FEASIBILITY STUDY FOR DYNAMIC CONTEXT SWITCHING IN PARTIALLY RECONFIGURABLE FPGAS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ESAT YILMAZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

FEASIBILITY STUDY FOR DYNAMIC CONTEXT SWITCHING IN PARTIALLY RECONFIGURABLE FPGAS

submitted by **ESAT YILMAZ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering, METU _____

Prof. Dr. Gözde B. Akar
Electrical and Electronics Engineering, METU _____

Prof. Dr. Ece G. Schmidt
Electrical and Electronics Engineering, METU _____

Prof. Dr. Ali Ziya Alkar
Electrical and Electronics Engineering, Hacettepe University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Esat Yılmaz

Signature :

ABSTRACT

FEASIBILITY STUDY FOR DYNAMIC CONTEXT SWITCHING IN PARTIALLY RECONFIGURABLE FPGAS

Yılmaz, Esat
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı

September 2019, 74 pages

Reconfiguration of computing and control circuits according to dynamically changing needs is a supportive concept which saves design-time and the space needed for floorplanning in comparison to application specific integrated circuits (ASIC). FPGAs which are commonly used reconfigurable devices have both full and partial reconfiguration features. Dynamic partial reconfiguration is a technique which enables some part of the circuit to be reconfigured while other parts are running. This feature allows the user to switch between different and successive tasks working in a particular block of an FPGA device. Preemption of a task might also be needed in dynamically running circuits for real-time/time-critical application requirements. Preemption requires that all current state information of the circuit is saved somewhere else before running another circuit and to run the previously saved circuit where it was stopped from.

This thesis study investigates the feasibility of dynamic context switching in modern-day FPGAs. For this, a reconfigurable System-on-Chip (SoC) architecture is examined. Xilinx Zynq SoC is used and AXI4-based partially reconfigurable block struc-

ture is implemented. By using DMA, readback and reconfiguration structures are implemented. DDR memory is used to store bitstream files when a partial bitstream file is downloaded to the FPGA. The resulting system designed helps to reduce required resources for big size circuits by providing and enabling a context-save and context-restore mechanism for time-critical tasks with considerably low overhead.

Keywords: Partial Reconfiguration, Context-Switch, Zynq SoC, AXI protocol

ÖZ

KISMİ YENİDEN YAPILANDIRILABİLİR FPGA ÜZERİNDE DİNAMİK İÇERİK DEĞİŞTİRMENİN YAPILABİLİRLİK ÇALIŞMASI

Yılmaz, Esat
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Cüneyt F. Bazlamaçcı

Eylül 2019 , 74 sayfa

Hesaplama ve kontrol devrelerinin dinamik olan isteklere göre yeniden yapılandırılması işlemi Uygulamaya Dönük Entegre Devreleri'ne (ASIC) göre tasarım süresini kısaltan ve yeniden yapılandırılabilir alanın daha etkin kullanılmasını sağlayan tasarımcıya yardımcı bir kavramdır. Sıklıkla kullanılan Alan Programlanabilir Kapı Dizinleri (FPGA) hem tam hem de parçalı yeniden programlanma özelliğine sahiptir. Dinamik olarak kısmi programlama tekniği diğer kısımlar çalışırken sadece belli bir kısmı programlamak için kullanılan tekniktir. Bu teknik FPGA üzerinde farklı ve art arda çalışan devreler arasında geçiş yapmaya izin verir. Bir çalışan devrenin durdurularak çalıştığı bölgeden çıkarılması ve daha sonra tekrar çalıştırılması zaman-kritik ve gerçek zamanlı uygulamalarda gereklidir. Devrenin bulunduğu kısımdan çıkarılması bütün durum bilgilerinin başka bir yere kaydedilmesini ve daha sonra kaydedildiği yerden okunup çalışmasına kaldığı yerden devam ettirilmesini gerektirir.

Bu tez çalışmasında, günümüz FPGA entegrelerinde dinamik olarak içerik değiştirmenin uygulanabilirliği araştırılmıştır. Bunun için, Yonga üzeri Sistem (SoC) mimarisinde yeniden programlanabilir bir sistem incelenmiştir. Xilinx firmasına ait Zynq

SoC devresi kullanılmıştır ve AXI4 tabanlı kısmi programlanabilir blok mimarisi uygulanmıştır. DMA mimarisi kullanılarak, çalışan devreyi geri okuma ve yeniden programlama yapısı uygulanmıştır. DDR hafıza programlama dosyalarının saklanması için kullanılmıştır. Programlama dosyaları FPGA'yı programlamak için kullanılmaktadır. Tasarlanan sistem büyük bir mimari gerektiren devreler için gerekli kaynağı azaltıp, zaman kritik uygulamalarda yapılandırma hafızasını okuma ve hafızaya yazma mekanizmalarını az bir zaman kaybıyla mümkün kılmaktadır.

Anahtar Kelimeler: Kısmi Yeniden Yapılandırma, İçerik Değiştirme, Zynq SoC, AXI protokolü

To my wife and my son

ACKNOWLEDGMENTS

Firstly, I must thank my wife Merve Yılmaz, my parents Sadık and Ayşegül Yılmaz and my sister Şerife E. Duman for their support and patience.

I must thank my advisor, Assoc. Prof. Dr. Cüneyt Fehmi Bazlamaçcı for his encouragement, support and guidance during my MSc study.

I would like to thank my employer ASELSAN for MSc support during my study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

1D	1 Dimensional
2D	2 Dimensional
BLE	Basic Logic Element
BRAM	Block Random Access Memory
CLB	Configurable Logic Block
CS	Context-Switch
DMA	Direct Memory Access
DPR	Dynamic Partial Reconfiguration
DRC	Design Rule Check
FF	Flip Flop
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
GSR	Global Set Reset
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
MB	Megabyte
OOC	Out Of Context
PCAP	Processor Configuration Access Port
PL	Programmable Logic
PR	Partial Reconfiguration
PS	Processor System
SoC	System-On-Chip

CHAPTER 1

INTRODUCTION

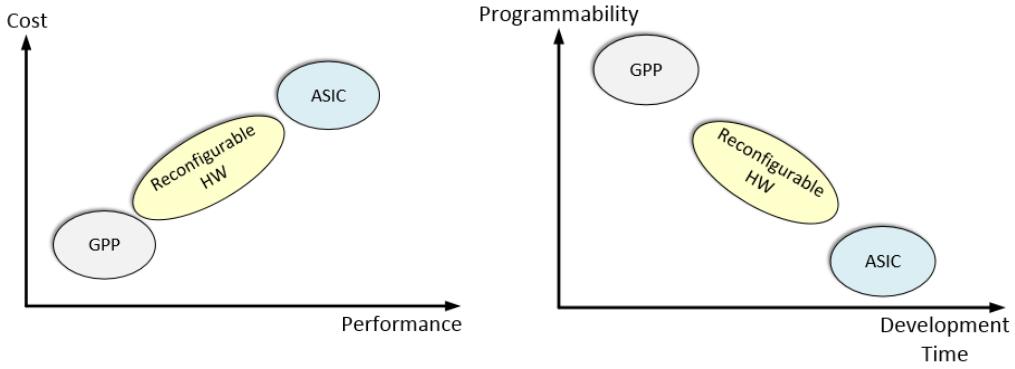
Embedded systems can contain different processing units that are manufactured with different purposes. The requirements of target system determine the type of processor at design stage. There are three commonly used integrated circuits as a processor in embedded systems. These are general purpose processor (GPP), field programmable gate array (FPGA) and application specific integrated circuit (ASIC). Each comes with different performance and cost options. Also, there are other important requirements such as development time, power usage and reconfigurability.

GPPs are designed to respond to its own instruction set which a user may send to processor unit. This model is commonly used in personal computers or workstations. Its architecture is usually based on Von-Neumann model in which the instruction and data are located at system memory and only a small set of read/write operations or computations are done at each clock cycle. There are three advantages of GPPs. These are availability of high-level programming which eases the development for most of the programmers, compatibility of instructions which decreases development time, and design flexibility which enables any computation to run. Although it has several advantages, these features are limited with constant clock cycle. Running the same serial and long set of instructions for different applications takes long time in every turn. This is not desired in computing intensive applications.

ASIC design process aims to decrease power consumption and increase the performance. The clock frequency can be much higher than other solutions. However, development and fabrication of an ASIC chip takes more time compared to other solutions since it has a fixed circuit running a specific task which needs to be verified under difficult conditions. When the application requirements change, it requires new

development and fabrication processes which needs time and money. Therefore, time to market is affected negatively. Additionally, it does not support full programmability so that no return is available after production.

Field programmable gate array (FPGA) chips are manufactured to meet the needs for programmability, performance, development cost & time [1, 2]. ASIC chips have generally better performance than FPGAs, but FPGAs have better performance than GPPs. Development time and programmability are also other factors where FPGA is in the middle. Reconfigurability feature of FPGAs is used to prototype a solution before the development of ASICs in order to decrease development cost. As depicted in Figure 1.1, reconfigurable hardware provides both features of GPPs and ASICs with allowable amount of penalty for embedded systems.



(a) (b)

Figure 1.1: GPP, ASIC and Reconfigurable Hardware Comparison in terms of Performance, Cost, Development Time and Programmability [1, 2].

The first generation of FPGAs requires to configure whole FPGA which causes long configuration time. They are not appropriate to use in dynamically changing computation platforms due to long reconfiguration time. Therefore, these devices were used as co-processor for acceleration because it supports massive parallelism. Performance improvement can be achieved with parallelism because smaller tasks can run concurrently rather than sequentially. Modern CPUs have multiple processing cores to decrease execution time of a task. FPGAs can also have multiple processing cores running in parallel due to its fine grained architecture.

Another architecture which takes advantages of both FPGA and GPP is System-On-Chip (SoC) chips. It includes both GPP and FPGA which are connected together with data buses inside. While an operating system runs on the GPP part with classical method of instruction fetching, a custom circuit can run coherently in the FPGA part to accelerate the computations.

New generation FPGAs can respond to dynamic changes during execution cycle. Partially reconfigurable FPGAs enable dynamic reconfiguration of pre-selected areas of FPGA while other areas are still in operation. It gives also flexibility to configure different circuits changeably on the reconfigurable part with a limited FPGA resource. Most importantly, configuration time significantly decreases due to smaller partial bitstream sizes. Considering computing applications in FPGA, reconfiguration time is critical since configuration overhead between different configurations should be minimized in comparison to computation time.

By combining the dynamic partial reconfiguration with SoCs, we can obtain a hardware platform running operating system in GPP side and a reconfigurable area which can be used to accelerate execution of time-consuming tasks. General SoC hardware architecture which supports partial reconfiguration is shown in Figure 1.2.

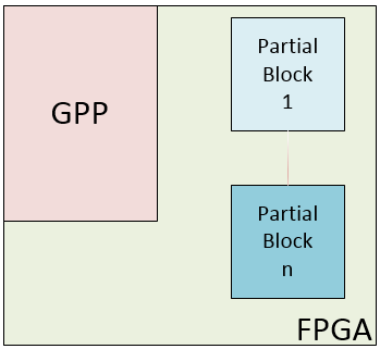


Figure 1.2: General Partially Reconfigurable SoC Hardware Architecture

While running a software in GPP side, some functions can be time consuming which makes the processor unable to respond other requests. Therefore, the processor may not be able to respond to some time-critical tasks until its deadline. However, if the time-consuming task is executed in partially reconfigured area, the processor can respond to other requests while also accelerating the execution of the task in FPGA

part. There might be also other tasks which need specifically FPGA implementation for acceleration. All of these can be reconfigured in partially reconfigurable area in case a request is triggered. Running both GPP and FPGA part at the same time could increase the overall performance, considerably in some applications.

1.1 Scope of the Thesis

This thesis work is conducted to implement context switching (CS) on a partially reconfigurable SoC. The SoC chip Xilinx Zynq-7020 is selected for this purpose. The board that runs the complete system is Avnet Zedboard which is commercially available in the market. Due to hybrid architecture of this SoC, both software flow and hardware implementation can run at the same time. Requirements for the implementation of dynamic partial reconfiguration (DPR) are established in FPGA part of the SoC. A Direct-Memory-Access (DMA) scheme is implemented to ease partial reconfiguration process. Preemption of a circuit is implemented with readback feature of FPGA configuration data. Addressable hardware task allocation is done with AXI bus.

Context switching (CS) in an FPGA implementation is relatively more complex with respect to GPP based system. When a personal computer manages the tasks with its CPU, several context switchings may occur due to preemptive scheduling scheme of the operating system. Time overhead for a context switch in reconfigurable hardware includes the readback of FPGA configuration data, bitstream manipulation and reconfiguration while it is just memory read/write operations in CPU based system. To sum up, CS is examined by employing partial reconfiguration on Xilinx SoC and some performance evaluation results are obtained.

1.2 Motivation

FPGAs are designed to get close to performance of ASIC designs while preserving a considerable level of flexibility/programmability like GPPs. On the other hand, SoCs are designed to get advantage of GPP while having the advantages of FPGA.

Since our implementation is based on a SoC architecture, VHDL and C languages are used to establish a system which runs both software and hardware tasks. This hybrid architecture makes the user to benefit both flexibility of GPP and performance of FPGA.

If a parallelized hardware task was run in GPP in a sequential manner, it could be inefficient in terms of power consumption. Another motivation is to decrease power requirement. By using partial reconfiguration, partially reconfigurable areas can be loaded with empty design to lower the power consumption.

Implementation and demonstration of the feasibility of context switching on reconfigurable hardware is the main motivation of the thesis. To be able to save the last state of the hardware to a memory and manipulate the bitstream which will be downloaded in the future is not as easy as in GPP. In addition, if the context switching is not fast enough, some time-critical tasks cannot be handled until its deadline which creates a problem for system requirements.

1.3 Contributions

The contributions of this work can be listed as follows:

- An AXI based addressable partially reconfigurable system is established. AXI4, AXI4-Lite and AXI4-Stream interfaces are used to control partially reconfigurable blocks which are connected to addressable AXI bus. Even if the partial block is reconfigured, it is always accessible from the processor system (PS) side with its 32-bit AXI address.

- An internal configuration access port (ICAP) controller is implemented to reconfigure the partially reconfigurable area. It is also used for the readback of configuration data to save the last state of registers and block RAM (BRAM) values. ICAPE2 hard macro is added to the controller as a submodule.

- Non-DMA based partial reconfiguration systems do not generally have high throughput. An AXI DMA is implemented to have a high partial reconfigura-

tion throughput. It has 380.1 MB/s throughput and it is supported by AXI4-Stream interface which is appropriate for fast data transaction.

- Bitstream manipulation is done to create a new bitstream from the last state information where the circuit is preempted. Frame based addressable structure of bitstream is resolved to create a new bitstream.
- Processor side has an infrastructure which controls the configuration data line. Processor configuration access port (PCAP) is used for full reconfiguration.

Preemptible and address-based reconfigurable blocks are designed to support parallelism on computing intensive embedded systems. Results show that dynamic context switching runs successfully on a reconfigurable FPGA. An FFT application example is presented to perform the hardware context switching.

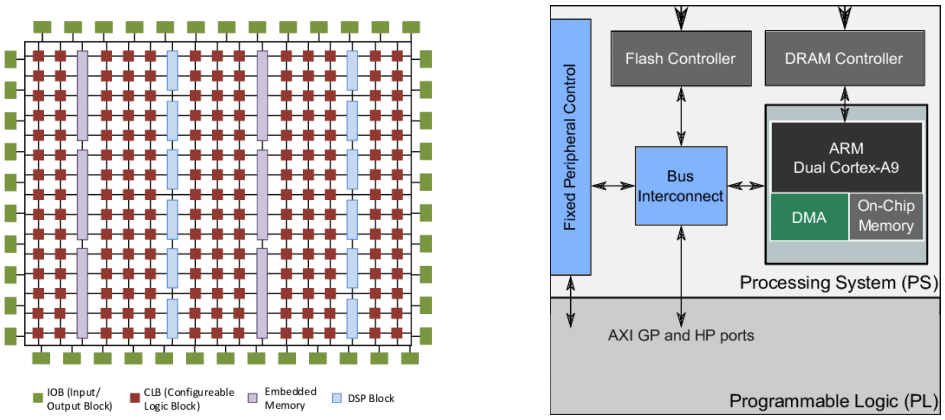
1.4 Thesis Organization

This thesis work contains background and related works for dynamic context switching in reconfigurable devices in Chapter 2. Concepts of reconfigurable computing is given and architectural background for FPGAs and SoCs is explained. For bitstream manipulation, its structure is given to understand how partial reconfiguration is performed. Context saving and restoring is explained in terms of timing and speed. Chapter 3 establishes some important system components and presents a task model corresponding to reconfigurable block model. It gives how context saving and restoring must be performed on the established system. Implementation phase is presented with an example application in Chapter 4. Firstly, a base system with reconfigurable blocks is designed and secondly, ICAP controller and other related designs are added to have a complete system for dynamic context switching. An FFT computation is performed on the established system with context saving, bitstream manipulation and context restoring successively. Floorplanning for this platform is given to visualize how static and partial areas of the Zynq SoC are used. Some throughput measurements are given in Chapter 5. The evaluation of these results is explained by comparing with other works. Chapter 6 finalizes this thesis with contributions, observations and future works.

CHAPTER 2

BACKGROUND AND RELATED WORK

Reconfigurable devices like FPGAs and SoCs are used in many sectors such as automotive, multimedia, military, scientific computing and enterprise computing. It is preferred because it can increase the performance while taking advantage of flexibility or programmability. When an embedded system is designed, the requirements usually contain high performance, low power consumption, flexibility, and low unit cost. Reconfigurable devices can be used to meet the requirements if the number of logic resources are enough. Otherwise, small number of logic cells in the selected hardware pushes the designer to select another bigger/expensive device since all tasks that will be executed cannot fit into the selected device. However, even if the device has limited logic resources, partial reconfiguration technique can be applied to run the tasks interchangeably.



(a) FPGA Architecture

(b) SoC Architecture

Figure 2.1: General FPGA and SoC architecture [29]

FPGA architecture consists of configurable logic blocks (CLBs) which creates an array in which each CLB is connected to each other via interconnects. Block RAMs and DSP blocks are located between CLBs. I/O pins are located at the boundary of the reconfigurable logic. General FPGA architecture is shown in Figure 2.1a

On the other hand, reconfigurable SoC architecture consists of both FPGA fabric and a GPP. These two are connected to each other via data buses and discrete lines. A processor, DMA, DRAM controller, flash memory controller etc. are already available in GPP side. General reconfigurable SoC architecture is shown in Figure 2.1b.

2.1 Concepts of Reconfigurable Computing

2.1.1 Static and Dynamic Configuration

There are some configuration concepts in reconfigurable hardware. A reconfigurable device like FPGA can be configured in two ways; static and dynamic configurations. When the device is configured just after power-up, it is called static configuration [3]. To reconfigure a running device statically, the user must stop the execution cycle and reconfigure it again. Dynamic configuration means changing the configuration while the device is running [4]. It does make sense when reconfiguration occurs partially in a selected area of reconfigurable device. It is called partial reconfiguration. Partial reconfiguration needs to be done carefully since FPGA has currently running circuit in specified area. The tools which are used to synthesize the partially reconfigurable system have stable synthesizer to protect the device from staying at intermediate state. Partial reconfiguration can also be done statically as shutting down the rest of the circuit and turning on again after configuration. Full reconfiguration is another configuration option to configure the whole device both statically and dynamically.

2.1.2 Partial Reconfiguration

Partial reconfiguration can be performed with either difference-based or module-based option [5].

