Chapter 2

FPGA and Dynamic Reconfiguration

This chapter will introduce a family of silicon devices, FPGAs exploring their architecture. This work is based on these particular devices. The chapter will illustrate a particular feature that is available with recent FPGA devices, called Dynamic Partial Reconfiguration. This work is based on use of this feature. Hence in order to fully understand this feature, the concepts of static and dynamic reconfiguration and the taxonomy of dynamic reconfiguration design techniques is presented in this chapter.

This work makes use of FPGAs by Xilinx® Inc. Hardware Description Language (HDL) such as VHDL and Verilog are used to describe the functionality to be implemented on device. Set of design tools provided by Xilinx are used to develop and implement all applications [8–18]. This thesis deals with low level configuration details of FPGA (particularly Xilinx Virtex® -4 FPGA), so a part of this chapter is also devoted to explore the configuration ports and file type that contains the configuration informa-
2.1 FPGA

FPGA is an Integrated Circuit (IC) that includes two-dimensional array of general-purpose logic circuits, whose functions are programmable. FPGA can be programmed to implement virtually any set of functions. Input signals are processed by the programmed circuit to produce the desired set of outputs. Such input signals flow from user’s system, through input buffers and through the circuit, and finally back out to the user’s system via output buffers. The programmed logic elements in the gate array are connected together by routing resources to form a desired integrated circuit. The routing resources are connected to each other and to the logic elements in the gate array by programmable elements. Programming of the logic blocks, the routing network and the Input/Output (I/O) cells can be selectively done to make the necessary interconnections that establish one configuration thereof to provide the desired system operation/function for a particular application.

The flexibility of having custom, changeable hardware in an application is the factor that has determined the popularity of FPGA devices in the broad range of application domains viz. digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and growing range of other areas. [19]

FPGA can be paired with a GPP, this methodology is commonly referred as hardware-software co-design. In this technique, most demanding
section of an application is implemented into the FPGA that can accelerate the program execution. This technique is increasingly used in conventional high performance computing applications where computational kernels such as FFT or Convolution are performed on the FPGA while microprocessor continues to execute other program. Considerable speed up can be observed when applications or algorithms can make use of the massive parallelism offered by FPGAs. In recent FPGAs embedded microprocessors are also available along with reconfigurable array to form a complete System-on-Chip (SoC). If such in-built processor core is not available, soft processor cores provided by FPGA vendors can also be used.

FPGAs can also be used without pairing with microprocessor, by implementing all of the application functionalities in hardware. In this case the hardware implemented on FPGA covers all the data paths from the inputs to the outputs of the application. The advantage in this case is that the hardware is easily replaceable rather than having the circuit physically replaced.

2.2 FPGA Architecture

Architecture of FPGA is briefly discussed in this section. The aim is to describe how FPGA can implement custom hardware and how it can be changed by reconfiguring the device. The FPGAs used in this work are manufactured by Xilinx. Hence the following architectural details apply to Xilinx FPGAs. However, the general principle of FPGAs across all manufacturer remains the same. The generic architecture of FPGA is shown in Fig 2.1.
The three main blocks of an FPGA are:

- Configurable Logic Blocks (CLB).
- Input-Output Blocks (IOB).
- Communication Channels.
2.2.1 Configurable Logic Blocks

CLBs are the main components of an FPGA. They can have one or more function generators realized with Look-up Table (LUT). That can implement any functionality, from basic logic gates to more complex combinatorial functions. In these components, the result of the function is stored for every possible combination of the inputs, such that a 4-bit LUT has 16 \((2^4)\) memory cells to store the function. Around LUT there is interconnect logic that routes signals to and from the LUT, implemented using standard logic gates, multipliers and latches. Therefore during configuration process of an FPGA, the memory inside the LUT is written to implement a required function, and a logic around it is configured to route the signals correctly in order to build more complex system around this basic building block. In the vast majority of FPGAs, the logic blocks also include memory elements, Flip-Flop (FF), that enables to implement sequential functionalities also. In Xilinx FPGAs a single CLB contains a set of four slices and each slice contains two LUTs and the required interconnect hardware. An architecture of a slice is shown in Fig. 2.2 (Source: [20])

