Module Based And Difference Based Implementation of Partial Reconfiguration On FPGA: A Review

Trailokya Nath Sasamal*, Rajendra Prasad**

*,**(Department of Electrical & Electronics Engineering ,Mewar University,Rajasthan)

ABSTRACT
Dynamically adaptable computing systems are promising research area at developing systems which can adapt to changes in their environment while executing. The premisses for such systems are reconfigurable computing systems which allow the system hardware to be changed periodically in order to execute different applications on the same hardware. Partial reconfiguration is the prerequisite of reconfigurable computing, as it allows time-sharing of physical resources for the execution of multiple design modules. Moreover, partial reconfigurable modules can be swapped in or out on the fly from the operating environment control while other modules in the base design continue functioning without incurring any system downtime. This results in dramatically increase in speed and functionality of FPGA based system. This paper presents a review of existing Partial Reconfiguration methods, Reconfiguration steps, Medium of partial reconfiguration, application on Xillinx FPGA.

***Keywords* - FPGA, Partial reconfiguration, Reconfiguration, Reconfigurable computing**

1. INTRODUCTION
GPPs (general purpose processors) are more flexible as they have capability to execute/compute any kind of task. This portion of an implementation simply as the “software” portion. *Design time* and *NRE cost* are low, because the designer must only write a program, but need not do any digital design. *Flexibility* is high, because changing functionality requires only changing the program. But from performance (in terms of silicon area, power usage and speed) point of view, they are far away from ASICs/ ASIP.
A single-purpose processor results in several design metric benefits and drawbacks, which are essentially the inverse of those for general purpose processors. Performance may be fast, size and power may be small, and unit-cost may be low for large quantities, while design time and NRE costs may be high, flexibility is low. An *application-specific* instruction-set processor (or ASIP) can serve as a compromise between the above processor options. Application specific integrated circuits (ASICs) and application specific instruction set processors (ASIPs) use in order to execute critical tasks quickly. This approach gives us much performance as hardware optimized for a particular application, but not flexibility because application is not always adapt to the hardware. Reconfigurable computing takes the benefits of both hardware and software. It tries to fill the gap between hardware (ASIC/ASIP) and software (GPP/microcontroller) approaches as shown in Fig.1. [2]. Reconfigurable computing is defined as the study of computation using reconfigurable devices [3]. The different models, architectures, compilation and scheduling of tasks, reconfiguration methods, optimal mapping of the design library and the state-of-the-art of reconfigurable computing systems (RCSs) are described in [4].
This paper is organized as follows: section 1 give the brief introduction about reconfigurable computing followed by section 2 that describes the different approaches for reconfiguration, section 3 describes the different types of partial reconfiguration (PR) and reconfiguration steps. Medium of partial reconfiguration are discussed in section 4, section 5 includes conclusion.

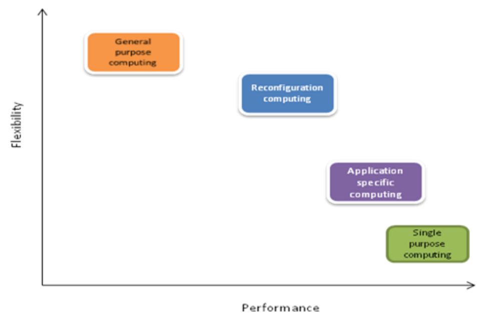

General purpose computing
Reconfiguration computing
Application specific computing
Single purpose computing
Flexibility
Performance

Fig.1 Flexibility vs. performance of existing processors

2. RECONFIGURATION METHODS

Reconfigurability denotes the capability of programmable devices such as FPGAs; to change customized designs by loading different configure [1][3][14]. Reconfiguration can be divided into two groups: Static and Dynamic as shown in Fig.2.