- Difference-based partial reconfiguration enables minor logic changes in the circuit by picking only differences to create a partial bitstream. Switching from one implementation to another is very rapid due to small size of partial bitstream [6]. This method is used for old-fashion FPGAs but it is no longer supported on new generation FPGAs [5]. It is also not appropriate for the mechanism where hardware task block is loaded and unloaded.

- Module-based partial reconfiguration requires the design to be hierarchical [5]. Each partially loadable hardware module needs to be synthesized first to fit into the pre-selected area of FPGA's floorplan. The floorplan area is divided as static and dynamic. Static area is locked for all implementation phases and other reconfigurable circuits cannot change the overall design except for the reconfigurable area. This design option is used in this thesis work.

2.1.3 Coarse-Grained and Fine-Grained Architectures

Programmable logic can be in coarse-grained or fine-grained architecture or both [7, 8]. Coarse-grained architecture has specific blocks which do a specific computation or logic operation. The blocks have an internal circuit which cannot be reconfigured with a bitstream. For example, digital signal processing (DSP), floating point unit (FPU) or arithmetic logic unit (ALU) can be used in coarse-grained architecture. Reconfiguration is performed only in I/O level so that input and output of a block are connected to other blocks. Therefore, the bitstream contains only small routing information due to limited reconfigurable logic. On the other hand, fine-grained architecture contains bit level circuits. The routing information calculation lasts long compared to other architecture because of larger number of reconfigurable logics and interconnections. Recent FPGAs can be an example which contains both resources.

2.1.4 Single and Multi-Context Configuration

As the context of reconfigurable devices, there are two options; single and multi-context configuration. In single-context configuration, a reconfigurable device has only one configuration memory so that when a bitstream is downloaded to the device,

each bit is copied to specific and addressable configuration memory cell. When global reset signal is de-asserted, the flip-flops (FFs), LUTs and block RAM values are initialized from single on-chip memory. In multi-context configuration, configuration can be loaded with a single switch from one of multiple on-chip configuration memories [9]. The required memory area for multi-context configuration can be larger so that physical size of device may increase. Reconfiguration overhead is very low in this option since another configuration can be loaded just after a single clock cycle. This may reduce the overhead to nanoseconds.

2.1.5 Off-chip and Context Configuration

Downloading a bitstream using the serial or parallel interface to which an external flash memory is connected is called off-chip configuration. If the configuration is performed using internal or on-chip memory, it is called context configuration [4].

2.1.6 Readback and Readback Capture

Configuration memory can also be read from the same interface which is used for programming. This is called readback. The data can be copied to another memory to be used especially in space applications to fix single bit errors which are caused by high energy charged particles [10, 11, 12]. Readback data represents currently running configuration and can be compared with the original bitstream to detect any error.

Configuration memory can also be used to store states of logic elements (FF, BRAM, etc.) to perform context saving. Each state information can be read back to external memory through a configuration port. This is called readback capture [13]. The latter word ‘capture’ is added to readback because state information of a running circuit is captured from each logic element. Later, the captured data is used to create a new bitstream for context restoring.

2.2 Design Considerations For Floorplanning

Multitasking in single-core GPP is accomplished by time-sharing for each task. A user cannot observe switching between tasks because the switching occurs in a very short time. However, time-sharing for each task creates a delay to execute the given tasks. Multi-core processors can overcome this issue if the tasks are divisible to all processing cores. This is called true multitasking on GPP [14]. Similarly, running multiple parallel tasks on processing elements in a reconfigurable hardware is preferable since the performance would be increased. True multitasking can be accomplished in this way in a reconfigurable system.

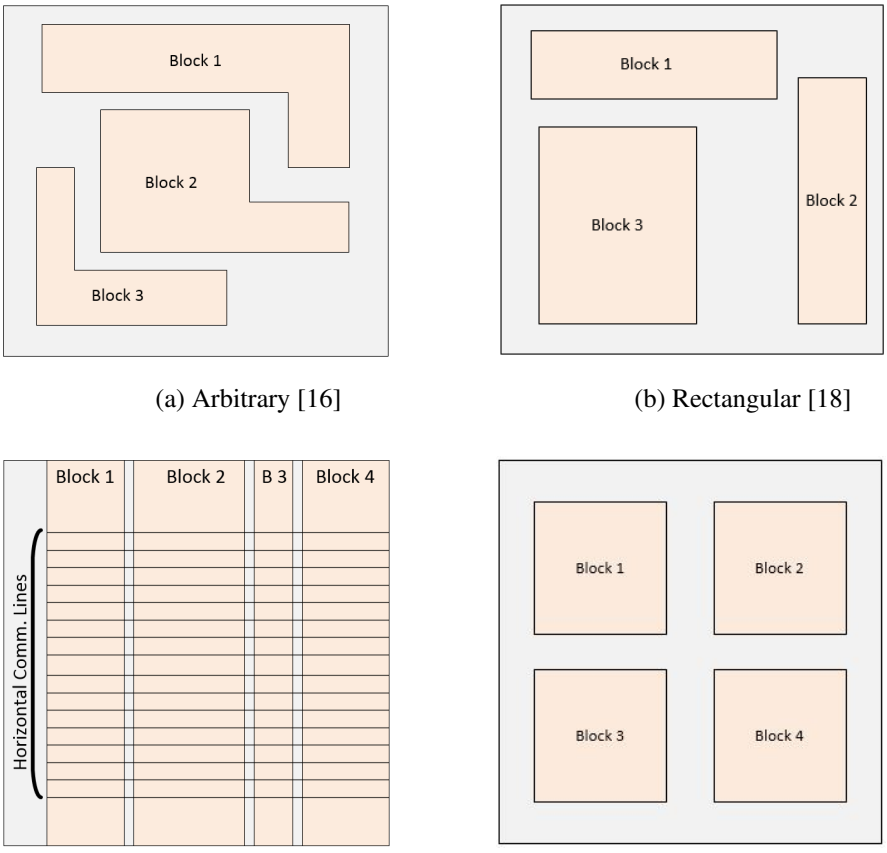
Partitioning of logic elements in a reconfigurable system enables designers to load multiple tasks to the selected partitions to run them concurrently. Utilization of the device can be increased by good partitioning. Considering a real time system running multiple tasks and scheduling the arrival of new tasks, if all tasks are executed without exceeding their deadlines and no task rejection occurs, it can be said that reconfigurable logic is capable to utilize all tasks. However, if a bad partitioning technique was applied, then the designer would choose a bigger device which costs much higher. As a result, partitioning is one of the key factors for high utilization [15].

In partially reconfigurable systems, floorplan of an FPGA can be designed to have several partitions which have different shapes and constraints. The shape of the partially reconfigurable area affects the overall utilization and fragmentation since the routing depends on the boundary of these areas. There are four different partitioning techniques to be considered;

- Arbitrary shaped partitioning
- Rectangular shaped partitioning
- 1D partitioning
- 2D partitioning

Arbitrary shaped partitioning enables multiple tasks to run together by getting all partitions closer to each other. This can be done by filling big spaces with smaller

tasks. Thus, maximum allowable frequency can be increased by this way. However, due to arbitrary shape of the partitions, utilization and fragmentation can be problem if running tasks for each partition cannot be placed and routed in an appropriate way compared to regular placement [16, 17]. An arbitrary shaped partitioning example is depicted in Figure 2.2a.



(a) Arbitrary [16]

(b) Rectangular [18]

(c) 1D [19]

(d) 2D [1]

Figure 2.2: Floorplanning

Rectangular shaped partitioning limits the circuit with a rectangular shape which has enough size to place and route the assigned circuit. In this methodology, reconfigurable partitions have higher utilization and lower fragmentation due to completely fitting circuit. However, this architecture is not appropriate for running multiple tasks together in any partition due to heterogeneity when task size is larger than the available empty partitions [18]. A rectangular shaped partitioning is shown in Figure 2.2b.

Both rectangular and arbitrary shaped partitioning is not suitable to have a communication data path among tasks and between reconfigurable logic to external system memory. The communication requirement is an application specific issue. If there is data transaction in system requirements, we need a partitioning where all partially reconfigurable tasks are ordered to communicate with each other and other system memory elements. This gives the design flexibility so that each partial block can load input data from system memory or output interface of another partial block.

One-dimensional (1D) partitioning method is applicable to reconfigurable devices to meet the need for communication. The architecture with 1D placement is to locate partitions repeatedly on the horizontal direction having all height of FPGA. A reconfigurable architecture with communication media is designed by Kalte and Porrmann [19]. In order to have communication infrastructure between dynamically changeable tasks, they implemented a horizontal bus to be connected to whole partitions. The bus is homogenous everywhere so that relocation of the tasks from one slot to another does not affect the behavior of relocated task. Design of reconfigurable 1D hardware task placement is shown in Figure 2.2c.

To have communication among all tasks in 1D placement comes with two disadvantages; fragmentation and maximum allowable frequency. Due to vertical placement of tasks in one slot, resources are not placed efficiently to get a reasonable routing. This results in fragmentation and long routing delays. On the other hand, the communication media is affected with long routing delays because all reconfigurable blocks have connections to other blocks. Therefore, communication line bandwidth is limited by long wiring delays.

Considering all issues with partitioning techniques, two-dimensional (2D) partitioning brings more flexibility to reconfigurable hardware floorplan design [1]. Reconfigurable logic is divided both vertically and horizontally and each partition has a border with static logic which contain communication structure and other necessary logic. 2D placement is shown in Figure 2.2d. A task which is implemented to run on one of the partial blocks can be implemented to run on other blocks as well. The partition size can be selected as the biggest circuit that will be executed. This depends on the user's application.

The partitioning methods mentioned above should be supported by both hardware and the software tools to realize them on hardware.

2.3 Architecture of Xilinx 7-Series FPGAs and SoCs

History of FPGA device starts with programmable logic devices (PLDs) around early of 1970s. Xilinx introduced the first product called XC2064 at 1984 which contains only 64 logic blocks [20]. Each block contains two 3-input LUTs and one register. It was manufactured with 2.5micron technology. General architecture of XC2064 is shown in Figure 2.3.

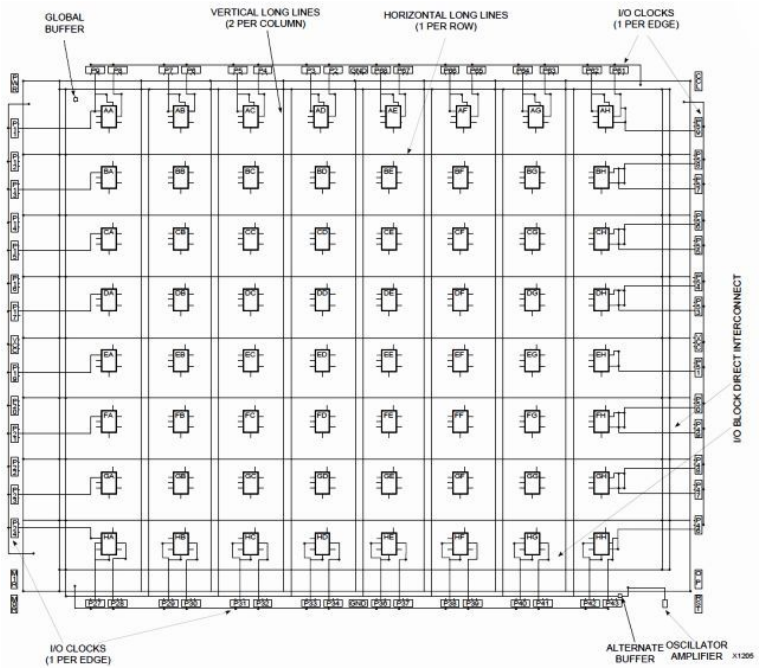


Figure 2.3: XC2064 Logic Architecture

Today, number of logic sources in an FPGA like Virtex-7 device can be up to 136,900 slices which contain 547,600 6-input LUT and 1,095,200 FFs [21]. The fabrication of Xilinx 7-Series FPGAs and SoCs is 28nm technology [22] and based on Static RAM (SRAM) configuration memory. The architecture of Xilinx 7-Series FPGAs are different from the previously released architectures as depicted in Figure 2.4.

A CLB contains two slices and each slice is composed of four 6-input LUTs and eight

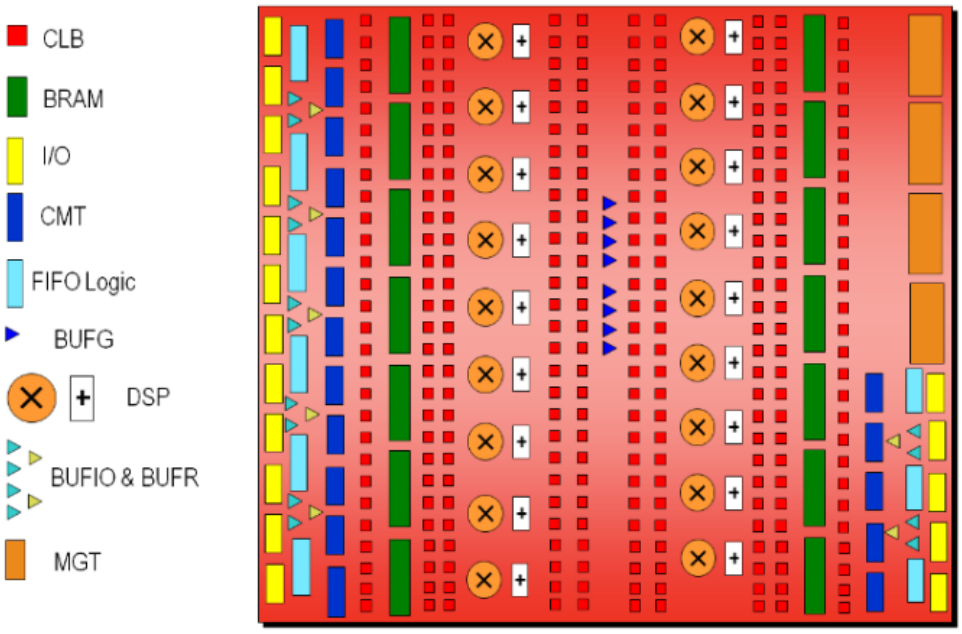


Figure 2.4: A Small Representation of Xilinx 7-Series Architecture

storage elements in Xilinx 7-Series FPGAs [21]. Block RAMs (BRAM) and Digital Signal Processing (DSP) slices are located between CLBs to make a connection easily. These three configurable elements CLB, BRAM and DSP can be inside partial blocks. It means that they can be reconfigured with a partial bitstream dynamically. Other logic elements can only be configured by full reconfiguration. Architecture of a slice in the CLB is shown in Figure 2.5.

Xilinx has also released a Zynq-7000 System-On-Chip (SoC) IC which is composed of FPGA and ARM processor running together on one die. The advantage of the hard processor in comparison to soft processor like Microblaze is to run at higher frequency and it contains DDR controller, DMA and have more peripherals available to use. When hard processor is used, no logic is required to be configured so that it does not consume configurable logic. ARM side of the SoC has the full control over the FPGA part. AXI based data lines can be used to have communication path between processor system (PS) and programmable logic (PL) side.

Configuration bitstream of SRAM-based FPGA can be downloaded with JTAG or SelectMAP configuration ports. It is called off-chip configuration as already mentioned in Chapter 2.1.5. There are also two options to configure an FPGA; Processor

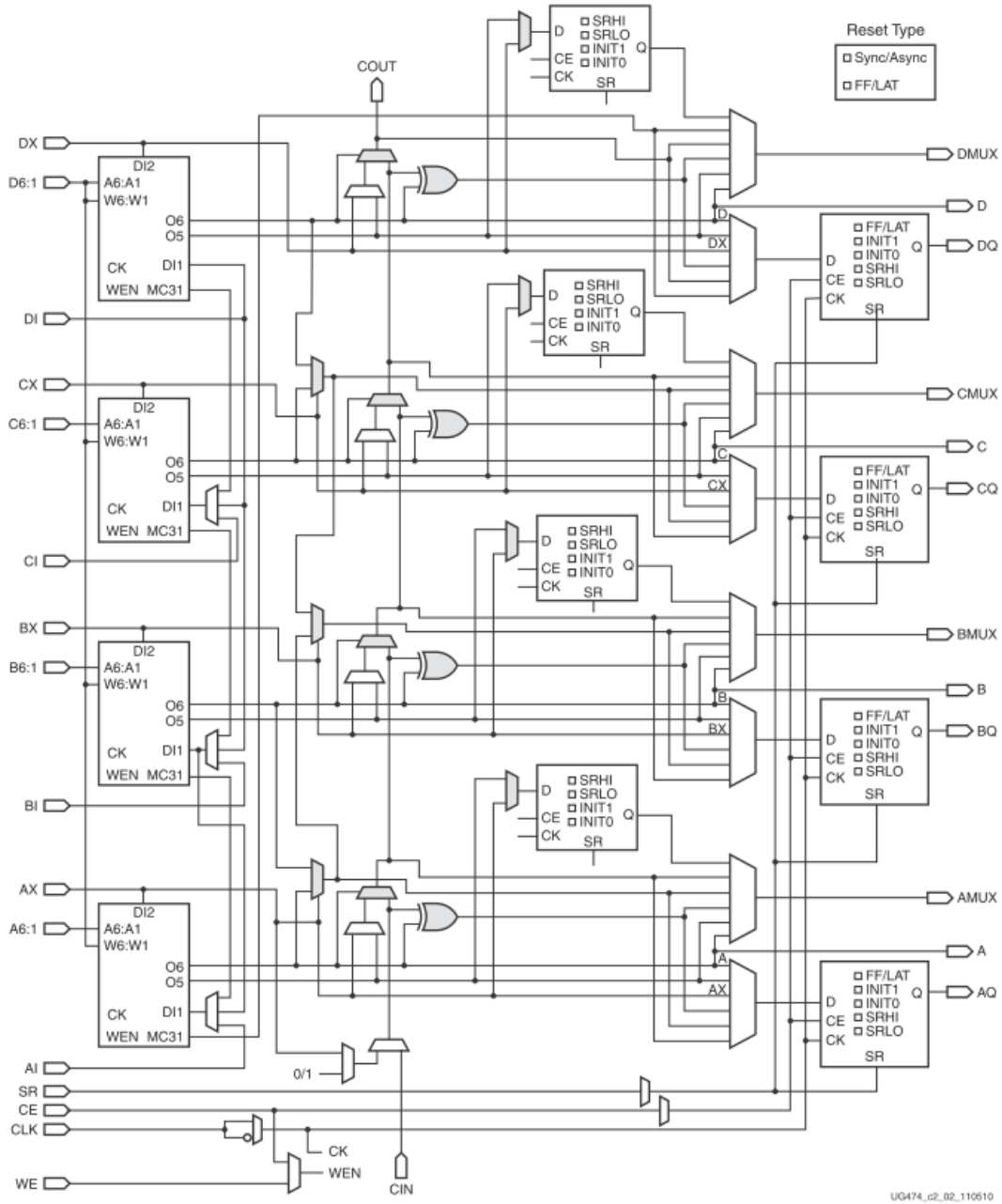


Figure 2.5: SLICEM in CLB Logic

Configuration Access Port (PCAP) and Internal Configuration Access Port (ICAP). If one of these two ports is used it is called context configuration. PCAP is only used in Zynq-7000 SoCs. It is capable of full and partial reconfiguration. ICAP can only be used for partial reconfiguration because it disrupts its own data line while full reconfiguration.