2.2.2 Input-Output Blocks (IOB)

IOBs contain circuitry that facilitates the transfer of signals to and from Input/Output pads of the FPGA. An IOB allows signals to be driven off-chip or brought onto the FPGA interconnect segments. IOBs provide interface between external pins of IC package and internal signal lines, including programmable interconnects. IOBs have their own configuration memory to store the voltage standard to which the I/O pin must comply and to configure the direction of communication through it. It is possible
to establish unidirectional as well as bidirectional links through it. IOBs can typically perform other functions, such as tri-stating outputs and registering incoming or out-going signals.

### 2.2.3 Communication Channels

interconnection resources within FPGA allow connection of CLBs and IOBs. Main modes of interconnection are direct and segmented. Direct interconnection is made of groups of connections that cross the device in all directions. Logic blocks put data on the most suitable channel according
to data destination. Segmented interconnection is based on lines that can be interconnected using programmable switch matrix for interconnection lines. In case of direct interconnection, parasitic resistance and capacitance of communication channel is constant. This helps in predicting the delay introduced by channels. In segmented interconnection connection lines are present only between the blocks, hence the power dissipation is less. Segmented interconnection also maximize speed of interconnection and limits the skew.

2.2.4 Additional Resources

FPGAs include other specialized blocks, such as Processor [21], Block RAM (BRAM), Digital Signal Processor (DSP) and Multipliers etc. These specialized blocks perform more specific tasks than the CLBs, but can still be configured in accordance with a variety of options to enable flexible operation of the FPGA. In addition, FPGAs also include other blocks, such as Digital Clock Manager (DCM), which contain circuitry required to manipulate clock signals. These additional cores on FPGA enrich its functionalities and improve the overall speed of architectures that make use of them. Fig. 2.3 shows internal structure of Virtex FPGAs from Xilinx.

A system consisting of above mentioned building blocks makes a reconfigurable hardware. The memory cell attached to every configurable element controls its functioning. Functionality is controlled by LUT equations which is in turn controlled by the contents of memory cells of LUT. The interconnection between the functional element is controlled by communication resources which is controlled by setting appropriate bits in switching matrix. The functionality is finally connected to I/O pins through IOBs. The I/O voltage standard and direction of signal is controlled by
Figure 2.3: Architecture of Virtex FPGAs
setting corresponding memory cell in IOB. The configuration memory associated with these blocks is Static Random Access Memory (SRAM) and is therefore can be changed. In this way by changing the contents of configuration memory, functionality provided by FPGA can be changed.

As SRAM is volatile memory, when the power of FPGA is made OFF, it looses its contents and can be started fresh with new contents for new functionality. Usually an external entity downloads the configuration on FPGA via one of the configuration interfaces. After the configuration has completed it sends start command to the device to begin with functionality. Generally, FPGAs are backed up with ROMs where configuration data is stored. On power-on this data is downloaded into the FPGA configuration memory. This configuration data is called bitstream and can be either full or partial according to the extent of configuration memory addressed in it. Next sections describe Dynamic Partial Reconfiguration (DPR), bitstream format and configuration ports available in FPGA.

2.3 Dynamic Reconfiguration

Hardware functionalities on FPGA can be changed by taking the application offline, downloading a new configuration on FPGA and restarting the execution as discussed in Section 2.2. This type of reconfiguration is called Static Reconfiguration. With static reconfiguration, in order to change the functionality FPGA has to be powered off which imposes overhead of time for functional diversity [2].

In contrast to the static reconfiguration, Dynamic Reconfiguration al-

\footnote{One time programmable (OTP) FPGAs based on Anti-fused Technology are not considered here.}
CHAPTER 2. FPGA AND DYNAMIC RECONFIGURATION

This scheme allows changing hardware functionality of FPGA without taking the application off-line. This scheme gives the capability for adapting the hardware configured on the chip according to the needs of a particular situation during the execution time. Dynamic reconfiguration is used extensively in this work. Dynamic reconfiguration has its own particular taxonomy according to features of the system, as explained hereafter.

