A Framework for Dynamic Partial Reconfiguration of Xilinx 7 Series FPGA/SOC

Author

Muhammad Usama Iqbal

Regn Number: 00000171114

Supervisor

Dr. Shoab A Khan

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

March, 2020

A Framework for Dynamic Partial Reconfiguration of Xilinx 7 Series FPGA/SOC

Author

Muhammad Usama Iqbal

Regn Number: 00000171114

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Engineering

Thesis Supervisor:

Dr. Shoab A Khan

Thesis Supervisor's Signature:_____

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

March, 2020

Declaration

I hereby certify that I have developed this thesis titled as *"A Framework for Dynamic Partial Reconfiguration of Xilinx 7 Series FPGA/SOC"* entirely on the basis of my personal efforts under the sincere guidance of my supervisor Dr. Shoab A Khan. All of the sources used in this thesis have been cited and contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Signature of Student
Muhammad Usama Iqbal
00000171114

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student
Muhammad Usama Iqbal
00000171114

Signature of Supervisor

Dr. Shoab A Khan

Copyright Statement

Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful

Alhamdulillah, all praises to Allah for the strengths and His blessings in the timely completion of this thesis. I would like to offer my sincerest gratitude to my supervisor Dr. Shoab A Khan, Committee Members Dr. Sajid Gul Khawaja and Dr. Arslan Shaukat for their supervision and tremendous support. Without their guidance and persistence help this dissertation would not have been possible.

Abstract

Dynamic Partial reconfiguration is extensively used for applications which requires the adaptive behavior of a system at run time. Almost all SRAM-based field programmable gate arrays (FPGAs) introduced in the last decade, supports partial re-configuration (PR). Resource utilization in the FPGAs can optimized using partial reconfiguration. Most of the resource hungry algorithms comprises of multiple steps which are implemented as separate modules in FPGAs. these modules are usually not used concurrently. Therefor all those modules are never required to be present in the FPGA at the same time. Partial reconfiguration allows to utilize the hardware resource of one module by the other.

Partial reconfiguration is ideal for applications which require huge number of computations and can afford a reasonable time to achieve this with a smaller/cheaper FPGA device. These techniques have been studied in detail but their use is yet not very famous in the industry. The goal this research is to design a framework which will allow to take advantage of the hardware architecture and integrate as much logic as possible without using extra hardware resource. A design example of jpeg compression is used to demonstrate the procedure involved in the utilization of technique and to show the advantages that can be achieved with the help of dynamic and partial reconfiguration-based framework. In addition, the effectiveness of the proposed methodology is quantified by comparing the FPGA resource utilization of the original JPEG compression design and that of the partial reconfigurable prototype.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

Field programmable gate arrays (FPGAs) were initially limited to be used for implementing glue-logic to platforms for implementing advanced mixed software-hardware systems-on-chip (SOCs). As their abilities and sizes have increased, FPGAs are being used in a wide range of domains, where their ability to reprogram offers a diverse advantage over fixed application specific integrated circuit (ASIC) implementations. This ability allows hardware designs to be re-purposed or reprogrammed even after deployment. An even more differentiating feature of FPGAs is their advanced feature of dynamic programmability, whereby their functionality is changed at runtime in response to application requirements. SRAM-based FPGAs in their conventional use are the hardware devices that require to be programmed once they are powered on. This type of programming the FPGAs is called static reprogramming. In most of the FPGA based products, the FPGAs are programmed via non volatile storage media. This storage media contains the configuration file for the whole FPGA. Once that configuration is copied onto the SRAM of FPGA, it starts functioning and keeps performing the same functionality until it is powered off. It does not change the configuration at run time even though most of these devices has the capability to do so. FPGAs can reprogram themselves at runtime and perform a totally different function. In order to do so it completely rewrites its SRAM configuration memory. An external storage media is required which contains the multiple configurations these configurations can also be updated or added remotely.

In the recent years a more advanced option of dynamic reconfiguration has been introduced by the FPGA manufacturers where FPGAs can configure portions of their configuration memory which in other terms means that the portion of the FPGAs hardware will change its functionality while the rest of the hardware keeps performing according to initial configuration. The flexibility of on-site programming of an FPGA reduced the requirement of refabricating for a modified design. Partial Reconfiguration (PR) takes this flexibility even further as it allows the amendment of an operational FPGA design by loading a partial configuration file, usually a partial BIT file. Hence Partial Reconfiguration is the process of modification of a functional FPGA by loading a partial bit file. To avail this option full BIT file is first required to configure the FPGA, later partial configuration files can be loaded to alter reconfigurable regions in the

FPGA without compromising the integrity of the applications that was currently running on those parts of the FPGA that are not part of the reconfigurable region. Figure below shows a placement of reconfigurable module within the FPGA.

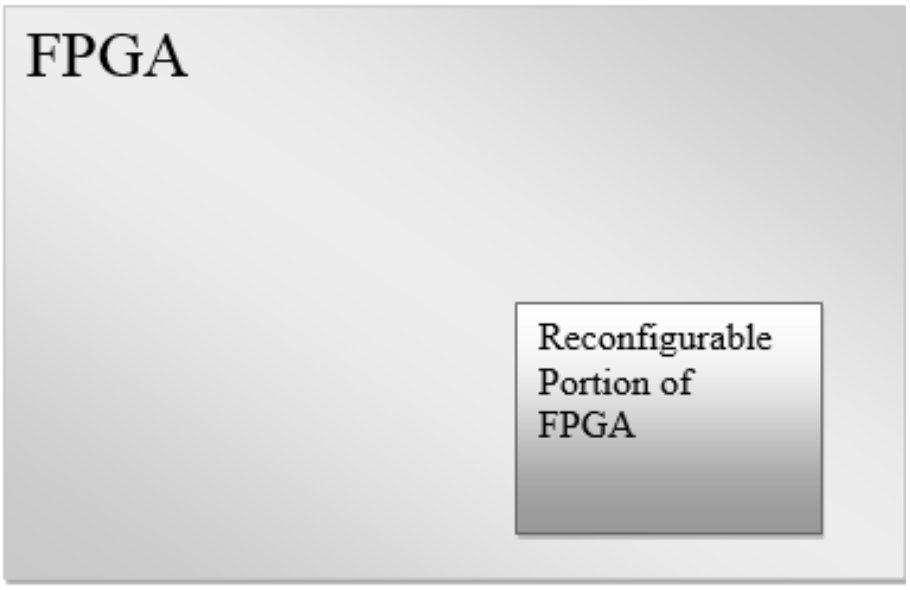

Figure 1: Reconfigurable Block Within an FPGA

This thesis presents a framework that helps in utilizing the dynamic partial reconfiguration ability of the FPGAs/SOCs in order to implement resource hungry algorithms/application on a relatively small FPGA device.

Basic structure of all the Field Programmable Gate Array devices is composed of two separate layers: the configuration memory layer and the hardware layer [1] where the logic mapped on, as shown in Figure 2. FPGAs devices achieve this flexible ability to reprogram themselves because of this composition. The hardware layer contains the computational hardware resources, including flip-flops, lookup tables (LUTs), memory blocks, digital signal processing (DSP) blocks, transceivers, and others. Hardware layer is also responsible in the formation of a complete logic circuit using the routing resources along with other components such as muxes and gates. Some dedicated hardware resources such as Digital Clock Manager (DCM), configuration ports such as ICAP and PCAP are also part of the hardware layer.

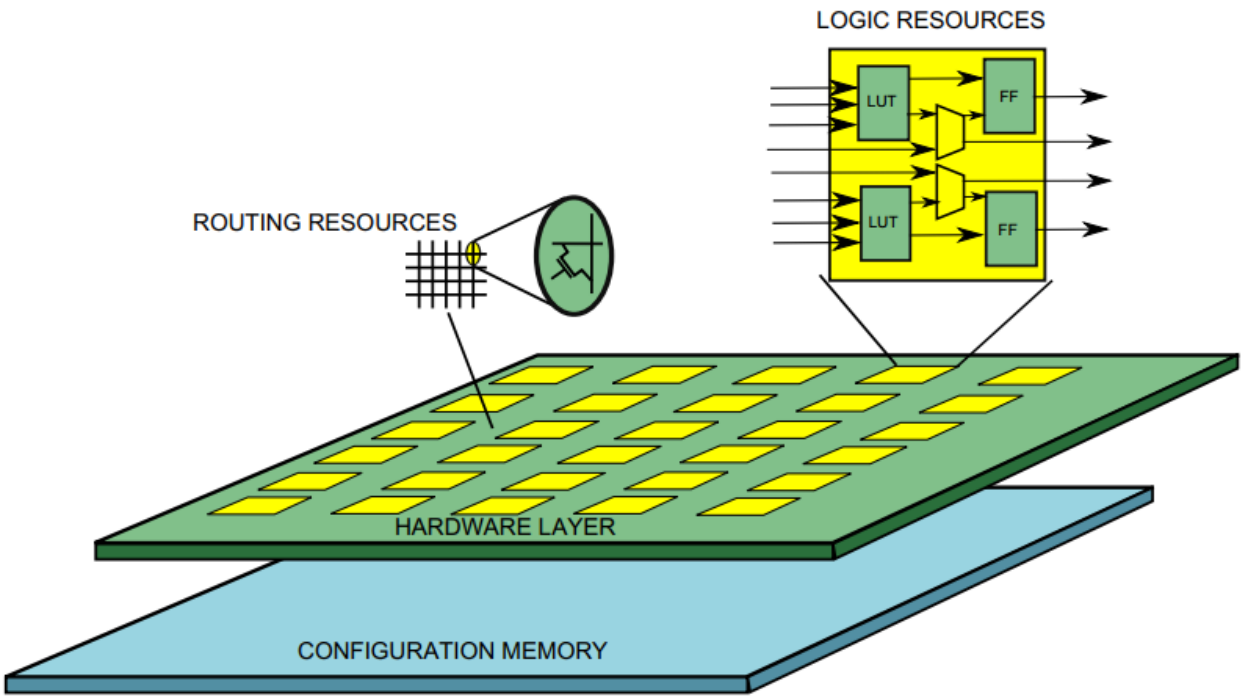

Figure 2: FPGA Architecture based on configuration memory layer and hardware resource layer

Configuration memory of these FPGA devices is SRAM based therefore the on-chip configuration data is lost when the device is powered off as SRAMs are volatile memories. So on each power up the FPGA's static configuration file is downloaded externally either from a nonvolatile configuration flash memory, or via JTAG programmer.

1.1 Motivation

Even though the effectiveness of FPGA's ability to dynamically reconfigure and partially reconfigure its hardware resources has been demonstrated widely in research community, these techniques are not commonly used in industrial products because of the implementation challenges, architecture and design. And relatively lesser available support makes it an even harder decision to go for. Whereas the benefits of utilizing these features of FPGAs are far more significant and surpasses all the complexity involved in the procedure. Therefore, the main objective of this thesis is to demonstrate the effectiveness of partial reconfiguration in implementing resource hungry algorithms on smaller and cheaper circuits. An example of JPEG compression is used to demonstrate the results and resource utilization on Xilinx Zynq device.

The procedure followed in this example will help to overcome the challenges faced in procedure of dynamic partial reconfiguration.

1.2 Research Question

How can Dynamic Partial Reconfiguration help in implementation of hardware hungry algorithms on smaller/cheaper FPGA/SOC devices?

The latest Xilinx 7 series FPGAs supports the dynamic partial reconfiguration feature. It can help solving many problems concerned with the resource utilization in FPGAs and reduce the cost of redesigning a hardware in many products.

1.3 Hypothesis

We are conducting this research on a Xilinx Zynq SOC. Xilinx offers the advance feature of dynamic partial reconfiguration it Zynq family devices. We will use the PCAP interface provided by the Xilinx and design a frame work for the efficient implementation of complex and hardware extensive algorithms on low end devices. Or target device in this research is Xilinx Zynq-7000 AP SoC XC7Z020-CLG484 which available on ZedBoard [11].

1.4 Objective and scope of Thesis

The objective of this investigational undertaking are as follows:

- To develop understanding of the complex architecture of FPGA bitstream, how it is translated on to the FPGA fabric which includes the hardware layer of LUTs.
- Explore the available options to perform dynamic partial reconfiguration such as PCAP and ICAP and examine the complexity involved in process, study the advantages and disadvantages of using these methods.
- Compare the complexity and efficiency of ICAP and PCAP resource of Xilinx FPGA/SOC
- Devise a framework to make use of these features to enhance the efficiency of any hardware accelerator.
- Demonstrate the designed framework with a particle example and perform all the steps of dynamic partial reconfiguration design either by using the Internal configuration port or by using the more advanced Processor Configuration Access port.
- Prepare a demo of dynamic partial reconfiguration on ZedBoard.

1.5 Challenges

Dynamic partial reconfiguration is complex technique which is not usually preferred because of the complexities involved in the methodology. Therefore, there is very little practical information available on the internet regarding this topic. In order to implement a dynamic partial reconfiguration design on a hardware there are lot of details required such as structure of FPGA bitstreams which is kept confidential by the vendors. Since these methods are mostly practiced by the military products development so the FPGA vendors do not provide the related software data in Pakistan. We will have to rely on the data gathered by the third-party sources.

Furthermore, the Bitstream structures of the latest FPGAs/SOCs are far more complex and data related these devices about dynamic partial reconfiguration is even more scarce. For example, if in case of Xilinx FPGAs, Xilinx provide some basic level details on how to start with the dynamic partial reconfiguration design, but any attempt to access the application notes is nullified by Xilinx with error:" We *cannot fulfill your request as your account has failed export compliance verification*".

Another major challenge involved in this field is that there is no proper way to simulate the dynamic partial reconfiguration-based design. All FPGA vendors provide the simulation software mainly for the timing verification of the implemented design. But the dynamic partial reconfiguration is low level device operation that cannot be simulated like other designs. Vendors do suggest some workarounds such as visually observing the effects of reconfiguration or partial reconfiguration on the hardware by switching in multiple bit file or partial files. However, these methods do not represent the actual behavior of dynamic partial reconfiguration process.

1.6 Structure of Thesis Report

The report is organized as follows:

- Chapter 2 contains the detailed overview of dynamic partial reconfiguration process. It discusses the different available popular vendors of FPGA and their support for partial reconfiguration.
- Chapter 3 is about the related work already done regarding dynamic partial reconfiguration process.
- Chapter 4 is about the technique and algorithm used for the design of dynamic partial reconfiguration framework in this thesis. It explains all the details of methodology used.
- Chapter 5 explain the technique with help of a case study i.e. JPEG compression algorithm implementation on ZedBoard with the help of partial reconfiguration
- Chapter 6 concludes the results and comparisons of conventional techniques vs the implementation performed with dynamic partial reconfiguration.
- Chapter 7 includes the Future work.

CHAPTER 2: DYNAMIC PARTIAL RECONFIGURATION

Dynamic partial reconfiguration is the ability of the Field Programable Gate Arrays (FPGAs) or latest System on Chips (SOCs) to reconfigure/reprogram a portion of their hardware themselves on the fly. It means once a system with an FPGA or SOC based design has been deployed in the field or made available in the market, it can be reconfigured partially. By partially here means that a part of the FPGA/SOC will keep working normally (static region), where as a portion (reconfigurable region) of its hardware will be reprogrammed. The logic implemented in the static portion can grab new configuration file for the reconfigurable region over ethernet or any other communication medium. Its will than store the partial configuration file (known as partial bi file) on a nonvolatile storage device. Reconfiguration controller embedded in the static part of the FPGA can than reconfigure the reconfigurable region any time.

This ability of the FPGAs to reconfigure themselves makes them stand out and a preferred choice while designing an embedded system. The basic hardware architecture of FPGAs is the reason behind the availability of this unique feature. Basic structure of all the Field Programmable Gate Array devices is composed of two separate layers: the configuration memory layer and the hardware layer [1] where the logic mapped on, as shown in Figure 2. FPGAs are reprogrammed via writing the bits into the configuration memory. This configuration memory is arranged in small blocks called frames. A frame is the smallest addressable unit in configuration memory of the FPGA. Multiple frames are required to program a single tile of FPGA, Size of the configuration frame can vary for different device family. Currently there are two major vendors supporting partial reconfiguration in their FPGAs. Those are Xilinx and Altera (now owned by intel).

1.1 Xilinx

Out of all the FPGA vendors, Xilinx's FPGAs devices are the most popular ones and have supported PR for two decades, Dynamic partial reconfiguration was first supported in Xilinx XC6200 series FPGAs [6]. This device's architecture composed of a single configurable memory plane. It had a tiled architecture and each tile was separated into a number of cells holding

functional cells as shown in Figure 3. The composition of functional cells consisted of following elements:

a) 2:1 multiplexer for combinational logic
b) A flip-flop
c) Routing resources

With the configuration SRAM mapped to an external processor address space, any functional cell could be accessed by the processor using a special interface and modify its configuration. Due to a consistent routing scheme and regular structure with every cell, reconfiguration was easier with these devices than for current ones. Circuit relocation in run-time was also possible with these devices.