2.4 Bitstream Structure

The relation between the bitstream and the configuration memory of an FPGA is very structural. The content of the bitstream is composed of configuration packets and raw circuit configuration bits. Xilinx 7-Series FPGA architecture has addressable configuration frames. Each frame needs 101x32 bits data to be configured. The content of the frame depends on the frame address so that each bit can corresponds to one FF, BRAM and LUT, etc. Each bit value in the bitstream is directed to the configuration memory of the related logic by this way.

Bitstream structure is very important for designers to establish a context-switchable partially reconfigurable FPGA system because bitstream manipulation is required to restore the previous circuit on an FPGA. For this reason, the configuration packets mentioned above is worth examining. There are two types of configuration packets in the bitstream of 7-Series Xilinx FPGAs [23].

- **Type 1 Packet**: It is used for register reads and writes. It contains a 32-bit header and payload. The bit-ordering for its header is shown in Figure 2.6.

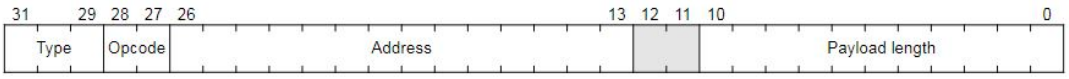


Figure 2.6: Packet Header for Type 1 Packet

Header type represents the type of packet so that "001" corresponds to Type 1. Opcode represents whether it is read or write operation, "01":Read, "10":Write. No operation (NOP) is also used for synchronization of the configuration cycle, "00":NOP. Register address space is limited only with 5 out of 14-bits config-

uration register address. Configuration registers will be discussed later. The number of 32-bit payload data coming after the Type-1 header is specified by the payload length section of the header.

- **Type 2 Packet**: It follows the Type 1 packet header and no address is specified as shown in Figure 2.7.

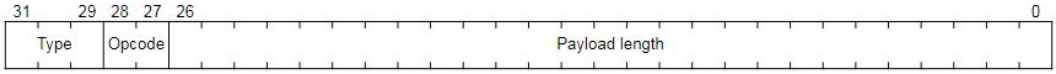


Figure 2.7: Packet Header for Type 2 Packet

Its header has 32-bit word. Header type equals to "010" which corresponds to Type 2 packet. Opcode is the same with the one in Type 1. The payload length portion points the number of 32-bit payload word coming after this header. It is used for long writes.

Xilinx 7-Series FPGAs have internal 32-bit configuration registers which is addressed by Type 1 packet. These registers are low-level registers to control the configuration sequences or to obtain information about the device. Some important registers are as follows;

- **CRC**: This register is used to compare two CRC values. One of them is calculated on the chip from each 32-bit word during configuration and the other one is written in bitstream while creating bitstream. This register is used in CRC check for the integrity of bitstream when a write operation occurs on it. This check can be disabled when bitstream manipulation is required.

- **FAR**: Frame address register uses bitstream as input so that configuration is loaded on the specified frame. It is automatically incremented by the device to fill all specified areas.

- **FDRI**: Frame data register input. Both commands and payload data are written to this register to fulfill the configuration. 32-bit data is registered in each clock cycle.

- **FDRO**: If the configuration memory is read back, it outputs the configuration data corresponding to the frame address written in FAR.
- **CMD**: Command register takes the basic configuration commands such as write configuration data (WCFG), read configuration data (RDFG), begin the startup sequence (START), shutdown the device (SHUTDOWN) and capture the last state of registers (GCAPTURE), etc.
- **MFWR**: Multi-frame write register is used when the bitstream is compressed. Bitstream generation tool adds a write operation to this register to write the same data to multiple frame addresses. Normal bitstream does not include MFWR register write command.
- **IDCODE**: Each device has its own ID code. For verification of the bitstream, internal register is compared with the value inside the bitstream. If they are not the same, configuration sequence stops in order not to configure with wrong data.

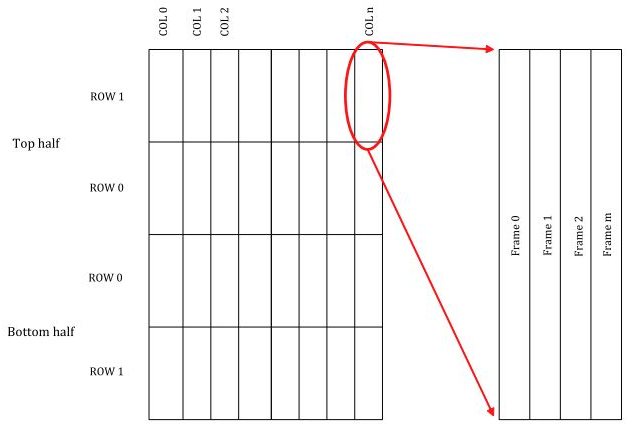


Figure 2.8: Addressable FPGA Surface

Partially reconfigurable area of an FPGA is addressable by its frame address. When a partial reconfiguration process begins, only the pre-selected portion of the surface is programmed. FAR register is used to give a start address to the configuration mechanism. The address can be described with FPGA's configuration memory architecture. The 7-Series devices are divided into two halves, the top and the bottom. 32-bit FAR register has one field for block type and four address types: top/bottom bit, row ad-

dress, column address and minor address. Figure 2.8 and Figure 2.9 show addressable FPGA surface and bit indexes of the FAR register content successively.

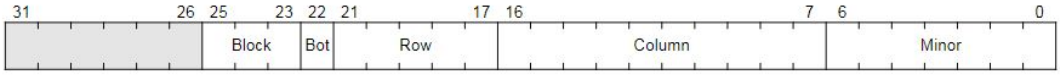


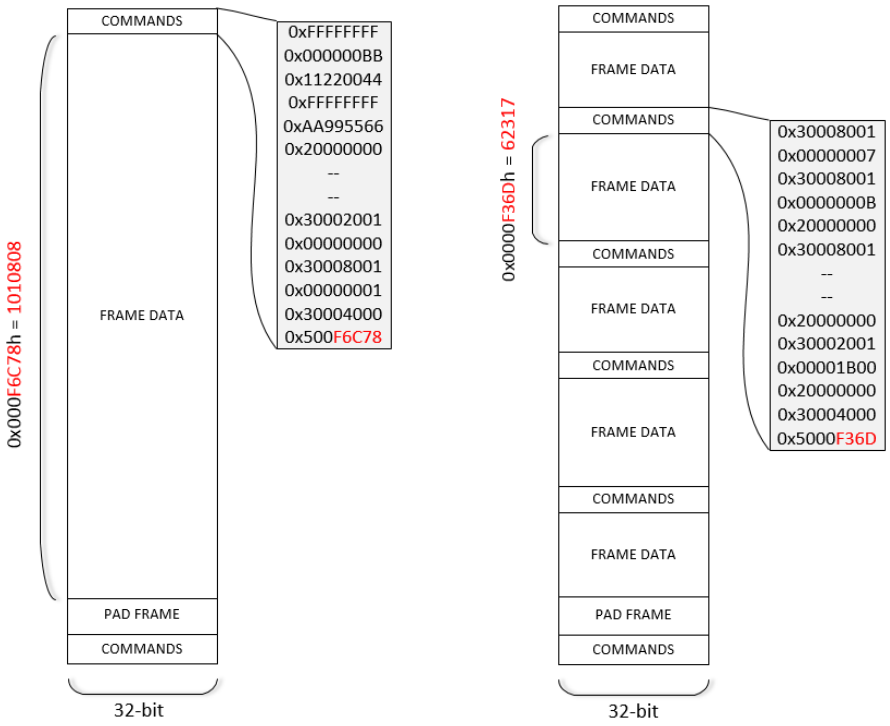
Figure 2.9: FAR Register Content

Block type is the type of logic elements such as CLB, IO, CLK, BRAM and CFG_CLB. Top/Bottom bit represents which half of the FPGA is configured. It is either top-half (0) or bottom-half (1). Both top and bottom of the FPGA surface is divided with rows and each row is divided with columns. Each column location contains many frames which are also addressable with their minor address. The frame address written in the bitstream should be valid for the selected FPGA or SoC package. Otherwise, the configuration fails. Vivado and other software tools creates the bitstream with respect to project options which has package information of the FPGA used.

Xilinx provides different file formats for bitstream creation. Each file format is used for different programming interfaces. Most common file formats are as follows:

- **BIT**: This file format is used to program FPGA through a software tool via JTAG programming interface. It contains a textual information header about the bitstream. The following data is binary configuration data.
- **RBT**: This file format is the same with BIT format except that data is not in binary, it is in ASCII representation where each 32-bit is given with textual 0s and 1s. It can only be used to program FPGA with a parser since it is a text file.
- **BIN**: This file format is used to program FPGA through SD Card via ICAP, PCAP and optionally SelectMAP interfaces. It contains only raw 32-bit binary configuration data so that there is no need for software tool to parse it.
- **MCS**: This file format is used to program FPGA through a FLASH memory or PROM via SPI or BPI programming interfaces. The bit ordering in the file is dependent on the architecture of memory device. Software tools provide an interface to choose an option for each type of commercially available memories.

Most suitable file format is the BIN format for a partially reconfigurable system since it is coded in binary and contains only commands and payload. It is easily readable from an external memory such as SD Card. The size of the bitstream file is proportional to the number of logic units in an FPGA. A full bitstream contains all frame data while a partial bitstream contains only related frame data. A general structure of a full and partial bitstream is shown in Figure 2.10.



(a) Full Bitstream Structure

(b) Partial Bitstream Structure

Figure 2.10: A Full and Partial Bitstream Structure of Xilinx 7-Series FPGAs [24]

The payload size can be calculated with the given number of 32-bit configuration lines in a full bitstream. The command 0x500F6C78 given in Figure 2.10a is in Type 2 packet format so that only 0xF6C78 is meaningful to the user as word count. The number is equal to 1010808 in decimal format. The number of lines after that command is 1010808x32-bit payload data which is equal to 4043232 bytes. Additionally, the number of configuration frames can be calculated using this number. Since all 7-Series Xilinx FPGA's have 101x32-bit configuration bits for each frame address,

1010808/101 = 10008 gives the number of frame addresses belonging to the FPGA package used in the design.

Partial bitstream includes CLB and BRAM frames as separate. FAR register cannot jump from one frame address to another, but it increments one by one. It should be manually set by bitstream commands so that it can jump from one CLB frame address to another BRAM frame address. Internal commands are used to do that. In addition, number of frame data given in Type-2 packet format is as described in full bitstream. The command 0x5000F36D given as an example in Figure 2.10b represents that there are 62317 word payload data. Number of frames in that frame data interval is equal to 62317/101 = 617. Each interval includes different number of frames and starts with a different frame address depending on the design.

2.5 Concepts in Software Tool

The overall system design is based on both statically and partially reconfigurable areas. The full bitstream structure mentioned before involves whole surface of the FPGA. Partially reconfigurable areas can only be programmed with the corresponding partial bitstream file. During a design phase, surface partitioning needs to be done to determine the details of the partial bitstream such as frame addresses and number of logical sources.

Some concepts in the software tool for a partially reconfigurable system is given as follows: [17]

- **Bottom-Up Synthesis**: The design starts with the synthesis of each module independently. It is called out-of-context (OOC) synthesis in Xilinx Vivado tool. Overall design uses these modules as black boxes ensuring that no optimization is performed across inputs and outputs of the related module. In this way, synthesized module can be used in multiple projects.

- **Configuration Frame**: FPGA configuration memory consists of many frames. A frame is the smallest addressable segments of the FPGA. A frame can contain one of these elements, CLB, Block RAM and DSP.

- **Partition Definition**: It defines the reconfigurable modules which run on the reconfigurable partition. Each module has the same I/O ports.
- **Partition Pin**: It is the logical and physical connection point between static and reconfigurable logic.
- **Reconfigurable Frame**: Each logic cannot be reconfigured independently. It defines the smallest reconfigurable region on the surface of FPGA.
- **Reconfigurable Logic**: It is a simple logical element in a reconfigurable module. Partial bitstream may change the state of this logic.
- **Static Logic**: It is a simple logical element running outside reconfigurable partition. Partial bitstream cannot change the state of this type of logic and it is active during partial reconfiguration.
- **Reconfigurable Partition**: A logical section of the device and it is selected by the designer. It defines all dedicated modules as reconfigurable. The expression HD.RECONFIGURABLE = TRUE inserted into the design file indicates that the partition is reconfigurable. Otherwise, it cannot be used for partial reconfiguration.
- **Reconfigurable Module**: It is a circuit or HDL description that can be implemented in reconfigurable partition. Each module must have the same partition pins.

Some architectural partitioning methods are already discussed. In fact, all of the FPGA surface consist of basic logic elements (BLE). Surface partitioning is affected by heterogeneity of the surface and other architectural reasons. Therefore choosing an option from discussed partitioning methods is not directly applicable for Xilinx 7-Series FPGAs.

Some of the restrictions and permissions for floorplanning is given as follows:

- Xilinx restricts the content of a reconfigurable partition such that clock circuits (MMCM and PLL), I/Os, mult-gigabit transceivers (MGTs) and hard blocks like ICAPE2, CAPTUREE2, etc. must be inside static region, hence they cannot be reconfigured.

- Partition pins are placed by Vivado with default option. If the number of pins is higher than that a partition can contain due to its small size, this can cause some routing and timing problems and eventually not a routable design.

- Upper and lower boundaries of reconfigurable partition must align vertically to the clock region boundaries. Although the rectangle can be drawn in anywhere on the surface, the global reset signal is not applicable to randomly drawn partition. Applying global set reset (GSR) signal is highly recommended because it resets the reconfigured circuit to the initial state after configuration completes and keeps all logical elements in reset until configuration is done. If not used, additional decoupling circuit is required for clocks and other inputs to prevent internal logic from changing unintentionally. RESET_AFTER_RECONFIG property must be checked so that GSR feature is embedded in partial bitstream to shut down only related area during configuration. Reconfigurable partition drawn not suitably for the GSR feature is shown in Figure 2.11. Pink rectangle shows selection of surface without RESET_AFTER_RECONFIG property. Yellow grids show the area affected by the GSR reset if it is applied. Upper and lower boundaries of the yellow rectangle are attached to a clock region boundary.

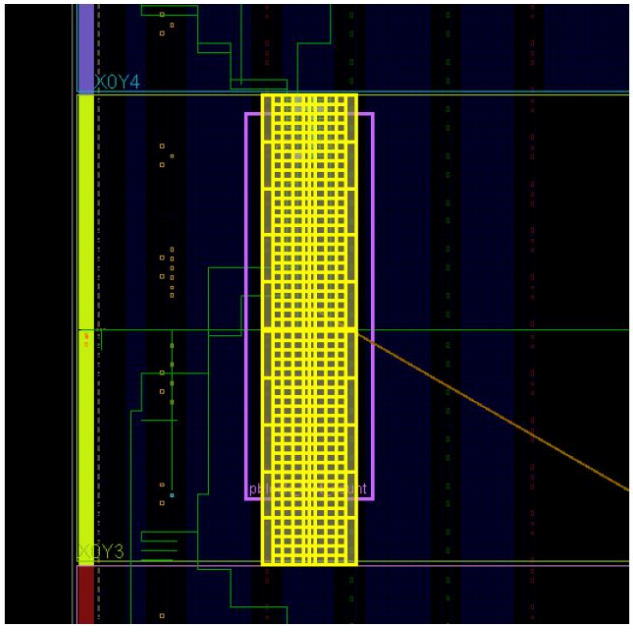


Figure 2.11: Unsuitable RP for GSR Feature [17]

- Partition size cannot be increased in the vertical direction which passes to the other clock region. However, the size can be increased in horizontal direction both in the same clock region and the other neighbor clock region. When the size increases, number of available logical sources increases. The need for logical sources depends on the application. The size and location may also be changed with routing and timing problems.

- Two partially reconfigurable regions cannot be overlapped. If it was possible, a configuration frame would be in two different partitions so that the circuit behavior would be unstable.

- Left and right boundary of the partition must be between two CLBs or between CLB and BRAM or between CLB and DSP as depicted with yellow arrows in Figure 2.12. This constraint is required to route signals between static and reconfigurable region. If the SNAPPING_MODE property is checked, surface partitioning tool does not give DRC error since it automatically adjusts boundaries. In Figure 2.11, it adjusts left and right boundaries and reduce the size of the RP. SNAPPING_MODE is also used to remove the non-reconfigurable hard blocks from the reconfigurable part.

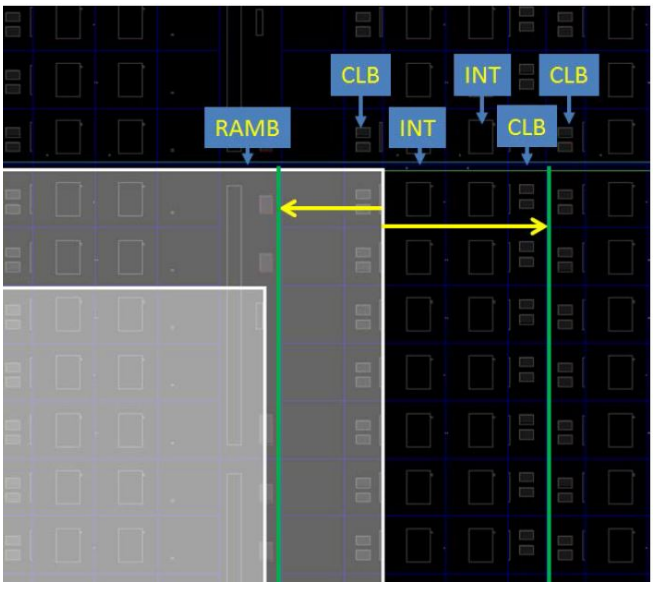


Figure 2.12: Adjustments of RP to the Suitable Column [17]

2.6 Context Switching

In [24], context switching time needed for partially reconfigurable system is given by the sum of context saving, restoring and bitstream manipulation time. Context saving and restoring is a time-consuming operation on reconfigurable devices in comparison to GPP devices. To configure all logical elements to a pre-determined state differs from an operation like loading some values to a couple of registers. The time overhead changes with respect to configuration speed and bitstream size. It cannot be eliminated fully by just increasing the configuration clock frequency and decreasing bitstream size with some compression techniques, but it can be reduced to some level [25]. A method to fully eliminate the reconfiguration overhead is proposed by Say and Bazlamaçcı [1]. They state that the next reconfiguration can be done on an available reconfigurable partition, while the previous configuration executes on another partition. By this way, switching from one execution to another is performed with no time overhead. However, performance of this method depends on surface partitioning, task scheduling and total number of tasks in the system.

In multi-context FPGAs, the time overhead for reconfiguration can be reduced to nanoseconds due to their architecture. However, it is applicable if all configurations are loaded into the on-chip memory. If a new reconfiguration is required rather than one of ready configurations, there must be some time overhead. To reduce time overhead in multi-context FPGAs, virtualization of partial reconfiguration can be performed as stated in [25].

2.6.1 Context Restoring Time

ICAP primitive is used for partial reconfiguration in Xilinx FPGAs. Recommended clock frequency for ICAP is 100MHz and configuration data bus width is 32-bits. Theoretical configuration speed is 381.46 MB/s. Xilinx offers a configuration controller IP called HWICAP which utilizes ICAP as a submodule. The speed of the configuration with HWICAP is 14.6 MB/s which is much slower than the theoretical speed [26]. Readback operation with HWICAP is limited with configuration registers due to small FIFO.