2.3.1 Total vs. Partial Reconfiguration

In total reconfiguration the configuration bitstream, containing the FPGA configuration data, provides the information regarding the complete chip and it configures entire FPGA. In partial reconfiguration, only a portion of the device is reconfigured, while the rest of the hardware mapped on the FPGA can continue to operate transparently with respect to the reconfiguration process. The Dynamic Partial Reconfiguration (DPR) scheme is explored within this work.

DPR provides a way to reconfigure a chosen area of the FPGA while the other areas remain operational without having to shut it down. Thus, the hardware is modified on-the-fly, triggered by the application. This is particularly of great advantage in real-time embedded systems when turning off the system is not affordable [6]. DPR offers countless benefits like adapting hardware algorithms during system runtime, share resources, reduced power consumption and shortening reconfiguration time.

2.3.2 Models of Dynamic Partial Reconfiguration

Dynamic Partial Reconfiguration capabilities of FPGA allows great flexibility in HW design and as a consequence, they make it possible to create
different models for reconfigurable systems as shown in Fig. 2.4 [1]

- **Module based vs. Difference based.**

  Depending upon the granularity levels or the size of reconfigurable area, DPR can be categorized as *Difference based* and *Module based*. Difference based PR is used for very small changes like changing logic of one CLB, DCM, contents of BRAM etc. Module based PR is used when considerably larger area is to be reconfigured. This is done by creating hardware modules that can be added and removed from the system dynamically. [22–24]

- **Internal vs. External.**

  Dynamic reconfiguration can be either *External* or *Internal/Self reconfiguration* depending upon the entity that performs the reconfiguration. If the reconfiguration is handled by an external device like PC or processor it is known as *External reconfiguration*. In this case all the commands for reconfiguration are issued from outside the FPGA. However some devices can reconfigure it’s own part internally. In this case it is possible to have processor on the FPGA to reconfigure some resources. This is known as *Internal or Self reconfiguration*, performed within the FPGA. Internal reconfiguration can give way to full systems on chip that can adapt their own hardware in response to changing application needs.

- **One Dimensional vs. Two Dimensional.**

  Dynamic reconfiguration can take place in a *One Dimensional* (1D) or in a *Two Dimensional* (2D) space as shown in Fig. 2.5. In case of 1D reconfiguration the resources that can be reconfigured using DPR al-
CHAPTER 2. FPGA AND DYNAMIC RECONFIGURATION

1. Module based
2. Difference based

1. Internal / Self
2. External

1. One Dimensional (1D)
2. Two Dimensional (2D)

1. Static Allocation
2. Dynamic Allocation

Figure 2.4: Classification of Dynamic Partial Reconfiguration
ways span the whole height of the device. In other words it is mandatory to configure the whole column of the device. This compulsion is imposed by the architecture of some devices such as Spartan-3 and Virtex-II from Xilinx, since the smallest configurable unit in these devices is the frame, that reconfigures entire column of the device. This obligation is not present in further generations of devices like Virtex-IV and onwards. Since these devices support two dimensional reconfiguration, the designer can choose width and height of reconfigurable module. It gives further flexibility to the designer. [1]

Fig.2.5 (a) illustrates 1D reconfiguration, showing that the entire FPGA height is occupied by Partial Reconfigurable Region (PRR). It can be observed in Fig.2.5 (b) that the constraint on height is relaxed, thus reconfigurable regions do not span the entire height of the device.

- **Static Allocation vs. Dynamic Allocation.** Depending upon when the decision about reconfigurable module placement is taken it can be classified as static allocation and dynamic allocation. In case of static
allocation generation of configuration is done in static way for the respective location on the device. Dynamic location gives the flexibility to relocate the reconfigurable region on device based on available space during run time.

