TECHNICAL UNIVERSITY OF CRETE

DEPARTMENT OF ELECTRONIC & COMPUTER ENGINEERING

Diploma Thesis:

Implementation and Evaluation of Dynamic Partial Reconfiguration Techniques on Cryptographic Algorithms

Kasampalis Vasileios

Advisor: Professor Pnevmatikatos Dionisios

Evaluation Committee:

Professor Pnevmatikatos Dionisios

Professor Dollas Apostolos

Assistant Professor Koutroulis Eftichios

Chania, December 2014

Acknowledgments

I would like to thank my advisor, Professor Dionisios Pnevmatikatos, for giving me the opportunity to work on this field of research for the purpose of this thesis, and for his support and valuable help. I would also like to thank Professor Apostolos Dollas and Associate Professor Ioannis Papaefstathiou who agree to evaluate my diploma thesis.

I would like to thank Dr. Papadimitriou for his interest and support. His help was very important to me especially when I started working on this thesis. I would like to thank Anargiros Ilias and Haralampos Vatsolakis for their help to my thesis. I would like to thank Markos Kimionis for providing me all the equipment I needed for my thesis.

Finally, I would like to thank all my friends for all these great moments we have been through during my studies and my family for their help, support and understanding during all these years as a student.

Abstract

In recent years the advantages of reconfigurable computing have make FPGAs important parts of many applications. One interesting characteristic of FPGAs is their ability to change parts of the design that runs on them dynamically. This procedure is called Partial Reconfiguration (PR). One disadvantage of PR is that sometimes it takes too much time to be completed and for real-time applications it has to be able to be executed fast. The purpose of this thesis is the implementation of designs that can perform PR with high throughput.

When a reconfiguration is taken place a partial bitstream file must be transferred from a memory where it is stored to the reconfiguration memory of the FPGA. This transfer can be executed by software code that runs on a processor (e.g. PowerPC or MicroBlaze) or by hardware. For this thesis several designs were implemented, each one uses one of these methods to transfer the partial bitstream and a deferent memory where the partial bitstreams are stored. The memories that were used on these designs were a Compact Flash, a DDR2 SDRAM and a SRAM. The final result was a design that can perform PR with high throughput.

Contents

Chapter 1

Introduction

This chapter is an introduction about FPGAs and dynamic partial reconfiguration. It also presents the contribution of this thesis and the structure of this report.

1.1 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are integrated circuits that can be configured by the user. This characteristic gives them the ability to execute many different logic functions. The FPGA configuration is generally specified using a hardware description language, like VHDL or Verilog. Any logical function that could be performed by an application-specific integrated circuit (ASIC) can be also implemented on a FPGA. FPGAs however have the advantage of being able to be reconfigured, fully or partially, which gives them the flexibility of a software application with the performance of a hardware application.

FPGAs contain programmable logic components called "logic blocks". Logic blocks can be connected with each other via a hierarchy of reconfigurable interconnectors and can be configured to perform logic functions from simple logic gates like AND or XOR to complex combinational functions. Logic blogs in most FPGAs also include memory elements like flip-flops or more complete blocks of memory. Some FPGAs also contain embedded microprocessors (like the PowerPC in various Xilinx FPGAs) which allow even more flexibility. Figure 1.1 shows the generic structure of an FPGA.

FPGAs are used in various applications like digital signal processing, software-defined radio, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation and radio astronomy.

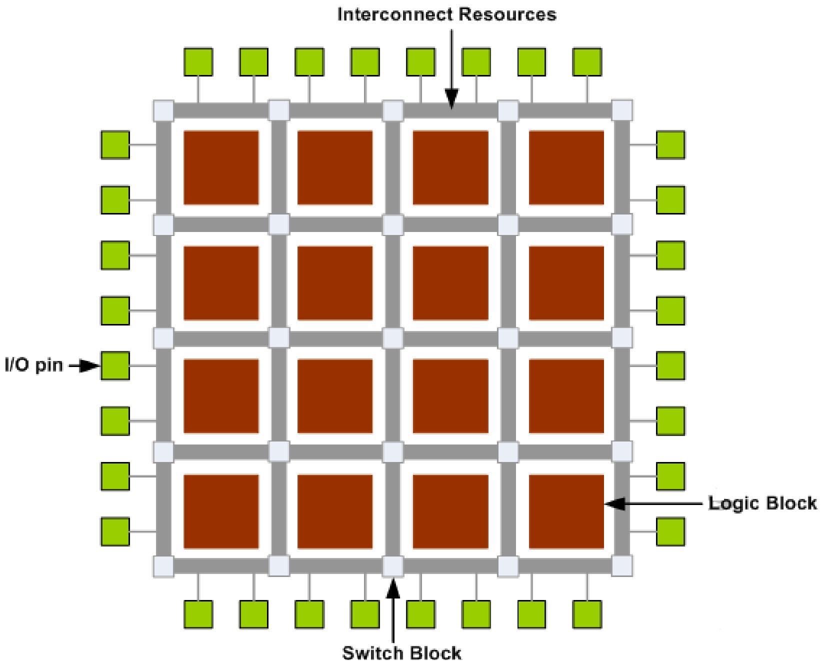


Figure 1.1 Generic internal structure of a FPGA

1.2 Xilinx Virtex-5 XC5VLX110T FPGA [18]

The FPGA that was used for this thesis is the Virtex-5 XC5VLX110T. The basic element of this FPGA is the Configurable Logic Block (CLB). A CLB is made up of two slices. Each slice is equivalent and contains:

- Four 6-input LUTs
- Four flip-flops
- Arithmetic logic gates
- Large multiplexers
- Fast carry look-ahead chain

Other elements of this FPGA are DSP48E slices, memory blocks, Clock Management Tiles (CMTs) and I/O blocks. Each DSP48E slice contains a 25 x 18 multiplier, an adder and an accumulator. Each CMT contains two Digital Clock Manager (DCM) blocks and one PLL block.

The FPGA contains a special memory block the configuration memory. When a new design is downloaded on the FPGA the bitstream file that has the information about the programming of the logic elements and their interconnections is stored in this memory.

Configuration memory is arranged in frames that are tiled about the device. These frames are the smallest addressable segments of the configuration memory space. The configuration memory of the Virtex-5 XC5VLC110T FPGA has 23,712 configuration frames, and the length of each frame is 41 32-bit words.

1.3 Partial Reconfiguration

One important characteristic of FPGAs is that a part of the design that runs on them can be changed while the rest remain the same. This procedure is called Partial Reconfiguration (PR). Partial reconfiguration can be performed on an FPGA either when it is not active, which is known as static PR, or when it is active, which is called dynamic PR. Some of the benefits of PR are the following:

- **Increased System Performance.** Sometimes there are different versions of a part of a design, each one more preferable based on different situations. PR allows only the optimal version of this part to run on the FPGA each time while the rest of the design remains unaffected and so the design can have better performance.
- **Reduced Power Consumption.** PR can be used to replace parts of a design with blank beatstreams when they are not needed which reduce the power consumption of the design. Lower power consumption can also be achieved by using optimal versions of the reconfigurable parts of the design.
- **Adaptability.** PR allows the system to adapt to changes in the environment they work, their input data or their mission specifications.
- **Self-Test and Fault Tolerance.** A design can have self-test components that check its integrity. When a fault is detected PR allows the reconfiguration of just the part responsible of this fault and thus the reconfiguration of the entire design is prevented.

Xilinx FPGAs allows two deferent styles of partial reconfiguration, which are described in the following paragraphs.

1.3.1 Module-Based Partial Reconfiguration

Module-Based PR permits to reconfigure specified regions of the FPGA. These regions are called Partially Reconfigured Regions (PRR). For each PRR there are several designs that can run on it, but only one runs on it each time. These designs are called Reconfigurable Modules (RM). The rest of the design that is not changed is the static design. For each RM there is a Partial Bitstream file which is used when a PRR is reconfigured to implement this RM. For each PRR there is also a Blank Bitstream file. When a Blank Bitstream runs on a PRR, this region is inactive. For the connection of the static design with the PRRs special buses had to be used which are called bus macros. Bus macros were provided by Xilinx and they provided a guaranteed connection point between the static design and the PRR. Bus macros are not used anymore because they have been replaced by PROXY LUTs. A PROXY LUT is a 1-point LUT that accomplishes the same thing as the bus macros. PROXY LUTs are also automatically inserted into the design by the design tool during the generation of the bitstream files. During the FPGA initial configuration a specific RM runs on each PRR. These RMs are determined by the designer during the generation of the bitstream file of the complete design.

1.3.2 Difference-Based Partial Reconfiguration

Difference-based PR can be used when a small change is made to the design. It is especially useful in case of changing LUT equations or dedicated memory blocks content. The partial bitstream contains only information about differences between the current design structure (that runs on the FPGA) and the new content of an FPGA. There are two ways of performing difference-based reconfiguration known as frond-end and back-end. Frond-end is based on the modification of the design in the hardware description language. It is clear that such a solution requires full repeating of the synthesis and implementation process. The back-end permits to make changes at the implementation stage of the prototyping flow. Therefore there is no need for re-synthesis of the design. The usage of both methods leads to creation of a partial bitstream that can be used for a partial reconfiguration of the FPGA.

1.4 Contribution

In every design that uses partial reconfiguration there is a unit that performs the reconfigurations whenever they are needed. This thesis focuses on this unit and on how it can perform PR faster regardless the application that it is used on. On this thesis several designs of this unit were implemented. The purpose of this thesis is to evaluate each design and with each design to achieve a higher reconfiguration throughput. The final design can be used on any application that uses PR.

1.5 Thesis Structure

Chapter 2 provides a background on Partial Reconfiguration and references to related work. Chapter 3 describes the architecture of the designs that were implemented. Chapter 4 describes the evaluation and the software of the designs, and the verification method. Chapter 5 presents the experimental results. Chapter 6 describes some conclusions regarding this work and provides ideas for future work.

Chapter 2

Background and Related Work

This chapter presents a background on Partial Reconfiguration and describes the reconfiguration tools that were used for this thesis. Finally it presents several projects related to Partial Reconfiguration.

2.1 Methods to Perform Partial Reconfiguration

There are several different ways to perform partial reconfiguration. In each way the design has a memory were the partial bitstraems are stored. This memory is called storage memory. Every design that uses PR has a component that decides when a reconfiguration must be performed and performs it. This component is the reconfiguration controller. One method to perform PR is the external reconfiguration. With this method the storage memory and the reconfiguration controller are not parts of the design that runs on the FPGA. The partial bitstreams are sorted in an external PC which also is the reconfiguration controller. The partial bitstreams can be transferred to the configuration memory through JTAG. External reconfiguration can also be achieved through SPI or SelectMAP interfaces. Another method is the internal reconfiguration. With this method the reconfiguration controller is part of the design that runs on the FPGA. It transfers the partial bitstreams from an external memory to the configuration memory through the Internal Configuration Access Port (ICAP).

2.2 Internal Configuration Access Port [19]

The Internal Configuration Access Port is a hardcore that is used for internal partial reconfiguration. It allows for writing to or reading from the configuration memory of the FPGA. Its input width for the Virtex-5 FPGAs is up to 32 bits and the maximum frequency of operation is 100MHz. Figure 2.1 shows interface if the ICAP and its ports are explained at Table 2.1.

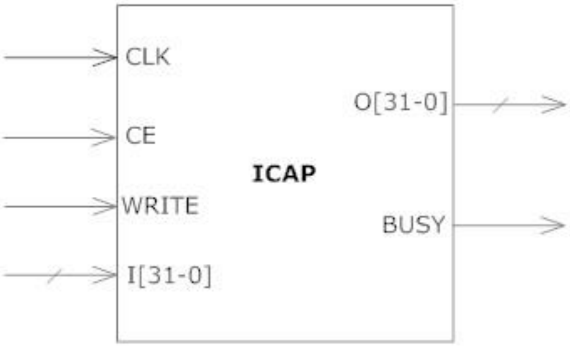


Figure 2.1 Input and Output Signals of the ICAP

Pin Name	Type	Description
CLK	Input	ICAP interface clock
CE	Input	Active-Low ICAP interface select. 0 = ICAP data bus enabled 1 = ICAP data bus disabled
WRITE	Input	0 = WRITE 1 = READ WRITE input can only be changed while CE is set to 1, otherwise an abort occurs.
I[31:0]	Input	ICAP write data bus.
O[31:0]	Output	Unregistered ICAP read data bus.
BUSY	Output	Active-High busy status. Only used in read operations. BUSY remains Low during writes.

Table 2.1 ICAP Interface Pin Table

Figure 2.2 shows an example of a non-continuous data transfer to the ICAP. The numbered points of this figure are explained below:

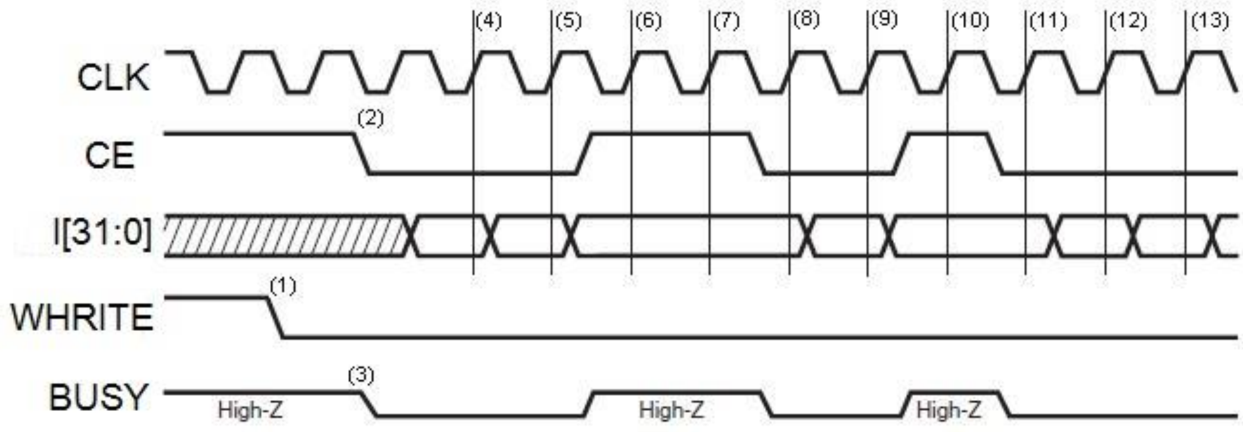


Figure 2.2 Time Diagram of a Non-Continuous ICAP Data Loading

1. WHRITE is driven Low by the user in order to enable the ICAP write data bus. WHRITE can be tied Low if read back is not needed.
2. The user asserts CE Low, enabling the ICAP data bus.
3. BUSY goes Low shortly after CE is asserted.
4. A word is loaded on the rising CLK edge.
5. A word is loaded on the rising CLK edge.
6. The user deasserts CE, and the word is ignored.
7. The user deasserts CE, and the word is ignored.
8. A word is loaded on the rising CLK edge.
9. A word is loaded on the rising CLK edge.
10. The user deasserts CE, and the word is ignored.
11. A word is loaded on the rising CLK edge.
12. A word is loaded on the rising CLK edge.
13. A word is loaded on the rising CLK edge.

2.3 Partial Reconfiguration Design Tools

For the design and implementation of partially reconfigurable systems the following Xilinx Tools were used:

EDK 9.1.02 and 12.3: This tool allows the design of a processor system, a system that contains a MicroBlaze processor and several processor peripherals that can communicate with it through a local bus (OPB or PLB depending on the tool's version).

ISE 9.1.02 and 12.3: This tool was used for the design of components that are not parts of the processor system and for the synthesis of the static designs and the RMs.

