rather than the Forward DWT, therefore HW/SW partitioning solution refers to the JPEG2000 Decoder in lossless mode. Figure 5.5 illustrates the partitioning solution for this co-design project.

5.3.8 Scheduling

The next step in the architecture exploration process involves scheduling, which determines the orders of executing the behaviors on SW (Microblaze soft processor) and HW (FPGA fabric core). The scheduler ensures that the line-up of tasks does not breach any dependencies imposed by the specification. The highest level of the codec specification model requires serialization of concurrent behavior tasks. Figure 5.6

illustrates the scheduling of the JPEG2000 Decoder. The decoder blocks represent the main program that runs in synchronization with an external compressed image stream.

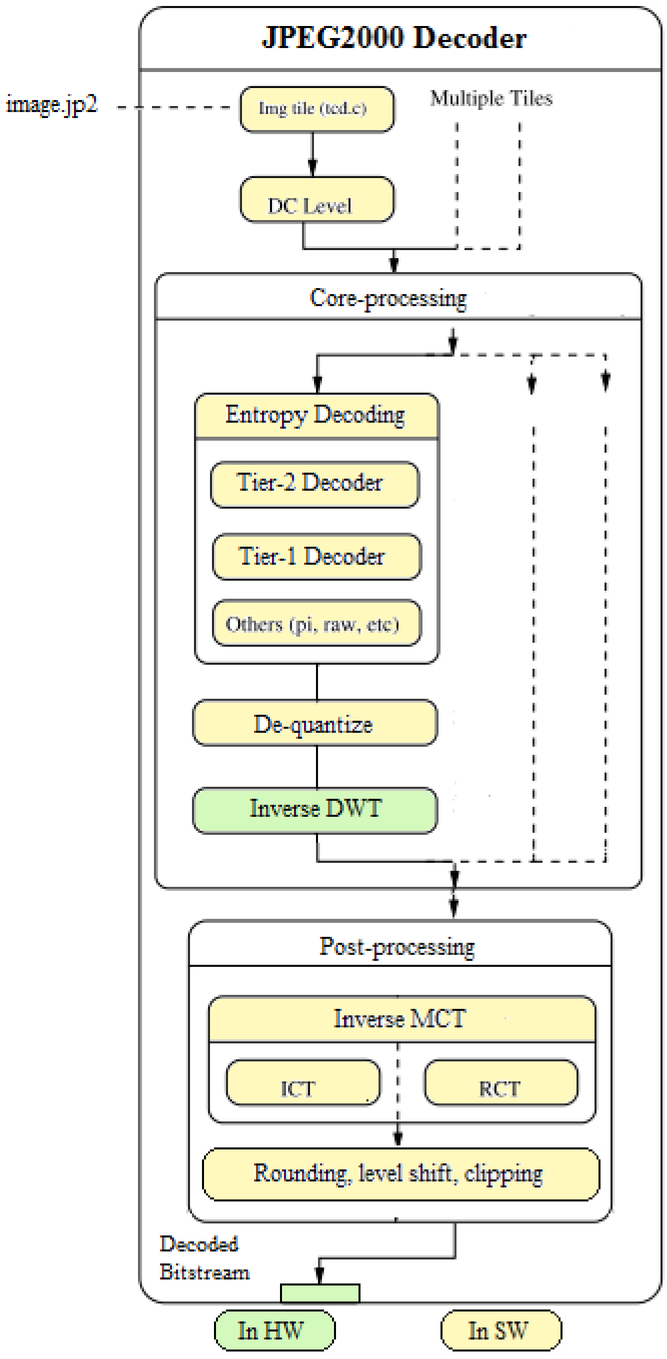


Figure 5.5: HW/SW Partitioning Solution

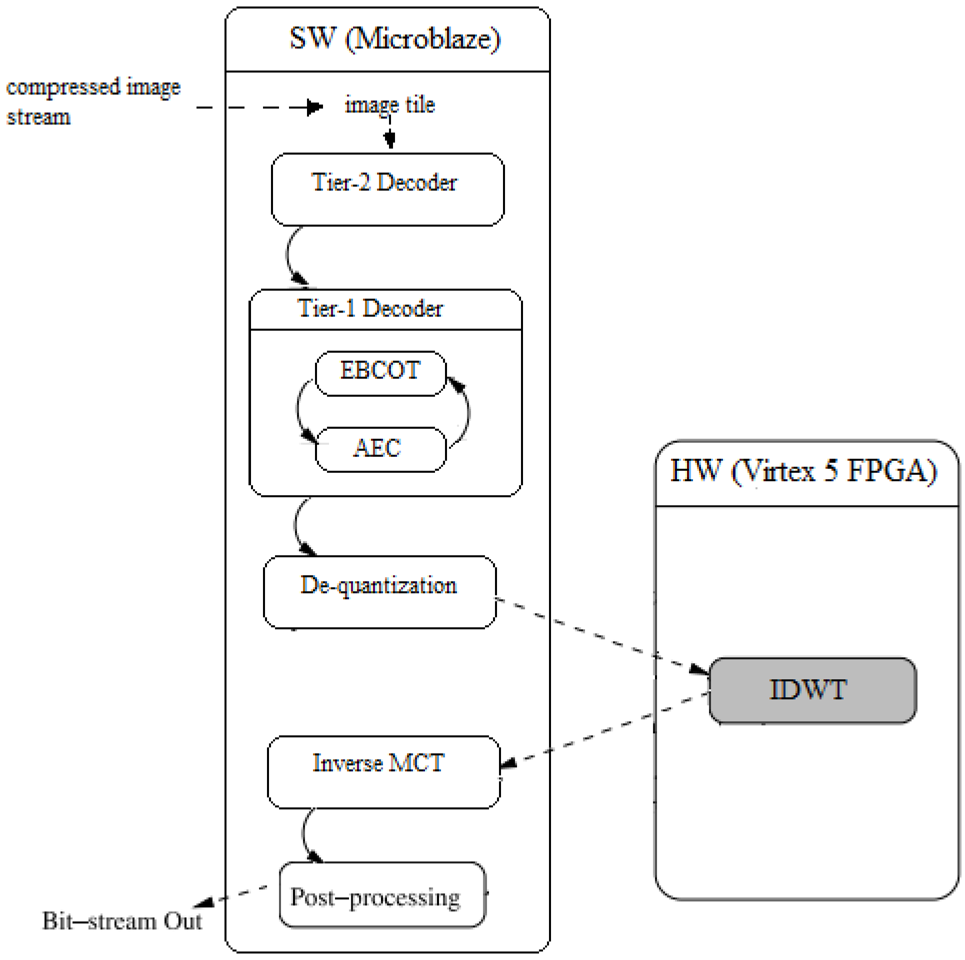


Figure 5.6: JPEG2000 Decoder Scheduling

Chapter 6

Implementation of the Inverse Discrete Wavelet Transform

Overview

This chapter discusses the development of a VHDL design that implements an Inverse Discrete Wavelet Transform core, which will then be attached to a larger Microblaze based system that implements the JPEG2000 codec on the Xilinx Virtex-5 board. Prior to presenting the design methodology that was followed and the respective simulation results, a description of the Discrete Wavelet Transform algorithm is given.

6.1 Introduction to Discrete Wavelet Transform

The term *wavelet* was originally used in the field of seismology to describe disturbances that emanate and proceed out of sharp seismic impulse. Properties of Fourier representation of signals is known to be effective in analysis of time-invariant (stationary) periodic signals. In contrast to sinusoidal function, properties of wavelet allow for both time and frequency analysis of signals simultaneously due to the fact that energy in wavelet is concentrated both in time and still possesses the wave-like (periodic) features. As a result, wavelet representation gives a versatile mathematical tool to analyze transient, time-variant (non-stationary) signals that may be statistically predictable especially at the regions of discontinuities – a special feature for that is typical of images having discontinuities at the edges.

Wavelets are functions generated from one single function (basis function) called the *prototype or mother wavelet* by the *dilations* (scaling) and *translations* (shifts) in time (frequency) domain. If the mother wavelet is denoted by $\psi(t)$ the other wavelet $\psi_{a,b}$ can be expressed as:

$$\psi_{a,b}(t) = \frac{1}{|\sqrt{a}|}\psi\left(\frac{t-b}{a}\right) \quad (6.1)$$

Where *a* and *b* are real numbers. The variables a and b represent the parameters for the dilations and translations respectively in the time axis. From Eq. 6.1 we can derive that:

$$\psi_{a,0}(t) = \frac{1}{|\sqrt{a}|}\psi\left(\frac{t}{a}\right) \quad (6.2)$$

As shown in the Eq. 6.2 $\psi_{a,0}(t)$ is time-scaled and amplitude-scaled. The parameter *a* causes contraction of $\psi(t)$ in the time axis when $a<1$ and expansion or stretching when $a>1$. For $a<0$, the function $\psi_{a,b}(t)$ leads time reversal with dilation. The function $\psi_{a,b}(t)$ is ashift in the left along the time axis by an amount *b* when $b>0$, whereas it is a shift in right along the time axis by the amount *b* when $b<0$. Therefore variable *b* represents the translation in time (*shift in frequency*) domain.

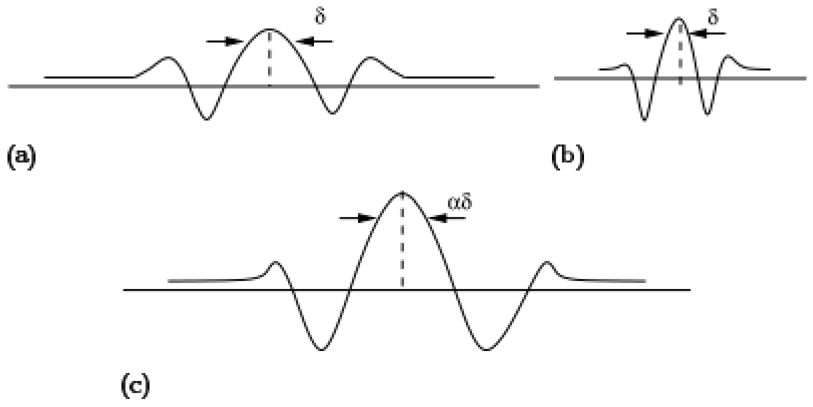


Figure 6.1: Illustration of mother wavelet and its dilations in time domain. (a) mother wavelet $\psi(t)$, (b) $\psi(t/a)$: $0<a<1$, (c) $\psi(t/a)$: $a>1$

Figure 6.1 is an illustration of a mother wavelet and its dilations in time domain. Signal contraction and expansion in time axis is shown in figure 6.1(b) and 6.1(c) respectively. Based on this definition, *wavelets transform* (WT) of the function (signal) *f(t)* is represented by:

$$W(a,b) = \int_{-\infty}^{\infty} \psi_{a,b}(t) f(t) dt \quad (6.3)$$

The inverse transform to reconstruct f(t) from W(a,b) is mathematically represented by:

$$f(t) = \frac{1}{C}\int_{a=-\infty}^{+\infty}\int_{b=-\infty}^{+\infty}\frac{1}{|a|^2}W(a,b)\psi_{a,b}(t)da\,db \quad (6.4)$$

Where

$$C = \int_{-\infty}^{+\infty} \frac{|\Psi(w)|^2}{|w|} dw$$

And $\Psi(w)$ is the Fourier transform of the mother wavelet $\psi(t)$.

If *a* and *b* are two continuous variables and *f(t)* is also a continuous function *W(a,b)* is called *continuous wavelet transform.*

6.2 Discrete Wavelet Transform

The discrete wavelet transform (DWT) refers to wavelet transforms for which the wavelets are discretely sampled. The discrete wavelets can be represented in Eq. 6.5

$$\psi_{m,n}(t) = a_0^{-m/2}\psi(a_0^{-m}t - nb_0) \quad (6.5)$$

After substituting $a = a_0^m$ and $b = nb_0a_0^m$ into Eq. 6.1. We choose $a_0 = 2$ and $b_0 = 1$, then we can represent the discrete wavelet in Eq. 6.6.

$$C_{m,n}(f) = a_0^{-m/2} \int f(t)\psi(a_0^{-m}t - nb_0)dt \quad (6.6)$$

And hence for dyadic decomposition, the wavelet coefficients can be given as:

$$C_{m,n}(f) = 2^{-m/2} \int f(t)\psi(2^{-m}t - n)dt \quad (6.7)$$

This permits us to reconstruct the signal *f(t)* from Eq. 6.7 as:

$$f(t) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} C_{m,n}(f)\psi_{m,n}(t) \quad (6.8)$$

When the input function and its wavelet parameters are represented in discrete form, the such transformation is called DWT of signal f(t). The introduction of multi-resolution representation of signals based on wavelet decomposition gave DWT the recognition as a very versatile signal processing tool. The method of multi-resolution represents a function as a group of coefficients, each of which gives information about the position and frequency of signal (function). The usefulness of DWT over Fourier transform is its performance of multi-resolution analysis of signals. Therefore DWT decomposes a digital signal into different subbands so that the lower frequency subbands have finer frequency resolution and rough time resolution in contrast to higher frequency subbands. The DWT is used in image compression because DWT can be applied to features like progressive image transmission (by quality and resolution), flexibility of compressed image manipulation, region of interest coding etc.

6.3 The Concept of Multi-resolution Analysis

The theory of multi-resolution analysis introduces a systematic approach to generate the wavelets. The design of multi-resolution analysis is to approximate a function *f(t)* at different levels of resolution. In this analysis, we consider the mother wavelet $\psi(t)$ and the scaling factor $\phi(t)$. The dilated (scaled) and translated (shifted) variant/version of the scaling function is given by

$$\phi_{m,n}(t) = 2^{-m/2}\phi(2^{-m}t - n).$$

The set of scaling functions $\phi_{m,n}(t)$ are orthonormal when m is fixed. We generate a set of functions in Eq. 6.9 by the linear combination of the scaling functions and its translations.

$$f(t) = \sum_n \alpha_n \phi_{m,n}(t) \tag{6.9}$$

Let's consider the representation of an image with few pixels at consecutive levels of approximation. The wavelet coefficients is assumed as an extended information required to move from coarse to finer approximation. Therefore in each level of decomposition the signal can be decomposed into two parts, one is coarse of the signal in the lower resolution and the other is the detail information that was lost due to approximation. This can be represented in Eq. 6.10, where f_m denotes the value of input function at resolution 2^m, $c_{m+1,n}$ is the detail information and $a_{m+1,n}$ is the coarser approximation of the signal at resolution 2^{m+1}. The functions $\phi_{m+1,n}$ and $\psi_{m+1,n}$ are the dilation and wavelet basis functions (orthonormal).

$$f_m(t) = \sum_n a_{m+1,n}\phi_{m+1,n} + \sum_n c_{m+1,n}\psi_{m+1,n} \tag{6.10}$$

6.3.1 Two-Dimensional Signals in DWT

A two-dimensional signal can be represented by two-dimensional array *X[M,N]* with *M* rows and *N* columns. Where *M* and *N* are non-negative integers. The implementation of two-dimensional DWT is to perform one-dimensional DWT row-wise to produce an intermediate result and then perform the same one-dimensional DWT column-wise on this same intermediate result to generate a final result. This approach is shown in figure 6.2(a). This is possible because the two-dimensional scaling functions can be expressed as separable functions that are product of two-dimensional scaling functions such as $\phi_2(x,y) = \phi_1(x)\phi_1(y)$. The same applies for wavelet function $\psi(x,y)$.

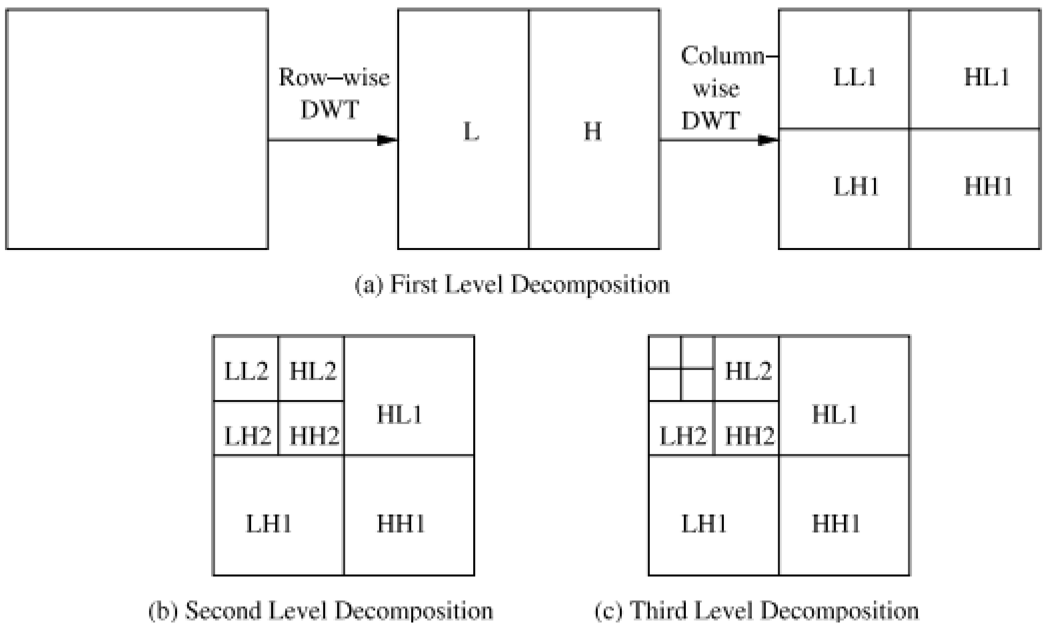

Row–wise DWT
L
H
Column–wise DWT
LL1
HL1
LH1
HH1
(a) First Level Decomposition
LL2
HL2
LH2
HH2
HL1
LH1
HH1
(b) Second Level Decomposition
HL2
LH2
HH2
HL1
LH1
HH1
(c) Third Level Decomposition

Figure 6.2: Row-Column Computation of 2D DWT

Two subbands in each row are produced after applying one-dimensional transformation in each row. When the low frequency of all rows (L) are put together this results into a zone of MxN/2 the size of the input signal as we can see in Figure 6.2(a). Similarly, putting together the higher frequency subbands of all rows produces the H subband (again of size MxN/2) which predominantly contains the high-frequency information around discontinuities (edges in an image) of the input signal. Applying a one-dimensional (1D) DWT column-wise on these L and H subbands produces LL, LH, HL, and HH subbands of size MxN/4 each. LL is the coarser version of the original input signal (image) whilst LH, HL, HH form the high frequency subband that contains the detail information. The same result is obtained when applying 1D DWT column-wise and then row-wise.

Multi-resolution decomposition is illustrated in Figure 6.2(b). The first level of decomposition generates four subbands LL1, HL1, LH1, and HH1. For an image, the LL1 subband can be considered as a 2:1 subsampled (both horizontally and vertically) version of the original image. The other three subbands HL1, LH1, HH1 contain much higher frequency detail information. These spatially oriented subbands mostly contain detail information of discontinuities in the image and a bulk of the energy in each of these three subbands is concentrated in the neighborhood of areas realting to edge activities in the original image. LL1 has similar spatial and statistical characteristics to the original image because it is a coarser approximation of the imput. Therefore it can further be decomposed into four subbands LL2, HL2, LH2, and HH2 as shown in Figure 6.2(b). Accordingly the image can be decomposed into ten subbands LL3, HL3, LH3, HH3, LL2, HL2, LH2, HH2, HL1, LH1, and HH1 as shown in figure 6.2(c). Even more levels of decomposition can be applied.

The application of DWT with 3 levels of decomposition in a real image ("Lena") is shown in Figure 6.3.

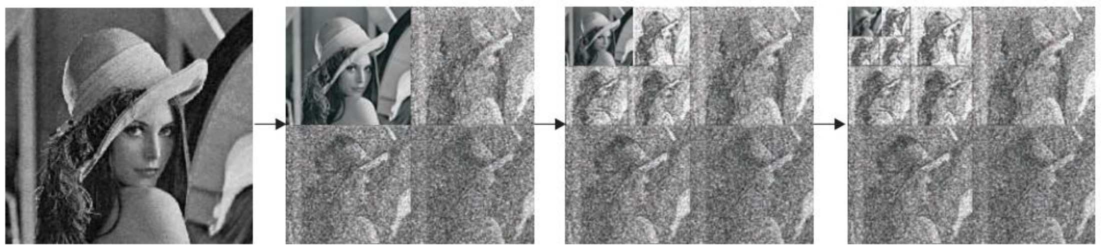


Figure 6.3: DWT with 3 decomposition levels applied on "Lena" image.

6.4 Lifting Implementation of DWT

DWT has traditionally been implemented by convolution or FIR filter bank structures. In contrast to block based implementation in discrete cosine transform (DCT), DWT is frame-based. Such an implementation requires both large number of arithmetic computations and a large memory for storage – a feature that is undesirable for high speed or low-power video processing applications. The introduction of the lifting-based wavelet transform requires fewer computations compared to the convolution based one. It promises reduction of computational complexity up to 50%, 'in-place' computation of DWT, integer-to-integer wavelet transform (IWT), symmetric forward and inverse transform requiring no extension, etc. The main feature of this scheme is to break up the high-pass and low-pass wavelet filters into a sequence of small filters that in turn can be converted to a sequence of upper and lower triangular matrices. In traditional forward DWT using filter banks, the input signal *(x)* is filtered separately by a low-pass filter *($\hat{h}$)* and a high-pass filter *($\hat{g}$)* at each transform level. The two output streams are then subsampled by dropping alternate output samples in each stream to produce a low-pass *(y_L)* and a high-pass *(y_H)* subband. These two filters form the analysis filter bank. The original signal can be reconstructed by the synthesis filter bank *(h,g)* starting from y_L and y_H. The procedure is shown in Figure 6.4 and Figure 6.5.

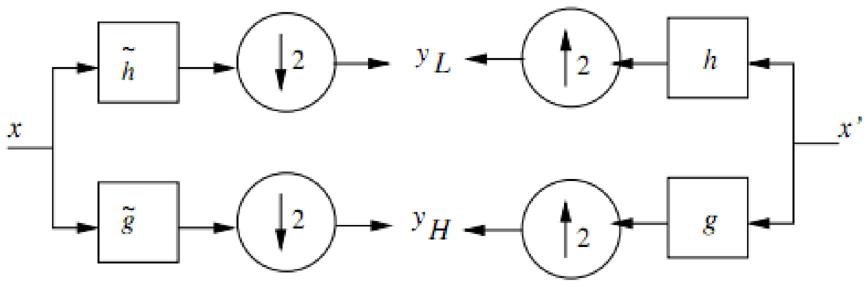


Figure 6.4: Row-Column computation of 2D DWT

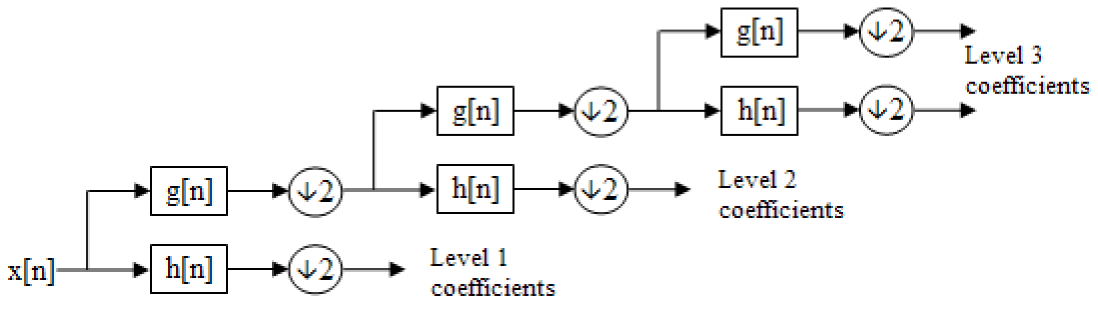


Figure 6.5: 3–level lifting schema

6.4.1 Lifting Scheme

There are two kinds of lifting and they are:

Primal Lifting: is defined as the computation of the upper triangular matrix of a resulting DWT polyphase matrix, hence lifting the low-pass subband with the help of the high-pass subband.

Dual Lifting: is defined as the computation of the lower-triangular matrix of a resulting DWT polyphase matrix so that the high-pass subband is lifted with the help of the low-pass subband.

These lifting steps are often called ***Update*** and ***Predict.*** In Figure 6.6 a lifting based forward DWT schema is presented where steps P and U refer to Predict and Update stages respectively.

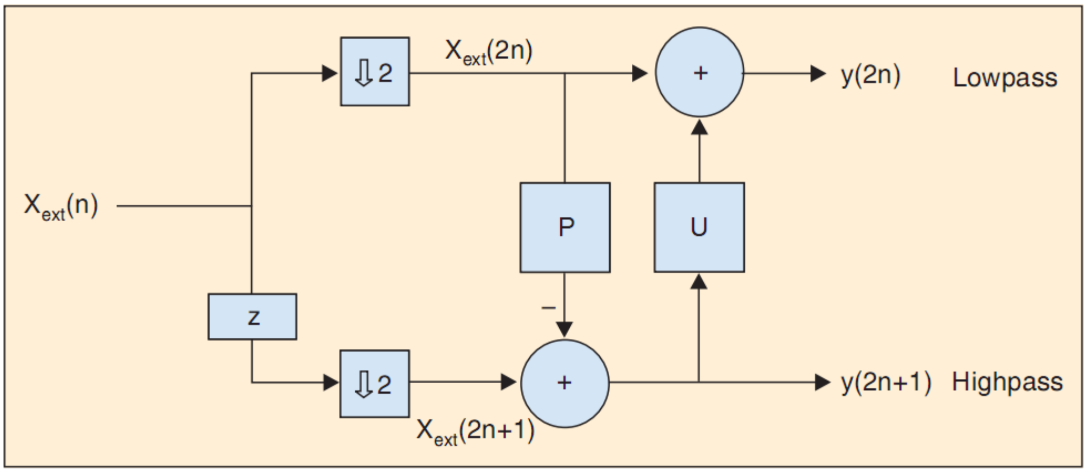


Figure 6.6: Lifting based FDWT

6.4.2 Data Dependency Diagram for Lifting Computation

Computation of lifting-based DWT can be described using the diagram in Figure 6.7. The lifting requiring four 'lifting' factors, such as (9,7) filter, are done in four stages. For the DWT filters that require only two lifting factors, such as (5,3) filter, the intermediate two stages are bypassed. The outcome produced in the first stage of the data dependency diagram is stored in the registers containing odd samples of the input data because these samples may not be used in later stages of the computation. In the same way, results produced in the second stage can be stored back to the registers assigned to the even samples of the input data. Following the same pattern, the high-pass (low-pass) output samples are stored into the registers where the odd (even) samples of the input data are originally stored at the beginning of the computation. Therefore no extra memory is required at any stage. This property of the lifting-based computation is called "in-place computation'.

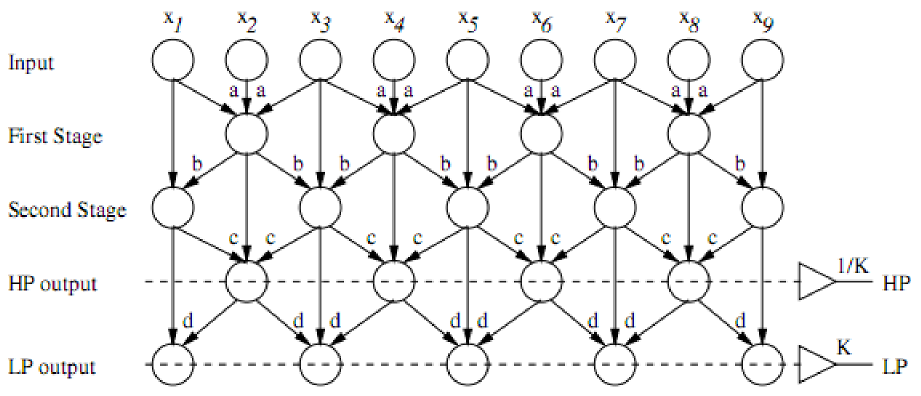


Figure 6.7: Data dependency with four lifting factors

6.5 Lifting-based DWT in JPEG2000

In JPEG2000, the DWT is implemented using a lifting-based scheme, as described before. The transform that is applied both in lossy and lossless compression uses the (5,3) filter (Le Gall). In fact (5,3)-DWT is the only one that allows for lossless compression as it is an integer-to-integer transform, and thus can be inverted. The (9,7) (Daubechies) filter is used mainly in lossy compression, leads to a real-to-real transform, and is of higher complexity as it uses all 4 lifting stages. As it was described previously, 2D DWT is achieved by sequentially applying 1D DWT row-wise and then column-wise. In (5,3) filter only one Predict and one Update stage are needed for each application of 1D DWT. If *x(n)* is the sequence of spatial coefficients of the input signal, then the Forward DWT coefficients *y(n)* are given by the following equations:

$$
\begin{aligned}
y(2n+1) &= x(2n+1) - \left\lfloor \frac{x(2n)+x(2n+2)}{2} \right\rfloor \\
y(2n) &= x(2n) + \left\lfloor \frac{y(2n-1)+y(2n+1)+2}{4} \right\rfloor
\end{aligned}
$$

Whereas the Inverse DWT is computed using the inverse system:

$$
\begin{aligned}
x(2n) &= y(2n) - \left\lfloor \frac{y(2n-1)+y(2n+1)+2}{4} \right\rfloor \\
x(2n+1) &= y(2n+1) + \left\lfloor \frac{x(2n)+x(2n+2)}{2} \right\rfloor
\end{aligned}
$$

6.6 Motivation for applying lifting-based DWT in JPEG2000

The advantages of lifting-based DWT over convolution ones are outlined below:

- *Computational efficiency:* Lifting-based DWT requires less computation (up to 55%) compared to convolution based one.
- *Memory savings:* In lifting implementation, no additional memory buffer is needed due to in-place computation feature of lifting.
- *Integer-to-integer transform:* This offers integer-to-integer transformation which is suitable for lossless image compression.
- *No boundary extension:* Is avoided due to the fact that the original data can be reconstructed using the integer-to-integer transform.

Analysis of the JasPer source code and its specifications showed that the DWT stage is implemented using the lifting scheme described above using both (9,7) and (5,4) filters. Therefore our VHDL implementation should be based on the same lifting-based architecture after analyzing and thoroughly examining the JPEG2000 specification model.

6.7 Implementation Requirements

The primary requirement of the VHDL implementation was one of compliance. The code written was required to conform to the JPEG2000 standard for the IDWT stage. Additionally, it was required that the VHDL design be written in synthesizable code. If the code failed to conform to the JPEG2000 standard or failed to be synthesizable, the co-processing system could not function correctly.

6.7.1 JPEG2000 Specifications regarding Inverse DWT

The first step in the development process was to analyze thoroughly the JPEG2000 specification for the Inverse Discrete Wavelet Transform. By its very nature, the standard is unambiguous in regard to the algorithm the transform must carry out. Analyzing the standard led to a good knowledge of the internal workings of the DWT stage as well as to initial ideas about how a hardware implementation could be designed.

The JPEG2000 standard explains basic data inputs and outputs of the IDWT as shown in Figure 6.8. In Table 6.1 gives a short description of the information passed to and from the IDWT stage. As expected, the JPEG2000 specification model follows the lifting-based DWT architecture that was previously described in this section.

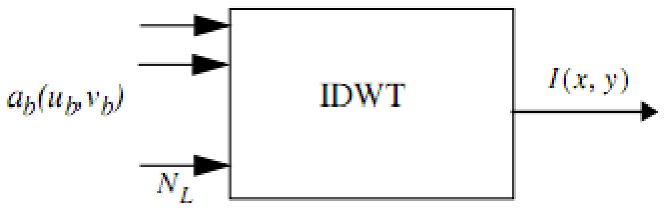


Figure 6.8: Inputs and outputs of the IDWT procedure

$a_b(u_b,v_b)$	Coefficients of sub-bands (compressed image pixels)
N_L	Levels of decomposition
$I(x,y)$	Shifted tile component samples (reconstructed image pixels)

Table 6.1 Basic inputs and outputs of IDWT

6.7.2 JasPer's Implementation of the IDWT

Following an examination of the JPEG2000 specification itself, the JasPer software implementation of the IDWT was assessed. Most of the code for this stage of the algorithm is contained in the file 'jpc_qmfb.c', though several other source code files were referred to as well. Each function in the JasPer implementation was compared to the JPEG2000 standard. This step was carried out for a number of reasons. Firstly, to ensure that JasPer followed the standard accurately. This was indeed found to be the case. Secondly, examining a working implementation of the IDWT was a valuable way to reinforce an understanding of its internal operation. Thirdly, examining the JasPer code was necessary to determine the nature of the interface JasPer expects the IDWT stage to service. Table 6.2 summarizes the functional role of specific functions used in the software implementation of IDWT and more specifically of the (5,3) filter that was implemented for the purposes of this thesis.

Functions	**Role**
jpc_ft_synthesize()	Basic function that implements IDWT on a MxN image matrix by calling functions that manage 1D IDWT column-wise and row-wise

jpc_ft_invlift_row()
 	
jpc_qmfb_join_row()
	Functions that implement 1D IDWT row-wise. As required by the (5,3) filter, only the Predict and Update lifting steps are implemented.

jpc_ft_invlift_colgrp()
 	
jpc_qmfb_join_colgrp()
	Functions that implement 1D IDWT column-wise on 16xN tiles. Column group of 16 is an implementation specific strategy for the JasPer codec. Again, only the Predict and Update lifting steps are implemented.

jpc_ft_invlift_colres()
 	
jpc_qmfb_join_colres()
	Functions that implement 1D IDWT column-wise on the remaining columns (as M in not always a multiple of 16).

Table 6.2: Basic software functions that implement IDWT

Finally, the JasPer code was examined to determine where the IDWT was accessed in the overall decoding process. This was performed with a view to determining where data buffering between JasPer and the Virtex-5 board could occur.

6.8 Proposed IDWT architecture

In this section we discuss the architecture proposed and used in this project. The architecture is a modified version of the one hinted in [11]. The global architecture is shown in Figure 6.9.

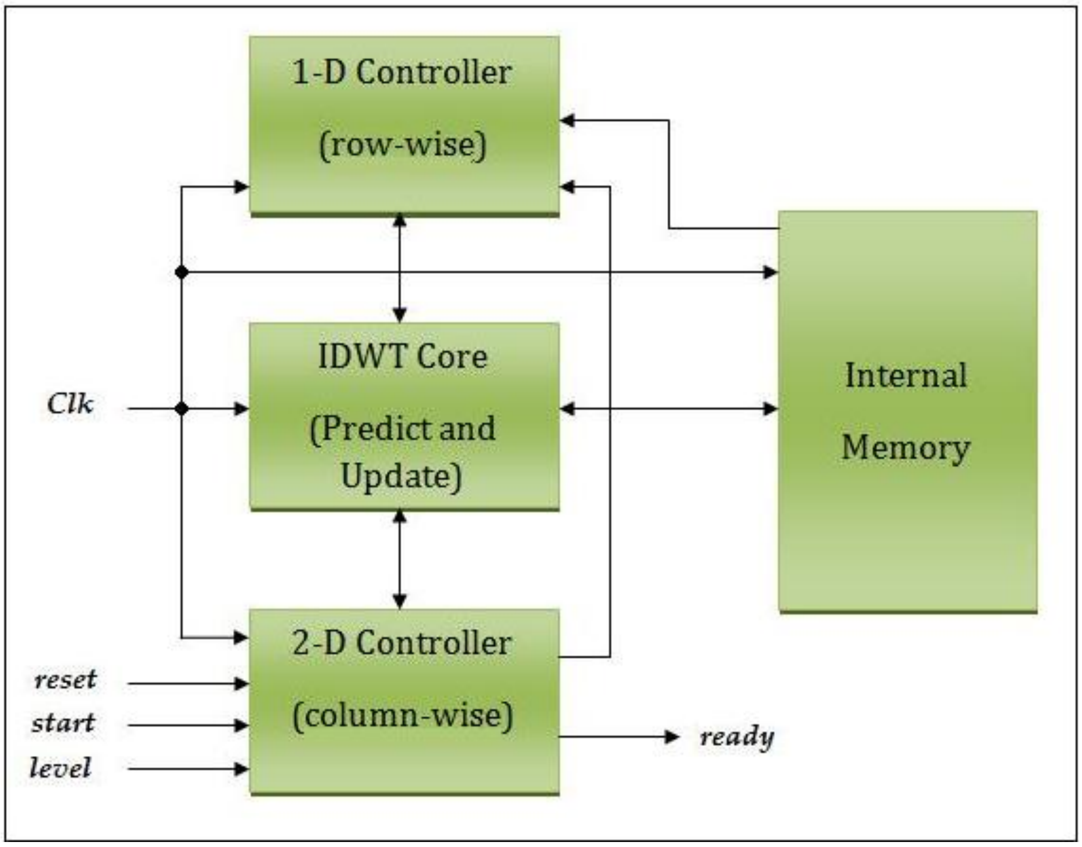


Figure 6.9: Proposed IDWT architecture

As shown in Figure 6.9 the IDWT unit consists of two control units implemented as FSM: 1-D Controller unit for horizontal (row-wise) and another 2-D Controller for vertical (column-wise) transforms. The IDWT Core (Predict and Update) is designed to accomplish the arithmetic operations of Predict and Update steps. To process an image, all rows are transferred to the IDWT Core from the internal memory and transformed on the fly by the horizontal 1-D Controller unit. Then the vertical transform takes place by the 2-D Controller unit. This allows a pipelined approach because the intermediate results do not have to be transposed. The control units coordinate steps in order to process the whole image and are responsible for generating enable signals, address lines, etc. At the end, the inverse transformed image coefficients are available in the internal memory. All necessary boundary information is included in the computation. Level of decomposition is controlled by a 3-bit "level" signal (the architecture supports up to 7 levels of decomposition).