2.4 Configuration Bitstream

Bitstream is a binary file in which configuration information for a particular device is stored. This is the data which has to be copied onto the FPGA SRAM memory in order to configure the device for functionality. Bitstreams can be either partial or full. A full bitstream configures the whole configuration memory of the device, and it is used for static configuration at the power up of FPGA. Partial bitstreams configure only a portion of the device and generated through partial reconfiguration design flow [9, 10]. Irrespective of whether the bitstream is full or partial it has fixed pattern. In this section details of Xilinx Virtex-4 configuration bitstream is given. But the bitstream composition and underlying mechanism remains same for all FPGAs with minor variations. The reference documents used for understanding the bitstream details are [25–28].

2.4.1 Bitstream Structure

A bitstream consists of following components as shown in Fig. 2.6 (Source: [29]).

- **Header**: It has information about file name, device name, date and time of creation and size of raw-bitstream \(^2\). All this information is

\(^2\) actual configuration data
represented in ASCII format [30]. It uses keys and length to divide the header as shown in Table 2.1

- **Dummy and Synchronization word**: Raw bitstream always starts with a dummy word (0 X FFFFFFFF) and a synchronizations word (0 X AA995566). The purpose of these words is to align 32-bit word boundaries and to signal the start of the actual configuration commands.

- **Configuration Register Write**: This part contains series of packets to write various configuration registers. These packets consists of payloads and respective header.

- **Configuration Frame Data Write**: This part contains series of packets to write the frame data. These packets consists of payloads and respective header.
• **Configuration Register Write**: Another sequence of packets to write configuration registers and issue end of configuration command and a D-synchronization word.

### 2.5 Configuration Ports

FPGAs can be configured through an interface to the configuration memory. Different interfaces are available with the contemporary FPGAs to load the bitstream data into it. An overview of these interfaces available in Xilinx Virtex-4 FPGA is given in this section. For detailed information [26] can be referred.

#### 2.5.1 Static Reconfiguration

SelectMAP, Boundary-Scan and Master/Slave serial are the three external interfaces available in Virtex4 for its configuration. Static reconfiguration of Xilinx FPGA devices can be done through one of these ports.

• **Slave or master SelectMAP mode**. SelectMAP mode allows parallel reading and writing of the configuration memory. It works in 8 bit and 32 bit mode. An external clock source, microprocessor, download cable or other FPGA is required for reconfiguration. The data is loaded 8-bit or 32-bit per clock. This mode is typically used as a configuration mode on devices when configuration speed is an important factor.

• **Boundary-Scan mode**. It is an industry standard (IEEE 1149.1, and 1532) serial programming mode. External logic from a cable, microprocessor, or other device is used to drive the JTAG specific pins,
Table 2.1: Structure of Bitstream Header