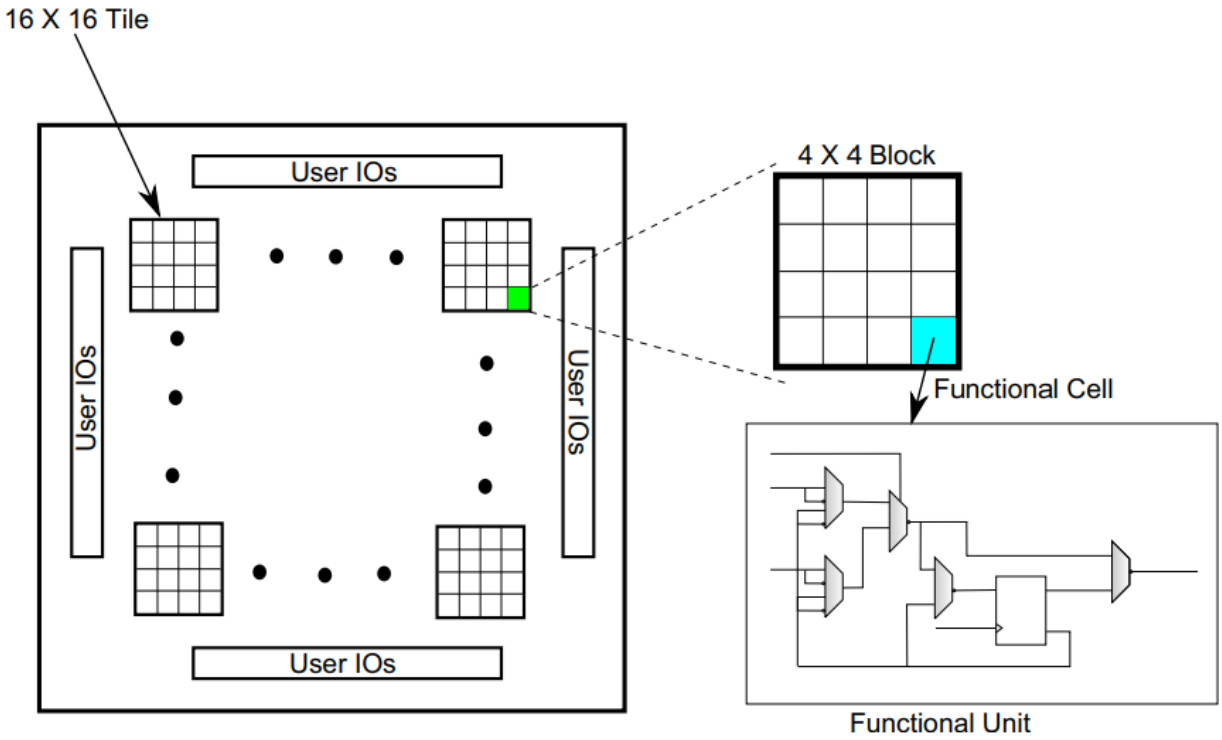

Figure 3: Xilinx XC6200 architecture

Dynamic partial reconfiguration gained more popularity when Xilinx introduced Virtex-II and Virtex-II pro series FPGAs [7]. A columnar architecture was introduced in these devices in which all the FPGA primitives were arranged in a columnar fashion. These primitives include:

a) Configurable Logic Blocks (CLBs)

b) Block RAMs

c) Multipliers

Configurable Logic Blocks (CLBs) are the basic logic elements in Xilinx FPGAs, CLBs consists LUTs and flip-flops in two slices. The number of flip-flops and LUTs in a slice is dependent on the device family. Figure 4 shows Xilinx device architecture with independent resources blocks highlighted in hierarchy

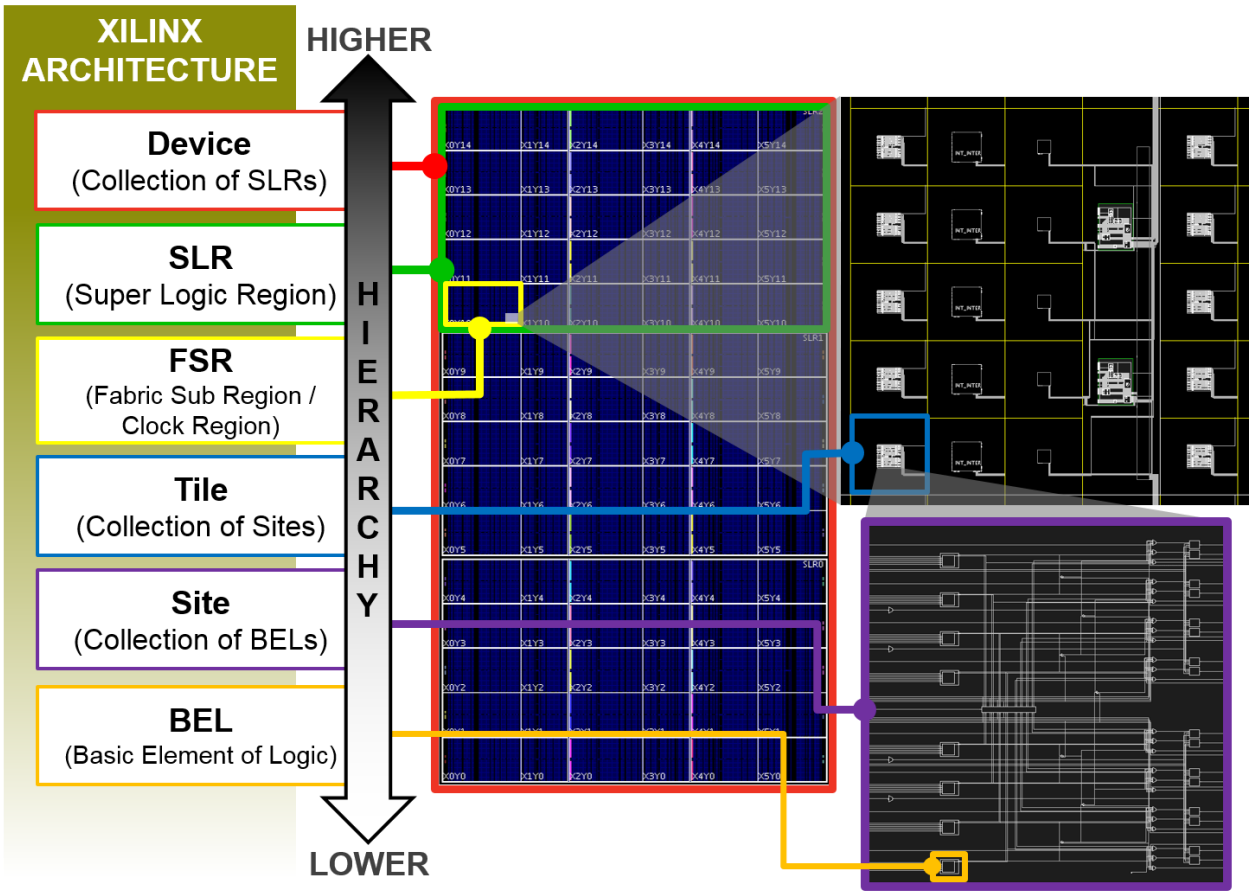

Figure 4: Xilinx device architecture hierarchy

A configuration binary file (known as bitstream or partial bitstream in Xilinx terminology) can be downloaded externally using the JTAG interface. Xilinx introduced a new configuration interface called the Internal Configuration Access Port (ICAP). With the help of ICAP is was possible to download bitstreams from within the fabric of FPGA thus eliminating the

requirement of an external processor for reconfiguration purpose. A built in soft-processor or even a custom designed state machine could get the configuration data from external memory and transfer it to the configuration memory of FPGA through the ICAP, thereby letting a circuit instigated on the FPGA to amend itself without external interference. In vertix device series the configuration memory is divided in smaller segments called frames, these frames are 1 bit wide and their length is equal to the height of the device therefore a frame's length may vary in case of a different device. All the resources that lies in a single narrow vertical column of the device would be configured by the single frame. Furthermore, the frames are grouped in 6 different types for different configuration columns on the basis of the corresponding hardware mapping. Configuration columns 6 types are:

a) **IOB:** These columns configure the voltage standard, internal pull-up, and other options for the I/O interfaces.

b) **IOI:** All other IOB logic such as capture and send FFs

c) **CLB:** These columns program the configurable logic blocks, routing, and most interconnect

d) **GCLK:** Global clock routing resource (not supported in 7 series)

e) **BlockRAM:** These columns program the small internal memory blocks

f) **BlockRAM Interconnect:** Provide interconnection between multiple BRAM blocks

In order to configure a portion for partial reconfiguration in Xilinx devices, it is required to select the specific region that will undergo PR. These portions of FPGA are called partially reconfigurable regions (PRRs) and usually consists several frames. This region can be selected from the device view of Floorplanning tool. There are some limitations on the size and shape of Partially Reconfigurable Regions (PRRs). In case of Virtex-II PRRs must be of full height of the device and horizontally they should also align with a multiple of four slices. These restrictions make the architecture simple but they can cause hurdle in efficient system design in terms of hardware utilization. Because PRRs spread to the height of device, floorplanning is only

concerned with the thickness of partial reconfigurable regions and therefore these regions are shaped as vertical slots as shown in Figure 5.

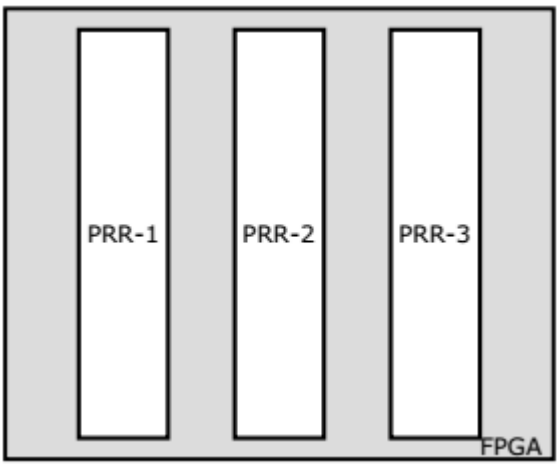

Figure 5: PRR in Xilinx Virtex-II Device using Floorplanning

Xilinx Virtex family devices that were launched after Virtex-II incorporated improvements in their architecture. The earlier device family used tri-state buffers in order to manage connectivity of partially reconfigurable regions. The number of these tri state buffers were also limited due to fixed position causing more restriction in size and positions of PRRs. This problem was solved in Virtex-4 where bus macros are used instead of tri state buffers as shown in Figure 6. In this figure the CLB slices on the right side are implemented in the static region and the ones on the left side of the module boundary lies in the reconfigurable region. This modification permitted for a more flexible planning of connectivity.

Another major improvement on Virtex-4 was change in frame size [8]. Unlike the variable size frame of Virtex-II on the basis of device, the frame size is constant in Virtex-4. All the frames are 1 bit wide and 16 CLB high. This modification allowed more flexible design options in the floorplanning because now the frame height is not required to be equal of the device height. It also increased the complexity of design process because the PRR is now a two-dimensional task in floorplanning tool. Another considerable change in architecture regarding the partial reconfiguration was increase in the bus width of ICAP from 8 to 32 to achieve better throughput during reconfiguration.

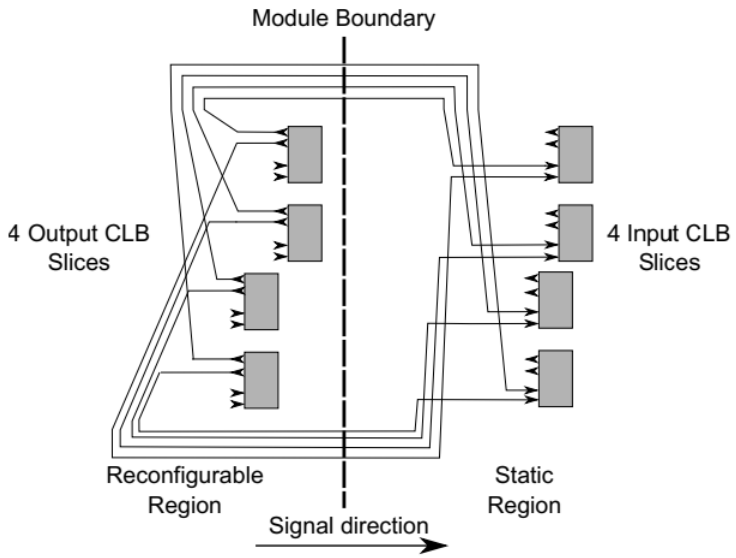

Figure 6: Bus macro connectivity between static and reconfigurable region

Xilinx made further architectural modifications in Virtex-5, the device is separated into multiple rows and columns as shown in Figure 7. A row basically denotes a clock region and device size controls how many clock regions are there. The columns also known as blocks, extent the entire device height. Each column consists of a sole type of FPGA primitive organized in a columnar fashion.

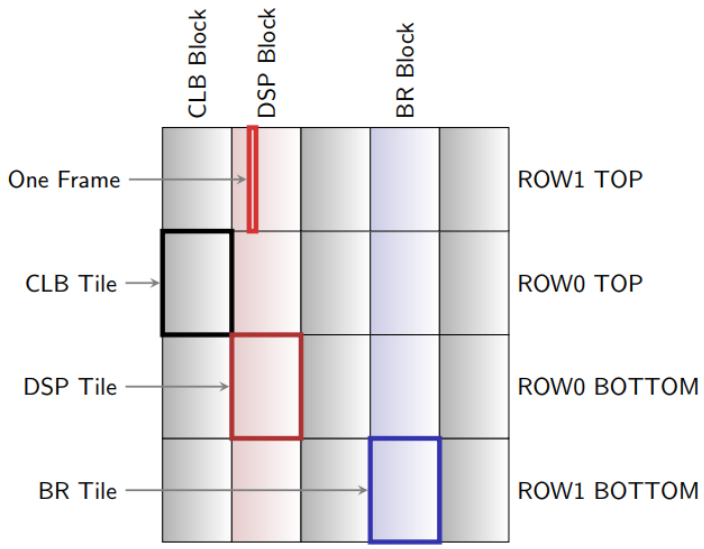

Figure 7: Xilinx Virtex-6 and above FPGA architecture

There are several tiles in FPGAs where intersections of rows and columns occurs e.g. CLB tiles, DSP tiles, and BRAM tiles. Xilinx denote these tiles with the term reconfigurable frame. These reconfigurable frames are the smallest unit for PR. Xilinx Virtex-6 and Xilinx 7-series FPGAs (Artix, Kintex and Virtex-7) have the similar basic architecture as Virtex-5. Number of CLBs, DSP slices and Block RAMs in CLB tile, DSP tile and BRAM tile are shown in Table 1.

Xilinx Device	**CLBs**	**DSP Slices**	**Block RAMs**
Virtex-5	20	8	4
Virtex-6	40	8	8
7-series FPGAs	50	10	10

Table 1: Number of CLBs, DSP slices and Block RAMs in Xilinx FPGA's Tiles

Figure 8 show the CLBs, IOBs, DSP48 slices BRAMs and DCM allocation in the Vivado device view for a Xilinx 7 series FPGA.

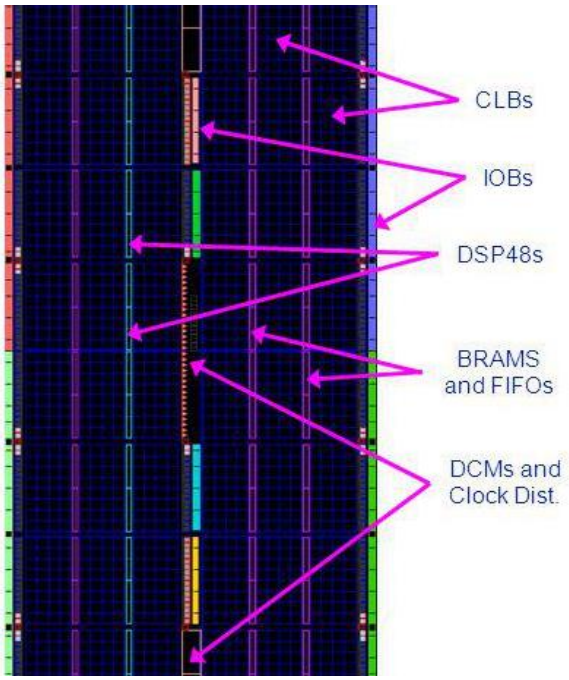

Figure 8: Vivado device view of 7 series FPGA

These complex architectures introduced complexity in design phase. It also makes some tasks like run-time circuit relocation almost impossible, but with help of these enhanced architectural modification in field programmable gate arrays users are able to implement complex logic problems and reduce the resource wastage. In terms of partial reconfiguration more PRRs can be defined of variable sizes with all sort of resources.

1.1.1 7-Series Configuration Logic Block Overview

7 series FPGA provide advance, High performance CLB Architecture; which contain real 6 input LUT technology. That can also be used as dual 5-input LUT option. There are two type of slices in each CLB, SliceL and SliceM; SliceL is used for logic and carry only while SliceM has the capability to form distributed RAM and shift register. Single LUT in Slice M can be a 32-bit shift register 64×1 RAM. Approximately two-third of the Slices are SLICEL logic cell and rest are SLICEM (Xilinx, 7 Series FPGAs Configuration Logic Block, 2016).

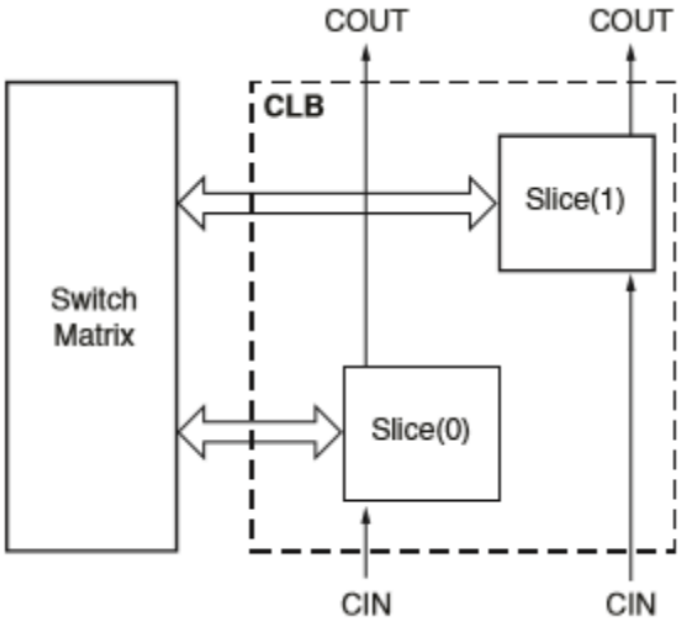

Figure 9: Arrangement of Slices with CLBs

7 series CLB Architecture has columnar architecture which was not present in previous FPGA architecture Virtex 6 and Spartan 6 that allow scalability to higher densities and allow more routing between CLBs. Routing resources are also increased in size with respect to Virtex 6

family that improves the quality of automatic place and route. So each slice in CLB contain four Look up Table (LUT) and each LUT has two FlipFlop so four 6 input LUT, 8 Flip-Flops as well as multiplexer and carry-logic form a Slice. Two Flip-Flop per 6-LUT is excellent for heavy Pipelined Design.

1.1.2 CLB/SLICE Configuration

Following table summarize the Logic resources in one CLB. And SliceM LUT can be configured as a Look up Table, distributed RAM and shift register.

Slices	LUTS	Flip-FLOPS	Arithmetic and Carry chain	Distributed RAM	Shift Registers
2	8	16	2	256bits	128bits

Table 2: 7-Series SLICE Configuration

1.1.3 Look Up Table (LUT)

Look up table in 7 series FPGA architecture is used to implement any 6 input or 5 input arbitrary Boolean function. It can be used individually by adding primitive in the design but its location must be identified in constraint file. Each LUT location in configuration memory is determined by the frame address register which is determined by translating slice location to frame address register. Slices in FPGA device is places in X, Y coordinate where X is row and Y is Column of SLICE

Xilinx has introduced a new family of devices with hybrid architecture [9] which includes a core of ARM Cortex-A9 processor known as Processor System (PS) along with a 7 series FPGA device known as Programmable Logic (PL) on a single chip. This family of devices is known as Zynq SOC. Block level architecture of Zynq device is shown in Figure 10. The ARM processor in Zynq can communicate with the rest of the resources using Advanced eXtensible Interface (AXI).