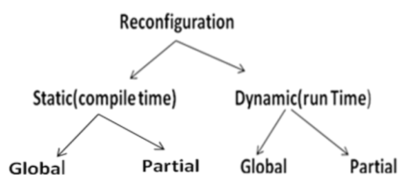


Fig.2. Types of reconfiguration

2.1Static Reconfiguration

This is the most common and simplest reconfiguration approach also referred as compile-time reconfiguration [14]. In this approach, each application consists of a single configuration bitstream. All reconfigurable modules are loaded with their respective configurations before commencing the operation. Furthermore, after starting the operation hardware resources remain static during the whole life span of the application as depicted in Fig.2.

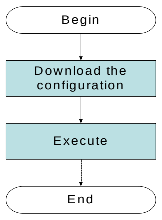


Fig.2.1Static reconfiguration

2.2 Dynamic Reconfiguration

Run-time reconfigurable systems, or dynamic reconfigurable systems, are capable of modifying parts of the functionality configured on the FPGA while the chip is running. By exploiting dynamic reconfiguration, we could build a system which dynamically adapts to the executed application with a fraction of time. This is an advance technique that uses a dynamic scheme to re-allocates the hardware resources at run-time [5], [11]. It utilizes the physical hardware resources in a much better way. As per application demand, it allows hardware resources to be logically added or removed from the operating control environment on the fly while other base modules continue to operate. Dynamic reconfiguration (also called as RTR) allows reconfiguration and execution to proceed at the same time as depicted in Fig.2.1. Statically reconfigurable devices require execution interrupt. The idea behind Partial dynamic reconfiguration or PR technique is to reconfigure only the needed part of the device. Partial reconfiguration is not supported on all FPGAs. For example, the Xilinx Virtex series of FPGAs (Virtex,Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4) allows partial reconfiguration of the FPGA.

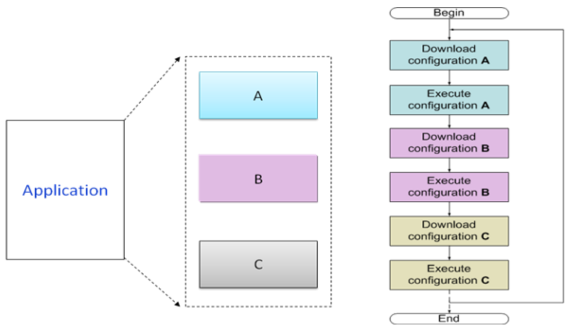


Fig.2.2 Dynamic Global reconfiguration

3. PARTIAL RECONFIGURATION

An important feature in the Virtex architectures is the ability to reconfigure a portion of the FPGA while the remainder of the design is still operational. Partial reconfiguration is useful for applications that require the flexibility to change portions of a design without having to completely reconfigure the entire device. With this capability, entirely new application areas become possible: In-the-field hardware upgrades and updates to remote sites, Runtime reconfiguration, Other potential benefits include: Reduced device count, Reduced power consumption, More efficient use of available board space. This vendor-dependent technology provides common benefits in adapting hardware algorithms during system runtime, sharing hardware resources to reduce device count and power consumption, shortening reconfiguration time, etc. [5][6][16]. There are two styles of partial reconfiguration of FPGA devices from Xilinx: **module-based** and **difference-based**.Typically partial reconfiguration is achieved by loading the partial bitstream of a new design into the FPGA configuration memory and overwriting the current one. Thus the reconfigurable portion will change its behavior according to the newly loaded configuration. The reconfiguration speed (or reconfiguration throughput) is a significant parameter, which determines the switching time of PR modules. This factor must be taken into account in many cases where performance-critical applications require fast switching of IP cores.

3.1Module-based Partial Reconfiguration

Module-based partial reconfiguration permits to reconfigure distinct modular parts of the design. To ensure the communication across the reconfigurable module boundaries, special bus macros ought to be prepared. It works as a fixed routing bridge that connects the reconfigurable module with the rest part of the design. Module-based partial reconfiguration requires to perform a set of specific guidelines during at the stage of design specification. Finally for each reconfigurable module of the design, separate bit-stream is created. Such a bit-stream is used to perform the partial reconfiguration of an FPGA.