PCAP can also be used in partial reconfiguration. Reconfiguration speed of PCAP is measured as 126.8 MB/s in Xilinx's official application note [27]. Liu et al. [26] reduced timing overhead with a different ICAP controller IP called BRAM_HWICAP. The ICAP hard macro is fed with the output of BRAM so that the controller avoids the time-consuming data transactions. It provides 371.4 MB/s as throughput. It gets close to the theoretical value. However, it consumes many BRAM sources to hold the configuration bitstream.

Vipin et al. [28] designed a DMA-based partial reconfiguration mechanism called ZyCAP on Xilinx Zynq SoC employing ICAP hard macro. Partial bitstreams are stored at DDR memory. ARM side of the Zynq SoC is connected to a DDR memory through a DDR controller. A DMA engine is implemented in FPGA side of the SoC and it is set to read a partial bitstream to transfer it to the ICAP controller. Clock frequency is selected 100MHz as default ICAP frequency and ZyCAP transfers a partial bitstream with the theoretical speed of 381.47 MB/s assuming that no software and hardware overhead exist. DMA-based ICAP controller IPs are also used in other configurable FPGAs for different circuit architectures [29, 30, 31, 32]. They provide the best throughput among many others by staying at the safe side in terms of ICAP clock frequency. Overclocking the ICAP clock is not supported in Xilinx 7-Series FPGAs because 100MHz is the maximum allowable frequency [17]. There are some overclocking experiments conducted on older Virtex-5 FPGAs so that ICAP can run at up to 550 MHz [33]. However, this is not a case for 7-Series FPGAs.

2.6.2 Context Saving Time

Context saving on reconfigurable devices is required for preemptable multitask execution environments. Instantaneous states of logic elements in a reconfigurable partition running on an FPGA is saved to restore and run any time after saving. To discuss time overhead for context saving, some readback methods should be considered. There are two methods to capture the states of a running task:

- Configuration Port Access (CPA): Each state can be accessed through a configuration port like ICAP. There can be a configuration port controller structure

for readback procedure. Bitstream structure should be known in detail to copy state bits to the new bitstream.

- Task Specific Access Structures (TSAS): Each task has its own state saving mechanism in it. Task state is accessed through a pin implemented on the task. It captures only the state-related data so that no redundant data is read. However, it needs extra logic and reduces the utilization of the device. Timing and routing errors can appear on the design [34, 35]. The circuit can be restored after assigning state information to the state-related logic.

In both methods, reading a BRAM content results in the same time overhead [29]. Eliminating reading time overhead for the BRAM content, TSAS-based method is more data efficient since only the related logic is saved instead of all logic in the module. However, one drawback of TSAS-based readback mechanism is to insert state saving logic for all registers which reduces hardware task utilization. In addition, some commercial IPs can be locked by the provider in order not to give HDL design to the user. In that case, state saving logic cannot be inserted into the design. In CPA method, the number of readback data is very large compared to TSAS method. However, Jozwik et al. [29] states that a high throughput ICAP controller can overcome the issue of reading large number of redundant data by reading only the related frames so that it can have better performance compared to TSAS method. In addition, BRAM contents should be read back by disabling the main clock source, otherwise, BRAM contents may be changed unintentionally when configuration clock is driven into BRAMs [36].

Partial bitstream created by the software tool configures all configurable frames in the reconfigurable partition even if no circuit is available in some frames. If those frames are configured to do nothing through bitstream, then there is no need to readback those frames. The address information of filled frames can be extracted from the bitstream or logic location file (.ll) [37] and only related frames are used for readback. Time overhead of readback can be reduced by this way. As a result, CPA method is also preferable for context saving procedure in terms of time overhead and hardware utilization.

PCAP has a functionality of reading back configuration data to DDR memory. Stoddard et al.[12] observed that readback data throughput is 145 MB/s when PCAP clock is 100MHz which is the default frequency. Duhem et al. proposed an ICAP controller called FaRM which has 95 MB/s readback throughput [38]. For low time overhead readback mechanism, MiCAP-Pro system is proposed by Kulkarni et al. [32]. It is a DMA-based system mentioned before which resembles the ZyCAP model with extra readback feature as depicted in Figure 2.13. It is implemented in Xilinx Zynq SoC. The data throughput is 272 MB/s for readback running at 100MHz. MiCAP-Pro is not efficient for throughput in comparison to ZyCAP since it has I/O buffers which delays the configuration data streaming. A FIFO is placed before the ICAP primitive so that it slows down the throughput. It can be enhanced to reach the rate of ZyCAP for readback option.

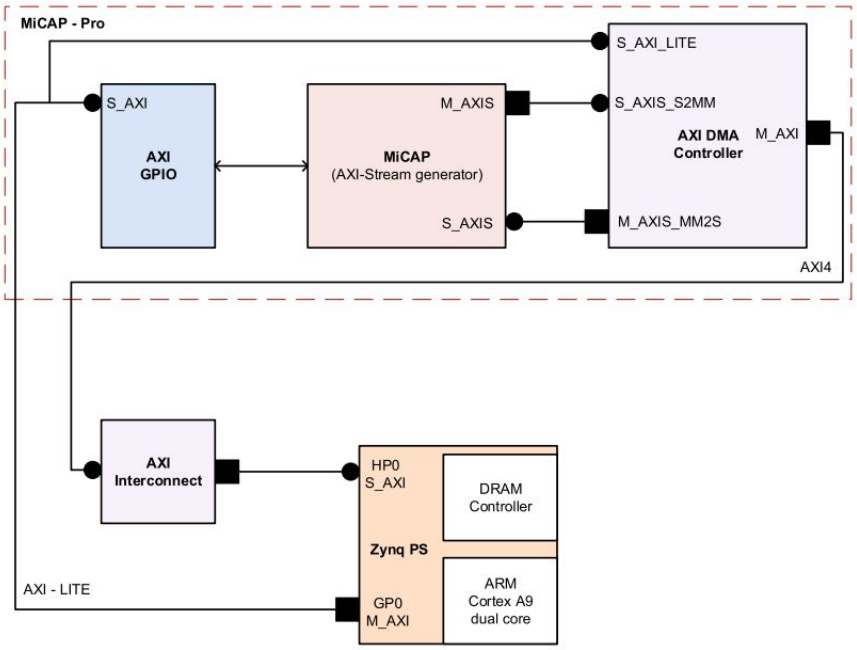


Figure 2.13: MiCAP-Pro Readback and Reconfiguration Model [32]

2.6.3 Bitstream Manipulation Time

To make a bitstream manipulation, state information of FFs and content of BRAMs should be written into the correct location of the bitstream. While writing, there can be some modifications on captured data. Morales et al. [39] state that manipulation

should be done with a mask file provided by Xilinx ISE tool. New state of a bit is determined by the formula $S_{new} = captured\,bit * mask + initial * (not\,mask)$. However, in 7-Series configuration user guide [23], Xilinx states that a '1' in the mask file indicates "don't care" while a '0' indicates "important" to capture. Using the formula, if a captured bit is important, it is multiplied by zero and initial value is loaded as a new state. If a captured bit is not important it is multiplied by one and initial bit is not loaded. Considering the result of this formula, it loses the captured data if the data is important. Therefore, it cannot be applied for 7-Series FPGAs and SoCs.

Bitstream manipulation time is important for preemptable circuits. It depends on where the manipulation is performed. Processor architecture, clock rate and algorithm change the total elapsed time. Morford presents BitMaT tool to manipulate a Virtex-II FPGA bitstream with a speed of 304 KB/s on a Pentium 4 machine [40]. BITMAN bitstream manipulation tool perform a manipulation which needs low-level details about sources and routings. It can run on ARM Cortex-A9 processor and it manipulates one CLB state in 94 us and one BRAM content in 229 us [41]. Number of bytes manipulated per second with BITMAN tool cannot be calculated directly since the throughput is given for only one CLB and BRAM content.

CHAPTER 3

BEHAVIORAL MODEL OF THE CONTEXT SWITCHING SYSTEM

It is already discussed that GPP based computing platforms provide design flexibility and high level programming environment for designers. GPP based personal computers meet low level requirements for average users without considering power consumption, reconfigurability, performance and operating system support. In enterprise computing, high performance criterion is the main goal. Multi-processor systems can increase the performance to a considerable level with a power consumption penalty. On the other hand, it can lose operating system support. Reconfigurable devices have the facility to increase the performance by parallelizing computing tasks with extra development time and low power consumption. There is a trade-off between reconfigurable and GPP based systems.

Hybrid SoC architecture is composed of a GPP and reconfigurable hardware. It eases the development of a system employing many peripherals. The performance also increases due to short distance/high throughput communication lines between GPP and reconfigurable part. A soft-processor implemented in a reconfigurable device can also be used to get a hybrid architecture, but a hard-processor has much more performance due to higher clock frequency and internal structure. In addition, a hard processor can increase the utilization of reconfigurable part because of its static structure.

The overall system architecture which can run dynamically context-switchable tasks is shown in Figure 3.1. It contains both GPP and reconfigurable hardware. Reconfigurable hardware is capable of partial and full reconfiguration and hence, dynamically switching between hardware tasks on the reconfigurable part can be performed with a control mechanism running in GPP.

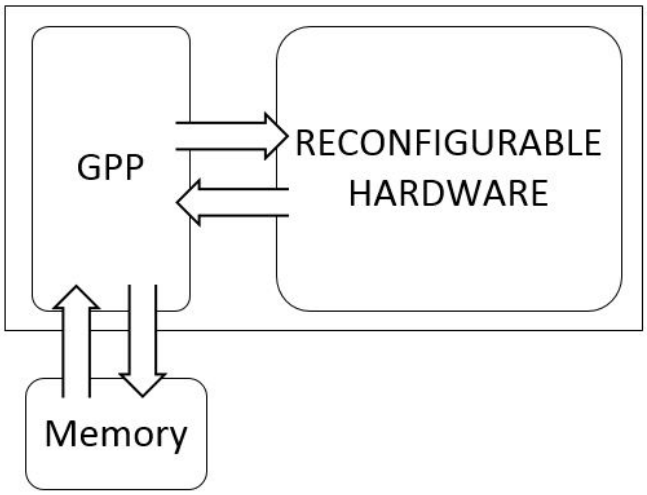


Figure 3.1: System Architecture for Dynamic CS

The system memory is connected to GPP and it is used to store partial and full bit-streams. Xilinx has commercially available SoCs having the architecture shown in Figure 3.1. Xilinx 7-Series, Ultrascale, Ultrascale+ Zynq SoCs are suitable for the context-switchable hardware design. It can be used in an embedded system which needs the criteria mentioned above.

Some of the design concepts needs to be examined to run dynamically context-switchable system on SoC architecture. Software execution on GPP and hardware execution on FPGA can be done together. Partial reconfiguration is performed on FPGA to switch between hardware executions.

3.1 Operating System Model

In personal computers, operating system (OS) handles all low-level executions without showing processing steps to the user. Peripherals are used to give inputs to the system and user does not care about the rest. OS must be capable of executing given inputs with a good performance by working in harmony with other hardware parts.

In an embedded SoC system, OS must handle all given tasks by cooperating with both GPP and reconfigurable hardware. Tasks can consist of a software part and a hardware part. Depending on the system requirements, a context switch can be required

to complete given tasks. Context-switch in a GPP is executed by saving internal processor registers to a memory to run at another time and running other tasks instead. Context-switch in reconfigurable hardware is executed by reading back states of FFs and BRAM contents located at partially reconfigurable area and copying them to a memory and reconfiguring the related area for other circuits.

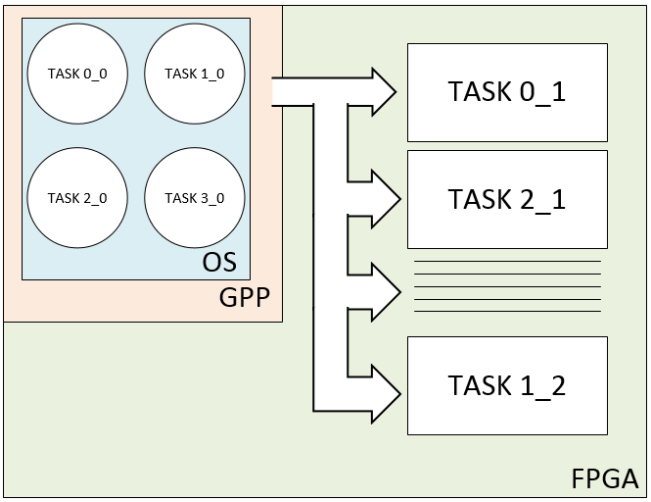


Figure 3.2: Embedded OS Running on SoC

An embedded OS should do these operations without user intervention. Reconfigurable part of SoC should be managed by OS as if the system contains only a hard processor. It can be programmed to respond to all requests including real-time tasks. The user does not see any low-level operation running in between GPP and reconfigurable hardware. Thus, flexibility and performance of the whole system can be increased. A general model for embedded OS handling both software and hardware tasks is depicted in Figure 3.2.

There are some available operating systems in the market for embedded systems such as FreeRTOS, VxWorks, etc. BareMetal OS can also be used for single thread.

3.2 Task Model

The task in the hybrid reconfigurable system can be composed of both software and hardware executions. Software task can run individually or can manage hardware task executions while running.

Spatial and temporal partitioning requirements should be analyzed during design phase. If a hardware task cannot fit into a reconfigurable partitions, it can be divided into two tasks if possible, which can run on different reconfigurable partition. Partition size is also important, and it can be set for the largest hardware task to fit into. In contrast to that, two or more small sized hardware tasks can be combined to run together in one reconfigurable partition to reduce fragmentation. This depends on the application.

All hardware tasks can run on any partition so that if a hardware task cannot find an available partition for a long time, an equivalent software task should be available so that it can be executed on GPP before deadline, if there is.

Inter-task communication between consecutive tasks due to data dependency should be provided by the processor. Output data of one hardware task cannot be directly transferred through a separate bus to the consecutive task. There is no direct or addressable data line between running tasks. All tasks are connected to each other through processor.

Context-switch operation in reconfigurable computing can be considered in preemptive and non-preemptive manner. If a software task is preempted, it can be saved to memory to run on another time. The preemption operation for a software task is relatively easy as mentioned in previous chapters. On the other hand, hardware task preemption requires capturing and saving states of CLBs and content of BRAMs to a memory. Hardware task can be preempted with the conditions given below:

- Hardware task must contain only CLB and BRAM sources. State of DSP sources cannot be saved and captured so that hardware task containing a DSP source cannot be preempted.
- Hardware task must have higher priority than other hardware task running already in one of reconfigurable partition. This condition depends on the application specifications.
- If a hardware task execution gets close to the end of its execution so that remaining execution time is lower than context-switching time overhead, it should not

be preempted. This condition is also application dependent, but user should avoid for unnecessary context-saving and restoring which results in time wasting.

For overall system, an example flow of software and hardware task execution with a context switch operation on two reconfigurable partitions is shown in Figure 3.3. T_s^{11} represents the first sub-task of task 1 running in GPP. 's' stands for software and 'h' stands for hardware. T_h^{12} represents second sub-task of task 1 running in hardware and so on. After running T_h^{12}, T_h^{13} and T_h^{14} continue on hardware. They are stopped before the end of their executions and a context saving occurs at 4$^{\text{th}}$ cycle by saving task 1 and a partial reconfiguration occurs for task 2. After the execution of T_h^{22} and T_h^{23}, a context restoring occurs at 5$^{\text{th}}$ cycle for task 1 to complete the rest of the execution. Task 1 and 2 end in software task at the end.

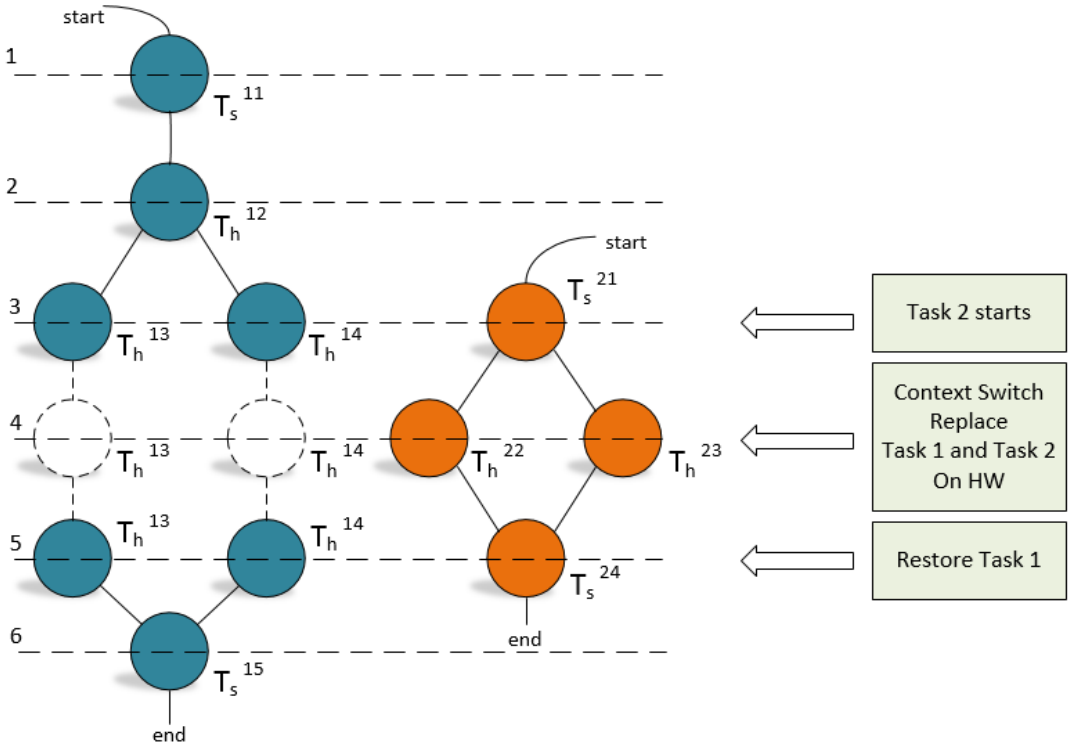


Figure 3.3: Task Model for Context Switching

To reduce or remove the configuration time overhead, partial reconfiguration can be done before the execution of the task, if there is an available slot that will not be used until the execution. An example task execution flow showing a software/hardware

combined task for pre-reconfiguration is shown in Figure 3.4.

```
run_task_11(args)
{
    do_something_in_GPP();
    ..
    ..
    preload_hw_task_12(); //optional
    ..
    ..
    run_hw_task_12();
    ..
    ..
    end_of_task_1();
}
```

Figure 3.4: Software Flow for SW/HW Combined Task

3.3 Reconfigurable Block Model

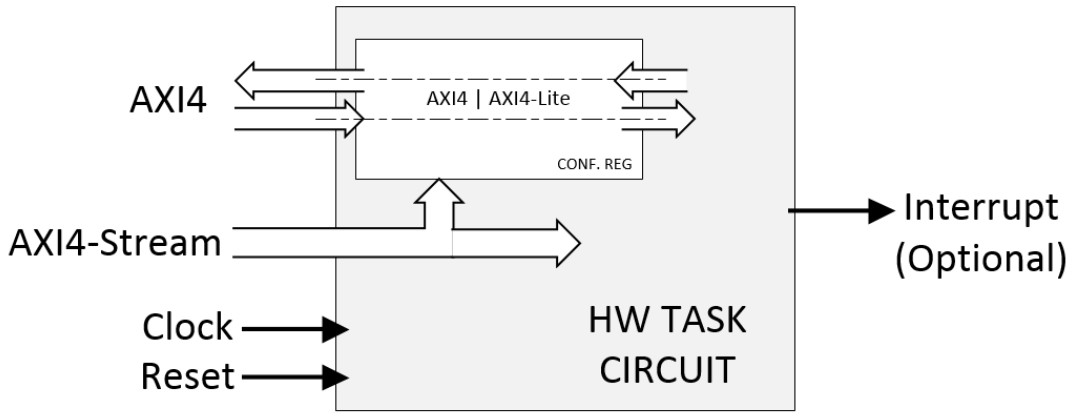


Figure 3.5: AXI4-based Reconfigurable Block Model

Input and output ports of the reconfigurable modules must be defined to be compatible with reconfigurable partition. The connection between the static and reconfigurable logic is only through partition pins. The partition pins must be well defined to meet the needs for reconfigurable block. An example for a reconfigurable block model is depicted in Figure 3.5. The model for the reconfigurable block has an AXI4-based address structure. Addressable design is used to direct input and output data easily to the correct location. On the other hand, reconfigurable block have configuration

registers which is used to control execution cycle of the reconfigurable module. If one of the commercial Intellectual-Property (IPs) is used in reconfigurable module, it can be easily connected as a submodule without extra logic to convert the data. Clock and reset pins are also available. An optional interrupt output can be added depending on the overall system requirements.