IDWT implementation requires 2 data inputs at a time in order to calculate an output coefficient. The architecture proposed accommodates this by introducing appropriate pipeline/delay stages for the data inputs within the IDWT core.

The 1-D Controller module is performing data multiplexing that also generate the address for memory reads and writes. The image height and width are passed as parameters to the 2-D Controller module. The IDWT Core module computes the transform coefficients of the input image pixels obtained by the memory read operation. After the computation, the high-pass and low-pass coefficients are passed to the right memory location. After each level of decomposition, the roles of memory banks are swapped. The input and output address are generated through the 2-D Controller module during vertical operation. These generated addresses are supplied to necessary memory address input and output buses to the internal memory. The internal ram accepts the write addresses generated by the 1-D Controller module and the coefficients produced by the IDWT Core. A single execution of this module writes the two coefficients to the right memory bank.

The aforementioned architecture was implemented in VHDL, and in a top-down hierarchical and behavioral fashion.

6.9 Simulation Results

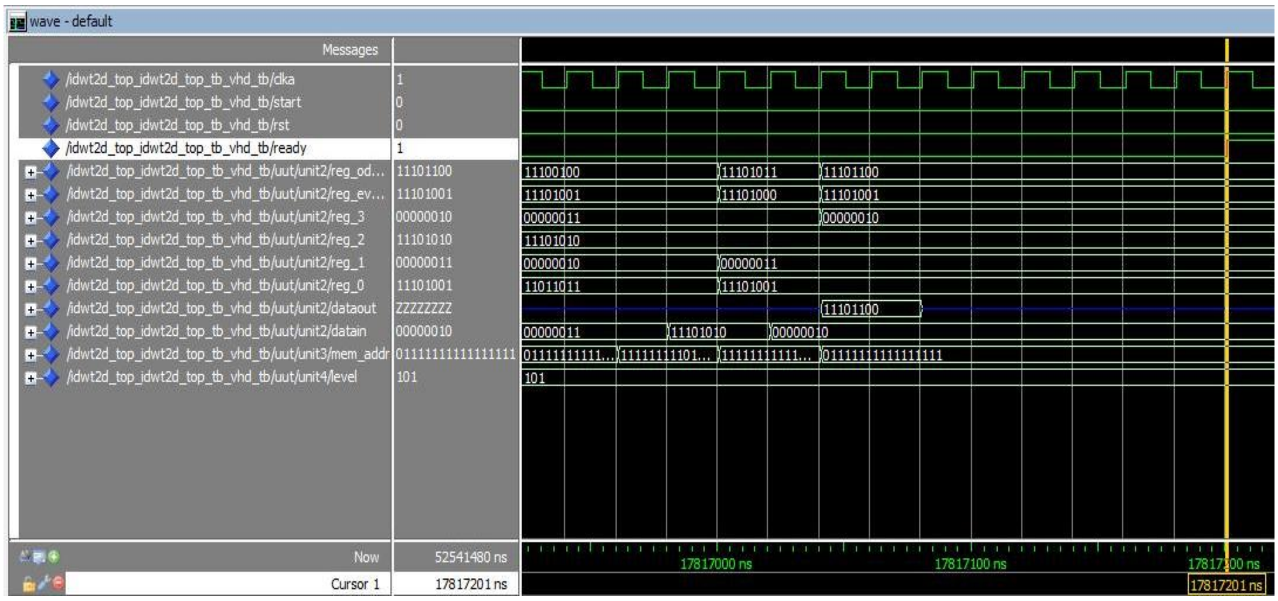
For verification purposes the design was simulated using Modelsim SE 6.3f. It is known that this particular tool only offers functional (logical) simulation of a specific VHDL design, thus not providing a "realistic" simulation of the system that depends on timing constraints associated to the target implementation platform (FPGA). However, it is the first stage to verify that our systems presents an accepted behavior and that will eventually work after synthesis, probably with some alterations regarding timing constraints.

The task of creating a test bench for this simulation needed a careful approach. A trivial test bench would initialize the system's memory with image data that have been transformed by FDWT (implemented in Matlab or C), wait for the IDWT system to finish the computations, then read back the reconstructed image data and compare them to the expected output, which again could probably be produced by a

C or Matlab program that applies DWT on an image. However, the fact that our VHDL design is part of a HW/SW Co-design system implies that the system should be tested with actual data generated from the JasPer software. Therefore, the JasPer software was modified to allow it to produce a log file of its activity and any data transfers during the IDWT stage, and specifically the input matrices and the output matrices of the IDWT stage, along with any information regarding the current level of decomposition. In this way, several text files were produced containing pixel values of a different set of images. These text files were used to initialize the system's memory or be compared to the output that the system generated during simulation.

The VHDL implementation of the IDWT successfully passed all of these tests on real image data (4 different images x 3 different sizes each) with 100% accuracy on generated pixel values.

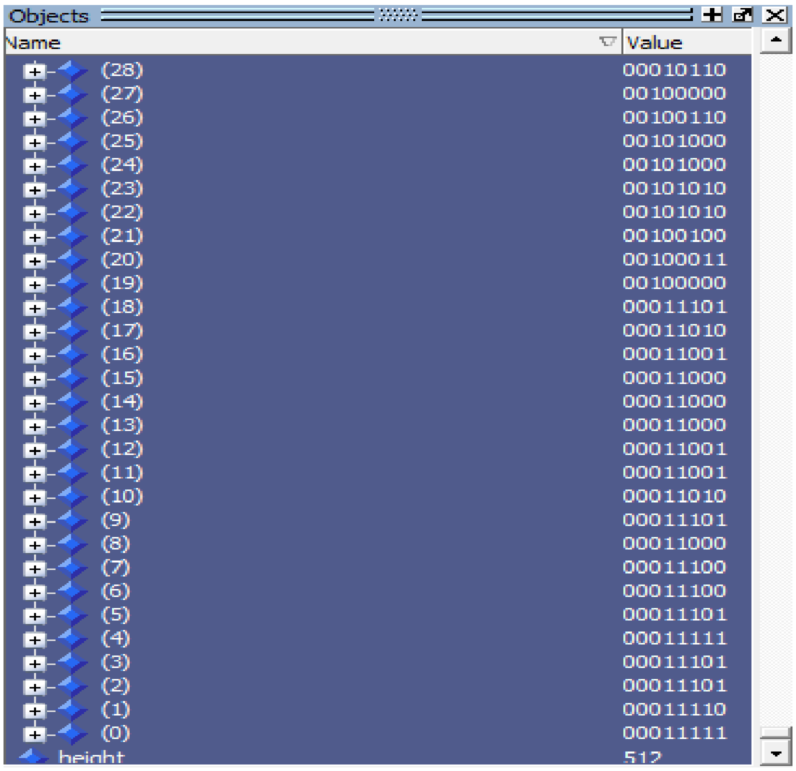
The following figure (Figure 6.10) shows a snapshot of the simulation in Modelsim, at the time frame when signal "ready" becomes high, marking the end of the IDWT process. In the waveform view one can see the addresses generated, the registers that contain the samples that undergo computation by the IDWT core, the output coefficients that are written to memory and the current level of decomposition. This simulation was performed on a 256x256 grayscale image with 5 levels of decomposition (JasPer also uses 5 levels by default).



```
wave - default
                                              Messages
/idwt2d_top_idwt2d_top_tb_vhd_tb/clka                       1
/idwt2d_top_idwt2d_top_tb_vhd_tb/start                      0
/idwt2d_top_idwt2d_top_tb_vhd_tb/rst                        0
/idwt2d_top_idwt2d_top_tb_vhd_tb/ready                      1
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/reg_od...        11101100
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/reg_ev...        11101001
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/reg_3            00000010
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/reg_2            11101010
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/reg_1            00000011
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/reg_0            11101001
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/dataout          ZZZZZZZZ
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit2/datain           00000010
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit3/mem_addr         0111111111111111
/idwt2d_top_idwt2d_top_tb_vhd_tb/uut/unit4/level            101
                                                   Now      52541480 ns
                                              Cursor 1      17817201 ns
```

Figure 6.10: Simulation of IDWT core

Part of the memory contents after the end of all computations and data transfers are presented in Figure 6.11.



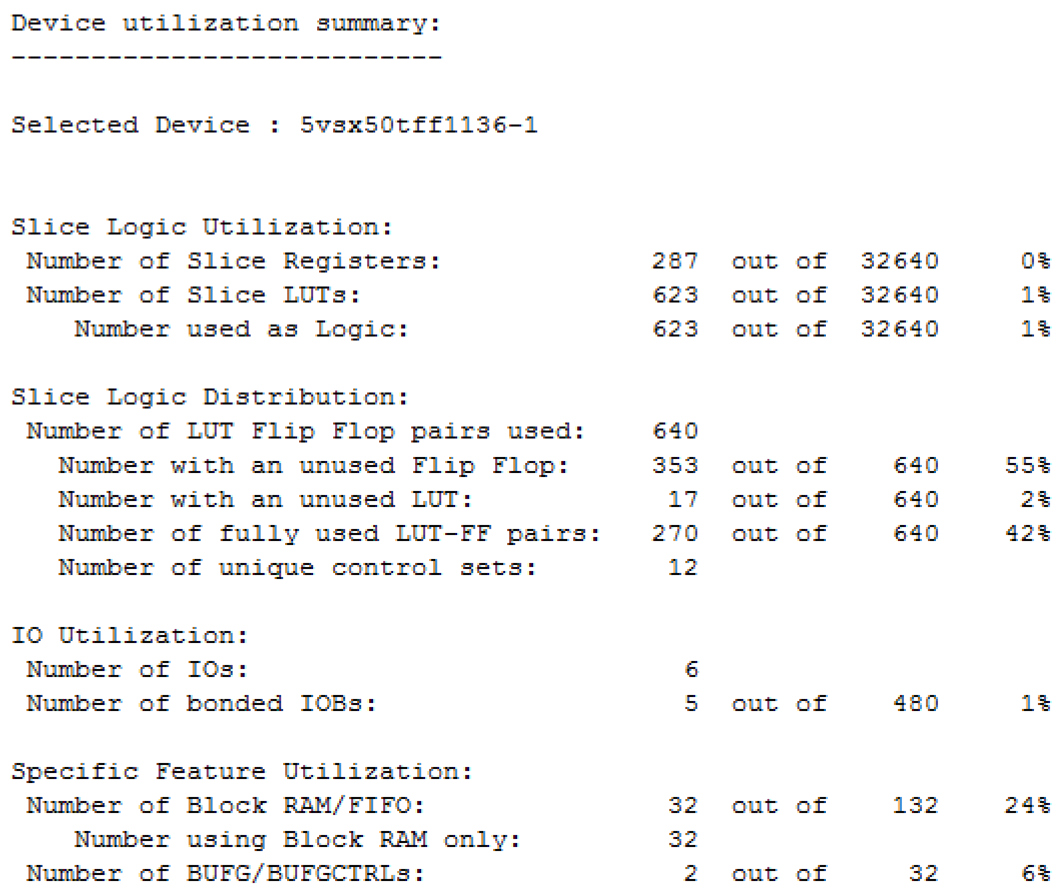
Name	Value
(28)	00010110
(27)	00100000
(26)	00100110
(25)	00101000
(24)	00101000
(23)	00101010
(22)	00101010
(21)	00100100
(20)	00100011
(19)	00100000
(18)	00011101
(17)	00011010
(16)	00011001
(15)	00011000
(14)	00011000
(13)	00011000
(12)	00011001
(11)	00011001
(10)	00011010
(9)	00011101
(8)	00011000
(7)	00011100
(6)	00011100
(5)	00011101
(4)	00011111
(3)	00011101
(2)	00011101
(1)	00011110
(0)	00011111
height	512

Figure 6.11: Memory contents

6.10 Synthesis Results

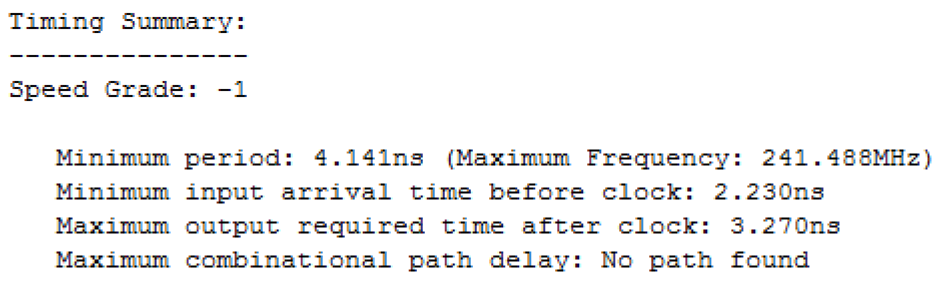
The second and most important stage of the implementation procedure was to synthesize our system on the Virtex-5 ML506 FPGA (XC5VSX50T). This was accomplished using Xilinx ISE 12.1. Passing successfully the synthesis stage means that our design can be downloaded on the FPGA fabric. Furthermore the ISE tool provides us with useful reports containing information regarding maximum allowed frequency, resource utilization, area cost, etc.

Figure 6.12 presents the device utilization summary that was generated by Xilinx ISE. Figure 6.13 shows the timing summary generated, including maximum allowed frequency. Figure 6.14 presents the schematic of the IDWT core as it was generated again by Xilinx ISE.



```
Device utilization summary:
---------------------------

Selected Device : 5vsx50tff1136-1


Slice Logic Utilization:
 Number of Slice Registers:             287  out of  32640     0%
 Number of Slice LUTs:                  623  out of  32640     1%
    Number used as Logic:               623  out of  32640     1%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    640
   Number with an unused Flip Flop:     353  out of    640    55%
   Number with an unused LUT:            17  out of    640     2%
   Number of fully used LUT-FF pairs:   270  out of    640    42%
   Number of unique control sets:        12

IO Utilization:
 Number of IOs:                           6
 Number of bonded IOBs:                   5  out of    480     1%

Specific Feature Utilization:
 Number of Block RAM/FIFO:               32  out of    132    24%
    Number using Block RAM only:         32
 Number of BUFG/BUFGCTRLs:                2  out of     32     6%
```

Figure 6.12: Device utilization summary



```
Timing Summary:
---------------
Speed Grade: -1

   Minimum period: 4.141ns (Maximum Frequency: 241.488MHz)
   Minimum input arrival time before clock: 2.230ns
   Maximum output required time after clock: 3.270ns
   Maximum combinational path delay: No path found
```

Figure 6.13: Timing Summary

These results are presented here only as a sample. They are produced by an instantiation of the core that uses a 256x25x8 bit Block Ram to store image pixels, therefore being able to apply the inverse wavelet transform in a 256x256 grayscale image. Detailed results are presented in Chapter 9 "Results and conclusions".

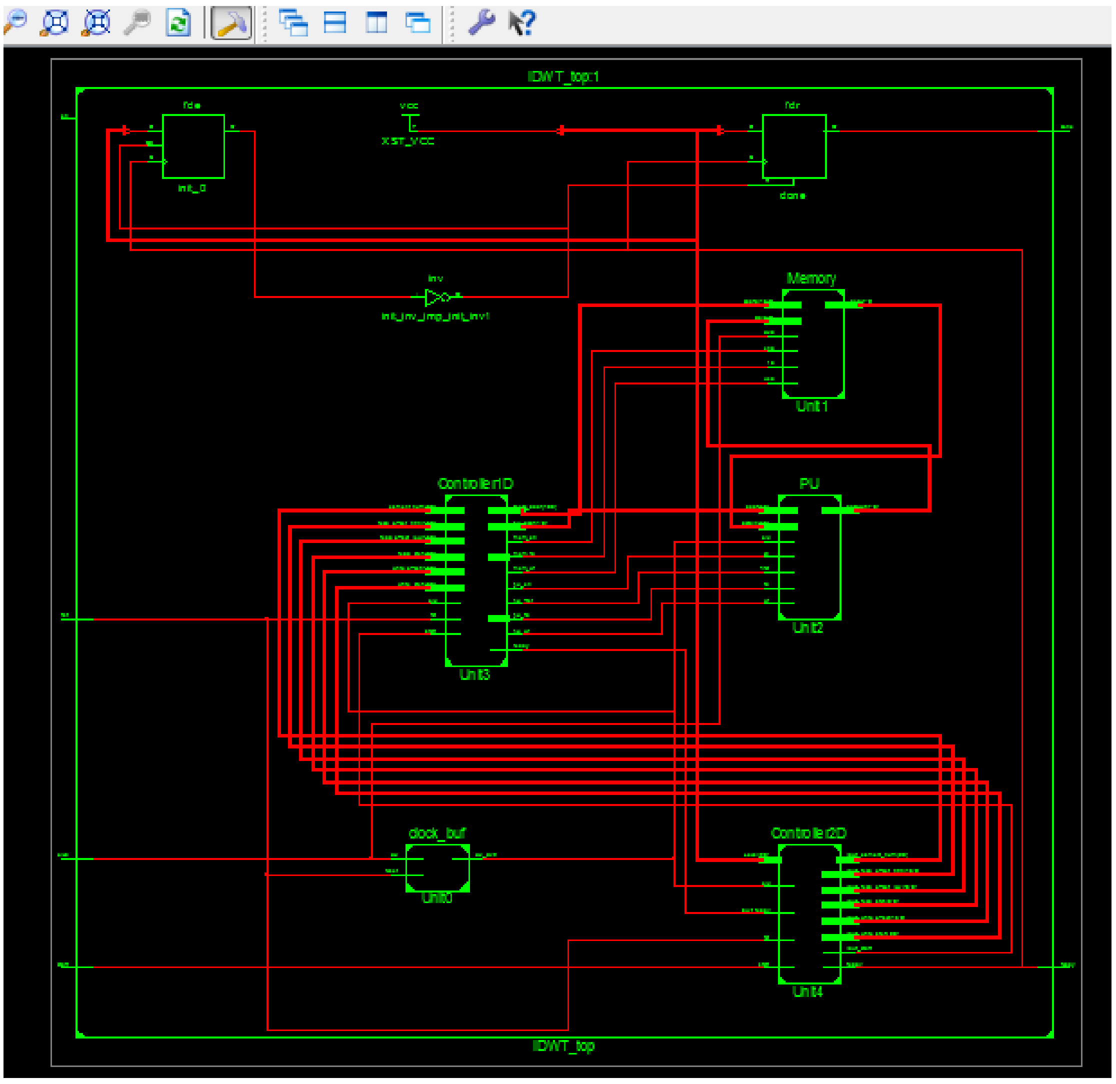


Figure 6.14: IDWT core schematic

Chapter 7

The Xilinx Embedded Development Kit (EDK)

Overview

Now that we have completed and tested the VHDL design, we need a set of tools that will bring HW and SW together, and that will help us manage the co-design flow in order to implement the JPEG2000 decoder on the Xilinx Virtex-5 ML506 board. The task of making the HW part communicate with the SW part and vice versa, and simultaneously meeting the requirements and constraints that are set at the early stages of the design, seems to be an extremely complex and time-consuming task. However, the Xilinx Embedded Development Kit (EDK) is a set of tools that provides the designer with the power to control the whole co-design flow, by automating tasks that would otherwise be almost impossible for the designer to accomplish within possible deadlines. Furthermore combining all the needed tools in one suite, and having them co-operate, significantly decreases the complexity of the HW/SW co-design. In this Chapter, a short description of the aims and capabilities of the EDK tool set is presented, and then a step-by-step guide is included that describes the whole procedure of implementing the JPEG2000 decoder on the board using the EDK tool set.

7.1 Introduction to EDK

The Xilinx Embedded Development Kit (EDK) is a suite of tools and Intellectual Property (IP) that enables you to design a complete embedded processor system for implementation in a Xilinx Field Programmable Gate Array (FPGA) device. Embedded systems are somewhat complex. Getting the hardware and software portions of an embedded design to work are projects in themselves. Merging the two design components so they function as one system brings additional challenges. Add an FPGA design project to the mix, and the situation has the potential to become very confusing indeed. For example, a typical Microblaze-based system (Figure 7.1) consists of a Microblaze soft-logic processor, an FPGA fabric, various IPs (RS232, Ethernet, LCD controllers, custom IPs), external memories (SRAM), a software that runs on Microblaze, C drivers for the hardware, etc. Similarly complex is a PowerPC based system (Figure 7.2). The description of all these components and their correct integration in a final working embedded system is a complex task indeed.

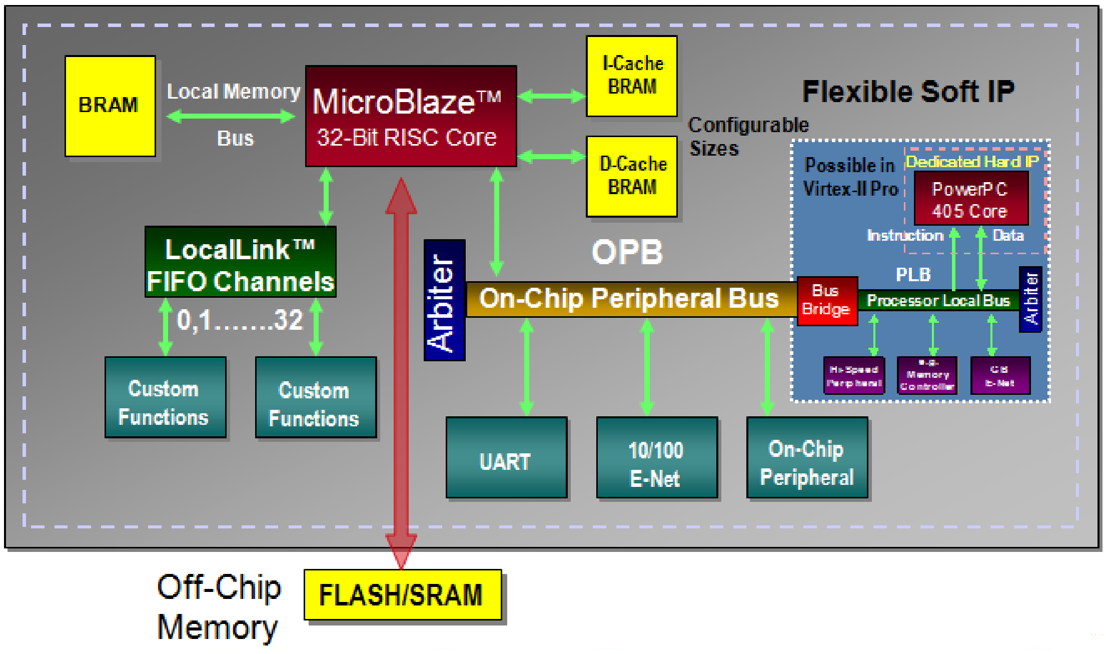


Figure 7.1: A typical Microblaze-based embedded system

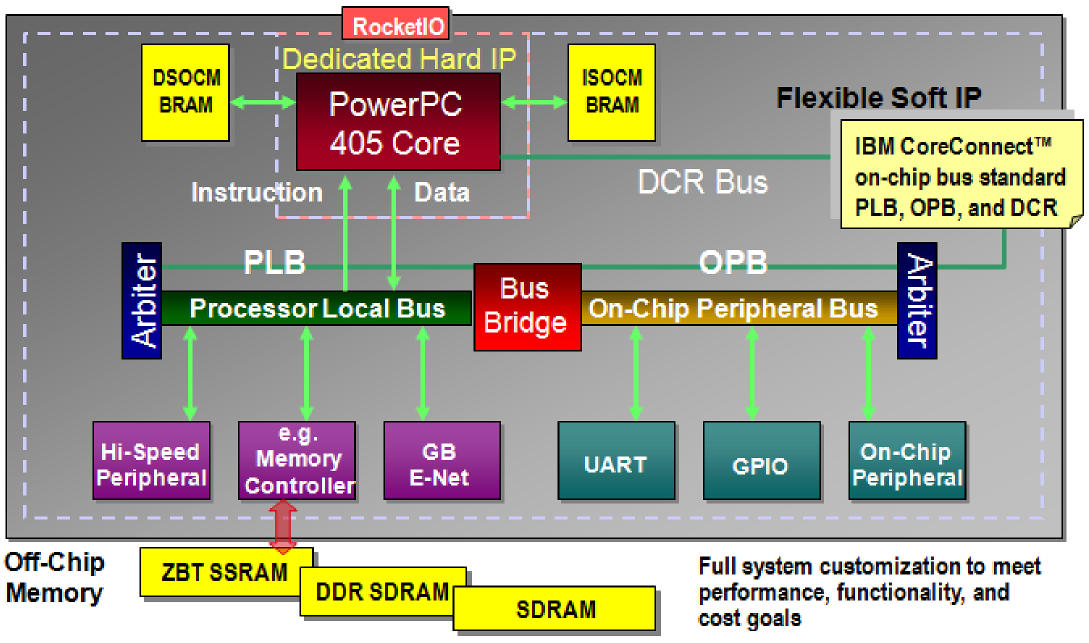


Figure 7.2: A typical PowerPC-based embedded system

To simplify the design process, Xilinx offers several sets of tools, mainly ISE (which was used previously to synthesize our VHDL design) and EDK. The development tools included in the EDK tool set can be classified into two major categories: those that manage HW design and those that are used for the development of SW. Xilinx Platform Studio (XPS) and Software Development Kit (SDK) are the basic development environments for each category. The tools provided with EDK are designed to assist in all phases of the embedded design process, as illustrated in Figure 7.3.

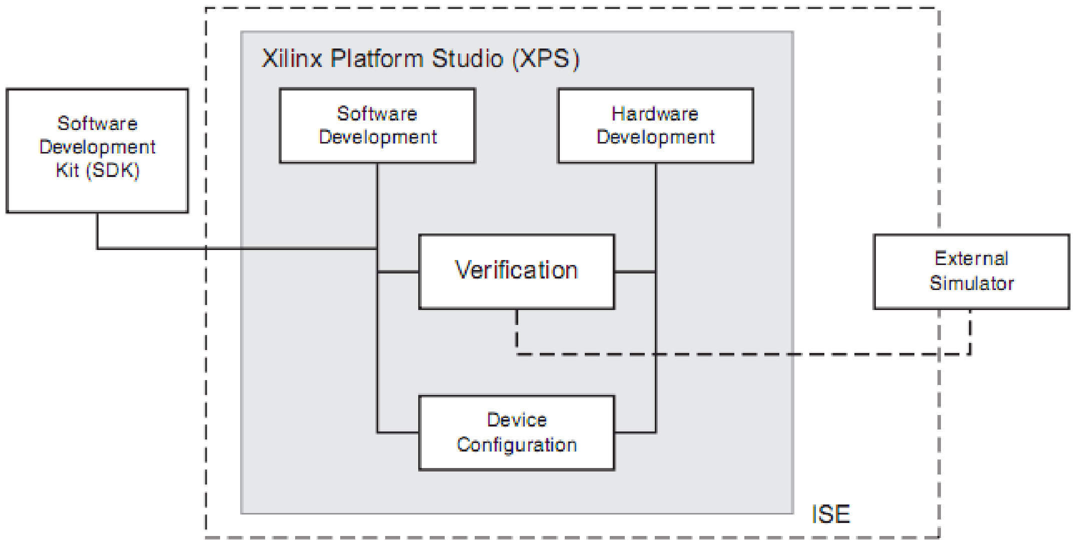


Figure 7.3: Basic Embedded Design Process Flow

The terms "Software Development" and "Hardware Development" that are present in Figure 7.3 are explained below.

Hardware Development

Xilinx FPGA technology allows the user to customize the hardware logic in the processor subsystem. Such customization is not possible using standard off-the-shelf microprocessor or controller chips. The term, "Hardware platform", describes the flexible, embedded processing subsystem the user is creating with Xilinx technology for her/his application needs. The hardware platform consists of one or more processors and peripherals connected to the processor buses. EDK captures the hardware platform in the Microprocessor Hardware Specification (MHS) file.

Software Development

A software platform is a collection of software drivers and, optionally, the operating system on which to build an application. The software image created consists only of the portions of the Xilinx library that the user uses in her/his embedded design. EDK captures the software platform in the Microprocessor Software Specification (MSS) file. The user can create multiple applications to run on the software platform.

7.1.1 Xilinx Platform Studio (XPS)

XPS provides an integrated environment for creating software and hardware specification flows for embedded processor systems based on MicroBlaze and PowerPC processors. XPS also provides an editor and a project management interface to create and edit source code. It offers customization of tool flow configuration options and provides a graphical system editor for connection of processors, peripherals, and buses. It is available on Windows®, Solaris®, and Linux platforms. There is also a batch mode invocation of XPS available. From XPS, the user can run all embedded system tools needed to process hardware and software system components. The designer can also perform system verification within the same environment. Figure 7.4 is a simple illustration of the managing role that XPS plays in HW/SW co-design and co-verification.

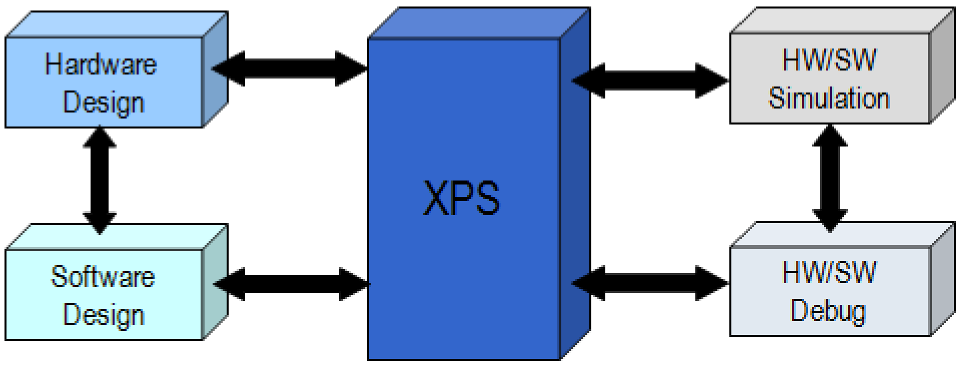


Figure 7.4: Xilinx Platform Studio (XPS)

XPS offers the following features:

• Ability to add cores, edit core parameters, and make bus and signal connections to generate an MHS file

• Ability to generate and modify the MSS file

• Ability to generate and view a system block diagram and/or design report

• Multiple-user software applications support

• Project management

• Process and tool flow dependency management

7.1.2 Software Development Kit (SDK)

The Xilinx Platform Studio SDK is a complementary GUI to XPS (Xilinx Platform Studio) and provides a development environment for software application projects. SDK is based on the Eclipse open-source standard. Platform Studio SDK features include:

• Feature-rich C/C++ code editor and compilation environment

• Project management

• Application build configuration and automatic makefile generation

• Error navigation

• Well integrated environment for seamless debugging and profiling of embedded targets

• Source code version control

7.1.3 Other EDK Components

Following is a list of some of the other EDK elements.

• Hardware IP for the Xilinx embedded processors

• Drivers and libraries for embedded software development

• GNU Compiler and debugger for C/C++ software development targeting the MicroBlaze™ and PowerPC™ processors

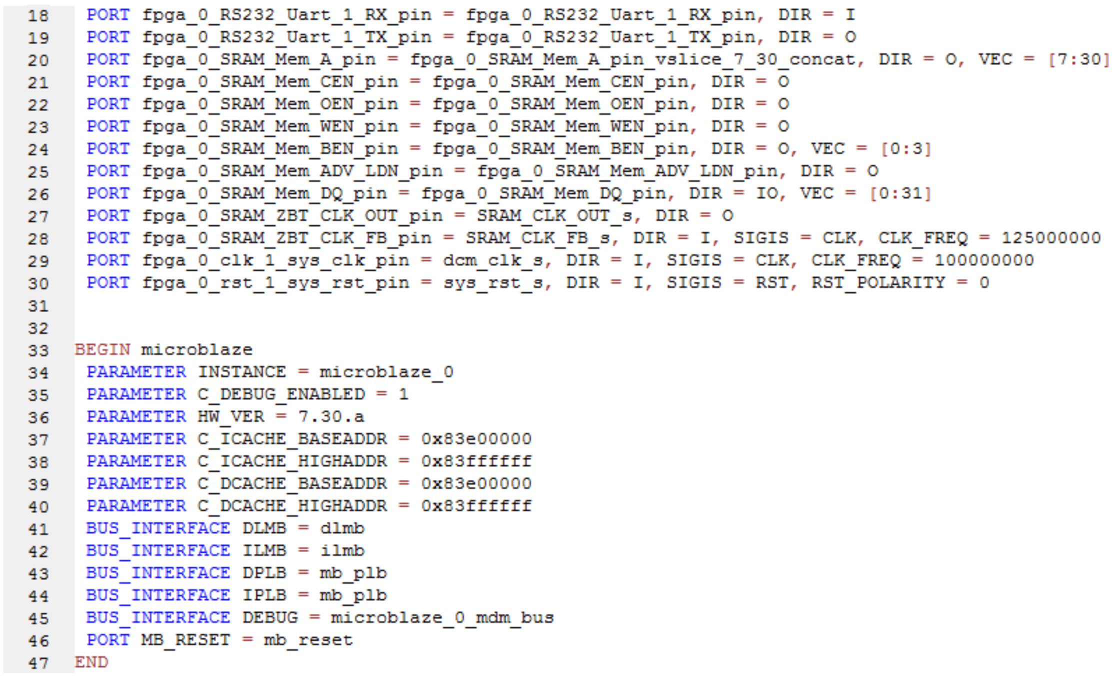
• Sample projects

7.2 The MHS and MSS Description Files

7.2.1 The MHS file and PlatGen

As stated before, the hardware platform is fully described by the Microprocessor Hardware Specification (MHS) file (ASCII text). The MHS file is integral to the design process. It contains all peripheral instantiations along with their parameters. The MHS file defines the configuration of the embedded processor system and includes information on the bus architecture, peripherals, processor, connectivity, and address space (see Figure 7.5). Peripherals are either provided from EDK as

Intellectual Property (IPs) or are developed and described by the user following specific instructions. The Hardware platform development is illustrated in Figure 7.6.



```
 PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX_pin, DIR = I
 PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX_pin, DIR = O
 PORT fpga_0_SRAM_Mem_A_pin = fpga_0_SRAM_Mem_A_pin_vslice_7_30_concat, DIR = O, VEC = [7:30]
 PORT fpga_0_SRAM_Mem_CEN_pin = fpga_0_SRAM_Mem_CEN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_OEN_pin = fpga_0_SRAM_Mem_OEN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_WEN_pin = fpga_0_SRAM_Mem_WEN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_BEN_pin = fpga_0_SRAM_Mem_BEN_pin, DIR = O, VEC = [0:3]
 PORT fpga_0_SRAM_Mem_ADV_LDN_pin = fpga_0_SRAM_Mem_ADV_LDN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_DQ_pin = fpga_0_SRAM_Mem_DQ_pin, DIR = IO, VEC = [0:31]
 PORT fpga_0_SRAM_ZBT_CLK_OUT_pin = SRAM_CLK_OUT_s, DIR = O
 PORT fpga_0_SRAM_ZBT_CLK_FB_pin = SRAM_CLK_FB_s, DIR = I, SIGIS = CLK, CLK_FREQ = 125000000
 PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
 PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 0


BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER HW_VER = 7.30.a
 PARAMETER C_ICACHE_BASEADDR = 0x83e00000
 PARAMETER C_ICACHE_HIGHADDR = 0x83ffffff
 PARAMETER C_DCACHE_BASEADDR = 0x83e00000
 PARAMETER C_DCACHE_HIGHADDR = 0x83ffffff
 BUS_INTERFACE DLMB = dlmb
 BUS_INTERFACE ILMB = ilmb
 BUS_INTERFACE DPLB = mb_plb
 BUS_INTERFACE IPLB = mb_plb
 BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
 PORT MB_RESET = mb_reset
END
```

Figure 7.5: Microprocessor Hardware Specification (MHS)

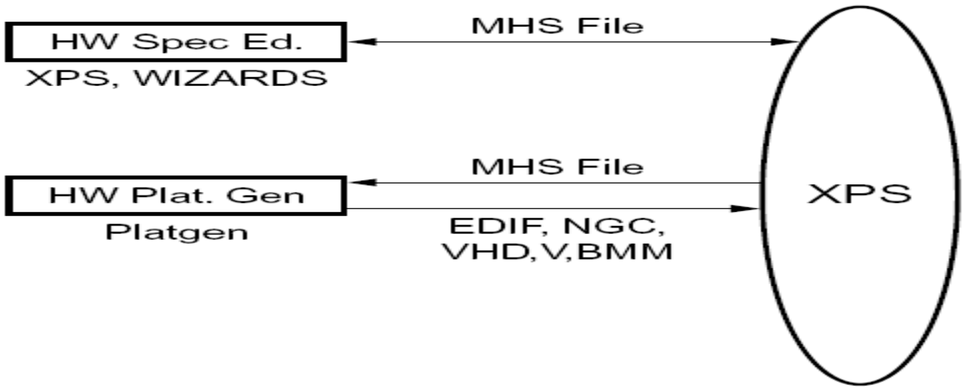
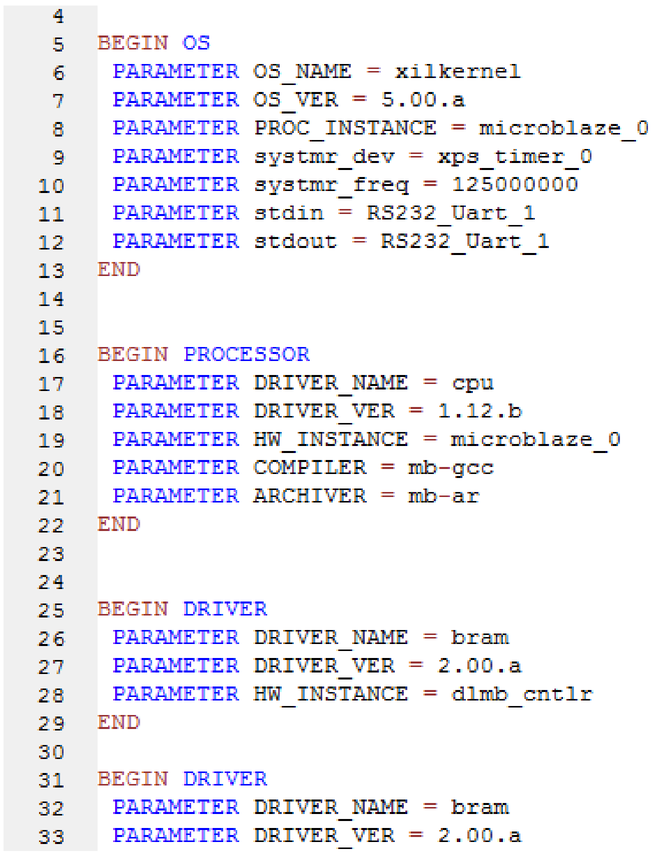


Figure 7.6: Hardware Platform Development

The Platform Generator (PlatGen) tool creates the hardware platform using MHS as its input. Platgen also reads various processor core (IP core) hardware description files (MPD, PAO) from the EDK library and any user IP repository. Platgen produces the top-level HDL design file for the embedded system that stitches together all the instances of parameterized IP cores contained in the system. In the process, it resolves all the high-level bus connections in the MHS into the actual signals required to interconnect the processors, peripherals and on-chip memories. It also invokes the XST (Xilinx Synthesis Technology) compiler to synthesize each of the instantiated IP cores. PlatGen generates all the netlist files (NGC, EDIF) plus VHDL files that allow the user to add custom logic to the system. These files along with other tools (like XST,) that can be seen in Figure 7.7 (end of chapter) generate the bitstream that will eventually configure the device.

