DESIGN AND IMPLEMENTATION OF 32-BIT CONTROLLER FOR INTERACTIVE INTERFACING WITH RECONFIGURABLE COMPUTING SYSTEMS

Ashutosh Gupta and Kota Solomon Raju

Digital System Group, Central Electronics Engineering Research Institute (CEERI)\
Council of Scientific and Industrial Research (CSIR), Pilani-333031 (Raj.), India
ashu_gupta09@rediffmail.com,
solomon@ceeri.ernet.in

ABSTRACT

Partial reconfiguration allows time-sharing of physical resources for the execution of multiple functional modules by swapping in or out at run-time without incurring any system downtime. This results in dramatically increase in speed and functionality of FPGA based system. This paper presents the designing an interface controller through UART for execution & implementation of reconfigurable modules (RM) on Xilinx Virtex-4(XC4VFX12), (XC4VFX20) and (XC4VFX60) devices. To verify partial reconfiguration execution at run-time an interface has been designed to make user interaction with the system at run-time. Interface design includes the controllers for controlling the flow of data to and from the reconfigurable modules to the external world (host environment) through busmacros. The controller is designed as static module. All the static as well as dynamic modules are designed and simulated to verify the functionality with supporting simulation tool using ModelSim-6.0d and synthesized with Xilinx 9.1.02i_PR10 (ISE).

KEYWORDS

Reconfigurable computing systems, Partial reconfiguration, FPGA, Reconfigurable modules, Busmacros

1. INTRODUCTION

All Partial reconfiguration is one of the prerequisite in reconfigurable computing as it allows for swapping modules into and out of the device without having to reset the complete device for total reconfiguration. Therefore, reconfiguration time can be reduced by reloading data only to the needed portion of the chip. It also saves silicon space which could be used to implement other functional modules [1]. How to design a system with run-time reconfiguration (RTR)? What are the different technologies that support partial reconfiguration? What are the benefits of RTR (also called as PR) over the fixed implementation? The answers to these questions are the aim of this paper. The design process involves several steps starting from the HDL description of the design to finally merging the static and reconfigurable modules. This paper focuses on: (i) a system design with two partial reconfiguration regions (PRRs), one(PRR1) consists of two arithmetic functional modules (32-bit ADD/SUB) and other(PRR2) consists of left and right shifting functional modules for LEDs, (ii) implementation on XC4VFX12, XC4VFX20 and XC4VFX60 FPGA devices and (iii) controller has been designed for interfacing of RMs with general purpose I/O devices and UART. The rest of the paper is organized as follows: Section 2 describes the partial reconfigurable design methodologies. Section 3 describes the design of controller for interactive interface. Section 4 includes the implementation results, placement and floorplan of the top level design and the run-time execution of various functional modules with time sharing has been shown. Finally, Section 5 includes the conclusions drawn and future scope related to this work.

2. PARTIAL RECONFIGURATION METHODOLOGIES

Partial reconfiguration can be implemented in two ways: one is module based and other is difference based [2, 3]. In module based design total functionality is divided into three parts: top level design, base design, and partial reconfiguration regions. Top level design includes various base and reconfigurable modules instantiated as black-box, all global signals, such as I/Os, global clocks. Whereas difference based partial reconfiguration is used only when small change is made to the design. In difference based method changes are made to the design parameters such as logic equations, I/O standards, and block RAM contents [4]. The changes can be made by directly editing the routed NCD file. Switching configuration from one implementation to other is very quick in difference based partial reconfiguration, but this flow is not recommended for making the design with large change in functionality from one implementation to other. In the work presented in this paper, module based partial reconfiguration design flow is used to implement partially reconfigurable (PR) design on Virtex-4 FPGA. The implementation of PR design is divided into two parts: PR procedure and design methodology.