PlanAhead 10.1 and 12.3: This tool allows the placement of the PRRs on the FPGA and uses the netlist files that were generated from the synthesis to generate the initial and the partial bitstream files.

The reason why two different versions of each tools were used for this thesis is because the newer versions, which allow for better designs, need a special license to support partial reconfiguration, and this license became available later after the work for this thesis had started.

2.4 Related Work

B. Griese [1] developed a Real-Time Reconfiguration Manager; a hardware component that is used as reconfiguration controller for external PR. C. Clause [2] developed an ICAP Controller on a Virtex-II Pro FPGA. It is connected to the PLB and equipped with DMA capabilities. With this controller he achieved throughput almost 20 times better than the throughput of the OPBHWICAP controller which is provided by Xilinx. This controller was later redesigned and used on both Virtex-II Pro and Virtex-4 FPGAs [3]. I. Gelado [4] used partial reconfiguration on hardware accelerators for software applications. When a specific accelerator is needed it is loaded into the reconfigurable logic. P. Sedcole [5] suggested a new style of module-based PR, the Merge Partial Reconfiguration. On a design that uses this style, when a partial bitstream is loaded it is not written directly to the configuration memory, but instead the current configuration is read back from the device and modified with information from the partial bitstream before being written back. K. Papademetriou [6] suggested the prefetching of PRRs when they are not used with parts of RMs that are about to be used in the near future. By this way he managed to reduce reconfiguration time up to 87%. In [7] he developed a simple design that performs PR on a Virtex-II Pro FPGA with a Compact Flash as storage memory and the PowerPC as the reconfiguration controller. In [8] he split the total reconfiguration time in several parts and presented the time results of each one. In [9] he developed a mathematical model that calculates the total reconfiguration time of a design based on several parameters. M. French [10] developed an autonomous partially reconfigurable signal processing system. This system uses cognitive algorithms to modify and tune signal processing in real-time using active PR. P. Manet [11] developed a system that uses PR for signal and image processing. The reconfiguration controller of this system implements a DMA that transfers the bitstream from the storage memory to the ICAP. This controller is almost 84 times faster than the OPBHWICAP. S. Liu [12] suggested two techniques to reduce the reconfiguration time. The first technique is the use of fully streaming DMA engines. A Master DMA engine was added in the ICAP controller and a Slave DMA engine was added in the SRAM controller which is the interface of the storage memory. Those DMA engines communicate through a FIFO and transfer the partial bitstreams directly from the SRAM to the ICAP. The second technique is the reduction of each bitstream size by using a simple encoding algorithm. M. Liu [13] developed several reconfiguration controllers on a Virtex-4 FPGA and compared them with the OPBHWICAP and the XPSHWICAP that are provided by Xilinx. The first controller is the DMA_HWICAP which is the XPSHWICAP with a DMA controller attached on it. The DMA controller contains both a master and a slave bus interface. The slave interfaced is used to receive commands like the source address, the destination address and the length of a transaction. The master interface is used to initiate the transfer of the bitstream from the storage memory to the ICAP. The second controller is the MST_HWICAP. This controller does not use the HWICAP. Instead of DMA it contains an integrated master bus interface with burst transmission support. This interface is directly connected to the controller of the storage memory. It also contains a slave bus interface to receive control commands which is connected to the host PLB. The final controller is the BRAM_HWICAP. This controller contains a BRAM which is used as the storage memory and is large enough to hold the entire partial bitstream. Figure 2.3 shows the block diagram of the MST_HWICAP and the BRAM_HWICAP. K. Vipin [26] developed an ICAP Controller that can achieve high reconfiguration throughput on a Virtex 6 FPGA. H. Kashyap [27] proposed an approach to secure dynamic partial reconfiguration when an unsecure external memory is used as the storage memory. This approach uses the AES algorithm to encrypt and decrypt the partial bitstreams. T.D.A. Nguyen [28] developed a Partially Reconfigurable Heterogeneous System-on-Chip. This system uses dynamic partial reconfiguration to support multiple processors and hardware accelerators. A. Morales-Villanueva [29] described a method of saving or restoring the state of each RM when it is replaced or loaded on the PRR. E. Cetin [30] proposed a framework for implementing FPGA circuits that can recover from configuration memory errors within a desired maximum recovery period by using PR. C. Effraimidis [14] developed an autonomous genetic algorithm system that supports the change of the fitness function at run time by using dynamic PR. This system can theoretically support infinite fitness functions. G. Nikoloudakis [15] developed and evaluated a cryptography system that uses PR to change the cryptographic algorithm that runs on it each time. E. Spanakis [16] developed a Linux-based Task Manager for dynamic reconfiguration. This Task Manager runs on a PowerPC of a Virtex-II Pro FPGA and controls the reconfiguration of the PRRs. A. Ilias [17] designed a system that uses partial reconfiguration to repair parts of it when they do not work correctly because of the reversal of one or several bits in the configuration memory. Such reversals can be caused by several reasons like the exposure of the chip to radiation. When the system detects such faults it does not reconfigure the entire design but only the problematic part of it.

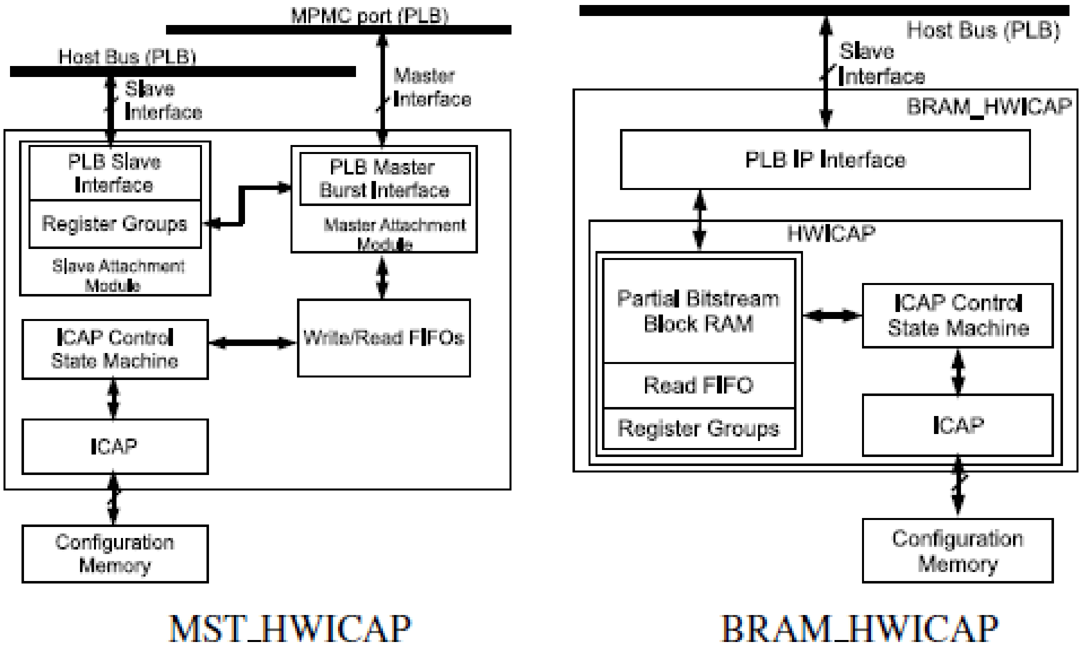


Figure 2.3 MST_HWICAP and BRAM_HWICAP Block Diagrams.

Chapter 3

System Designs and Architectures

This chapter describes a generic Partially Reconfigurable system and every system that was designed in order to achieve high reconfiguration throughput on a Virtex-5 FPGA. On the first designs the bus that is used for communication with the MicroBlaze is the OPB because this was the only bus supported by the older versions of the Xilinx tools for Virtex-5. On the later designs where the newer versions were used the OPB was replaced by the faster PLB bus.

3.1 A Generic Partially Reconfigurable System

Figure 3.1 shows a generic Partially Reconfigurable design. This design consists of the static and the reconfigurable part. The reconfigurable part is the part that can be changed and it contains the PRRs. The static part is the rest of the design that cannot be changed. The PR Support unit is a part of the static design that executes every reconfiguration. It contains the storage memory, the reconfiguration controller and the ICAP. The reconfiguration controller can be the MicroBlaze and so it will use software code to execute the reconfigurations or a complete hardware unit. The selection of the storage memory is also an important factor for the performance of the PR support unit.

3.2 OPB System version 1

On the first design the reconfiguration controller is the MicroBlaze. When a reconfiguration is taken place, MicroBlaze reads the partial bitstream from the storage memory and transfers it to the ICAP. The storage memory is a Compact Flash memory. Figure 3.2 shows the top diagram of the design. The DCM unit provides the clock signal for the Processor System and the PRR. The clock frequency for this design is 100 MHz. The ROPB bus is a bus similar to the OPB that is used for the communication between the PRR and the MicroBlaze.

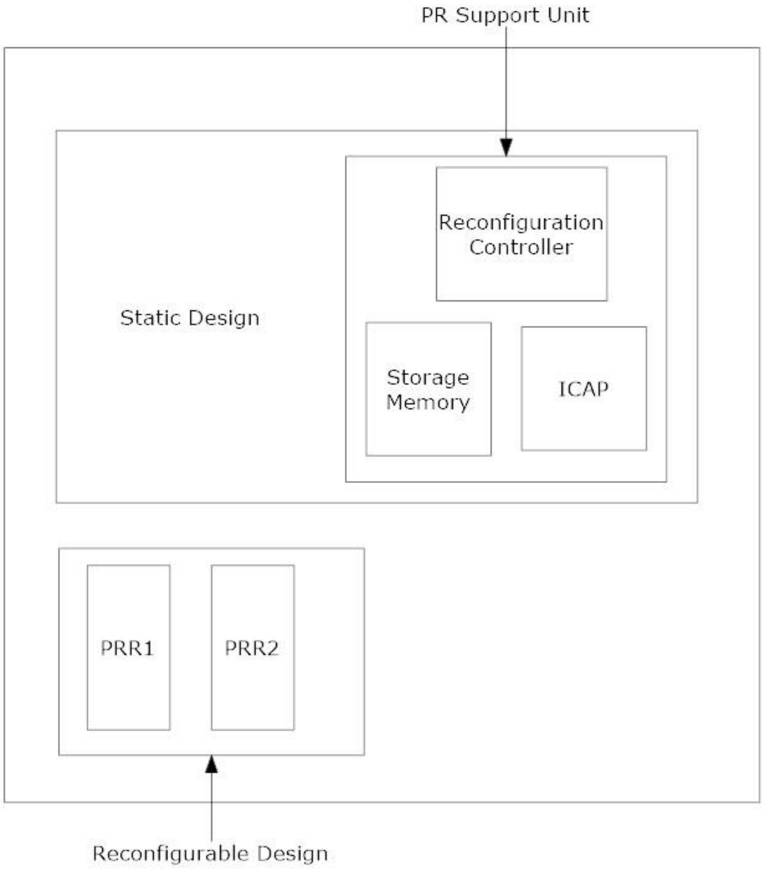


Figure 3.1 Top Diagram of a Generic Partially Reconfigurable Design

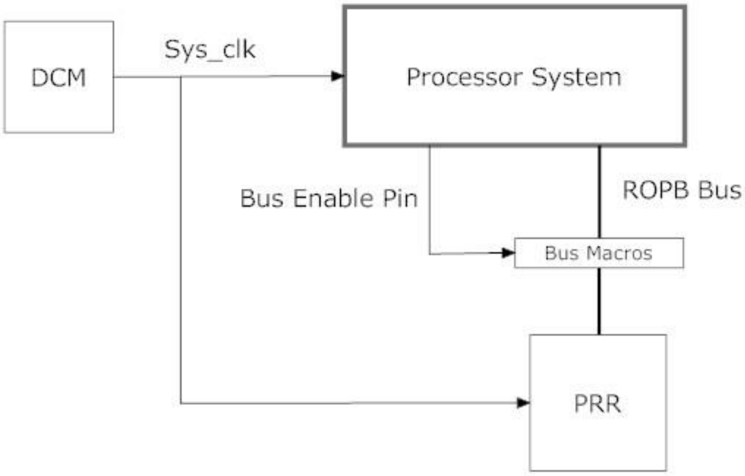


Figure 3.2 Top Diagram of the OPB System version 1

The Processor System is a component that contains the MicroBlaze and all the peripherals that are connected with it through the OPB bus. The block diagram of the Processor System is shown at figure 3.3.

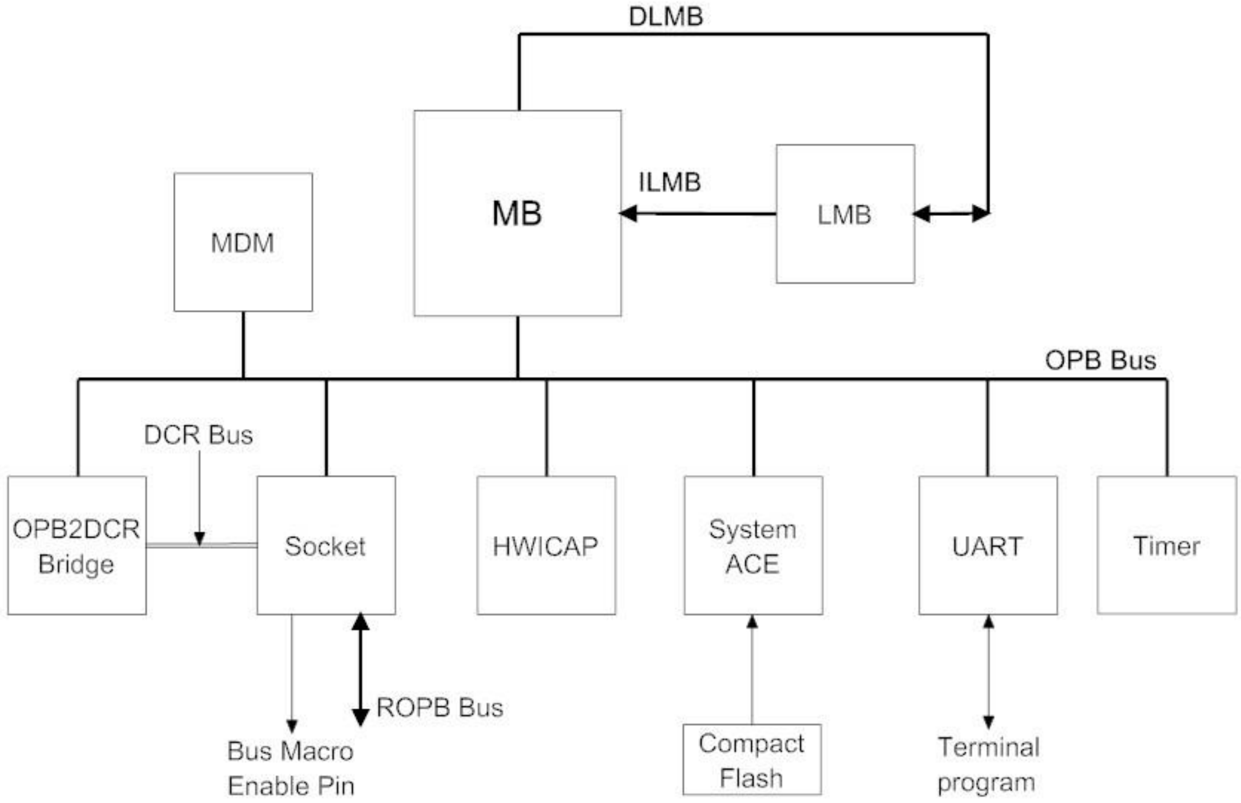


Figure 3.3 Block Diagram of the Processor System of the OPB System version1

This system consists of the following components:

1. The MicroBlaze processor. It uses 64 Kbytes of the Bram (LMB) as its main data and instruction memory.
2. The UART peripheral which allows communication between the MicroBlaze and a host PC through an RS232 port. It is used to send data to the RM and read back the results to check if the PRR was reconfigured correctly, to request a new reconfiguration or to receive results regarding with reconfiguration times.
3. The Timer is used for producing time results for each face of the reconfigurations.