7.2.2 The MSS file and LibGen

Like MHS, XPS creates an analogous software system description in the Microprocessor Software Specification (MSS) file. The MSS file, together with the user's software applications, are the principal source files (written in C/C++ or assembly) representing the software elements of the embedded system (Figure 7.7).



```

BEGIN OS
 PARAMETER OS_NAME = xilkernel
 PARAMETER OS_VER = 5.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER systmr_dev = xps_timer_0
 PARAMETER systmr_freq = 125000000
 PARAMETER stdin = RS232_Uart_1
 PARAMETER stdout = RS232_Uart_1
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.12.b
 PARAMETER HW_INSTANCE = microblaze_0
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = dlmb_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 2.00.a
```

Figure 7.7: Microprocessor Software Specification (MSS)

This collection of files, used in conjunction with EDK installed libraries and drivers, and any custom libraries and drivers for custom peripherals the user provides allows SDK to compile the applications. The compiled software routines are available as an Executable and Linkable Format (ELF) file. The ELF file is the binary ones and zeros that are run on the processor hardware. Figures 7.5 and 7.6 illustrate the software platform development and the files and flow stages that generate the ELF file.

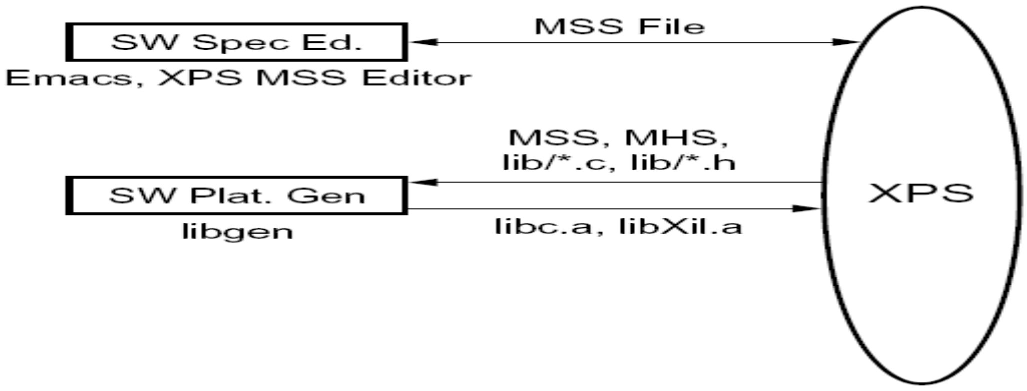


Figure 7.8: Software platform development

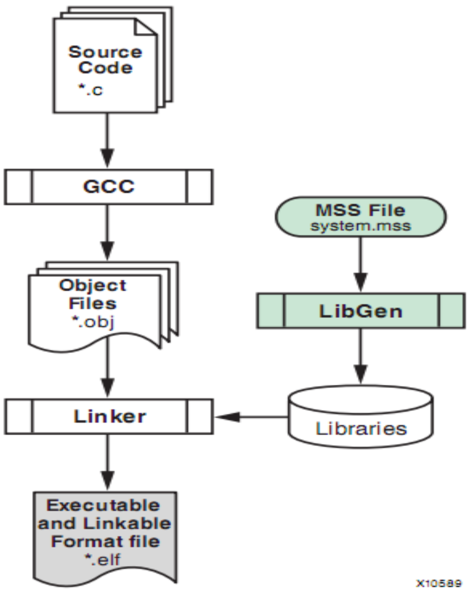


Figure 7.9: ELF file generation

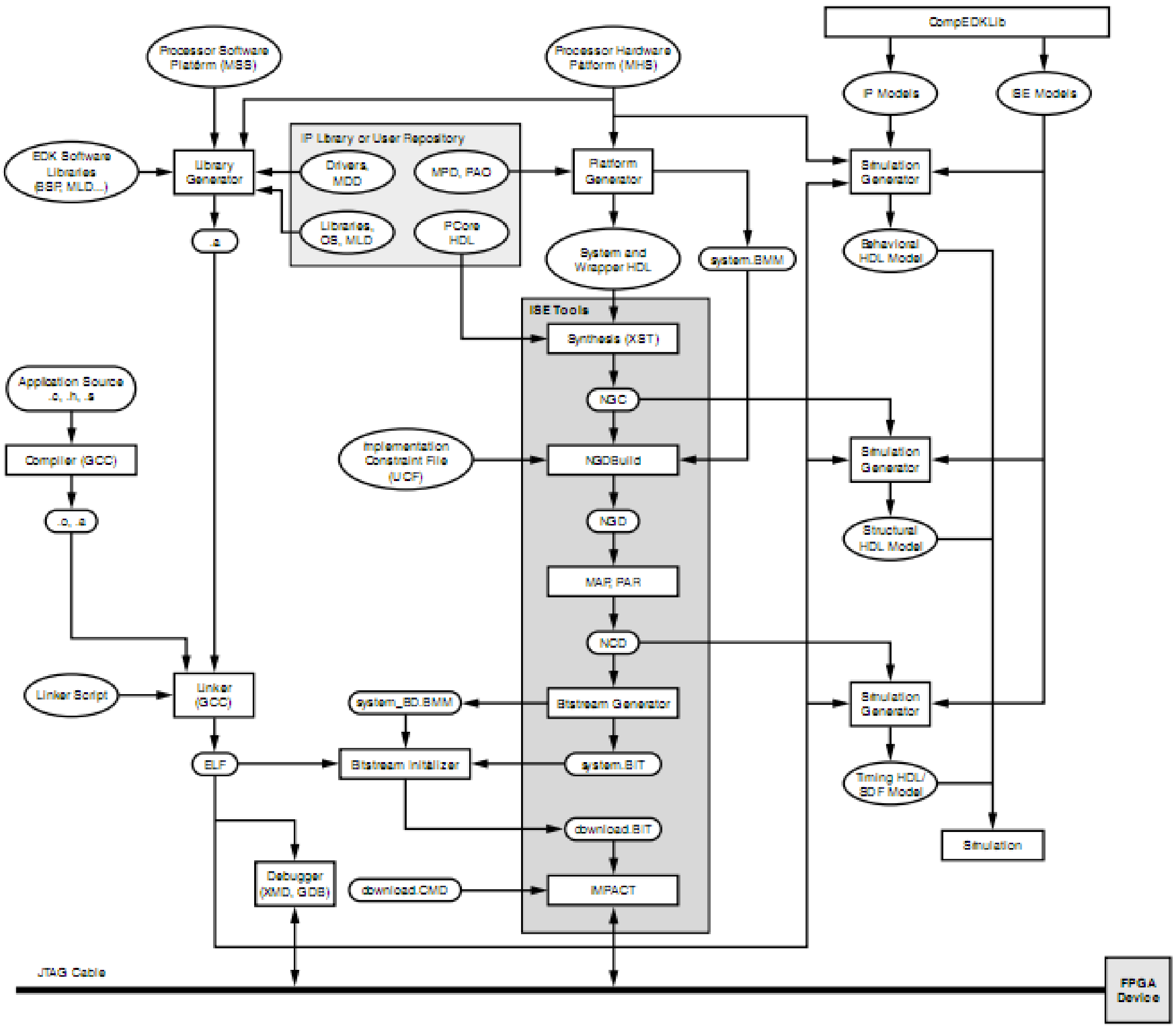


Figure 7.10: Embedded Development Kit Tools (EDK) Architecture

Chapter 8

JPEG2000 Co-design using EDK

Overview

In this chapter we present the basic components of the co-design architecture and introduce some basic structures that are essential to the whole system. This chapter also aims to provide a first understanding of the steps that are going to be presented in the step-by-step guide that follows.

8.1 Introduction

Before proceeding to the co-design architecture and the step-by-step guide that will follow in the next chapter, it would be wise to provide some information regarding some basic system components such as the Microblaze soft-logic processor, the Processor Local Bus (PLB), the Xilinx Kernel Operating System, etc.

8.1.1 The Microblaze Processor

The MicroBlaze™ embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs). Figure 8.1 shows a functional block diagram of the MicroBlaze core.

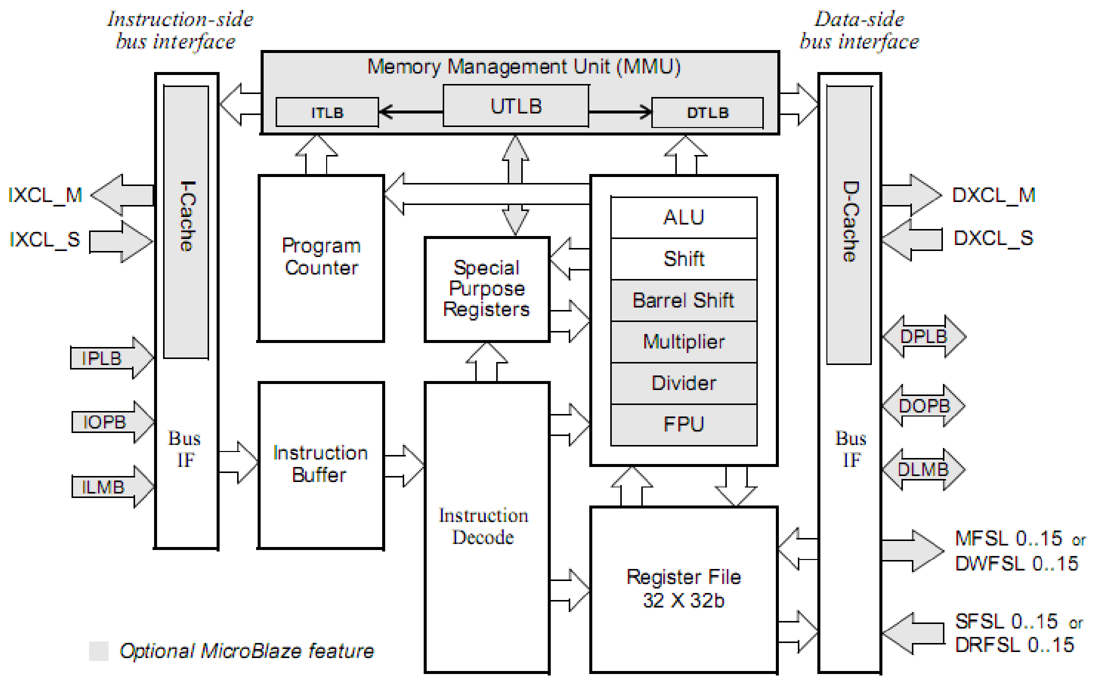


Figure 8.1: MicroBlaze Core Block Diagram

Features

The MicroBlaze soft core processor is highly configurable, allowing the user to select a specific set of features required by her/his design.

The fixed feature set of the processor includes:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus
- Single issue pipeline

Memory Architecture

MicroBlaze is implemented with a Harvard memory architecture; instruction and data accesses are done in separate address spaces. Each address space has a 32-bit range (that is, handles up to 4-GB of instructions and data memory respectively). The instruction and data memory ranges can be made to overlap by mapping them both to the same physical memory. The latter is useful for software debugging. Both instruction and data interfaces of MicroBlaze are 32 bits wide and use big endian, bit-reversed format. MicroBlaze supports word, half-word, and byte accesses to data memory.

Data accesses must be aligned (word accesses must be on word boundaries, halfword on half-word boundaries), unless the processor is configured to support unaligned exceptions. All instruction accesses must be word aligned. MicroBlaze does not separate data accesses to I/O and memory (it uses memory mapped I/O). The processor has up to three interfaces for memory accesses:

- Local Memory Bus (LMB)
- Processor Local Bus (PLB) or On-Chip Peripheral Bus (OPB)
- Xilinx CacheLink (XCL)

The LMB memory address range must not overlap with PLB, OPB or XCL ranges. MicroBlaze has single cycle latency for accesses to local memory (LMB) and for cache read hits, except with area optimization enabled when data side accesses and data cache read hits require two clock cycles. A data cache write normally has two cycles of latency (more if the posted-write buffer in the memory controller is full).

The MicroBlaze instruction and data caches can be configured to use 4 or 8 word cache lines. When using a longer cache line, more bytes are prefetched, which generally improves performance for software with sequential access patterns. However, for software with a more random access pattern the performance can instead decrease for a given cache size. This is caused by a reduced cache hit rate due to fewer available cache lines.

8.1.2 Bus Interfaces

The MicroBlaze core is organized as a Harvard architecture with separate bus interface units for data and instruction accesses. The following three memory interfaces are supported: Local Memory Bus (LMB), the IBM Processor Local Bus (PLB) or the IBM On-chip Peripheral Bus (OPB), and Xilinx® CacheLink (XCL). The LMB provides single-cycle access to on-chip dual-port block RAM. The PLB and OPB interfaces provide a connection to both on-chip and off-chip peripherals and memory. The CacheLink interface is intended for use with specialized external memory controllers. MicroBlaze also supports up to 16 Fast Simplex Link (FSL) ports, each with one master and one slave FSL interface.

Features

- A 64-bit version of the PLB V4.6 interface (see IBM's 128-Bit Processor Local Bus Architectural Specifications, Version 4.6).
- A 64-bit version of the OPB V2.0 bus interface (see IBM's 64-Bit On-Chip Peripheral Bus,
- Architectural Specifications, Version 2.0)
- LMB provides simple synchronous protocol for efficient block RAM transfers
- FSL provides a fast non-arbitrated streaming communication mechanism
- XCL provides a fast slave-side arbitrated streaming interface between caches and external memory controllers
- Debug interface for use with the Microprocessor Debug Module (MDM) core
- Trace interface for performance analysis

8.1.2.1 Processor Local Bus (PLB)

The Xilinx 128-bit Processor Local Bus (PLB) v4.6 provides bus infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB system. It consists of a bus control unit, a watchdog timer, and separate address, write, and read data path units, as well as an optional DCR (Device Control Register) slave interface to provide access to its bus error status registers.

8.1.2.2 Local Memory Bus (LMB)

The LMB is a synchronous bus used primarily to access on-chip block RAM. It uses a minimum number of control signals and a simple protocol to ensure that local block RAM are accessed in a single clock cycle. LMB signals and definitions are shown in the following table. All LMB signals are active high.

8.1.3 Xilkernel Operating System

Xilkernel is a small, robust, and modular kernel. It is highly integrated with the Platform Studio framework and is a free software library that you get with the Xilinx EDK. It allows a very high degree of customization, letting users tailor the kernel to an optimal level both in terms of size and functionality. It supports the core features required in a lightweight embedded kernel, with a POSIX API. Xilkernel works on both the MicroBlaze™ and PowerPC™ 405 processors. Xilkernel IPC services can be used to implement higher level services (such as networking, video, and audio) and subsequently run applications using these services.

Key Features

- A POSIX API targeting embedded kernels.
- Core kernel features such as:
    - POSIX threads with round-robin or strict priority scheduling
    - POSIX synchronization services - semaphores and mutex locks
    - POSIX IPC services - message queues and shared memory
    - Dynamic buffer pool memory allocation
    - Software timers
    - User level interrupt handling API
- Highly robust kernel, with all system calls protected by parameter validity checks and proper return of POSIX error codes.
- Highly scalable kernel that can be accommodated into a given system through the inclusion or exclusion of functionality as required.
- Complete kernel configuration, deployment within minutes from inside of Platform Studio
- Statically creating threads that startup with the kernel.
- System call interface to the kernel.
- Support for creating processes out of separate executable Executable Link Files (ELF)

Figure 8.2 shows the various modules of Xilkernel

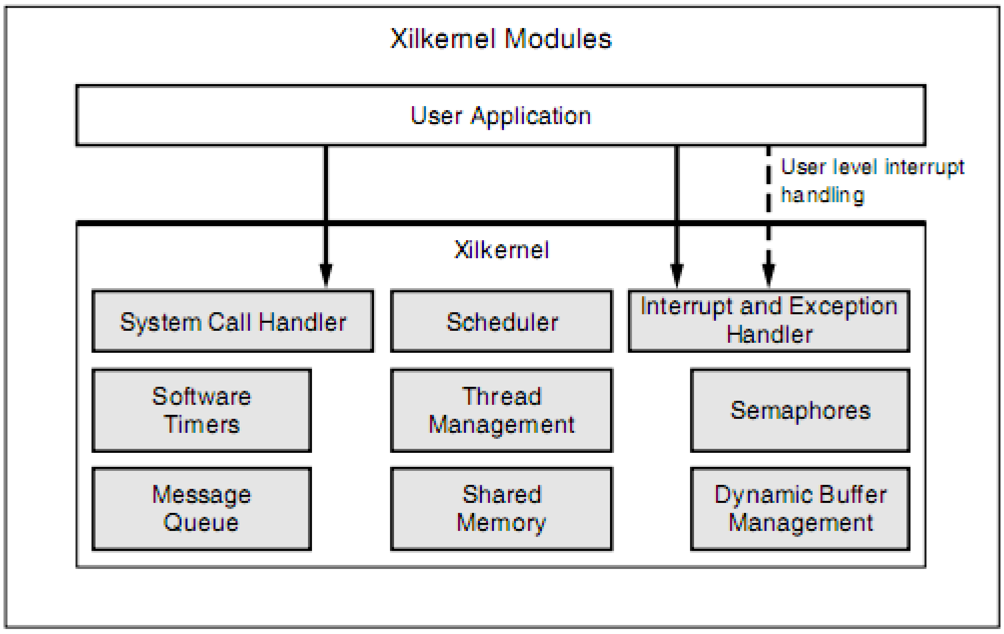


Figure 8.2: Xilkernel Modules

8.2 Co-design Architecture

The system architecture is decided after HW/SW partitioning. As stated before, Microblaze was selected as the target processor to run the software (JasPer) and an FPGA fabric core called IDWT, was synthesized to implement the Inverse Discrete Wavelet Transform functions that were ported to HW. Figure 8.3 illustrates an abstract architectural model of the system. Bus interfaces are not introduced yet.

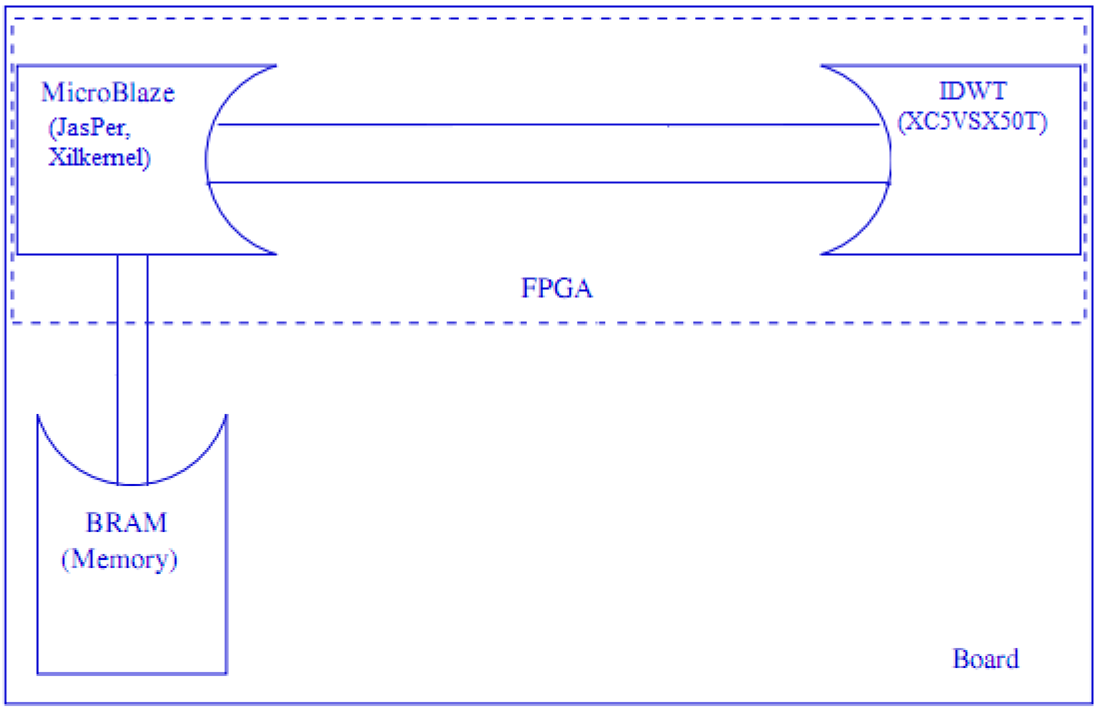


Figure 8.3: Architectural Model

The final System on Chip (SoC) included an SRAM in order to store the input and output images, a DDR2 SDRAM in order to store the executable ELF file (1,3MB), and a RS232 UART serial port for sending the output image to the host PC. The final block diagram is presented at the end of this chapter.

8.2.1 Communication Protocol

In Figure 8.3 an abstract architectural model of the system was presented. Figure 8.4 shows the communication model after insertion of communication protocols. The Local Memory Bus (LMB) and Processor Local Bus (PLB) protocols that were presented previously are used in this project. The Microblaze soft processor, IDWT, and BRAM are interconnected via the system bus. All components connected to the same bus are clocked at the same speed. Interfaces are inserted between bus and components. PLB is the interface between Microblaze and the system bus, whilst the one between the BRAM and Microblaze is LMB. The interface negotiates between components to ensure a successful completion of data transfers.

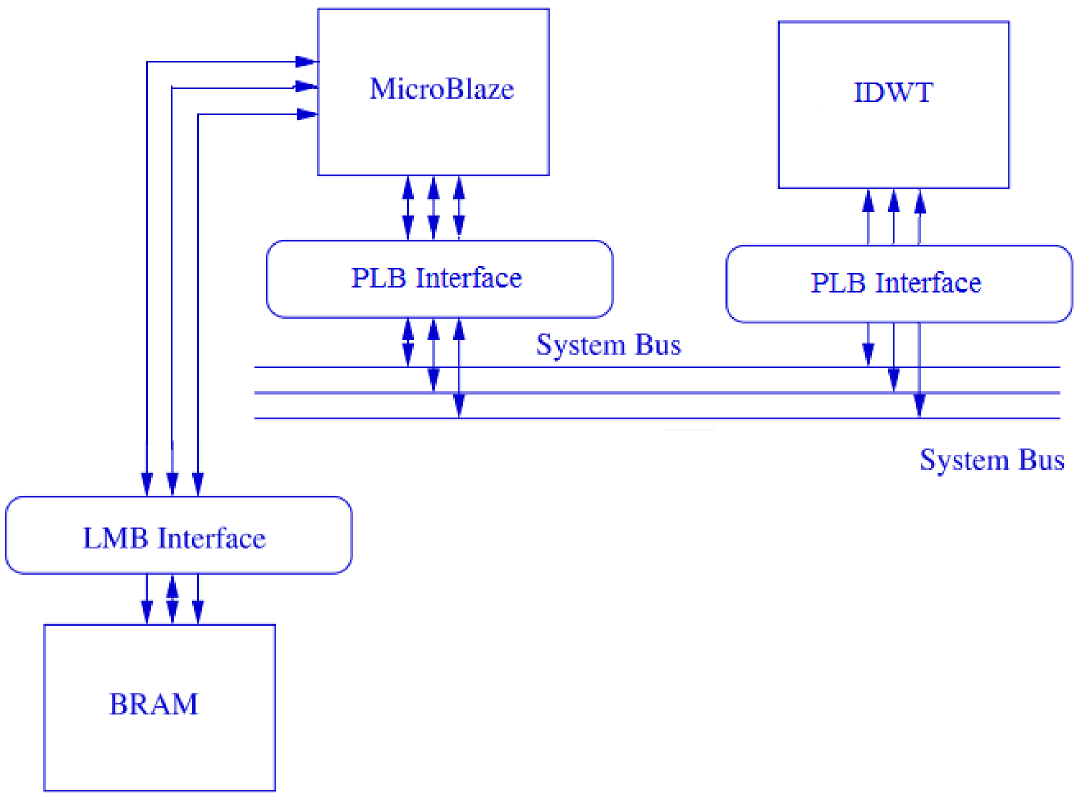


Figure 8.4: Communication Model

8.2.2 Microblaze/IDWT Interface

The communication between the Microblaze soft processor and IDWT (which is an external device) is based on the Processor Local Bus protocol of the Xilinx Virtex-5 platform. The protocol has 2 representations: **Master** and **Slave** bus protocols. PLB peripherals are created to work either as slaves or masters for the PLB. A peripheral connected to the master ports of the PLB pushes data and control signals onto the bus, whereas a peripheral that is connected to the slave ports reads and pops data and control signals from the PLB. The working idea behind the PLB bus system is shown in Figure 8.5 with an example of PLB connections in a system with three masters and three slaves.

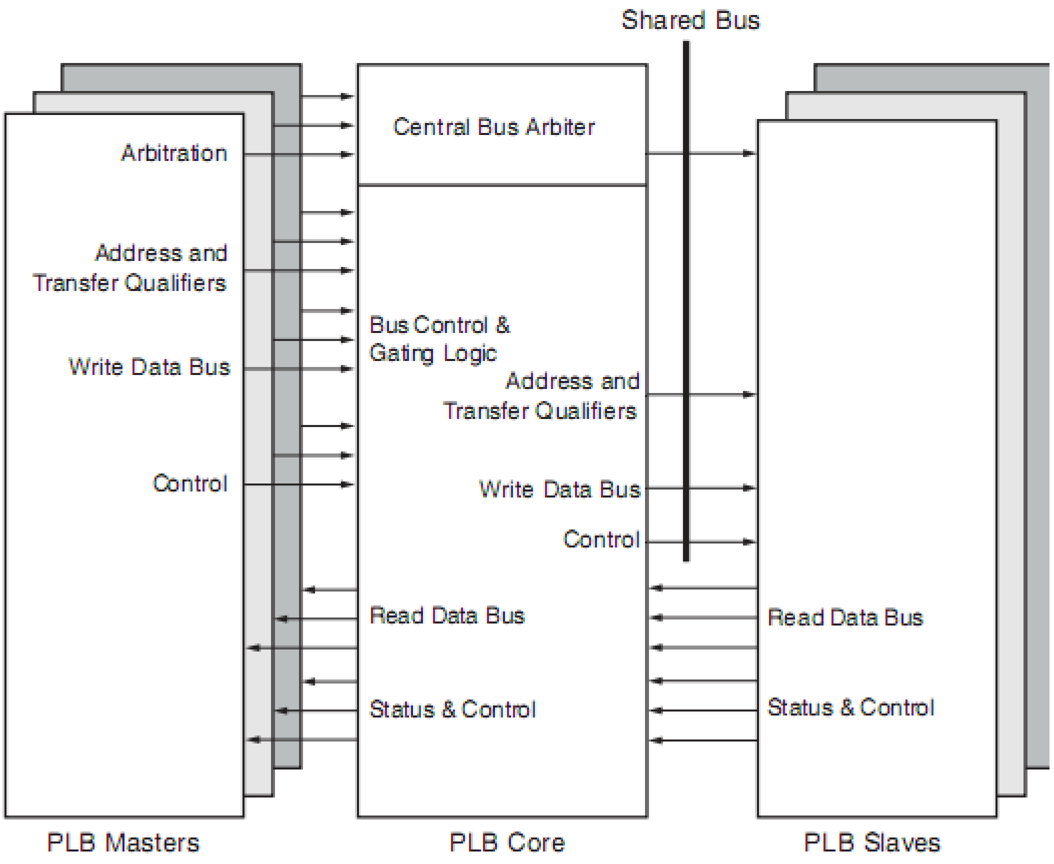


Figure 8.5: PLB interface

8.2.2.1 Master Bus Protocol

The Processor Local Bus protocol that was used for this project is too complex to present a detailed list of the essential master bus signals that hook up the Microblaze soft processor on the system bus. It would be also futile to present just a subset of the master bus signals as this wouldn't provide a realistic understanding of the working idea behind PLB.

8.2.2.2 Slave Bus Protocol

In contrast to the master bus signals for Microblaze, a subset of the slave bus signals can be presented and still provide a clear and simple view of how the external device (IDWT) communicates via the PLB with Microblaze. This is mostly due to the fact that EDK uses what is called PLB slave and burst peripherals to implement common functionality among various processor peripherals. These PLB slave and burst peripherals can act as bus masters or bus slaves. The PLB slave and burst peripherals are verified, optimized, and highly parameterizable interfaces. They also provide a set of simplified bus protocols. This is all IP Interconnect (IPIC), which is much easier to work with when compared to operating on the PLB or FSL bus protocols directly. Figure 8.6 illustrates the relationship between the bus, a simple PLB slave peripheral, IPIC and the user IP design.

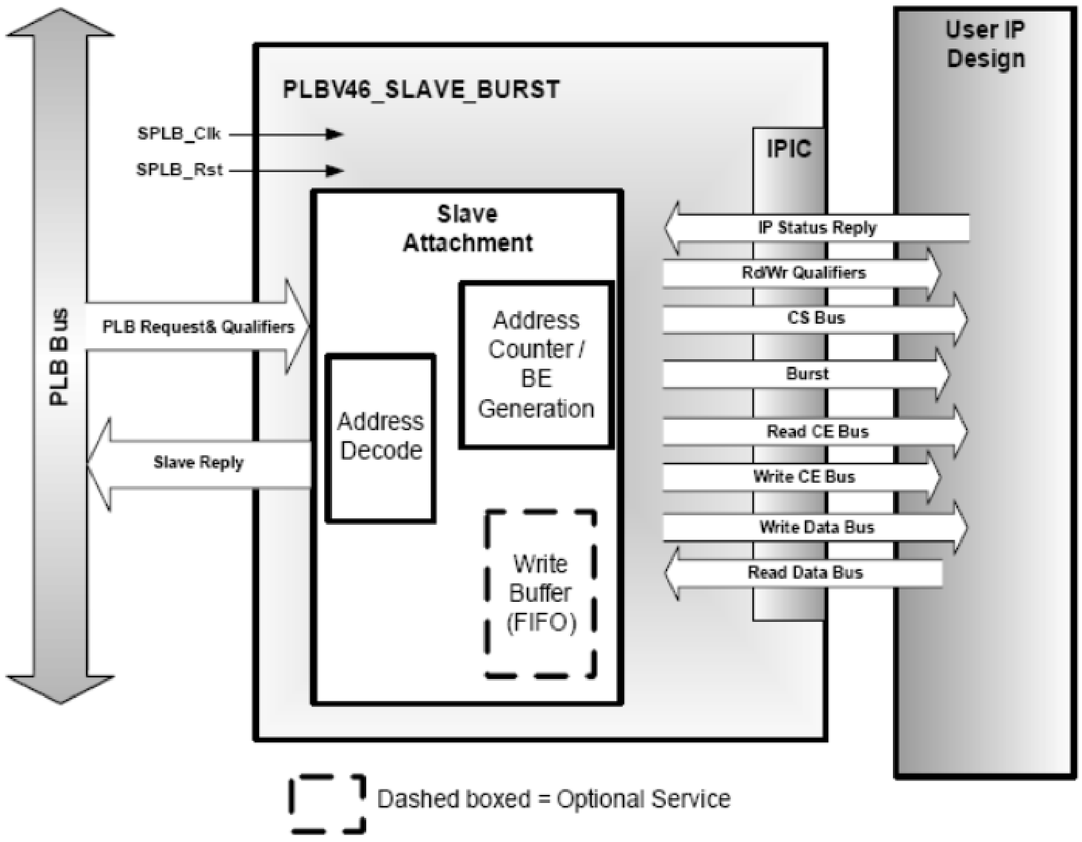


Figure 8.6: PLB Slave Module

A subset of the slave bus signals is presented below:

Bus2IP_Clk: slave device clock

Bus2IP_Reset: slave device reset

Bus2IP_Addr: bus to IP address for writing to and reading from user IP memory

Bus2IP_Data: bus to IP data bus, which pushes data into the user design

Bus2IP_RNW: bus to IP read/not write

Bus2IP_BE: bus to IP byte enables

Bus2IP_CS: bus to IP chip select for user IP memory selection

Bus2IP_RdCE: bus to IP read chip enable

Bus2IP_WrCE: bus to IP write chip enable

IP2Bus_Data: IP to Bus data bus, which pops data back to the main bus

IP2Bus_RdAck: IP to Bus read transfer acknowledgement

IP2Bus_WrAck: IP to Bus write transfer acknowledgement

IP2Bus_Error: IP to Bus error response

Using the above protocol the IDWT IP core is able to exchange data with Microblaze and connect to appropriate control signals that trigger the beginning of processing or mark the end of processing by the core.

The following figure (Figure 8.7) illustrates the final System on Chip (SoC), including basic components, memories and bus communication protocols.

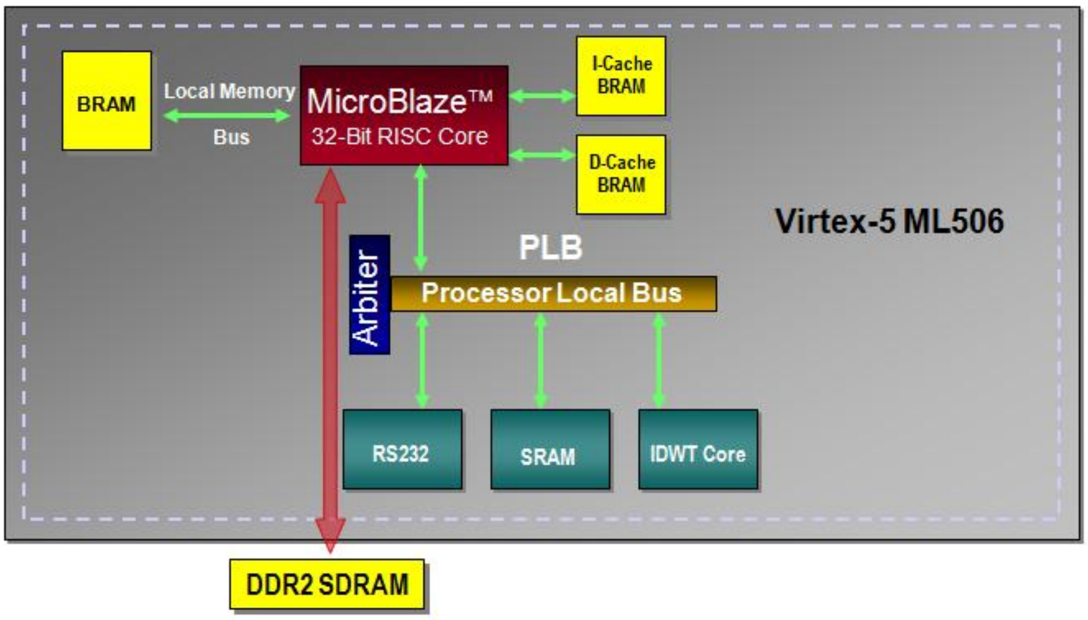


Figure 8.7: Final System on Chip

Chapter 9

Implementation Results

Overview

In this chapter we present a set of detailed results for the IDWT core that has been designed and the final SoC that has been developed for the JPEG2000 compression standard. The results refer to power consumption, slice utilization and performance in terms of maximum frequency achieved. For the purposes of this thesis, the final SoC was implemented and tested on a Virtex-5 XC5VSX50T board (Virtex-5 SX sub-family). However, results have been acquired and analyzed for other platforms as well. These include: Virtex-5 XC5VLX110T (Virtex-5 LX sub-family), Virtex-6 XC6VLX75T (Virtex-6 LX sub-family), Virtex-6 XC6VSX315T (Virtex-6 SX sub-family) and Spartan 6 XC6SLX100. Finally, the speed-up of the JPEG2000 decoder is calculated, after we implement the application on the Virtex-5 SX development platform.

