Dynamic Partial Reconfiguration in Low-Cost FPGAs

K.Bhuvaneswari,V.Srinivasa Rao

Abstract— Field Programmable Gate Array (FPGA) market is growing rapidly with various applications in different industries. There is a new concept evolving in FPGA industry called dynamic partial reconfiguration (DPR) with has a greater exposure in different applications. Partial reconfiguration is nothing but reconfiguring the selected areas of an FPGA after its initial configuration at runtime. In this paper we reconfigure some specific region of the FPGA with a new functionality at runtime while the remaining areas remain static during this time. The complexities during the runtime can be simplified by a tool called PLANAHEAD which was introduced by Xilinx that is able to implement run time reconfigurable systems for all VIRTEX FPGAs. This results in low computational cost and low power FPGAs, PLANAHEAD is the first graphical environment for partial reconfiguration. In this context partial reconfiguration gives the flexibility for reducing the board space (effective utilization of resources), change a design in the field and also reduces the power consumption.

Index Terms— FPGA, Reconfigurable Modules,Partial Reconfiguration,Reconfigurable Modules,Dynamic Reconfigurability,Urdhva Tiryagbhyam Sutra, PlanAhead,ChipScope.

1 INTRODUCTION

FPGAs provide users with an environment where the user quickly develop and implement a circuit or module at low cost and fast turnaround time. [1] FPGA hardware allow the reconfiguration of a programmable logic partial reconfiguration [2] can also be used to allow larger complex designs to be implemented on devices with fewer resources than would normally be required for a complete implementation. The most common method for implementation of partial reconfiguration is one in which a modular design approach is used. The logic is configured in a manner such that one module is dynamically reconfigured [5] while another module is retained in a static configuration for performing operations which do not change. Partial reconfiguration is the practice of reprogramming only a portion of an FPGA.Specifically,dynamic partial reconfiguration denotes the ability to reprogram a portion of a circuit while it is operating. This is done without a need for the chip to power down or be reset. Partial reconfiguration can be implemented through the use of the Xilinx ISE tools in conjunction with PlanAhead software [8]. Partial reconfiguration generally requires the use of a modular design flow. Modular design [4] requires additional procedures for synthesis and implementation and adds complexity not found when utilizing the basic design flow. In general, modular design allows for simultaneous development of different modules which together complete the design of an FPGA.

The modular design flow consists of two phases. PlanAhead's main purpose is to customize the way circuits designed in ISE are laid out on the FPGA as well as providing timing and placement analysis to improve circuit function. By using this tool users can group circuits and modules. The benefit of this is that if each module is in its own area, the task of partial reconfiguration becomes much easier as only one area is being programmed and will not affect any of theother sections. This task is known as floor planning.PlanAhead also provides a useful set of design rule checks which can be used to improve designs and provide suggestions to the designer.

2 DYNAMIC PARTIAL RECONFIGURATION

Partial reconfiguration [4] is of two types namely dynamic partial reconfiguration and static partial reconfiguration. In this paper we concentrate on the dynamic partial reconfiguration process. Dynamic partial reconfiguration is also known as active partial reconfiguration, permits to change a part of the device while the rest of an FPGA is still running. In other words, while the partial data is sent into the FPGA,the rest of the device will be still running and the new data is being configured into FPGA.

There are two styles of dynamic partial reconfiguration[4] named as Difference based partial

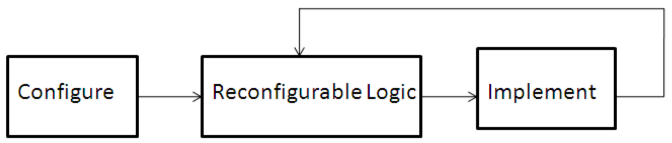


Fig 1. Dynamic Partial Reconfiguration

reconfiguration and Module based partial reconfiguration.Difference based partial reconfiguration can be used when a small change is made to the design.It is especially useful in case of changing Look-Up Table(LUT) equations or dedicated memory blocks content.Module based partial reconfiguration uses modular design concepts to reconfigure large blocks of logic.

3 ARCHITECTURE OF EA PR

The dynamic partial reconfiguration is supported on all types of Virtex series device.Early Access Partial Reconfiguration[10] is the latest design method used for partial reconfiguration.The EAPR allows signals in the base design to cross through a partially reconfigurable region without the use of a busmacro.This improves timing performance and simplifies the process of building a PR design.