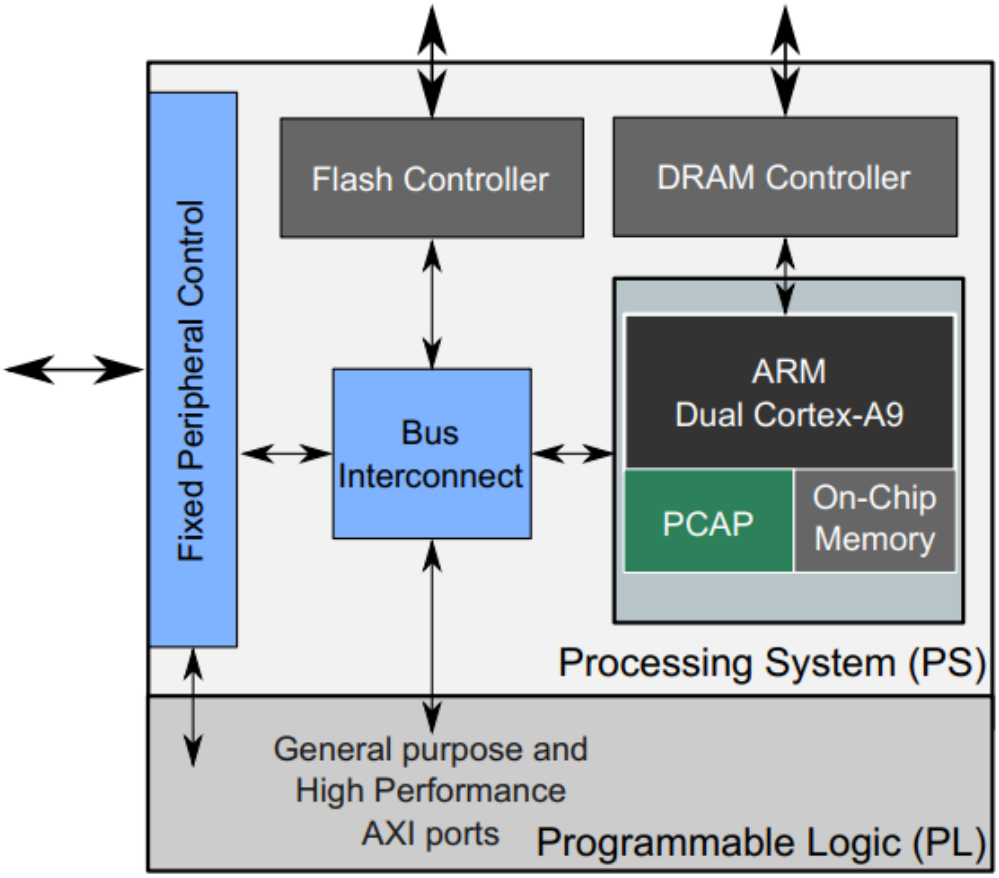

Figure 10: Xilinx Zynq SOC Architecture

There are multiple AXI interfaces are present on Zynq devices in order to facilitate the communication between ARM processor and Programmable Logic (PL). AXI interconnect offers high bandwidth for the 2 major components of Zynq device i.e. PS and PL. Zynq family devices also supports the dynamic partial reconfiguration. In Zynq architecture Xilinx added a new option to reconfigure the programmable logic portion via processor configuration access port (PCAP) port. It is controlled by the ARM cortex processor for complete or partial reconfiguration of PL. The programmable logic in Zynq is based on the Xilinx 7-series architecture. Therefore, the reconfiguration using the ICAP is also available in the Zynq.

1.1.4 Reconfiguration Modes from Xilinx

The reconfiguration time is dependent on the size of the bitstream/partial bitstream being programmed. Table 3 summarize the different modes available for loading the bitstream on an FPGA.

Configuration Mode	Max Clock Rate	Data Width	Max Bandwidth
ICAP	100MHz	32 bit	3.2Gbps
PCAP	100MHz	32 bit	145MBps
SelectMap	100MHz	32 bit	3.2Gbps
Serial Mode	100MHz	1 bit	100Mbps
JTAG	66MHz	1 bit	66Mbps

Table 3: Attributes of Configuration modes

In case of the Xilinx Zynq family devices PCAP is a preferred choice as it can be controlled from the PS and offers advanced security features as compared to ICAP and other configuration modes. Still the ICAP offers more speed in comparison with the PCAP. There are hybrid models proposed by researchers where ICAP is accessed from the PS using an AXI interface and some logic is implemented on the programmable logic in order to translate the PS generated commands for execution on ICAP. [10] represented a similar hybrid reconfiguration controller ZyCAP, it befits from the high bandwidth of ICAP and provides a software based accessed from the PS. Details of these reconfiguration modes is further discussed in chapter 3.

1.2 Altera

Altera which is now owned by intel also provides the features of partial reconfiguration in their and Arria-10 series, Cyclone-V and Stratix-V FPGAs. In altera FPGAs Adaptive Logic Modules (ALMs) are the basic building blocks just like the CLBs in Xilinx. ALMs contain 8 input LUTs, 4 flip flops along with the auxiliary circuit of multiplexers and adders. Logic Array Blocks (LABs) are formed with the combinations of multiple ALMs. LABs are arranged in a columnar fashion in the device. In Altera devices partial reconfiguration is supported for routing resources, DSP slices, logic elements and memory blocks.

Altera's Stratix-V compose of the similar architecture as that of Xilinx Virtex series FPGAs. Programming frame is smallest reconfigurable portion of Stratix-V FPGAs [12]. These devices are divided into multiple columns but arranged in a single row only which causes more restrictions and PR based designs e.g. a PR region can not be extended to the complete height of device and also contains memory blocks within the region as shown in Figure 11.

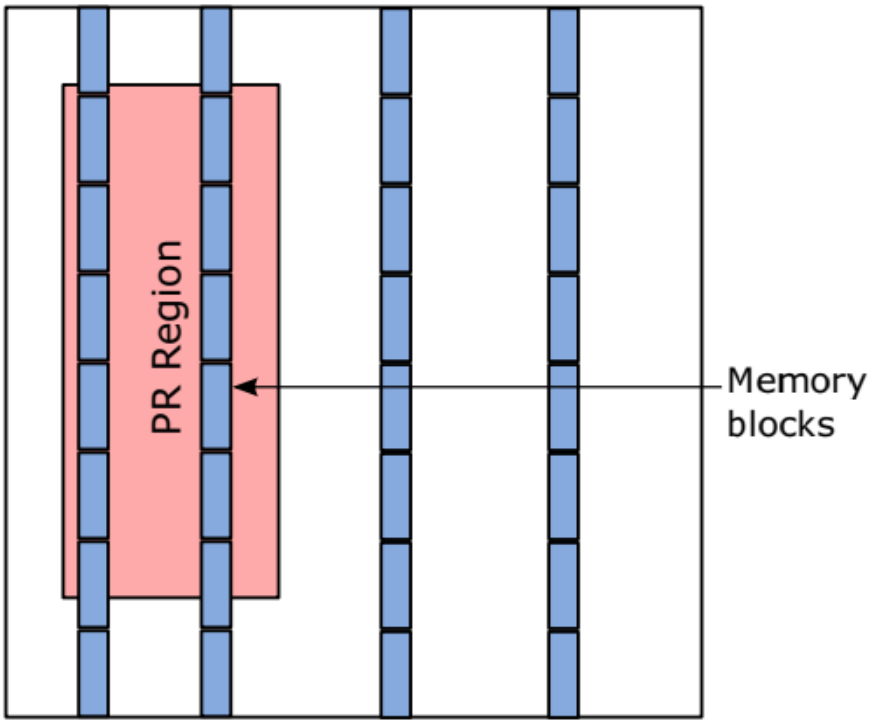

Figure 11: A reconfigurable region in a Stratix-V FPGA

Because of these restrictions when a PR region is being reprogrammed it also over writes the memory contents that even lies outside the PR region but the same column. Only way to avoid such issues is to avoid using the resources of a column in static logic, that has a PR region in it. This can lead to extensive waste of resources in some cases.

The newly introduced Arria 10 FPGA and SOCs from Altera also supports partial reconfiguration. These devices contain ARM processor along with the programmable logic fabric similar to Xilinx Zynq devices. The reconfiguration process is yet similar to that of Stratix-V FPGAs. Altera has provided the IP-block in order to load the partial bit stream data on to the configuration memory either from external hosts or from the integrated PR controller [13]. These IP-blocks support the data width of 1-32. Altera has also provided the ability to reconfigure PR region via PCIe hard macro [14].

Altera has announced Altera Stratix 10 devices which has some advancements in their structure regarding partial reconfiguration. The new design architecture of these devices has multiple sectors and each sector has its own reprogrammable infrastructure along with separate configuration memory [15]. There are small processors to accommodate the reconfiguration in all the sectors. These small processors are called Secure Digital Managers (SDMs). The partial bit streams of these sectors are identical so it offers an amazing feature of relocation bitstreams within an FPGA resulting in higher reconfiguration bandwidth. These devices are not commercially available yet.

CHAPTER 3: LITERATURE REVIEW

FPGA's were designed to remove the limitations of ASIC's by providing the flexibility to reprogram and reconfigure in run-time; as per user needs. With the emergence of cloud computing and IoT's FPGA's are now widely used as hardware accelerators to off-load intensive computational tasks to FPGA's reducing time and saving resources.

BBC; one of the radio channel pioneers is also utilizing Zynq SoC to leverage FMC for audio codec replacement, switching 32-years old rack-mount audio codec with commercial of the shelf component. Example of such applications is Enyx, a company which provide users with an online FPGA hardware acceleration services by providing them with software/hardware experience of their own choosing. User can just drag and drop the type of service desired and it will be provided in shape of Intellectual property (IP's) tailored specific to their needs.

2.1 FPGA Configuration Architecture

FPGA design is implemented and then mapped by the configuration memory as configuration memory hold the functionality of the design. Configuration memory architecture details are discussed later in this chapter. FPGAs are categorized in term of type of memory technology used for configuration memory which are; SRAM, flash and antifuse. Majority of the FPGAs in market are SRAM based, Xilinx also provide SRAM based FPGA. As the SRAM memory is volatile so FPGA's are required to be reprogrammed every time when device is powered up and external memory is used for application storage.

A digital design is implemented in FPGA using logical resources, memory elements and interconnects provided by the FPGA. When synthesized bitstream is generated and loaded to configuration memory to program the device. So configuration memory contain the configuration information that is mapped to FPGA device. There are certain ways to load external full-reconfiguration data or partial reconfiguration data ranging from serial to parallel. To control the resources like LUT and Flip-Flop configuration memory is accessed from outside or inside the FPGA. In view of above discussion, we can say that the FPGA architecture is divided into three configuration layer; Functional layer, Configuration Layer and Clock layer. As

the details are described below but we can see in the following figure that the layers are connected to each other through interconnect Clock layer is connected to functional layer through programmable interconnect point (PIP) and the logic on functional layer is mapped to configuration memory through configuration port.

2.1.1 Functional Layer

Functional Layer comprises the user resource Logic like CLB, BRAM and DSP etc. Which are connected through switch box and connection box. Connection Box is used to directly connect the programmable resources to Input/output. While switch box is used to connect the connection box, vertically and horizontally allowing wide routing network. Circuit designed on FPGA is made by the logic implemented using resource block and then configured by loading bit stream to configuration memory. In some architecture like Zynq and ultrascale series functional layer also include processor like CPUs or GPUs, these are the hard cores. But sometime if hardcore is not provided processing functionality can be achieved using CLBs, BRAM and DSPs as in microblaze processing system.

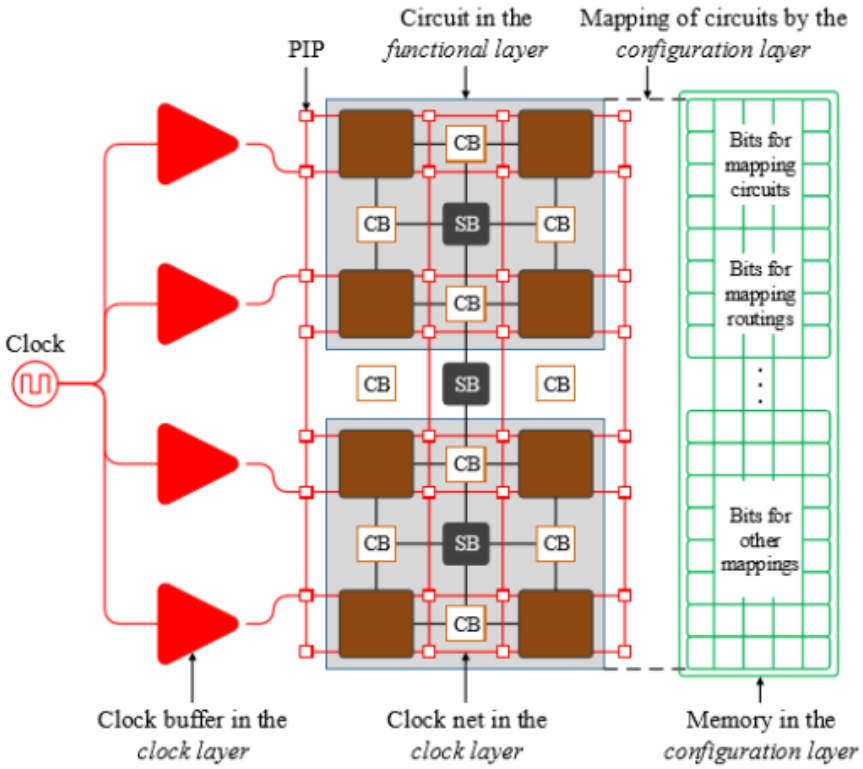

Figure 12: Xilinx 7-Series FPGA Layes

2.1.2 Configuration Layer

Configuration layer determine the functionality of Logic implemented on FPGA circuit. It consists of Configuration memory, Configuration Port and Control logic to load bit stream into configuration memory through configuration Port. Configuration memory is composed of configuration frame that specify each bit of the design from top to bottom of a clock region. Each frame contains (101×32) 3232 bits which are stored in SRAM cell. The configuration frame is the smallest unit of configuration memory so to make changes in configuration memory the whole frame is required. So multiple frames are required to reconfigure the amount of resources in a design.

2.1.3 Clock Layer

FPGA device has dedicated Clock network for synchronization of CLBs and BRAM etc., This Layer can be considered as part of functional Layer. In 7 series FPGA architecture each clock regions is made up of specific no. of CLBs, BRAM and other Logic resource which are routed by the number of clock buffer and net. So, to cater the diverse need of clocking resources multi-region clock buffer, Horizontal clock buffer, Global clock buffer and regional clock buffers are also provided. These clock buffers are using dedicated physical interconnect resources so we can consider the clock interconnect network as Clock layer.

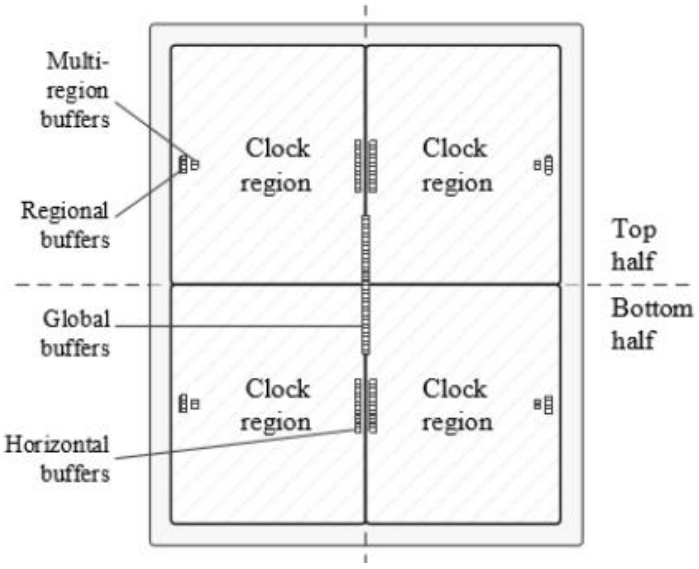

Figure 13: Clock Buffer Layer in Clock distribution region

2.2 Reconfiguration

Reconfiguration of FPGA is arguably “The Selling point”. The flexibility provided by FPGA is what attracts the vast number of users towards it. Reconfiguration has enabled FPGA to gain a massive acceptance among big data analytics, parallel and distributed computational systems. FPGA now-a-days hold multi-tenants in cloud computing as their general-purpose computation machine. PR is one of the leading name in the cloud computing. PR allows user to time multiplex FPGA resources enabling effective use of chip logic density. The reconfiguration time has thus reduced as the partial bit stream size is fairly smaller than the whole bit stream [22]. PR allows designed to be portioned in Static and Dynamic parts.

The Partial Reconfiguration in FPGAs was initially introduced by Xilinx in 1995. Xilinx secured a patent in which an FPGA was able to store multiple configuration at a time. In the initial designs there were two configuration memory arrays in the FPGA device in order to store different configuration data. In this architecture the configuration data was chosen from either of the configuration memory on every alternative half of the clock cycle through a switch. FPGA would output the result on end of every other clock cycle. In 1997 the idea of partial reconfiguration from Xilinx was further extended. This time Xilinx introduced a time multiplexed FPGA in XC4000E series. In this concept user was able to time multiplex the combinational logic but out put of each stage needed to be stored as well. For this purpose, micro registers were used to store the previous stage output of LUTs and flip-flops. XC4000E devices were able to store eight configurations and to switch between the configurations it only took 1 clock cycle i.e. as low as 5ns reconfiguration time. In order to update any of the eight configurations it needed to be in active while updating the configuration contents. This can be done with copying the contents from any external storage in to one the eight configuration spaces. In order to update the configuration space a special “RAM” mode is provided, hence allowing the ability of self-modifying hardware.

The major drawback of these devices was their power consumption. These devices consumed tens of watts because of the high switching activity. Another limitation was the full device granularity.

Xilinx has made progress through the years and the latest 7 series devices from Xilinx support partial reconfiguration where user is able to reprogram any frame of the device. Xilinx has introduced a new family of devices with hybrid architecture [9] which includes a core of ARM Cortex-A9 processor known as Processor System (PS) along with a 7 series FPGA device known as Programmable Logic (PL) on a single chip. This family of devices is known as Zynq SOC. Block level architecture of Zynq device is shown in Figure 10. The ARM processor in Zynq can communicate with the rest of the resources using Advanced eXtensible Interface (AXI). These Zynq devices have the similar architecture of programmable logic as of the other 7-series devices from Xilinx.