9.1 Synthesis Design Goals and Strategies

In order to evaluate the impact of different implementation strategies on area, speed and power estimations we used three different and predefined by XST implementation strategies:

Area Reduction: this strategy sets an **area-oriented** goal for the synthesizer. The area reduction strategy will try to minimize area while enabling the physical synthesis options available in map. The tools perform logical optimizations on the design in order to achieve area requirements.

Timing Performance: this strategy sets a **speed-oriented** goal for the synthesizer. The timing performance strategy will try to achieve timing closure while enabling the physical synthesis options available in map. It can also try to achieve timing closure while packing registers into the IOBs if possible.

Power Optimization: this strategy sets a **power-oriented** goal for the synthesizer. The power optimization strategy will try to minimize power while enabling the physical synthesis options available in map. The tools perform logical optimizations on the design in order to achieve power reduction.

9.2 IDWT Core Results

The application specific nature of the JPEG2000 implementation requires the embedding of the IDWT core to the final SoC as an instantiation that can manage 64x64 image tiles. However, the core can be altered in order to be able to manipulate other image sizes, and more specifically 128x128, 256x256 and 512x512 grayscale images. The following results refer to 4 different instantiations of the core being implemented on the platforms mentioned above. For the extraction of these results, ISE 12.1 design suite was used. All results are extracted after Place And Route (PAR) has taken place.

9.2.1 Slice Utilization Results

The following diagram (Figure 9.1) depicts the slice utilization on different platforms and different instantiations of the IDWT core that can manage different image sizes. For this set of results, the synthesizer's goal was set to be speed-oriented and the optimization effort was set to "High".

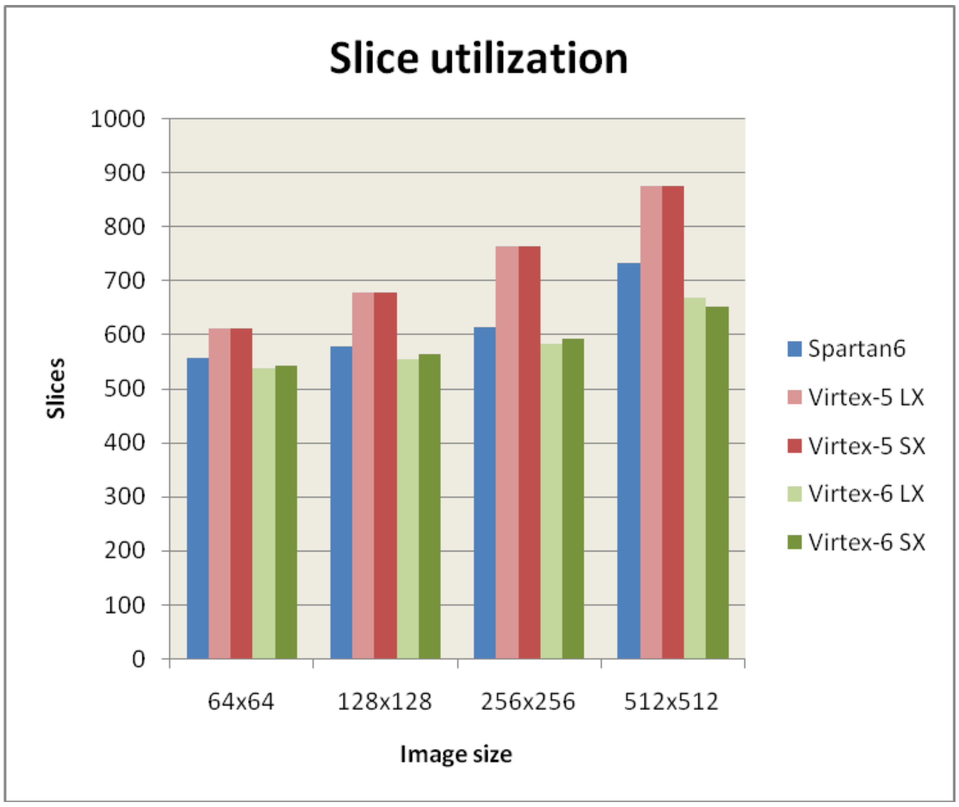


Figure 9.1: Slice utilization (speed-oriented optimization)

A first conclusion is that the IDWT core utilizes only 0.5% - 1.5% of the total slices available in every case. This is not the case with BRAM resources that are used, as the instantiation that manages 512x512 image sizes requires more than 80% of the total BRAMs available in every device, apart from the Virtex-6 SX FPGA. This means that the core cannot be set to work on 1024x1024 images in the rest devices.

Second, it is clearly seen that the implementation on Virtex-6 and Spartan-6 devices requires less slices in comparison to implementing the design on Virtex-5 devices. This difference is attributed to the fact that Virtex-5 slices contain 4 6-input LUTs and 4 Flip Flops each, whereas Virtex-6 and Spartan-6 technologies utilize slices that contain 4 6-input LUTS and 8 Flip Flops each. The increase in the number of Flip Flops leads to fewer slices being utilized in these devices.

The following diagrams (Figure 9.2, Figure 9.3 and Figure 9.4) illustrate the slice utilization results for each of the above FPGA families separately. However, in these diagrams the impact of the synthesizer's goal can be observed, as it switches between area-oriented and speed-oriented settings. The optimization effort is kept to "High".

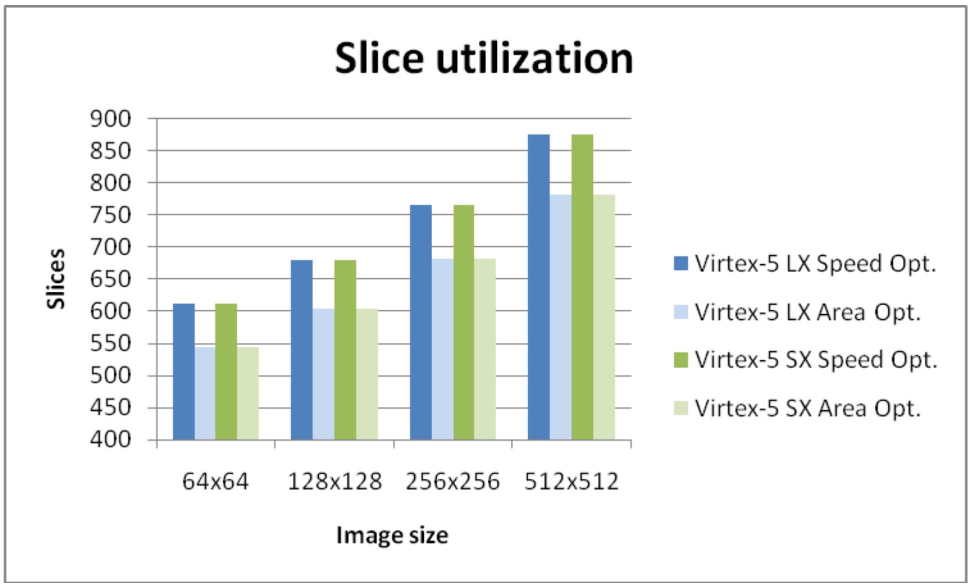


Figure 9.2: Slice utilization – Virtex-5

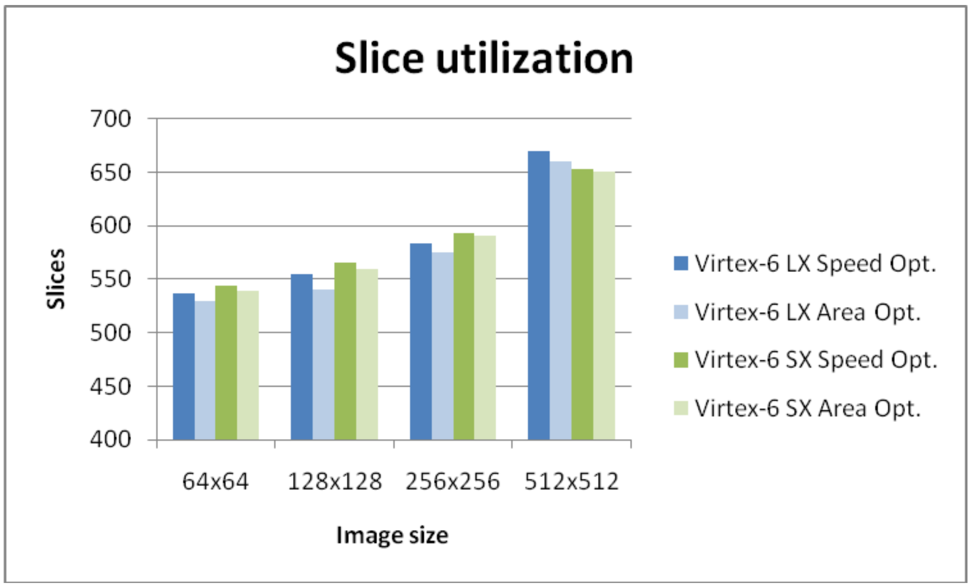


Figure 9.3: Slice utilization – Virtex-6

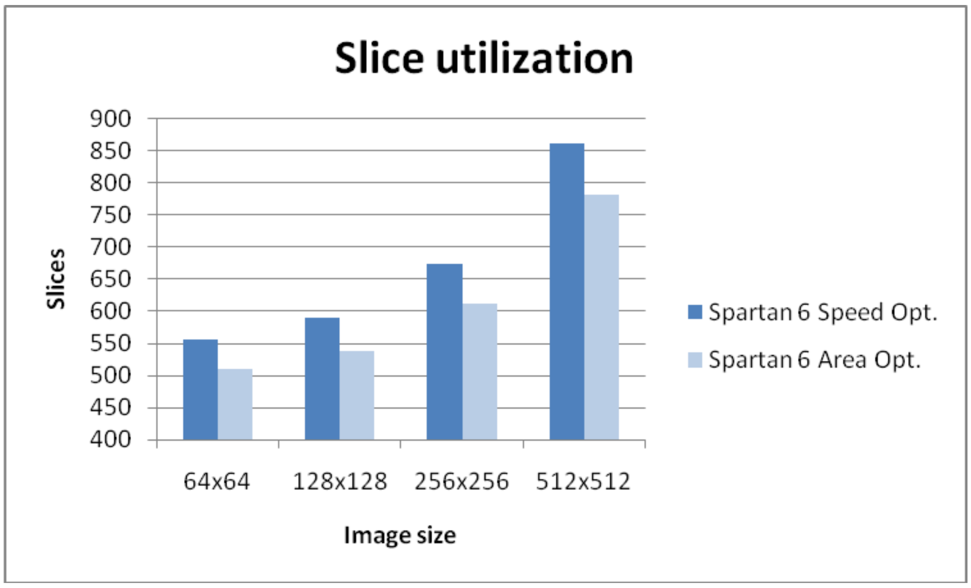


Figure 9.4: Slice utilization – Spartan 6

The above diagrams show that there is a decrease factor in slice number when the algorithm switches from speed to area oriented, especially in Virtex-5 and Spartan 6 devices. More precisely:

Virtex-5: 10% slice number reduction

Virtex-6: 2% slice number reduction

Spartan 6: 9% slice number reduction

9.2.2 Performance Results

In order to evaluate the performance on each device the maximum frequency (minimum clock period) was estimated in each case. The following diagram (Figure 9.5) illustrates the maximum frequency achieved when the IDWT core is implemented on Virtex-5, Virtex-6 and Spartan 6 devices. For this set of results, the synthesizer's goal was set to be speed-oriented and the optimization effort was set to "High".

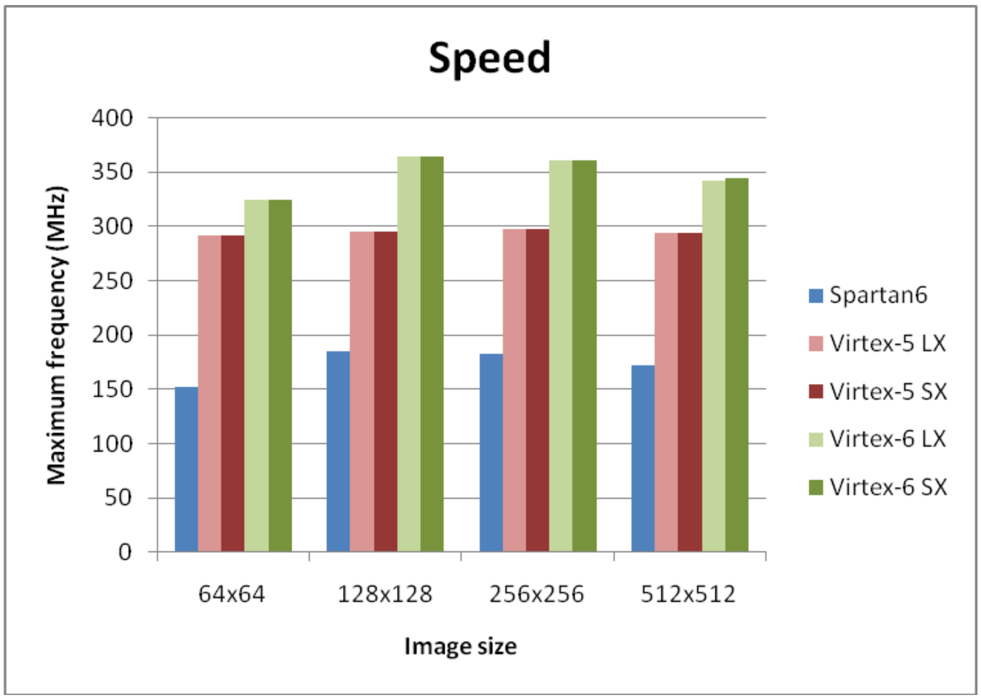


Figure 9.5: Maximum frequency (speed-oriented optimization)

From the above diagram it is clearly observed that Virtex-5 and Virtex-6 technologies achieve much higher clock frequencies in comparison to Spartan 6 technology. From Spartan 6 to Virtex-5 there is a speed improvement of 85% and from Virtex-5 to Virtex-6 a further 20% speed improvement. However, this comes with a cost to power consumption, as it will be presented later in this chapter. Furthermore, it is worth noticing that no significant differences between LX/SX FPGA sub-families were found to be, as far as speed is concerned.

The diagrams that follow (Figure 9.6, Figure 9.7 and Figure 9.8) illustrate the slice utilization results for each of the above FPGA families separately. In these diagrams the impact in maximum frequency is estimated by setting different goals to the synthesizer.

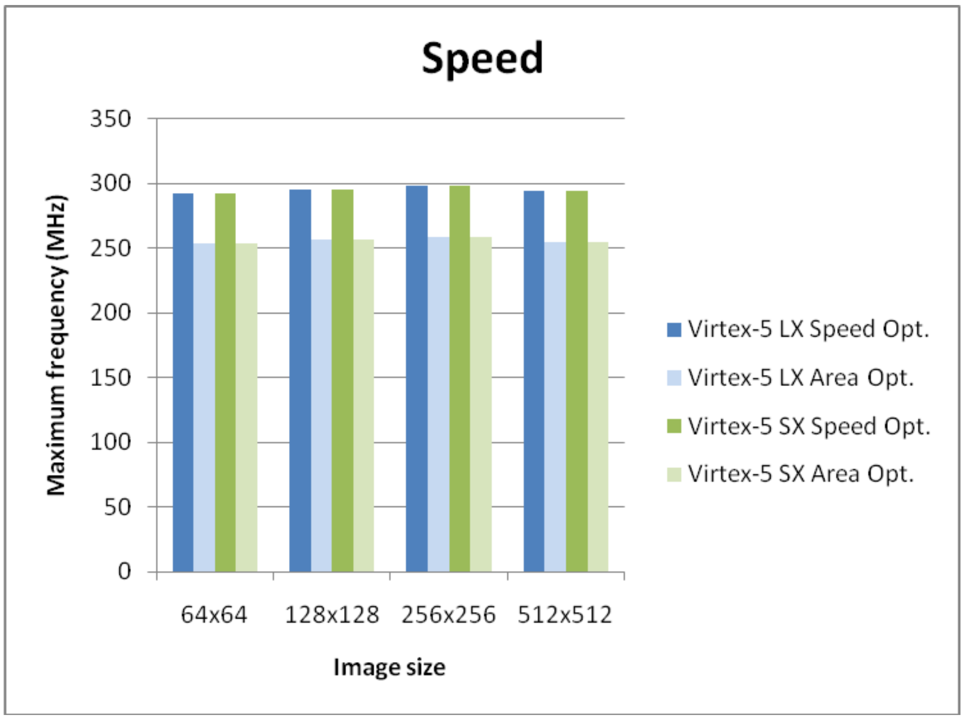


Figure 9.6: Maximum frequency – Virtex-5

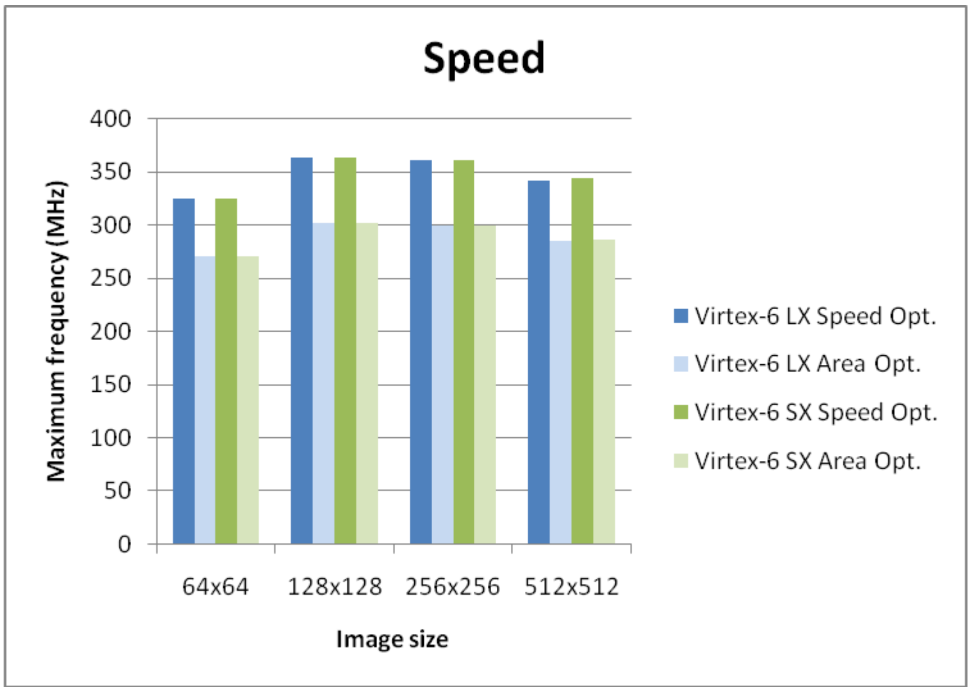


Figure 9.7: Maximum frequency – Virtex-6

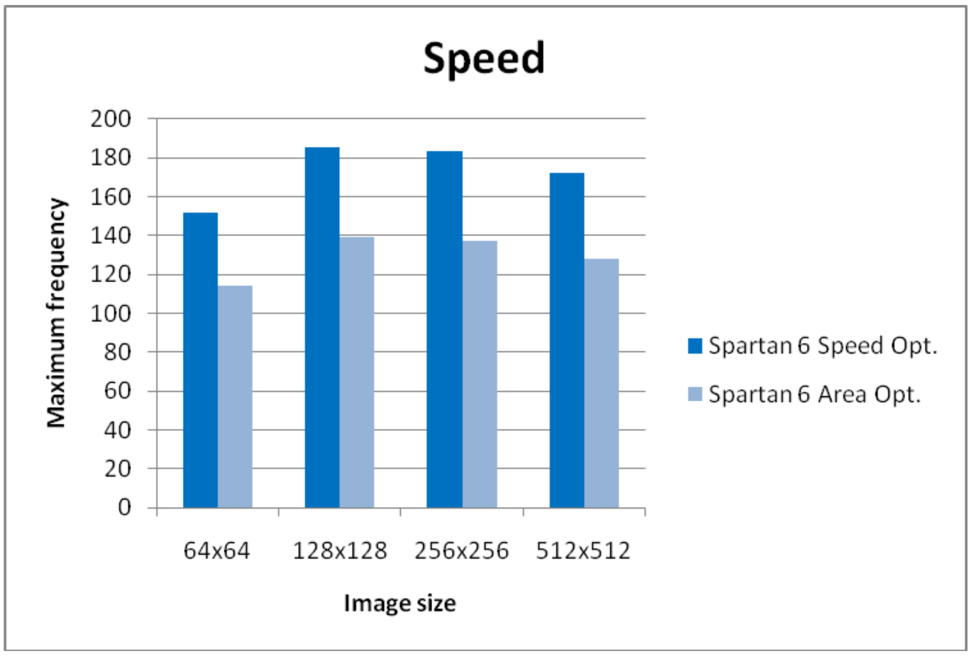


Figure 9.8: Maximum frequency – Spartan 6

The above diagrams show that there is an increase factor in maximum frequency when the algorithm switches from speed to area oriented, especially in Virtex-5 and Spartan 6 devices. More precisely:

Virtex-5: 15% speed increase

Virtex-6: 12% speed increase

Spartan 6: 13% speed increase

9.2.3 Power Consumption Results

The power consumption of the design was evaluated on the same boards by using the Xpower tool. As it was shown in the slice utilization diagrams only a small fractal of the total number of slices available in each device was utilized (around 1%). This means that FPGA leakage (static power) is going to overshadow the design's power consumption. Consequently, no noticeable changes in total power consumption were found to be as image size increases, and therefore total power consumption remains almost the same in every case. The following diagram (Figure 9.8) presents the power consumption estimates that were extracted using Xpower. The IDWT core is set to work on 256x256 size images and the clock frequency in every device is constrained to 180MHz. For the synthesizer's goals, optimization is set to be area-oriented, power reduction option is selected, and optimization effort is set to "High".

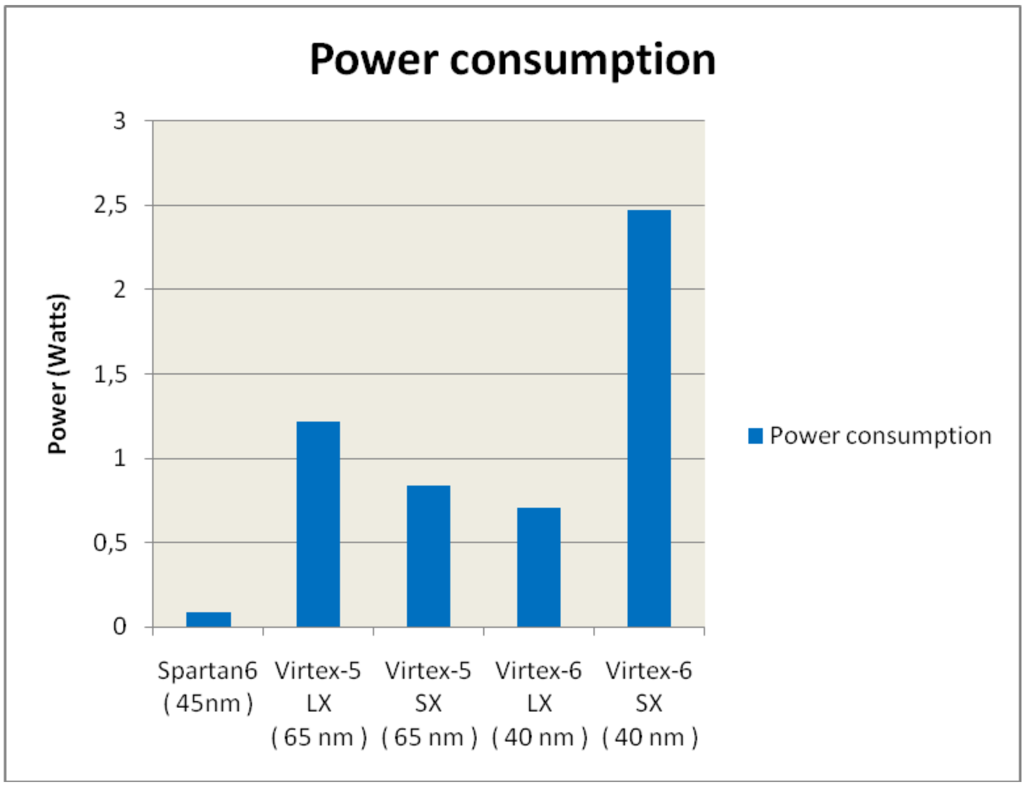


Figure 9.9: Power consumption

After evaluating the results and observing the above diagram, some clear conclusions come out.

First, Spartan 6 is obviously the low-power, low-cost solution for implementing the design, presenting a power consumption that is almost 10% of the one presented by

Virtex-5 SX (the board that was used for implementation). The differences in 45nm logic process (Spartan 6) and 65nm process (Virtex-5), being the reason for this significant decrease in power consumption.

Second, Virtex-5 SX consumes less power than Virtex-5 LX. This is important, because the Virtex-5 LX device that was selected (XC5VLX110T) is the "smallest" device from the LX sub-family, on which the design can fit when instantiated for 512x512 image sizes. Furthermore, in the previous speed diagrams it was shown that maximum frequencies achieved are the same for these devices. Therefore, Virtex-5 SX sub-family proves to be a better solution in comparison to Virtex-5 LX sub-family, especially when we wish to exploit large image sizes.

On the other hand, between Virtex-6 LX and Virtex-6 SX, the LX device proves to be a better solution as it combines lower power consumption (75% power reduction) in comparison to SX, while the maximum frequencies achieved are approximately the same in both devices.

Lastly, the benefits of the novel 40nm copper process technology on which Virtex-6 is built, are depicted in the diagram by the difference in power consumption between Virtex-5 and Virtex-6 families. Virtex-6 LX has 120% the capacity of Virtex-5 SX in slice number, but consumes 15% less power. Even the power-hungry Virtex-6 SX FPGA presents a 200% increase in power consumption compared to Virtex-5 SX, while though providing 6 times the area capacity of the Virtex-5 SX FPGA.

For the purposes of estimating power consumption, changing the synthesizer's goals as far as area, speed and power is concerned, did not have any impact on the final estimation. This is because, as stated before, the IDWT core utilizes only 1% of the total available slices in each device; therefore FPGA leakage is literally the only factor affecting the total power consumption estimation.

9.3 System on Chip Results

In this section, the final results regarding SoC slice utilization, performance and power consumption are presented. Apart from the Virtex-5 SX on which the JPEG2000 codec was implemented and tested, two more devices (Spartan 6 XC6SLX100 and Virtex-5 XC5VLX110T) were evaluated.

9.3.1 Slice utilization results

The following diagram (Figure 9.10) shows the number of slices used to implement the SoC on Virtex-5 SX, Virtex-5 LX and Spartan 6 devices. For this set the synthesizer's goal is set to be speed-oriented and optimization effort is set to "High".

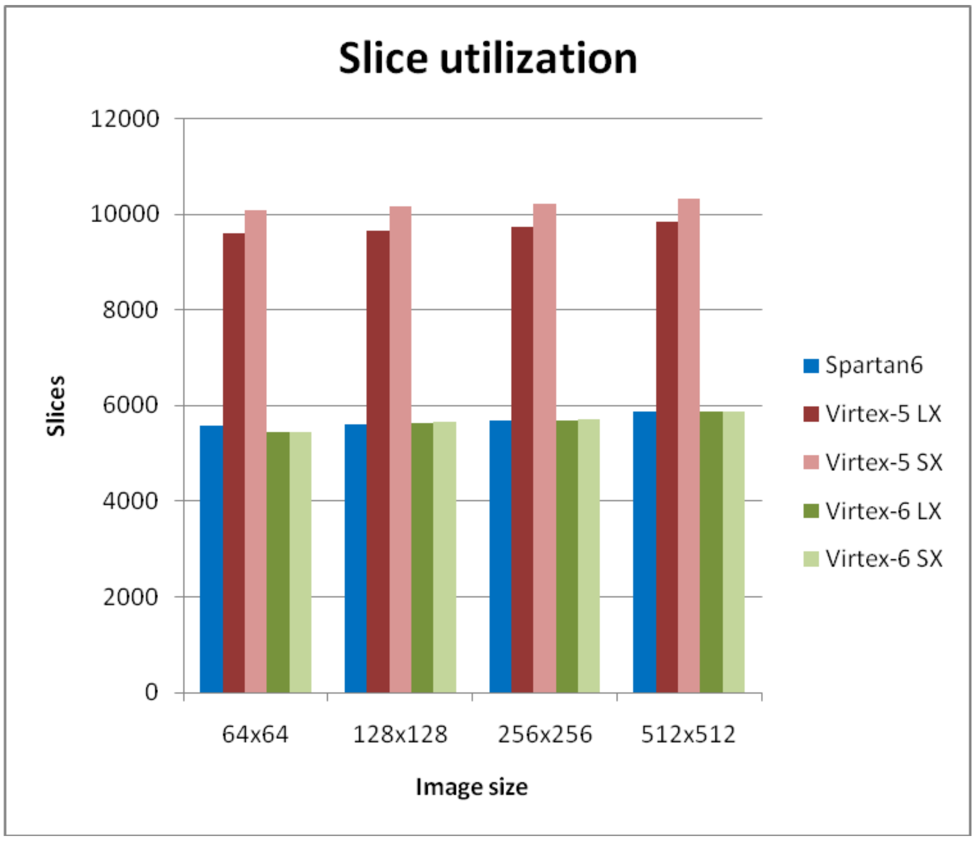


Figure 9.10: Slice utilization (speed-oriented optimization)

The same conclusions are reached here as in the previous section that was referring to the IDWT core. Spartan 6 technology requires fewer slices to implement the SoC, due to the fact that slices in Spartan 6 and Virtex-6 contain double the number of Flip Flops in comparison to Virtex-5 technology.

The diagrams that follow (Figure 9.11, Figure 9.12 Figure 9.13) illustrate the slice utilization results for each of the above FPGA families separately. In these diagrams the impact in maximum frequency is estimated by setting different goals to the synthesizer. The decrease in number of slices is noticeable. More precisely:

Virtex-5 SX: 7% slice utilization decrease

Virtex-5 LX: 3% slice utilization decrease

Spartan 6: 9% slice utilization decrease

Virtex-6: 8% slice utilization decrease

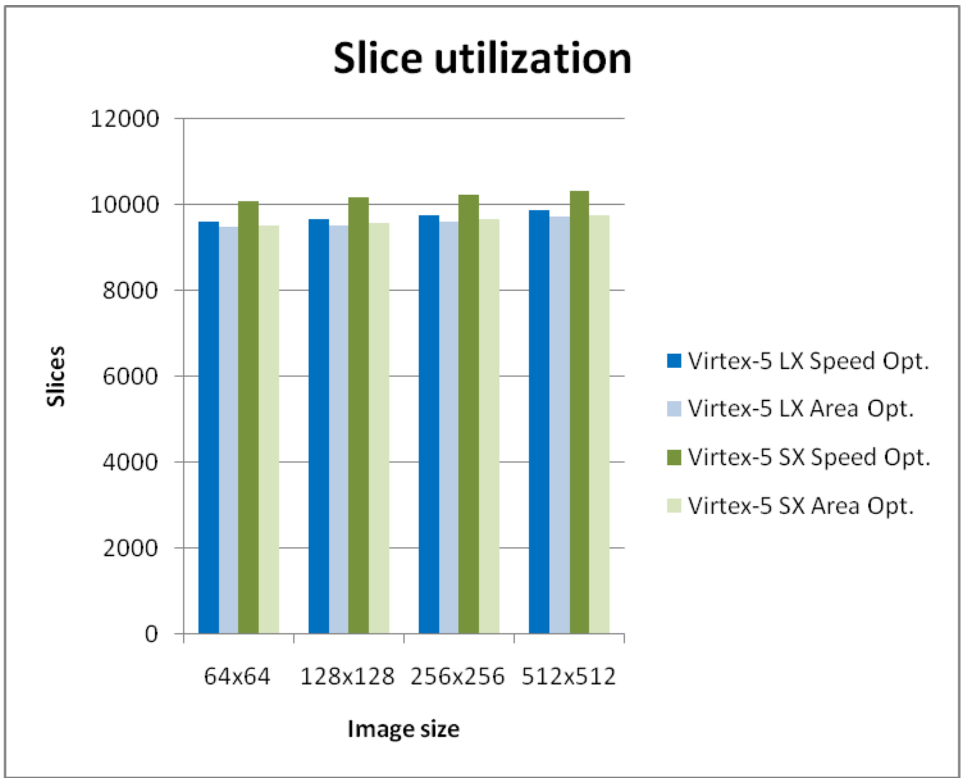


Figure 9.11: Slice utilization – Virtex-5

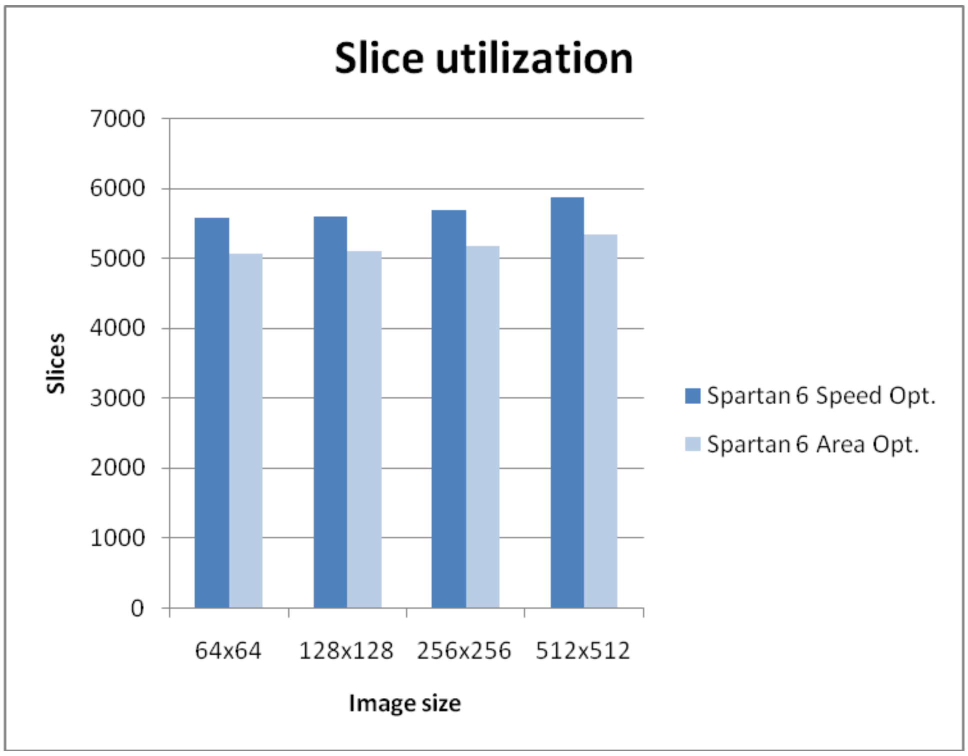


Figure 9.12: Slice utilization – Spartan 6

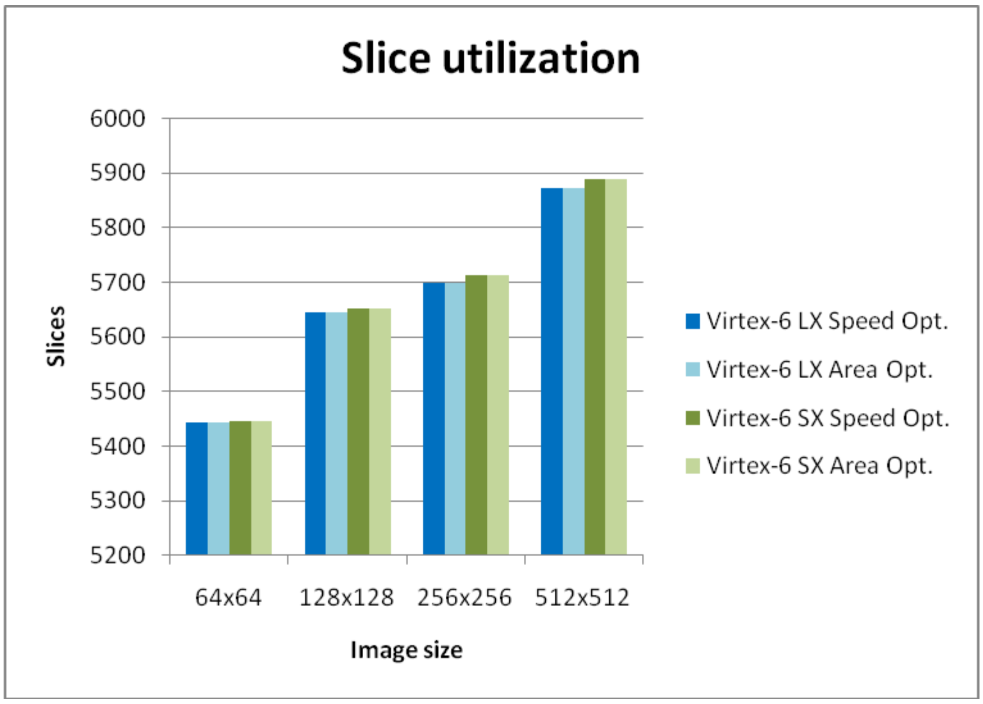


Figure 9.13: Slice utilization – Virtex-6

9.3.2 Performance results

The speed diagram in Figure 9.14 shows the maximum frequency achieved by Virtex-5 SX/LX, Virtex-6 LX/SX and Spartan 6 boards. The synthesizer's goal switches between speed-oriented and area-oriented and the optimization effort is set to "High".