Reconfigurable modules (RM) has been modeled in VHDL and implemented on Xilinx Virtex-4 (XC4VFX12) FPGA board with partial reconfiguration. Partial reconfiguration saves the silicon area by allowing multiple configurations to be swapped in or out of the device and provide flexibility to selectively replace the one configuration by the other explains in [17].

Problem of this design flow is that the partial reconfiguration bitstream can only be placed to a fixed region. During dynamic reconfiguration if the allowed partial reconfigurable region occupied by other partial reconfigurable module, then a new partial bitstream file must be generated. This increase the cost of storing the partial bitstream.

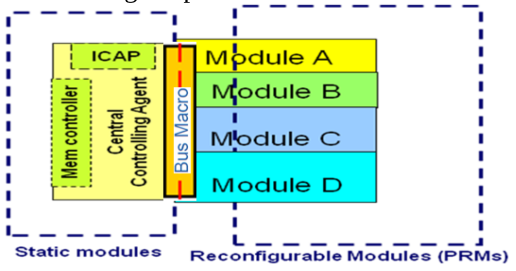


Fig.3.1 static and reconfigurable Modules

FPGA modules are divided into two parts: static and reconfigurable modules. The reconfigurable parts (A, B, C, D modules, etc.) are independent parts of the input design that need not be active during the whole application runtime, Fig.3.1. They share common areas (slots) inside a target device; this is based on the assumption that they are not required to run at the same time in parallel. Usually they reside as a bitstream in a memory outside the FPGA. There some cases they also can be stored in the memory available inside the FPGA (Block RAM). The static part is such a part of an input design that is active during the whole application runtime. It is placed in the “static” area of a target device that is kept intact all the time. In addition to its standard function it has to provide an infrastructure to load and unload other (dynamic) parts of the design, which is system scheduling, data management, and interface management.

The static part must include the configuration controller and logic required for data and interface management. All inputs/outputs of the application are managed by the static part that communicates with dynamic modules through a fixed interface called Bus micros.

3.1.1 Bus micros

Bus macros are tri-state buffers at relative fixed positions which used as fixed data paths for signals going between a reconfigurable module and another module i.e. locking the routing between the dynamic and the static part and all connections between the dynamic and static part must pass through a bus macro.The HDL code should ensure that any reconfigurable module signal that is used to communicate with another module does so only by first passing through a bus macro. There are Virtex, Virtex-E, Virtex-II, and Virtex-II Pro series specific versions of the bus macro. Each bus macro provides for 4 bits of intermodule communication, Fig.3.1.1. As many bus macros as needed must be instantiated to match the number of bits traversing the boundaries of the reconfigurable modules.

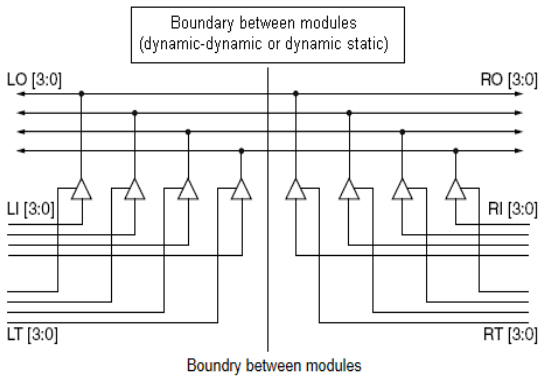


Fig.3.1.1 Bus macro for 4 bits of intermodule communication

As an example, if reconfigurable module A communicates via 32 bits to module B, then eight (32/4) bus macros will need to be instantiated to define the data paths between modules A and B.