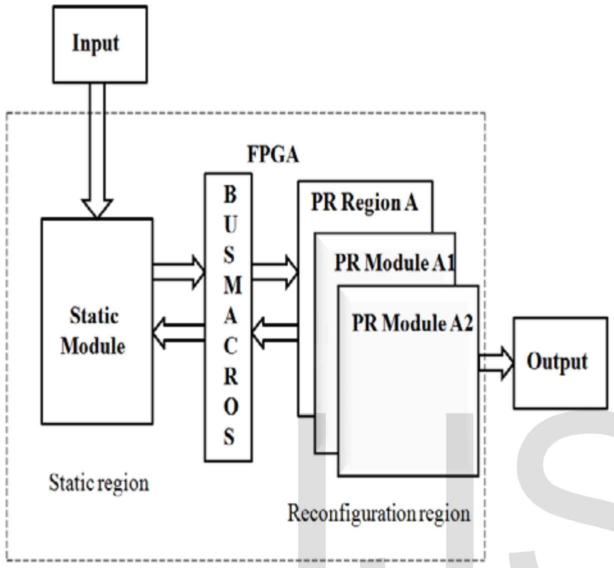


Fig 2. Architecture of EAPR design

Static module is the design remains in operation during the PR process.Partial Reconfiguration Module(PRM)[6] can be swapped in and out of the device.Multiple PRMs can be defined for a specific region.Partial Reconfiguration Region(PRR) is the part of the FPGA that is set aside for partial reconfigurable modules.More than one PRRs can be set for reconfiguration.Here a Proxy LUT is used which automatically connects the reconfigurable modules to the static module.This was possible in Virtex4 and above series of devices.There is no need to use busmacros for the purpose of connecting the modules which is the greatest advantage.The EAPR design flow can be implemented using PlanAhead[8].

4 EA PR DESIGN FLOW USING PLAN AHEAD

The EAPR design flow is shown in the fig 3.This can be implemented using PlanAhead tool[8].

i) HDL design description : The HDL design description for the top module,static module and the reconfigurable module are implemented here.The top module doesnot consist of any logic.It contains only I/O instantiations,static modules,clock signals etc.The static module is nothing but the base module which remains in the stable state during the partial reconfiguration state.The HDL description for the reconfigurable modules. The HDL description for the reconfigurable modules is

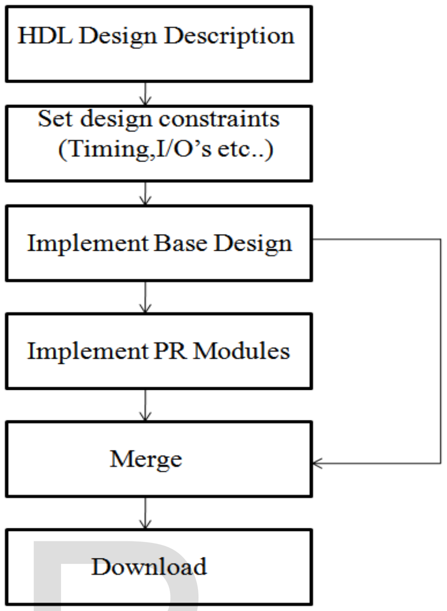


Fig 3.EAPR design flow

is implemented.These make use of the clock signals from the top level modules.

ii) Set design constraints : The area group,reconfiguration mode,timing constraint and location constraints are defined here inorder to place the modules on the FPGA area.

iii) Implement base module : The static modules will be created.It is nothing but a black box module which doesn't consist of any logic.The constraints files will be checked whether it has been properly created.

iv) Implement PR modules : There are various steps to be performed for creating a PR module

- Create a Reconfigurable Partition(RP).
- Add a Reconfigurable Module(RM)
- Define P block ranges for the reconfigurable partitions.
- Run pre-specific DRC checks.
- Implement configurations.
- Verify the configuration.