4. The SystemACE peripheral is used for the support of an external compact flash memory. This memory contains the partial bitstreams and the initial bitstream which is used to reconfigure the FPGA when the system starts.
5. The Socket is used so we can connect an external OPB peripheral with the OPB bus through the registers of the DCR bus. The external peripheral that is connected to the Socket is the algorithm that runs on the PRR.
6. The Microprocessor Debug Module (MDM) can be used for debugging the MicroBlaze software code.
7. The final component is the OPB HWICAP [20]. The OPB HWICAP is an OPB peripheral provided by Xilinx that enables an embedded microprocessor, such as the MicroBaze, to read and write the FPGA configuration memory through the ICAP. The version of this peripheral that is used in this work is the v1.00.b. Figure 3.4 shows the top-level block diagram of the OPB HWICAP. The DPRAM is a 2KB BRAM where the data from the partial bitstream are stored before they are transferred to the ICAP. The ICAP Control State Machine controls the data transfers from the bus to the BRAM and from that memory to the ICAP.

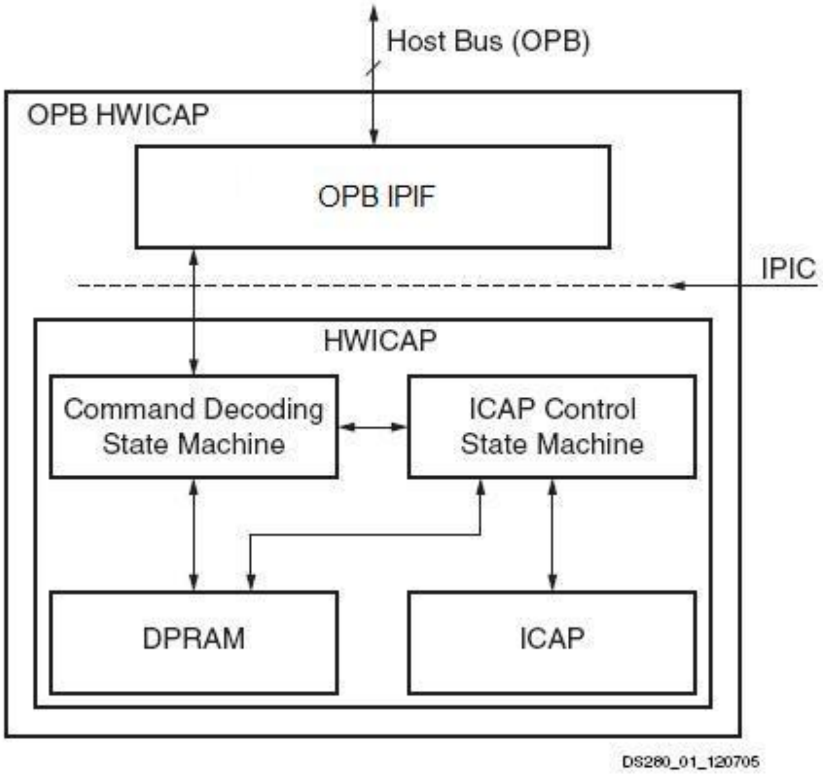


Figure 3.4 OPB HWICAP Block Diagram

3.3 OPB System version 2

The main problem of the first system was the Compact Flash Memory. This memory is too slow and affects negatively the reconfiguration performance of the design. To improve the performance Compact Flash was replaced by a DDR2 SDRAM which is much faster and more efficient. Another advantage of the DDR2 is that it can be used as data cache memory for the MicroBlaze, which allows for even better performance. The controller of the DDR2 is the Multi-CHannel OPB Double Data Rate 2 Synchronous DRAM Controller (mch_opb_ddr2) v1.02.a which is provided by Xilinx [21]. The size of the DDR2 is 256MB. The System Ace peripheral however was not removed because the Compact Flash is needed for the transfer of the partial bitstreams to the DDR2. When the system starts the partial bitstreams are copied from the Compact Flash to the DDR2 and then the Compact Flash is not used again. The processor system also has a second DCM module that provides some additional clock signals that are needed for the DDR2 controller. Except of the addition of the DDR2 controller and the second DCM the rest of the design is the same as at the previous version.

3.4 OPB System version 3

Another way to improve the performance even more is to remove the software from the process of the reconfiguration. To achieve this, a DMA peripheral was used to transfer the bitstreams from the DDR2 directly to the ICAP. In this architecture the HWICAP is not used because the MicroBlaze does not participate in the data transfer from the storage memory to the ICAP. When a reconfiguration is taken place the DMA peripheral transfers the bitstream from the DDR2 to a 2K FIFO and then it is transferred from there to the ICAP.

3.4.1 The ICAP Controller

For this design a new component was implemented, the ICAP controller, that contains the ICAP and a F.S.M. which controls the transfer of the bitstream from the FIFO to the ICAP. The block diagram of this component is shown at figure 3.5.

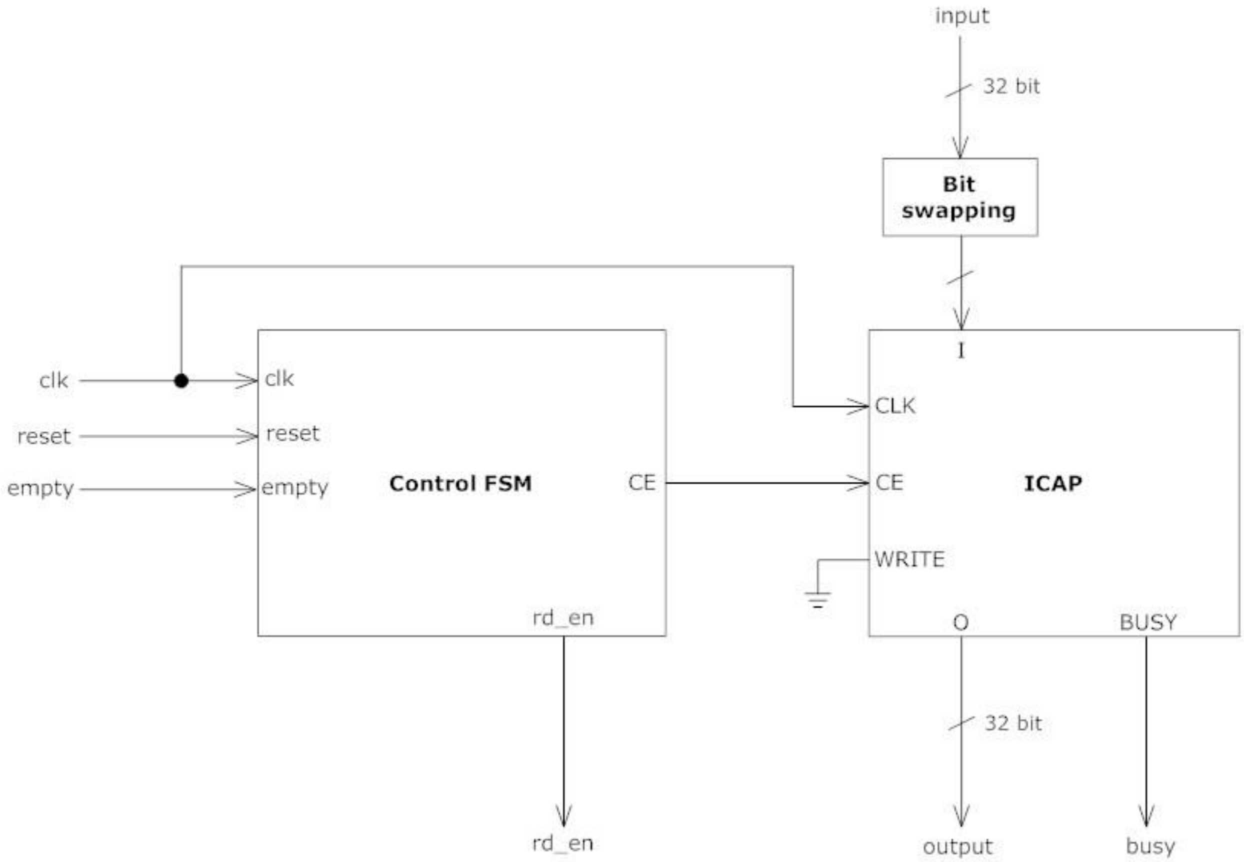


Figure 3.5 Block Diagram of the ICAP Controller

The first part of the ICAP Controller is the Bit swapping unit. When a 32 bit word is transferred from the bitstream to the ICAP the bits of every byte must be swapped. For example if the input word is 0xAA995566 then its bytes must be changed as shown in table 3.1 and so this word will be changed to 0x5599AA66. The Bit swapping unit rearranges the bits of every word of the bitstream.

Bitstream Format	0xAA (10101010)	0x99 (10011001)	0x55 (01010101)	0x66 (01100110)
Bit swapped	0x55 (01010101)	0x99 (10011001)	0xAA (10101010)	0x66 (01100110)

Table 3.1 Bit Swapping

The second part of the ICAP Controller is the Control FSM. This unit controls the transfer of the bitstream from the FIFO to the ICAP. The only thing that this F.S.M. needs to know is if there are data in the FIFO and so its input signals are the clock, the reset and the empty signal which comes from the FIFO. Its output signals are the rd_en signal which goes to the FIFO and the CE which goes to the ICAP. The Control FSM has five states. Figure 3.6 shows the state diagram of the Control FSM and table 3.2 shows the values of each output signal of the F.S.M. for each state.

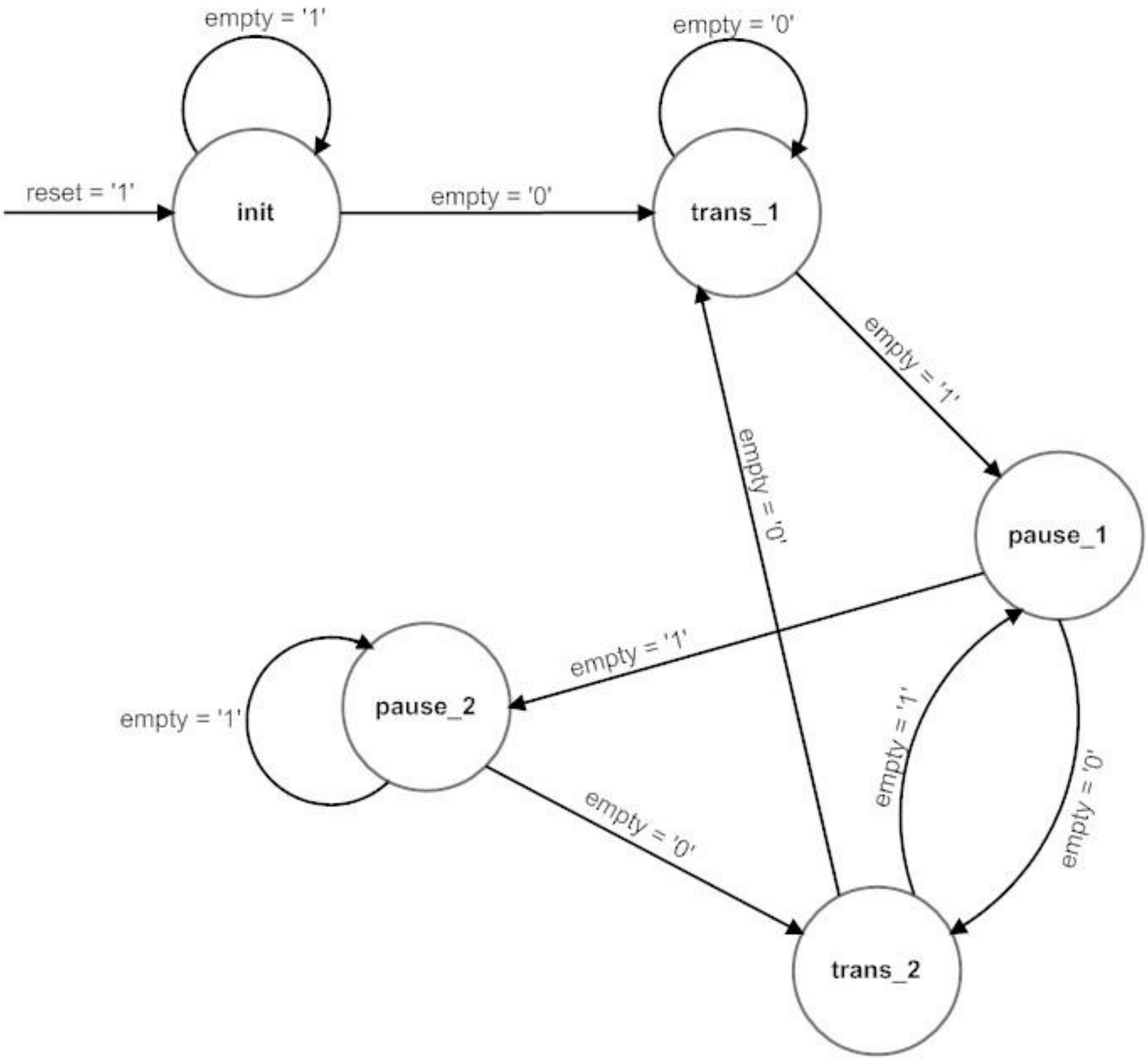


Figure 3.6 State Diagram of ICAP Controller's FSM

State	rd_en	CE
init	0	1
transfer_1	1	0
transfer_2	1	1
pause_1	0	0
pause_2	0	1

Table 3.2 Control FSM output values

Because the ICAP and the FIFO reads and sends data respectively on the rising edge of the clock, to avoid synchronization problems the Control FSM change states on the falling edge of the clock and so the rd_en and CE signals are not changed on a rising edge. When the F.S.M. is reset it goes to the “init” state where rd_en is set to 0 and CE to 1 and so no transfer is taken place. As long as the FIFO is empty it stays on that state. When the empty signal becomes 0 the F.S.M. goes to state “transfer_1”. There rd_en is set to 1 and CE to 0 and so on the next rising edge of the clock the first word of the bitstream will be read from the FIFO. These bits however will be written to the ICAP on the following rising edge of the clock and so the CE signal must remain 0 for at least one more cycle. As long as there are data in the FIFO the F.S.M. remains on this state and on each rising edge a new word is read from the FIFO and the previous is written to the ICAP. When the empty signal becomes 1 the F.S.M. goes to state “pause_1”. There the rd_en signal is set to 0 because there are not any new data in the FIFO. The CE signal remains 0 because the last word from the FIFO has not been written yet to the ICAP and so the ICAP data bus must remain enabled for on more cycle. If the empty signal remains 1 on the next cycle the F.S.M. goes to state “pause_2” where the CE signal is 1 because the last word that has been read from the FIFO has been transferred to the ICAP and there are no new data. As long as the empty signal remains 1 the F.S.M remains on state “pause_2”. According to the example of paragraph 1.5 when the CE signal is asserted Low the ICAP loads the word of the write data bus on the next rising edge of the clock except of the first word which is loaded on the second rising edge. Because of this when the F.S.M. is on one of the “pause” states and the empty signal is 0 it cannot go to “transfer_1” state because on the next rising edge the CE signal would be 0 and wrong data would be loaded to the ICAP. The CE signal must remain 1 for at least one cycle and so the F.S.M. goes to “transfer_2” state where the CE signal is set to 1. This is also the reason why “init” and “pause_2” are different states. If the empty signal remains 0 the F.S.M. goes to the “transfer_1” state where the CE signal is set to 0 and the data start to be loaded to the ICAP. If the empty signal is 1 the F.S.M. goes to “pause_1” state just like when it is on “transfer_1” state. Because of the way Control FSM works it must have been reset before every reconfiguration.