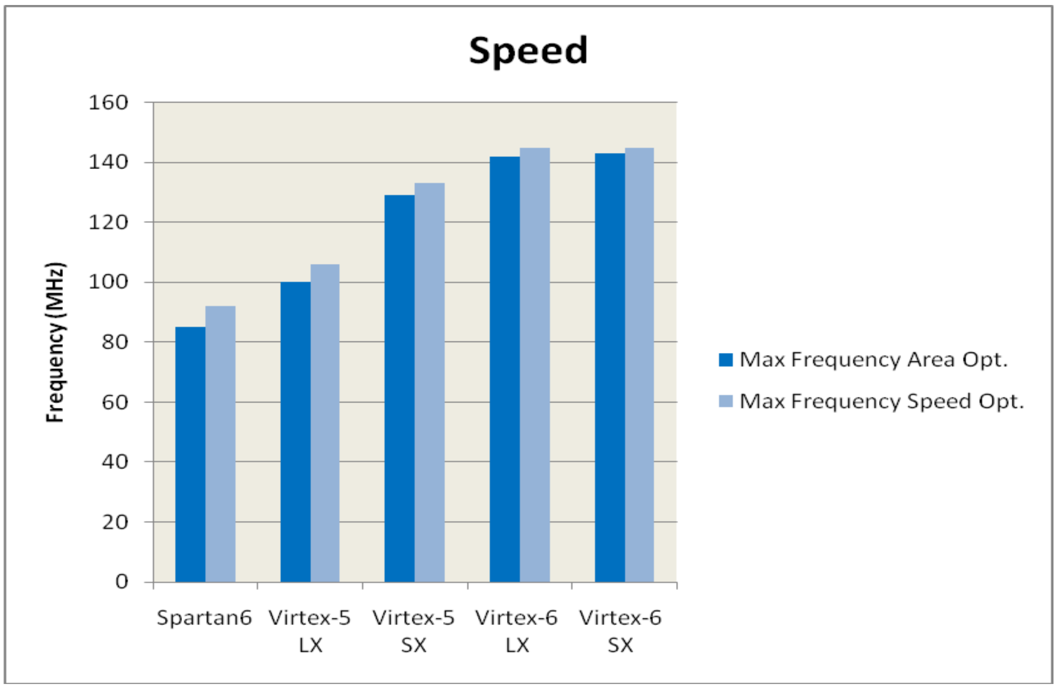
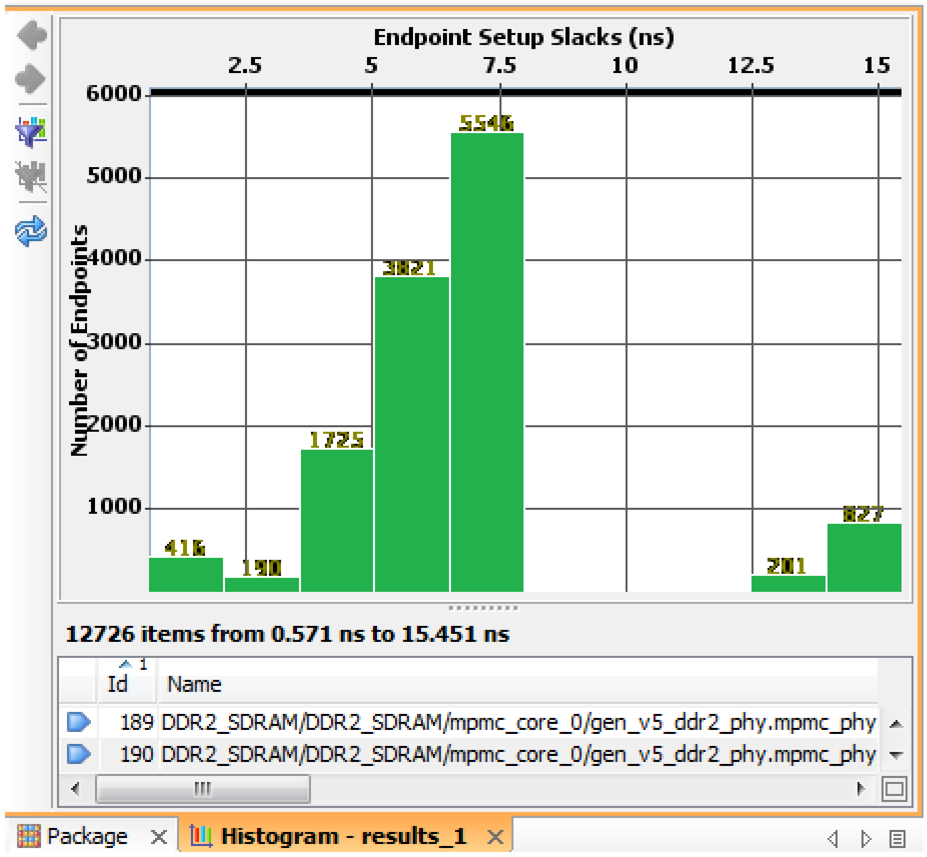


Figure 9.14: Maximum frequency (speed–oriented optimization)

As it was expected Virtex-6 SX achieves the highest speed, while Spartan 6 achieves the lowest one. What is more interesting though is that the SoC speed is almost half the speed achieved when we only implement the IDWT core on the same boards. The existence of a DDR2 SDRAM memory, which is essential in order to download the software application, is the defining factor. The speed bottleneck is the DDR2 SDRAM Multi-Port Multi-Channel Controller (MPMC) as it can be easily seen in the slack histogram below (Figure 9.15), which was generated using the PlanAhead 12.1 tool. The endpoint setup slack for the IDWT core is around 5 ns in every case. Therefore, the IDWT core is not contributing to the decrease in speed when it is embedded into the SoC.



12726 items from 0.571 ns to 15.451 ns

Id	Name
189	DDR2_SDRAM/DDR2_SDRAM/mpmc_core_0/gen_v5_ddr2_phy.mpmc_phy
190	DDR2_SDRAM/DDR2_SDRAM/mpmc_core_0/gen_v5_ddr2_phy.mpmc_phy

Figure 9.15: Slack histogram

For the same reason that was explained above, no significant speed up could be achieved when changing the synthesizer's goal to be speed oriented.

9.3.3 Power consumption results

The power consumption of the final SoC was again estimated by using the Xpower tool. The following diagram (Figure 9.16) shows the differences is power consumption among Spartan 6, Virtex-5 and Virtex-6 development boards. As previously discussed, the existence of the IDWT core does not affect the final estimation.

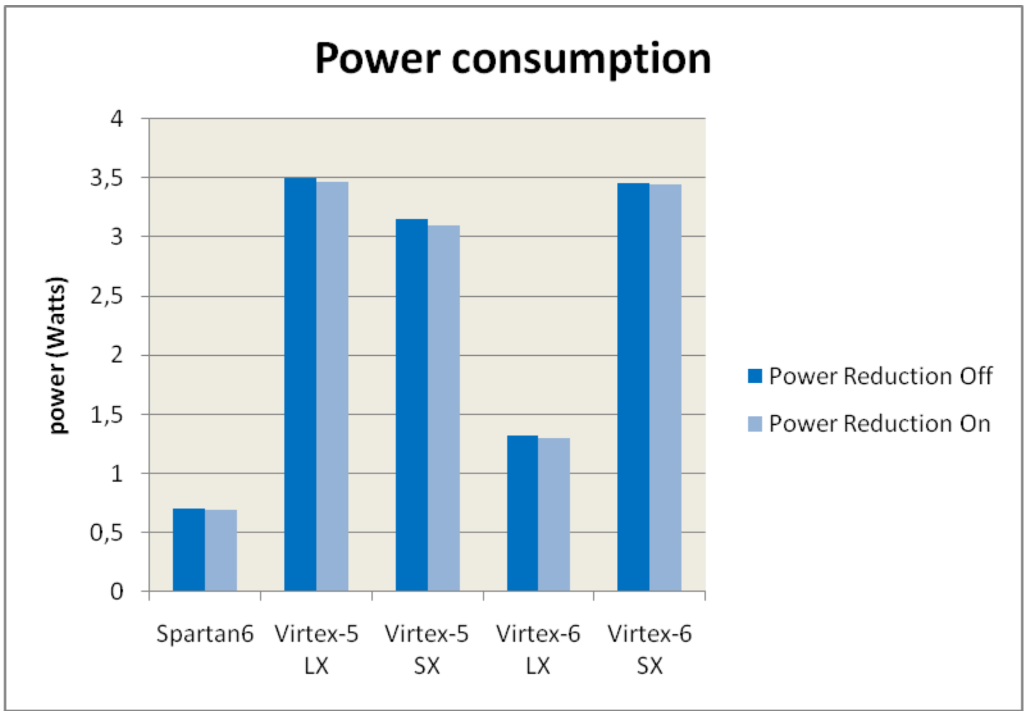


Figure 9.16: SoC power consumption

Again Spartan 6 device proves to be, as expected due to 45nm process technology, the low power, low cost solution for the implementation of the SoC. The power consumption estimate in the Spartan 6 device is 22% of the power consumption estimate for the Virtex-5 SX device and 20% of the power consumption estimate for Virtex-5 LX device. The same conclusions that applied to the IDWT core apply here as well. Virtex-5 SX series proves to be a better solution to Virtex-5 LX series, as far as performance and power trade-offs are concerned. This is again, because larger devices of the LX series need to be used in order to fit the design when it is set to work on larger images. Last, Virtex-6 LX FPGA consumes 63% less power compared to Virtex-5 LX and 58% less power compared to Virtex-5 SX. The Virtex-6 SX device has approximately the same power consumption estimate compared to Virtex-5 families, while being capable to fit a much larger design. Apart from the 45nm and 40nm copper process technologies that have great impact on power in Spartan 6 and Virtex-6 FPGAs, the SoC that is implemented on these devices consumes less energy as it also incorporates a DDR3_SDRAM instead of the DDR2_SDRAM that was used in Virtex-5 SX, which consumes 30% more power.

9.5 JPEG2000 Speed-up

In order to estimate the potential speed-up of the JPEG2000 Decoder after porting the Inverse Discrete Wavelet Transform to hardware, we performed several executions of the application, for different image sizes, with and without the partitioning that was implemented.

First, the JasPer software was downloaded on the development board and run on Microblaze at 100MHz. Using Xilinx Microprocessor Debugger we managed to acquire cycle results both for the whole execution of the application as well as the execution of the Inverse Discrete Wavelet functions alone. After mapping these functions on the FPGA fabric, we performed respective executions for the partitioned application and acquired relevant cycle results. The diagram below (Figure 9.17) shows the cycle results that were acquired, for different image sizes (8-bit grayscale) and Microblaze running at 100MHz (same frequency means that execution time results are straightly analogous to cycle results).

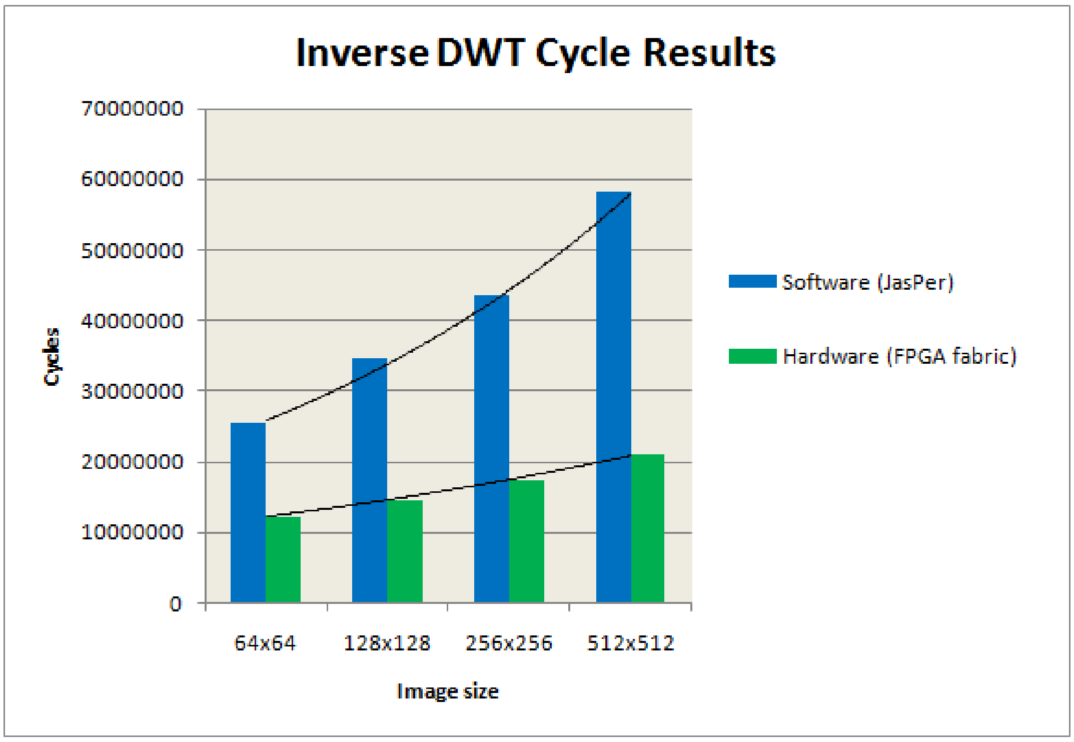


Figure 9.17: Cycle results for IDWT

The above diagram shows an increasing gain in speed as the image size increases. For 64x64 image sizes the hardware implementation of the IDWT performs its computation in 48% of the processing time taken by the JasPer IDWT stage. In other words, we gain a 52% speed-up in execution time. This gain increases up to 63% for 512x512 image sizes.

The impact of this improvement in execution time was also calculated for the whole decoding stage. The following diagram (Figure 9.18) shows the cycle results for the execution of the JPEG2000 decoding stage, as a software-only and as a software/hardware implementation.

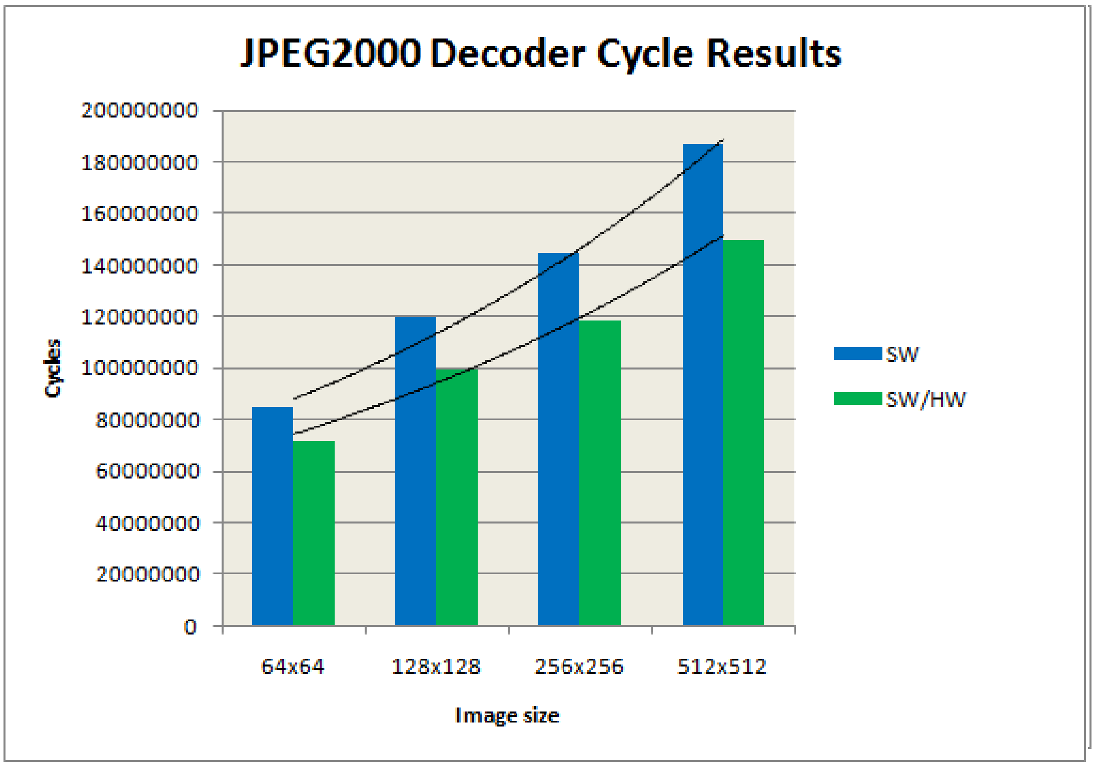


Figure 9.17: Cycle results for JPEG2000 Decoder

It is clear by this diagram that the JPEG2000 Decoder stage benefited from the decision to port the IDWT stage to hardware. We calculated a decrease in cycles, and thus a decrease in execution time, that ranges between 16% and 20%, slightly increasing as the image size increases. As the image size increases, the software implemented IDWT stage performs even more memory accesses, thus becoming even more computationally intensive. Therefore, the implementation of the IDWT on FPGA fabric has even greater impact when we decompress large images.

Chapter 10

Conclusions & Future Work

10.1 Conclusions

In this thesis we presented the co-design and implementation of the JPEG2000 still image compression standard on a Xilinx Virtex-5 development platform. The whole procedure, from the early stages of specification analysis and hardware/software partitioning to the latest stages of implementation and verification, provided a first understanding of the capabilities that are given to the designer by modern CAD design tools, but also the many challenges that are yet to be taken.

Modern CAD design tools, like the Xilinx ISE design suite and the Xilinx EDK tool set that were used for the implementation stage in this thesis, offer great flexibility and automation in hardware/software co-design and co-verification. This brings rapid prototyping to the next level and allows for short time-to-market deadlines to be achieved with greater efficiency. However, big challenges for the designer still exist. Decisions that have to be taken early in the design stage, such as hardware/software partitioning, have eventually great impact on the final result. Therefore, design strategies that are espoused before implementation will almost surely require for the implementation stage to be more flexible. This issue is something that is covered by modern tools generally. However, the stages of co-debug and co-verification of HW/SW systems, as it was also the case in this project, prove to be the most time consuming ones. This especially affects Embedded Systems as the integration of dedicated software running on dedicated hardware bridges these two domains, presenting new challenges not traditionally found on hardware-only systems. Thus, the design tasks of verification and debug of h/w and s/w systems that are written from two different sets of designs -with possibly incomplete specifications-, become even more challenging.

The co-design and implementation of the JPEG2000 compression standard also showed the benefits of implementing a DSP application on FPGA fabric in terms of speed-up gains. An Inverse Discrete Wavelet Transform core was designed in order to map a time consuming and computationally intensive function of the JPEG2000 compression engine on FPGA fabric. The result was an improvement in speed, up to a factor of 20%. Such gains in speed are often critical for DSP applications, where throughput is essential.

For the purposes of this thesis, both the IDWT core and the final System-on-Chip were evaluated as far as area utilization, power consumption and speed is concerned. It was found that when the IP core is integrated in the larger SoC then its maximum speed potential is almost halved down, due to the stricter implementation constraints of a system incorporating a "soft" processor, other IPs, memories, IOs, etc.

Performance, power consumption and area utilization results were also estimated for other modern platforms provided by Xilinx, apart from the Virtex-5 SX (65nm logic process) subfamily that was the development platform for implementing the JPEG2000 standard. More specifically, estimates were extracted for the Spartan 6 (45nm logic process) and Virtex-6 (40nm logic process) FPGA families. This provided us with some conclusions. First, the Spartan 6 FPGA family offers a low-cost, low-power solution for the System-on-Chip that was developed. The differences between 45nm process and 65nm process, as expected, had a great impact on power consumption, reducing power up to 500%. Second, the Virtex-6 FPGA families with 40nm process seem to be a better solution regarding speed and power trade-offs, in comparison to the Virtex-5 FPGA families, as a 30% improvement in speed and a 60% reduction in power consumption were estimated.

Last but not least, in the second part of the present thesis, a step-by-step guide has been presented that allows one to follow or get familiar with some basic steps and procedures regarding the Xilinx Embedded Development Kit. This guide is a good example of the way this tool-set can aid the designer by greatly automating critical parts of the implementation phase.

10.2 Future Work

Obviously there are more ideas to be improved or investigated in this project. The following have been considered for investigation in future work:

- FPGAs are a powerful and compelling option for high-performance, demanding digital signal processing (DSP) applications, whether as part of a co-processing acceleration system or a dedicated hardware implementation. In the future, other modern DSP applications (such as H.264 for broadcast) could be investigated for co-design and implementation on DSP-oriented FPGA platforms.

- In this thesis, as far as the HW/SW partitioning solution is concerned, it was decided to port the Inverse Discrete Wavelet Transform to hardware. However, this is not the optimal partitioning solution. Design Space Exploration methodologies can be used in order to broaden the search for possible potential design solutions, including the Encoder stage. This means porting Tier-1 Coder, FDWT, ROI and MCT sub-functions to hardware.
- For the implementation of the JPEG2000 standard, we created an IP core that was integrated into a larger SoC. The PLB bus communication protocol was used to establish communication between the Microblaze processor and the IDWT core. In the future, the potential gains of using the FSL bus protocol could be investigated.
- Implementation of the JPEG2000 compression standard on Virtex-7 FPGA families could give interesting results regarding the novel 28nm logic process.
- Investigating the implementation of the Discrete Wavelet Transform in ASIC.

<u>*Part 2*</u>

Chapter 11

Step-by-step Guide

Overview

In this chapter a step-by-step guide is presented that describes the whole procedure of implementing and testing our JPEG2000 co-design on the Xilinx Virtex-5 ML506 FPGA (XC5VSX50T). It is assumed that Xilinx EDK 12.1 and Xilinx ISE 12.1 are properly installed and all the standard libraries are generated as described in Embedded System Tools Reference Manual by Xilinx [15].

HARDWARE DEVELOPMENT

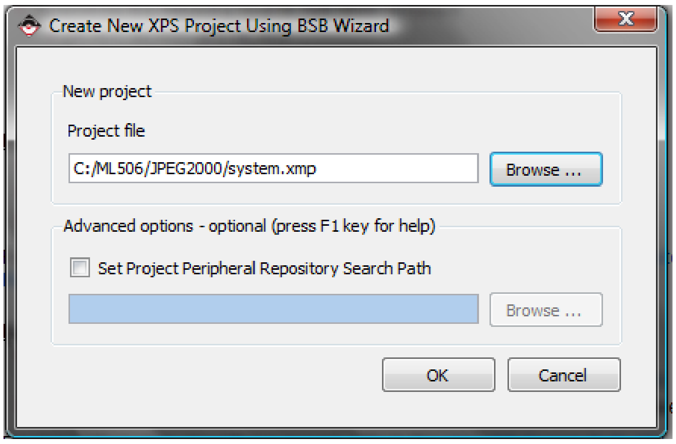
11.1 Creating a new project

In order to create a new project, we will use the **Base System Builder (BSB)** wizard that quickly and efficiently establishes a working design that can then be further customized. Xilinx recommends using the BSB Wizard to create the foundation for any new embedded design project, as it saves a lot of time by automating basic hardware and software platform configuration tasks common to most processor designs.

<u>Steps</u>

1.1 Open XPS. From the dialog box, select "Base System Builder wizard" and OK.

1.2 Click "Browse" and create a new folder for the project. Click "OK".


Create New XPS Project Using BSB Wizard

New project

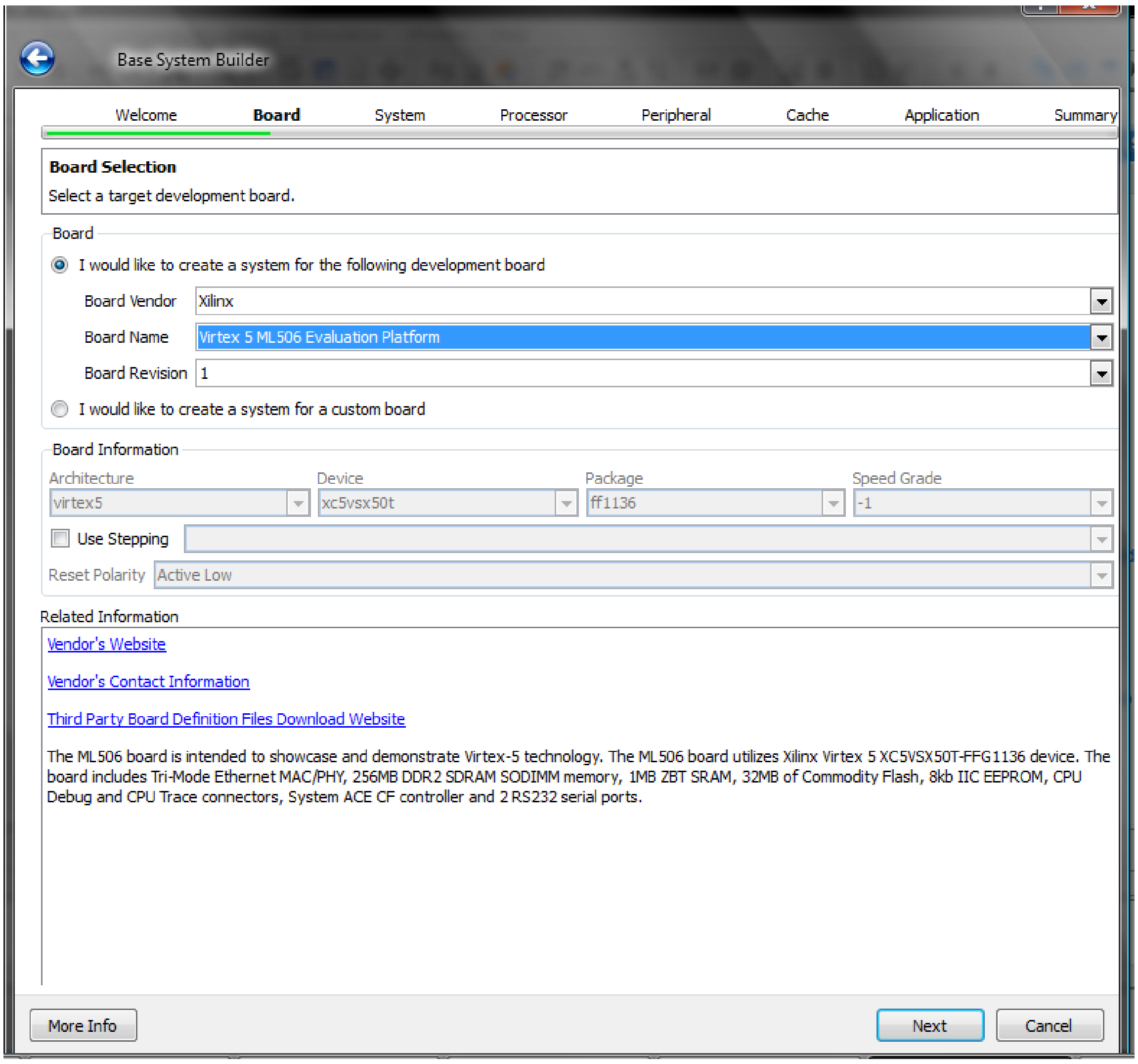
Project file: C:/ML506/JPEG2000/system.xmp

Advanced options - optional (press F1 key for help)

Set Project Peripheral Repository Search Path

1.3 We are given the choice to create a new project or to create one using the template of another project. Tick "I would like to create a new design" and click "Next".

1.4 On the "Board" page, select "Xilinx" as the board vendor. Then select the board "Virtex 5 ML506 Evaluation Platform" board. Select "1" as the board revision. Click "Next".



Board Selection
Select a target development board.

Board
(●) I would like to create a system for the following development board
Board Vendor: Xilinx
Board Name: Virtex 5 ML506 Evaluation Platform
Board Revision: 1
( ) I would like to create a system for a custom board

Board Information

Architecture	Device	Package	Speed Grade
virtex5	xc5vsx50t	ff1136	-1

☐ Use Stepping
Reset Polarity: Active Low

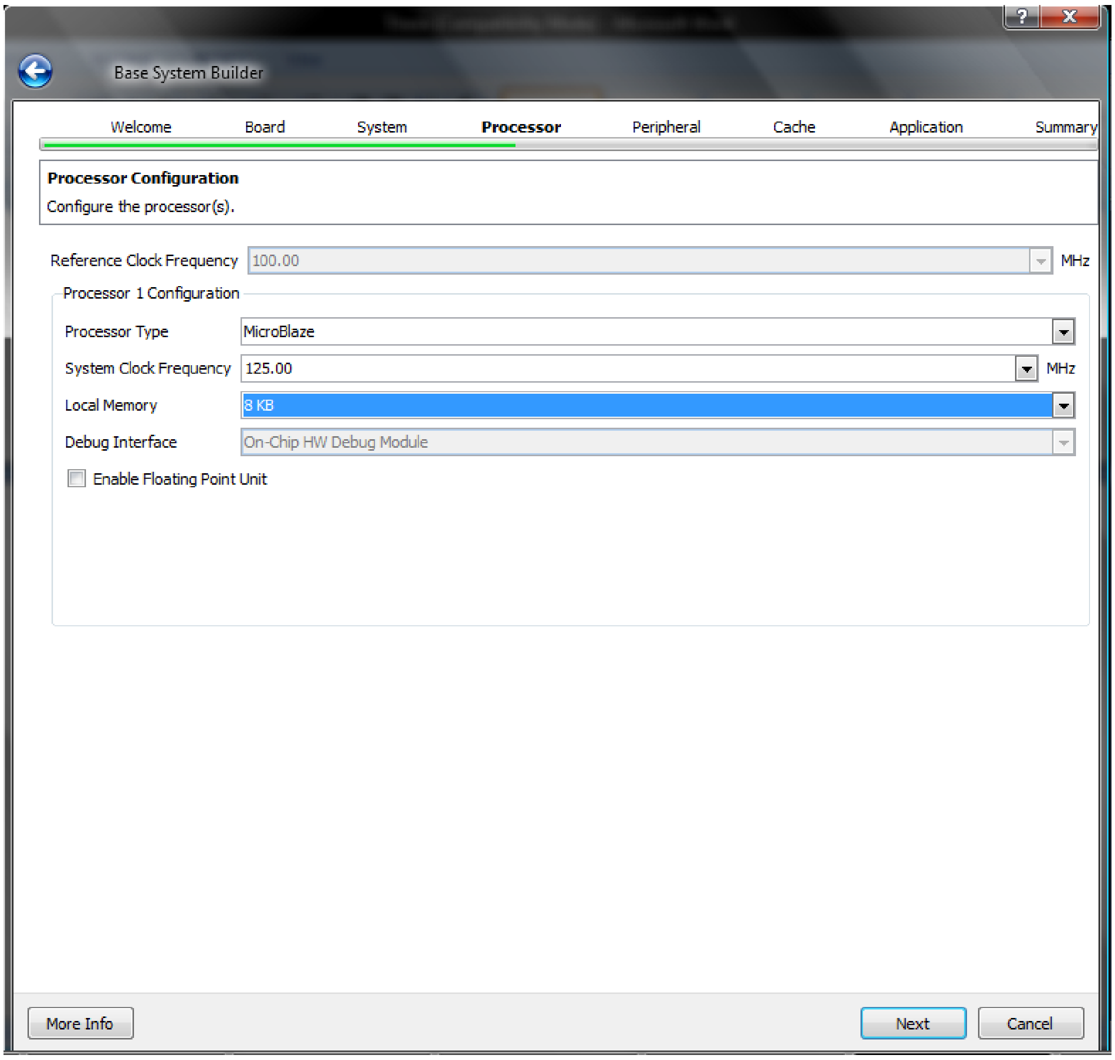
Related Information
Vendor's Website
Vendor's Contact Information
Third Party Board Definition Files Download Website

The ML506 board is intended to showcase and demonstrate Virtex-5 technology. The ML506 board utilizes Xilinx Virtex 5 XC5VSX50T-FFG1136 device. The board includes Tri-Mode Ethernet MAC/PHY, 256MB DDR2 SDRAM SODIMM memory, 1MB ZBT SRAM, 32MB of Commodity Flash, 8kb IIC EEPROM, CPU Debug and CPU Trace connectors, System ACE CF controller and 2 RS232 serial ports.

1.5 On the "System" page, select "Single-Processor System"

1.6 On the "Processor" page, we normally have a choice between using the PowerPC "hard" processor, or the Microblaze "soft" processor. Since the Virtex-5 does not contain any PowerPCs, we can only select Microblaze. Leave "System Clock

Frequency" and "Local Memory" in their default values (125.00 MHz and 8KB respectively). Click "Next".



Processor Configuration

Configure the processor(s).

Reference Clock Frequency: 100.00 MHz

Processor 1 Configuration

Processor Type: MicroBlaze

System Clock Frequency: 125.00 MHz

Local Memory: 8 KB

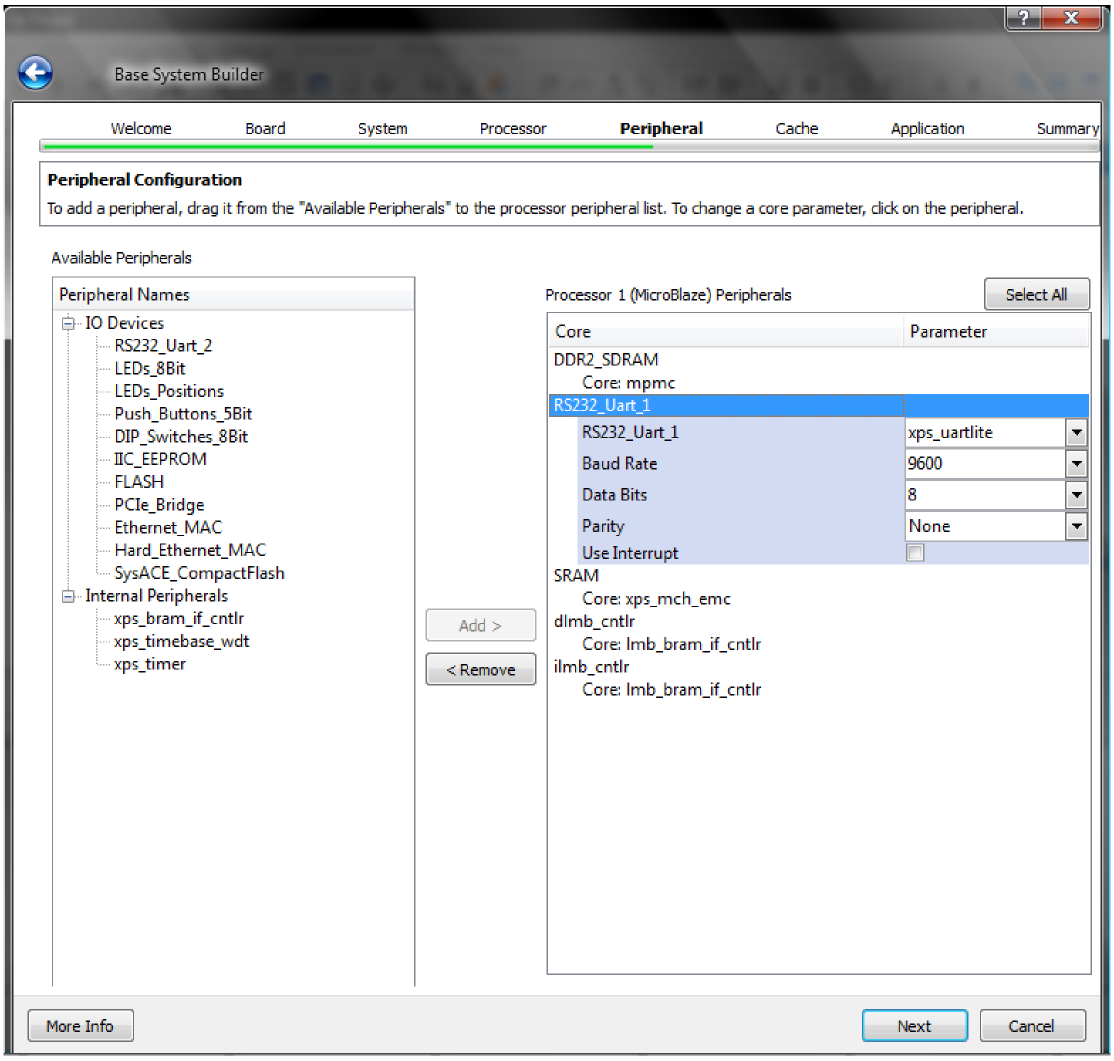
Debug Interface: On-Chip HW Debug Module

☐ Enable Floating Point Unit

1.7 On the "Peripheral" page, use "Add" and "Remove" buttons to add or remove peripherals. Leave RS232_Uart_1, SRAM, DDR2_SDRAM, dlmb_cntlr and ilmb_cntlr in the "Peripherals" list. Click "Next".

1.8 On "Cache" and "Application" pages, click "Next".

1.9 On "Summary" page, click "Finish" and the basic working design is established.



Peripheral Configuration

To add a peripheral, drag it from the "Available Peripherals" to the processor peripheral list. To change a core parameter, click on the peripheral.