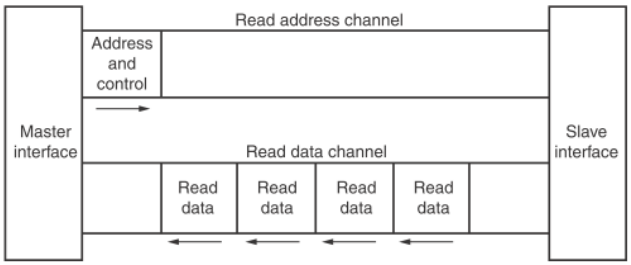
Xilinx uses Advanced eXtensible Interface (AXI) protocol for IP cores for newer devices including 7-Series FPGAs and SoCs [42]. AXI is a part of AMBA. AMBA is used for on-chip communication as a standard protocol. AXI4 is the second major version after AXI3 and it was released with AMBA 4.0 in 2010. AXI4 defines the following three interfaces;

- **AXI4**: Memory-mapped architecture and high throughput is offered. Up to 256 data transactions can be done for 1 address range via bursting mode.
- **AXI4-Lite**: Memory-mapped architecture and low-throughput is offered for simple read/write operations. Only 1 read/write operation can be done for 1 address which slows data throughput.
- **AXI4-Stream**: High throughput is offered but it is not used in memory-mapped applications. It is used in long data streaming.

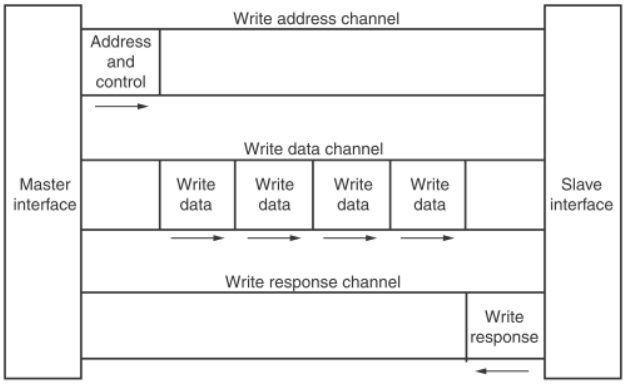
AXI4 and AXI4-Lite

The AXI4 interface is composed of master and slave sides. The reconfigurable block is the slave side. Input data is given by master side and the execution starts with the permission of master side. When the execution ends, output data is transferred to master side. Read/write operations can be performed at the same time through AXI4. The general data transaction architecture for AXI4 is shown in Figure 3.6.

The AXI protocol provides the independent read address, read data, write address, write data and write response channels. Each channel has its own handshake signals called valid and ready. Master side asserts valid signal indicating that the data or address is valid while slave side is asserting ready signal indicating ready to receive address or data. The data bus width can be 8, 16, 32, 64, 128, 256, 512, 1024 bits. Most commonly used data width is 32 bits.



(a) Read Cycle



(b) Write Cycle

Figure 3.6: AXI Protocol R/W Data Transactions

One or more master can start data transaction to multiple slaves with the help of interconnect mechanism between master and slave sides. Since each slave has different address, data transaction is established with the correct slave and master. General structure for AXI interconnect is shown in Figure 3.7.

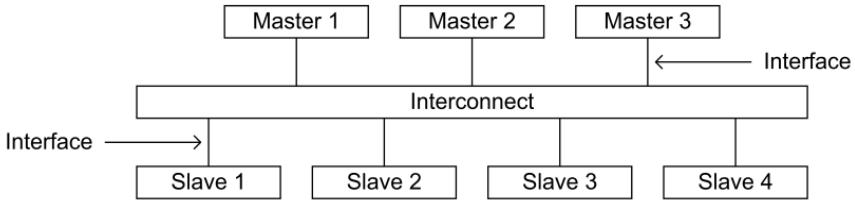


Figure 3.7: General Structure of AXI4 Interconnect

AXI4 interface on both sides can work together even if no interconnect circuit is used. However, the interconnect is used for addressing between different slaves and masters. AXI4-Lite on slave side can work with AXI4 on master side through both

the interconnect and direct connection. AXI Interconnect provides smooth interoperability between AXI4 and AXI4-Lite. On the other hand, direct connection is also possible by directing unconnected signals to the right way. Considering master side using AXI4, some important pin names and existence of those pins in slave side corresponding to AXI4-Lite is given in Table 3.1.

Table 3.1: AXI4 and AXI4-Lite Pin Names

AXI4 (MASTER)	**Description**	**Direction**	**AXI4-Lite (SLAVE)**
awaddr	Address of data to be written	→	YES
awburst	Burst type for write operation	→	**NO**
wdata	Data of write operation	→	YES
wlast	Current wdata is the last data	→	**NO**
wvalid	Current wdata is valid	→	YES
wready	Slave is ready to accept data	←	YES
WRITE CHANNEL			
araddr	Address of data to be read	→	YES
arburst	Burst type for read operation	→	**NO**
rdata	Data of read operation	←	YES
rlast	Current rdata is the last data	←	**NO**
rvalid	Current rdata is valid	←	YES
rready	Master is ready to accept data	→	YES
READ CHANNEL			
bresp	Write Response	←	YES
bvalid	Current bresp is valid	←	YES
bready	Master is ready to accept bresp value	→	YES
rresp	Read response	←	YES
RESPONSE CHANNEL			

Some signals are not supported by AXI4-Lite interface. Reconfigurable block has the interface of AXI4, but reconfigurable module (RM) can have either AXI4 or AXI4-Lite interface. If RM has AXI4 interface, there is no change in I/O connections. If RM has AXI4-Lite interface, non-existent output pin 'rlast' should be driven by '1' since the master side is affected by seeing always '0'. In addition, non-existent input pins are handled by software tool not to dangle.

AXI4-Stream

AXI4-Stream is not memory-mapped so that it cannot be directly connected to AXI4 interface. A DMA engine can be used to transfer data between streaming and memory-mapped interfaces. AXI DMA IP block can be added to the design to convert and transfer data. General I/O pins of a AXI DMA block is shown in Figure 3.8.

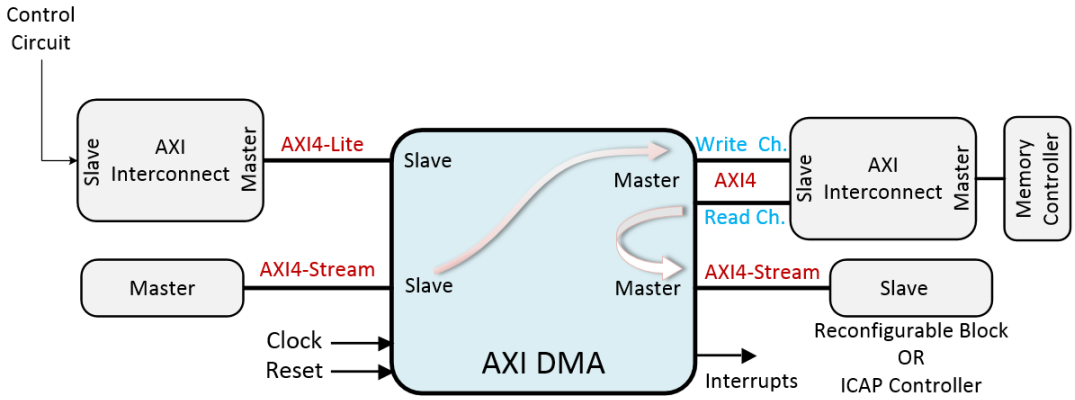


Figure 3.8: AXI DMA I/O structure

The control circuit sends the address, direction and length of the data to AXI DMA IP. If the direction is from read channel to AXI4-Stream, the DMA starts a data transaction on read channel. Memory controller takes the address and reads corresponding values and DMA converts incoming AXI4 signal to AXI4-Stream signal. If the direction is from AXI4-Stream to Write Channel, the DMA takes streaming data and converts it to memory-mapped data transaction. The memory controller writes the data to the memory using given address information. If no data comes from AXI4-Stream, the DMA waits until any transaction starts and does not abort the transaction until the number of words is equal to the length.

The reconfigurable block takes the input from AXI4-Stream interface in addition to AXI4. Streaming interface have the pins shown in Table 3.2.

Connecting an IP which have slave AXI4-Stream interface in partially reconfigurable block eases the development and increase the data transaction throughput.

Table 3.2: I/O pins of AXI4-Stream Interface

AXI4-Stream	Direction	Description
tdata	M → S	Data to be transferred
tkeep	M → S	Indicates which byte is valuable
tlast	M → S	Indicates the last tdata
tvalid	M → S	Current tdata is valid
tready	S → M	Slave is ready to accept tdata

Combining the requirements for I/O interface of execution block model, AXI protocol provides all solutions for performance and interoperability which a reconfigurable system requires.

3.4 ICAP Controller Model

ICAP is a primitive which can be added in partially reconfigurable HDL designs for Xilinx FPGAs. ICAPE2 is the version used in Xilinx 7-Series FPGAs. ICAP stands for Internal Configuration Access Port and it is used for only partial reconfiguration. Full reconfiguration cannot be done due to disrupting its input routing with the new configuration. ICAP primitive can also be used for reading back configuration memory. It has an 32-bit input and output pins, enable pin, read/write selection pin and clock pin.

ICAP controller uses ICAPE2 as a submodule and drives the I/O ports with its state machine. It consumes 57 LUTs, 94 FFs and no BRAM. However, it must be used with a FIFO which follows the controller and takes the configuration readback data as input. General structure of ICAP controller is shown in Figure 3.9.

It takes the first input from streaming interface to determine whether it is a write operation or read operation. For example, if the first data is 0x80000001, it is a read operation and only one 32-bit data will be read from the output of ICAP primitive. 0x80000001 can be a read request for a configuration register which is one 32-bit data. If the first data is 0x40000000, it is a write operation and rest of the stream is

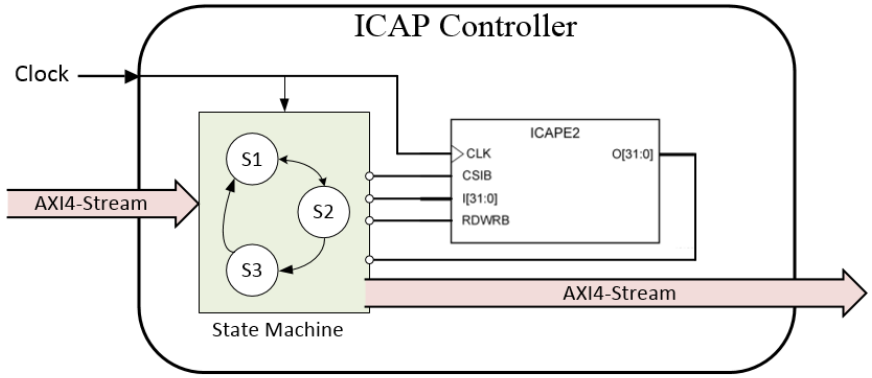


Figure 3.9: ICAP Controller and ICAPE2 Primitive

configuration data. It does not give a length for input data since the state machine waits for the tlast signal on AXI4-Stream interface.

ICAP controller converts not-bitswapped bitstream to the format of bit-swapped. If the configuration bitstream is created with -disablebitswap option, it needs to be converted. Table 3.3 shows bit ordering for bitswapped and not-bitswapped version of 32-bit data in configuration bitstream. 32-bit configuration data is bitswapped and loaded to the input pin of ICAP primitive.

Table 3.3: Bit Ordering Types of Configuration Data

Not-Bitswapped Bit Ordering
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bitswapped Bit Ordering
24 25 26 27 28 29 30 31 16 17 18 19 20 21 22 23 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7

3.5 Context Saving and Restoring Model

Context-switching operation on a reconfigurable hardware needs context extraction, bitstream manipulation and context restoration successively as a summary. There are some other steps which need to be done by the application. In Xilinx 7-Series Zynq SoCs, there are two internal configuration ports called PCAP and ICAP. Zynq architecture chooses default configuration port as PCAP but provides a device con-

figuration (DevC) library to choose ICAP also from ARM side of the SoC. There is a PL configuration path in Zynq architecture which is shown in Figure 3.10.

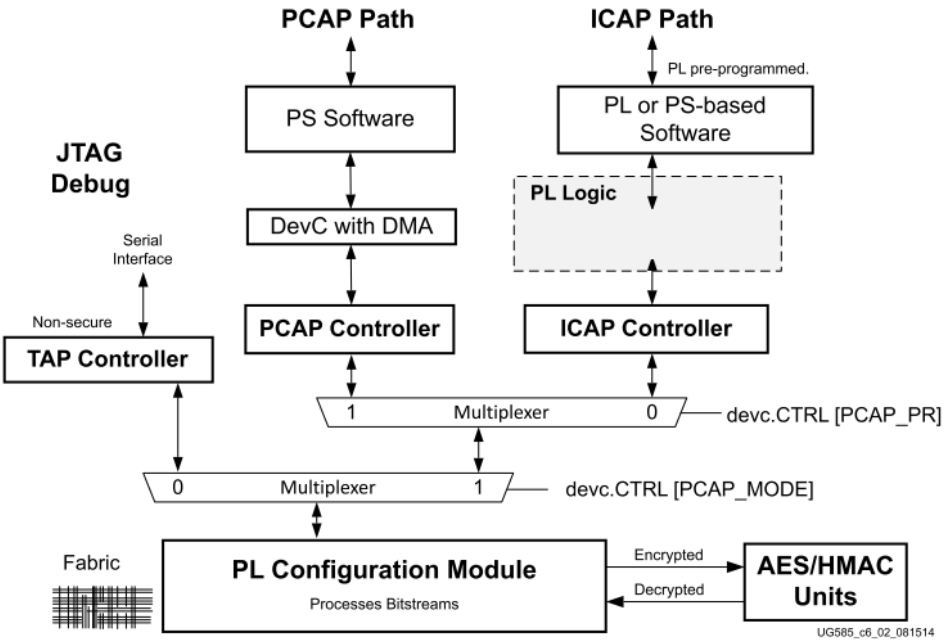


Figure 3.10: PCAP and ICAP Configuration Paths

XDCFG_CTRL_OFFSET register controls the PCAP_PR bit (27^{th} bit) to switch between PCAP and ICAP [43]. It is selected as 1 after each power-up. This bit should be selected as 0 to use ICAP hard macro. PCAP_MODE is selected as 1 after power-up. Therefore, there is no need to change that value.

Clock and reset signals for the reconfigurable block should be managed before reading states back. The clock is driven by 0 while reset is driven by 1. Thus, all FFs and BRAMs stops at a stable point. Reset should be 1 because AXI protocol uses active-low reset. This can be done with partial reconfiguration decoupler IP, which decouples clock and reset from the reconfigurable part by a control signal 'decouple'.

To capture all states and BRAM contents, there are two options available.

- **CAPTUREE2 Primitive**:

  This primitive is used to capture current states of the flip-flops and latches so that captured values can be read from ICAP or PCAP interfaces. CAPTUREE2

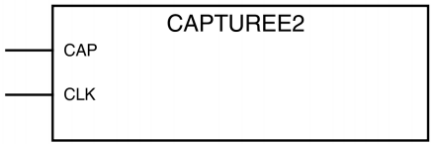


Figure 3.11: CAPTUREE2 Hard Macro

has two inputs which are shown in Figure 3.11. If the CAP input is high and the CLK has a transition from low-to-high, current state data is captured. Data is captured every CLK transition by default, but we can disable it with the generic attribute ONESHOT by setting it to true. In the design, only one data capture cycle is wanted so that if the second capture event is wanted, CAP input needs to be low and high again. CAP input is set and reset via a GPIO IP. CAPTUREE2 primitive needs to be instantiated in the top module.

- **Sending GCAPTURE Command**: Another option to obtain last state of flip-flops and latches is to use GCAPTURE command register code. FPGA architecture incorporates configuration registers to manage reconfiguration and other FPGA-related options. Command Register (CMD) takes GCAPTURE command (01010) and does the same capturing event that CAPTUREE2 primitive does. The sequence of the commands and descriptions are shown in [23].

After capturing the last states, configuration memory must be readback to the system memory. To be able to readback, the user must send a readback request by sending readback commands to ICAP primitive through ICAP controller. A DMA engine transfers all readback data to system memory.

Readback can be performed for each CLB state and BRAM content in frame address intervals. Frame addresses for BRAM blocks can be obtained from logic location file (.ll) created by Vivado. CLB frame address interval can be obtained from original partial bitstream file. A separate readback procedure can be applied for CLBs and BRAMs successively to reduce the time overhead. If whole FPGA surface was read back, it would take much longer time.

Bitstream manipulation can be done on ARM processor with copying captured values to the original partial bitstream. Only the related payload intervals must be manipu-

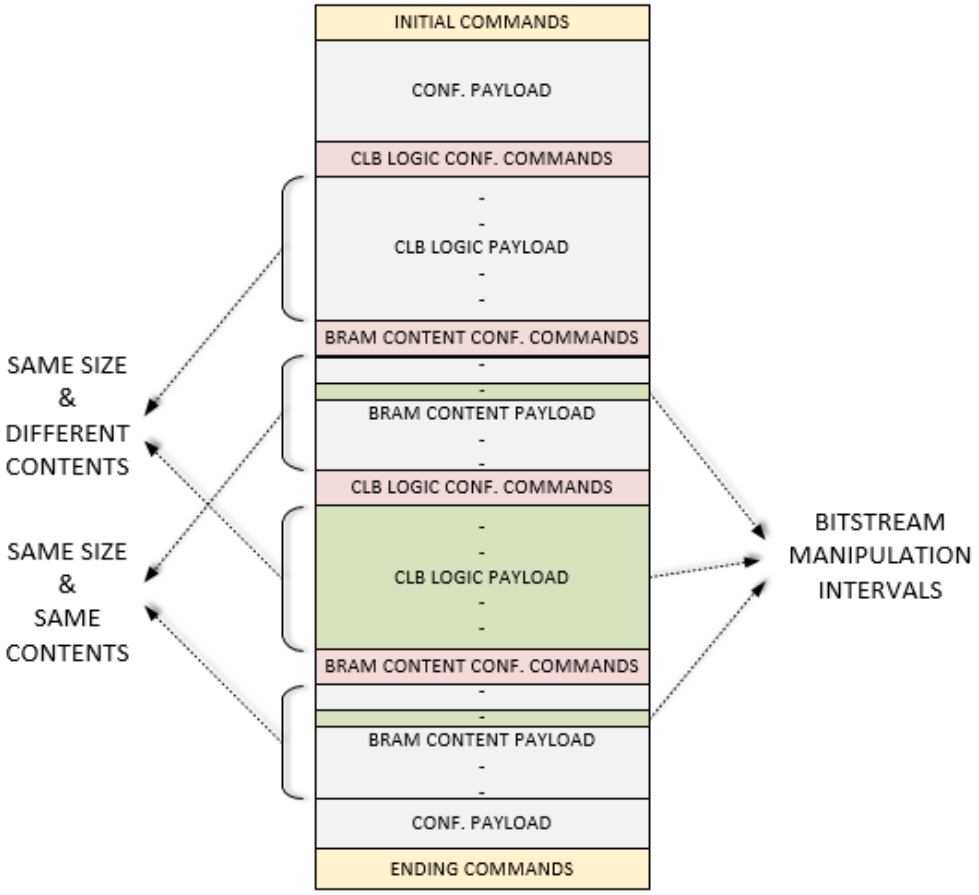


Figure 3.12: A general partial bitstream structure and manipulation intervals

lated. Figure 3.12 shows the intervals to be manipulated in a partial bitstream.