On the software side Xilinx initially introduced a difference-based technique for partial reconfiguration flow [2]. User was able to make smaller changes by using the FPGA Editor tool and making amendments on an already routed design. Partial bitstreams were generated by the implementation tool of Xilinx ISE. The generated partial bitstreams contained only the differences between the actual configuration and secondary partial bitstream. Xilinx later introduced PlanAhead [3] toll which supported modular design of partial reconfiguration. Figure 14 shows the steps involved in the procedure followed by the PlanAhead tool. Every PR design can have number of modules. All the modules can be written in hardware description language such as Verilog or VHDL.

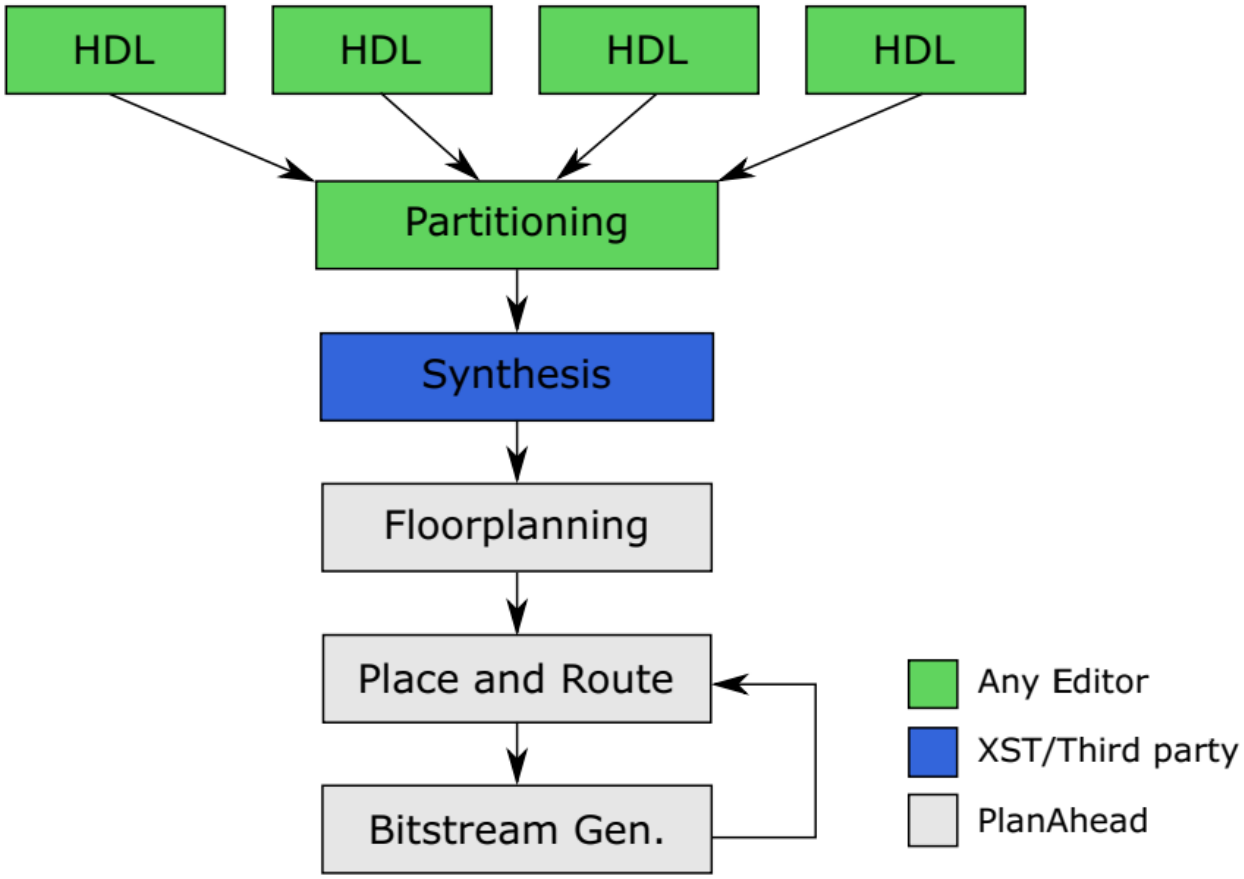

Figure 14: Partial Reconfiguration Flow using Xilinx PlanAhead

On the hardware side there are two main portions i.e. static and reconfigurable. There is only one static portion where as a device can have multiple reconfigurable regions. These partially reconfigurable [region can have resources like DSP slices, BRAMs and LUTs. They cannot have clock buffers and PLLs. They only reside in static part of the device. The static portion of device remains fix during operation and performs the pre-configured functionality. Reconfiguration management is also performed in the static part. One reconfigurable region can have multiple modules multiplexed into it.

The dynamic or partial reconfigurable module can be arranged on chip in various configurations (As shown in Figure 15).

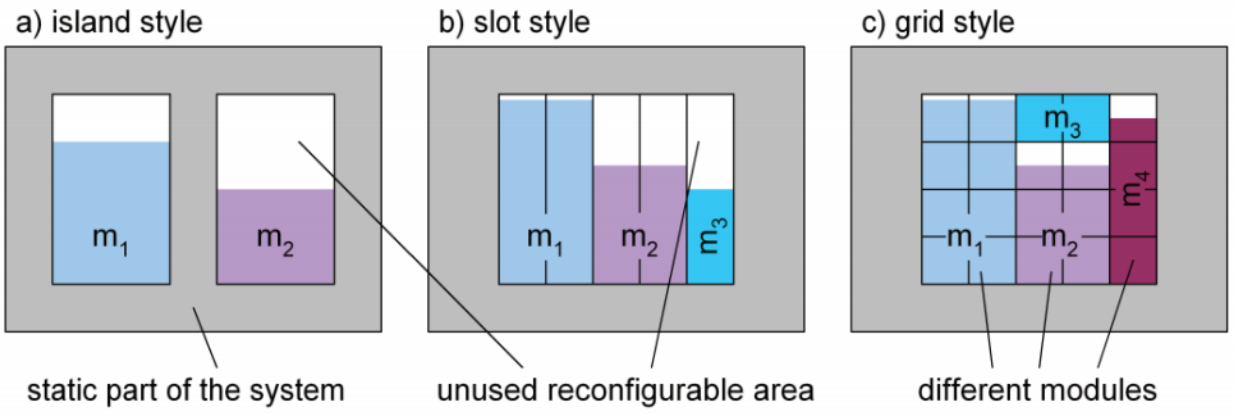

Figure 15: Reconfiguration styles

[23] Each style has its own pro's and con's. Island style is the simplest to implement but suffers from internal fragmentation resulting in high percentage of resource wastage. Slot style configuration does not have fragmentation problems where module can occupy resources as per its need. Tiling of RM region is a very complex task in which one has to keep in mind the placement of routes and their cross-over from static to dynamic regions. The optimal PR style to use depends on the user requirement and may vary or consist of a model that is hybrid of two or more styles.

2.3 Configuration detail

2.3.1 Configuration memory Architecture

Configuration memory has the same internal structure as the physical arrangement of FPGA resource in a Clock region column. As for one CLB Column, 36 frames are required which contain specific LUT, Flip-Flop and routing information. Besides that, most significant bit of the frame is associated with the upper resource of column and the least significant bits are for lower resources. As CLB Column require 36 frames for reconfiguration similarly 128 frames are required for BRAM and 28 frames are required for DSP slices. Depending on the amount and type of resources to be configured, number of frames have to be written to FPGA. For e.g. four frames are required to configure one LUT and 6 frames are for BRAM. Configuration related to regional clock wire that cross through the center of the region is mapped to 51st word of each frame. Each Configuration frame is protected through Error Correction code (ECC) which is also

written in 51st frame word. This Error correcting code is used to detect and correct error in single bit but not able to correct dual bit.

The smallest entity of configuration memory is the frame and each frame consist of 101 words in 7 series FPGA Architecture. In Zynq architecture whole device is covered with 71×149 CLB so the size of configuration file required to program whole device is 3.7 MB. That include 272 words for header and control information and rest of the information include configuration frame corresponding programmable logic.

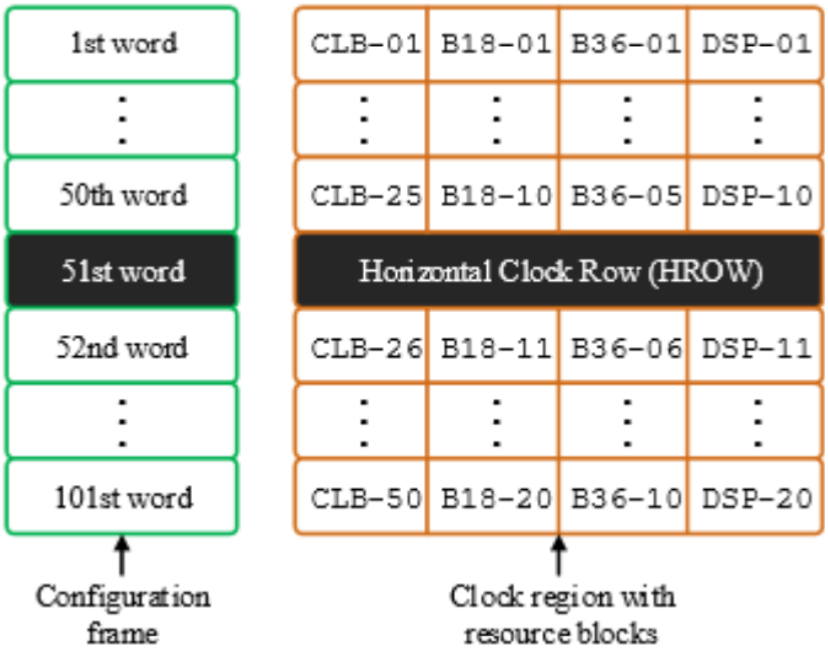

Figure 16: CLB Height of Different Resources

2.3.2 Configuration Registers

Configuration Registers are used in configuration process. Bit stream is composed of configuration commands and data sent to configuration register. Some configuration register has read/write capability while some have only read/write access. Each register write takes 2 word, one is for the command and the other is for register value. Some of the important configuration register and their functionality is described in

Table 4.

Name	R/W	Address	Description	Functionality
CMD	R/W	00100	Command register	Instruct configuration module to perform specific function, important CMD commands are WCFG (write configuration data), RCFG (read configuration data)
CTL0	R/W	00101	Control Register 0	Specify certain options for configuration, most important bits of CTL0 is the Global LUT mask (GLUTMASK). When enabled, the GLUTMASK bit causes any changeable memory cell read back values such as distributed RAM, SRLs, and DRP memories to be read back as all zeros or all ones.
FAR	R/W	00001	Frame address register	FAR contain FRAD of the frame. Any read/write in configuration memory is accessed by frame address register.
FDRO	R	00011	Frame data register output	Used to Read configuration data from configuration memory from address in FAR.
FDRI	W	00010	Frame data register input	Used to write data into configuration memory from location addressed in FAR.
Stat	R	00111	Status register	Used for debugging and status checking, useful in determining certain errors such as decryption, Device ID, and CRC check errors.
IDCODE	R/W	01100	Device ID register	Each unique FPGA has its own IDCODE, IDCODE is necessary in writing configuration memory
CRC	R/W	00000	CRC register	Used in the device-wide CRC check and control the behavior of the CRCERROR signal

Table 4: Configuration Registers

Frame Address register

Configuration frame is addresses by the frame address register (FAR). FAR is a 32bit word containing information of a frame that correspond to FPGA resource in which data is to be read/write. Frame address register is composed of Block type, HCLK, CLB Column address and minor frame address. Block type determine the type of logic resource like CLB, DSP, Block RAM etc., HCLK is the address of horizontal clock region row whereas Column address is the column of the CLB in clock region, and minor address is the address of the frame within configuration frames. Configuration memory is loaded with start frame address which is incremented depending on the no. of word to read/write. Following table represent no. of bits assigned to each Field.

Field	Bits assigned	Description
Block Type	[25:23]	Block types are; CLB, I/O, CLK, BRAM and CFG_CLB
Top/Bottom bit	[22]	Select between top-half (0) row and bottom half (1) row
Row Address	[21:17]	Select the current row, row address decrement from top to center and increment from center to bottom
Column Address	[16:7]	Column Address is the CLB column start from 0 and increment to right
Minor Address	[6:0]	Minor address the frame address among other frames in configuration memory

Table 5: Frame Address Register Details

2.3.3 Configuration commands

Configuration commands are sent to configuration port to access configuration memory. Synchronization command play a significant role to sync data being loaded to configuration port. After receiving synch word (0xAA559966) processing of subsequent data words begin and similarly after desynch command no new data is accepted by the configuration module. The Xilinx universal configuration word is the bus width sync word (0x000000BB) which align configuration module to 32bit data boundary. Following the dummy word synch word

(0xAA559966) is sent after which configuration module start accepting data. DESYNC command (0x000000DD), processed by CMD register releases the configuration module allowing low priority interfaces to access it. DESYNC command ignores any subsequent data coming to configuration port unless a new sync word is detected. Table 6 shows multiple configuration commands details.

Command	**Code**	**Description**	**Use**
NOOP	00000	No Operation	No Operation
WCFG	00001	Write Configuration Data	Used prior to writing configuration data to FDRI
RCFG	00100	Read Configuration data	Used prior to reading configuration data from FDRO
START	00101	Begin startup sequence	Startup sequence begin after successful CRC check and Desynch command
RCRC	00111	Reset CRC	Reset CRC register
DESYNC	01101	Reset DEALIGN Signal	Used at the end of configuration to desynchronize the device.
Synch	00010	Set DEALIGN Signal	Used to synchronize incoming configuration data

Table 6: Configuration commands

2.3.4 Bitstream Composition

Bitstream is comprised of a common pattern for each device of Xilinx FPGA. Bitstream is organized as; Header, Footer and configuration data.

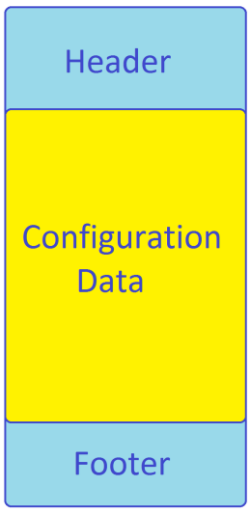

Table 7: Bit Stream Composition

Header

Header of the bitstream contain readable ASCII character which includes time stamp information and initial configuration command. These commands are series of 32bit data words that are written to configuration registers to read configuration memory. These commands include initialization the CRC check, setting the configuration and control option, setting the frame address register to starting address and issuing the write configuration command.

Footer

After writing configuration data, footer section is included which runs the command, wide CRC check, begin start up sequence and desynchronize the FPGA configuration sequence. Device with more logic resources have more frames and size of bitstream depends on the logic resources utilized in a design.

Configuration frame data

Configuration data is organized into frames which is to be written into configuration memory. This is done by the FDRO/FDRI register. Size of bitstream depends upon the no. of frames being loaded to configuration memory. Configuration data is further subdivided into Type 0 (CLB frame) and Type 1 (BRAM frame) Configuration frames. On-chip resources are configured by sending data to configuration registers. Configuration registers are accessed using command packets which has the minimum size of 64 bit, first 32 bits are for command and the next 32 bits are for data. As the width of configuration port is 32bit so a configuration register is accessed in

two clock cycle. Configuration command has two type of packet header, one is Type 1 header which define the type of register, register address and no. of words to read/write. The other is Type 2 header which is defined with Type 1 header if data words are greater than 2048. During write operation command packet are used with variable length frame words. So with FDRI register multiple of 101 frame words are defined which need to be written. Tables below define the composition of command register with Header of Type 1 and Type 2 also the Type 1 and Type 2 packet format composition and Opcode defined for different configuration commands

Field	**Header**		**Body**
Content of command register	Type1	Type2	Data words
Data width	32	32	Depending on frame data to be read/write
Description	Always used	Used when data words are greater than 2048	Only required when data words need to be written to configuration memory

Table 8: Packet Format of Command Register

Header Type	**Opcode**	**Register Address**	**Word count**
[31:29]	[28:27]	[26:13]	[10:0]
001	xx	xxxxxxxxxxxxxx	Xxxxxxxxxxx

Table 9: Type 1 Packet Format

Header Type	**Opcode**	**Word count**
[31:29]	[28:27]	[26:0]
010	xx	xxxxxxxxxxxxxxxxxxxxxxxxxxx

Table 10: Type 2 Packet Format

Function	Opcode
NOP	00
Read	01
Write	10
Reserved	11

Table 11: Opcode for Different Configuration Commands

2.3.5 Configuration Operation

Two main operation can be performed on configuration memory

a) Read back
b) Writing

Readback operation is performed by reading FDRO register from the location addressed in Frame address register. While writing is performed by writing to FDRI register followed by writing the CRC register which authenticate the valid data to be written in configuration memory.

Readback

Readback Operation is composed of set of command sequence which perform reading/writing to configuration registers and read configuration data from configuration memory. Configuration commands sequence are divided into three steps. First step is to write header which contain synch command, Reset CFG command, write Frame address register and set of NOOP commands. After that number of frame word to be read from configuration memory are defined and data is read through FDRO register. In readback one dummy frame is also read before the actual frames as if four actual frames are to be read 505 words are defined including the dummy frame. In third step Footer is written which contain START, reset CRC and Desynch command. Readback operation is further categorized as; Readback Verifier and readback Capture.

Read back verifier is a comparison which confirm that the configuration logic is programmed and results are modified according to user requirement. Read back capture is used to determine the content of user state elements, which by design change during operation. Readback command sequence for readback capture is the same as for readback verifier.

Writing

Writing command sequence is similar to readback command sequence except header include command to write IDCODE register which is necessary to specify device ID. FDRI register is used to write data into configuration memory and dummy frame is also written in the end to clear configuration register. To verify configuration data integrity pre-computed CRC checksum is written to CRC register which then compared to CRC value generated by the bitstream and then allow for reconfiguration. Other registers for device setup and status are also used like CTL0, STAT and COR0 registers.

2.4 Reconfiguration in Zynq-7000 Architecture

The Zynq-7000 APSOC consists of a dual-core ARM cortex-A9 based Processing system and Programmable logic in a single device. Standard communication infrastructure and integrated reconfigurable fabric is also coupled with powerful ARM cortex A-9 processor. AXI bus is used for communication between on-chip memory, memory controller and peripheral blocks. PS is attached with PL through AXI port offering high bandwidth between two components of architecture. The reconfigurable fabric of Zynq is based on 7-series FPGA Architecture which can be reconfigured using PCAP and ICAP.