2.1. PR Procedure

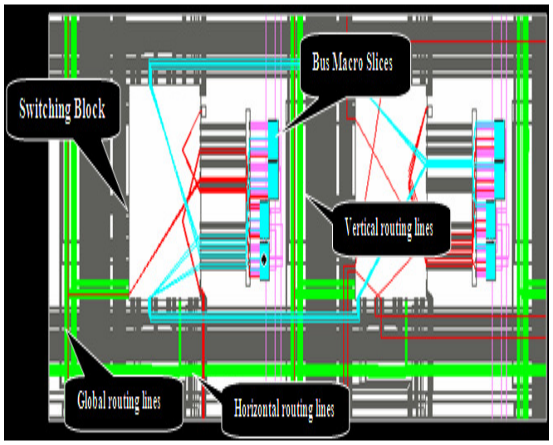
The PR design flow followed is consists of four steps: (1) Design Entry & Synthesis (2) Initial Budgeting (3) Active Module Implementation (4) and finally Assembling the static and dynamic modules. These steps will now be discussed next. During the first steps all the modules, static as well as dynamic are designed using HDL and synthesized. Separate netlists are generated for each module. All modules are ported to the top-level module and top-level module is ported to the I/O pins. Thus, In synthesis properties, I/O buffer insertion must be enabled while synthesizing top module, and disabled for all sub-modules. Top level module is synthesized with certain restrictions such as all the design modules instantiated in the top module with only ports and port directions and signals that connect modules to each other and to I/O pins, all global logic, I/Os must be included in the top level design, and the top module include no logic except clock generation and distribution circuit. All signals except clock signals between PR modules and base design must pass through busmacros. After synthesizing all the modules, the next step (initial budgeting) is to place constraints on the design for place and route. Various types of constraints such as timing constraints, AREA GROUP, AREA GROUP RANGE, LOC, and MODE are placed on the top module. All these constraints are set in the design constraint file called as UCF (user constraint file). Finally, in this phase top level design and static design are implemented via NGDBuild, MAP, and PAR. During base design implementation *static.used* and routed *.ncd* file are generated. The *static.used* file contains the list of routes in PR region that are used by the base design. This static.used file is used as an input to PAR during active module implementation step, whereas the *.ncd* file contains the implemented static design. In third step, all reconfigurable modules are implemented through NGDBuild, MAP, and PAR utilities. The routed *.ncd* files are generated for each reconfigurable module which contains the corresponding implemented reconfigurable module. The last step in PR design flow is assembling the implemented static and reconfigurable design. During this phase, top, base, and reconfigurable modules are merged and a complete design is built from the each reconfigurable module and base design. Complete as well as partial bitstreams are generated using the specific tools. Complete bitstream contains the full design and is used to first configure FPGA, after that partial bitstreams are used to configure FPGA in order to change the logic inside the PR regions at run-time without affecting the functionality of other reconfigurable and static modules.

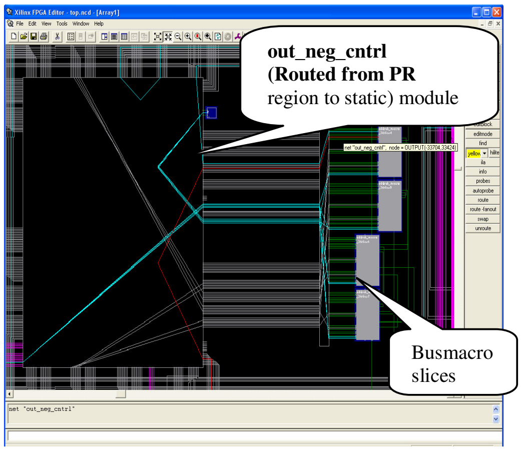
2.2. Design Methodology

In the entire device two PRR regions are selected each having two PRMs. PRR1 contains two arithmetic functions: 32 bit addition and subtraction, whereas the other PRR contains the two logical functions: left shift and right shift. Top-level functionality is divided into the three parts:

top module, base design and reconfigurable modules (RMs). The communication between the base design and PR region has been achieved via special busmacros. These busmacros are used to provide a fixed communication between static and reconfigurable modules. Synchronous bus macros are used here in order to achieve superior timing performance. The bus macros used are narrow bus macros and are two CLB wide and all the signals (except clock signals) routed between base and PR region are passes through busmacros as shown in Figure 1.