The final part of the ICAP Controller is the ICAP which is connected to the Control FSM as shown at figure 3.4. In our design the O and BUSY outputs of the ICAP are not used and so the WHRITE input is always set to 0.

3.4.2 The FIFO Peripheral

The FIFO peripheral is an OPB peripheral which contains a 2K FIFO which is connected to the OPB bus through an OPB IP Interface. The block diagram of the FIFO Peripheral is shown at figure 3.7. The fsm_reset, empty, and dout outputs are connected at the respective inputs of the ICAP Controller. The fsm_reset signal is set through the OPB bus by the MicroBlaze.

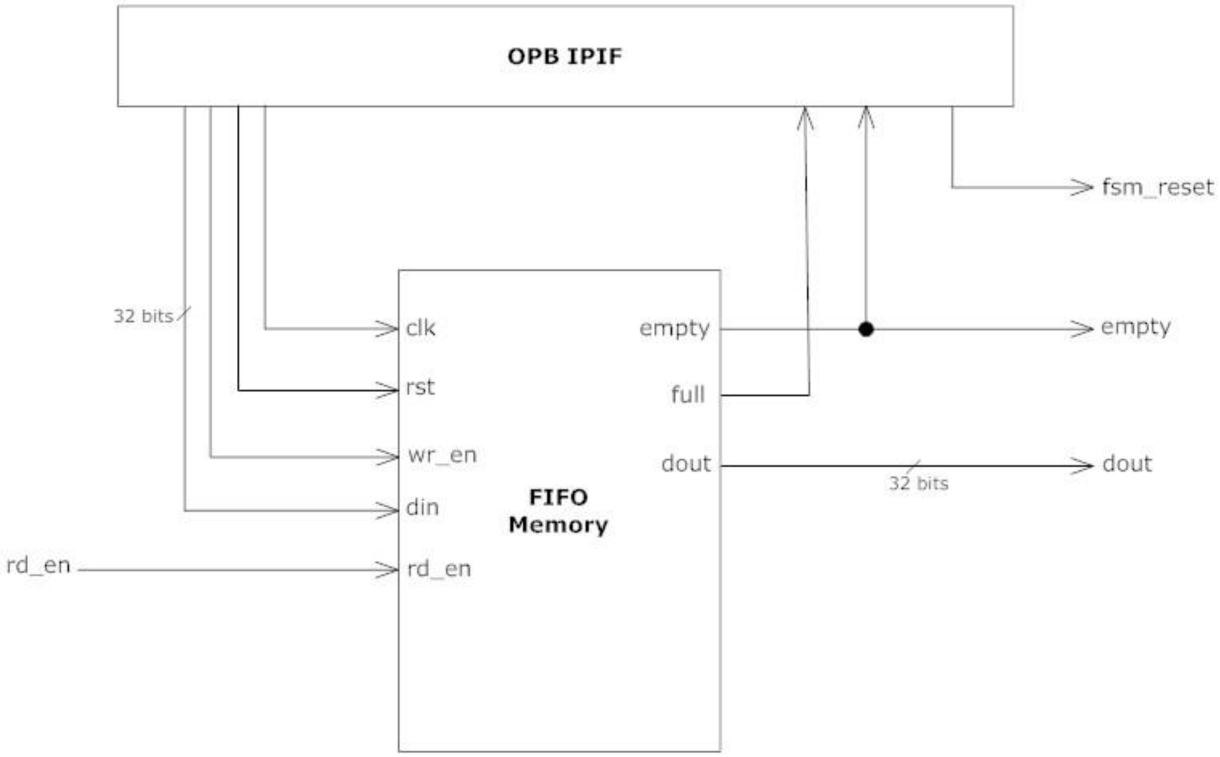


Figure 3.7 Block Diagram of the FIFO Peripheral

3.4.3 The DMA System

Figures 3.8 and 3.9 show the block diagrams of the design and the Processor System respectively. There we can see that the HWICAP has been removed because the MicroBlaze does not need to communicate with the ICAP on this design. The DMA peripheral is the opb_central_dma v1.00.c and it is provided by Xilinx [22].

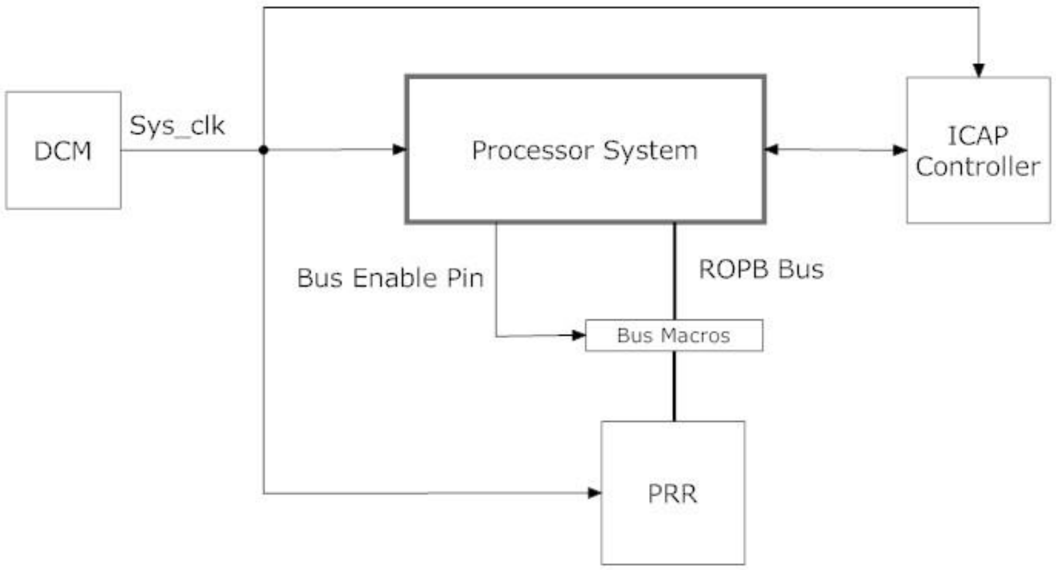


Figure 3.8 Top Diagram of the OPB System version 3

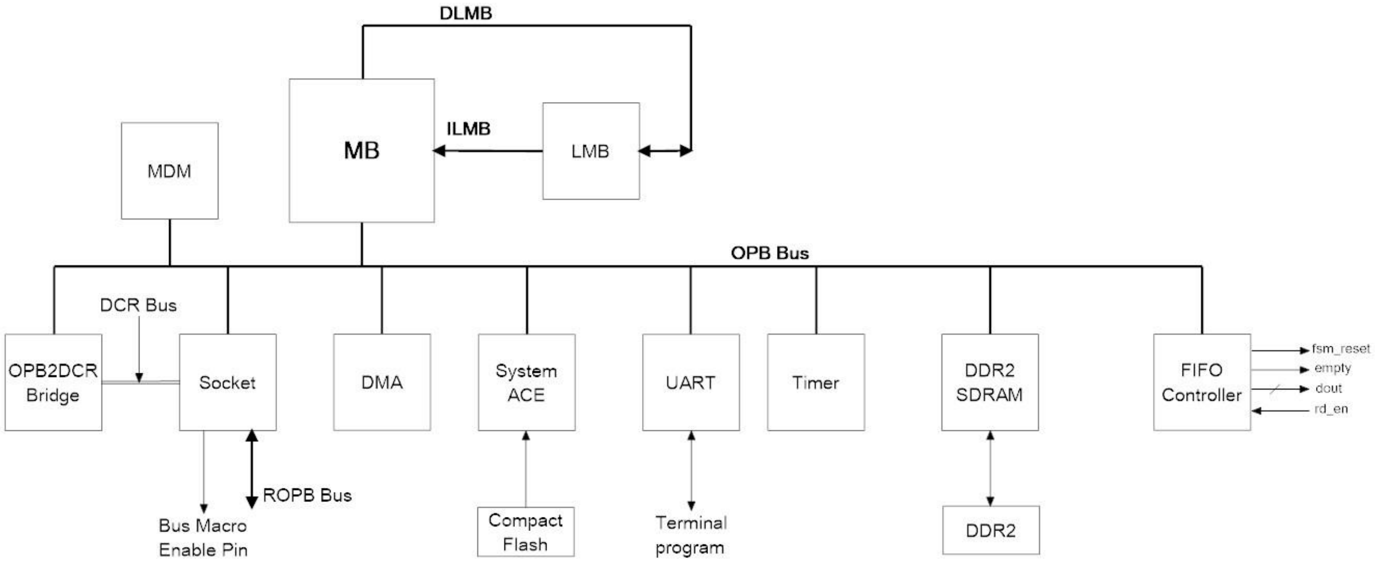


Figure 3.9 Block Diagram of the Processor System of the OPB System version3

3.5 The PLB Systems

The newer versions of the Xilinx tools support a better and faster bus than the OPB, the PLB bus. Every design that had been implemented on the oldest version of Xilinx tools was redesigned and implemented on the newer version. These new designs use the PLB as the bus for the communication between the MicroBlaze and the peripherals but there are some other differences from the oldest designs. Firstly, as it was mentioned in chapter 1, the Bus Macros have been replaced by PROXY LUTs that are inserted by the tools. Secondly the DCR bus is not supported by the later tools and so the Socket peripheral is not used. In these designs each RM is connected directly to the PLB like any other PLB peripheral. By this way the bus interfaces of the RMs are parts of the static design. Finally the peripheral that is used for the communication between the MicroBlaze and the ICAP is now the XPS HWICAP [23]. Figure 3.10 shows the block diagram of this new core. As shown in this figure, the XPS HWICAP does not contain a BRAM but instead it has the write and read FIFOs. Incoming data is stored within the write FIFO, from where it can be fed to the ICAP. The XPS HWICAP also provides for read back of configuration resource states. In this case, the frames are read back into the read FIFO one at a time and the processor will then be able to read the frame data directly from the read FIFO.

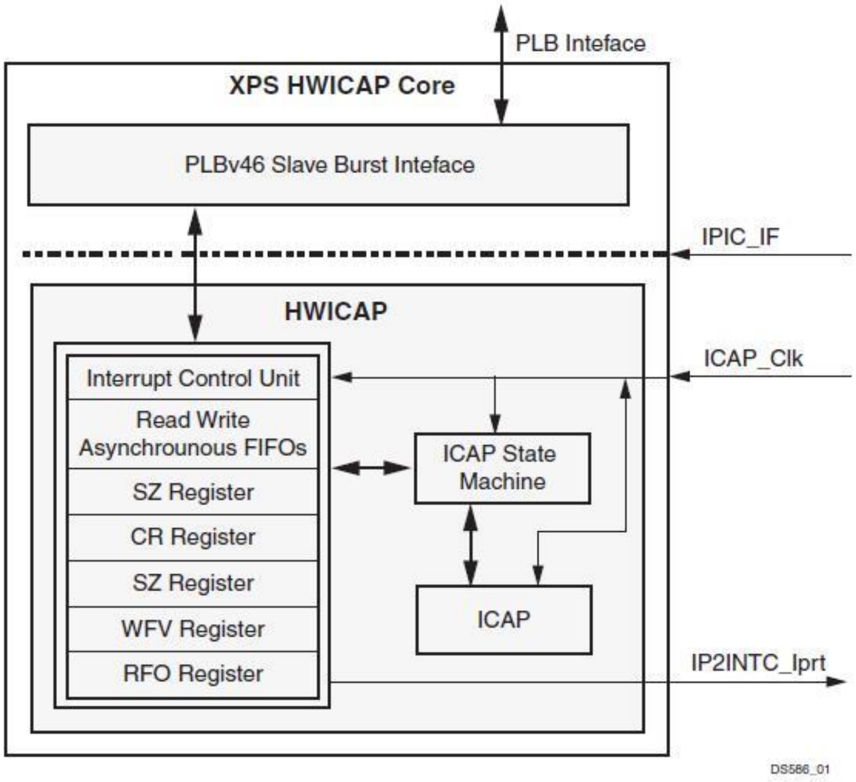


Figure 3.10 XPS HWICAP Block Diagram

Figure 3.11 shows the top diagram of the first PLB design that uses the compact flash as storage memory. The Clock Generator is a new component that uses the DCMs to produce clock signals. The clock frequency for this design is 125MHz except for the HWICAP that runs on 100MHz which is the maximum frequency that Xilinx suggests for the ICAP.

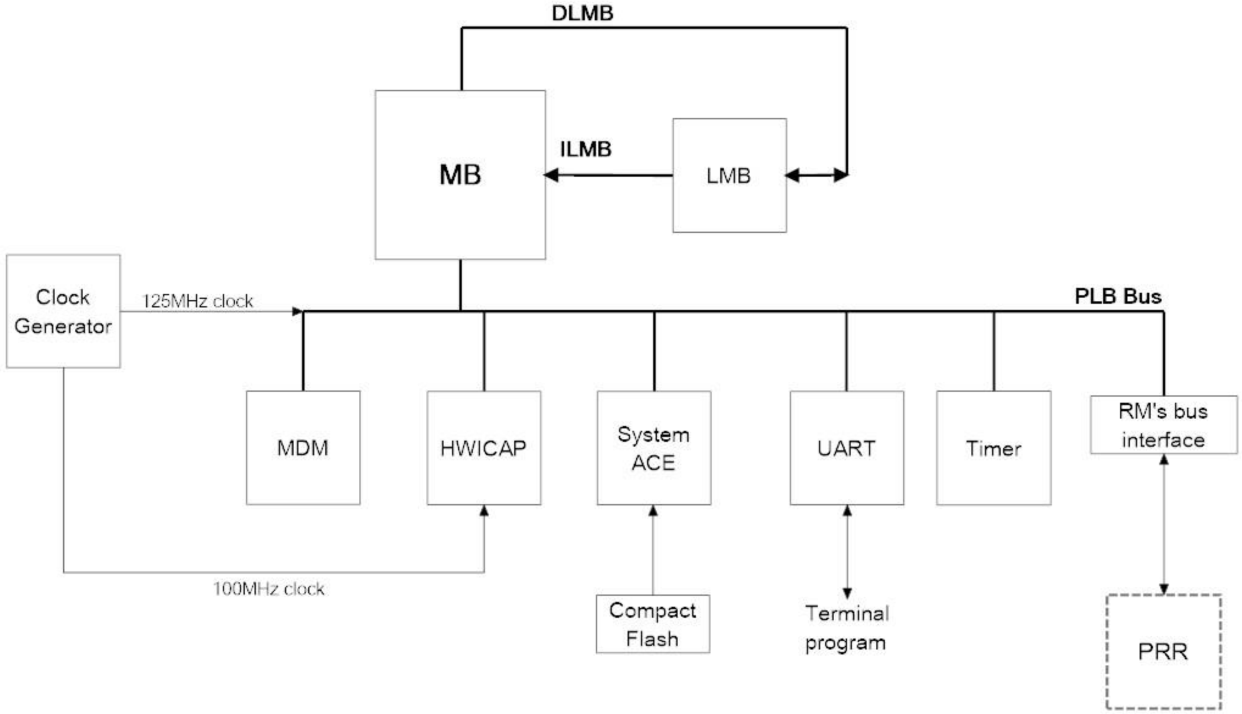


Figure 3.11 Top Diagram of the PLB System version 1

The second design uses the DDR2 as the storage memory. The new controller for this memory that Xilinx provides is the Multi-Port Memory Controller (MPMC) [24]. The third design uses the DMA peripheral to transfer the bitstreams from the DDR2 to the ICAP. In this design a new FIFO peripheral was design that can be connected to the PLB bus but its functionality is the same as the OPB peripheral. The ICAP controller did not need to be changed and so the same component was used on this design. In this design the ICAP Controller and the FIFO should run on the same frequency otherwise there would be synchronization problems and the design would not work correctly. The FIFO however for similar reason should run on the same frequency with the bus and so for the whole design the clock frequency was 100MHz. Finally two more designs were implemented that are the same as the last two designs with the difference that they use an SRAM as the storage memory. The SRAM if faster than the DDR2 but its size is only 1MB and so it can be used only for designs with less or smaller partial bitstreams. The controller of the SRAM that was used for these designs is the XPS Multi-CHannel External Memory Controller (xps_mch_emc) v3.01.a which is provided by Xilinx [26].