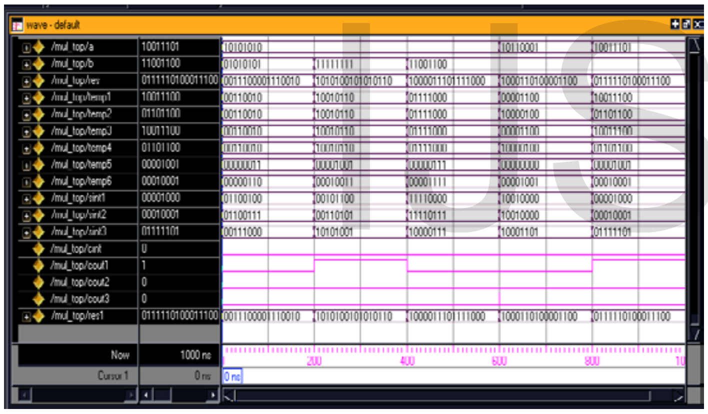
The PR modules can be implemented using the PlanAhead tool.Partitions will be created for each and every module and the p block ranges will be defined.In this paper two reconfigurable modules such as an array multiplier and a Vedic multiplier being reconfigured.The partitions and the pblock ranges will be created for these modules using the PlanAhead tool.The DRC checks will be performed and the configuration will be verified.

v) Merge : As soon as the configurations have been checked, all the base and the PR modules will be merged in order to complete a design. Furtherly many partial bit streams for each PRM and initial full bit streams are created to configure the FPGA. These partial bit files will be further downloaded onto the FPGA.

5 RECONFIGURABLE MODULES

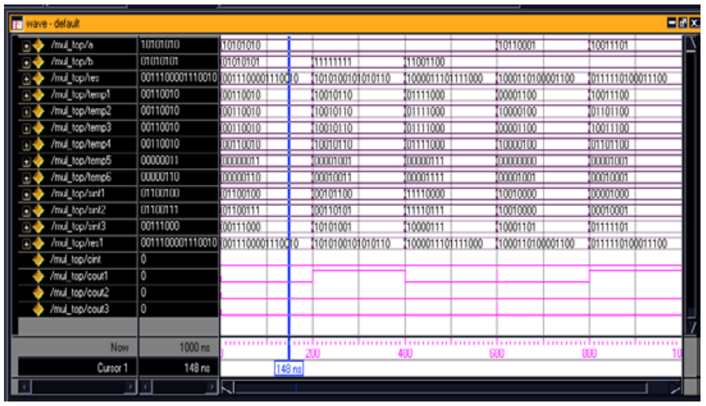
In this paper two different modules have been taken and they are reconfigured on the FPGA at runtime. The effective utilization of the resources will be taken place due to this runtime reconfiguration process. The work goes on replacing the functionality of one module with the functionality of another module at runtime. So any kind of module can be reconfigured with a different functionality on FPGA dynamically. Reconfiguration time can be greatly reduced.The module Array multiplier [12]will be reconfigured with Vedic multiplier.

Fig 4 and Fig 5 shows the simulated results for the Array and Vedic multipliers.The functionality is verified using the Modelsim 6.2c.



```
wave - default
/mul_top/a       10011101          10101010                                                 10110001          10011101
/mul_top/b       11001100          01010101          11111111          11001100
/mul_top/res     0111110100011100  0011100001110010  1010100101010110  1000011101111000  1000110100001100  0111110100011100
/mul_top/temp1   10011100          00110010          10010110          01111000          00001100          10011100
/mul_top/temp2   01101100          00110010          10010110          01111000          10000100          01101100
/mul_top/temp3   10011100          00110010          10010110          01111000          00001100          10011100
/mul_top/temp4   01101100          00110010          10010110          01111000          10000100          01101100
/mul_top/temp5   00001001          00000011          00001001          00000111          00000000          00001001
/mul_top/temp6   00010001          00000110          00010011          00001111          00001001          00010001
/mul_top/sint1   00001000          01100100          00101100          11110000          10010000          00001000
/mul_top/sint2   00010001          01100111          00110101          11110111          10010000          00010001
/mul_top/sint3   01111101          00111000          10101001          10000111          10001101          01111101
/mul_top/cint    0
/mul_top/cout1   1
/mul_top/cout2   0
/mul_top/cout3   0
/mul_top/res1    0111110100011100  0011100001110010  1010100101010110  1000011101111000  1000110100001100  0111110100011100
Now              1000 ns           200   400   600   800   10
Cursor 1         0 ns              0 ns
```