Available Peripherals

Peripheral Names
- IO Devices
  - RS232_Uart_2
  - LEDs_8Bit
  - LEDs_Positions
  - Push_Buttons_5Bit
  - DIP_Switches_8Bit
  - IIC_EEPROM
  - FLASH
  - PCIe_Bridge
  - Ethernet_MAC
  - Hard_Ethernet_MAC
  - SysACE_CompactFlash
- Internal Peripherals
  - xps_bram_if_cntlr
  - xps_timebase_wdt
  - xps_timer

Processor 1 (MicroBlaze) Peripherals

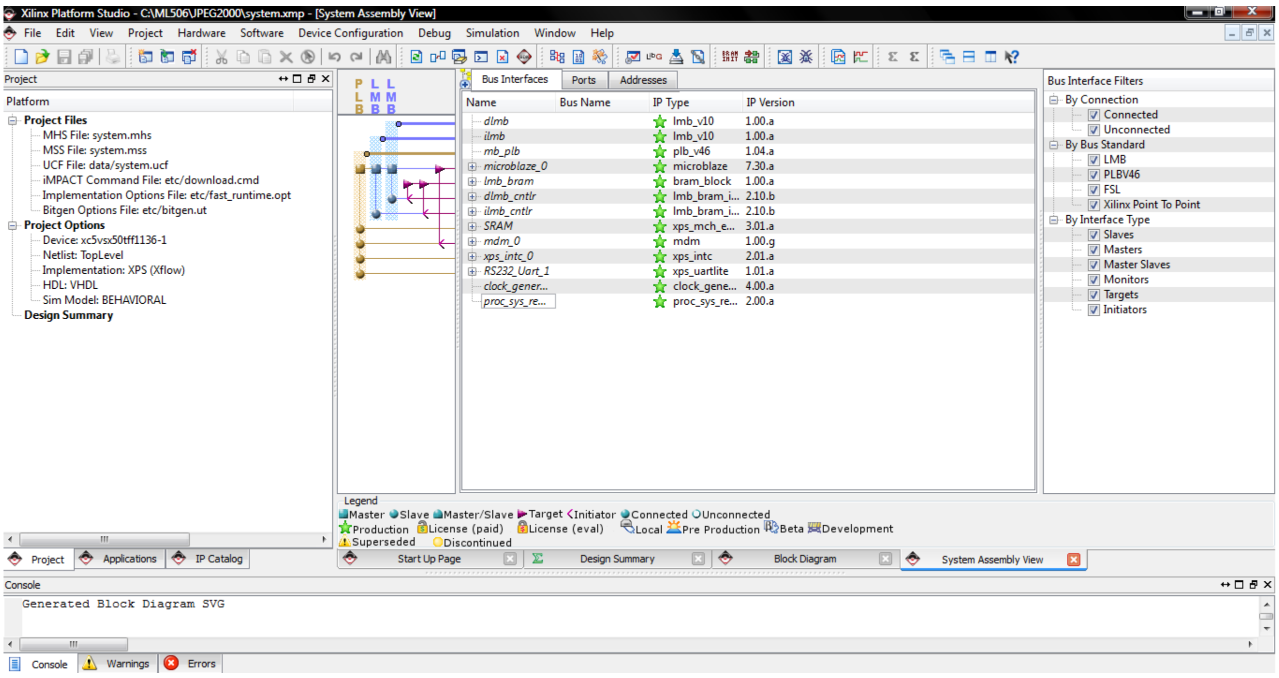
Core	Parameter
DDR2_SDRAM	
Core: mpmc	
RS232_Uart_1	
RS232_Uart_1	xps_uartlite
Baud Rate	9600
Data Bits	8
Parity	None
Use Interrupt	☐
SRAM	
Core: xps_mch_emc	
dlmb_cntlr	
Core: lmb_bram_if_cntlr	
ilmb_cntlr	
Core: lmb_bram_if_cntlr	

11.2 The XPS GUI

Now that the basic configuration has been completed and the basis for our design has been established using BSB, it is time to explain some of the most essential tasks that can be accomplished through the XPS GUI and take a closer look on what information can be straightly provided to the designer by using this GUI.

The XPS main window is divided into three different areas (Figure 11.1):

- Project Information Area
- System Assembly View
- Console Window


Xilinx Platform Studio - C:\ML506\JPEG2000\system.xmp - [System Assembly View]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project
Platform
Bus Interfaces
Ports
Addresses
Bus Interface Filters
Legend
Master Slave Master/Slave Target Initiator Connected Unconnected
Production License (paid) License (eval) Local Pre Production Beta Development
Superseded Discontinued
Project
Applications
IP Catalog
Start Up Page
Design Summary
Block Diagram
System Assembly View
Console
Warnings
Errors

- Project Files
  - MHS File: system.mhs
  - MSS File: system.mss
  - UCF File: data/system.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.opt
  - Bitgen Options File: etc/bitgen.ut
- Project Options
  - Device: xc5vsx50tff1136-1
  - Netlist: TopLevel
  - Implementation: XPS (Xflow)
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Design Summary

Name	Bus Name	IP Type	IP Version
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.04.a
microblaze_0		microblaze	7.30.a
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
SRAM		xps_mch_e...	3.01.a
mdm_0		mdm	1.00.g
xps_intc_0		xps_intc	2.01.a
RS232_Uart_1		xps_uartlite	1.01.a
clock_gener...		clock_gene...	4.00.a
proc_sys_re...		proc_sys_re...	2.00.a

- By Connection
  - ✓ Connected
  - ✓ Unconnected
- By Bus Standard
  - ✓ LMB
  - ✓ PLBV46
  - ✓ FSL
  - ✓ Xilinx Point To Point
- By Interface Type
  - ✓ Slaves
  - ✓ Masters
  - ✓ Master Slaves
  - ✓ Monitors
  - ✓ Targets
  - ✓ Initiators

```
Generated Block Diagram SVG
```

Figure 11.1: XPS GUI

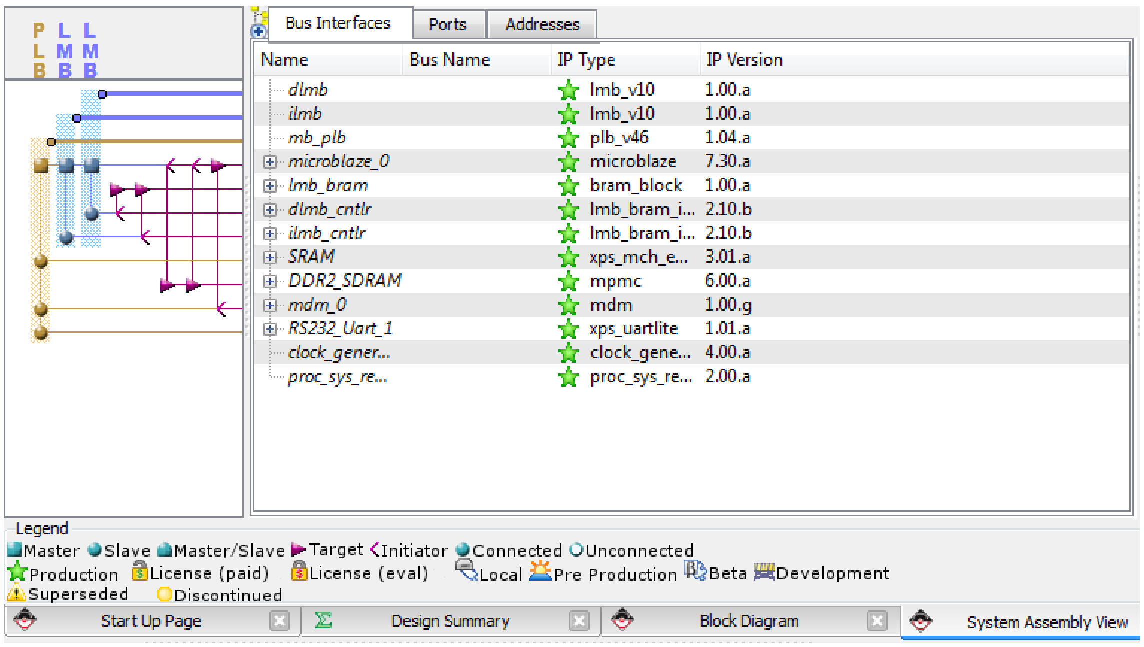
Project Information Area

The Project Information Area offers control over and information about the project. It is divided into three tabs:

- *Project Tab:* lists all project related files such as the MHS, MSS, User Constraints File (UCF), iMPACT command files, Device, HDL and Netlist options, log files, etc.
- *Applications Tab:* lists all software application option settings, header files, and source files that are associated with each application project.
- *IP Catalog Tab:* lists all the EDK IP cores and any custom IP cores.

System Assembly View

The System Assembly View allows the user to view and configure system block elements. XPS provides Bus Interface, Ports, and Addresses tabs in the System Assembly View (Figure 11.2), to organize information about the design and allow the designer to more easily edit the hardware platform. The Connectivity Panel that accompanies the Bus Interface tab is a graphical representation of the hardware platform interconnects.



P L B
L M B
L M B
Bus Interfaces
Ports
Addresses
Legend
Master Slave Master/Slave Target Initiator Connected Unconnected
Production License (paid) License (eval) Local Pre Production Beta Development
Superseded Discontinued
Start Up Page
Design Summary
Block Diagram
System Assembly View

Name	Bus Name	IP Type	IP Version
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.04.a
microblaze_0		microblaze	7.30.a
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
SRAM		xps_mch_e...	3.01.a
DDR2_SDRAM		mpmc	6.00.a
mdm_0		mdm	1.00.g
RS232_Uart_1		xps_uartlite	1.01.a
clock_gener...		clock_gene...	4.00.a
proc_sys_re...		proc_sys_re...	2.00.a

Figure 11.2: System Assembly View – Bus Interface tab and Connectivity Panel

- A vertical line represents a bus, and a horizontal line represents a bus interface to an IP core.
- If a compatible connection can be made, a connector is displayed at the intersection between the bus and IP core bus interface.
- The lines and connectors are color-coded to show bus compatibility.
- Differently shaped connection symbols indicate whether IP blocks are bus masters or bus slaves.
- A hollow connector represents a connection that you can make, and a filled connector represents a connection made. To create or disable a connection, the user can simply click the connector symbol.

11.3 Creating and Importing the Peripheral

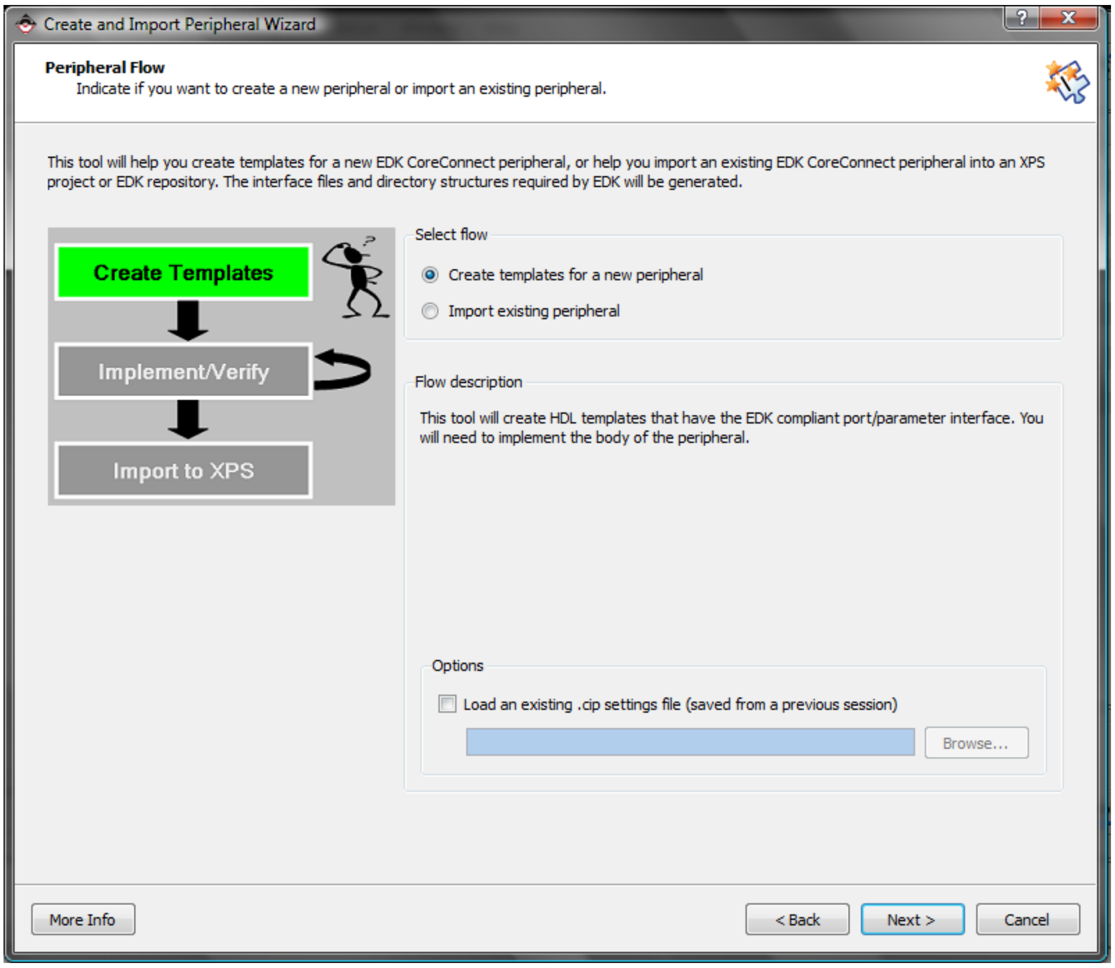
The next step after having established a basic working design is to import the IDWT VHDL design in order to create a new Intellectual Property (IP) core and import it as a slave peripheral in the embedded system. EDK offers the Create and Import Peripheral (CIP) Wizard, which simplifies the procedure by automating many critical steps, like the creation of a slave interface for the IP, proper updating of the MHS and MPD file, etc.

In order to follow the steps bellow it is assumed that the .vhd source files for the IDWT core are available to the user.

<u>Create the IDWT Peripheral-Steps</u>

3.1 Select from the menu "Hardware->Create or Import Peripheral". Click "Next".

3.2 Select "Create templates for a new peripheral" and click "Next".



Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK CoreConnect peripheral, or help you import an existing EDK CoreConnect peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow

[✓ Create templates for a new peripheral]
Import existing peripheral

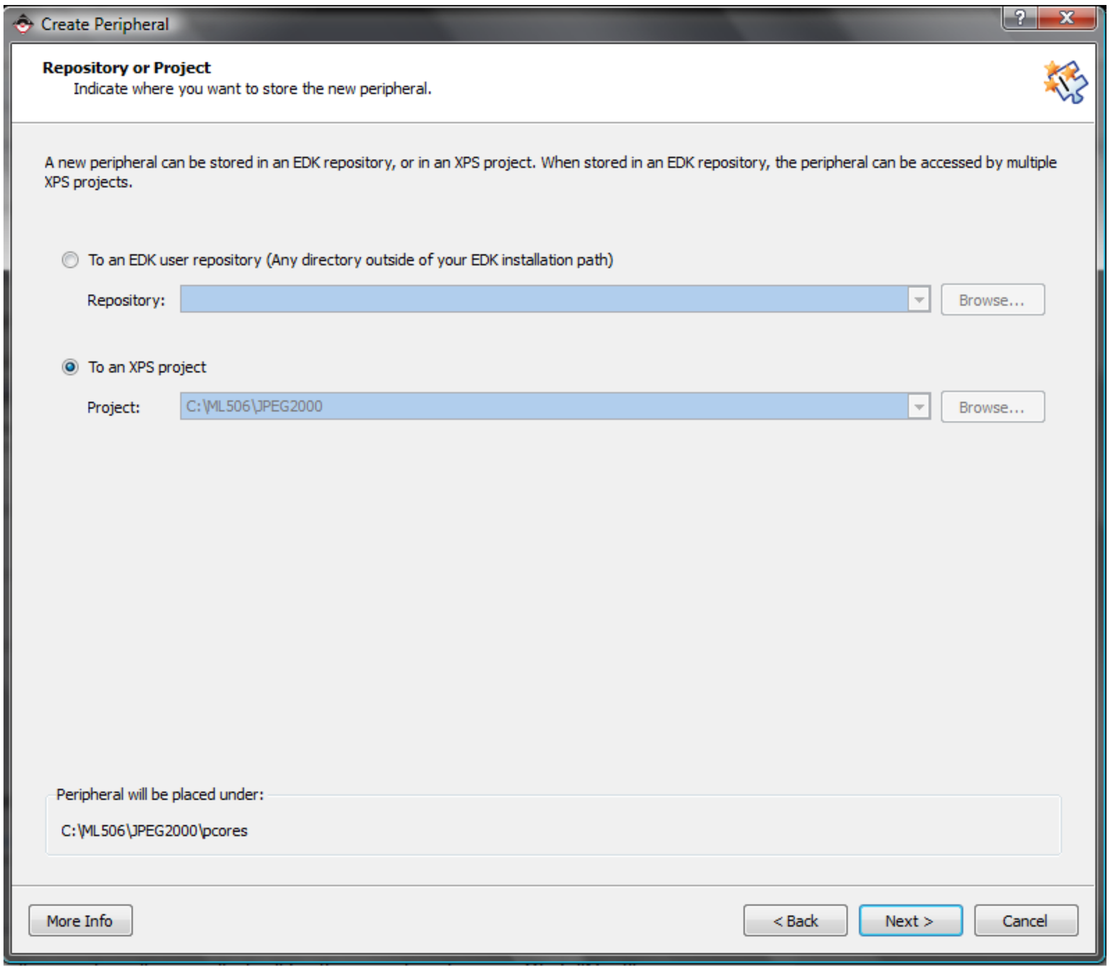
Flow description

This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

Options

☐ Load an existing .cip settings file (saved from a previous session)

3.3 We must now decide where to place the files for the peripheral. They can be placed within this project, or they can be made accessible to other projects. Select "To an XPS project". Click "Next".



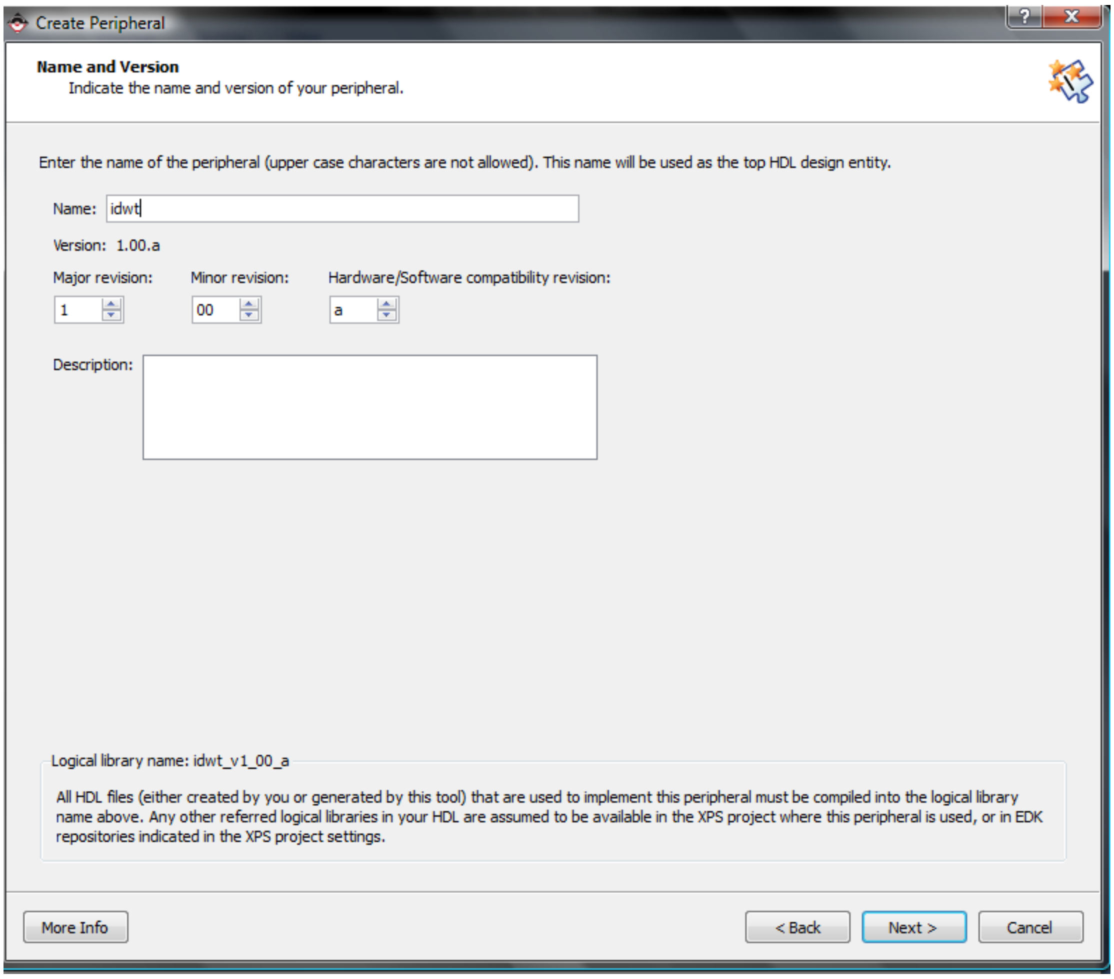
A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

- ○ To an EDK user repository (Any directory outside of your EDK installation path)
  Repository:
- [✓ To an XPS project]
  Project: C:\ML506\JPEG2000

Peripheral will be placed under:
C:\ML506\JPEG2000\pcores

3.4 On the "Name and Version" page, type "idwt" for the peripheral name. Click "Next". Notice the logic logical library that is created: idwt_v1_00_a. All HDL files, both user created and tool generated, must be compiled into this logical library name above.

3.5 On the "Bus Interface" page, select "Processor Local Bus" (PLB) and click "Next".


Create Peripheral

Name and Version

Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name: idwt

Version: 1.00.a

Major revision: 1

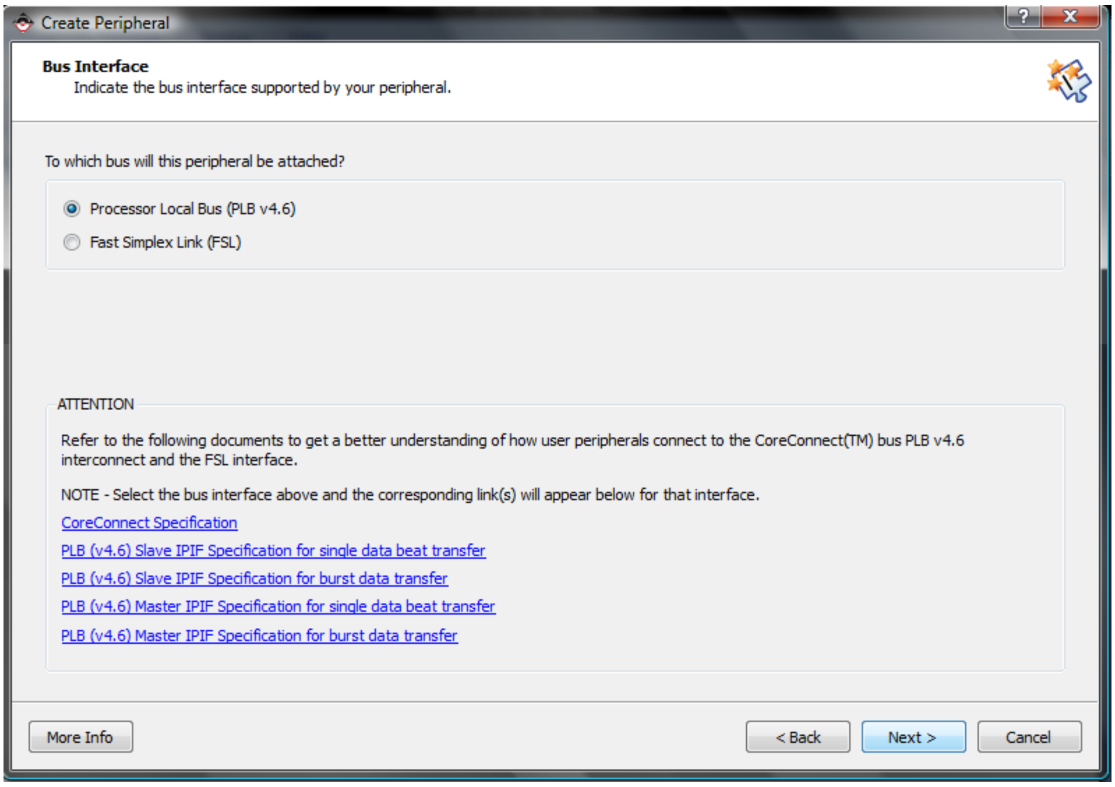
Minor revision: 00

Hardware/Software compatibility revision: a

Description:

Logical library name: idwt_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.


Create Peripheral

Bus Interface

Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

● Processor Local Bus (PLB v4.6)

○ Fast Simplex Link (FSL)

ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) bus PLB v4.6 interconnect and the FSL interface.

NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.

CoreConnect Specification

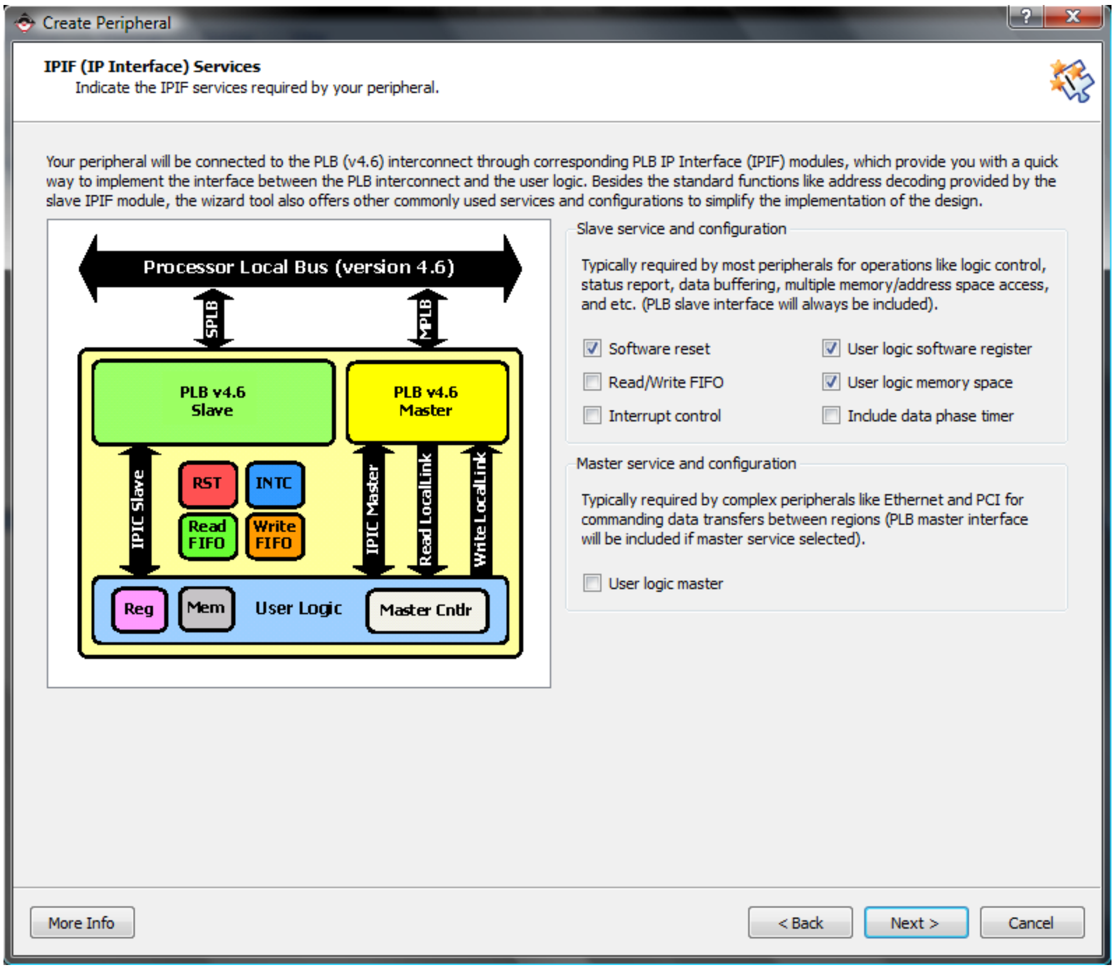
PLB (v4.6) Slave IPIF Specification for single data beat transfer

PLB (v4.6) Slave IPIF Specification for burst data transfer

PLB (v4.6) Master IPIF Specification for single data beat transfer

PLB (v4.6) Master IPIF Specification for burst data transfer

3.6 On the "IPIF Services" page, we can make the Peripheral Wizard generate our VHDL template to include different features. We need a software reset to give us the ability to reset the IDWT peripheral in the software application, software accessible registers for debugging and user memory space to store the data that our peripheral needs. Select "Software Reset", "User logic software register", and "User logic memory space", un-tick everything else and click "Next".


Create Peripheral

IPIF (IP Interface) Services

Indicate the IPIF services required by your peripheral.

Your peripheral will be connected to the PLB (v4.6) interconnect through corresponding PLB IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the PLB interconnect and the user logic. Besides the standard functions like address decoding provided by the slave IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation of the design.

Slave service and configuration

Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (PLB slave interface will always be included).

- [✓] Software reset
- [✓] User logic software register
- [ ] Read/Write FIFO
- [✓] User logic memory space
- [ ] Interrupt control
- [ ] Include data phase timer

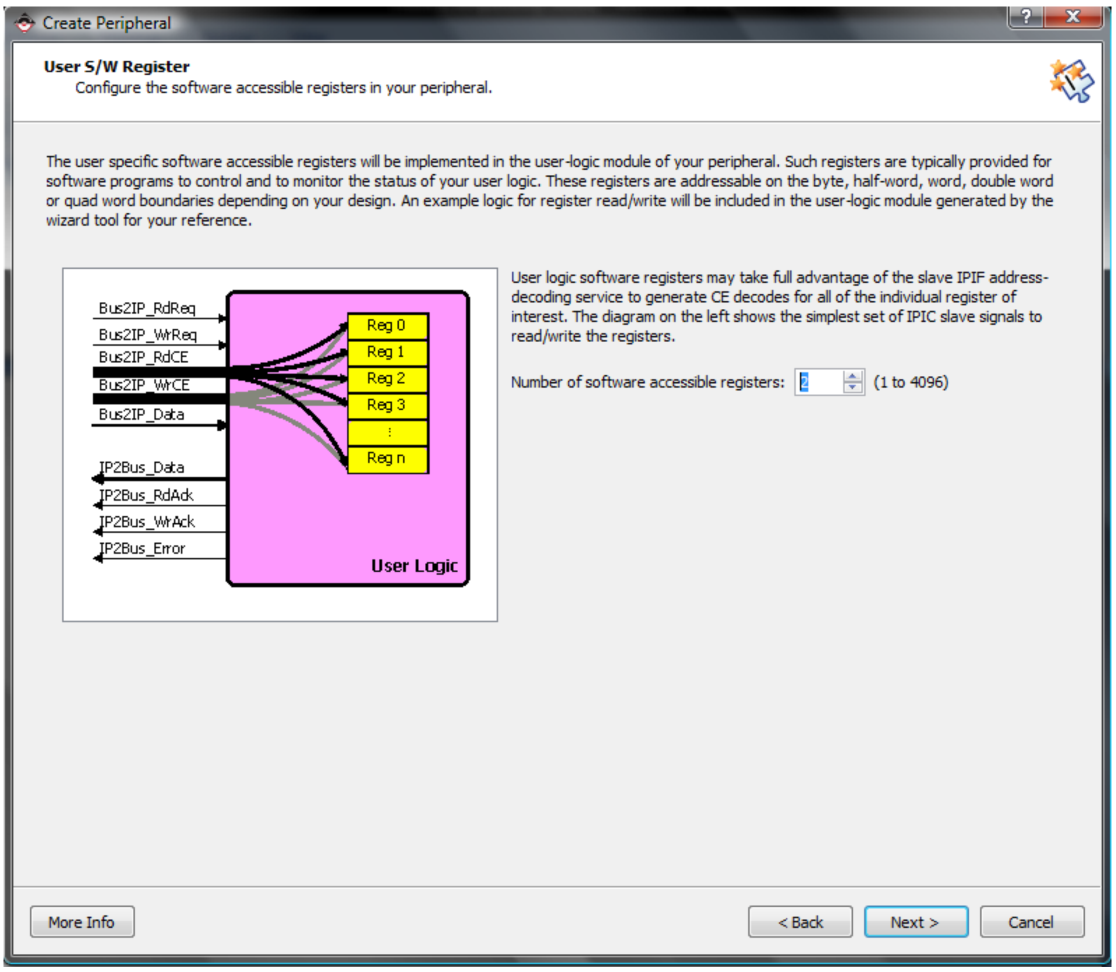
Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected).

- [ ] User logic master

3.7 On the "Slave Interface" page, click "Next".

3.8 On the "User S/W Register" page, select 2 software accessible registers to be instantiated with our design.


Create Peripheral

User S/W Register

Configure the software accessible registers in your peripheral.

The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIC slave signals to read/write the registers.

Number of software accessible registers: 2 (1 to 4096)

3.9 On the “User Memory Space” select 1 “User address range”. Click “Next”.

3.10 On the "IP Interconnect" page we can customize our connection to the PLB but we will leave everything as is for simplicity. Click "Next".

3.11 On the "Peripheral Simulation Support" page, we can specify if we want the wizard to create a simulation platform for our peripheral. Click "Next" without ticking the option to generate.

3.12 After the "Peripheral Implementation Support" page, the wizard will generate all the template files for us. Tick "Generate ISE and XST project files" and "Generate template driver files". Click "Next".

3.13 Click "Finish". Now our templates are created.

<u>Create the IDWT core in VHDL - Steps</u>

3.14 In this step we have to import our VHDL design in XPS. The source code files are assumed to be available already to the user. The files are: "IDWT_pkg.vhd", "IDWT_top.vhd", "Controller1D.vhd", "Controller2D.vhd", "PU.vhd" and "Clock_buf.vhd".

3.15 The files should then be placed in "pcores\idwt_v1_00_a\hdl\vhdl" folder, which has been already created by XPS into our project directory.

<u>Modifying the .PAO file – Steps</u>

The .pao file contains a list of all the source files that compose our peripheral. We use this list when we run the Peripheral Wizard in Import mode. Now that we have added source files to the project, we must include it in the .pao file.

3.16 In XPS select "File->Open" and browse to the "pcores\idwt_v1_00_a\data" folder. Select the file "idwt_v2_1_0.pao" and click "Open".

3.17 At the bottom of this file you will see these two lines:

```
lib idwt_v1_00_a user_logic vhdl

lib idwt_v1_00_a idwt vhdl
```

3.18 Now, in a similar format, we have to import the files that we created before. We have to insert the lines that "point" to our files just above these two lines. After that, the .pao file should look like the picture in the following page. Notice that the .pao file lists the source files in hierarchical order. Thus if we have a VHDL design consisting of multiple files (like the IDWT core), it is important to know the hierarchical order of the components. The components at the <u>top of the chain</u> are listed at the <u>bottom of the file</u>.

3.19 Save the .pao file.

```
##############################################################################
## Filename:          C:/ML506/JPEG2000/pcores/idwt_v1_00_a/data/idwt_v2_1_0.pao
## Description:       Peripheral Analysis Order
## Date:              Mon Mar 14 01:48:44 2011 (by Create and Import Peripheral
##############################################################################

lib proc_common_v3_00_a proc_common_pkg vhdl
lib proc_common_v3_00_a ipif_pkg vhdl
lib proc_common_v3_00_a or_muxcy vhdl
lib proc_common_v3_00_a or_gate128 vhdl
lib proc_common_v3_00_a family_support vhdl
lib proc_common_v3_00_a pselect_f vhdl
lib proc_common_v3_00_a counter_f vhdl
lib plbv46_slave_single_v1_01_a plb_address_decoder vhdl
lib plbv46_slave_single_v1_01_a plb_slave_attachment vhdl
lib plbv46_slave_single_v1_01_a plbv46_slave_single vhdl
lib proc_common_v3_00_a soft_reset vhdl
lib idwt_v1_00_a Clock_buf vhdl
lib idwt_v1_00_a PU vhdl
lib idwt_v1_00_a Controller2D vhdl
lib idwt_v1_00_a Controller1D vhdl
lib idwt_v1_00_a IDWT_top vhdl
lib idwt_v1_00_a IDWT_pkg vhdl
lib idwt_v1_00_a user_logic vhdl
lib idwt_v1_00_a idwt vhdl
```

<u>Modifying the Peripheral – Steps</u>

Now we will add code in our peripheral template to instantiate an IDWT core and we will connect it to the system bus (PLB).

3.20 Select from the menu "File->Open" and look in the project folder.

3.21 Open the folders: "pcores\idwt_v1_00_a\hdl\vhdl". This folder contains two source files that describe our peripheral "my_multiplier.vhd" and "user_logic.vhd". The first file is the main part of the peripheral and it implements the interface to the PLB. The second file is where we place our custom logic to make the peripheral do what we need it to do. This part is instantiated by the first file.