In Figure 1, we can observe a signal *out_neg_cntrl* which is used for communication between a static module and one RM in PR region is passes through a busmacro. Now, there is need to make interface to the user. Interface has been designed to accomplish the same. The first PRR is interfaced to the host environment through RS-232 port. User can interact with this PRR through HyperTerminal. This interface design included a controller which has been designed to controls the flow of data from the serial port to PRMs in the PRR and vice-versa. This controller is made as a static logic module ported to the top module. The second PRR is interfaced to the user via general purpose switches. This included another controller which has been designed to control the tasks of taking input from the general purpose switches such as push buttons at run-time and send necessary control signals to the reconfigurable module inside PR region.





(a) (b)

Figure 1. (a) Internal view narrow synchronous busmacro and (b) Communication between static and reconfigurable module through busmacros

3. Design of Controller for Interactive Interface

The state machine model of the controller designed to control the tasks of taking input from serial port and routing inputs to the reconfigurable module in PR region through busmacros and again sending the output result from the reconfigurable module back to serial port through busmacros is shown in Figure 2. The controller establishes the communication between the reconfigurable modules and the serial port. Initially, the controller has been set in state 0 in which it continuously waits for input data from UART receiver. As soon as it receives data, it stores the operands into the registers. In case of invalid or unknown inputs it sends an error message to the HyperTerminal through serial port during the state 18 to state 23. Then it sends the operands to the reconfigurable modules in the PR region through busmacros during the state 5 to State 7. The result received from the PR region is then displayed on the HyperTerminal through UART during state 10 to state 17.