Fig 4. Simulation result for 8×8 Array type of Vedic multiplier



```
wave - default
/mul_top/a       10101010          10101010                                                 10110001          10011101
/mul_top/b       01010101          01010101          11111111          11001100
/mul_top/res     0011100001110010  0011100001110010  1010100101010110  1000011101111000  1000110100001100  0111110100011100
/mul_top/temp1   00110010          00110010          10010110          01111000          00001100          10011100
/mul_top/temp2   00110010          00110010          10010110          01111000          10000100          01101100
/mul_top/temp3   00110010          00110010          10010110          01111000          00001100          10011100
/mul_top/temp4   00110010          00110010          10010110          01111000          10000100          01101100
/mul_top/temp5   00000011          00000011          00001001          00000111          00000000          00001001
/mul_top/temp6   00000110          00000110          00010011          00001111          00001001          00010001
/mul_top/sint1   01100100          01100100          00101100          11110000          10010000          00001000
/mul_top/sint2   01100111          01100111          00110101          11110111          10010000          00010001
/mul_top/sint3   00111000          00111000          10101001          10000111          10001101          01111101
/mul_top/res1    0011100001110010  0011100001110010  1010100101010110  1000011101111000  1000110100001100  0111110100011100
/mul_top/cint    0
/mul_top/cout1   0
/mul_top/cout2   0
/mul_top/cout3   0
Now              1000 ns           200   400   600   800   10
Cursor 1         148 ns            148 ns
```

Fig 5. Simulation result for 8×8 Vedic multiplier using Urdhva Tiryaghbhyam sutra

Fig 6 shows the synthesized RTL schematic of the proposed multiplier module-8×8 Vedic multiplier using Urdhva Tiryagbhyam Sutra[11].The synthesis ptocess was performed in Xilinx ISE 9.2i.

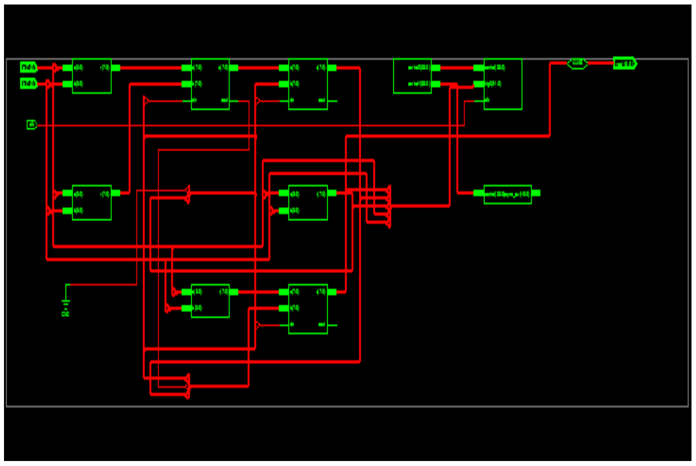
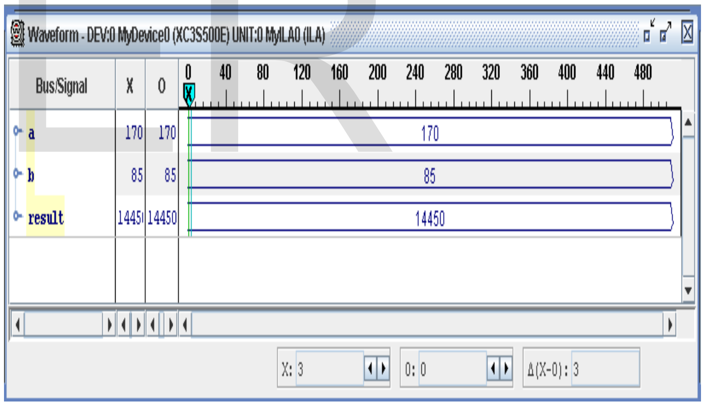


Fig 6.RTL Schematic of 8×8 Vedic multiplier

The functionality of the two modules are verified and they are implemented on the FPGA at runtime using ChipScope Pro 9.2i.Fig 7 shows the runtime output view in chipscope.



```
Waveform - DEV:0 MyDevice0 (XC3S500E) UNIT:0 MyILA0 (ILA)
Bus/Signal   X      O      0   40   80   120   160   200   240   280   320   360   400   440   480
a            170    170    170
b            85     85     85
result       1445   14450  14450
X: 3    O: 0    Δ(X-O): 3
```

Fig 7. Runtime output view in chipscope

Thus the proposed reconfigurable module-8×8 Vedic multiplier using Urdhva Tiryagbhyam Sutra achieves a highest perfomace over the array multiplier module.The two modules are implemented using the Spartan 3E FPGA.The difference in the delay and system performace are clearly shown in the Fig 8..It shows that the resource have been effectively utilized thus the system performance.