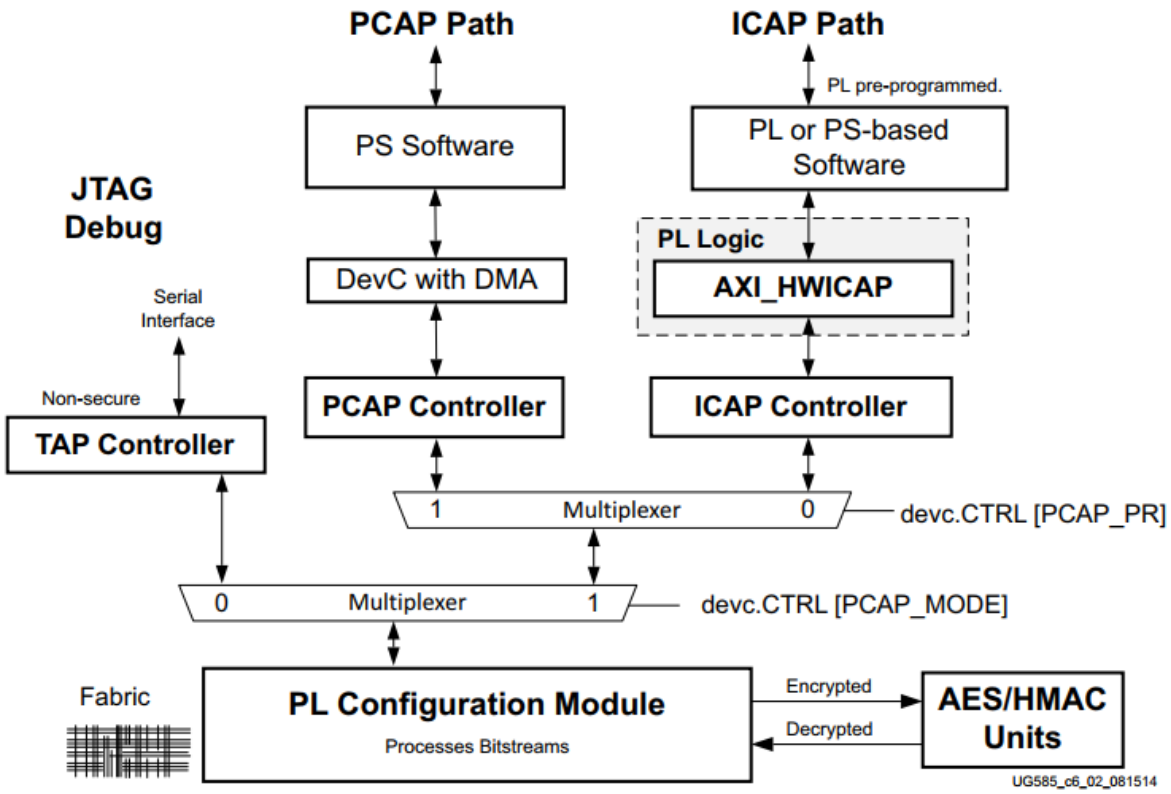

Figure 17: Zynq configuration modes

Generally, there are four ways to Configure PL region of a Zynq SOC from PS as shown in figure below

a) JTAG

b) ICAP

c) PCAP

d) Select-MAP

2.4.1 JTAG

The Joint-Test Action Group interface is among the most commonly used serial interface for programming and debugging of embedded processors, microcontrollers and FPGAs. It supports IEEE 1149.1 standard for boundary scan and Test Access Port architecture. JTAG was originally intended to debug and check serviceability of high pin packages of integrated circuits.

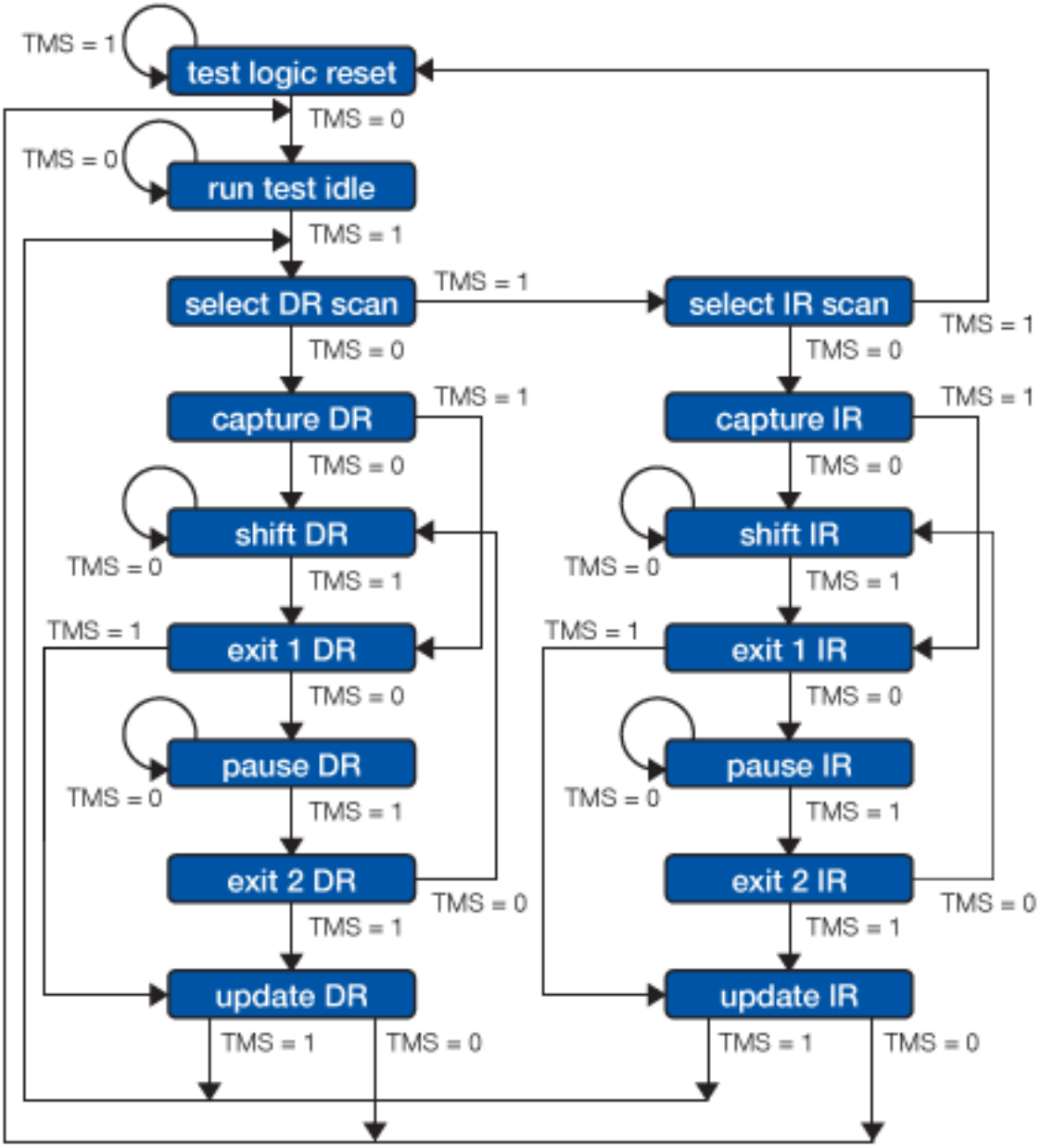

Figure 18: JTAG state machine

JTAG is also the highest priority configuration interface, and will always be able to access the configuration module regardless of the configuration mode pins. This interface can program multiple devices, but must do so in a serial chain, (not in parallel like Select-MAP). This interface is typically accessed from an external source, with clock rates up to 66 MHz [16]. The problem with this interface is that only some of its structure is defined in IEEE and rest of the protocol is vendor-specific and its detail are not available for general public usage. However,

there are commercial-off-the shelf solutions that claim to provide generic JTAG manipulation tool which anybody can use to inject fault in the system. One of the examples is JTAGulator [17].

2.4.2 ICAP

Another option for performing configuration operations is to program the device from within the FPGA itself. Internal configuration Access Port or ICAP can only be accessed from a user design via primitive instantiation. The ICAP allows a user design to reconfigure the device (either partially or fully), after the initial configuration has been completed. However, It is not able to perform the initial configuration.

Figure 19: ICAP primitive for 7-series FPGA devices

The ICAP is primarily used for partial reconfiguration. However, ICAP is an important consideration that can also help in development of tamper resistant systems [19]. ICAP can be used to issue IPROG command that can help reset the FPGA to its initial stage in an event of breach. IPROG is a specialized FPGA command that can be sent through the ICAP interface which results in clearing of the entire FPGA configuration memory such as contents of flip-flop, and key expansion memory. It should be noted here that IPROG commands does not the clear the decryption key itself as bit-stream decryption key is stored in BBRAM or E-Fuse. Its functionality is almost equivalent to the insertion/applying of the external PROGRAM_B pin. This effectively clears FPGA's configuration memory (flip-flop state, block-RAMs, and configuration data) and can be joined with the KEYCLEARB signal as a reaction to

tampering event [19]. However, it is to be noted here that to IPROG command can only be sent to the configuration engine, if the ICAP primitive is present in the user design and the proper order of commands are followed to write to it.
Like Select MAP, one of ICAP primary advantages is its configuration speed, an operating frequency of 100 MHz [20]. The ICAP is the interface of choice for several internal scrubbers in academia and industry.

2.4.3 PCAP

The Processor Configuration Access Port or PCAP is a unique interface that enables access from a hard processor to the configuration module. The PCAP is only found on the Zynq-7000 family. It is the bridge between the Zynq's dual-core ARM processing system and the FPGA configuration memory. One of its most important features is that it allows software programs running on the processors to access the configuration module at runtime via configuration commands written in software.
The PCAP clock can run at frequencies as high as 500 MHz, though it usually runs at no higher than 100 MHz for most applications. PCAP clock is also used to clock the bit stream data-path to the PL configuration module. This clock is basically a divided clock; typically, PCAP_2x clock. (The supportable clock frequency range for the PCAP clock is can be seen from Zynq TRM [21]). Hence if the user wishes to get a 100 MHz PCAP clock in is design, he must set the PCAP_2X clock bit to 200 megahertz.
PCAP also supports configuration in non-secure mode, which was used in our research as security of the design was not our primary concern. It is worth mentioning here that data transfer rate using PCAP is limited; which is roughly 145 MB/s.

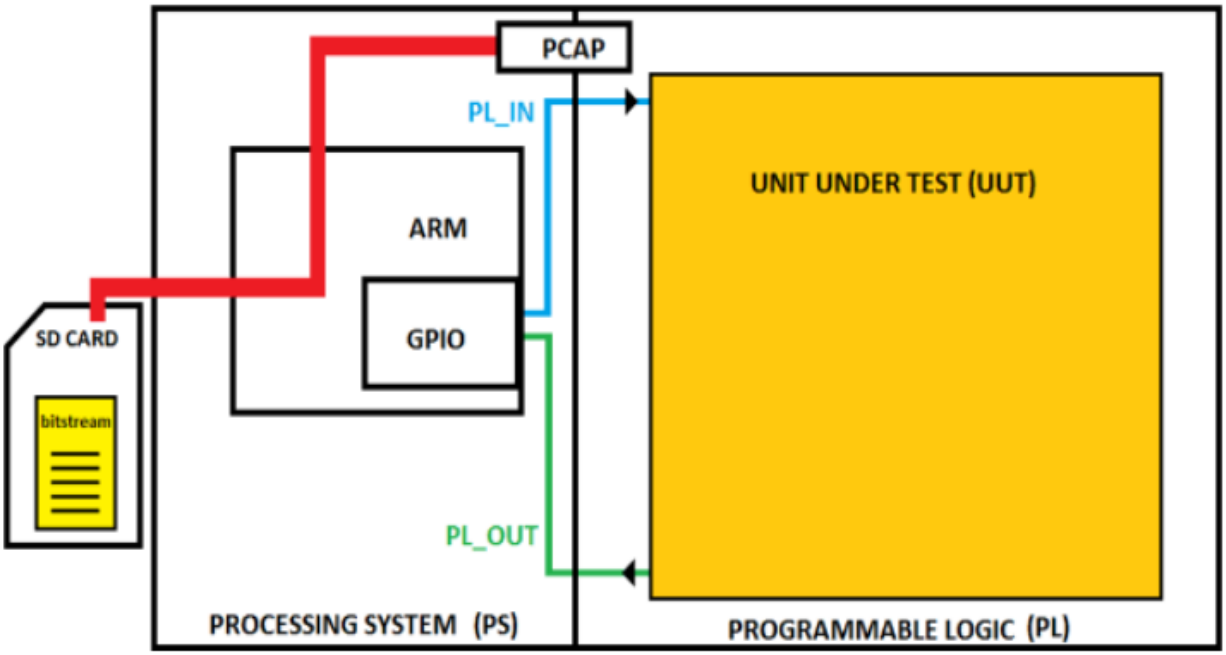

Figure 20: An example of PCAP usage

We know that the PL configuration module although, can handle data rate of 32 bits per PCAP clock cycle, but the overall throughput and performance of PCAP gets limited due to PS AXI interconnect. This approximation is noted from Zynq TRM and assumes a 100 MHz PCAP clock frequency, a 133 MHz APB bus clock frequency, a read issuing capability of four on the PS AXI interconnect, and a DMA burst length of eight [21].

2.4.4 Select-MAP

Select-MAP is a parallel, high-bandwidth interface with a bi-directional data bus supporting data widths of 8, 16, or 32 bits [18]. SMAP interface provides the ability to configure multiple FPGAs in parallel and can be used with high-speed clock rates as much as up to 100 MHz. The main disadvantage of this interface is that a number of I/O pins (equal to the data bus length) must be reserved during configuration by Select-MAP, and thus are temporarily not available to the user design.

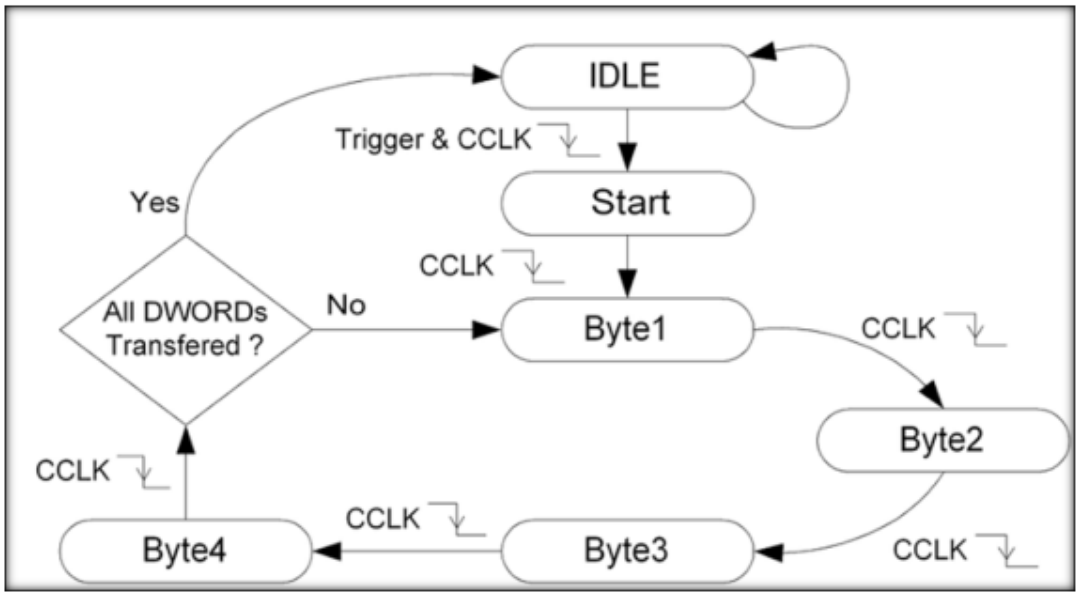

Figure 21: Select-Map Interface FSM

Select-MAP is an FPGA configuration mode that allows user to program as much as up to 3 Xilinx devices in parallel all the while providing simultaneous reading and writing capability through byte-wide ports. All of these devices are however programmed one at a time which is realized through assertion of the correct CS pin at specific time intervals. An external clock source which can be anything such as a download cable, a microprocessor or any other FPGA, is required for successful programming. The data is loaded one byte per CLK pulse.

This interface was not explored as an option in this research as this mode is supported by obsolete devices such as Spartan and Virtex families where it is typically used as a primary mode of configuration especially when configuration time is a significant concern or aspect. Because the configuration module is the only gateway into the configuration memory, only one of the interfaces may actively process commands through it at any given time. A multiplexer function decides which interface controls the configuration module, so that that interface has exclusive control. The details of this multiplexer function is not publicly documented. A model based on experimental observations and the documentation is shown in Figure 22, but it does not necessarily represent the actual implementation

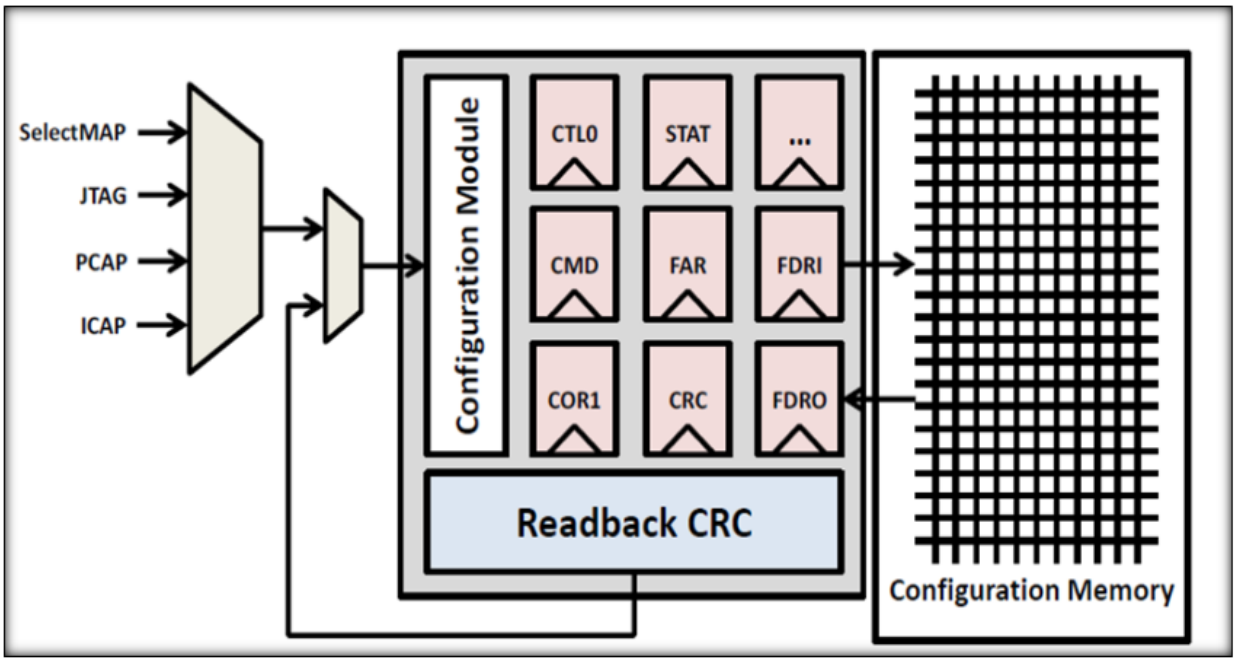

Figure 22: Selection of configuration mode selection in a zynq device

CHAPTER 4: METHODOLOGY

In this chapter we will discuss the details of proposed framework of partial reconfiguration. As the objective of this research indicates that we are required to utilize the feature of partial reconfiguration in FPGAs in order to implement the hardware hungry algorithms with minimum resources. In order to do so we will need to break down the algorithm in to multiple modules. The break down needs to be organized in such a manner that we can divide the resource utilization as uniform as possible. After breaking down the main algorithm in multiple modules, we will define a flow of algorithm in which inter-module dependencies are defined.

