DOI: 10.15415/jtmge.2010.11006

Journal of Technology Management for Growing Economies

Volume 1 Number 1 April 2010

Chip Architecture for Data Sorting Using Recursive Algorithm

Megha Agarwal
Indra Gupta
Indian Institute of Technology, Roorkee,Uttarakhand

Chip Architecture for Data Sorting Using Recursive Algorithm

Megha Agarwal
Indra Gupta
Indian Institute of Technology, Roorkee,Uttarakhand

Abstract

This paper suggests a way to implement recursive algorithm on hardware with an example of sorting of numeric data. Every recursive call/return needs a mechanism to store/restore parameters, local variables and return addresses respectively. Also a control sequence is needed to control the flow of execution as in case of recursive call and recursive return. The number of states required for the execution of a recursion in hardware can be reduced compared with software. This paper describes all the details that are required to implement recursive algorithm in hardware. For implementation all the entities are designed using VHDL and are synthesized, configured on Spartan-2 XC2S200-5PQ208.

Keywords: Binary search tree, Field programmable gate arrays (FPGA), Recursive Algorithms, Very high-speed integrated circuits hardware description language (VHDL).

INTRODUCTION

Recursion is an approach to write algorithms for repetitive tasks. In recursion the whole problem is decomposed into smaller sub-problems which are exactly similar to the original problem. Recursion can be efficiently used in optimization problems, non-linear data structures algorithms like binary trees, searching in dictionary, recursive filter, data compression etc. Many examples demonstrate advantages of recursion (Lipschutz, 2002). However, this technique is not always appropriate, particularly when a clear efficient iterative solution exists (Sklyarov, 2005). This is primarily due to the large amount of states that are accumulated during deep recursive calls because at each level of recursive call the current values of the parameters, local variables, return addresses etc of the subprogram are pushed on the stack until the subprogram is completed and these allocation records are popped out when the subprogram is reactivated (Lipschutz, 2002). This also consumes time to execute. But this execution time part can be taken care of by implementing the developed recursive algorithm in hardware e.g. FPGA.

If any high-level language which admits recursion is used then computer keeps the track of all the values of the parameters, local variables and return addresses. But if a high-level languages which does not admits recursion is used then a recursive procedure must be translated into a non-recursive procedure at the programmer hand. For example VHDL does not

Journal of Technology Management for Growing Economies
Vol. 1 No. 1, April 2010
pp.93-102

CHITKARA UNIVERSITY

©2010 by Chitkara University. All Rights Reserved.

support for recursion. So, the algorithm is required to be converted into non-recursive algorithm. By combining the activation of a recursive subsequence of operations with the execution of the operations that are required by the respective algorithm, the recursion can be implemented in hardware much more efficiently (Sklyarov, 2004). The same event takes place when any recursive sub-sequence is being terminated, i.e. when control has to be returned to the point which is just after the last recursive call and an operation of the executing algorithm that follows the last recursive call has to be activated.

The results obtained for some known methods for implementing recursive calls in hardware, have shown FPGA circuits to be significantly faster than software programs executing on general-purpose computers.