Partial bitstream file structure contains two programming payloads for CLBs and BRAMs. If the original file is reviewed, it can be seen that BRAM contents are the same for two programming payloads. In case of CLB states, second programming payload block of CLB logics represents desired initial states while the first one is unknown. This can be verified with the readback of initial configuration without capturing event.

After manipulation, new configuration bitstream is programmed when the context restoring is requested. Programming operation consists of only sending the bitstream. Partially reconfigurable area is initialized with the last states captured. Clock and reset circuit should be activated again to start the execution. Whole context saving and restoring procedure can be summarized as below:

Step-1: Change configuration path to ICAP.

Step-2: If context saving is required by the system, enable decoupling for the partially reconfigurable block so that the circuit stops.

Step-3: Toggle CAPTUREE2 primitive or send GCAPTURE command to capture the states.

Step-4: Send readback commands to the ICAP controller through the DMA engine.

Step-5: Wait for readback data to be copied to the DDR memory. Start bitstream manipulation using the technique as mentioned in Figure 3.12.

Step-6: Load another hardware task to the reconfigurable block, disable decoupling and wait for the end of execution or another context-saving.

Step-7: If context restoring is required, enable decoupling for the reconfigurable block.

Step-8: Send new bitstream data to the ICAP controller through the DMA engine and wait for the end of partial reconfiguration.

Step-9: Disable decoupling for the reconfigurable module so it can run.

Step-10: Context restoring is done.

CHAPTER 4

IMPLEMENTATION

4.1 Base System Architecture

An AXI4 based reconfigurable computing platform is developed to meet the need for context switching on a partially reconfigurable system. The system is based on Xilinx Zynq-7000 SoC architecture which is composed of two main parts. It has dual-core ARM Cortex-A9 processor system (PS) running at 666 MHz and 7-series programmable logic (PL), namely FPGA side. DDR3 SDRAM and SD card interface which are used mainly for bitstream storage is connected to the PS side of the system. Reconfigurable blocks are located at the PL side of the system.

The SoC architecture connects PS and PL sides with AXI protocol and some discrete lines. Reconfigurable blocks are directed with AXI interface and they are used as accelerator for the processor (ARM) side in case the processor wastes more time to execute related task. The SoC board used in implementation is Avnet ZedBoard which has Xilinx Zynq XC7Z020-1CSG484 SoC on it.

Zynq SoC has an internal clock generator supported by 3 PLLs at PS side which can be distributed to PL side. These are ARM PLL, I/O PLL and DDR PLL. Those PLLs use the same clock source which is 33MHz and they are supported by several MUXes to direct different frequencies for DDR3 SDRAMs, I/Os and ARM side. There is also one 100MHz clock input at PL side. This clock can be divided, or another clock can be synthesized by Mixed-Mode Clock Manager (MMCM) and the generated clock can be used in both PL and PS side to run together synchronously.

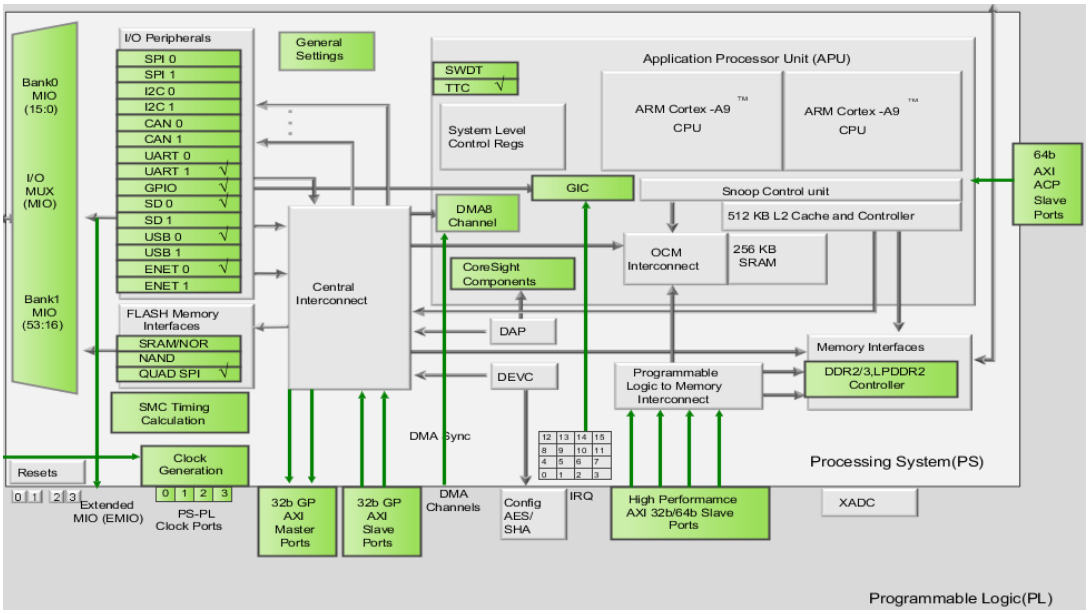


Figure 4.1: Zynq Architecture

Additionally, the SoC architecture has an SD Card interface connected to the PS side via Central Interconnect shown in Figure 4.1. Full and partial bitstreams are stored in an SD Card and can be transferred to DDR memory via DDR controller which is also connected to Central Interconnect. RS232 (UART) interface is used to send commands or output debugging results. It works at baud rate of 115200 bits/sec. 1Gbit Ethernet can also be used to receive bitstreams or to take some commands from network. Internal 16/32bit timers (TTC) are used to count clock cycles needed for data transactions and computations.

Application Processor Unit (APU) has a DMA facility to support mainly long data transfers between DDR RAMs and other peripherals which have a connection to Central Interconnect such as 32bit General Purpose (GP) AXI4 based master and slave ports. Another AXI DMA block is added at PL side to support AXI4-Stream interface.

General Zynq SoC interfaces and AXI4 based implementation is shown in Figure 4.2. There are two different interfaces between PS and PL sides in the design. One of them is 32-bit General Purpose (GP) AXI master ports. There are two separate GP AXI master ports to manage designs at PL side. Only one of them is used. AXI GP0 port is connected to AXI Interconnect IP. Both AXI4 and AXI4-Lite based slaves

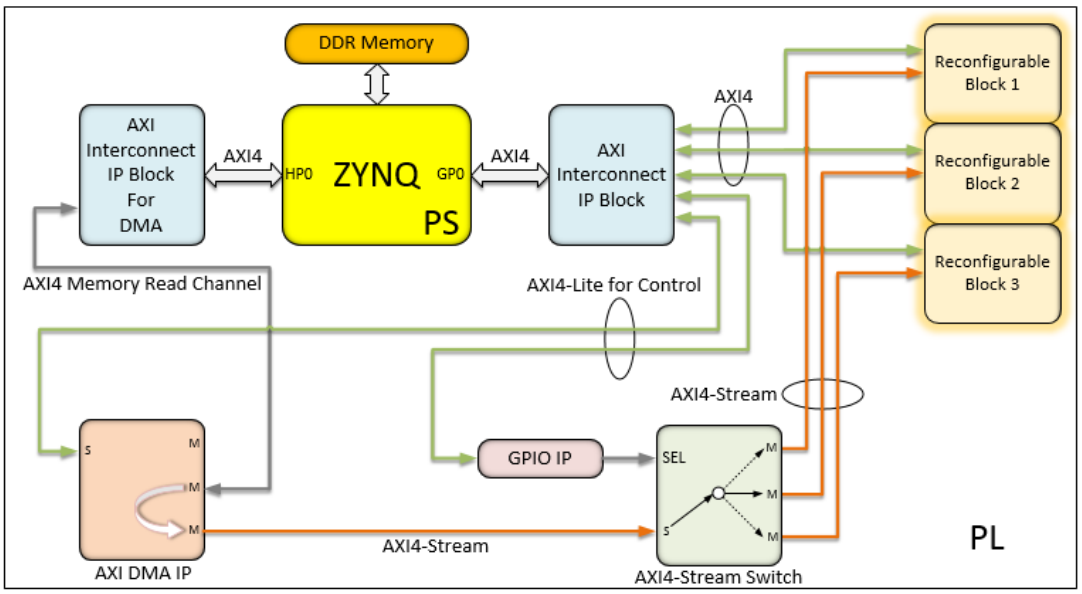


Figure 4.2: Base System for HW Tasks Without ICAP Controller

can be connected to AXI Interconnect IP. Obviously, the interconnect works to operate different slaves coherently. When the master wants to transfer input data to a slave, the data comes from the PS side to PL side via the interconnect and reaches the slave. Conversely, when the master wants to read output data from a slave, the data transaction way is the inverse. Since AXI4 has address based architecture, this transaction is all done with the help of addressing scheme of registers. The 32-bit address determines which data is transmitted or received under the supervision of master side.

Second interface between PS and PL is high performance port (HP) which runs in slave mode since the managing side is the PL for this connection. There are four HP ports on Zynq in total and only one of them is used. AXI Interconnect IP is also mandatory for this interface and it has one master memory reading channel connected to AXI DMA and one master port connected to HP port of Zynq. AXI DMA IP block manages the traffic from DDR memory to reconfigurable blocks running with AXI4-Stream interface. AXI4-Stream is completely different from AXI4/AXI4-Lite interface since it has no addressing signals but basic handshake and data signals. All data transactions and AXI4 to AXI4-Stream conversion is managed by the DMA which receives directives by its own control registers through GP master port as a slave as shown in Figure 4.2. AXI DMA IP block has some customization options

which affects the throughput and hence performance. Specifications selected in AXI DMA IP block is shown in Table 4.1. All address and data width values must be 32-bit. Width of Buffer Length Register states maximum size of data transfer in bytes as power of two. It is selected as the highest value. Write channel of the DMA engine is also selected because it will be used for ICAP controller in the next section.

Table 4.1: AXI DMA IP Block Specifications

Option	**Selection**	**Value**
Enable Scatter/Gather Engine	Not Selected	-
Enable Micro DMA	Not Selected	-
Width of Buffer Length Register	-	23 bits
Address Width	-	32 bits
Enable Read Channel	Selected	-
Enable Write Channel	Selected	-
Memory Map Width for R/W Ch.	-	32 bits
Stream Data Width for R/W Ch.	-	32 bits
Max. Burst Size for R/W Ch.	-	256
Allow Unaligned Transfers for R/W	Not Selected	-

At PL side, there are three reconfigurable blocks which have both AXI4-Stream (slave) and AXI4 (slave) interfaces to take input data and process the related task. Therefore, there are two ways on which data is transferred to the slave reconfigurable blocks. If input data comes from AXI4-Stream interface, valid, ready and last signals must be switched to related transaction path because stream does not have addressing scheme. Since there are three computing blocks and one master AXI4-Stream data sender on DMA side, this single output of DMA must be switched by AXI4-Stream Switch IP block to one of the computing blocks. SEL pin of AXI4-Stream Switch IP block is used to select the path and it can be set by GPIO IP block. After choosing proper data transaction path, task runs and can create output data. When the task completes or some part of output data is generated before the completion, the output data is read back to PS side via AXI4 port.

The base system architecture shown in Figure 4.2 cannot be reconfigured through the ICAP. It can be reconfigured through JTAG or PCAP with a slower throughput. On the other hand, the readback operation cannot be performed reliably because there

is no decoupling circuit in front of partially reconfigurable blocks to decouple clock and reset signal. Therefore, context-switching is not suitable for the base system and requires more equipped system components.

4.2 Complete Context-Switchable System Architecture

The overall system must have readback and partial reconfiguration features. To accomplish this, some readback and reconfiguration mechanisms are added to the existing system components to run coherently with the final design.

Partial Reconfiguration (PR) Decoupler IP is recommended for a design which enables partial reconfiguration due to metastability conditions during reconfiguration [44]. PR Decoupler IP is added in front of each three partial blocks to decouple clock and reset signals from the partial block. Decoupling of clock and reset is required when the circuit is wanted to be stopped before readback and to be started after partial reconfiguration operation.

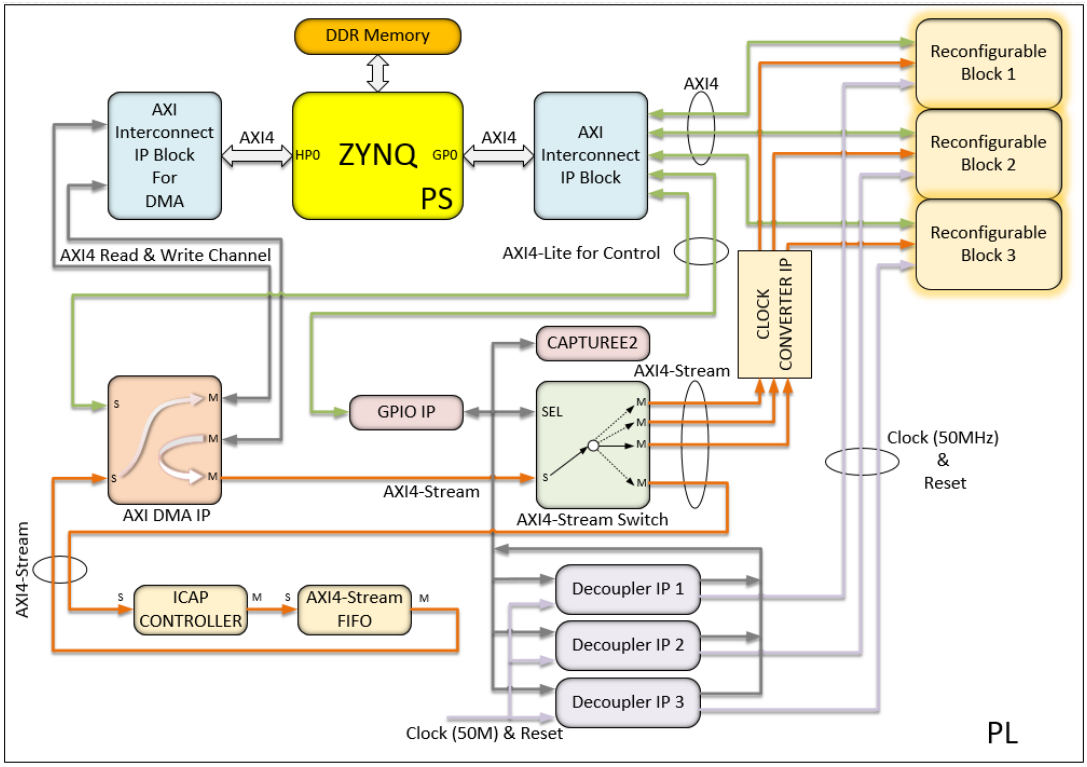


Figure 4.3: Complete Architecture for Context-Switchable PR System on Zynq SoC

Since reset logic is active-low in AXI standard, reset should stay at logic ‘1’ in order not to reset related circuit to the initial state. PR Decoupler IP is managed by one pin of General Purpose I/O IP (GPIO). It has an input called “decouple” which is connected to one of the outputs of GPIO IP and it has an output called “decouple_status” which is connected to one of the inputs of GPIO IP. These signals are used to control and to see the status of decoupling logic. PR Decoupler IP and its connections are shown in Figure 4.3.

Partial reconfiguration can be performed through both ICAP and PCAP by toggling PCAP_PR bit as mentioned in Chapter 3. However, when PCAP is used, configuration throughput is slower than the one in ICAP. To increase the configuration speed, ICAP controller can be added to the design. It controls reading and writing procedures for ICAPE2 primitive. ICAP controller has both master and slave ports of AXI4-Stream. It can be connected to the AXI DMA IP like ZyCAP and MiCAP-Pro [28, 32]. However, reconfigurable blocks have also slave streaming port so that streaming output of the DMA was connected to AXI4-Stream Switch IP block to switch related signals in the base system. Since ICAP controller has also streaming (slave) port, it can be connected to the switching block. In addition, master streaming output of ICAP controller cannot be directly connected to the DMA. This is because the streaming output of ICAP controller provides output consistently while the DMA and the Interconnect have some delays which causes data loss. To compensate the delay, an AXI4-Stream Data FIFO with size 1024 is added between ICAP controller and the DMA. The connections for ICAP controller are shown in Figure 4.3

PCAP has also full reconfiguration capability since it is located at PS side. BareMetal operating system has a driver so that PCAP can be used for full and partial reconfiguration. Device configuration driver (XDcfg) is added to the main C file. When full reconfiguration needed, PCAP is used for switching between different configurations.

To obtain the last state of the circuit when the clock and reset are decoupled from the partial block, there are two options to save the last logic state (0 or 1). One of them is to use CAPTUREE2 hard macro and other option is to use GCAPTURE command sequence to capture current state information. The command sequence can be sent to ICAP or PCAP interfaces in software. CAPTUREE2 hard macro is added to the

design so that it can be triggered with a control signal. GPIO IP is used to control 'CAP' input of CAPTUREE2 primitive.

Clock distribution of the design is done with two generated clocks from one 100MHz source as shown in Figure 4.4. MMCM is used to forward 100MHz input to the first output which drives the circuit running AXI4-Stream interface such as AXI DMA IP, AXI Interconnect IP for DMA, ICAP controller, AXI4-Stream FIFO and HP pin of Zynq side. Second output clock is 50MHz and drives AXI4 related circuits which contains the rest of the circuit. This clock distribution is done due to timing problems when sourcing all clock inputs with a single 100MHz. This customization depends on the application specifications.

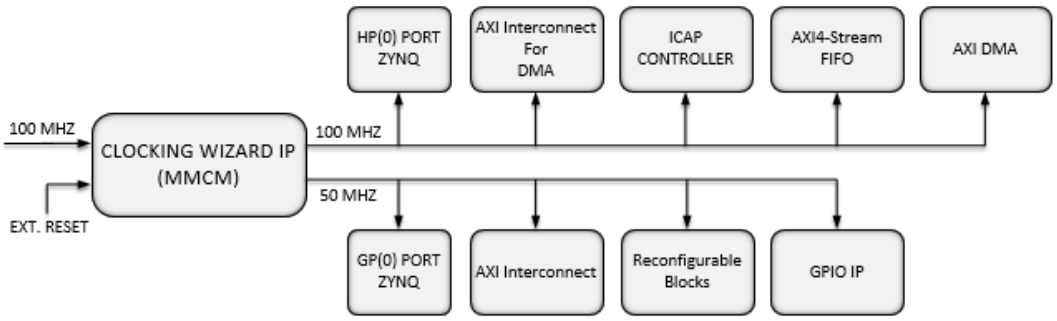


Figure 4.4: Clock Distribution for Overall Design

Reconfigurable blocks be misguided on AXI4-Stream interface since only one clock source 50MHz is driven into the block while streaming interface run at 100MHz. When AXI DMA IP transfers the data on streaming interface with 100MHz, about half of the streaming data is lost at slave side. To overcome this issue, AXI4-Stream Clock Converter IP is used to manage input data coming at 100MHz and transfer the data to the partial blocks at 50MHZ. This IP block uses the technique of toggling tready signal to decrease the clock frequency. Thus, actual frequency is 50 MHz from DDR memory to reconfigurable blocks so that the problem of data loss is solved.