After defining module dependencies, we will group the modules which can be fixed into similar partially reconfigurable regions. A static portion will be designed which will organize and initiate all the reconfiguration related task. It will also be the responsibility of static part to latch the output of one module which may be required by the next module which is to be configured. When breaking down an algorithm into multiple modules it is to be kept in mind that there are three types of resources which are to be considered i.e. DSP slices, Slice LUTs and Slice registers. Their priority for modular break down can vary for different types of algorithms. For example, if an algorithm requires huge number of computational resources and has a lot of multiplication operations in it than it will consume more DSP slices. In this case we will try to keep the uniform division of DSP slices in modules. In order to explain the steps in detail we will suppose an algorithm with following resource requirements.

a) Slice LUTs: 60000
b) Slice Registers: 40000
c) DSP Slices: 150
d) BRAM: 20 RAMB36

Let's assume that the complete implementation of above-mentioned problem is divided in to five main modules. 1st module is the input buffer module which will receive the data using serial protocol and fills the input fifo. 2nd, 3rd and 4th modules perform some mathematical calculation on the input data in such a way that output of 2nd module is input of 3rd module and 4th module

takes the output of 3rd modules as input. The 5th module than assembles the output data of 4th module and sends data out serially. Figure 23 shows the flow of the implementation

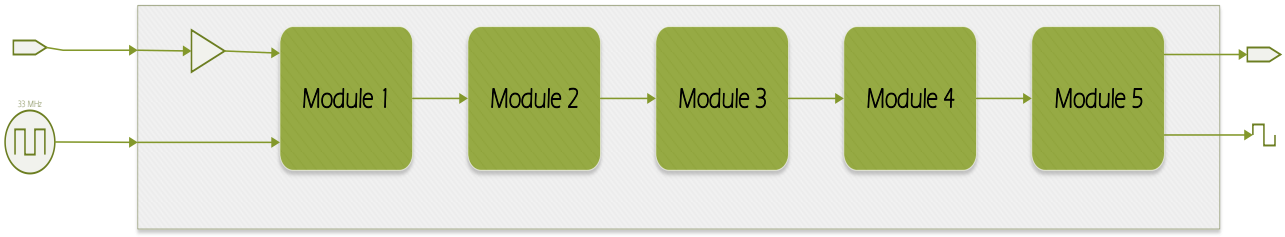

Figure 23: Flow diagram of implementation without PR

In this scenario all execution is taking place serially among the modules and each module is dependent on the previous module. In order to apply the partial reconfiguration, we need the individual resource utilization of modules which is provided in table below

Modules	Slice LUTs	Slice Registers	DSP Slices
Top	1150	2000	0
Module 1	2200	3000	0
Module 2	10550	10000	100
Module 3	11000	17000	30
Module 4	32800	5000	10
Module 5	2300	3000	0

Table 12: Resource utilization of example problem

The top module also uses 2 BRAMs for intermodular data buffering. Now in this case the 3 main modules are in the center which are performing the calculation on the data. The top module along with input buffering module and output module will be the part of static region as these

modules use the minimum of the resources. The resource utilization shows that module 2, 3 and 4 are performing the main tasks thus utilizing most of the resources but the resource utilization among these modules is not uniform, module 2 is utilizing the most of the DSP slices, module 3 is consuming the major part of slice registers and module 3 three is leading in utilization of Slice LUTs. Now in order to reduce the overall resource utilization via partial reconfiguration we will fit the three most resource hungry modules into one partially reconfigurable module. The PR module will be able to accommodate any of the three modules. PR module's resource utilization will be considered max of all three modules for each resource i.e.

a) Slice LUTs: **Max (**M2 Slice LUTs, M3 Slice LUTs, M4 Slice LUTs**)**

   =32800

b) Slice Registers: **Max (**M2 Slice Registers, M3 Slice Registers, M4 Slice Registers**)**

   =17000

c) DSP Slices: **Max (**M2 DSP Slices, M3 DSP Slices, M4 DSP Slices**)**

   =100

Therefore, the resource allocation of partially reconfigurable region will be as follow

Partially Reconfigurable Region Resources		
Slice LUTs	**Slice Registers**	**DSP Slices**
32800	17000	100

Table 13: Partially Reconfigurable Region Resources

It is also to be made sure that number of inputs and outputs of all the modules of one partially reconfigurable region should be same. If the number of inputs or outputs vary for different modules then maximum number of inputs and outputs should be considered for the PRR.

The static region will accommodate the PR controller which will decide when to reconfigure the next module in the PR region. According to the new flow of implementation the input module (M1) residing in the static part, will capture the data and store it in the BRAM of static region. On startup the PR region will be configured by the module 1's partial bitstream. Once the data

has been captured by module 1, it will send a start signal to module 2. After completing its operation, the module 2 will refill the BRAM with new data and indicate PR controller when it's done. PR controller will reconfigure the PR region with M3's partial bitstream, on successful reconfiguration module 3 will receive the data from BRAM and perform the desired operation and indicate the PR controller on completion of its task. PR controller will reconfigure the PR region with M4's partial bitstream thus shifting the control of data and address lines of BRAM to the M4. On completion of M4's task PR controller will start the module 5 which can now get data from the BRAM and output it serially. Figure 24 shows the flow of operation with PR

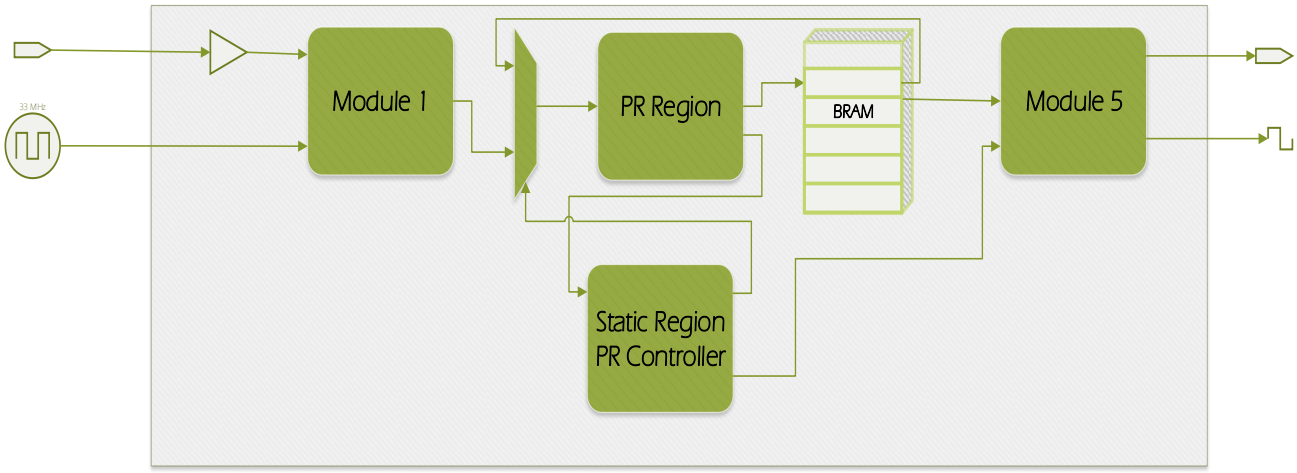

Figure 24: Flow diagram of implementation with PR

The overall resource utilization of the any problem can be reduced using PR framework. Percentage of reduction may vary for different cases. The new resource utilization of our example problem is shown in Table 14

Modules	**Slice LUTs**	**Slice Registers**	**DSP Slices**
Top	1150	2000	0
Module 1	2200	3000	0
PR Module	32800	17000	100

Module 5	2300	3000	0
Total	**38450**	**25000**	**100**

Table 14: Resource utilization of example problem after PR

3.1 Software Flow

In Xilinx Zynq and 7-series FPGAs onwards, PR is supported through the Vivado Design Suite [4]. This partial reconfiguration flow is not much different from PlanAhead, but it does not support all the functionalities via GUI, therefore user has to use TCL commands in some cases. Vivado has the same floorplanning limitations as for PlanAhead but the Xilinx 7-series FPGAs has some extra limitations that partition boundaries are not allowed to intersect interconnect tiles. These tiles are distinct resources that are responsible for managing routing between different resource columns. Vivado makes use of these interconnect tiles for anchor logic rather then using LUT based bus macros [5]. Using interconnect tiles actually improves the effectiveness of routing thus enhancing overall performance.

This portion of document describes the procedure of creating of partial bitstreams of partially reconfigurable modules. The first thing is to divide the problem into module and decide the number of partially reconfigurable regions required, also decide how many modules will fit in each PR region. Once the number of PR regions and modules have been decided then each PR region is assigned the associating modules using the partial reconfiguration wizard in Vivado. It is to be noted that partial reconfiguration is to be manually enabled for each association module. All the modules need to be synthesized separately and produce the matching netlist. After synthesizing we need to assign rectangular areas via floorplanning for each reconfigurable region. One P-block is drawn for each PR region. P-block can be manually reshaped according to the resource requirement. P-block needs to rectangular and aligned with the boundaries of clock region. Figure 25 shows P-blocks of different sizes in a zynq device which can be configured independently.

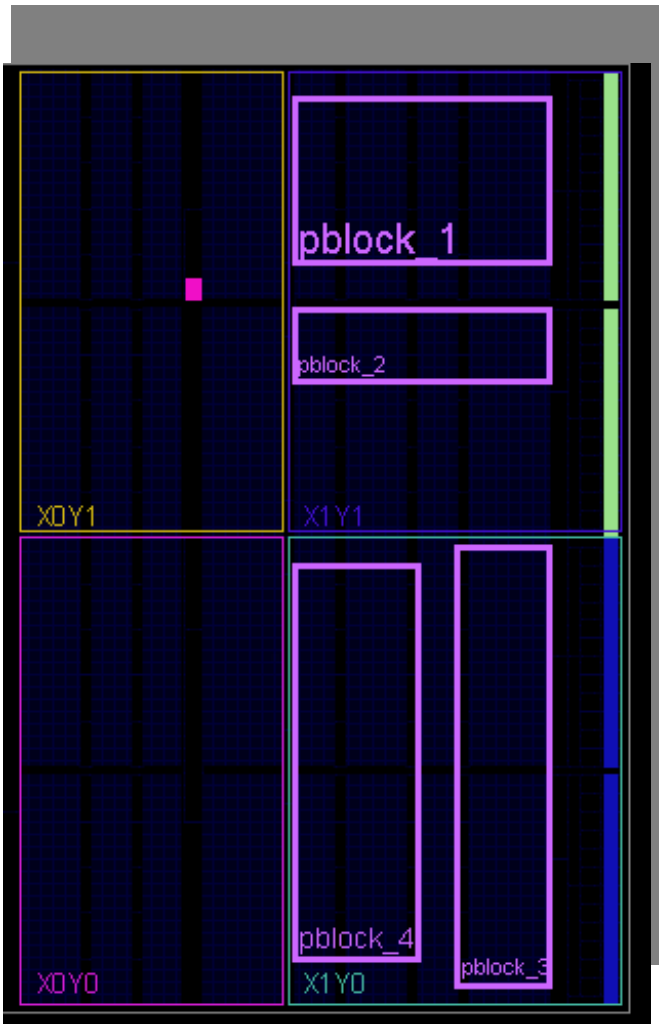

Figure 25: Multiple Reconfigurable modules along with a static region

User constraint file (.ucf) or Xilinx design constraint file (.xdc) contains the details of each P-block drawn in the device i.e. its width height and starting address in the device. Basic knowledge of the low-level architecture of FPGA fabric is required in order to generate a correct combination of modules for a partially reconfigurable region. Each PR region with its combination of modules makes a configuration. Multiple configurations can be generated with different partial bitstreams loaded in the PR regions on startup. Static part of the system is only implemented once in the design phase thus the routing and placement of the static region stays same for all the configurations generated. It is to be noted that routing resources in the PR region can be utilized by static region logic implementation but LUTs and Flip-Flops of PR region cannot be used by the static part logic. Partial reconfiguration controller in the static part can switch/ reconfigure the PR regions using configuration ports like PCAP or ICAP. It can only

reconfigure on PR region at a time and only one port can be used at time. If PCAP is active then ICAP cannot be used to perform reconfiguration operations.

In order to isolate inputs and outputs of PR-region from static region signals, Xilinx has provided a Partial Reconfiguration Decoupler IP. It can be configured with different number of signals with different bus width. PR Decoupler IP is controlled by the logic implemented in static region. When the static region is about to initiate the reconfiguration of a PR region it should decouple the in outs of PR 9region otherwise the PR module can end up in unknown state because of external interference during reconfiguration. Figure 26 shows Partial Reconfiguration Decoupler IP block with 3 single bit signals.

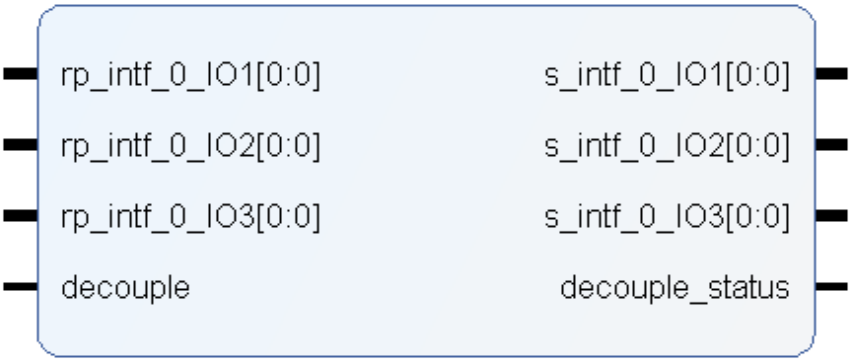

Figure 26: Partial Reconfiguration Decoupler IP block with 3 single bit signals

Once floorplanning is completed the Vivado tool then proceeds to generate a complete configuration file along with the partial bit files for each configuration. A different folder is created for all the configuration files and all of the folders contains a complete bit stream along with multiple partial bit streams. The complete bit stream of each configuration contains the similar static region logic and different startup configuration within the full bit stream for every corresponding configuration folder. Once the full bit file is loaded on to the device, any partial bitstream can be loaded into its corresponding region. A single partially reconfigurable region can have multiple modules and a partial bitstream for each one of them. For example, in our example problem we had three PR modules in a sing PR region as shown in Figure 27.

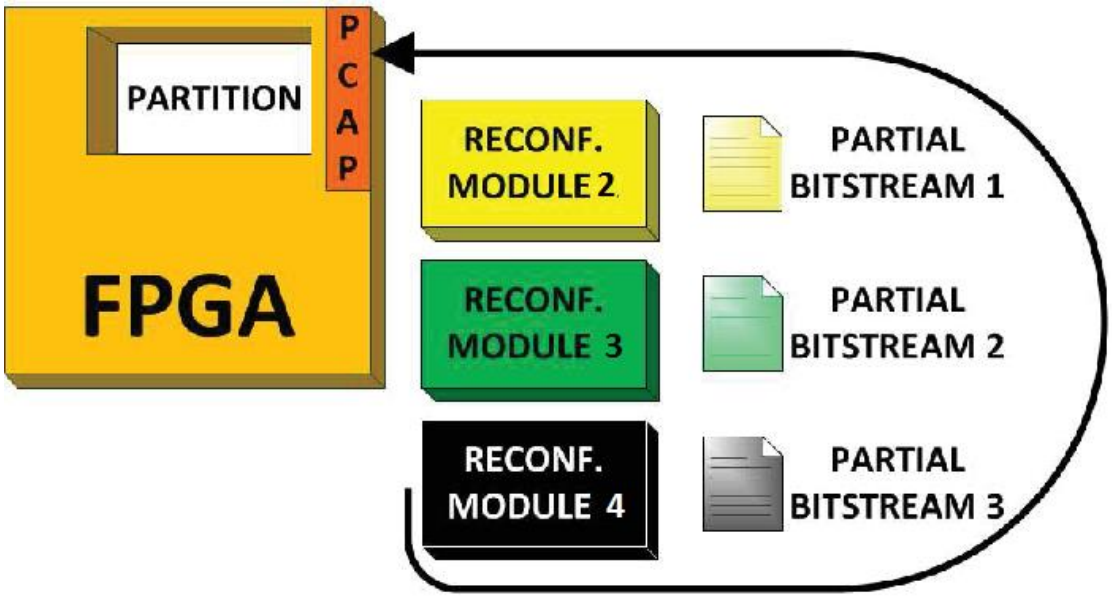

Figure 27: Multiple bitstreams for single PR region

Once the complete and partial bitstreams have been generated these files can be loaded on to the device via external source like JTAG or they can be copied into a non-volatile onboard memory linked with zynq device and can be configured via PCAP. The ZedBoard has an option of SD card which can be read through the PS. Once the files have been copied to the SD card then they can be read from the PS using the file name. data read from the SD card is not directly loaded on the device, first the bitstream is copied to the DDR memory so that while reconfiguring the bitstream is readily available and no delay is caused. Once the bitstream is loaded into the DDR than it can be transferred to the PL configuration memory. Figure 28 shows the complete procedure of reconfiguration. It is to be noted that in case of complete reconfiguration the PL fabric requires to be reset prior to reconfiguration, in case of partial bitstream reset is not required. Partial bitstream cannot be loaded until the full configuration file has been loaded first.