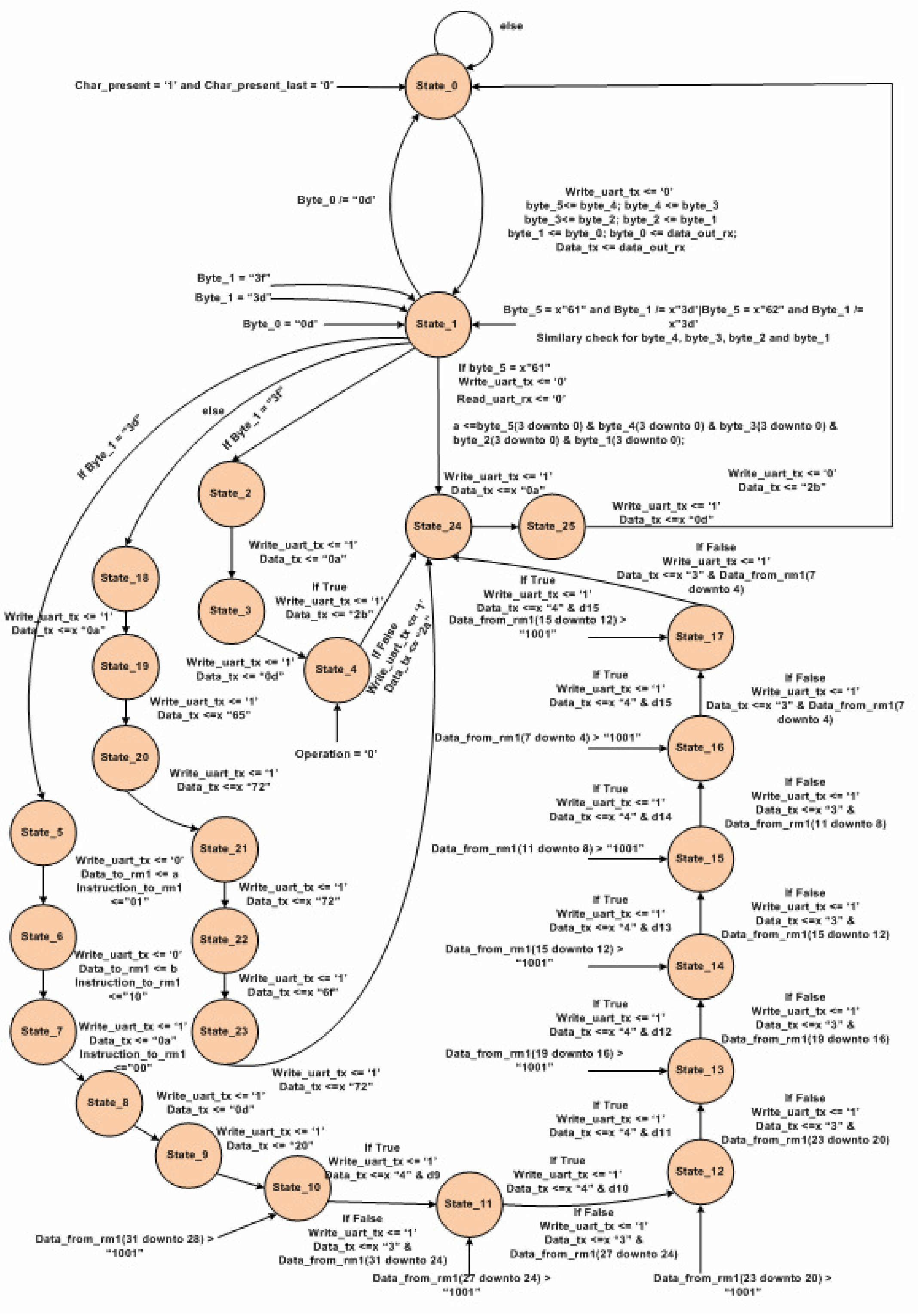


Figure 2. State Machine model of controller

The simulation result of state machine is shown in Figure 3.