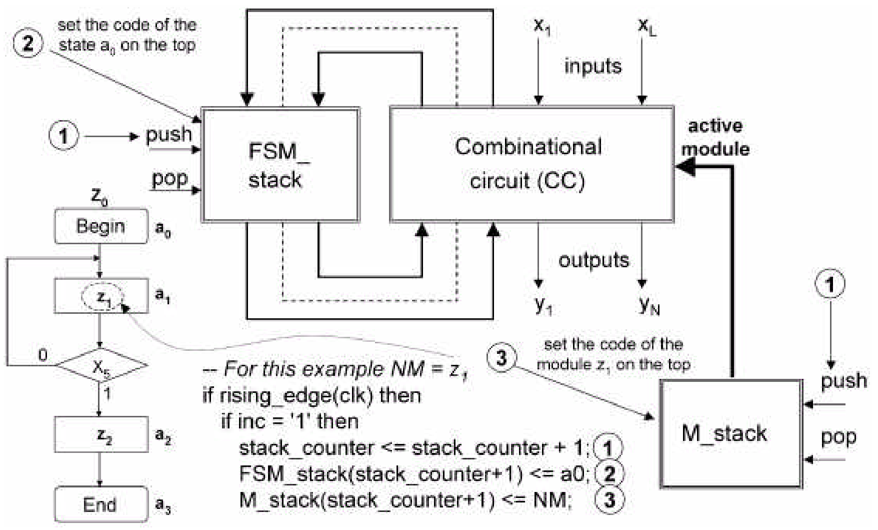
Recursion shows better performance in case of non-linear data structures problems (Ninos and Dollas, 2008). For example a binary search tree can be constructed and used for sorting various types of data. In order to build such tree for a given set of values, the appropriate place for each incoming node in the current tree should be found out. Afterwards in order to sort the data, a special technique using forward and backtracking propagation steps that are exactly the same for each node is required. Thus, a recursive procedure is very efficient in this area. Other application to implement recursive algorithm in hardware may be in the field of lossless data compression such as Huffman coding, Dictionary search tree, recursive filter etc (Sklyarov et al., 2005).

The design of data sorting are coded with VHDL (Roth, 2001), synthesized and configured onto Spartan-2 XC2S200-5Q208 FPGA, from Xilinx family (www.xilinx.com). The concept behind sorting is discussed in section 2 and results of synthesis and simulation are discussed in section 3. Here Modelsim 6.0d is used as simulation tool and ISE 8.1i project navigator is used as synthesis tool.

IMPLEMENTATION OF SORTING ALGORITHM

To develop the sorting algorithm firstly a binary search tree should be constructed and after that inorder traversal of that binary search tree provides sorted data. Both of theses functions can be effectively illustrated by recursive procedure. The concept used for the design is to divide the algorithm to a discrete number of modules (labeled zi, i.e. z1, z2 etc), each of which will have a number of discrete states (labeled ai, i.e. a1, a2 etc). Two separate stacks are used one to store the current module executed (the M_stack), one to store the current state of the current module (the FSM_stack). So, this needs support of control and execution mechanism. Control Unit is to transfer the control among the different modules and states so it needs to store and restore data from the different stacks (Sklyarov, 2005).

Figure 1 demonstrates the function of Recursive hierarchical finite state machine (RHFSM) which works as the Control Unit. This unit is having two stacks and one combinational circuit (CC). Functions of FSM_stack and M_stack are as discussed before. A combinational circuit is connected to the stacks and operates on the inputs (some of which are receives from the stacks) and produces the appropriate outputs (Sklyarov, 2004). It is worth noting that both the stacks (M_stack and FSM_stack) use the same stack pointer.


2 set the code of the state a0 on the top
1 push
pop
FSM_ stack
Combinational circuit (CC)
x1
xL
inputs
active module
outputs
y1
yN
z0
Begin a0
z1 a1
0
X5
1
z2 a2
End a3
3 set the code of the module z1 on the top
M_stack
1
push
pop

```
-- For this example NM = z1
if rising_edge(clk) then
  if inc = '1' then
    stack_counter <= stack_counter + 1; 1
    FSM_stack(stack_counter+1) <= a0; 2
    M_stack(stack_counter+1) <= NM;  3
```

Figure 1: Control Unit (RHFSM) demonstrating a new module z1 invocation

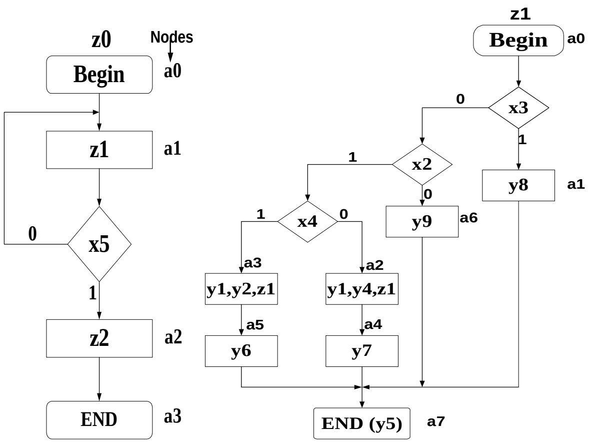


Figure 2: Modular representation of module z0, z1

Complete problem of sorting is decomposed into three modules z0, z1, z2. In Figure 2 module z0 is the main module and z1 is the module to construct binary search tree (Sklyarov, 1999). Module z0 is invoking the module z1 in the state a1.