All the designs that were in this chapter are summarized in table 3.3:

System Version	Storage Memory	Reconfiguration Controller
OPB Version 1	Compact Flash	OPB HWICAP
OPB Version 2	DDR2	OPB HWICAP
OPB Version 3	DDR2	DMA with ICAP Controller
PLB Version 1	Compact Flash	XPS HWICAP
PLB Version 2	DDR2	XPS HWICAP
PLB Version 3	DDR2	DMA with ICAP Controller
PLB Version 4	SRAM	XPS HWICAP
PLB Version 5	SRAM	DMA with ICAP Controller

Table 3.3 Designs Summary

Chapter 4

Evaluation of each design

This chapter describes the evaluation of each design and presents the resources that each design requires.

4.1 Resource Requirements and Component Placements

4.1.1 OPB Designs

The first part of the evaluation was to synthesize each part of the designs. The static part and the RMs had to be synthesized separately. After the synthesis the netlist files that were generated were used by the PlanAhead tool to generate the bitstream files. Each RM in these designs consists of two parts, the cryptographic algorithm and the OPB interface. The interface was produced by the EDK tool and so it was synthesized separately from the algorithm. Table 4.1 shows the resources that each algorithm and each interface requires and the total resources that each RM requires. Table 4.2 shows the resources that the static part of each OPB design requires.

Component	Number of Slice Registers Used	Slice Registers Utilization	Number of Slice LUTs Used	Slice LUTs Utilization
AES	709	1.03%	1646	2.38%
AES IPIF	371	0.54%	224	0.32%
AES Total	1080	1.56%	1870	2.71%
Blowfish	91	0.13%	214	0.31%
Blowfish IPIF	323	0.47%	302	0.44%
Blowfish Total	414	0.6%	516	0.75%

Table 4.1 Resource Requirements of each RM for the OPB designs

OPB Design	Number of Slice Registers Used	Slice Registers Utilization	Number of Slice LUTs Used	Slice LUTs Utilization
Version 1	2426	3.51%	2784	4.03%
Version 2	4671	6.76%	5119	7.41%
Version 3	4752	6.88%	5085	7.36%

Table 4.2 Resource Requirements of the static parts of each OPB design

After the synthesis of each design the PlanAhead tool was used to determine the area of the FPGA that will be used for the PRR. As we can see from Table 4.1 both RMs does not require a big part of the FPGA and so the area of the PRR does not need to be very large. Some designs, however did not work with specific selections of the PRR area and so this area is not the same in every design. The PlanAhead tool was also used to place the bus macros and the DCM which should be placed manually. Figure 4.1 shows the area selection for the PRR of the first design and figure 4.2 shows the placement of the DCM.

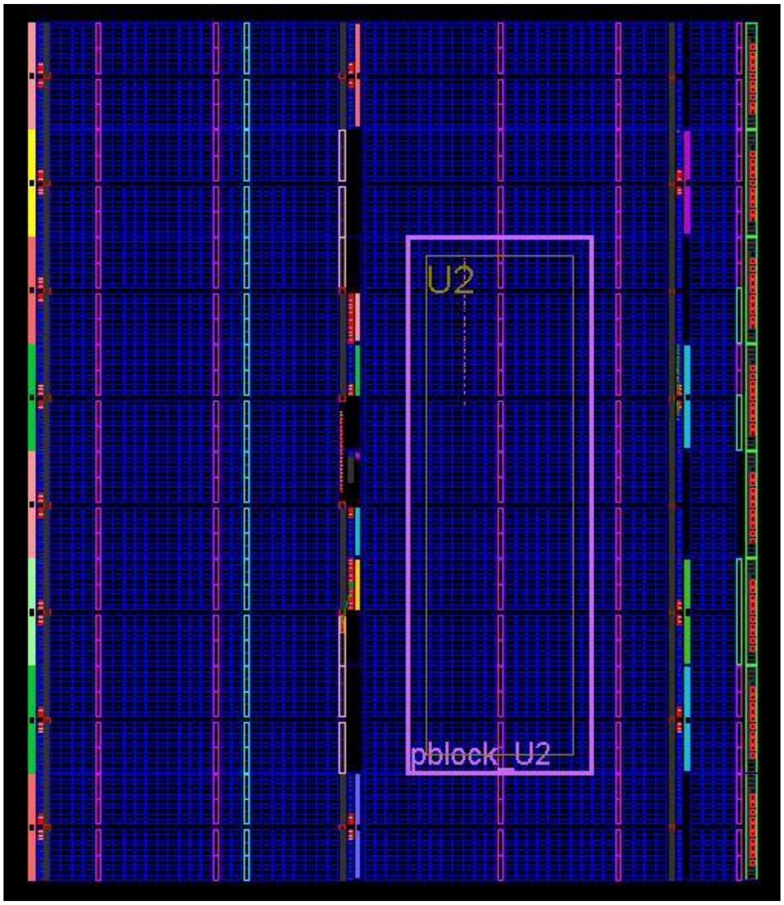


Figure 4.1 PRR Placement of the OPB System version 1

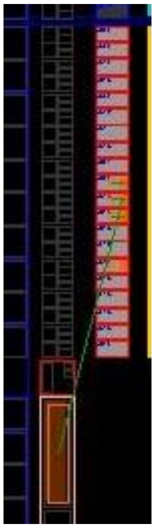


Figure 4.2 DCM Placement of the OPB System version 1

4.1.2 PLB Designs

In the PLB designs the bus interfaces of each RM is part of the static design and so the RMs contain only the algorithms. The algorithms were synthesized again, this time with the newer tools. Tables 4.3 and 4.4 show the resource requirements of the algorithms and the static designs respectively.

Component	Number of Slice Registers Used	Slice Registers Utilization	Number of Slice LUTs Used	Slice LUTs Utilization
AES	713	1.03%	1628	2.36%
Blowfish	107	0.15%	212	0.31%

Table 4.3 Resource Requirements of each RM for the PLB designs

PLB Design	Number of Slice Registers Used	Slice Registers Utilization	Number of Slice LUTs Used	Slice LUTs Utilization
Version 1	3739	5.41%	3771	5.46%
Version 2	7351	10.64%	6350	9.19%
Version 3	7550	10.92%	6776	9.8%
Version 4	4730	6.84%	4776	6.91%
Version 5	4389	6.35%	4410	6.38%

Table 4.4 Resource Requirements of the static parts of each PLB design

After the synthesis of each design the PlanAhead tool was used for the generation of the bitstream files. On the newer version the DCMs that are used by the Clock Generator are placed automatically by the tool but the area of the PRR must be determined by the user. The MPMC peripheral has a parameter that determines the locations where three of its components must be placed. This parameter is called C_IDELAYCTRL_LOC and is set be the EDK tool when the MPMC is used. The PlanAhead tool however cannot read this parameter and so these components were placed manually based on the value of this parameter.

4.2 Processor Software

For every design a software code was developed to control the reconfiguration. For the System ACE, each HWICAP and each DMA peripherals Xilinx provides libraries that were used in these codes in order to use these peripherals. Table 4.5 describes the functions that were used for each of these libraries. As shown in this table the OPB HWICAP provides a function that transfers only 4 bytes to its BRAM and a different function that transfers data from this memory to the ICAP. The PLB HWICAP however provides one function that allows the user to determine the amount of data that are about to be transferred and it also executes the transfer of these data to the ICAP from the write FIFO.

Peripheral	Libraries	Functions	Description
System ACE	xsysace.h xsysace_l.h sysace_stdio.h	sysace_fopen()	Opens a bitstream file.
		sysace_fread()	Transfers data from the CF to MicroBlaze's memory.
		sysace_fclose()	Closes a bitstream file.
OPB HWICAP	xhwicap.h xhwicap_i.h xhwicap_l.h	XHwIcap_Initialize()	Initializes a HWICAP instance.
		XHwIcap_StorageBufferWrite()	Transfers a 4byte word from the memory of the MicroBlaze to the BRAM of the HWICAP.
		XHwIcap_DeviceWrite()	Transfers a specified amount of data from the BRAM of the HWICAP to the ICAP device.

XPS HWICAP	xhwicap.h xhwicap_i.h xhwicap_l.h	XHwIcap_LookupConfig()	Looks up the device configuration based on the unique device ID.
		XHwIcap_CfgInitialize()	Initializes a HWICAP instance.
		XHwIcap_DeviceWrite()	Writes data to the Write FIFO and starts the transfer of the data to the ICAP device.
DMA	xdmacentral.h xdmacentral_l.h	XDmaCentralInitialize()	Initializes a specific DMA instance.
		XDmaCentralReset()	Forces a software reset to occur in the device.
		XDmaCentralSetControl()	Sets the contents of DMA Control register.
		XDmaCentralTransfer()	Start the DMA transferring data from a memory source to a memory destination
		XDmaCentralGetStatus()	Get the contents of DMA Control register.

Table 4.5 Xilinx Software Drivers

The first few bytes of a partial bitstream file are the bitstream's header. The header contains useful information like the size of the bitstream and it must not be transferred to the ICAP. At the designs that do not use the Compact Flash as the storage memory when the bitstreams are copied to the storage memory (DDR2 or SRAM) the headers are skipped and the size of each bitstream is saved into variables. In every design, except those with the DMA, when a reconfiguration is taken place MicroBlaze runs a loop where a specific amount of data are transferred to the ICAP in every iteration. When this loop is completed if there are any remaining data they are transferred to the ICAP as well. Before this loop starts the number of its iteration is calculated based on the size of the bitstream file, as well as the remaining bytes of the bitstream. In the designs that use the DMA, before a DMA transfer starts a register of the DMA peripheral must be set. This register is called Control Register and determines the size of each data transfer and if the DMA must increment the source or the destination addresses with each transfer. The DMA peripheral also has a read only register, the Status Register that contains information like if the DMA is busy or if there is a bus error. When this register shows that the DMA is not busy and there are no errors and the FIFO is empty, the reconfiguration has been completed. Figures 4.3 and 4.4 show the flow diagrams of the reconfiguration for each OPB designs. Figure 4.5 shows the flow diagrams of the reconfiguration for the PLB designs. The diagram for the design that uses the DMA is the same as the OPB design.

4.3 System Verification

To verify that each design works properly a system was implemented for each one of them. These systems use a switch to determine which algorithm will run in the PRR. They were also connected to a terminal through a RS232 to send data to the algorithm and return the results so we can know that the PRR has been reconfigured correctly. When a system is downloaded on the FPGA the AES algorithm runs in the PRR. First MicroBlaze waits the terminal for the first input for the algorithm. When these data are sent to MicroBlaze it transfers them to the algorithm and when the algorithm returns the results, MicroBlaze sends them to the terminal and checks the switch. If the switch has changed it performs the reconfiguration and then asks for the input of the other algorithm. Each time MicroBlaze returns results to the terminal checks the switch and if it is needed performs a reconfiguration.

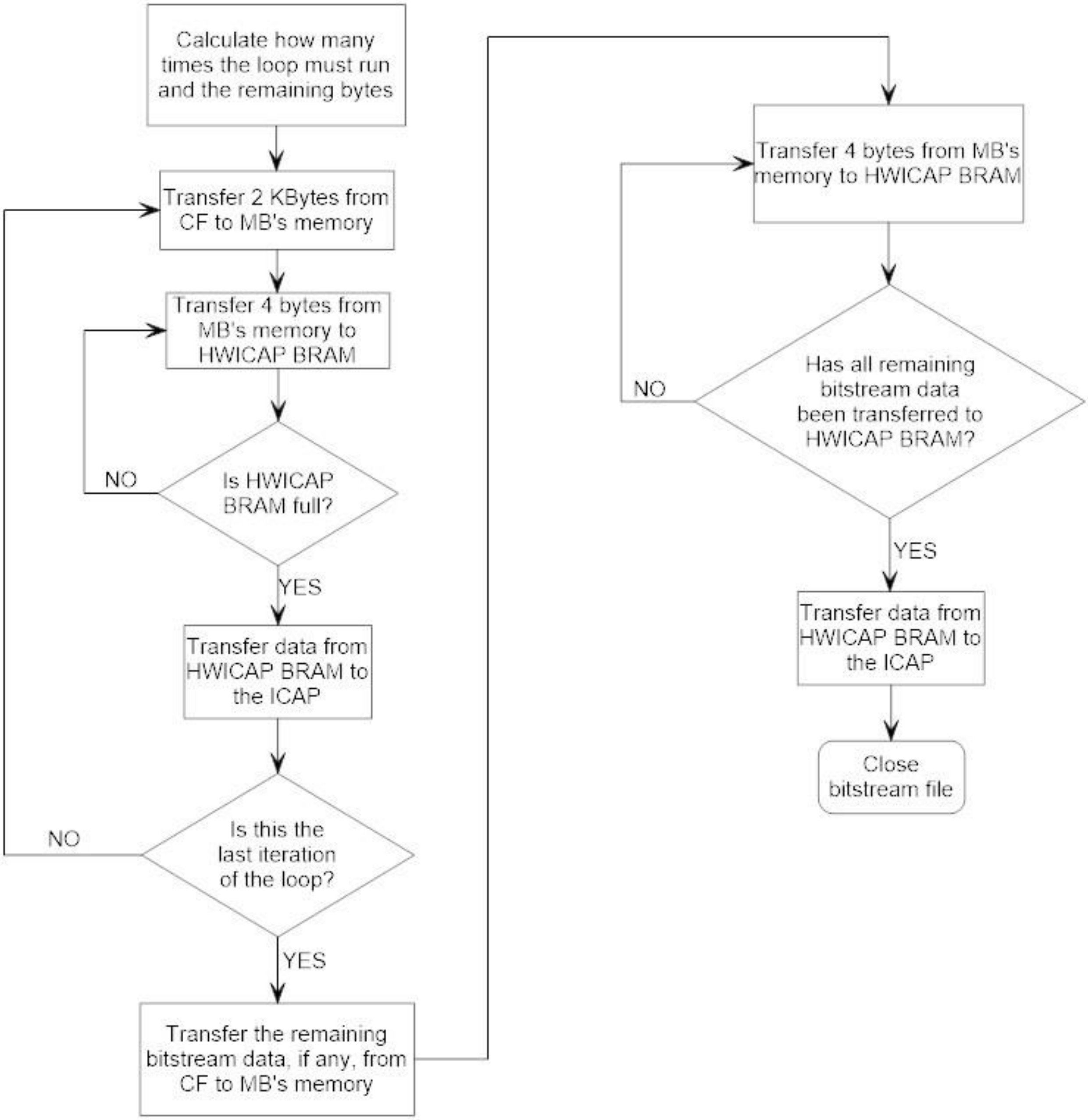
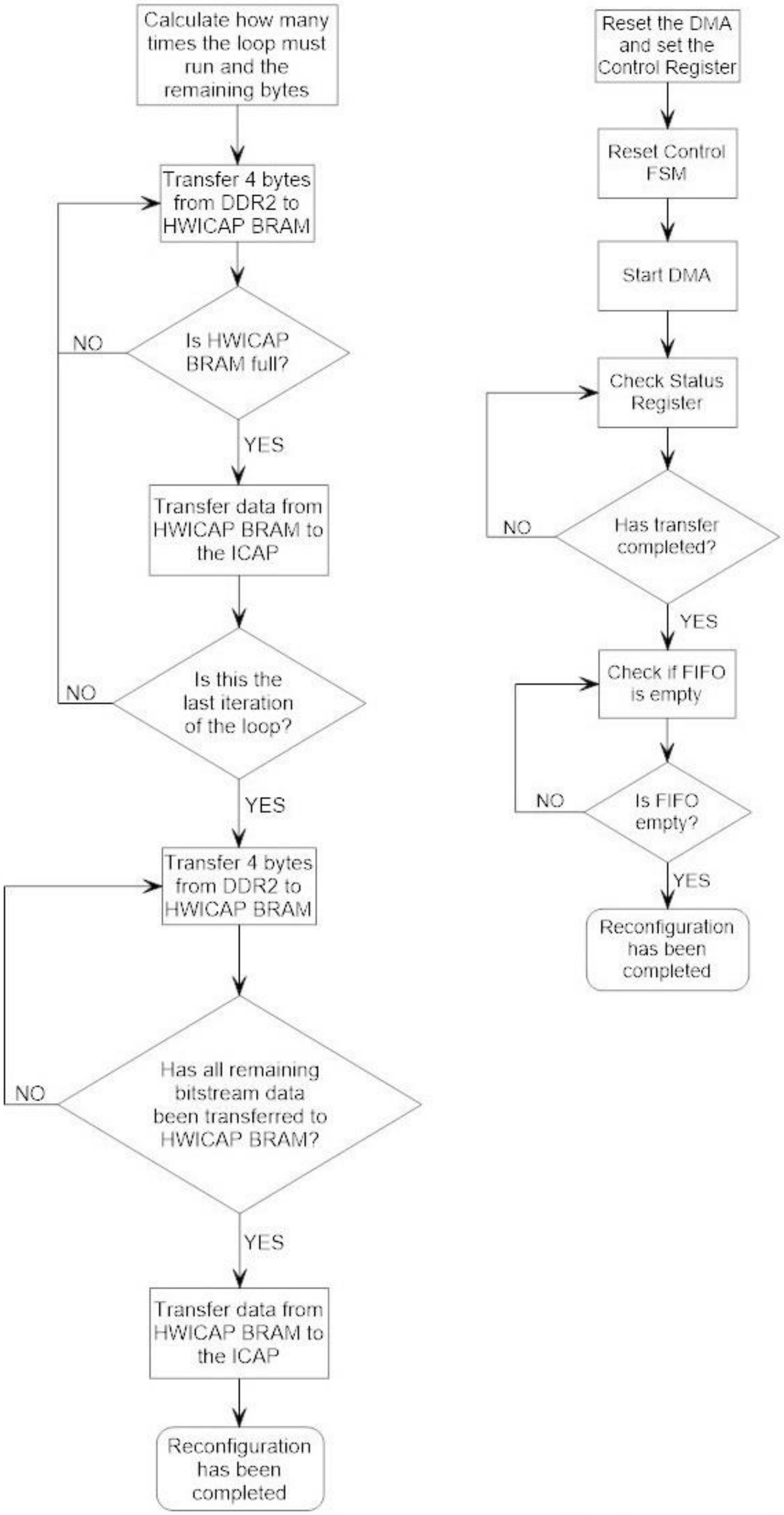