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field 1</td>
<td>2 bytes Length 0x0009 (Big Endian)</td>
</tr>
<tr>
<td></td>
<td>9 bytes Some sort of Header</td>
</tr>
<tr>
<td>Field 2</td>
<td>2 bytes Word 0x0001</td>
</tr>
<tr>
<td>Field 3</td>
<td>1 byte Key 0x61 (The letter &quot;a&quot;)</td>
</tr>
<tr>
<td></td>
<td>2 bytes Length 0x value (value depends upon file name length)</td>
</tr>
<tr>
<td></td>
<td>value bytes string &quot;design name&quot; e.g.&quot;test.ncd&quot; (including a trailing 0x00)</td>
</tr>
<tr>
<td>Field 4</td>
<td>1 byte Key 0x62 (The letter &quot;b&quot;)</td>
</tr>
<tr>
<td></td>
<td>2 bytes Length 0x value1 (value1 depends upon device name length)</td>
</tr>
<tr>
<td></td>
<td>value1 bytes string &quot;device name&quot; e.g.&quot;4vx45ff668&quot; (including a trailing 0x00)</td>
</tr>
<tr>
<td>Field 5</td>
<td>1 byte Key 0x63 (The letter &quot;c&quot;)</td>
</tr>
<tr>
<td></td>
<td>2 bytes Length 0x000b</td>
</tr>
<tr>
<td></td>
<td>11 bytes string &quot;date of creation&quot; e.g.&quot;2010/06/24&quot; (including a trailing 0x00)</td>
</tr>
<tr>
<td>Field 6</td>
<td>1 byte Key 0x64 (The letter &quot;d&quot;)</td>
</tr>
<tr>
<td></td>
<td>2 bytes Length 0x0009</td>
</tr>
<tr>
<td></td>
<td>9 bytes string &quot;time of creation&quot; e.g.&quot;06:51:17&quot; (including a trailing 0x00)</td>
</tr>
<tr>
<td>Field 7</td>
<td>1 byte Key 0x65 (The letter &quot;e&quot;)</td>
</tr>
<tr>
<td></td>
<td>4 bytes size of raw bit file</td>
</tr>
</tbody>
</table>
Test Data In (TDI), Test Mode Select (TMS), and Test Clock (TCK) and sense device response on Test Data Out (TDO). This mode is the most popular mode of configuration due to its standardization and ability to program FPGAs, PLDs, and PROMs through the same four JTAG pins. The data in this mode is loaded at one bit per TCK.

- **Slave or master serial mode.** Serial mode allows daisy-chain configurations and is supported by all Xilinx FPGA families. In this mode, an external clock, a master such as a microprocessor or another FPGA and download cable is required. Data is loaded at one bit per clock.

### 2.5.2 Dynamic Partial Reconfiguration

- **External DPR.**

SelectMAP and Boundary Scan interfaces allow reading and writing of configuration data from and to configuration memory respectively during application run time. Hence these ports can also be used for dynamic reconfiguration. With Boundary-scan interface one bit of configuration bitstream can be written per clock cycle. SelectMAP interface works in 8 bit and 32 bit mode. SelectMAP is faster interface for configuration as well as reconfiguration [31].

In order to use SelectMAP pins for reconfiguration, Virtex4 devices has to be configured initially to retain the SelectMAP pins, allowing further reconfiguration via those pins. Otherwise SelectMAP pins become user I/O pins after configuration. Since external entity such as processor has to initiate and complete the DPR these ports can be used for External DPR.
CHAPTER 2. FPGA AND DYNAMIC RECONFIGURATION

• Internal DPR.

Since the reconfiguration is performed internal to the FPGA it is the fastest way of carrying out DPR. For carrying out internal DPR an internal interface to the configuration memory is required. Internal Configuration Access Port (ICAP) is such interface available in Xilinx Virtex FPGAs. It is a functional subset of SelectMAP as shown in Fig. 2.7.

![SelectMAP and ICAP Reconfiguration Ports in Virtex-4](image)

It is a hardwired FPGA logic that behaves just like the SelectMAP interface in slave mode. The ICAP allows read and write access to the FPGA configuration memory. ICAP interface consists of separate data ports for reading and writing, write, chip enable and busy signals and a clock input. Just like SelectMAP. ICAP data ports can be used as 8 or 32 bit wide in Virtex-4. There are two ICAP sites in
Virtex-4 devices. Functionally they are same and one out of which can be activated at any time. Users must make sure not to reconfigure the circuitry controlling the ICAP. Thus ICAP does not allow configuration of entire FPGA. In contrast to SelectMAP, ICAP does not support multiple modes. Besides the mode pins (M2, M1, M0) that can be found in SelectMAP interface. Other pins such as Done, Init, and Program are missing too. The SelectMAP CS pin has been renamed CE on ICAP but it provides same function. More configuration details about Virtex-4 can be found in [26] [27] [28]

As contribution of this thesis is use of ICAP optimally to enhance throughput of Dynamic Partial Reconfiguration concept of reconfiguration time and throughput are discussed in next section.