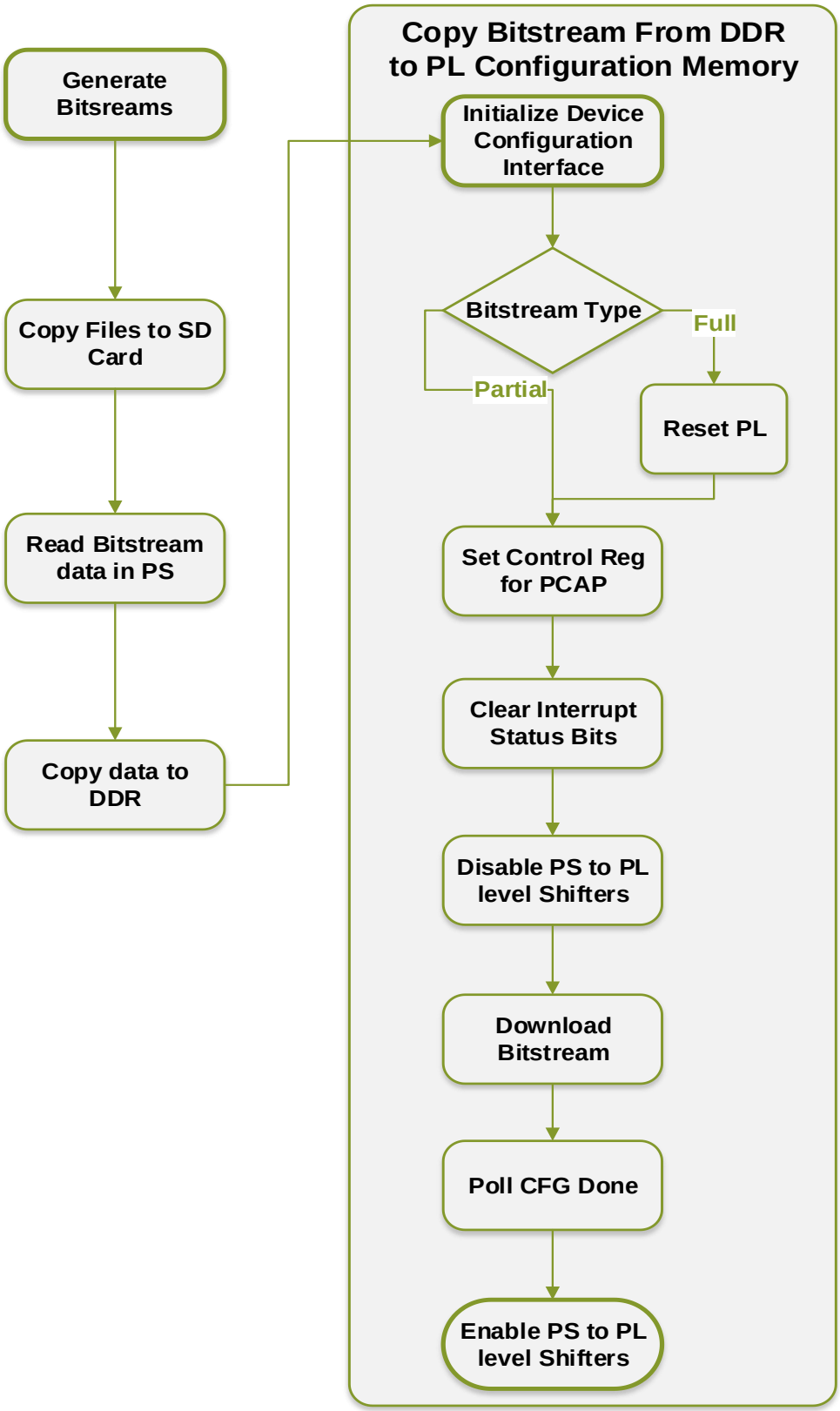

Figure 28: PCAP Configuration Process flow chart

CHAPTER 5: JPEG COMPRESSION IMPLEMENTATION USING PARTIAL RECONFIGURATION FRAMEWORK

In this portion we will discuss the implementation of JPEG compression which requires a lot of hardware resource in order to facilitate all its computations implemented concurrently on FPGA device. Since our scope here is to optimize the resource utilization of FPGA, not the quality of compression so in this example is using a lossy JPEG compression method and to keep the implementation simple we will take input image of size 96 x 96 pixels. So that we can process the image in chunks of 8x8 pixels. In order to process an image of different dimensions padding is required. Figure 30 shows the modular flow diagram of JPEG implementation without partial reconfiguration on Xilinx zynq device.

4.1 Interface

The top-level module encapsulates two modules, one for receiving and processing the data and second is the ff checker. The main input signals of top module are enable, reset, clock along with 24 bit wide data and control lines. 8 MSBs of data i.e. data_in[23:16] represents the green value, data_in[15:8] corresponds to red values and 8 LSBs represents the blue values. The whole algorithm runs on a single clock, and all of the registers are synchronized to the rising edge of this clock. The enable signal should be brought high when the data from the first pixel of the image is ready. The enable signal needs to stay high while the data is being input to the core. Since the data is 24bit wide therefore in order one 8x8 block of an RGB image will require 64 cycles. After the 64 pixels of data from each block has been input, the enable signal needs to stay high for a minimum of 33 clock cycles. There should not be any new data during this delay of 33 or more clock cycles. Enable signal is required to be de-asserted for one clock cycle in between the multiple 8x8 blocks input. This pattern needs to continue for each of the 8x8 blocks of data.

When the last 8x8 block of the is being input to the module, the end_of_file_signal should go high with first valid 24bits of the final 8x8 bock. The output bitstream is a 32-bit bus, and normally between blocks, any bits that don’t fill the whole 32-bit width output bus will not be

output. Instead, they will become part of the next 8x8 block of the image. On the last 8x8 block, the core will output any extra bits so that there are not any missing bits from the image.

Figure 29: JPEG compression Top Module In outs

The JPEG bitstream is output on the signal JPEG_bitstream, a 32-bit bus. The first 8 bits will be in positions [31:24], the next 8 bits are in [23:16], and so on. Data ready indicates the valid data coming out of JPEG_bitstream. 32-bit data out of JPEG_bitstream will have 1 cycle of data ready high. If the last bits do not fill the 32-bit bus in the last block of data, the signal eof_data_partial_ready will go high for one clock cycle aligned with extra bits in the JPEG_bitstream. The 5bit wide end_of_file_bitstream_count signal indicates the number of extra bits asserted.

4.2 Operation

4.2.1 Color Space Transformation

The functionality of JPEG compression involves several major steps, first of all the input data is received in RGB values which is required to be converted into Y_Cb_Cr (Luminance and Chrominance) values. This operation is implemented in the RGB2YCBCR module based on the formulas below:

$$Y = .587 * Green + .299 * Red + .114 * Blue$$
$$Cb = -.3313 * Green + -.1687 * Red + .5 * Blue + 128$$
$$Cr = -.4187 * Green + .5 * Red + -.0813 * Blue + 128$$

In order to perform these calculations, fixed point multiplications are used. All of the constant values in the above 3x3 matrix are multiplied by 2^14 (16384). The color space conversion

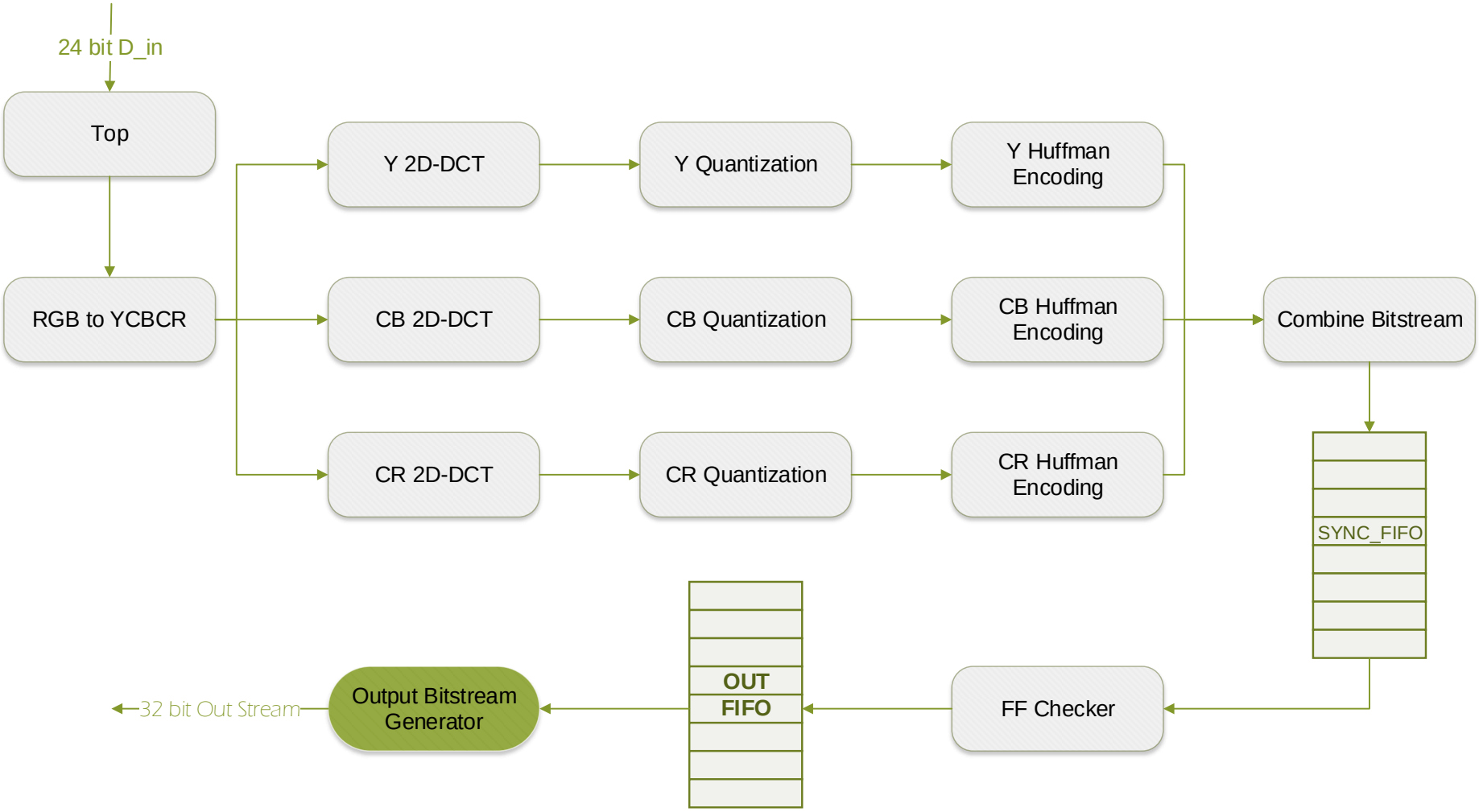

Figure 30: Modular Flow chart of JPEG Compression Implementation

multiplications are performed on one clock cycle, then we take sum of all the products on the next clock cycle. This is done to achieve a fast clock frequency during synthesis. Then the sums are divided by 2^14, which is implemented by discarding the 14 LSBs of the sum values, instead of actually performing a divide operation. Rounding is performed by looking at the 13^{th} LSB and adding 1 to the sum if the 13^{th} LSB is 1.

4.2.2 Discrete Cosine Transform

The next step after calculating the Y, Cb, Cr values is to compute the Discrete Cosine Transform (DCT). This is commonly referred to as a 2D DCT. The actual formula is the following:

$$DY = T * Y * inv(T)$$

T is the DCT matrix. Y represents the 8x8 block of the image which is being processed. The resultant matrix of the 2D DCT computation is denoted by DY. The DCT needs to be performed separately on the Y, Cb, and Cr values for each block. The DCT of the Y values is performed in the y_dct module. The DCT of the Cb and Cr values occurs in the cb_dct and cr_dct modules. To perform the DCT, the values of Y, Cb, and Cr need to be centered around 0 and, in the range, –128 to 127. The DCT matrix is multiplied by the constant value 16384 or 2^14. The rows of the T matrix are orthonormal (the entries in each row add up to 0), except for the first row. Multiplication of the T rows by the Y columns of data is performed, and the extra 128 in each of the Y values is cancelled out by the orthonormal T rows. After multiplying the T matrix by the Y matrix, the resulting matrix is multiplied by the inverse of the T matrix. This operation is performed in the code with the goal of achieving the highest possible clock frequency for the design.

4.2.3 Quantization

After the 2D-DCT module, quantization is performed on the Y, Cb, and Cr values independently. The 64 matrix entries calculated after performing the 2D DCT are inputs to this quantization module. This module quantizes the entire 8x8 block of values. The outputs from this module are the quantized values for one 8x8 block.

4.2.4 Huffman Encoding

After quantization is performed on 8x8 blocks of Y, Cb and Cr values, the results from quantizer are passed to the Huffman encoding module. Transposed output matrix of the quantizer are passed to the Huffman module. Transpose is taken so that the values in the 8x8 block can be written in left to right order because the computation in the DCT module generates the transposed results because of matrix multiplication. The Huffman table values can not be altered, during the execution. These values can be changed in the code and bit files are required to be generated again after changing the values.

A full Huffman table is generated, even if encoding a small image file and do not expect to use all of the Huffman codes. The calculations in this core may differ slightly from how you do your calculations, and if you use a Huffman table without all of the possible values defined, the core may need a Huffman code that is not stored in the RAM, and the result will be an incorrect bitstream output. The DC component is calculated first, then the AC components are calculated in zigzag order. The output from the huffman module is a 32-bits signal containing the Huffman codes and amplitudes for the either of Y, Cb, or Cr values.

4.2.5 Creating the Output JPEG Bitstream

After performing the above calculation, the output Y, Cb and Cr values are combined in an order that contains the huffman output of 8x8 block of Y, then huffman output for 8x8 block of Cb followed by the huffman results of corresponding Cr 8x8 block. After that huffman result of next 8x8 Y block is attached. After combining the values, FF check operation is performed on the output bitstream. When an FF is found on byte boundaries, an additional 00 is placed after the FF.

4.3 Implementation using Dynamic Partial Reconfiguration

In the JPEG compression implementation, there are 3 major computation modules (Discrete Cosine Transform, Quantization and Huffman Encoding) which consumes most of the hardware resource. In order to fit them in the partial reconfiguration framework we have multiple options. For example, we can create 3 reconfigurable regions, one for each color space so that each of

those reconfigurable regions will have enough resource to house all three modules (Discrete Cosine Transform, Quantization and Huffman Encoding). Figure 31 shows the block diagram for JPEG compression implementation based on the dynamic partial reconfiguration framework.

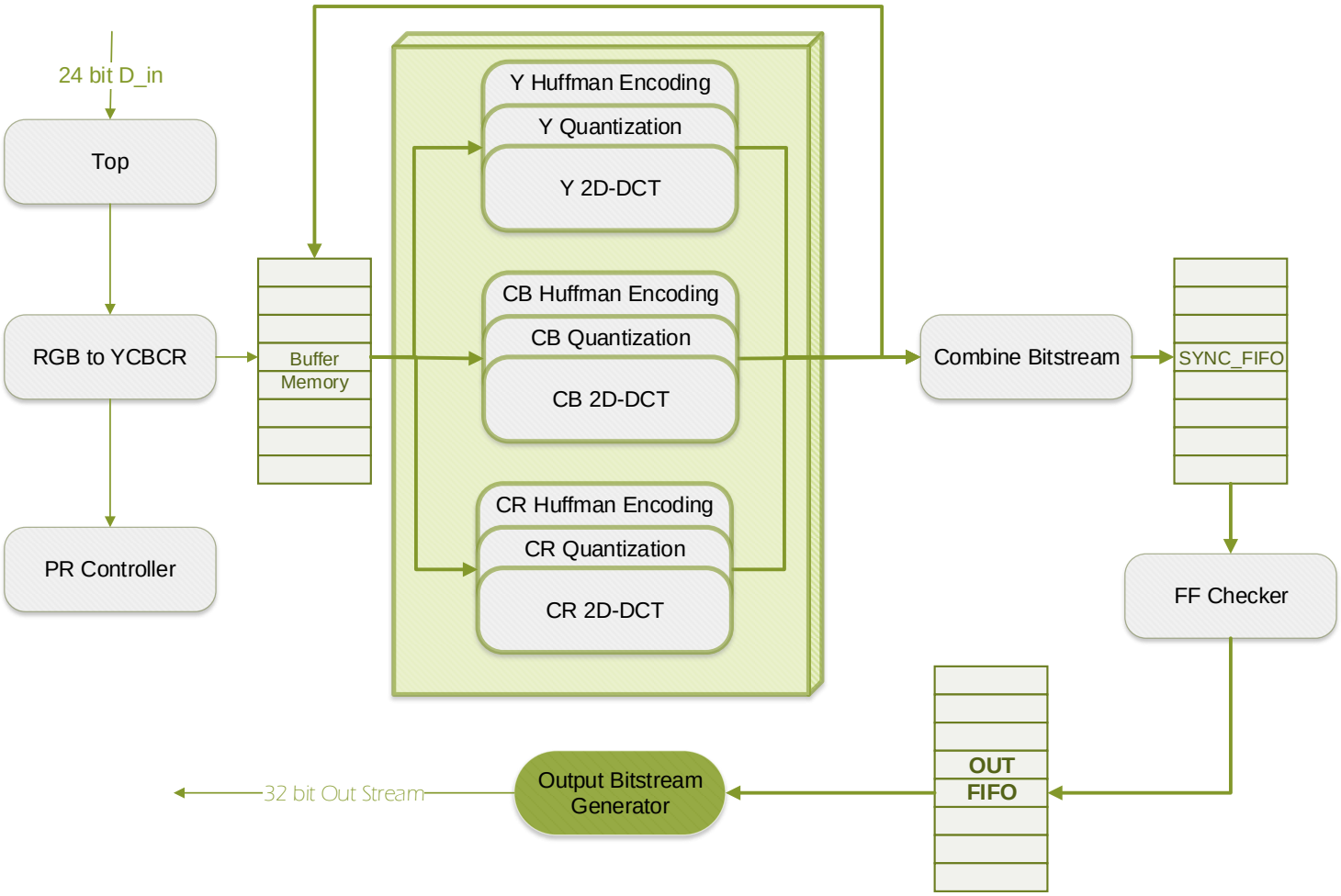

Figure 31: Block Diagram for PR based JPEG implementation

CHAPTER 6: RESULTS AND CONCLUSION

This chapter contains the detailed results of adopting the dynamic partial reconfiguration. We will discuss the results of the JPEG encoding in terms of picture quality and resource utilization. The Verilog implementation of the JPEG encoding is focused to achieve the maximum frequency from the core. This implementation has been tested via simulation. To keep the implementation simple, we will take input image of size 96 x 96 pixels. So that we can process the image in chunks of 8x8 pixels. In order to process an image of different dimensions padding is required. The values of red green and blue pixels is read from the image with 24 bits designated for each pixel value i.e. 8 bits for each red, green and blue values. These values are input to the core serially along with the control signals. 8 MSBs of 24 bit data i.e. data_in[23:16] represents the green value, data_in[15:8] corresponds to red values and 8 LSBs represents the blue values. The whole algorithm runs on a single clock, and all of the registers are synchronized to the rising edge of this clock. The enable signal is brought high when the data from the first pixel of the image is ready. The enable signal needs to stay high while the data is being input to the core. Since the data is 24bit wide therefore in order one 8x8 block of an RGB image will require 64 cycles. Simulation waveform of data and control signals is shown in Figure 32