Calculate how many times the loop must run and the remaining bytes
Transfer 2 KBytes from CF to MB's memory
Transfer 4 bytes from MB's memory to HWICAP BRAM
Is HWICAP BRAM full?
NO
YES
Transfer data from HWICAP BRAM to the ICAP
Is this the last iteration of the loop?
NO
YES
Transfer the remaining bitstream data, if any, from CF to MB's memory
Transfer 4 bytes from MB's memory to HWICAP BRAM
Has all remaining bitstream data been transferred to HWICAP BRAM?
NO
YES
Transfer data from HWICAP BRAM to the ICAP
Close bitstream file

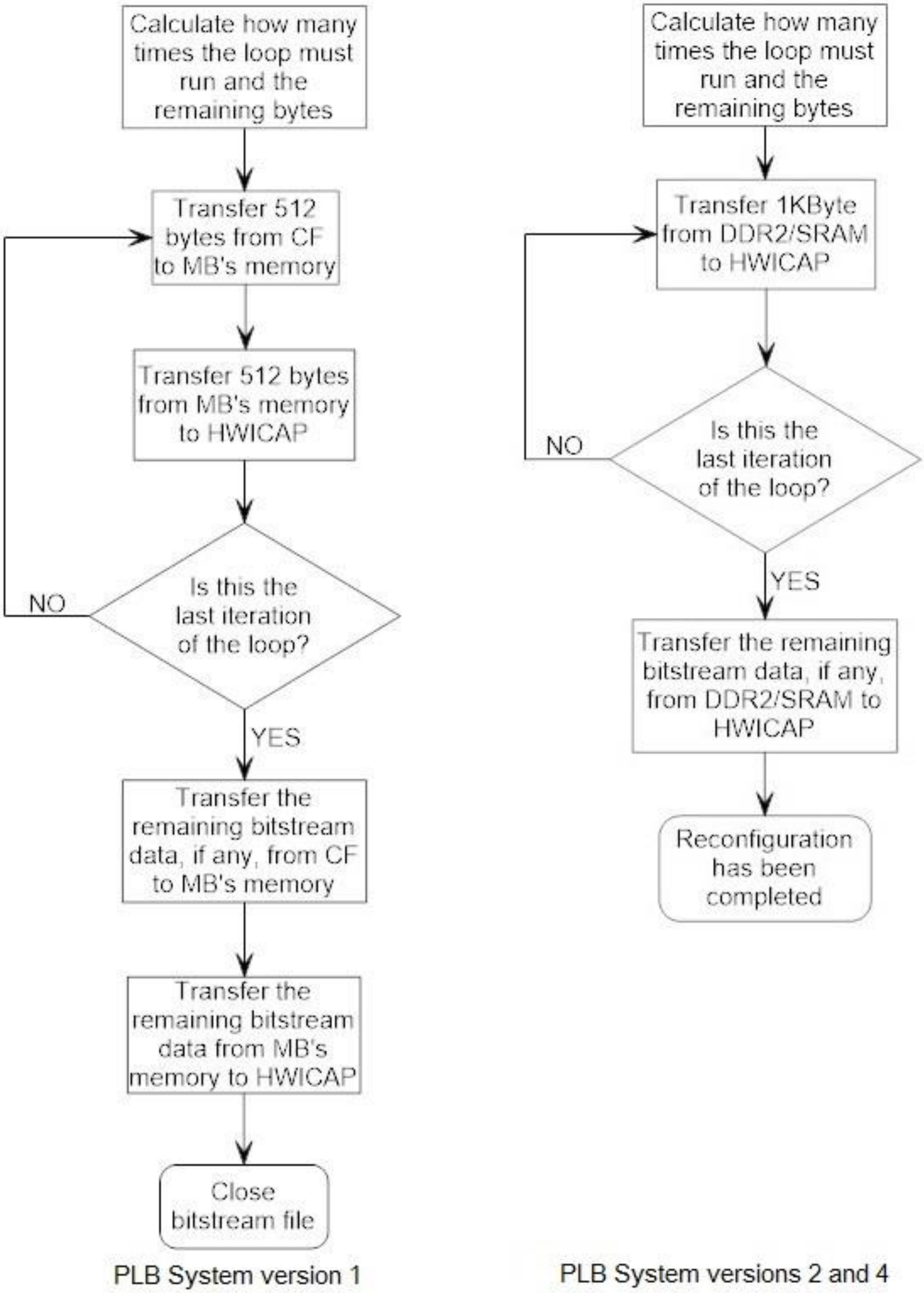
Figure 4.3 Flow diagram of reconfiguration for the OPB System version 1


Calculate how many times the loop must run and the remaining bytes
Transfer 4 bytes from DDR2 to HWICAP BRAM
Is HWICAP BRAM full?
NO
YES
Transfer data from HWICAP BRAM to the ICAP
Is this the last iteration of the loop?
NO
YES
Transfer 4 bytes from DDR2 to HWICAP BRAM
Has all remaining bitstream data been transferred to HWICAP BRAM?
NO
YES
Transfer data from HWICAP BRAM to the ICAP
Reconfiguration has been completed
Reset the DMA and set the Control Register
Reset Control FSM
Start DMA
Check Status Register
Has transfer completed?
NO
YES
Check if FIFO is empty
Is FIFO empty?
NO
YES
Reconfiguration has been completed

OPB System version 2

OPB System version 3

Figure 4.4 Flow diagrams of reconfiguration for the OPB System version 2 and 3


Calculate how many times the loop must run and the remaining bytes
Transfer 512 bytes from CF to MB's memory
Transfer 512 bytes from MB's memory to HWICAP
Is this the last iteration of the loop?
NO
YES
Transfer the remaining bitstream data, if any, from CF to MB's memory
Transfer the remaining bitstream data from MB's memory to HWICAP
Close bitstream file
Calculate how many times the loop must run and the remaining bytes
Transfer 1KByte from DDR2/SRAM to HWICAP
Is this the last iteration of the loop?
NO
YES
Transfer the remaining bitstream data, if any, from DDR2/SRAM to HWICAP
Reconfiguration has been completed

PLB System version 1

PLB System versions 2 and 4

Figure 4.5 Flow diagrams of the reconfiguration for the PLB Systems

Chapter 5

Experimental Results

This chapter presents some time results that were measured to check the performance of the designs. To take time results a timer peripheral was used that is provided by Xilinx. This peripheral can count clock cycles which then, based on clock frequency can be converted to time. It is also connected to the main bus (OPB or PLB) and can send the measurements to the processor. To count time using this peripheral the following functions were used:

- **XTmrCtr_Initialize():** This function initializes a timer instance.
- **XTmrCtr_SetResetValue():** This function sets the value that the timer will get when it is reset.
- **XTmrCtr_Reset():** This function resets the timer.
- **XTmrCtr_Start():** This function starts the timer.
- **XTmrCtr_Stop():** This function stops the timer.
- **XTmrCtr_GetValue():** This function return the number of the cycles that have been counted so far.

As it was explained in chapter 4 at some of the designs the partial bitstream file is transferred directly from the storage memory to the ICAP during a reconfiguration and at others it is transferred through other memories. For the later designs except of the total reconfiguration time, the total time for each transfer was measured. More specifically for the designs that use the OPB bus we have the following time measurements:

- t_{CFtoMB}: The total time for the transfer of the bitstream from CompactFlash to MicroBlaze's main memory.
- $t_{MBtoBRAM}$: The total time for the transfer of the bitstream from MicroBlaze's main memory to the BRAM of the HWICAP.
- $t_{DDR2toBRAM}$: The total time for the transfer of the bitstream from DDR2 to the BRAM of the HWICAP.
- $t_{BRAMtoICAP}$: The total time for the transfer of the bitstream from the BRAM of the HWICAP to the ICAP.

For the designs that use the PLB bus we have the following time measurements:

- t_{CFtoMB}: The total time for the transfer of the bitstream from CompactFlash to MicroBlaze's main memory.
- $t_{MBtoICAP}$: The total time for the transfer of the bitstream from MicroBlaze's main memory to the ICAP.

For the time measurements three partial bitstream files were used: the AES algorithm, the Blowfish algorithm and a blank bitstream file. At the OPB designs each partial bitstream file has different size from the others of the same design. At each PLB design however all partial bitstream files have the same size. At the designs that use the SRAM as the storage memory the blank bitstream file was not used because the size of the SRAM was not enough for all three bitstream files.

One way to improve the performance when using the DDR2 or the SRAM as the storage memory is to set each one of these memories as the data cache for MicroBlaze. When the cache is enabled the processor can communicate with the corresponding memory faster. To see how much the design is improved when the cache is enabled, two different measurements were taken for each design, one with the cache disabled and one with cache enabled. The size of the cache was set to 16Kbytes. The following tables show the time results for each design.

Partial Bitstream	Bitsream Size (Kb)	CF to MB		MB to BRAM		BRAM to ICAP		Total	
		time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)
AES	355.8	710.5	0.5	25.5	13.6	3.8	92.6	751.2	0.5
Blowfish	275.1	506.1	0.5	19.7	13.6	2.9	92.6	537.8	0.5
Blank	215	393.7	0.5	15.4	13.6	2.3	92.5	418.5	0.5

Table 5.1 Time Results of OPB Design Version 1

Partial Bitstream	Bitsream Size (Kb)	DDR2 to BRAM		BRAM to ICAP		Total	
		time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)
AES	225.4	17.7	12.5	2.4	92.6	22.3	9.9
Blowfish	183	14.3	12.5	1.9	92.6	18.1	9.9
Blank	150.5	11.8	12.5	1.6	92.6	14.9	9.9

Table 5.2 Time Results of OPB Design Version 2 (cache enabled)

Partial Bitstream	Bitsream Size (Kb)	DDR2 to BRAM		BRAM to ICAP		Total	
		time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)
AES	214.9	27.1	7.7	2.3	92.6	31.2	6.7
Blowfish	176.3	22.3	7.7	1.9	92.6	25.6	6.7
Blank	133	16.8	7.7	1.4	92.6	19.3	6.7

Table 5.3 Time Results of OPB Design Version 2 (cache disabled)

Partial Bitstream	Bitsream Size (Kb)	Total time (ms)	Total throughput (Mb/s)
AES	222.7	3.1	68.2
Blowfish	181.3	2.6	68.1
Blank	141.5	2.0	68.1

Table 5.4 Time Results of OPB Design Version 3

Partial Bitstream	Bitsream Size (Kb)	CF to MB		MB to ICAP		Total	
		time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)	time (ms)	throughput (Mb/s)
AES	561	985.7	0.6	25.5	21.5	1011.2	0.5
Blowfish	561	899.9	0.6	25.5	21.5	933.2	0.6
Blank	561	864.8	0.6	25.5	21.5	890.2	0.6

Table 5.5 Time Results PLB Design Version 1

Partial Bitstream	Bitsream Size (Kb)	Total time (ms)	Total throughput (Mb/s)
AES	561	29.8	18.4
Blowfish	561	29.8	18.4
Blank	561	29.8	18.4

Table 5.6 Time Results PLB Design Version 2 (cache enabled)

Partial Bitstream	Bitsream Size (Kb)	Total time (ms)	Total throughput (Mb/s)
AES	670	61.5	10.6
Blowfish	670	61.5	10.6
Blank	670	61.5	10.6

Table 5.7 Time Results PLB Design Version 2 (cache disabled)

Partial Bitstream	Bitsream Size (Kb)	Total time	Total throughput
AES	526	4.2	120.6
Blowfish	526	4.6	112.0
Blank	526	4.4	116.4

Table 5.8 Time Results PLB Design Version 3

Partial Bitstream	Bitsream Size (Kb)	Total time	Total throughput
AES	449	22.9	19.1
Blowfish	449	22.9	19.1

Table 5.9 Time Results PLB Design Version 4 (cache enabled)

Partial Bitstream	Bitsream Size (Kb)	Total time	Total throughput
AES	449	35.4	12.4
Blowfish	449	35.4	12.4

Table 5.10 Time Results PLB Design Version 4 (cache disabled)

Partial Bitstream	Bitsream Size (Kb)	Total time	Total throughput
AES	421	3.0	138.6
Blowfish	421	3.0	138.6

Table 5.11 Time Results PLB Design Version 5

From these results we can see that both designs that use the Compact Flash as the storage memory are much slower than the other designs. Figures 5.1 and 5.2 show the time allocation for each transfer of the partial bitstream at these two design. From these charts we can see that the 94.1% and 97% of the reconfiguration time respectively is the duration of the transfer from the Compact Flash to the memory of the MicroBlaze. This is the reason why the throughput of this transfer is almost equal to the total throughput. Figure 5.3 shows the total throughput of the designs that use the DDR2 or the SRAM as storage memory with cache enabled and disabled. From this chart we can see that the use of the cache improves significantly the performance of the design, especially for the PLB designs. Finally figure 5.4 shows the highest throughput that was achieved for each design. From this chart we can conclude that the designs that use the DMA are much faster than the others and so this method of reconfiguration is the most efficient.