2.6 Throughput and Time of Dynamic Partial Reconfiguration

The theoretical reconfiguration throughput of DPR through ICAP can be calculated as:

$$\text{Theoretical Throughput} = \frac{\text{ICAP input data width}}{\text{clock period}}$$ (2.1)

However the ICAP is not always able to process the incoming data. When this happens it raises a busy signal and stalls the reconfiguration process. Moreover, the system could not be able to provide a datum per clock cycle, thus it could not use the ICAP at its theoretical throughput. In the light of these considerations, the reconfiguration time can be computed as:
Rec. Time = $\frac{\text{Bitstream size}}{\text{Data feed throughput}} + t_{\text{busy}}$ (2.2)

Where $t_{\text{busy}}$ is the time lapsed during the busy state of the ICAP and Data feed throughput is the rate at which data are sent to the ICAP. Based on the assumptions that ICAP can process an incoming data at every clock cycle (i.e. $t_{\text{busy}} = 0$), that the surrounding system is able to provide a datum to the ICAP per clock cycle, that the data width is set to 32 bit and that the component works at 100 MHz, the reconfiguration throughput is 400 MB/s.

2.7 A Dynamic Partial Reconfigurable System

Fig. 2.8 shows Module based Dynamically Partially Reconfigurable System. The system consists of following components

- Static Part.
- Dynamic Part.
- Communication Infrastructure.
- Configuration Controller.

The PRR in FPGA is a rectangular block where Reconfigurable Module (RM) are configured dynamically. A DPR system can have multiple PRRs and one PRR can have multiple RMs, out of which one RM can be active at a time. Set of all PRRs is known as dynamic region of DPR system.

Remaining part of the system is known as static region. It contains part of design that needs to be active during the whole application runtime. Reconfiguration controller takes care of loading the RM in respective PRR.
In case of internal reconfiguration the reconfiguration controller resides inside the FPGA as a part static area.

Bus Macro (BM) is slice-based prerouted elements made of simple LUT. They lock the starting and the ending point of the wires to ensure their correct connection after reconfiguration [9], [10]. Stable and safe communication links between RM and static part or other RM are achieved using Bus Macro.

2.8 Remarks

Since their introduction in 1980, Field Programmable Gate Arrays (FPGAs) are gaining importance both in commercial as well as research settings. The increased popularity of FPGAs results from significantly increased capability of FPGAs. Contemporary FPGAs contain thousands of Look-Up Tables (LUTs), Flip-Flops and large variety of built-in digital components e.g. processor, memory, multipliers, transceivers and many
more for implementing complex digital logic. As a result of these advancements, it has become possible to implement complete System on Chip (SoC) on an FPGA. Such FPGA based SoC comprise of a General Purpose Processor (GPP) and hardware logic. The hardware logic works as co-processor and executes application tasks that are computation intensive while GPP performs other functions. This system benefits from performance enhancement but number of possible co-processors is bounded by the available hardware resources. Also these co-processors persists on the device while they are not in use, occupying lot of resources.

Two new features have been introduced in most recent FPGAs to cope up with above mentioned limitations. First is DPR. DPR allows performing the reconfiguration of a portion of FPGA while keeping the unchanged part of device running. Using DPR co-processors can be changed without stopping other system. Hence it is possible to have more co-processors without being limited by the hardware resources and managing them as component library. The second feature is access to the configuration memory from within the device. With such access to the configuration memory within the device, now system can reconfigure itself. This has given flexibility to the system to adapt its hardware autonomously.

During the last decade, DPR has been interesting topic of research in various laboratories all over the world. It is being studied in academia, to evaluate advantages and challenges when using Dynamic Partial Reconfiguration in applications and providing guidelines to increase its applicability in real world applications. The scope of this work is studying and using this feature in applications. Further this thesis proposes a novel reconfiguration controller for Dynamic Partial Reconfiguration in real world, real time applications. The literature survey of the work done
by researchers in the area of dynamic partial reconfiguration is presented in next chapter. The next chapter summarizes previous work and indicates how the objectives of this work are formed.
When I Asked God for Happiness
He Showed Me Some Unhappy People