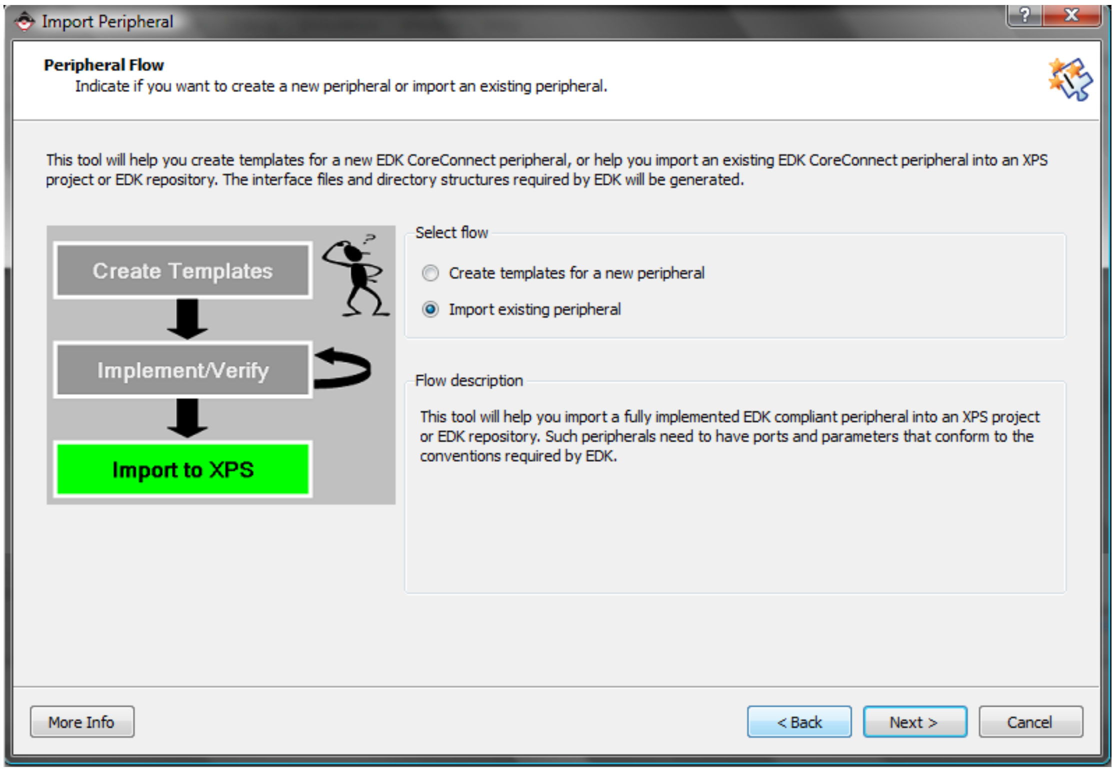
3.22 Open the file "user_logic.vhd". We will need to modify this source code to instantiate the IDWT and connect it to the user address memory and PLB slave bus protocol signals. It is supposed that the "user_logic.vhd" is already available to the user. Paste the contents over the original code in the file we opened. Then save the file.

<u>Importing the Peripheral – Steps</u>

Now we will use the Peripheral Wizard in Import mode.

3.23 Select from the menu "Hardware->Create or Import Peripheral" and click "Next".

3.24 Select "Import existing peripheral" and click "Next".


Import Peripheral

Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK CoreConnect peripheral, or help you import an existing EDK CoreConnect peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow
Create templates for a new peripheral
[✓ Import existing peripheral]

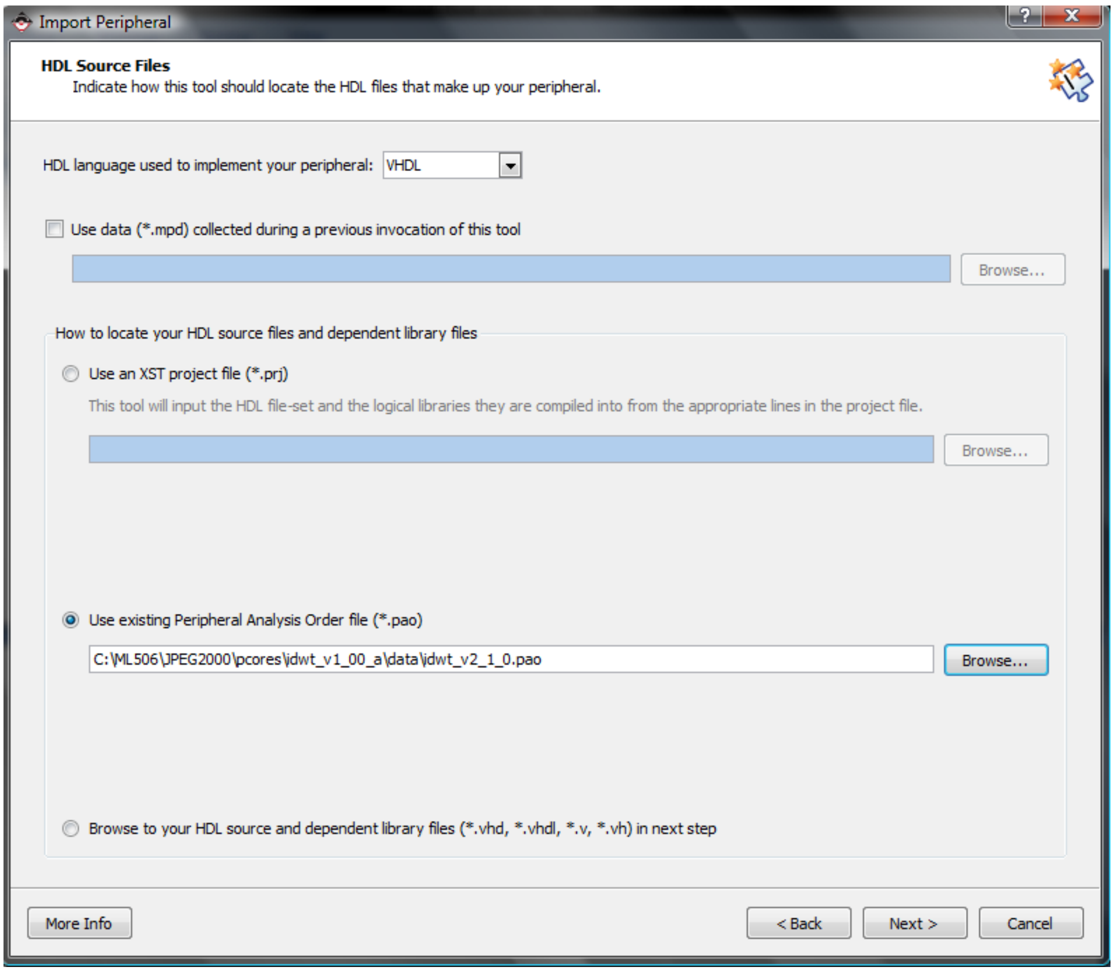
Flow description
This tool will help you import a fully implemented EDK compliant peripheral into an XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the conventions required by EDK.

3.25 Select "To an XPS project", ensure that the folder chosen is the project folder, and click "Next".

3.26 For the name of the peripheral, type "idwt". Tick "Use version" and select the same version number that we originally created. Click "Next". It will ask if we are willing to overwrite the existing peripheral and we should answer "Yes".

3.27 Now we are asked about the files that make up our peripheral. Tick "HDL source files" and click "Next".

3.28 Select "Use existing Peripheral Analysis Order file (*.pao)" and click "Browse". From the project folder, go to "pcores\idwt_v1_00_a\data" and select the "idwt_v2_1_0.pao" file. Click "Next".


Import Peripheral

HDL Source Files
Indicate how this tool should locate the HDL files that make up your peripheral.

HDL language used to implement your peripheral: VHDL

☐ Use data (*.mpd) collected during a previous invocation of this tool

How to locate your HDL source files and dependent library files

○ Use an XST project file (*.prj)
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.

● Use existing Peripheral Analysis Order file (*.pao)
C:\ML506\JPEG2000\pcores\idwt_v1_00_a\data\idwt_v2_1_0.pao

○ Browse to your HDL source and dependent library files (*.vhd, *.vhdl, *.v, *.vh) in next step

3.29 On the HDL analysis information page, if you scroll down, you will see the .vhd source files we added along with the 2 generated files (“idwt.vhd” and “user_logic.vhd”) listed in the bottom. Click "Next". The wizard will mention if any errors are found in the design.

3.30 On the Bus Interfaces page, tick "PLB Slave" and click "Next".

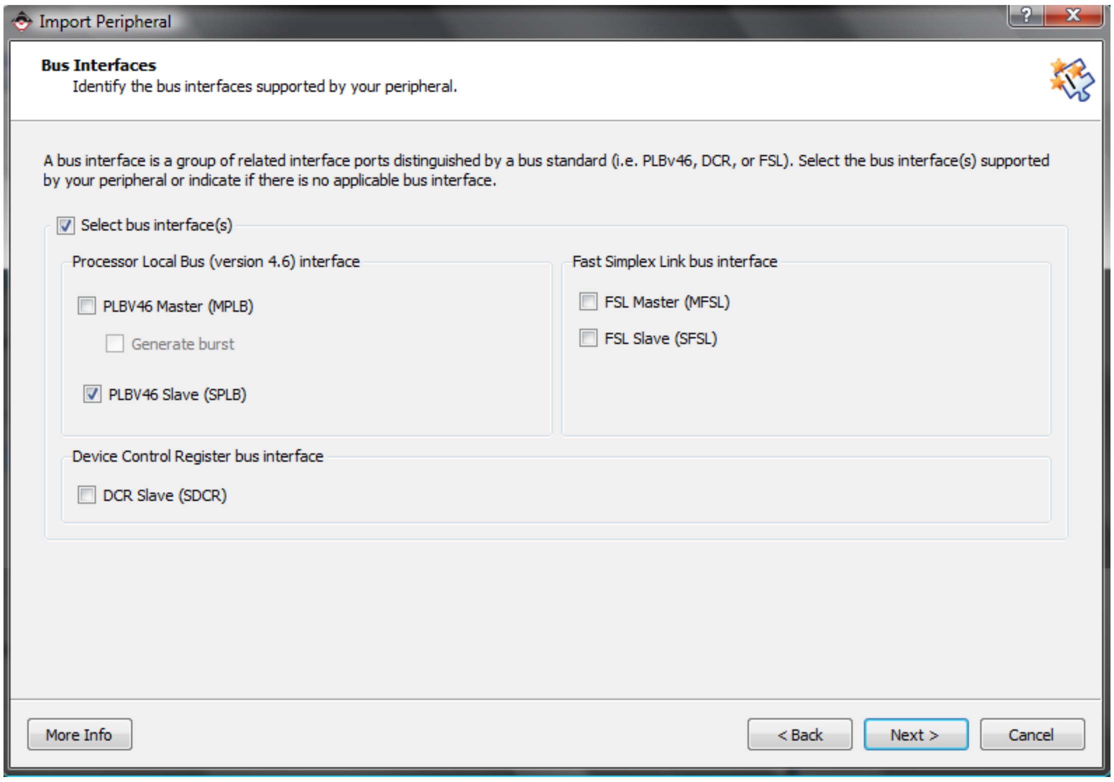
3.31 On the SPLB: Port page, click "Next".

3.32 On the "Parameter Attributes" page, in the register space field, select “C_HIGHADDR” for “Parameter determine high address” and click "Next".

3.32 On the "Port Attributes" page, click "Next".

3.33 Click "Finish".

The multiplier peripheral should now be accessible through the "IP Catalog->Project Local pcores" in the XPS interface.



Bus Interfaces
Identify the bus interfaces supported by your peripheral.

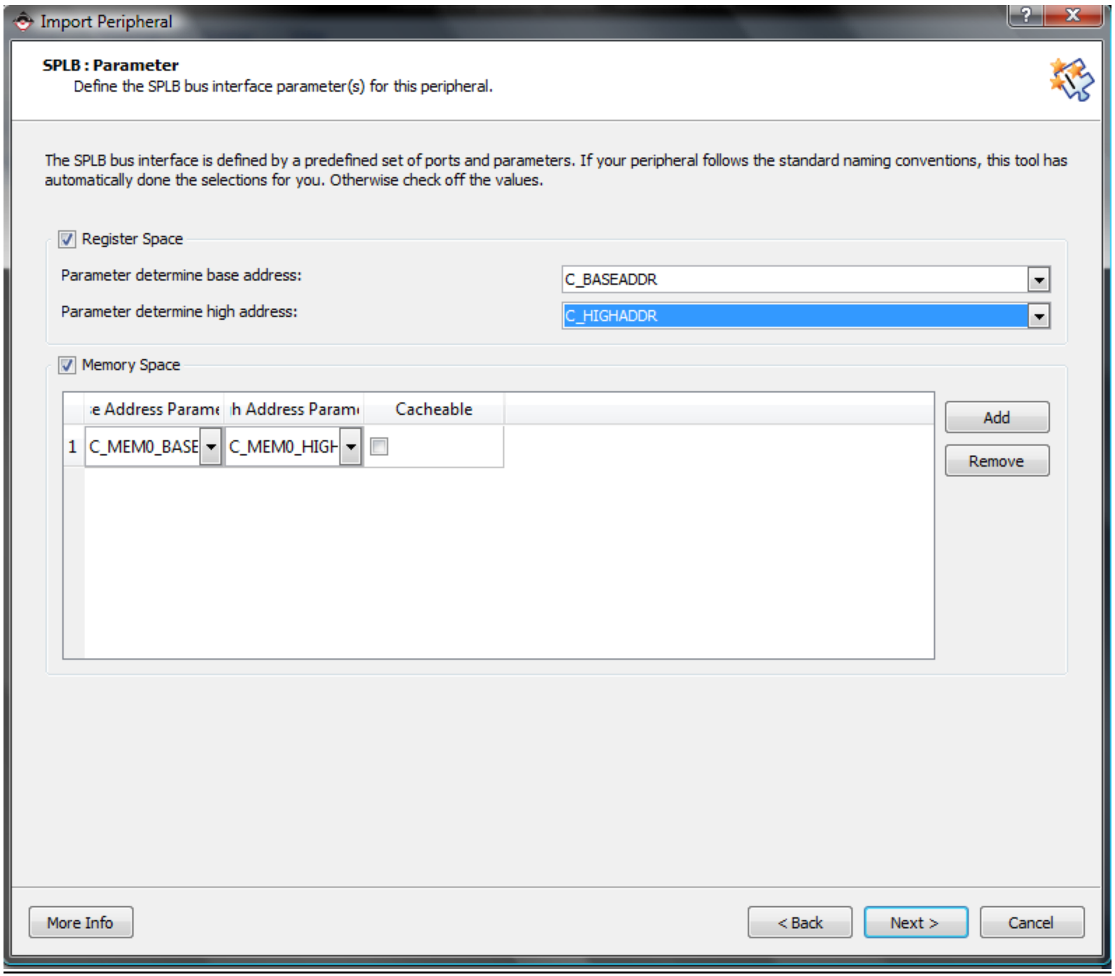
A bus interface is a group of related interface ports distinguished by a bus standard (i.e. PLBv46, DCR, or FSL). Select the bus interface(s) supported by your peripheral or indicate if there is no applicable bus interface.

[✓] Select bus interface(s)

Processor Local Bus (version 4.6) interface
[ ] PLBV46 Master (MPLB)
[ ] Generate burst
[✓] PLBV46 Slave (SPLB)

Fast Simplex Link bus interface
[ ] FSL Master (MFSL)
[ ] FSL Slave (SFSL)

Device Control Register bus interface
[ ] DCR Slave (SDCR)



SPLB : Parameter
Define the SPLB bus interface parameter(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise check off the values.

[✓] Register Space
Parameter determine base address: C_BASEADDR
Parameter determine high address: C_HIGHADDR

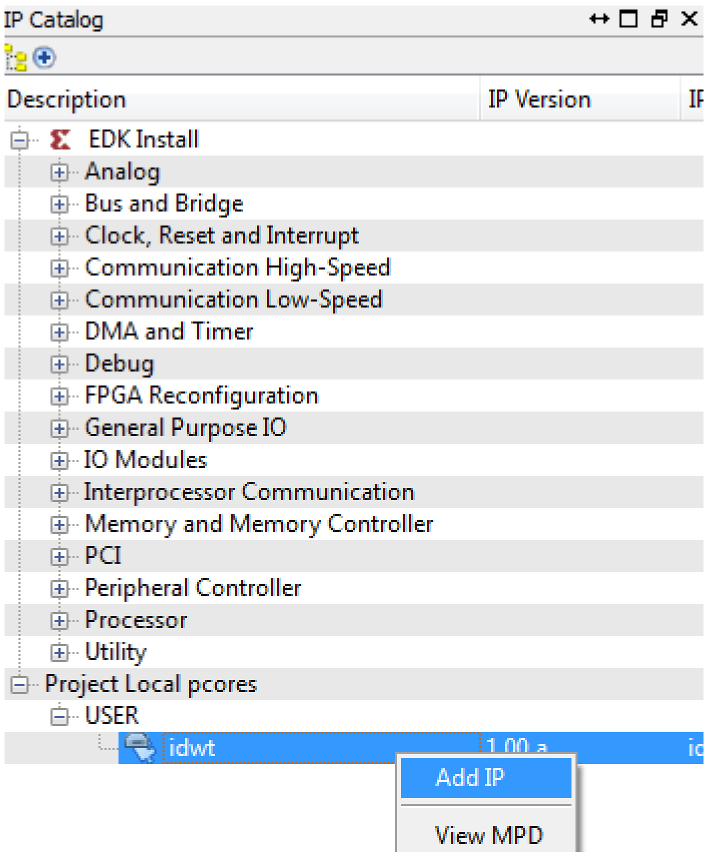
[✓] Memory Space

	;e Address Parame	h Address Parame	Cacheable
1	C_MEM0_BASE	C_MEM0_HIGH	[ ]

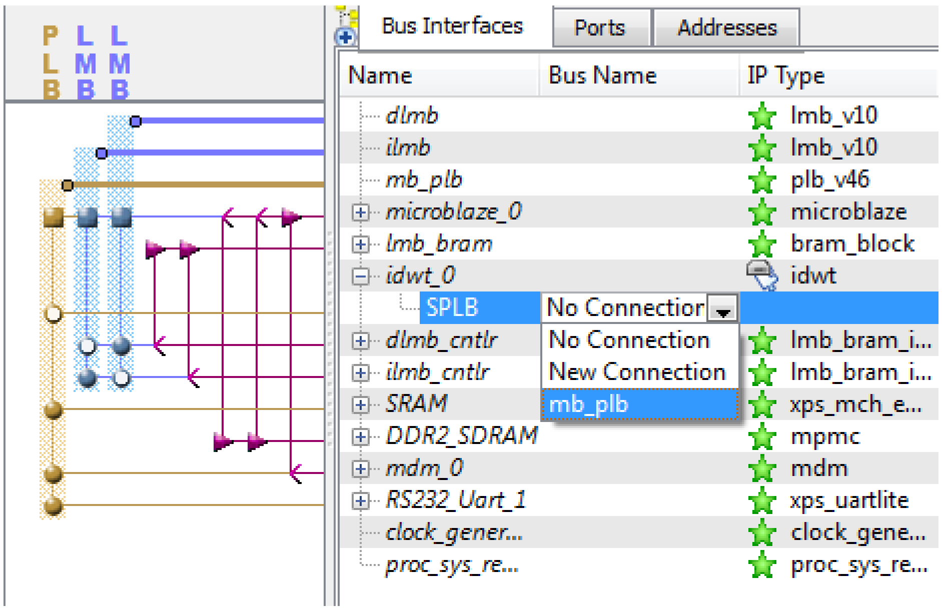
Create an Instance of the Peripheral – Steps

Follow these steps to create an instance of the peripheral in the project.

3.35 From the "IP Catalog" find the "idwt" IP core in the "Project Repository" group. Right click on the core and select "Add IP".

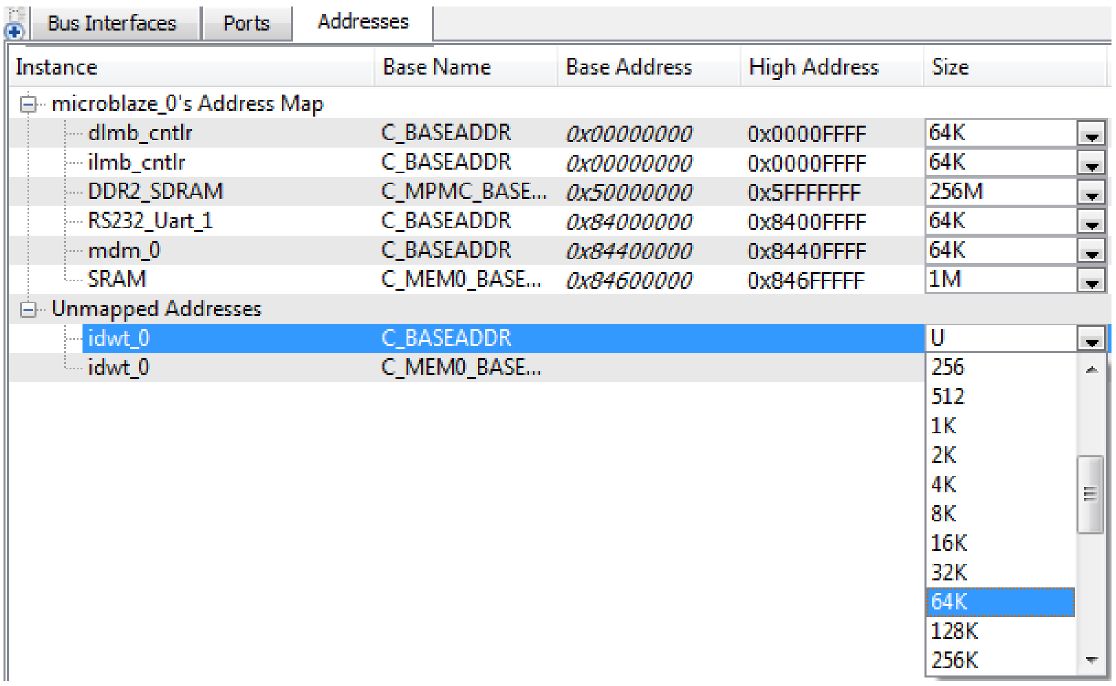


3.36 From the "System Assembly View" using the "Bus Interface" filter, connect the "idwt_0" to the PLB bus.



Name	Bus Name	IP Type
dlmb		lmb_v10
ilmb		lmb_v10
mb_plb		plb_v46
microblaze_0		microblaze
lmb_bram		bram_block
idwt_0		idwt
SPLB	No Connection	
dlmb_cntlr		lmb_bram_i...
ilmb_cntlr		lmb_bram_i...
SRAM		xps_mch_e...
DDR2_SDRAM		mpmc
mdm_0		mdm
RS232_Uart_1		xps_uartlite
clock_gener...		clock_gene...
proc_sys_re...		proc_sys_re...

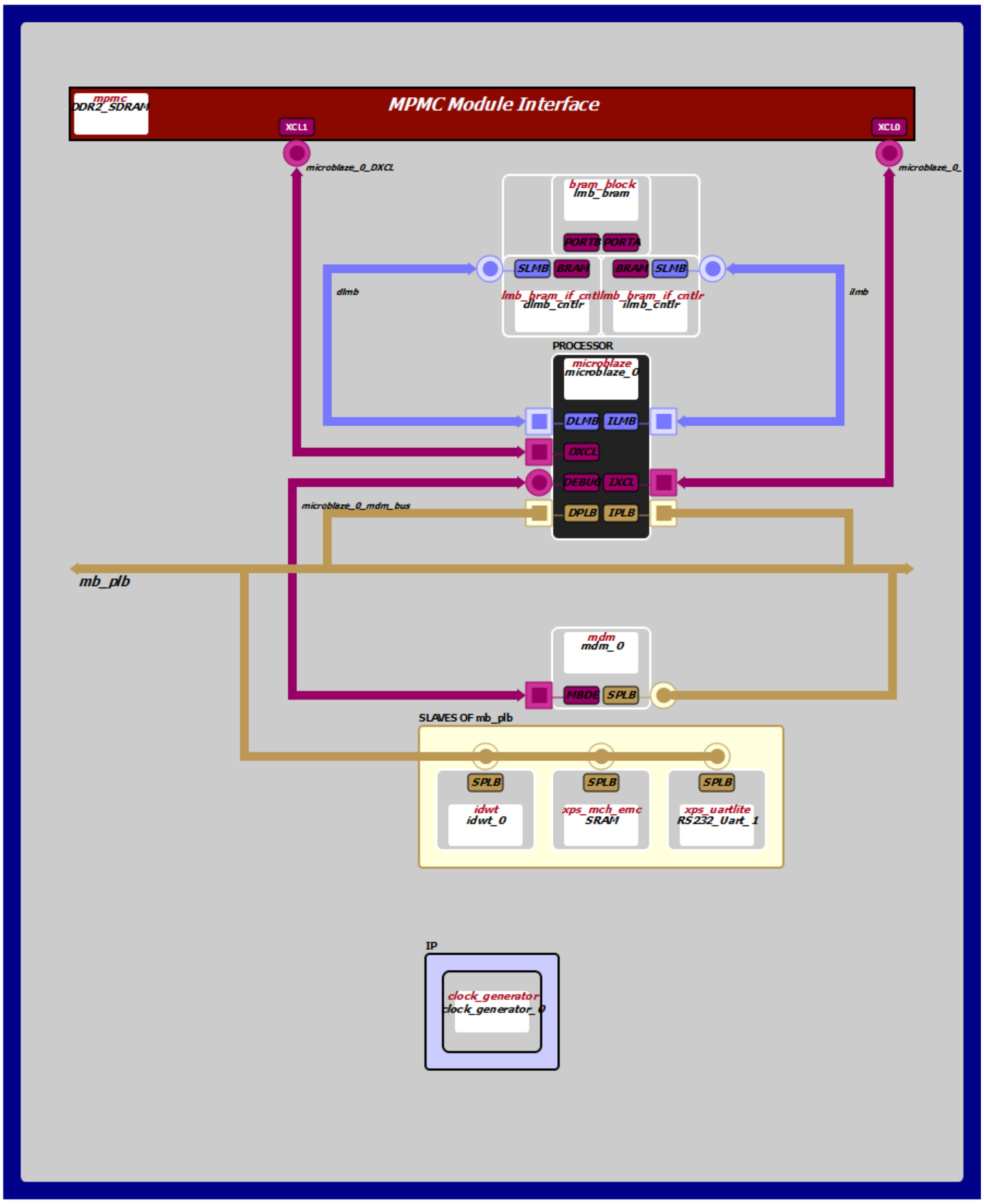
3.37 Click on the "Addresses" filter. Change the "Size" for "idwt_0" to 64K for BASEADDR and to 512K for C_MEM0_BASEADDR. These sizes refer to register and memory space respectively. Also change DDR2_SDRAM size to 32M. Then click "Generate Addresses".



Instance	Base Name	Base Address	High Address	Size
microblaze_0's Address Map				
dlmb_cntlr	C_BASEADDR	0x00000000	0x0000FFFF	64K
ilmb_cntlr	C_BASEADDR	0x00000000	0x0000FFFF	64K
DDR2_SDRAM	C_MPMC_BASE...	0x50000000	0x5FFFFFFF	256M
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K
SRAM	C_MEM0_BASE...	0x84600000	0x846FFFFF	1M
Unmapped Addresses				
idwt_0	C_BASEADDR			U
idwt_0	C_MEM0_BASE...			

Now we have an instance of the IDWT peripheral in our project, so our hardware design is complete for now. In Figure 11.3 the block diagram of the hardware design is illustrated. It can be observed by clicking on the “Block Diagram” tab of the XPS GUI and it shows all the bus connections, bus types, peripherals and their instance names, memories, clock generators and the processor of our design. Notice that the IDWT peripheral is hooked up to the PLB along with the UART, the SRAM, etc. Finally, notice the LMB that connects the on-chip Block Ram to the Microblaze soft processor and the distinction between instruction and data ports (Harvard architecture).

In the next section we will proceed with the software development part of the co-design process. During the procedure we might need to apply changes to the hardware design. Therefore the above hardware design is not final. However, any changes that will apply would be better presented in the next steps to demonstrate the close relation and the constraints between SW and HW and how software development decisions affect hardware design and vice versa.



SPECS	
EDK VERSION	12.1
ARCH	virtex5
PART	xc5vsx50tff1136-1
GENERATED	Wed Mar 16 02:00:17 2011

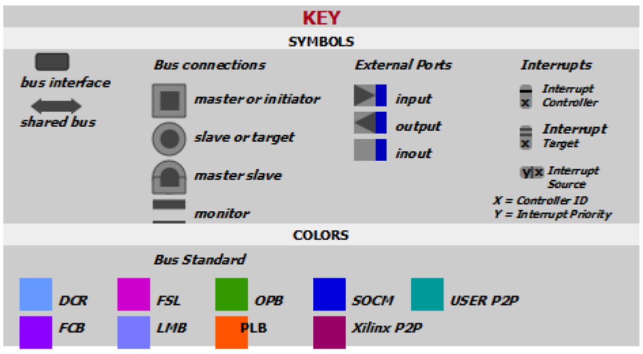


Figure 11.3: Block Diagram

SOFTWARE DEVELOPMENT

For the software development stage of the design process we can follow two ways: continue using XPS or use SDK instead. SDK is suggested for building big software projects from scratch. However, we will proceed by using XPS, as it is assumed that the software design, which is a modification of the JasPer software, is already available to the user. This will also provide a wider understanding of the XPS and XPS GUI functionalities and capabilities.

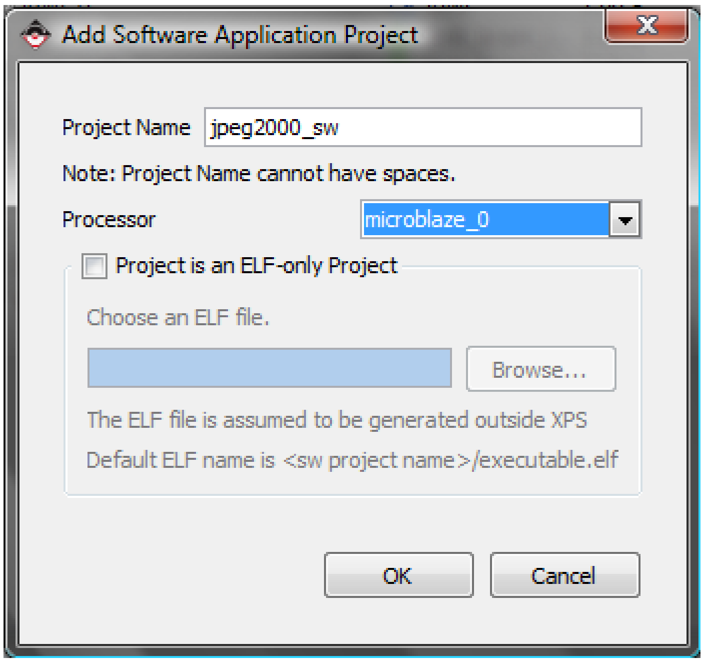
11.4 Creating a new software project

In this section we will create a new software project and import it as a new application using XPS.

Steps

4.1 In the "Applications" tab double click on "Add Software Application Project".

4.2 In the "Project name" field type "jpeg2000_sw" and click "OK". A project named "jpeg2000_sw" is immediately added in the "Applications" list.


Add Software Application Project

Project Name jpeg2000_sw

Note: Project Name cannot have spaces.

Processor microblaze_0

Project is an ELF-only Project

Choose an ELF file.

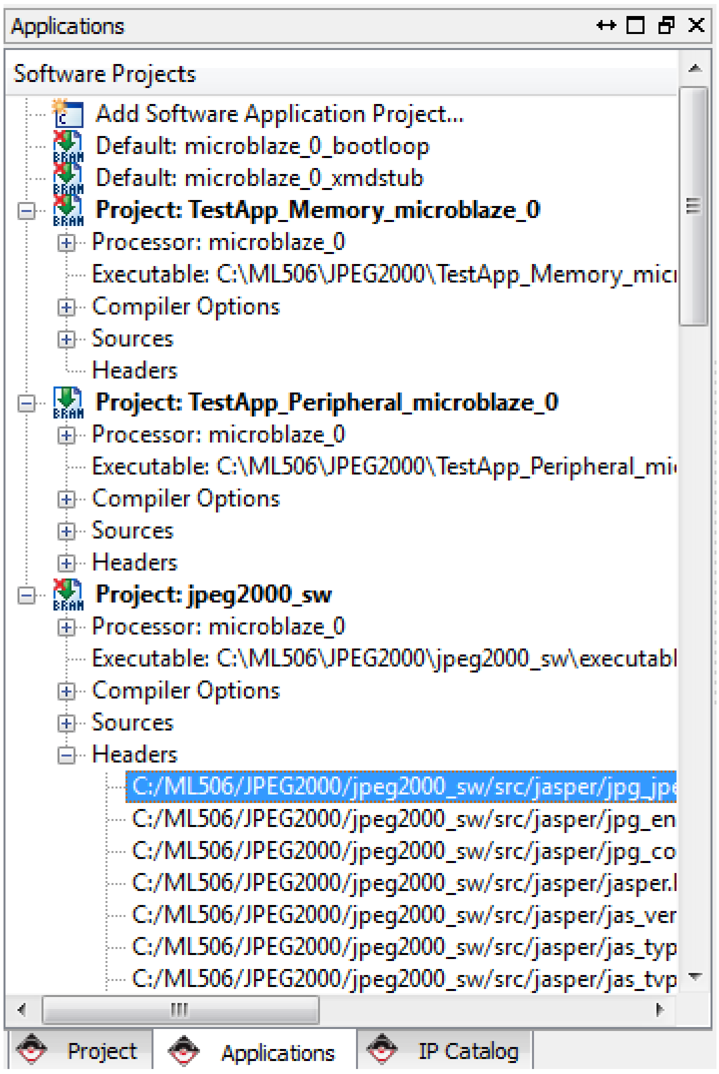
The ELF file is assumed to be generated outside XPS

Default ELF name is <sw project name>/executable.elf

4.3 Browse into the project's directory (for this case "C:\ML506\JPEG2000") and create a file named "jpeg2000_sw". In this file, XPS will store the linker script file and the ELF file. Open the new file and copy the "src" file that is included in the modified software directory.

4.4 From the "Applications" tab, right-click on "Sources" within the "Project: jpeg2000_sw" tree. Click “Add Existing Files...” and add all the .c source files that are included in the modified JasPer software directory.

4.5 Again, from the "Applications" tab, right-click on "Headers" within the "Project: jpeg2000_sw" tree. Click “Add Existing Files...” and add all the .h header files that are included in the modified JasPer software directory.

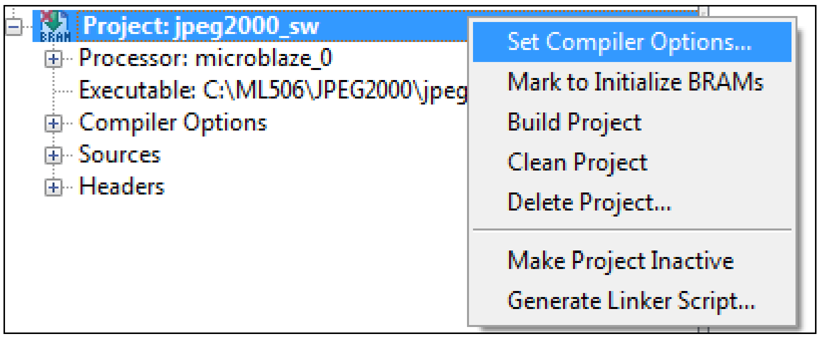


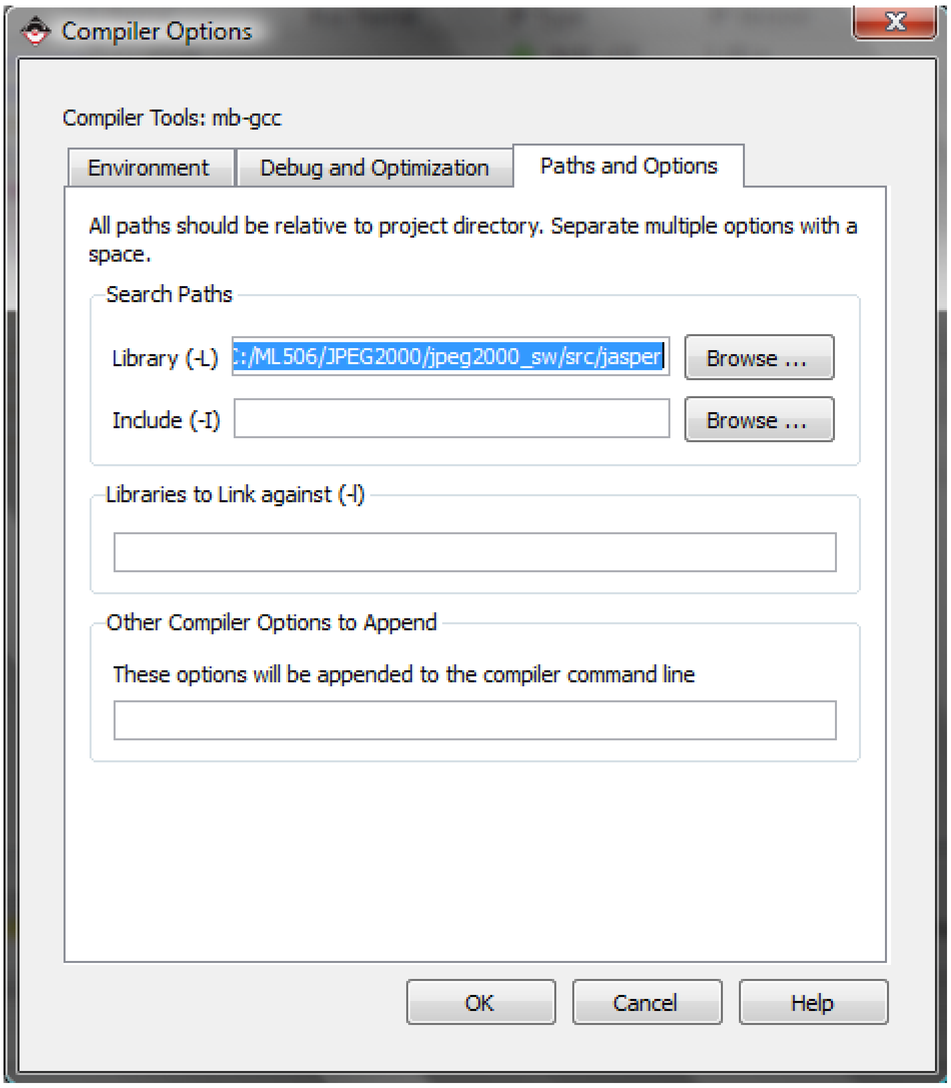
```
Applications
Software Projects
  Add Software Application Project...
  Default: microblaze_0_bootloop
  Default: microblaze_0_xmdstub
  Project: TestApp_Memory_microblaze_0
    Processor: microblaze_0
    Executable: C:\ML506\JPEG2000\TestApp_Memory_mici
    Compiler Options
    Sources
    Headers
  Project: TestApp_Peripheral_microblaze_0
    Processor: microblaze_0
    Executable: C:\ML506\JPEG2000\TestApp_Peripheral_mi
    Compiler Options
    Sources
    Headers
  Project: jpeg2000_sw
    Processor: microblaze_0
    Executable: C:\ML506\JPEG2000\jpeg2000_sw\executabl
    Compiler Options
    Sources
    Headers
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jpg_jpe
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jpg_en
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jpg_co
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jasper.l
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jas_ver
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jas_typ
      C:/ML506/JPEG2000/jpeg2000_sw/src/jasper/jas_tvp
Project | Applications | IP Catalog
```

<u>Setting the library path - Steps</u>

4.6 Right-click on "Project: jpeg2000_sw" and click “Set Compiler Options”.

4.7 In the “Paths and Options” tab, click “Browse” for Library search paths and open the folder named “jasper” (C:/ML506/JPEG2000/jpeg2000_sw/src/jasper, for this project). This way we set the search path for our library.