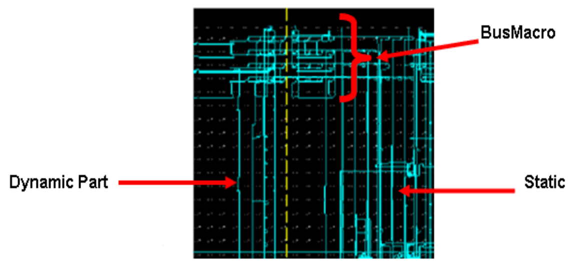


Fig. 3.1.2 Placement of bus micro using FPGA editor

Dataflow in bus macro is always unidirectional. So Bus macros configured for Data flow in only one direction and it is fixed, Fig.3.1.2. Xilinx Provides these Bus Macros.
Partial reconfiguration of Virtex devices can be accomplished through the SelectMAP, JTAG, or ICAP configuration interfaces. Instead of resetting the device and performing a complete reconfiguration, new data is loaded to reconfigure a specific area of a device, while the rest of the device is still in operation. The difference-based partial reconfiguration design flow described in this paper allows a designer to make small logic changes using *FPGA_Editor* and generate a bitstream that programs only the difference between the two versions of the design. Switching the configuration of a module from one implementation to another is very quick because the bitstream differences can be much smaller than the entire device bitstream. The partial reconfiguration feature has been investigated in some applications such as [11][12][17][18].

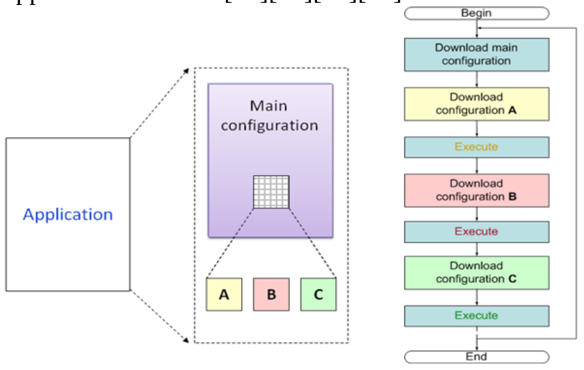
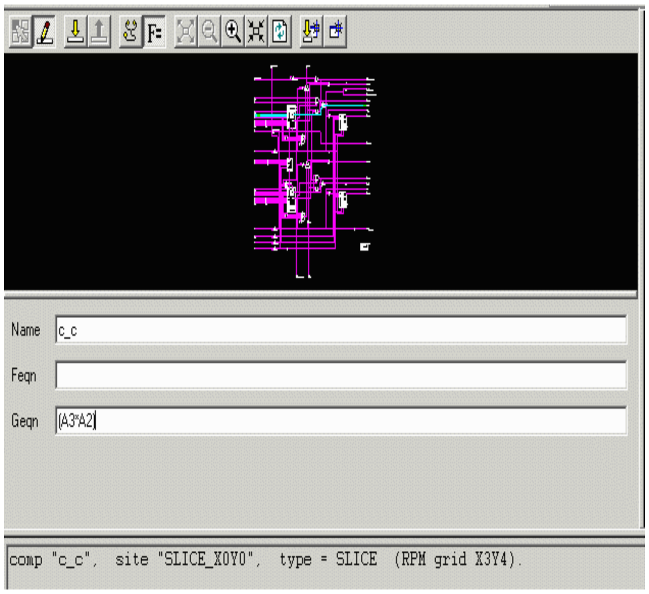


Fig.3.1.3 Dynamic partial Global reconfiguration

3.2 DIFFERENCE BASED PARTIAL RECONFIGURATION

Difference-based design is currently very efficient for small design .The idea of this type of PR is as following: Suppose there are two partial reconfigurable modules, namely X and Y .Instead of creating a full configuration bitstream for each of them, we only create the full configuration bitstream of X. For module Y, we compare its circuit description frame-by-frame with that of module X and then create a partial reconfiguration bitstream that only modifies the frames with differences. With that partial reconfiguration bitstream, we can reconfigure module X into module Y. Difference-based partial reconfiguration, which is useful for making small on-the-fly changes to design parameters such as logic equations, filter parameters, and I/O standards. This design flow is not recommended for making large changes in the functionality or structure of a design, for example, changing an entire algorithm. When there are sizable changes or the routing has to be modified, the recommended flow is to start from the HDL.