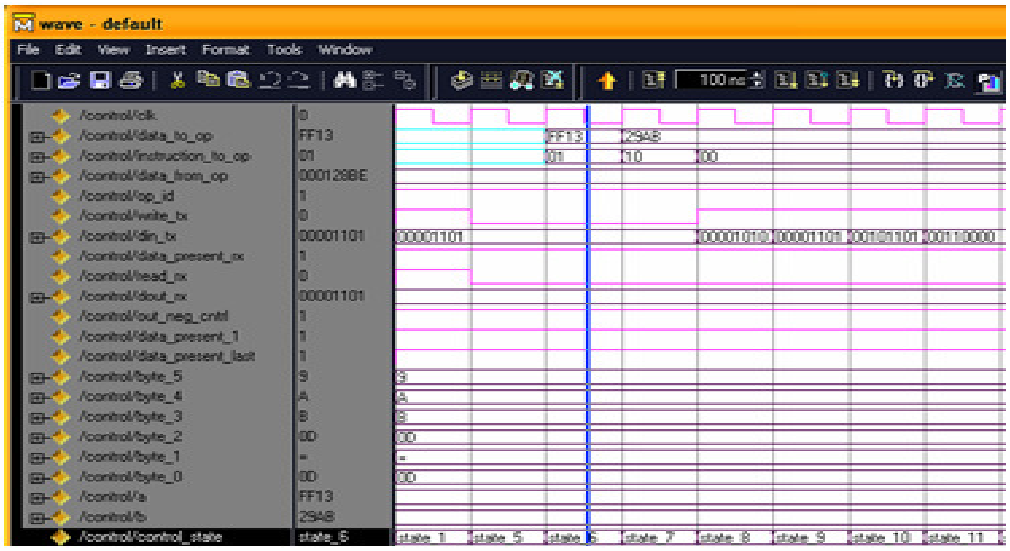


Figure 3. Simulation result for controller state machine

4. Hardware Implementation and Testing

The Boundary Scan mode is used here to configure the FPGA. On device power-up, a full bitstream is first loaded into the FPGA device. Subsequently, partial bitstreams are loaded to change the reconfigurable logic in PR regions on-the-fly. Fixed modules or other reconfigurable modules not being reconfigured are remains fully operational during the reconfiguration of a reconfigurable module. The partially reconfigurable design with user interface has been implemented on Virtex-4 XC4VFX12, FX20 and FX60 device successfully. The architecture view of the system designed is shown in Figure 4.

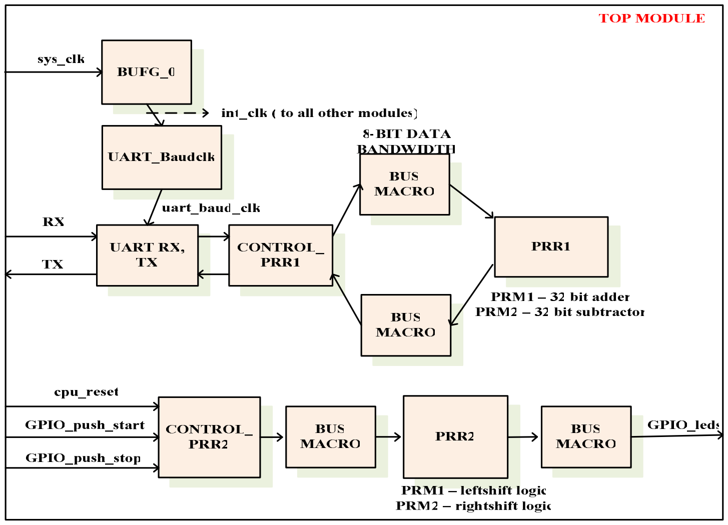
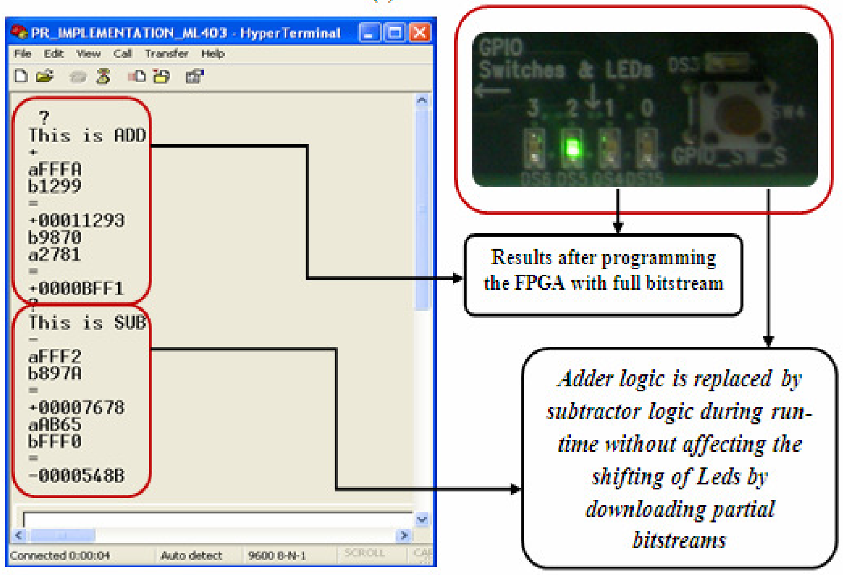


Figure 4. Complete System architecture

The testing of the PR design is shown in Figure 5. The controller controls the flow of data to and from the reconfigurable modules to the external world (host environment) through busmacros. The controller is designed as static modules.