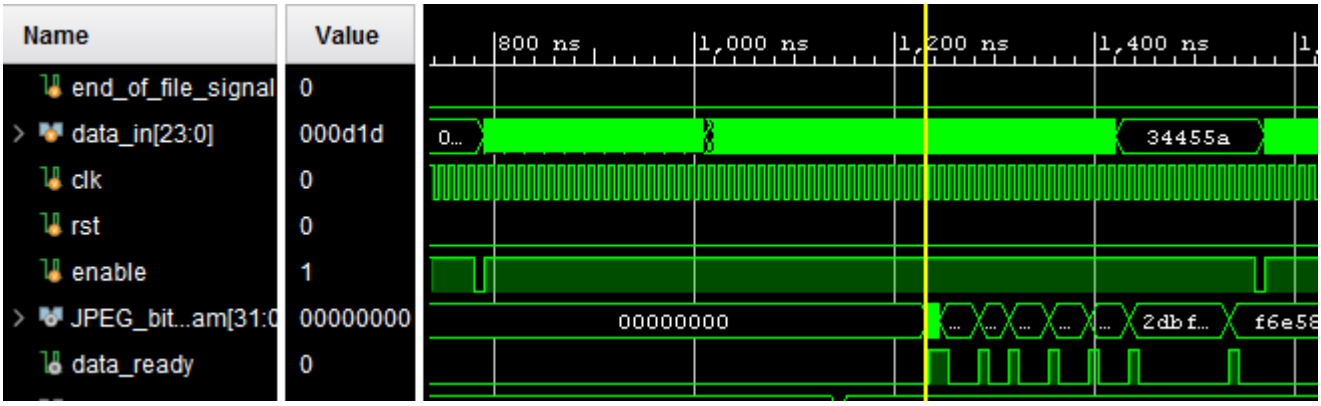

Figure 32: Simulation waveform of JPEG encoding

Output data is received from the 32-bit parallel interface and data ready signal indicating the valid output data. The output data is only the scan data portion of the image. The header of the

JPEG image is copied from another JPEG image of same dimensions. The Quantization and Huffman tables used in code are also similar to the sample image because of using the corresponding JPEG header. Data output from the core is combined with the header and checksum using a simple MATLAB script. Figure 33 shows the actual and compressed image side by side.

Figure 33: Input image and output image with file size

There is clear difference in the size of the image and visually the image does not look much distorted either which indicates that the code is functioning correctly.

5.1 Resource Utilization comparison

The complete implementation of above JPEG encoding has been synthesized for Xilinx Zynq 7000 AP SoC XC7Z020-CLG484 processor. Three main resources are responsible for covering the fabric area i.e. DSP slices, Slice LUTs and Slice Registers. Resource utilization shown in Table 15 are the resources utilized without applying any resource optimization technique. All three color spaces data is being executed in parallel in order to shorten the critical path of the system and achieve as much frequency as possible. The three main modules of implementation are discrete cosine transform, quantization and huffman encoding. All these modules executed in parallel for 3 different color spaces.

Table 15 separately shows the resources utilized by these modules, combined resources utilized for one color space and resource utilization for complete JPEG encoding core.

Parameter	DCT	Huffman	Quantization	Requirements for one color space	Complete Requirements
Slice LUTs	5483	1670	2	7198	47790
Slice Registers	6149	2334	2820	11308	34955
F7 Muxes	0	0	0	13	39
F8 Muxes	0	13	0	6	18
DSPs	60	6	0	66	180

Table 15: Resource utiliaztion of JPEG encoding on multiple levels

When the same implementation is optimized using the dynamic partial reconfiguration framework as shown in Figure 31 the resource utilization shows reduced usage as shown in Table 16.

Parameter	Complete Resource Utilization	Utilization After Partial Reconfiguration	Optimized Percentage usage
Slice LUTs	47790	31531	66%
Slice Registers	34955	21448	61.3%
DSPs	180	60	33.4%

Table 16: Resource utilization after partial reconfiguration

The results of dynamic partial reconfiguration-based implementation show significant reduction in hardware resource utilization as the Slice LUTs, Slice Registers and DSP are reduced by 34%, 39% and 66% respectively. These utilizations can further be reduced if the execution for all three

color spaces is organized serially through a single reconfigurable region but it will induce massive overhead of reconfiguration time.

5.2 Timing Comparison

Using the dynamic partial reconfiguration method to implement resource hungry problems adds the additional configuration time overhead in the execution of application. This time is usually in few milliseconds depending on the size of partial bitstream. In our case the total time of execution for the JPEG compression code is in micro seconds. Therefore, in order to save the hardware resource, it is mandatory to make a trade of with the execution time. Figure 34 shows the execution time for the JPEG compression at different levels. These values have been noted while the code was running at 100MHz frequency. However, the code is capable of running at higher frequencies.

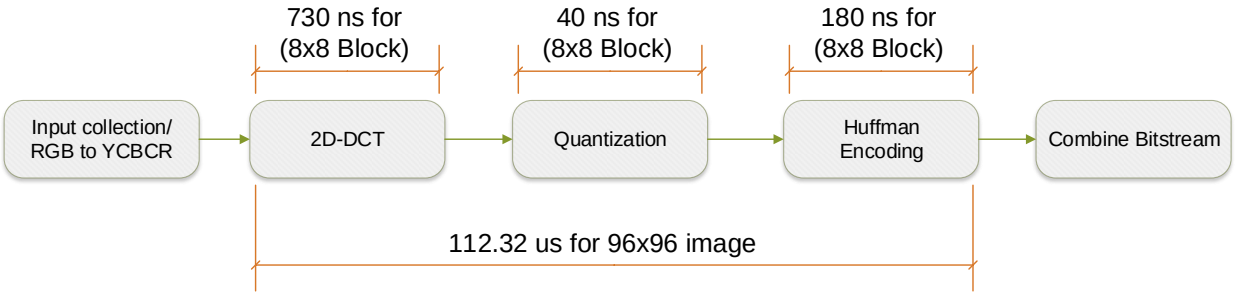

Figure 34: JPEG compression execution time without PR

Once the partial reconfiguration framework is used then we will have three set of partial bitstreams for along with a full bitstream for each reconfigurable region. Each color space will have a separate reconfigurable region. Initially the full bitstream fills the reconfigurable region with the partial bitstream corresponding to discrete cosine transform as it is the first step after converting RGB data to YCbCr. All the partial bitstream files are loaded in the DDR3 memory on startup in order to save the redundant reading time from the external storage (SD card). Figure 34 shows the execution time of JPEG core combined with the reconfiguration time at different stages.

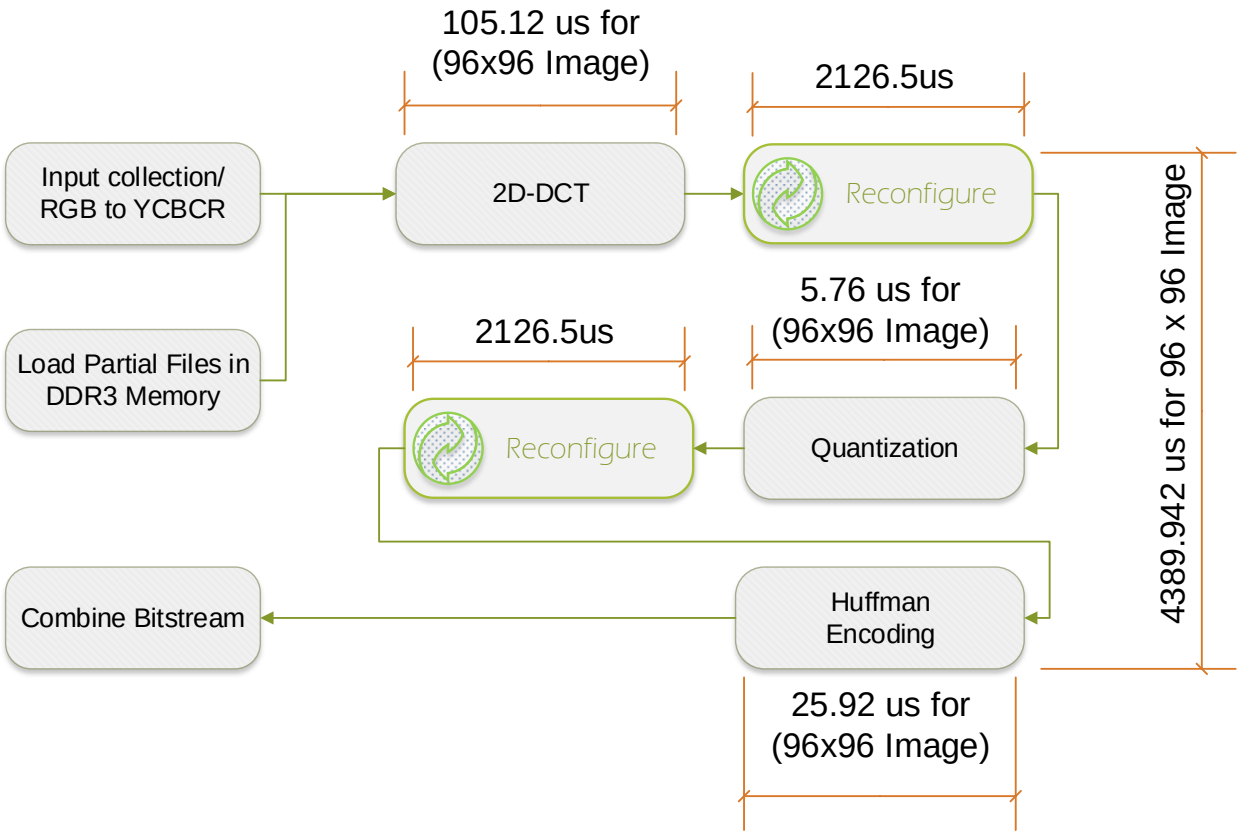

Figure 35: Execution time of JPEG core combined with the reconfiguration time

The bit file generated for DCT, Quantization and Huffman encoding are of same size because the reconfigure region is of fixed size. All partial files generated are of 267,948 bytes in size. The full bitstream carries 4,045,663 bytes. When running a standalone application on ZedBoard (without Linux running on the processor) the partial bit files take 2126.57us each for reconfiguration. These files are loaded in the DDR3 memory on startup which is running at 533 MHz (1066 MHz data rate). The DDR memory uses 32-bit wide data line and its theoretical throughput is:

4 Bytes x 1066 MHz = 4.264 GB/s

Reconfiguration occurs twice during the complete procedure of compression, 1st changing the reconfigurable logic from DCT to quantization and then from quantization to huffman. Rest of the logic resides in the static are of hardware.

CHAPTER 7: CONCLUSION AND FUTURE WORK

6.1 Conclusion

From the above results we can clearly see that the using the dynamic partial reconfiguration framework significantly reduces the hardware utilization. But it also comes at a cost of configuration time overhead which can be tolerated in certain applications. Table 17 shows comparison of implementing JPEG compression with and without dynamic partial reconfiguration framework in terms of resources and time

	Hardware Utilization	**Execution Time**
With DPR Framework	DSPs: 60 Slice LUTs: 31532 Slice Registers: 21448	4389.9 us
Without DPR Framework	DSPs: 180 Slice LUTs: 47790 Slice Registers: 34955	112.32 us

Table 17: Results comparison in terms of resources and time

Looking at the results we can safely conclude that partial reconfiguration framework can be used to reduce the hardware utilization in FPGA/SOC devices. However, it is recommended for those applications only where the execution time requirements are not very tight. Applications with strict timing requirements such as video streams are not suitable to be implemented with dynamic partial reconfiguration. It is more suited for the applications with complex algorithms and calculations. It is also useful for reconfiguration of systems that are deployed in remote locations.

6.2 Future Work

As it is mentioned above that partial reconfiguration of a reconfigurable region in device can be reconfigured by the PR controller which lies in the static portion of that same device and partial bitstreams are stored in a non-volatile storage attached to the FPGA/SOC. The static portion can also contain logic other than PR controller. We can make use of this feature and place a program that can read partial files from an external network. In this case there is no need for partial bitstreams to be available locally all the time and bit files can be transferred to the device remotely. To achieve this functionality, we will add ethernet controller in the static part of the device which can receive the bit files over the network and store them in DDR3 memory before initializing reconfiguration process.

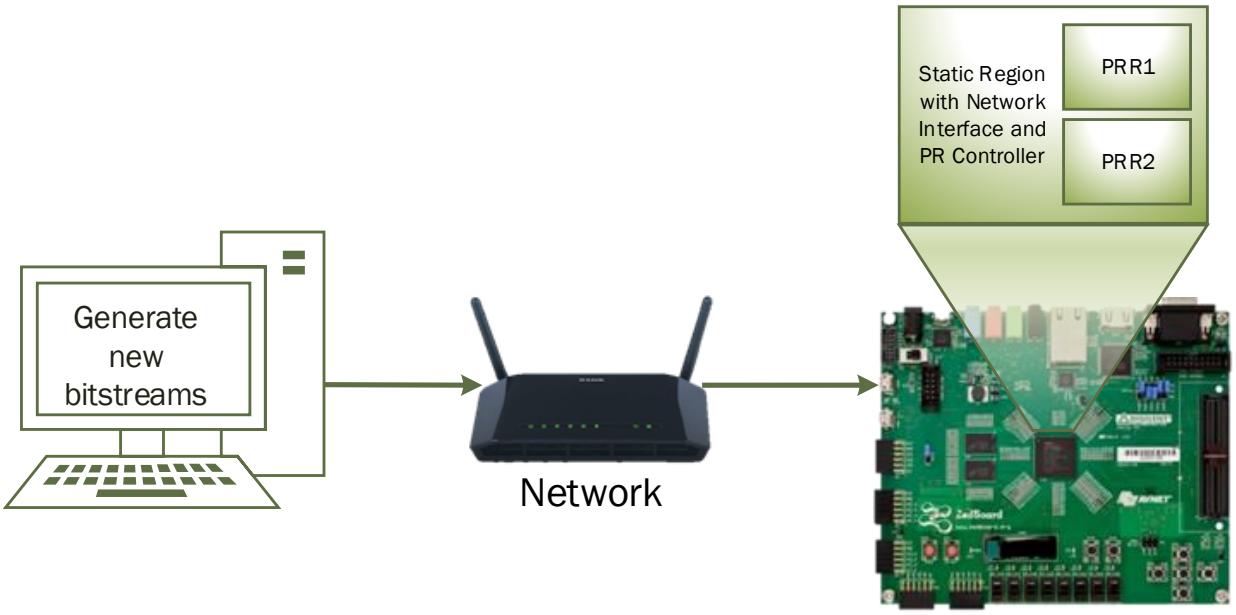

Figure 36: Block diagram for network based partial reconfiguration

This architecture is more suitable for reprogramming the device for entire new purpose rather than reprogramming it for the smaller chunks of a bigger implementation as more delays will be added while reconfiguration because of network involvement.

REFERENCES

1. T. Becker, W. Luk and P. Y. K. Cheung, "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration," 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007), Napa, CA, 2007, pp. 35-44, doi: 10.1109/FCCM.2007.51.
2. E. Eto. 2007. XAPP290: Difference-Based Partial Reconfiguration. Technical Report. Xilinx Inc.
3. Xilinx Inc. 2013b. UG682: PlanAhead User Guide. Xilinx Inc
4. Xilinx Inc. 2015. UG570: UltraScale Architecture Configuration. Xilinx Inc.
5. Xilinx Inc. 2017c. UG909: Vivado Design SuiteUser Guide Partial Reconfiguration. Xilinx Inc.
6. Xilinx Inc. 1996. Programmable Logic Data Book
7. Xilinx Inc. 2011a. DS083: Virtex-II Pro and Virtex-II Pro-X Platform FPGAs. Xilinx Inc
8. Xilinx Inc. 2008. UG070: Virtex-4 FPGA User Guide. Xilinx Inc
9. Xilinx Inc. 2013a. UG585: Zynq-7000 All Programmable SoC Technical Reference Manual. Xilinx Inc
10. Kizheppatt Vipin. 2014. ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq, Student Member, IEEE, and Suhaib A. Fahmy, Senior Member, IEEE
11. https://store.digilentinc.com/zedboard-zynq-7000-arm-fpga-soc-development-board/
12. Altera. 2013a. Design Planning for Partial Reconfguration. Altera
13. Altera. 2017. ug-partrecon : Partial Reconfguration IP Core
14. Altera. 2016a. Arria 10 CvP Initialization and Partial Reconfguration over PCI Express User Guide.

15. How and S. Atsatt. 2016. Sectors: Divide & Conquer and Softwarization in the Design and Validation of the Stratix-10 FPGA. In Proceedings of IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM).
16. Xilinx, 7 Series FPGAs Configurable Logic Block User Guide, 2014, UG474
17. JTAGulator A hardware hacking and On chip debugging tool for novice hackers and hobbyists http://www.grandideastudio.com/jtagulator/
18. Xilinx, 7 Series FPGAs Configuration User Guide, 2015, UG470 (v.1.10).
19. Xilinx, Inc. Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs
20. Xilinx, Soft Error Mitigation Controller, 2014, PG036 (v4.1). [Online]. Available: http://www.xilinx.com/support/documentation/ip documentation/sem/v41/pg036 sem.pdf
21. Xilinx, Zynq-7000 AP SoC Technical Reference Manual, 2018 UG585 (v.1.10).[Online]. Available: http://www.xilinx.com/support/documentation/userguides/ug585-Zynq-7000-TRM.pdf
22. Vipin, Kizheppatt, and Suhaib A. Fahmy, 2018 "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications." ACM Computing Surveys (CSUR) 51.4 ; 72.
23. Voros, Nikolaos, et al., 2018 eds. Applied Reconfigurable Computing.Architectures, Tools, and Applications: 14th International Symposium, ARC 2018,Santorini, Greece, May 2-4, Proceedings. Vol. 10824. Springer, 2018