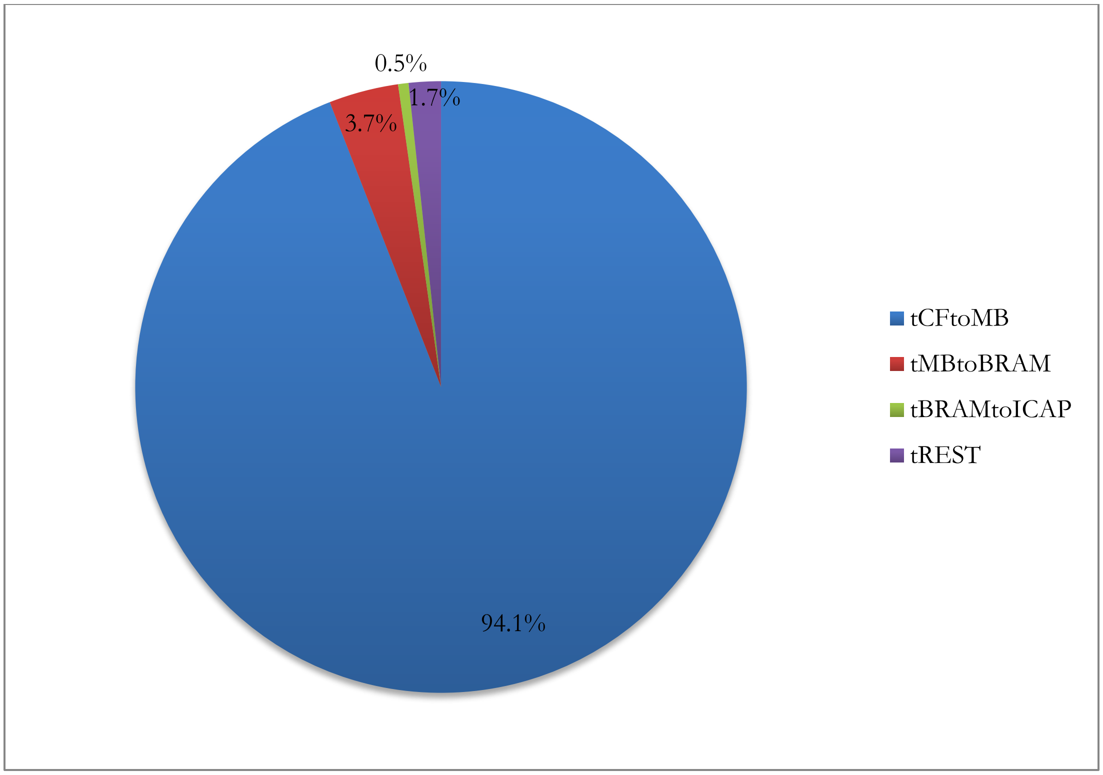


Figure 5.1 Time allocation for each transfer at OPB v1 design

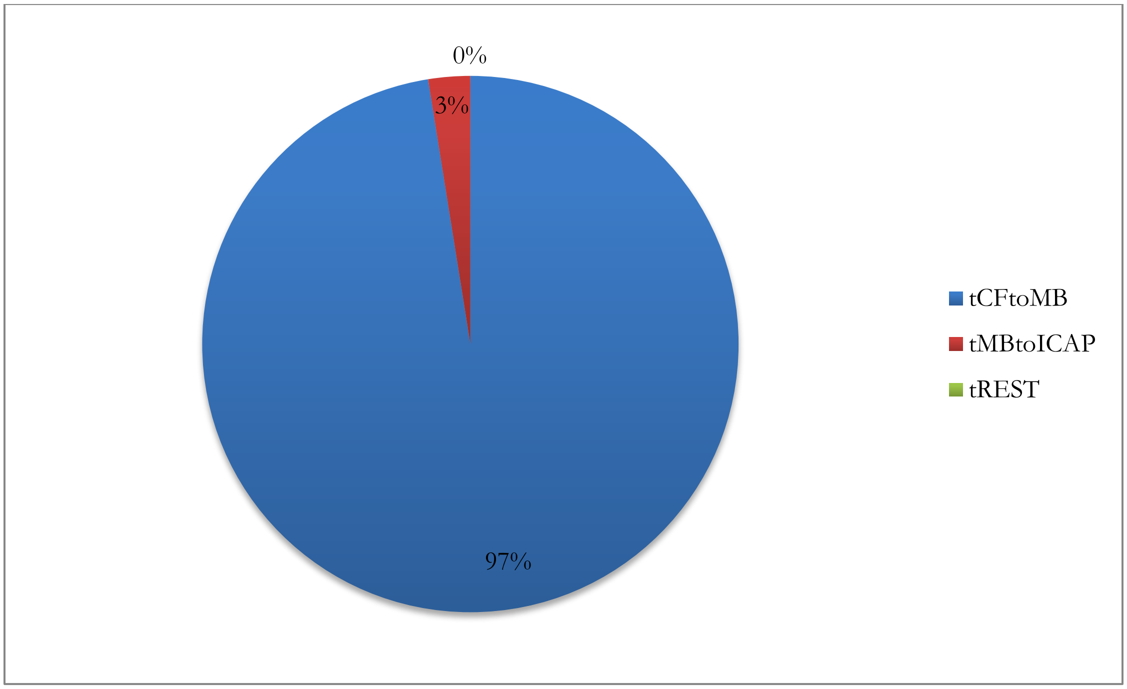


Figure 5.2 Time allocation for each transfer at PLB v1 design

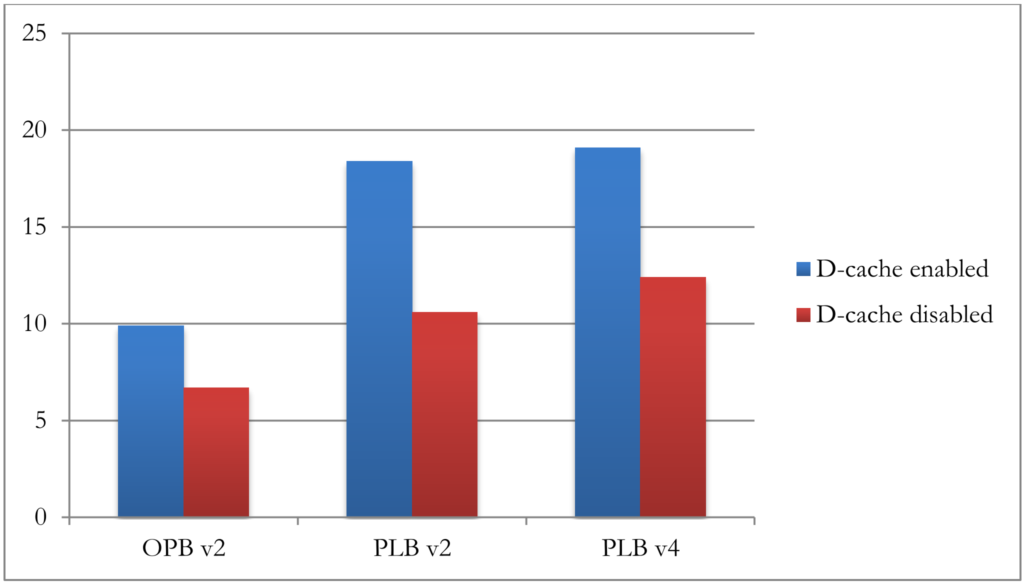


Figure 5.3 Throughput comparison chart for the designs that can use the storage memory as data cache

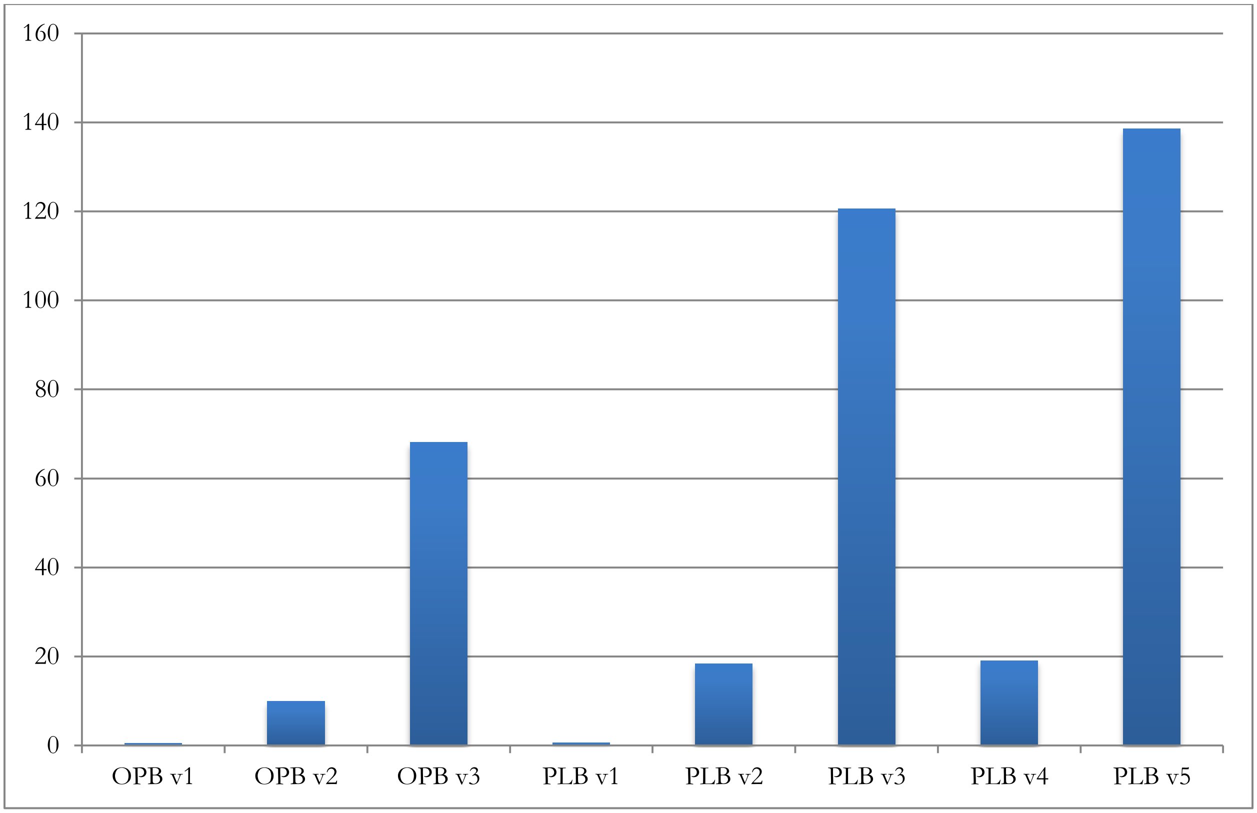


Figure 5.3 Total throughput of each design

Table 5.12 shows the maximum reconfiguration throughput that was achieved from each of the projects that were mentioned in paragraph 2.4 and from this work. The second column of this table has the memories that were used as the storage memory and the third column shows if the design uses a custom ICAP controller or the HWICAP. From this table we can see that the designs with the highest throughput use a custom ICAP controller. We can also see that some of these designs can achieve even higher throughput than the one that was achieved in this work which means that there are more improvements that can be applied to the final implementation.

Reference	Storage Memory	ICAP Controller	Max Reconfiguration Throughput (MB/s)
[1]	Host PC	Custom	3.88
[2]	DDR SDRAM	Custom	93.53
[3]	DDR2 SDRAM	Custom	295.4
[4]	BRAM	HWICAP	4.13
[7]	BRAM	HWICAP	1.46
[8]	CF	HWICAP	0.15
[10]	DDR2 SDRAM	HWICAP	4.48
[11]	DDR2 SDRAM	Custom	353.2
[12]	SRAM	Custom	290.23
[13]	BRAM	Custom	332.1
[14]	CF	HWICAP	0.12
[15]	CF	HWICAP	0.19
[16]	DDR SDRAM	HWICAP	5.16
[17]	CF	HWICAP	0.19
[26]	DDR3 SDRAM	Custom	838.55
[27]	DDR3 SDRAM	HWICAP	268.75
This work	SRAM	Custom	138.6

Table 5.12 Design Characteristics and Maximum Reconfiguration Throughput of Referenced Projects.

Chapter 6

Conclusions and Future Work

For the purpose of this thesis, we designed and implement several systems that perform PR on a Virtex-5 FPGA. For each system we used two cryptographic algorithms as the RMs. The main goal is to achieve better performance with each new system.

The first system was a simple design that uses a Compact Flash as the storage memory and the bitstream files were transferred through the Micro Blaze to the ICAP. The first step to improve the performance was to replace the slow Compact Flash with the much faster DDR2 or the SRAM. These memories can also be used as the cache of the MicroBlaze which improves the performance even more. Then a DMA peripheral was inserted to transfer the partial bitstreams directly from the storage memory to the ICAP and not through the MicroBlaze. The HWICAP was also replaced with a simple controller that consists of a FIFO where the bitsreams are transferred by the DMA and a FSM which controls the transfer of the bitstreams from that FIFO to the ICAP. The highest reconfiguration throughput that was achieved from the final design is 138.6 MB/s.

As part of future work, several modifications can be implemented to improve the reconfiguration unit:

- A master bus interface can be inserted in the FIFO peripheral, so it can transfer the partial bitstream from the storage memory to the FIFO without the use of a DMA peripheral. This modification may not improve the reconfiguration throughput but it will reduce the size of the reconfiguration controller
- Instead of using a FIFO the BRAM blocks of the FPGA can be used as the storage memory. In this case the transfer of the partial bitstream from the storage memory to the ICAP will be faster and it will not require the use of the PLB bus..
- Finally, overclocking techniques can be used for the ICAP to make it work on a faster frequency than 100MHz which will increase the reconfiguration throughput.