DEVICE UTILIZATION	**ARRAY TYPE OF 8×8 VEDIC MULTIPLIER**	**8×8 VEDIC MULTIPLIER USING URDHVA TIRYAGBHYAM SUTRA**
Selected Device	3s500efg320-5	3s500efg320-5
Number of slices	90 out of 4656 1%	91 out of 4656 1%
Number of 4 input LUTs	157 out of 9312 1%	161 out of 9312 1%
Number of IOs	17	17
Number of bonded IOBs	17 out of 232 7%	18 out of 232 7%
Delay	22.995ns at 43.48MHz	20.477ns at 48.83MHz

Fig 8 . Comparison between multipliers

6. CONCLUSION

In this manner, we showed that the design flow methodology EA PR have the clear advantage over the existing FPGA design methodology. This methodology is also applicable to detect some kind of problems in the FPGA architecture with the help of redundancy at the runtime. Partial reconfiguration begins new advantages to encryption implementation with the use of an FPGA.

REFERENCES

[1] Krzysztof Jozwik, Hiroyuki Tomiyama, Masato Edahiro, Shinya Honda and Hiroaki Takada, “comparison of Preemption Schemes For PArtially Reconfigurable FPGAS”, IEEE embedded systems letters, vol.4, no.2, June 2012.

[2] Kyprianos Papadimitriou, Student Member, IEEE, Tonis Anyfantis and Apostolos Dollas, Senior Member, IEEE, “An Effective Framework to Evaluate Dynamic Partial Reconfiguration in FPGA Systems”, IEEE transactions on instrumentation and measurement, vol. 59, no.6, June 2010.

[3] Zine EL Abidine Alaoui Ismailia ns Ahmedmoussa, “Self-partial and Dynamic Reconfiguration Implementation for AES using FPGA”, IJCSI International Journal of Computer Science Issues, Vol. 2, 2009.

[4] Wang Lie, Wu Feng-yan, “Dynamic Partial Reconfiguration in FPGAs”, Third International Symposium on Intelligent Information Technology Application,vol.2, 2009 IEEE.

[5] Eric J.Mcdonald, “ Runtime FPGA Partial Reconfiguration”, IEEE A&E systems Magazine, July 2008.

[6] S.S.Shriramwar and Kartik Ingole,”Partial Reconfiguration for Signal Processing Application using FPGA”,International Journal ofAdvances in Engineering &Technology,Vol.3,Issue 2, pp. 680-684,May 2012.

[7] Matthew G.Paris,”Optimizing Dynamic Logic Realizations for Partial Reconfiguration of FieldProgrammableGateArrays”,B.S.University of Louisville,2008.

[8] Xilinx,Inc.Partial Reconfiguration Tutorial PlanAhead Design Tool,UG743(V14.1) May 8,2012.

[9] Xilinx,Inc.Overview of PartialReconfiguration Flow,PlanAhead software tutorial http://www.xilinx.com.UG743(v12.2)July23,2010

[10] Xilinx,Inc.UG208,Early Access Partial Reconfiguration userguidehttp://www.xilin.com.2006.

[11] Swami Bharati Krishna Tirtha, Vedic Mathematics. Delhi: Motilal Banarsidass Publishers, 1965.

[12] Ming-Chen Wen,Sying-Jyan Wang, and Yen-Nan Lin,”Low Power Parallel Multiplier with Column By-passing”Electronics letters,vol.41,Issue Page(s):581-583, 10-12 May 2005.

[13] Kamboh,Hamid M,Khan Shoab A,”FPGA Implementation of fast adder”,7th International Conference on Computing and Convergence Technology(ICCCT), ,Page(s):1324-1327, IEEE,2012.

[14] Xilinx Inc,Chipscope Pro Software and Cores userguidewww.xilinx.com/chipscope_pro_sw_cores_ ug029.pdf, UG029 (v9.2) May 30, 2007.