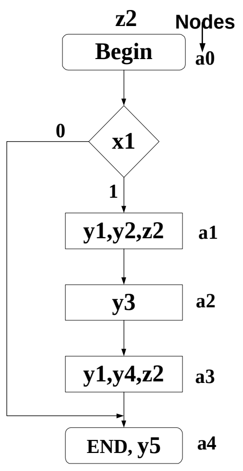


Figure 3: Modular representation of module z2

Figure 3 shows module z2. z2 accepts the constructed binary search tree and gives the sorted data. z2 is invoked by the a2 state of module z0 (Figure 2). Module z1 and z2 are recursive because these contain self reference. Every module has its states (ai), its input (xi) and output (yi). Each module begins with state a0, which is the Begin state, and ends with the End state, which is labeled according to the module's number of states (in module z0 it's a3).

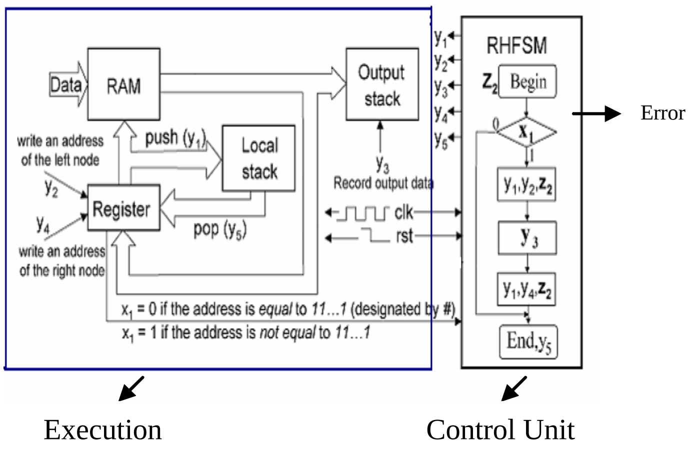

Data
RAM
Output stack
push (y1)
Local stack
write an address of the left node
y2
Register
y4
pop (y5)
write an address of the right node
y3
Record output data
clk
rst
x1 = 0 if the address is equal to 11...1 (designated by #)
x1 = 1 if the address is not equal to 11...1
y1
y2
y3
y4
y5
RHFSM
Z2
Begin
0
x1
1
y1,y2,z2
y3
y1,y4,z2
End,y5
Error
Execution
Control Unit

Figure 4: A composition on Control Unit and Execution Unit

Figure 4 gives the complete circuitry of implementation of sorting algorithm on hardware. In Figure 4 RHFSM works as the Control Unit and the rest circuitry works as Execution Unit (Sklyarov and Skliarova, 2006). The function of the Figure 4 is as follows; at some state the module produces the outputs (y1, y2, etc) that are inputs to the Execution Unit. At every decision point, the module's inputs (xi) are the outputs of the Execution Unit. Every time a new module is called from another module (i.e. module z2 from module z0) then the common stack pointer is incremented by one, the new module is saved at the M_stack, the new Begin state is also saved at the FSM_stack and the new module starts execution (Sklyarove, et al. (nd)). Every time a module state is changed, the new state is stored at the FSM_stack, overwriting the previous state stored at the same location. Using this concept, the recursive algorithm can be easily described in hardware, as it is easy for the circuit to determine new states, modules and recursive calls using the stacks.