References

[1] B. Griese, E. Vonnahme, M. Porrmann, U. Rücket. Hardware Support for Dynamic Reconfiguration in Reconfigurable SoC Architectures. *Proceedings of the International Workshop on Field Programmable Logic and Applications (FPL), 2004.*

[2] C. Clause, F. H. Müller, J. Zeppenfeld, W. Stechele. A New Framework to Accelerate Virtex-II Pro Dynamic Partial Self-Reconfiguration. *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2007*

[3] C. Clause, B. Zhang, W. Stechele, L. Braun, M. Hübner, J. Becker. A Multi-Platform Controller Allowing for Maximum Dynamic Partial Reconfiguration Throughput. *Proceedings of the International Workshop on Field Programmable Logic and Applications (FPL), 2008.*

[4] I. Gelado, E. Morancho, N. Navarro. Experimental Support for Reconfigurable Application-Specific Accelerators. *Proceeding of the Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA), 2006.*

[5] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, T. Becker. Modular Partial Reconfiguration in Virtex FPGAs. *Proceedings of the International Conference of Field Programmable Logic and Applications, 2005.*

[6] K. Papademetriou, A. Dollas. Performance Evaluation of a Preloading Model in Dynamically Reconfigurable Processors. *Proceedings of the International Conference of Field Programmable Logic and Applications, 2006.*

[7] K. Papademetriou, A. Anyfantis, A. Dollas. Methodology and Experimental Setup for the Determination of System-level Dynamic Configuration Overhead. *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2007.*

[8] K. Papademetriou, A. Anyfantis, A. Dollas. An Effective Framework to Evaluate Dynamic Partial Reconfiguration in FPGA Systems. *IEEE Transactions on Instrumentation and Measurement, 2010.*

[9] K. Papademetriou, A. Dollas, S. Hauck. Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model. *Journal ACM Transactions on Reconfigurable Technology and Systems, 2011.*

[10] M. French, E. Anderson, D. -I. Kang. Autonomous System on a Chip Adaptation through Partial Runtime Reconfiguration. *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2008.*

[11] P. Manet, D. Maufroid, L. Tosi, G. Gailliard, O. Mulertt, M. DiCiano, J. D. Legat, D. Aulagnier, C. garmat, R. Liberati, V. LaBarba, P. Cuvelier, B. Rousseau, P. Gelineau.

An Evaluation of Dynamic Partial Recnfiguration for Signal and Image Processing in Professional Electronics Applications. *EURASIT Journal on Embedded Systems, 2008.*

[12] S. Liu, R. N. Pittman, A. Forin. Minimizing Partial Reconfiguration Overhead with Fully Streaming DMA Engines and Intelligent ICAP Controller. *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2009.*

[13] M. Liu, W. Kuehn, Z. Lu, A. Jantsch. Run-TimePartial Reconfiguration Speed Investigation and Architectural Design Space Exploration. *Proceedings of the International Workshop on Field Programmable Logic and Applications (FPL), 2009.*

[14] C. Effraimidis. A Self Reconfigurable Architecture to Support Multiple Fitness Functions in Generic Algorithms. *Diploma Thesis Technical Report, ECE Dpt., Technical University of Crete, 2009.*

[15] G. Nikoloudakis. Design and Implementation, Using Dynamic Partial Reconfiguration, of a System Implementing Multiple Cryptographic Algorithms. *Diploma Thesis Technical Report, ECE Dpt., Technical University of Crete, 2009.*

[16] E. Spanakis. Design and Implementation of a Linux-Based Dynamic Reconfiguration Task Manager in FPGAs. *Diploma Thesis Technical Report, ECE Dpt., Technical University of Crete, 2009.*

[17] A. Ilias. Design and Implementation of a Partially Reconfigurable System for Fault Tolerant Applications. *Diploma Thesis Technical Report, ECE Dpt., Technical University of Crete, 2009.*

[18] Virtex-5 Family Overview.
http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

[19] Virtex-5 FPGA Configuration User Guide.
http://www.xilinx.com/support/documentation/user_guides/ug191.pdf

[20] OPB HWICAP (v1.00.b) Datasheet.
http://forums.xilinx.com/xlnx/attachments/xlnx/elinux/494/1/opb_hwicap.pdf

[21] Multi-CHannel OPB Double Data Rate 2 (DDR2) Synchronous DRAM (SDRAM) Controller (v1.02.a) Datasheet.

[22] OPB Central DMA Controller (v1.00.c) Datasheet.

[23] LogiCORE IP XPS HWICAP (v5.00a) Datasheet.
http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf

[24] Multi-Port Memory Controller (MPMC) (v6.02.a) Datasheet.

[25] XPS Multi-CHannel External Memory Controller (XPS MCH EMC) (v3.01.a) Datasheet.
http://www.xilinx.com/support/documentation/ip_documentation/xps_mch_emc.pdf

[26] K. Vipin, Suhaib A. Fahmy. A High Speed Open Source Controller for FPGA Partial Reconfiguration. *International Conference on Field-Programmable Technology (FTP), 2012.*

[27] H. Kashyap, R. Chaves. Secure Partial Dynamic Reconfiguration with Unsecure External Memory. *24th International Conference on Field Programmable Logic and Applications (FPL), 2014.*

[28] T. D. A. Nguyen, A. Kumar. PR-HMPSoC: a Versatile Partially Reconfigurable Heterogeneous Multiprocessor System-on-Chip for Dynamic FPGA-based Embedded Systems. *24th International Conference on Field Programmable Logic and Applications (FPL), 2014.*

[29] A. Morales-Villanueva, A. Gordon-Ross. On-chip Contex Save and Restore of Hardware Tasks on Partially Reconfigurable FPGAs. *IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2013*

[30] E. Cetin, O. Diessel, L. Gong, V. Lai. Towards Bounded Error Recovery Time in FPGA-Based TMR Circuits Using Dynamic Partial Reconfiguration. *23rd International Conference on Field Programmable Logic and Applications (FPL), 2013*

Appendix

This appendix describes same features of the Virtex-5 FPGAs, like the one that was used for this thesis.

Configurable Logic Blocks

Configurable Logic Blogs (CLBs) are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix for access to the general routing matrix as shown in figure 1. A CLB element contains a pair of slices. These two slices do not have direct connection to each other, and each slices organized as a column. Each slice in a column has 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 labeled as SLICE(1).

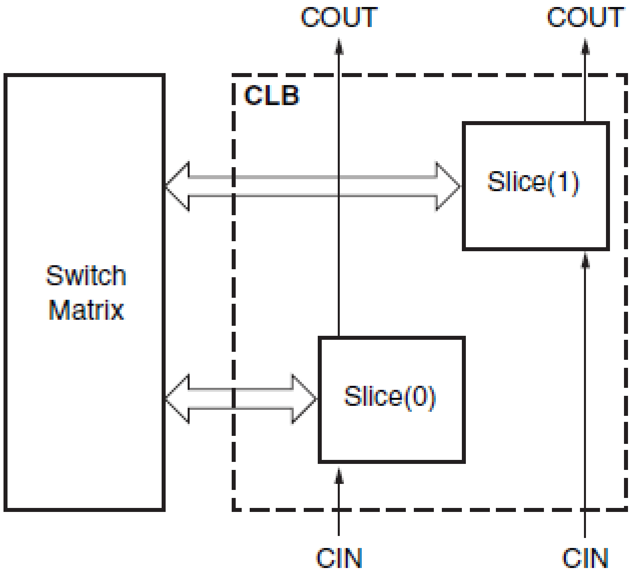


Figure 1 Arrangement of Slices within the CLB

Every slice contains four logic-function generators, four storage elements, wide-function multiplexers, and carry logic. These elements are used by all slices to provide logic, arithmetic, and ROM functions. In addition to this, some slices support two additional functions: storing data using distributed RAM and shifting data with 32-bit registers. Slices that support these additional functions are called SLICEM; others are called SLICEL. The function generators are configurable as 6-input LUTs or dual-output 5-input LUTs. The four storage elements can be configured as either edge-triggered D-type flip-flops or level sensitive latches.

Block RAM

Block RAM modules provide flexible 36 Kbit true dual-port RAM that are cascadable to form larger memory blocks. In addition, Virtex-5 FPGA block RAMs contain optional programmable FIFO logic for increased device utilization. Each block RAM can also be configured as two independent 18 Kbit true dual-port RAM blocks, providing memory granularity for designs needing smaller RAM blocks.

Input/Output Blocks

I/O blocks provide the interface between package pins and internal configurable logic. Most popular and leading-edge I/O standards are supported by programmable I/O blocks (IOBs). The IOBs can be connected to very flexible ChipSync logic for enhanced source-synchronous interfacing. Source-synchronous optimizations include per-bit deskew (on both input and output signals), data serializers/deserializers, clock dividers, and dedicated I/O and local clocking resources.

DSP48E Slices

Cascadable embedded DSP48E slices with 25 x 18 two's complement multipliers and 48-bit adder/subtracter/accumulator provide massively parallel DSP algorithm support. In addition, each DSP48E slice can be used to perform bitwise logical functions.

Clock Management Tile Blocks

Clock Management Tile (CMT) blocks provide the most flexible, highest-performance clocking for FPGAs. Each CMT contains two Digital Clock Manager (DCM) blocks (self-calibrating, fully digital), and one PLL block (self-calibrating, analog) for clock distribution delay compensation, clock multiplication/division, coarse-/fine-grained clock phase shifting, and input clock jitter filtering.

General Routing Matrix

The General Routing Matrix (GRM) provides an array of routing switches between each internal component. Each programmable element is tied to a switch matrix, allowing multiple connections to the general routing matrix. The overall programmable interconnection is hierarchical and designed to support high-speed designs. In Virtex-5 devices, the routing connections are optimized to support CLB interconnection in the fewest number of "hops". Reducing hops greatly increases post place-and-route (PAR) design performance.

Some other features are the following:

LXT, SXT, TXT, and FXT devices contain:

- Integrated Endpoint blocks for PCI Express designs providing x1, x4, or x8 PCI Express Endpoint functionality. When used in conjunction with RocketIO transceivers, a complete PCI Express Endpoint can be implemented with minimal FPGA logic utilization.
- 10/100/1000 Mb/s Ethernet media-access control blocks offer Ethernet capability.

LXT and SXT devices contain:

- RocketIO GTP transceivers capable of running up to 3.75Gb/s. Each GTP transceiver supports full-duplex, clock-and-data recovery.

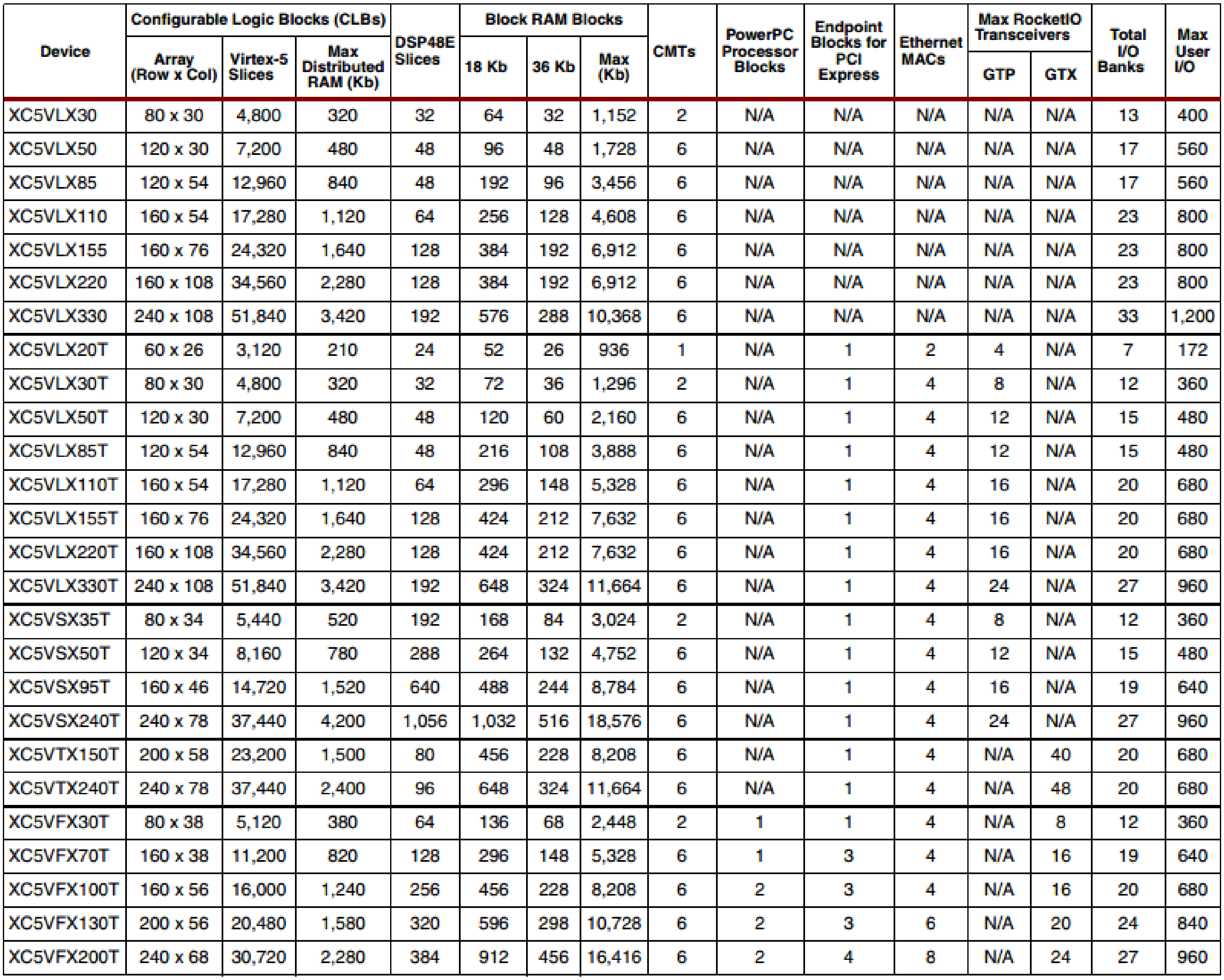
TXT and FXT devices contain:

- GTX transceivers capable of running up to 6.5Gb/s. Each GTX transceiver supports full-duplex, clock-and-data recovery.

FXT devices contain:

- Embedded IBM PowerPC 440 RISC CPUs. Each PowerPC 440 CPU is capable of running up to 550 MHz. Each PowerPC 440 CPU also has an APU (Auxiliary Processor Unit) interface that supports hardware acceleration, and an integrated cross-bar for high data throughput.

The following table shows the resources of each Virtex-5 FPGA.



Device	Configurable Logic Blocks (CLBs)			DSP48E Slices	Block RAM Blocks			CMTs	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs	Max RocketIO Transceivers		Total I/O Banks	Max User I/O
	Array (Row x Col)	Virtex-5 Slices	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480
XC5VLX85T	120 x 54	12,960	840	48	216	108	3,888	6	N/A	1	4	12	N/A	15	480
XC5VLX110T	160 x 54	17,280	1,120	64	296	148	5,328	6	N/A	1	4	16	N/A	20	680
XC5VLX155T	160 x 76	24,320	1,640	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX220T	160 x 108	34,560	2,280	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX330T	240 x 108	51,840	3,420	192	648	324	11,664	6	N/A	1	4	24	N/A	27	960
XC5VSX35T	80 x 34	5,440	520	192	168	84	3,024	2	N/A	1	4	8	N/A	12	360
XC5VSX50T	120 x 34	8,160	780	288	264	132	4,752	6	N/A	1	4	12	N/A	15	480
XC5VSX95T	160 x 46	14,720	1,520	640	488	244	8,784	6	N/A	1	4	16	N/A	19	640
XC5VSX240T	240 x 78	37,440	4,200	1,056	1,032	516	18,576	6	N/A	1	4	24	N/A	27	960
XC5VTX150T	200 x 58	23,200	1,500	80	456	228	8,208	6	N/A	1	4	N/A	40	20	680
XC5VTX240T	240 x 78	37,440	2,400	96	648	324	11,664	6	N/A	1	4	N/A	48	20	680
XC5VFX30T	80 x 38	5,120	380	64	136	68	2,448	2	1	1	4	N/A	8	12	360
XC5VFX70T	160 x 38	11,200	820	128	296	148	5,328	6	1	3	4	N/A	16	19	640
XC5VFX100T	160 x 56	16,000	1,240	256	456	228	8,208	6	2	3	4	N/A	16	20	680
XC5VFX130T	200 x 56	20,480	1,580	320	596	298	10,728	6	2	3	6	N/A	20	24	840
XC5VFX200T	240 x 68	30,720	2,280	384	912	456	16,416	6	2	4	8	N/A	24	27	960

Table 1 Virtex-5 FPGA Family Members