CRC checking is required by default when partial or full reconfiguration occurs. CRC value is appended to the end of bitstream to check bitstream integrity. It can be generated by Vivado during bitstream generation and the related FPGA/SoC device can

generate the same CRC value during configuration. When reconfiguration reaches to the end of bitstream, the device creates the same CRC value and compares the two CRCs to finish reconfiguration. Since bitstream manipulation is performed, CRC value in the partial bitstream must be changed. However, there is no online CRC generation algorithm provided by Xilinx, but there is an option to disable the CRC check in Vivado. It is disabled for this design since bitstream manipulation is performed so that a new CRC value is not required.

4.3 Example Application for Context Saving and Restoring

4.3.1 Reconfigurable Modules

There are two RMs designed to be run on a reconfigurable block. First one is FFT IP core which takes FFT of a given input. Fast Fourier Transform (FFT) is an algorithm to take Discrete-Time Fourier Transform which shortens the time needed for the computation by decreasing computational complexity. Xilinx provides an FFT IP block which can be customized for its size dynamically and statically. It uses the AXI4-Stream as an interface. There are two stream inputs and one stream output. One of the inputs is for FFT specifications and the other is for the data. Output stream is for the FFT result. There are also six single outputs which are used for the status. General I/O structure and configuration options for FFT IP core is shown in Figure 4.5 and Table 4.2 successively.

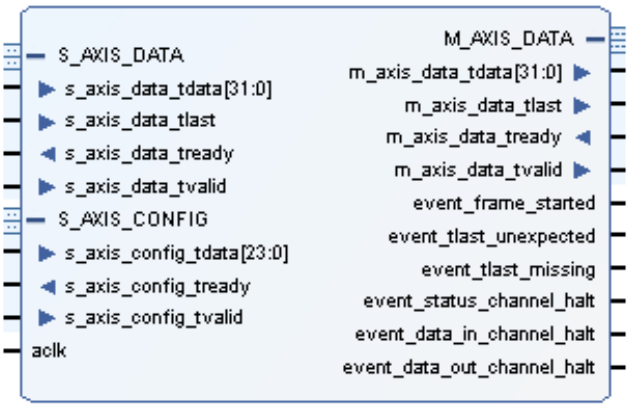


Figure 4.5: FFT IP Core I/O Structure

Table 4.2: FFT Configurations in FFT IP Core

Configuration Name	Value/Option	Configuration Name	Value/Option
Number of Channels	1	Input Data Width	16
Transform Length	1024	Phase Factor Width	16
Architecture	Radix-2 Lite,Burst I/O	Memory Option(Data)	Block RAM
Data Format	Fixed Point	Memory Option(Phase Factor)	Block RAM
Scaling	Scaled	Complex Multipliers	CLB Logic
Rounding Mode	Truncation	Butterfly Arithmetic	CLB Logic

It has 24-bit configuration port and 32-bit input data port. However, there is only one port for AXI4-Stream in reconfigurable block. To connect both, a simple state machine is written to map the input stream to data and configuration channels. In addition, output of FFT IP core uses AXI4-Stream. Since there is no output port for AXI4-Stream in reconfigurable block, FFT result can be handled with a FIFO which has both AXI4-Stream and AXI4-Lite interface. Input of FIFO IP core can be connected to output of FFT IP core and AXI4-Lite interface can be connected to reconfigurable AXI4 interface as shown in Figure 4.6.

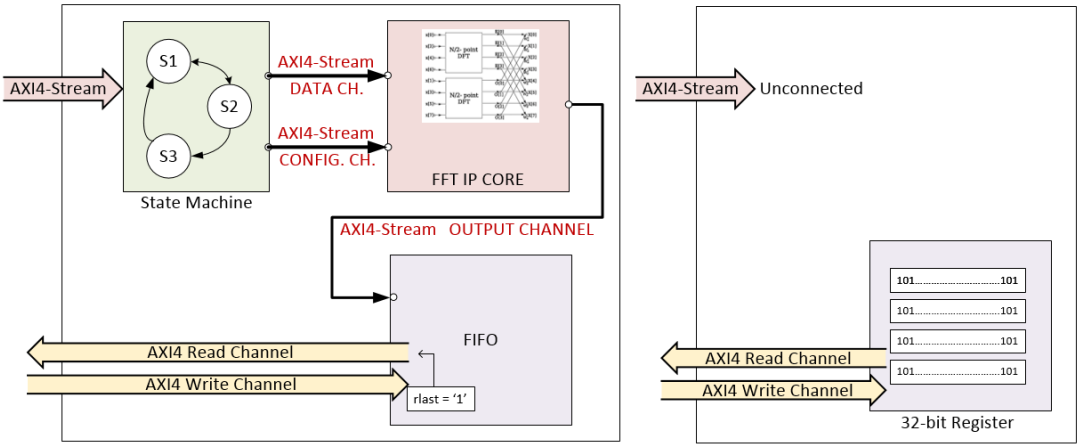


Figure 4.6: Reconfigurable Module Connections

Another IP core used for context switching is simple AXI4-based register. It contains four 32-bit registers which can be set with AXI4 data interface. It is used to show a register read/write operation between context saving and restoring of FFT circuit. It can be connected through AXI4 interface in reconfigurable block. Figure 4.6 shows connections between each RM and the reconfigurable block.

4.3.2 Context-Saving and Restoring

Context switching flow in the example application is given in Figure 4.7. Context saving occurs after FFT IP core takes whole input data and starts the calculation. During execution, clock and reset is decoupled from the partial block. Then, CAPTUREE2 primitive is triggered or GCAPTURE command is sent to ICAP controller to capture last states of FFs and content of BRAMs. Readback procedure is started for CLB and BRAM frames successively and readback data is copied to the DDR memory. Context saving is accomplished by this way.

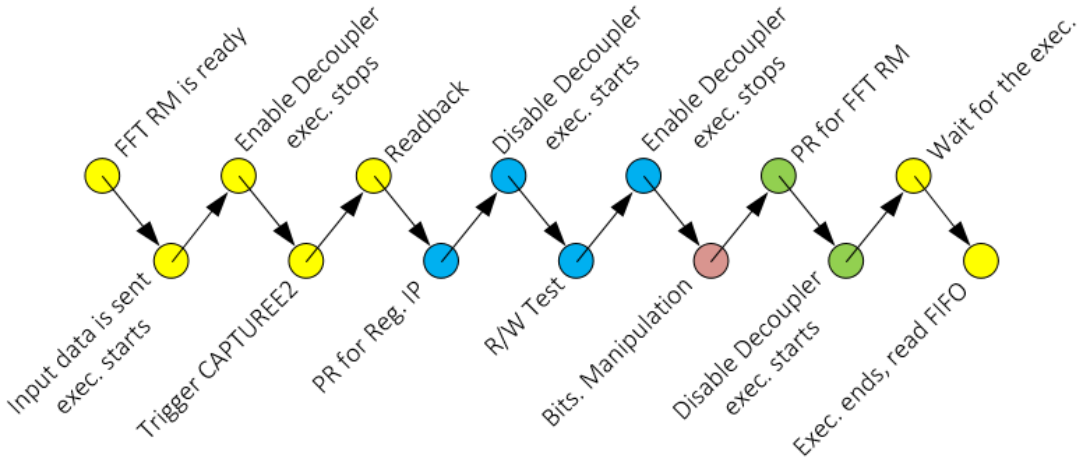
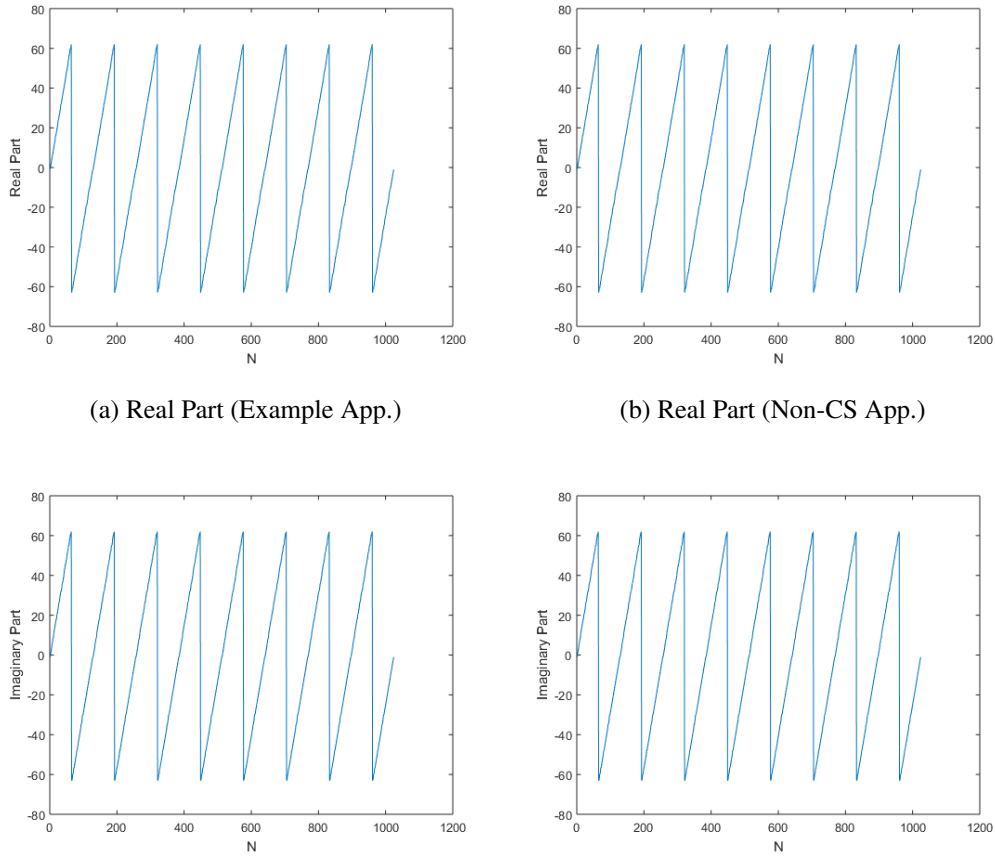


Figure 4.7: Example Application Context Switching Flow

The procedure between context saving and context restoring starts with partial reconfiguration for register IP. After partial reconfiguration, the decoupler is disabled so that the register circuit can run. Then a read/write test runs for registers. If the test is passed, it means partial reconfiguration is completed successfully. It is already known that partial reconfiguration is performed without a problem.

Then bitstream manipulation is performed on ARM processor. The method for bitstream manipulation is defined in Chapter 3.5. Clock and reset are decoupled again. New partial bitstream for FFT is used to configure the partial block. When reconfiguration is done, clock and reset are activated again. After FFT operation is complete, output stream sends output data to a FIFO which can be readable by the processor. If the result is the same with the expected result, then context-saving, bitstream manipulation and context-restoring works on the system. Figure 4.8 shows that real and

imaginary part of the FFT output are the same with the expected result. Expected result is created with a different project running FFT IP core with the same specifications but no context-saving and restoring is included. FFT result of the given input in MATLAB is not the same with the one in the example application since both platforms have different rounding and scaling policies. Therefore, MATLAB result is not comparable with the output of this system.



(a) Real Part (Example App.)

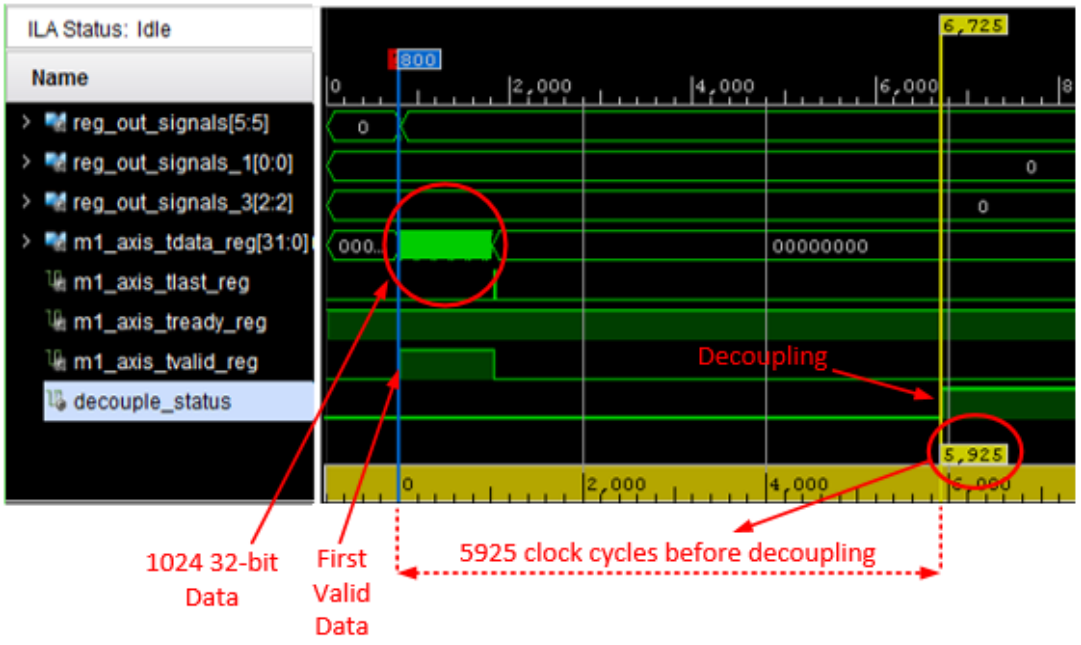
(b) Real Part (Non-CS App.)

(c) Imaginary Part (Example App.)

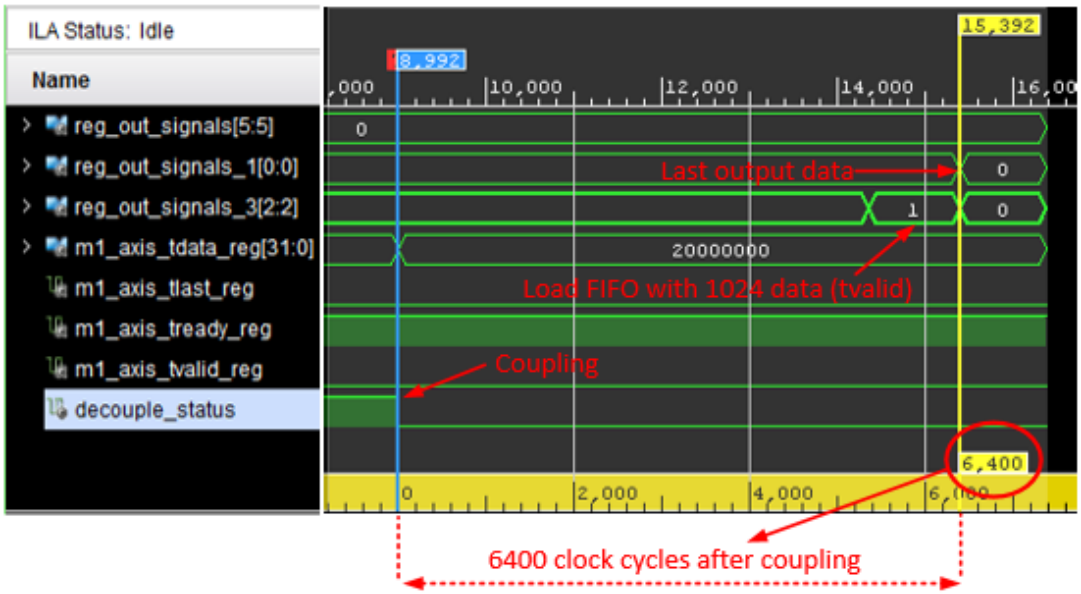
(d) Imaginary Part (Non-CS App.)

Figure 4.8: FFT Results for Example Application and Non-CS Project

Total execution takes 12325 clock cycles for 1024 point FFT according to core specifications. Figure 4.9 is taken from a debugging circuit and shows how and when decoupling/coupling is applied and shows that total number of execution cycle is consistent with the given information. It takes 5925 clock cycles before decoupling and takes 6400 clock cycles after coupling the clock and reset connections. Sum of



(a) Enabling Decoupling



(b) Disabling Decoupling

Figure 4.9: Decoupling Procedure for Clock and Reset of FFT RM

these two values is equal to 12325 which is needed clock cycle for FFT calculation by FFT IP core. This measurement is done to show that FFT operation continues where it was stopped from. Thus, context saving and restoring and bitstream manipulation is performed without disrupting running circuit.

4.4 Partitioning Properties

There are three reconfigurable partitions in overall system as mentioned in previous chapters. Floorplanning is done with manual area selection. Selected areas contain enough resources which is required for the largest circuit. On the other hand, reconfigurable partitions can have homogeneous or heterogeneous resources. Figure 4.10 shows where static and reconfigurable areas are located in the example application. Selected areas are homogeneous and have enough sources to execute FFT computation. Static areas are shown as orange while the reconfigurable ones are shown as green. Reconfigurable partitions have a pink rectangular border inside a clock region.

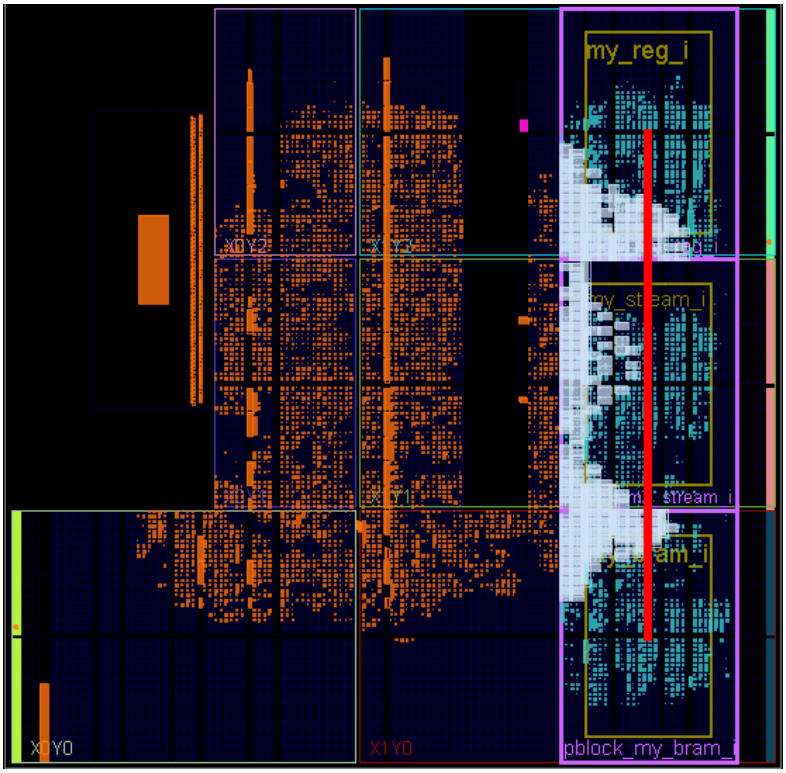


Figure 4.10: Floorplanning For The Example Application

ARM side of the SoC is located at top-left of the surface and AXI4-based HP and GP ports are located at the boundary. Partition pins are shown as small white rectangles inside each reconfigurable block. Each partition has 1400 slices, 20 RAMB36, 40 RAMB18, 40 DSP blocks. Table 4.3 shows utilization values for FFT example and other reconfigurable modules which can fit into those reconfigurable partitions.