The main objective for difference-based partial reconfiguration is allow small design changes. For example, perhaps LUT programming or an I/O standard needs to be simultaneously changed and loaded. These changes can be made easily by directly editing the routed NCD file in the Xilinx *FPGA_Editor* application as shown in Fig.3.2. While many different types of changes can be made to an FPGA design, this paper addresses changing I/O standards, block RAM contents, and LUT programming using *FPGA_Editor*.
After the changes are made, the BitGen program is used to produce a bitstream that only programs the differences between the original design and the new one. Depending on the changes, this partial bitstream can be much smaller than the original bitstream. These bitstreams can be loaded quickly and easily by the software. All that is required is an understanding of how to make logic changes using the *FPGA_Editor* application.



```
Name  c_c
Feqn
Geqn  (A3*A2)

comp "c_c",  site "SLICE_X0Y0",  type = SLICE  (RPM grid X3Y4).
```

Fig.3.2 shows changing the Geqn from A3*A2 to A3*~A2.

4. MEDIUM OF PARTIAL RECONFIGURATION

Four methods of reconfiguring a device, each has applications where desirable:

- Externally
  - Serial configuration port
  - JTAG (Boundary Scan) port
  - SelectMap port
- Internally
  - Though the Internal configuration access port (ICAP) using an embedded microcontroller or state machine

Xilinx is the main vendor whose silicon products and tools support the PR feature. The Virtex-II,Virtex-4 and Virtex-5 family FPGAs all provide a dedicated Internal Configuration Access Port (ICAP), which allows for internal access to and modification of the configuration data, Fig.4.1. Having ICAP on-chip leads to an easy and efficient way of building a run-time self-reconfigurable system, in which a soft-processor is using the ICAP as a user-friendly configuration controller. The ICAP is a simplified substitute of the Xilinx SelectMap reconfiguration solution, which is a byte-parallel external reconfiguration interface that can achieve the highest possible reconfiguration speed .It is assumed that the ICAP will only be used for partial reconfiguration, since the part of the FPGA that controls the ICAP must not reconfigured through ICAP. Xilinx's EDK (Embedded Development Kit) is the development package for building MicroBlaze (and PowerPC) embedded processor systems in Xilinx FPGAs. Hosted in the Eclipse IDE, the project manager consists of two separate environments: XPS and SDK. The SDK handles the software that will execute on the embedded system, the SDK enables programmers to write, compile, and debug C/C++ applications for their embedded system,Fig.4.2.
Bitstream flows shown in Fig.4.3.Configurable memory are SRAM(Static RAM) which reconfigured according to the bitstream files stored in Flash PROM. Performance measurement results on the development board, comparing the reconfiguration speed of different architectures; the resource utilization is also analyzed for ICAP designs in [16],[18].

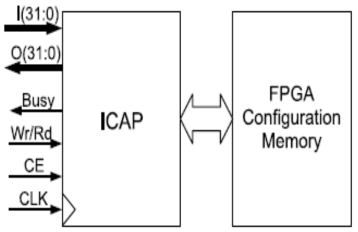
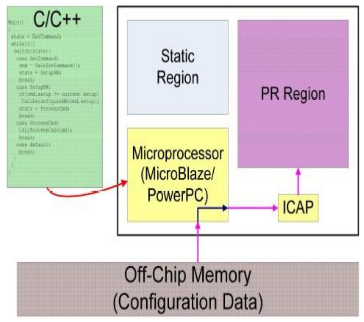


Fig.4.1 The ICAP primitive on Xilinx FPGAs


C/C++
Static Region
PR Region
Microprocessor (MicroBlaze/ PowerPC)
ICAP
Off-Chip Memory (Configuration Data)