PR_IMPLEMENTATION_ML403 - HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:04
Auto detect
9600 8-N-1
SCROLL
GPIO Switches & LEDs
Results after programming the FPGA with full bitstream
Adder logic is replaced by subtractor logic during run-time without affecting the shifting of Leds by downloading partial bitstreams

```
 ?
This is ADD
+
aFFFA
b1299
=
+00011293
b9870
a2781
=
+0000BFF1
?
This is SUB
-
aFFF2
b897A
=
+00007678
aAB65
bFFF0
=
-0000548B
```

Figure 5. Reconfiguring the module through FPGA device at run-time

The Percentage of hardware resources utilized by the top-level design is shown in Table 1.

Optimization Goal : Speed

Device, Package, Speed : xc4vfx12, ff668, -10

Table 1. Hardware Utilization for Top Module.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	278	10944	2%
Number of 4-input LUT's	484	10944	4%
Number of occupied Slices	348	5472	6%
Number of bounded IOB's	10	320	3%
Number of BUFGs	1	32	3%
Number of Hard Macro	10		
Number used as Shift registers	19		
Total Equivalent Gate Count (not including hard macro)	6,805		

The synchronized bus macros instantiated in the top-level design are as follows:

- *Busmacro_xc4v_l2r_sync_narrow: 6*
- *Busmacro_xc4v_r2l_sync_narrow: 4*

All the signals are completely routed and all the timing constraints are met for the top-level design. The minimum period for the source clock (sys_clk) is **9.126 ns** and therefore, the maximum frequency is **109.577 MHz**. The delay summary report generated after the PAR is as follows:

- The average connection delay for this design is: 0.937ns
- The maximum pin delay is: 3.318ns
- The average connection delay on the 10 worst nets is: 2.373ns

Floor-plan and Placement of top-level design is shown in Figure 6.