Table 4.3: Utilization for Reconfigurable Modules

Reconf. Module Name	**#SLICE**	**#RAMB36**	**#RAMB18**	**#DSP**
FFT	729(52%)	1(5%)	4(10%)	0(0%)
4x32-bit Register	127(9%)	0(0%)	0(0%)	0(0%)
Greybox (empty block)	111(8%)	0(0%)	0(0%)	0(0%)

FFT module and register IP are partially reconfigurable for this platform. None of them includes DSP resources because they would not be preemptable if they include DSP resources. Greybox indicates an empty partition which is useful for power saving policies. Each reconfigurable module is implemented for all three partitions. There are 3x3=9 partial bitstream files and each has the same size. The size does not depend on the utilization rates, but partition location and size. Table 4.4 shows full and partial bitstream sizes in bytes for .BIT and .BIN files. There is an extra textual information header in .BIT file in comparison to .BIN file. A reconfigurable block has roughly 20% of total resources according to bitstream sizes. Since the partitions are homogeneous, the values are equal to each other.

Table 4.4: Bitstream Sizes for Full and Partial Designs

Type	**.BIT (bytes)**	**.BIN (bytes)**
Full	4,045,663	4,045,564
Partial	799,676	799,564

CHAPTER 5

EVALUATION AND TEST RESULTS

5.1 Measurement Environment and Tools

Vivado 2017.4 and its SDK tool is used for measurements. They are both provided by Xilinx. On Vivado, a hardware logic analyzer is added to the design to measure delays and clock cycles. SDK provides a BareMetal operating system with debugging options. All hardware drivers are included in SDK. Throughput measurements are done via an internal timer running on ARM processor. The accuracy of the timer is confirmed with a counter running on FPGA side.

5.2 Throughput Measurements and Evaluations

Zynq SoC, a hybrid reconfigurable system, can execute both software and hardware tasks concurrently with an operating system running on ARM processor. It can increase the system performance by utilizing reconfigurable hardware with true-multitasking while managing software tasks on the processor. When a task preemption is required on hardware, it requires saving the circuit and reconfiguring it again. However, to load the circuit onto the hardware, a new bitstream must be created for the circuit to continue where it was saved from. This creation is performed by manipulating the bitstream which requires extra time. As a result, there are three timing requirements in hardware which needs to be evaluated. These are context-saving, bitstream manipulation and context-restoring time.

During context saving and restoring, ICAP controller is used with 100 MHz clock. ICAP has 32-bit interface so that at each rising edge of the clock, four bytes are

loaded into the ICAP. All throughput values are measured for long write/read cycles. The formula for calculation of each throughput is given as;

$$Throughput = \frac{Number\,of\,Bytes}{Elapsed Time}$$

5.2.1 Context-Saving Time Evaluation

Context-saving requires the following time-consuming operations;

- Decoupling of clock and reset circuit is done roughly in 40 clock cycles with software overhead. GPIO IP is loaded with a value in software which toggles decoupling circuit. 50 MHz clock is used to control decoupler, therefore, it lasts 800ns.
- Capturing states of FFs and BRAM contents can be performed by CAPTUREE2 primitive. If it is controlled with the GPIO IP, it takes the same time with decoupling logic. If it is controlled on hardware, it takes only 1 clock cycle. If GCAPTURE command is used, sending procedure is measured as roughly 7us with software overhead.
- Readback commands are sent two times. First one is for CLB frames and the second one is for BRAM frames. Each throughput is measured as 367.3 MB/s including software overheads. If software overhead is not included, it has 381.4 MB/s throughput which is nearly equivalent to theoretical value.

Considering software overhead in these measurements, total time consumption for a partial bitstream can be calculated. A configuration memory contains 353,500 bytes as CLB and BRAM data and readback commands are 1,272 bytes while the throughput is 367.3 MB/s. Dividing number of bytes to the speed of readback gives 921 μs. Decoupling and capturing procedure takes approximately 1 μs when CAPTUREE2 is controlled on hardware. Thus, total time for reading configuration memory to DDR memory is 922 μs.

There are other controllers used in context saving applications. PCAP can be used in readback without consuming any resource on FPGA. However, the throughput is

lower than DMA-based ICAP controllers. FaRM works on Virtex-5 FPGA through a PLB bus. It has 95 MB/s throughput. MiCAP-Pro has the nearest value since it has the same DMA-based architecture but there are I/O buffers in front of ICAP primitive which causes lower throughput. Comparison of the throughput with other context saving applications is given in Table 5.1.

Table 5.1: Throughput Comparison for Context Saving Applications

Source	**Cont. Name**	**Throughput**
[12]	PCAP	145 MB/s
[32]	MiCAP-Pro	272 MB/s
[38]	FaRM	95 MB/s
This Work	ICAP Controller	367.3 MB/s

While using PCAP option, 32-bit internal configuration data is loaded to a FIFO before saving on DDR memory. AXI protocol is used for the transaction and AXI-PCAP bridge in PS side reads the FIFO and directs the data to the DDR memory controller [27]. MiCAP-Pro uses a DMA which is located at PL side which directs the configuration data to the DDR memory controller via an HP port of Zynq [32]. FaRM uses two dedicated state machines and FIFO structures in front of the ICAP primitive. Reading and writing channels uses the different clock frequencies [38]. ICAP controller in this work uses an analogous architecture to MiCAP-Pro. The difference between our ICAP controller and MiCAP-Pro is how the FIFO is used for readback. MiCAP-Pro waits for the FIFO to be full before forwarding to the DDR memory. Our controller uses a FIFO which has AXI4-Stream slave and master interfaces and it does not wait to be full.

5.2.2 Bitstream Manipulation Time Evaluation

Bitstream manipulation is performed by reading a value from DDR memory and writing in an appropriate way to another DDR memory address. The address information depends on the size of partial bitstream and location of reconfigurable partition. Bitstream manipulation throughput is measured as 46 MB/s. It contains reading CLB and BRAM values from DDR memory and writing CLB data to one memory interval

and BRAM data to two memory intervals because of partial bitstream architecture. It takes about 7.3 ms when 353,500 bytes are manipulated.

It is not appropriate to compare different platforms for bitstream manipulation since there can be many options/algorithms to copy or manipulate data independently. Table 5.2 is just to show that there are other works performing manipulation.

Table 5.2: Throughput Comparison for Bitstream Manipulation

Source	Manip. Platform	Throughput
[40]	Pentium-4 CPU	304 KB/s
[41]	ARM Cortex-A9	N/A, 94 μs - CLB, 229 μs - BRAM
This Work	ARM Cortex-A9	46 MB/s

5.2.3 Context-Restoring Time Evaluation

Reconfiguration throughput is measured for many reconfigurable systems since it is seen as a bottleneck compared to run time of hardware tasks. DMA-assisted ICAP controller reduces the time needed for context-restoring. Partial bitstream size for this system is 799,564 bytes and the throughput is measured as 380.1 MB/s including software overhead. It is higher than the one in context-saving since number of data transaction is one while it is three for context-saving so that software overhead is higher. Calculating the time, context-restoring requires 2 ms in this system. Time consumption due to decoupler can be ignored since it is much smaller than 2 ms.

Table 5.3: Context-Restoring/Partial Reconfiguration Throughput Comparison

Source	Controller Name	Throughput	Software Overhead
[26]	HWICAP	14.6 MB/s	Included
[27]	PCAP	126.8 MB/s	Included
[32]	MiCAP-Pro	272 MB/s	Included
[26]	BRAM_HWICAP	371.4 MB/s	Unknown
[28]	ZyCAP	381.46 MB/s (Th.)	Not Included
This Work	ICAP Controller	380.1 MB/s	Included

Table 5.3 gives the throughput for some configuration controllers. HWICAP is a FIFO based and lightweight ICAP controller. It cannot be used in a system which needs low

reconfiguration time. ZyCAP and MiCAP-Pro are DMA-based ICAP controllers. The ICAP controller in this work depends on the same DMA-based architecture. It gives a good performance compared to others. BRAM_HWICAP has also high throughput but needs internal BRAMs.

All configuration controllers can be implemented on Zynq SoCs. Each has a considerable configuration speed except the Xilinx's HWICAP IP. ZyCAP is the based system for our ICAP controller but there is an extra state machine which determines the writing or reading mode.

5.3 Complete System Evaluation

Time overhead for whole process is sum of context saving, bitstream manipulation and context restoring. For a partial bitstream with size 799,564 bytes, it takes about 10.2 ms. Overall throughput is calculated as 74 MB/s. Obviously, bitstream manipulation is a bottleneck for this system. If bitstream manipulation is performed by overlapping other tasks so that its overhead is assumed to be zero, overall throughput would increase to 263 MB/s. Besides, it takes only 2.9 ms.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Contributions

A reconfigurable SoC hardware model is chosen in this thesis work to establish a system which is capable of context saving, bitstream manipulation and context restoring on hardware. While doing these on hardware, flexibility of a GPP is considered. Tasks are divided as software and hardware tasks but the management of hardware tasks is given to a software task like a supervisor. In designer's perspective, total development time is reduced to only software development time. Pre-implemented hardware tasks are loaded as if a new instruction set is assigned to GPP.

A commercial SoC Xilinx Zynq-7000 is selected to work on. Its configuration interfaces are analyzed and architectural background is reviewed for context saving applications. PCAP and ICAP interfaces are used to reconfigure the SoC. Some configuration controllers are analyzed to have lower overhead. It has been observed that ICAP controllers which use DMA engine have higher throughput than other alternatives because there is no bottleneck circuit which delays data transfer of configuration bitstream. A DMA-based ICAP controller is designed to be used in both reading and writing configuration data. It gives better performance than similar DMA-based solutions [28, 32] in terms of throughput and readback capability.

Context saving applications on reconfigurable hardware require preemption of a task when a higher priority task arrives. A preemptable hardware task model is proposed. The execution of the task starts under the supervision of a software task and context saving of running circuit is performed via ICAP controller. Partial bitstream structure is analyzed and both CLB states and BRAM contents are changed in DDR memory

to manipulate the bitstream. In task model, when context restoring is required, partial reconfiguration of the new bitstream is performed for the task to execute where it was saved from.

AXI protocol is used in many digital designs as a standart. A reconfigurable block model is designed to have both AXI4/AXI4-Lite and AXI4-Stream interfaces as slave side. An advantage of AXI protocol is to use available both commercial or free AXI4-based IPs. When a new AXI4-based IP is released for developers, it can be added to the design within a short synthesis time. Second advantage of AXI protocol is to give designer an addressable structure in which read/write data transactions are done via an address. Thus, each reconfigurable block has an address.

An FFT computation is demonstrated as an example on the designed system. Context saving, bitstream manipulation and context restoring is performed reliably with a good throughput. It is observed that the result for the given input is the same with the expected result. It shows that dynamic context switching is feasible on partially reconfigurable FPGAs.

6.2 Future Works

A hardware task can be saved from a reconfigurable partition on condition that it will be restored to the same partition. Since each partition has different circuit design due to routing and pin placement, hardware tasks cannot be relocated directly to any other available partition. Vivado provides logic locations of FF states and BRAM contents inside bitstream file. If it is possible to manipulate those bit locations, feasibility study for the relocation of a task can be conducted in Xilinx 7-Series FPGAs.

Bitsream manipulation overhead can be decreased with different architectures and algorithms because it depends on how ARM processor architecture is suitable to copy a word from one address to another address.

This complete system runs with the BareMetal OS. It can be used with one of Linux distributions to manage tasks and context switching on hardware.

REFERENCES

[1] F. Say and C. F. Bazlamaçcı, "A Reconfigurable Computing Platform for Real Time Embedded Applications," *Microprocess. Microsyst.*, pp. 13–32, 2012.

[2] G. Sklivanitis, A. Gannon, S. Batalama, and D. Pados, "Addressing Next-generation Wireless Challenges with Commercial Software-defined Radio Platforms," *IEEE Communications Magazine*, vol. 54, pp. 59–67, 2016.

[3] S. Gopinath, D. N. B. Balamurugan, and D. R. Harikumar, "Partial Reconfiguration using FPGA – A Review," in *International Conference on Innovations in Engineering and Technology*, pp. 139–144, 2016.

[4] S. Donthi and R. L. Haggard, "A survey of dynamically reconfigurable FPGA devices," in *Proceedings of the 35th Southeastern Symposium on System Theory*, pp. 422–426, 2003.

[5] K. Vipin and S. A. Fahmy, "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications," *ACM Comput. Surv.*, pp. 72:1–72:39, 2018.

[6] "Difference-Based Partial Reconfiguration, XAPP290." `https://www.xilinx.com/support/documentation/application_notes/xapp290.pdf`. Accessed: 2019-08-07.

[7] C. W. Yu, J. Lamoureux, S. J. E. Wilton, P. H. W. Leong, and W. Luk, "The Coarse-Grained / Fine-Grained Logic Interface in FPGAs with Embedded Floating-Point Arithmetic Units," in *2008 4th Southern Conference on Programmable Logic*, pp. 63–68, 2008.

[8] A. K. Jain, *Architecture centric coarse-grained FPGA overlays*. PhD thesis, Nanyang Technological University, 2017.

[9] Y. Birk and E. Fiksman, "Dynamic reconfiguration architectures for multi-

context FPGAs," *Computers and Electrical Engineering*, vol. 35, pp. 878–903, 2009.

[10] "Correcting Single-Event Upsets Through Virtex Partial Configuration, XAPP216." `https://www.xilinx.com/support/documentation/application_notes/xapp216.pdf`. Accessed: 2019-08-07.

[11] H. Michel, A. Belger, T. Lange, B. Fiethe, and H. Michalik, "Read back scrubbing for SRAM FPGAs in a data processing unit for space instruments," *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 1–8, 2015.

[12] A. Stoddard, A. Gruwell, P. Zabriskie, and M. Wirthlin, "High-speed PCAP configuration scrubbing on Zynq-7000 All Programmable SoCs," in *26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, 2016.

[13] "Configuration Readback Capture in UltraScale FPGAs, XAPP1230." `https://www.xilinx.com/support/documentation/application_notes/xapp1230-configuration-readback-capture.pdf`. Accessed: 2019-08-07.

[14] P. Deshmukh, A. Kurup, and S. V Kulkarni, "Effective use of Multi-Core Architecture through Multi-Threading towards Computation Intensive Signal Processing Applications," *International Journal of Computer Applications*, pp. 6–9, 2015.

[15] H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in *Proceedings International Parallel and Distributed Processing Symposium*, pp. 178–185, 2003.

[16] Qingxu Deng, Shuisheng Wei, Hai Xu, Yu Han, and Ge Yu, "A Reconfigurable RTOS with HW/SW Co-scheduling for SOPC," in *Second International Conference on Embedded Software and Systems (ICESS'05)*, pp. 116–121, 2005.

[17] "Vivado Design Suite User Guide Partial Reconfiguration, UG909." `https://www.xilinx.com/support/documentation/sw_manuals/`

xilinx2017_1/ug909-vivado-partial-reconfiguration.pdf. Accessed: 2019-08-07.

[18] V. Kizheppatt and S. Fahmy, "Architecture-Aware Reconfiguration-Centric Floorplanning for Partial Reconfiguration," in *8th International Symposium on Reconfigurable Computing: Architectures, Tools and Applications, ARC*, pp. 13–25, 2012.

[19] H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," *International Conference on Field Programmable Logic and Applications.*, pp. 223–228, 2005.

[20] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, pp. 318–331, 2015.

[21] "7 Series FPGAs Configurable Logic Block User Guide, UG474." https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf. Accessed: 2019-08-07.

[22] "7 Series FPGAs Data Sheet: Overview, DS180." https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. Accessed: 2019-08-07.

[23] "7 Series FPGAs Configuration User Guide, UG470." https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf. Accessed: 2019-08-07.

[24] S. Saha, A. Sarkar, and A. Chakrabarti, "Spatio-Temporal Scheduling of Preemptive Real-Time Tasks on Partially Reconfigurable Systems," *ACM Trans. Des. Autom. Electron. Syst.*, pp. 1–26, July 2017.

[25] M. Liu, Z. Lu, W. Kuehn, and A. Jantsch, "Reducing FPGA Reconfiguration Time Overhead using Virtual Configurations," *in ReCoSoC*, pp. 149–152, 2010.

[26] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration speed investigation and architectural design space exploration," *2009 International Conference on Field Programmable Logic and Applications*, pp. 498–502, 2009.

[27] "Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices, XAPP1159." https://www.xilinx.com/support/documentation/application_notes/xapp1159-partial-reconfig-hw-accelerator-zynq-7000.pdf. Accessed: 2019-08-07.

[28] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, pp. 41–44, 2014.

[29] K. Jozwik, H. Tomiyama, S. Honda, and H. Takada, "A Novel Mechanism for Effective Hardware Task Preemption in Dynamically Reconfigurable Systems," in *2010 International Conference on Field Programmable Logic and Applications*, pp. 352–355, 2010.

[30] S. Liu, R. N. Pittman, and A. Forin, "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller," in *FPGA*, pp. 292–292, 2009.

[31] K. Vipin and S. A. Fahmy, "A high speed open source controller for FPGA Partial Reconfiguration," in *2012 International Conference on Field-Programmable Technology*, pp. 61–66, 2012.

[32] A. Kulkarni and D. Stroobandt, "MiCAP-Pro: a high speed custom reconfiguration controller for Dynamic Circuit Specialization," *Design Automation for Embedded Systems*, pp. 341–359, 2016.

[33] S. G. Hansen, D. Koch, and J. Torresen, "High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro," *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 174–180, 2011.

[34] S. Jovanovic, C. Tanougast, and S. Weber, "A Hardware Preemptive Multitasking Mechanism Based on Scan-path Register Structure for FPGA-based Reconfigurable Systems," in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pp. 358–364, 2007.

[35] D. Koch, C. Haubelt, and J. Teich, "Efficient Hardware Checkpointing: Concepts, Overhead Analysis, and Implementation," in *Proceedings of the 2007*

ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays, pp. 188–196, 2007.

[36] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, p. 145, 2004.

[37] "Vivado Design Suite User Guide Programming and Debugging, UG908." https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug908-vivado-programming-debugging.pdf. Accessed: 2019-08-07.

[38] F. Duhem, F. Muller, and P. Lorenzini, "FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA," in *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 253–260, 2011.

[39] A. Morales-Villanueva and A. Gordon-Ross, "On-chip Context Save and Restore of Hardware Tasks on Partially Reconfigurable FPGAs," in *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 61–64, 2013.

[40] C. J. Morford, "BitMaT - Bitstream Manipulation Tool for Xilinx FPGAs," Master's thesis, Virginia Polytechnic Institute and State University, 2005.

[41] K. Dang Pham, E. Horta, and D. Koch, "BITMAN: A tool and API for FPGA bitstream manipulations," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 894–897, 2017.

[42] "Vivado Design Suite AXI Reference Guide, UG1037." https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf. Accessed: 2019-08-07.

[43] "Zynq-7000 SoC Technical Reference Manual, UG585." https:

//www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf. Accessed: 2019-08-07.

[44] M. Pagani, A. Balsini, A. Biondi, M. Marinoni, and G. Buttazzo, "A Linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pp. 96–101, 2017.