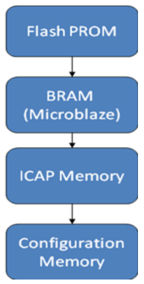


Fig.4.2 PR design using embedded microcontroller

Fig.4.3 Bitstream Flow

Reconfiguration Steps

- Reconfiguration is triggered within the FPGA
- Processor core loads the desired configuration data from external

- This could be from ROM, Flash, static Ram loaded at startup or filled up by the FPGA itself

- Processor reconfigures the PR region through the ICAP primitive

5. Conclusion

Virtex series FPGA have potential in terms of partial reconfiguration but the existing communication/reconfiguration conflict greatly limits the scale of the system in terms of numbers of reconfigurable units. In this paper we discuss about existing dynamic reconfiguration techniques, medium of reconfiguration considering Xilinx virtex-II,Virtex-II Pro FPGA. Discuss the problem related to module-based partial reconfiguration.

References

[1] K. Bondalapati and V. Prasanna. “Reconfigurable Computing systems,” in Proc. IEEE, vol. 90, no.7, pp.1201-1217, July 2002.
[2] Katherine Compton and Scott Hauck, “Reconfigurable Computing: A Survey of Systems and Software,” ACM Computing Surveys, vol. 34, no. 2, pp.171-210, June 2002.
[3] Christophe Bobda, “Introduction to Reconfigurable Computing” Springer 2007.
[4] K. Solomon Raju, M. V. Kartikeyan, R C Joshi and Chandra Shekhar, “Reconfigurable Computing Systems Design: Issues at System-Level Architectures”. The 5th Annual Inter Research Institute Student Seminar in Computer Science (IRISS - 2006), IITM, Chennai, India, January 2006.

[5] Run-time partial reconfiguration speed investigation and architectural design space exploration**,** Ming Liu, Wolfgang Kuehn, Zhonghai Lu, Axel Jantsch.

[6]Two Flows for Partial Reconfiguration: Module Based or Difference Based, Xilinx website [online]
http://www.xilinx.com/support/documentation/application_notes/xapp290.pdf,
[9] K. Paulsson, M. Huebner, S. Bayar, and J. Becker, “Exploitation of Run-Time Partial Reconfiguration for Dynamic Power Management in Xilinx Spartan III base Systems”, In Proc. of 2006 Reconfigurable Communication-centric SOCs,
Jun. 2007.
[10] J. Noguera and I. O. Kennedy, “Power Reduction in Network Equipment Through Adaptive Partial Reconfiguration”, In Proc. of the 2007 Field Programmable Logic and Applications, pp. 240 - 245, Aug. 2007.
[11] "An FPGA-Based Dynamically Reconfigurable Platform: From Concept to Realization", Mateusz Majer, Field Programmable Logic and Applications, 2006. FPL '06. International Conference on, Vol., Iss., Aug. 2006 Pages:1-2

[12] Modular partial reconfigurable in Virtex FPGAs", Sedcole, P.; Blodget, B.; Anderson, J.; Lysaghi, P.; Becker, T., Field Programmable Logic and Applications, 2005. International Conference on, Vol., Iss., 24-26 Aug. 2005 Pages: 211- 216

[13] "ug070 - Virtex4 User guide", www.xilinx.com

[14]"ug071- Virtex4 configuration guide",www.xilinx.com.

[15]"Virtex Series Configuration Architecture User Guide", www.xilinx.com.

[16]"ug208- Early Access Partial Reconfigureation user guide", www.xilinx.com.

[17] Solomon Raju Kota, Ashutosh Gupta, Shashikant Nayak, and Sreekanth Varma. “Module Based Implementation of Partial Reconfiguration Using VHDL on Xilinx FPGA,” in International Journal of Recent Trends in Engineering, Vol 2, No. 7, November 2009.

[18] Nikolaos S. Voros and Konstantinos Masselos, “System Level Design of Reconfigurable Systems-on-Chip” Springer 2005.