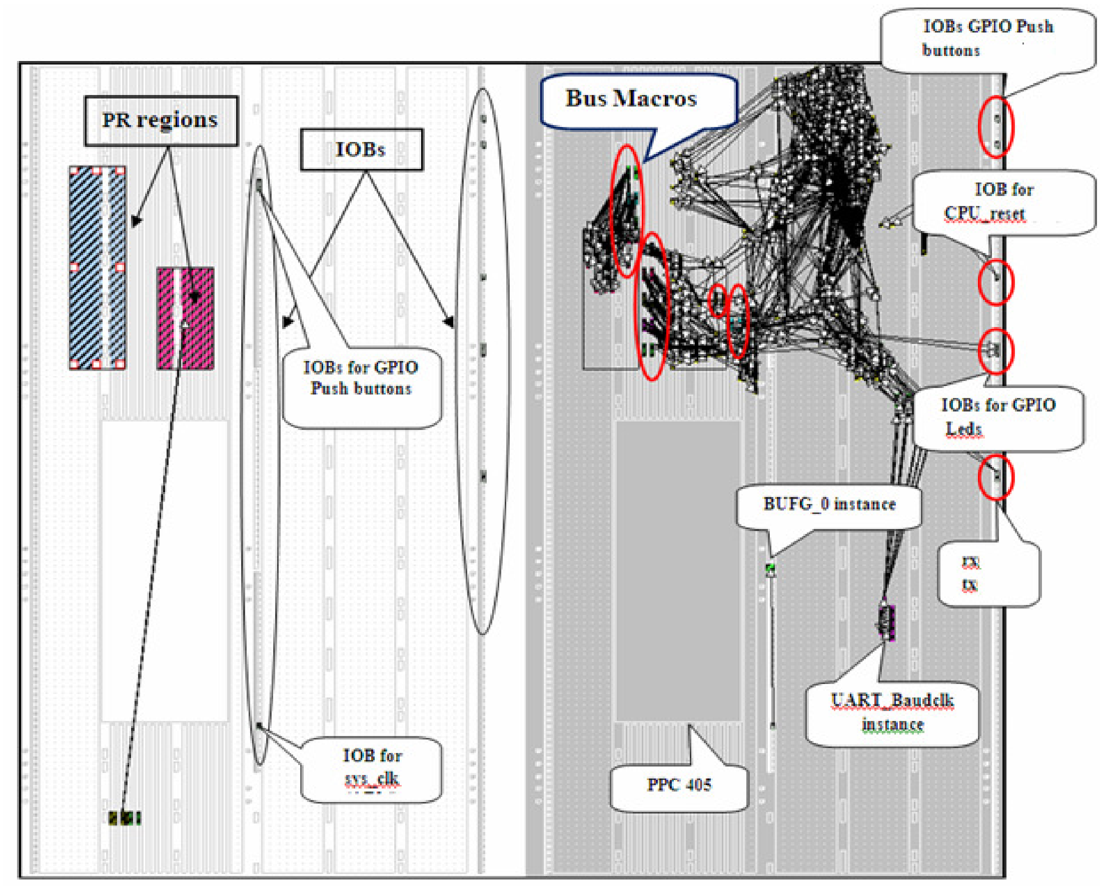


Figure 6. Floorplan and placement of top-level design on Virtex-4 FPGA chip

5. CONCLUDING REMARKS

In this paper we have addressed the swapping of modules in and out of the device on-the-fly and a user interface required to interact with the design implemented at run-time. Different configurations are swapped in or out of the device successfully. The future scope of this work includes implementation of partial reconfiguration technique on smart real-time applications like real-time tracking applications, wireless applications such as software defined radio, and in the development of intelligent robotic systems.

ACKNOWLEDGEMENTS

This work is under the project “Design and development of system level reconfiguration techniques for reconfigurable computing systems” sponsored by Department of Information Technology, Ministry of Communications and Information Technology, Govt. of India, New Delhi.
We would like to thank Dr. Chandrashekhar, Director, Central Electronics Engineering Research Institute (CEERI), Pilani, and Dr. P. Bhanu Prasad Group Leader, Digital Systems Group, CEERI, Pilani for their encouragement and support.

REFERENCES

[1] Christophe Bobda "Introduction to Reconfigurable Computing: Architectures, Algorithms and Applications" Springer 2007.

[2] Two Flows for Partial Reconfiguration: Module Based or Difference Based, Xilinx website http://www.xilinx.com/support/documentation/application_notes/xapp290.pdf,http://www.xilinx.com/itp/xilinx7/books/data/docs/dev/dev0038_8.html

[3] Sedcole, P. Blodget, B. Anderson, J. Lysaghi, P. Becker, T. " Modular partial reconfigurable in Virtex FPGAs," International Conference on Field Programmable Logic and Applications, 24-26 Aug. 2005 p.p.: 211- 216, ISBN: 0-7803-9362-7

[4] Emi Eto, "Difference-Based Partial Reconfiguration", XAPP290 (v2.0) December 3, 2007.

[5] Early Access Partial reconfiguration User Guide, UG208, Xilinx website, http://www.xilinx.com

Ashutosh Gupta

Received B.E. (ECE) and M.Tech. in Vlsi Design & Embedded systems from Guru Jambheshwar University of Science & Technology, Hisar in 2006 and 2008, respectively. Presently working as a Project Assistant in Digital Systems Group, Central Electronic Engineering Research Institute, Pilani, India. His field of interest is FPGA based Reconfigurable computing systems.

Dr. Kota Solomon Raju

Received B.E. (ECE) from SRKR Engineering College Bhimavaram, M.E. from Birla Institute of Technology & Science Pilani and Ph.D. from Department of Electronics and Computer Engineering, Indian Institute of Technology, Roorkee. Presently, he is working as senior scientist in Digital Systems Group, Central Electronic Engineering Research Institute, Pilani, India. He is working in the field of reconfigurable computing systems for communication, image processing and intelligent smart sensor systems.