Compiler Tools: mb-gcc

All paths should be relative to project directory. Separate multiple options with a space.

Search Paths

Library (-L) :/ML506/JPEG2000/jpeg2000_sw/src/jasper

Include (-I)

Libraries to Link against (-l)

Other Compiler Options to Append

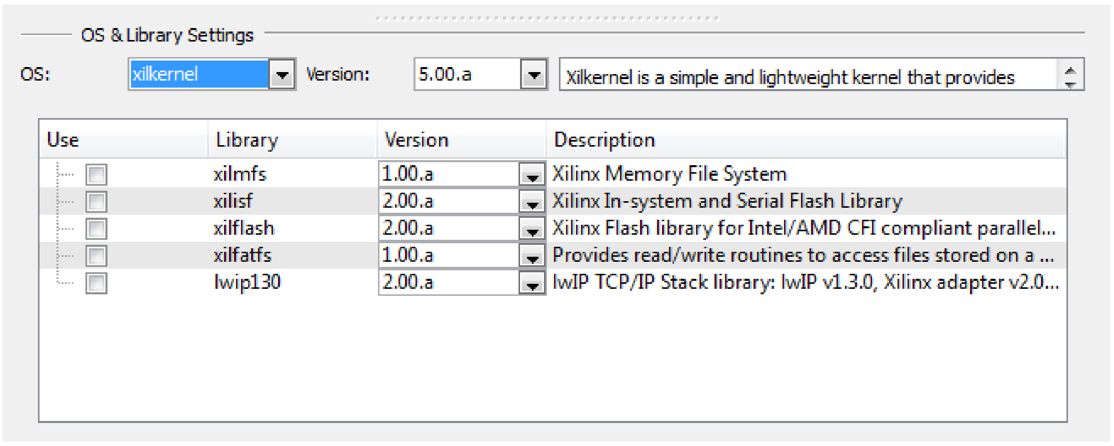
These options will be appended to the compiler command line

Setting and configuring the Xilkernel OS – Steps

Xilkernel, as explained in the previous chapter, is a highly customizable and lightweight kernel that also allows for (). Follow these steps to set and configure the kernel.

4.8 From the XPS software, select "Software->Software Platform Settings".

4.9 From the "Software Platform Settings" window, select "Software Platform". Under "OS & Library Settings", change "standalone" to "xilkernel".

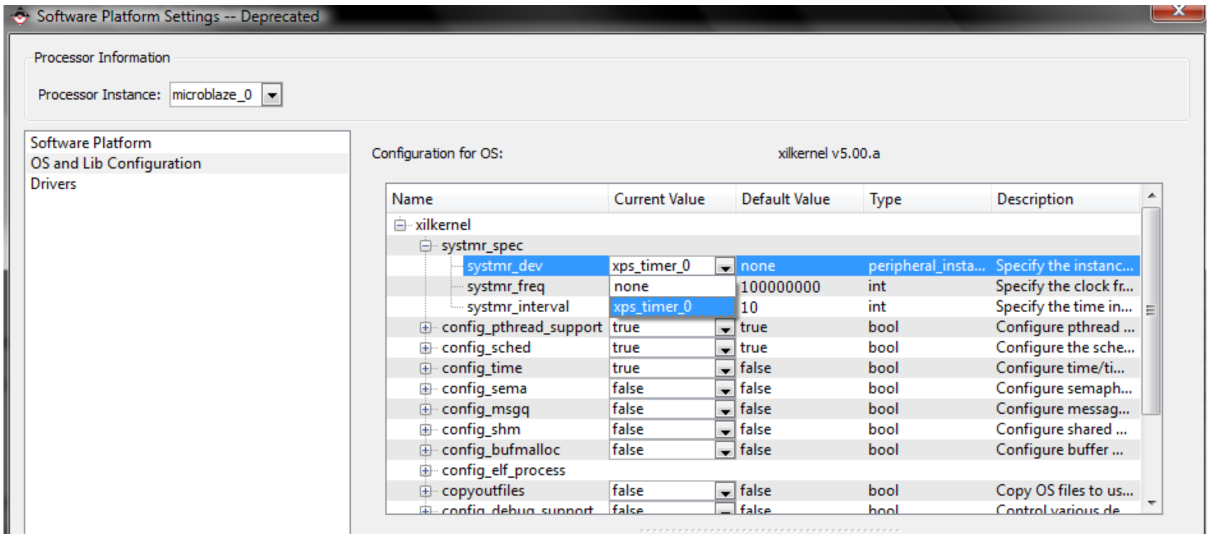


OS & Library Settings

OS: xilkernel   Version: 5.00.a   Xilkernel is a simple and lightweight kernel that provides

Use	Library	Version	Description
☐	xilmfs	1.00.a	Xilinx Memory File System
☐	xilisf	2.00.a	Xilinx In-system and Serial Flash Library
☐	xilflash	2.00.a	Xilinx Flash library for Intel/AMD CFI compliant parallel...
☐	xilfatfs	1.00.a	Provides read/write routines to access files stored on a ...
☐	lwip130	2.00.a	lwIP TCP/IP Stack library: lwIP v1.3.0, Xilinx adapter v2.0...

4.10 Again from the "Software Platform Settings" window, select "OS and Lib Configuration". Under "Configuration for OS", expand "xilkernel", expand "sys_tmr_spec" and for "sys_tmr_dev" select "xps_timer_0".



Software Platform Settings -- Deprecated

Processor Information

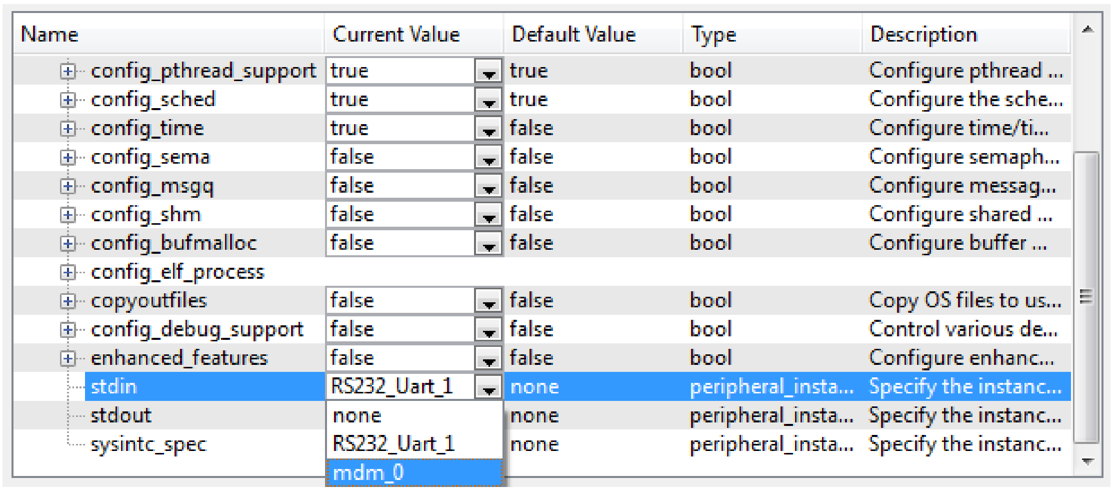
Processor Instance: microblaze_0

Software Platform
OS and Lib Configuration
Drivers

Configuration for OS: xilkernel v5.00.a

Name	Current Value	Default Value	Type	Description
xilkernel				
systmr_spec				
systmr_dev	xps_timer_0	none	peripheral_insta...	Specify the instanc...
systmr_freq	none	100000000	int	Specify the clock fr...
systmr_interval	xps_timer_0	10	int	Specify the time in...
config_pthread_support	true	true	bool	Configure pthread ...
config_sched	true	true	bool	Configure the sche...
config_time	true	false	bool	Configure time/ti...
config_sema	false	false	bool	Configure semaph...
config_msgq	false	false	bool	Configure messag...
config_shm	false	false	bool	Configure shared ...
config_bufmalloc	false	false	bool	Configure buffer ...
config_elf_process				
copyoutfiles	false	false	bool	Copy OS files to us...
config_debug_support	false	false	bool	Control various de

4.11 At the bottom of the same sub-window, spot "stdin" and "stdout" selections. Select "mdm_0" for both of them. This will redirect the standard input and output to the Xilinx Microprocessor Debugger, which we will use to test our design.



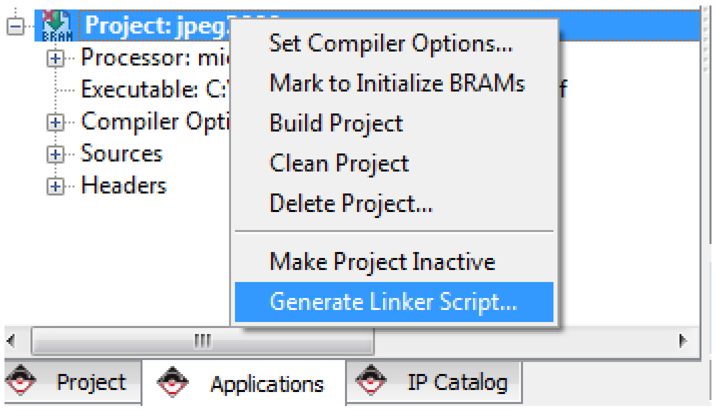
Name	Current Value	Default Value	Type	Description
config_pthread_support	true	true	bool	Configure pthread ...
config_sched	true	true	bool	Configure the sche...
config_time	true	false	bool	Configure time/ti...
config_sema	false	false	bool	Configure semaph...
config_msgq	false	false	bool	Configure messag...
config_shm	false	false	bool	Configure shared ...
config_bufmalloc	false	false	bool	Configure buffer ...
config_elf_process				
copyoutfiles	false	false	bool	Copy OS files to us...
config_debug_support	false	false	bool	Control various de...
enhanced_features	false	false	bool	Configure enhanc...
stdin	RS232_Uart_1	none	peripheral_insta...	Specify the instanc...
stdout	none	none	peripheral_insta...	Specify the instanc...
sysintc_spec	RS232_Uart_1	none	peripheral_insta...	Specify the instanc...
	mdm_0			

4.12 Click "OK". Notice that the MSS file has changed and has been saved by XPS.

Generating the linker script – Steps

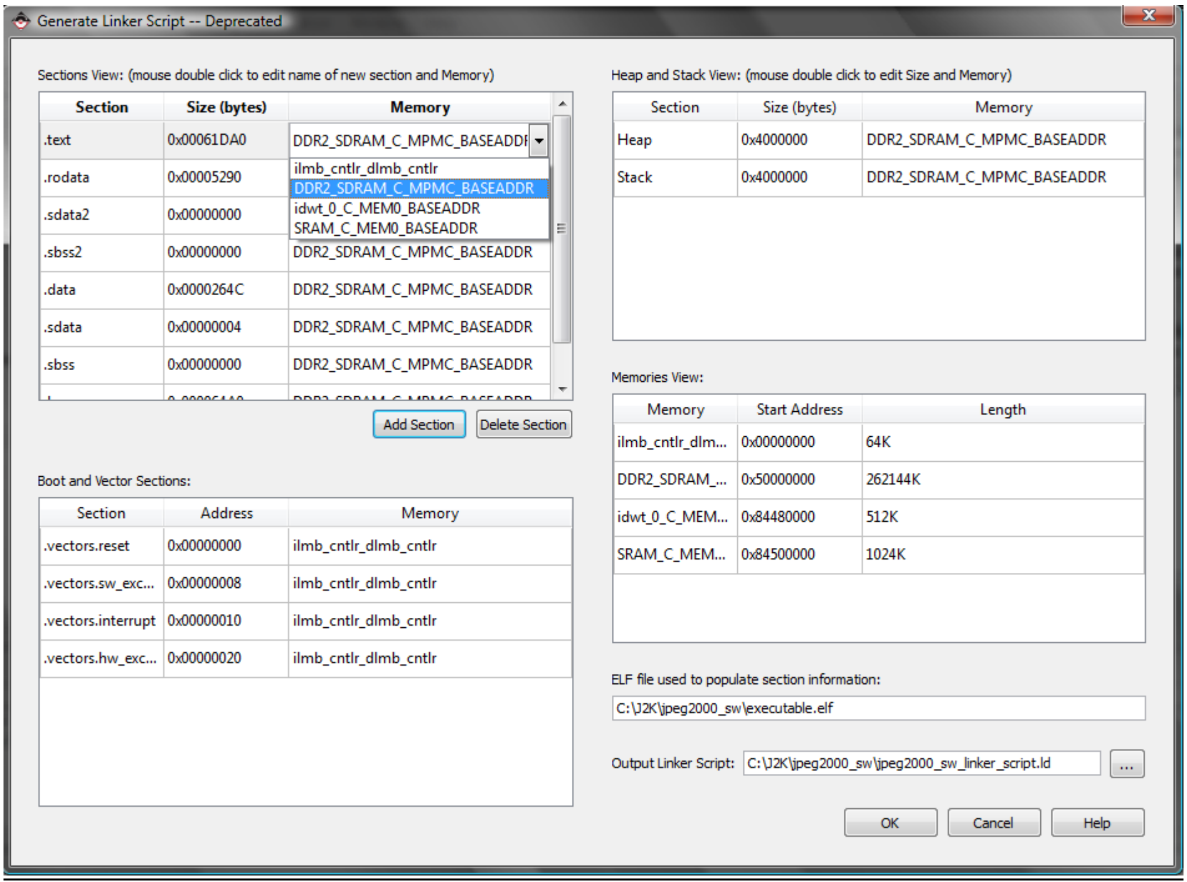
The next steps will help us create a custom linker script that will map different sections of the ELF executable file (.text, .heap, .stack, etc.) to memory. Our software project is generally big (the ELF file has a size of 1.2 MB), so it will be downloaded in the DDR2_SDRAM and be executed from there. The on-chip BRAM is initialized with a boot-loader application (along with boot and vector sections), that branches to the starting address of the DDR2_SDRAM where our main application will be downloaded. It is worth noticing that we will use large sized heap and stack sections, because the software uses memory allocation functions extensively, and manipulates large data arrays.

4.8 Right-click on "Project: jpeg2000_sw" and click "Generate Linker Script".



4.9 In "Sections View", for each section double click on the "Memory" fields and select "DDR2_SDRAM_C_MPMC_BASEADDR". This will load the sections to the SDRAM when we download the ELF file on the board.

4.10 In the "Heap and Stack View", both for heap and stack, double click on the "Size" field and change it to 0x40000. Double-click on the "Memory" fields and select "DDR2_SDRAM_C_MPMC_BASEADDR".


Generate Linker Script -- Deprecated

Sections View: (mouse double click to edit name of new section and Memory)

Section	Size (bytes)	Memory
.text	0x00061DA0	DDR2_SDRAM_C_MPMC_BASEADDR
.rodata	0x00005290	
.sdata2	0x00000000	
.sbss2	0x00000000	DDR2_SDRAM_C_MPMC_BASEADDR
.data	0x0000264C	DDR2_SDRAM_C_MPMC_BASEADDR
.sdata	0x00000004	DDR2_SDRAM_C_MPMC_BASEADDR
.sbss	0x00000000	DDR2_SDRAM_C_MPMC_BASEADDR

ilmb_cntlr_dlmb_cntlr
DDR2_SDRAM_C_MPMC_BASEADDR
idwt_0_C_MEM0_BASEADDR
SRAM_C_MEM0_BASEADDR

Add Section | Delete Section

Boot and Vector Sections:

Section	Address	Memory
.vectors.reset	0x00000000	ilmb_cntlr_dlmb_cntlr
.vectors.sw_exc...	0x00000008	ilmb_cntlr_dlmb_cntlr
.vectors.interrupt	0x00000010	ilmb_cntlr_dlmb_cntlr
.vectors.hw_exc...	0x00000020	ilmb_cntlr_dlmb_cntlr

Heap and Stack View: (mouse double click to edit Size and Memory)

Section	Size (bytes)	Memory
Heap	0x4000000	DDR2_SDRAM_C_MPMC_BASEADDR
Stack	0x4000000	DDR2_SDRAM_C_MPMC_BASEADDR

Memories View:

Memory	Start Address	Length
ilmb_cntlr_dlm...	0x00000000	64K
DDR2_SDRAM_...	0x50000000	262144K
idwt_0_C_MEM...	0x84480000	512K
SRAM_C_MEM...	0x84500000	1024K

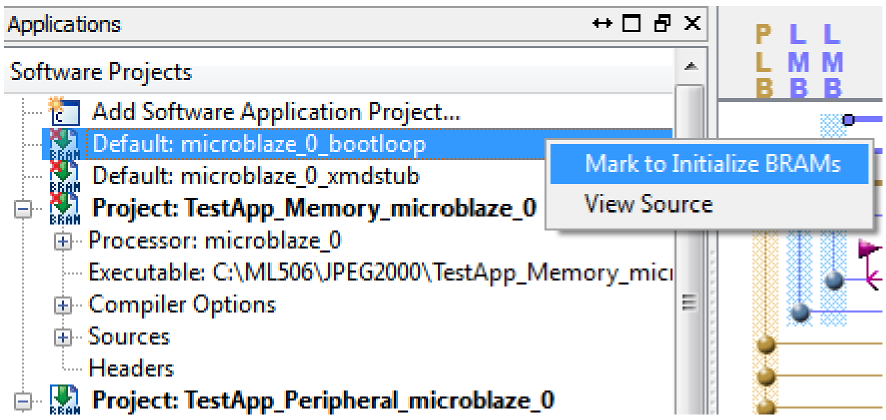
ELF file used to populate section information:
C:\J2K\jpeg2000_sw\executable.elf

Output Linker Script: C:\J2K\jpeg2000_sw\jpeg2000_sw_linker_script.ld

OK | Cancel | Help

<u>Initialize BRAMs and download bitstream – Steps</u>

4.11 From the "Applications" tab, right click on "Default: microblaze_0_bootloop" and select "Mark to Initialize BRAMs".


Applications
Software Projects
Add Software Application Project...
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp_Memory_microblaze_0
Processor: microblaze_0
Executable: C:\ML506\JPEG2000\TestApp_Memory_mici
Compiler Options
Sources
Headers
Project: TestApp_Peripheral_microblaze_0
Mark to Initialize BRAMs
View Source
PLB
LMB
LMB

4.12 Turn on the ML506 board.

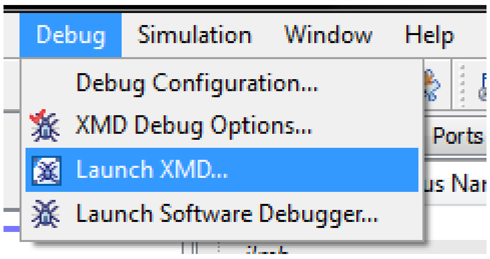
4.13 From the XPS software, select "Device Configuration->Download Bitstream".

11.5 Testing the design

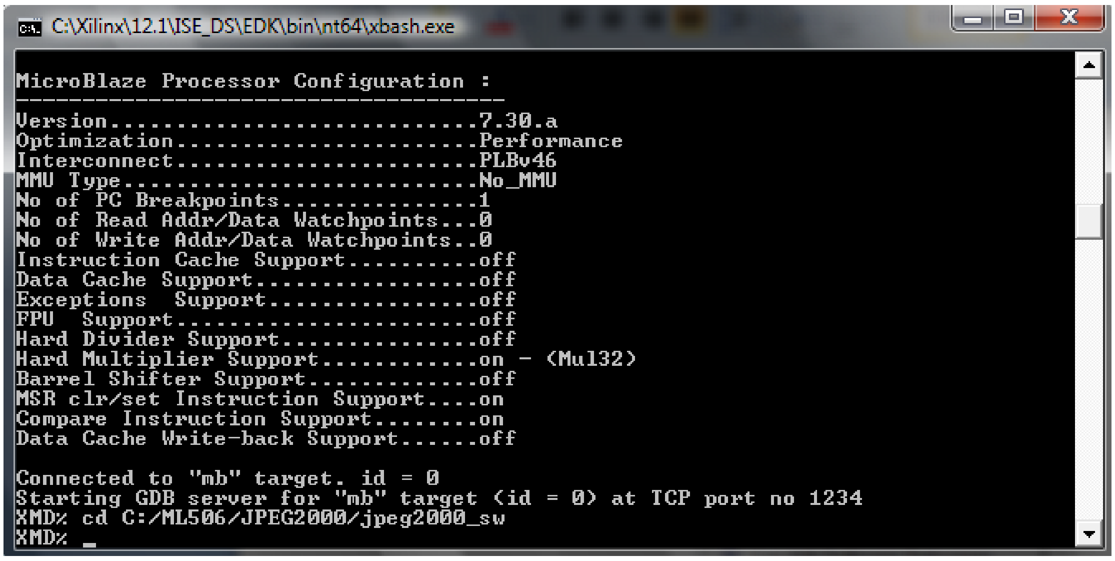
<u>Testing the application by using XMD – Steps</u>

Follow these steps to test the application using Xilinx Microprocessor Debugger. The system decompresses an input image that is in JPEG2000 format ("lena256.jp2") and produces an output log file via the serial port that can then be opened with an image viewer. The image viewer (for example Irfanview) will automatically convert the file to PNM format ("lena256.pnm"). It is assumed that the input image (and any other images that we may want to use as a test-case) are located into the C:/ML506/JPEG2000/jpeg2000_sw folder.

5.1 From the XPS software, select "Debug->Launch XMD".

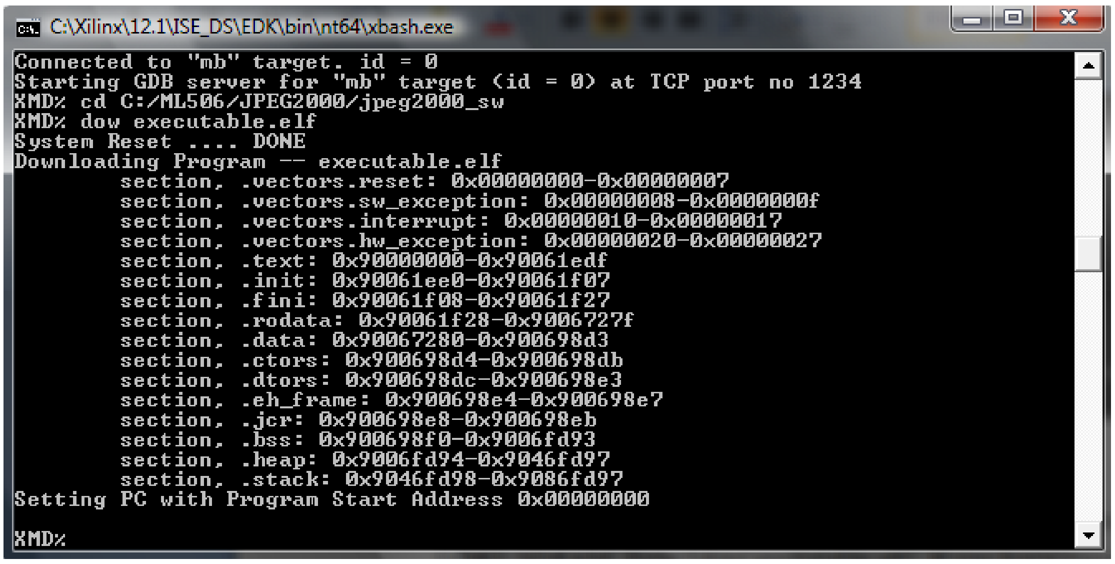


5.2 The XMD console opens and the connection is set, and the user can view the configuration of the Microblaze soft processor. In the console type the directory of the software project. For this project: "cd C:/ML506/JPEG2000/jpeg2000_sw".



```
MicroBlaze Processor Configuration :
-------------------------------------
Version............................7.30.a
Optimization.......................Performance
Interconnect.......................PLBv46
MMU Type...........................No_MMU
No of PC Breakpoints...............1
No of Read Addr/Data Watchpoints...0
No of Write Addr/Data Watchpoints..0
Instruction Cache Support..........off
Data Cache Support.................off
Exceptions  Support................off
FPU  Support.......................off
Hard Divider Support...............off
Hard Multiplier Support............on - (Mul32)
Barrel Shifter Support.............off
MSR clr/set Instruction Support....on
Compare Instruction Support........on
Data Cache Write-back Support......off

Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% cd C:/ML506/JPEG2000/jpeg2000_sw
XMD% _
```

5.3 Now use the "dow" command to download the software program on the DDR2_SDRAM. Type "dow executable.elf". If you have created the ELF file with a different name then use the correct ELF name for this command. After the program is downloaded you can view the way that the different sections of the program have been mapped on the memory.

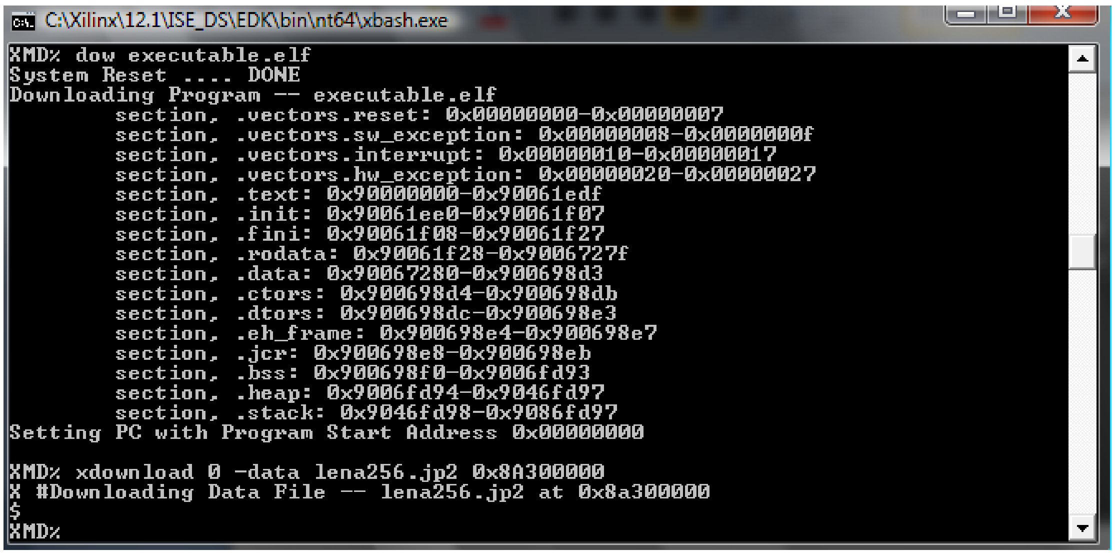


```
C:\Xilinx\12.1\ISE_DS\EDK\bin\nt64\xbash.exe
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% cd C:/ML506/JPEG2000/jpeg2000_sw
XMD% dow executable.elf
System Reset .... DONE
Downloading Program -- executable.elf
        section, .vectors.reset: 0x00000000-0x00000007
        section, .vectors.sw_exception: 0x00000008-0x0000000f
        section, .vectors.interrupt: 0x00000010-0x00000017
        section, .vectors.hw_exception: 0x00000020-0x00000027
        section, .text: 0x90000000-0x90061edf
        section, .init: 0x90061ee0-0x90061f07
        section, .fini: 0x90061f08-0x90061f27
        section, .rodata: 0x90061f28-0x9006727f
        section, .data: 0x90067280-0x900698d3
        section, .ctors: 0x900698d4-0x900698db
        section, .dtors: 0x900698dc-0x900698e3
        section, .eh_frame: 0x900698e4-0x900698e7
        section, .jcr: 0x900698e8-0x900698eb
        section, .bss: 0x900698f0-0x9006fd93
        section, .heap: 0x9006fd94-0x9046fd97
        section, .stack: 0x9046fd98-0x9086fd97
Setting PC with Program Start Address 0x00000000

XMD%
```

5.4 Open a hyperterminal (i.e TeraTerm or Hyperterminal) in order to establish a connection from the host PC to the FPGA device via the serial port. For this project the hyperterminal settings are:

Baud rate	9600 bps
Data bits	8 bits
Parity	None
Stop bits	1 bit
Flow control	None

5.5 Now use the "xdownload" command in order to download the input image in SRAM. Type "xdownload 0 -data lena256.jp2 0x8A300000". "0x8A300000" is the base address of the SRAM for this project (it can be viewed in the System Assembly window and in the xparameters.h header file).



```
XMD% dow executable.elf
System Reset .... DONE
Downloading Program -- executable.elf
        section, .vectors.reset: 0x00000000-0x00000007
        section, .vectors.sw_exception: 0x00000008-0x0000000f
        section, .vectors.interrupt: 0x00000010-0x00000017
        section, .vectors.hw_exception: 0x00000020-0x00000027
        section, .text: 0x90000000-0x90061edf
        section, .init: 0x90061ee0-0x90061f07
        section, .fini: 0x90061f08-0x90061f27
        section, .rodata: 0x90061f28-0x9006727f
        section, .data: 0x90067280-0x900698d3
        section, .ctors: 0x900698d4-0x900698db
        section, .dtors: 0x900698dc-0x900698e3
        section, .eh_frame: 0x900698e4-0x900698e7
        section, .jcr: 0x900698e8-0x900698eb
        section, .bss: 0x900698f0-0x9006fd93
        section, .heap: 0x9006fd94-0x9046fd97
        section, .stack: 0x9046fd98-0x9086fd97
Setting PC with Program Start Address 0x00000000

XMD% xdownload 0 -data lena256.jp2 0x8A300000
X #Downloading Data File -- lena256.jp2 at 0x8a300000
$
XMD%
```

5.6 Prepare the hyperterminal to receive a file or create a log file.

5.7 Now the program is ready to run and send the output file via the RS232 to the host PC. In the XMD console type “run”. The time needed to complete the transfer depends on the baud rate that has been set for the RS232 UART.

5.8 After the output file is received, open it with an image viewer software, such as Irfanview. De-compression is complete!

APPENDIX

In this section we present some of the issues and bugs that had to be dealt with during the implementation of JPEG2000. Some hints and tips are also given, regarding decisions that had to be taken during the design of the project.

Issue #1 - XPS vs. SDK for software development

Prior to version 11.x, SDK had always been dependent on XPS. In 12.x XPS continues to offer a basic software development IDE. However, Xilinx has focused on SDK for s/w development and XPS for h/w development, therefore s/w development on XPS is deprecated in latest versions. The flow should be: do h/w development in XPS, export the hardware specification, and move on to SDK for software development. Unfortunately this is not made "clear" by the tools. One example is that SDK doesn't make clear that it has a separate copy of the MSS file. Therefore, building some basic software in XPS and exporting the design to SDK does not mean that changes in XPS reflect changes in SDK. For the purposes of this thesis, using XPS to build an already developed software application was enough. However, when it comes to building big software projects from scratch SDK should be the only tool used, as it offers more software-oriented capabilities (for example, GNU debugger is no longer supported in XPS, but only in SDK).

Issue #2 – Memory allocation issues in Microblaze

MicroBlaze currently does not have a memory-management unit (MMU) in hardware or any memory management support in the basic libraries, as a hardware MMU would utilize a great deal more hardware resources. Malloc already works properly in that it allocates memory while memory is available and returns NULL if all memory is used. Free is very system-specific and is only an indication to the memory management subsystems that a given set of memory is no longer needed in the given program. It is up to the system to implement the actual functionality of free. Often, a memory management subsystem does not free memory right away, but rather only performs a freed memory sweep when malloc indicates that the available memory has fallen below a predetermined lower boundary. Only then will the memory manager actually de-allocate all memory that was pre-marked for freeing by calls to free. For code size reasons, free simply does nothing for now. The only safe way to write code that works on any system is to make sure that what malloc returns is greater than NULL. For the purposes of this thesis these memory allocation issues, taking into account the extensive use of malloc() functions in the JasPer software implementation, were simply solved by greatly increasing program heap size. However, when building a small project from scratch, it is wise to allocate memory only once and reuse this allocated memory as often as possible.

Issue #3 – The "xil_io_out32" and "xil_io_in32" bugs

If a peripheral is created by using the CIP Wizard in XPS and "user logic software registers" are enabled, then the "undefined reference to xil_io_out32" and "undefined reference to xil_io_in32" errors occur when the automatically created drivers compile.

This is a bug that can be solved by manually changing the function call from xil_io_out32 to XIo_Out32 and from xil_io_in32 to XIo_In32. These changes have to be done in the "…/microblaze_0/include/your_peripheral_name.h" header file. This problem has already been fixed in ISE 12.4 software.

Issue #4 – Xilkernel OS and sleep() functions

For the purposes of this thesis, the Xilkernel OS was used, as it provided functionalities that were critical in order to estimate the speed-up of the JPEG2000 decoder after the co-design and implementation on Virtex-5 SX. More specifically, Xilkernel provides functions that can count the number of clock cycles needed in order for a function or number of functions to finish their execution. However, during the software development phase it was also decided to implement the XMODEM file transfer protocol that would secure an error resilient transmission of the test images via the UART RS232 serial port. This could not be accomplished for this software design, because the sleep functions that are essential for the XMODEM protocol could not function properly, unless they were called within threads. The JasPer software could not allow such changes. The commenting on this issue is that if the software application or project needs sleep functions in order to work properly, then Xilkernel could be avoided by running uLinux on Microblaze instead.

Bibliography

[1] JPEG200 Final Committee Draft Version 1.0, ISO/IEC JTC 1/SC 29 WG 1 (ITU-T SG8).

[2] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi, "The JPEG2000 Still Image Coding System: An overview", IEEE Transactions on Consumer Electronics, Vol. 46, No. 4, pp. 1103-1127, November 2000

[3] M.D. Adams and F. Kossentini, "JasPer: A software-based JPEG-2000 Codec implementation", in *Proc. IEEE Int. Conf. Image Processing,* Vancouver, Canada, Sept. 2000, vol. II, pp. 53-56.

[4] M.D. Adams and F. Kossentini, "Reversible integer-to-integer wavelet transforms for image compression: Performance evaluation and analysis", IEEE Trans. Image Processing, vol. 9, pp. 1010-1024, June 2000.

[5] M.D. Adams, "Reversible wavelet transforms and their application to embedded image compression," M.S. thesis, Univ. Victoria, Canada, 1998. Available http://www.ece.ubc.ca/mdadams/.

[6] Michael Yaw Appiah, "An Efficient FPGA Implementation of High-Speed JPEG-2000 Encoder and Decoder", M.Sc.Eng thesis, Aalborg University, Aalborg, 2006.

[7] Roger Woods, John McAllister, Gaye Lightboy, and Ying Yi, "FPGA-based Implementation of Signal Processing Systems", ISBN: 978-0-470-03009-7.

[8] T. Acharya and P. S. Tsai. JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures. John Wiley & Sons, Hoboken, New Jersey, 2004.

[9] S. Mallat, "A theory for multiresolution signal decomposition: The Wavelet representation," IEEE Trans. Pattern Analysis And Machine Intelligence, Vol. 11, no. 7, pp.674-693, July 1989.

[10] W. Sweldens, \The lifting scheme: A custom-design construction of biorthogonal wavelets," Applied and Computational Harmonic Analysis, Vol. 3, no. 15, pp.186-200, 1996.

[11] M.S. Bhuyan, Md. Azrul Hasni Madesa, Masuri Othman, and Shabiul Islam, "FPGA realization of Inverse Discrete Wavelet Transform", IEICE Electronics Express, Vol.6, No.6, 277-282.

[12] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Applications*, 2nd ed. Norwell, MA: Kluwer, 1997.

[13] M. Boliek, J. Scott Houchin, and G. Wu, “JPEG 2000 next generation image compression system features and syntax,” in Proc. IEEE Int. Conf. Image Processing, Vancouver, Canada, Sept. 2000, vol. II, pp. 45-48.

[14] Xilinx, EDK Concepts, Tools, and Techniques: A Hands-On Guide to Effective Embedded System Design.

[15] Xilinx, Embedded Systems Tool Reference Manual, EDK 12.1.

[16] Xilinx, Microblaze Processor Reference Guide.

[17] http://www.xilinx.com/

[18] http://www.ece.uvic.ca/~mdadams/jasper/

[19] http://www.kakadusoftware.com/

[20] http://www.fpgadeveloper.com/