RESULTS AND DISCUSSION

In this paper sorting algorithm is considered for the hardware implementation of recursive algorithms. Two different data sizes namely 12 and 6 are considered and the results are analyzed. Simulation results for the above two cases which verifies the functionality of complete design are given in Figure 5 and Figure 6 respectively.

For this purpose Modelsim 6.0d is used as simulation tool. Design is also verified by the test bench. After simulation for synthesis and evaluation of design ISE 8.1.i project navigator synthesis tool is used.

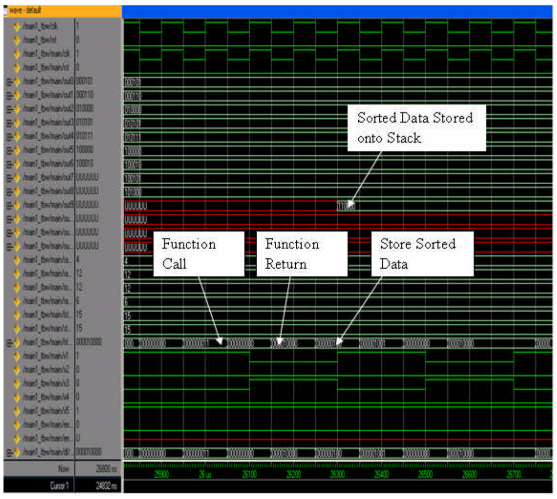


Figure 5: Simulation results with 12 data sets

Figure 5 is the simulation waveform obtained when the 12 dataset are given for data sorting. The waveforms show sorted output data and discrete steps for translation from recursive to non-recursive.

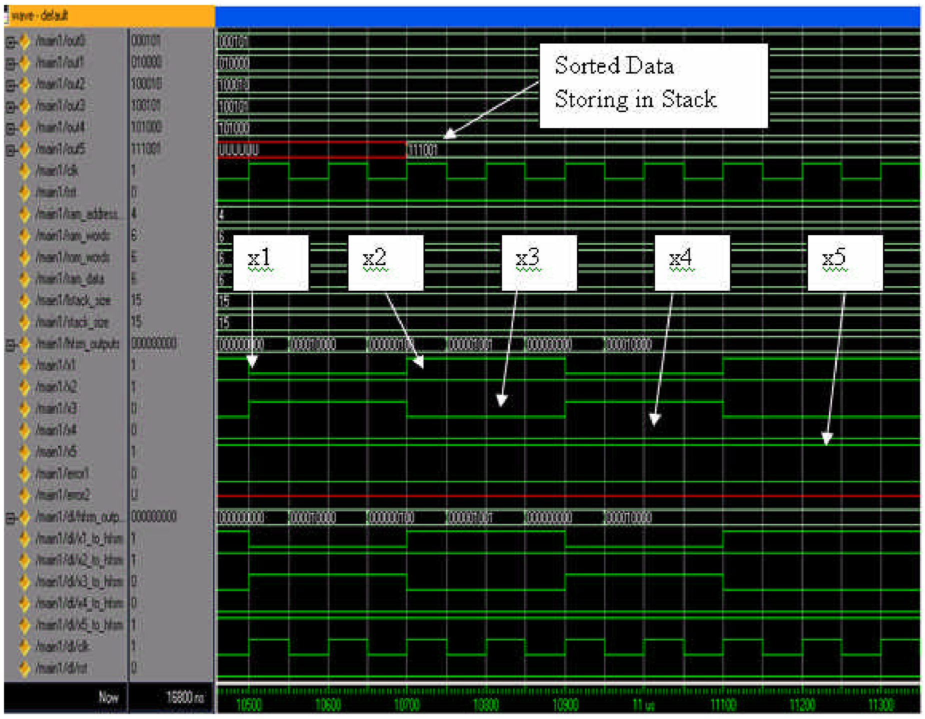


Figure 6: Simulation results with 6 data sets

Figure 6 shows the simulation waveform with 6 data sets. Waveforms showing change in outputs x1, x2,…., x5 of Execution Unit and storing of sorted data by Execution Unit.

After performing simulation on the data sets of length of 6 several times, the average simulation time is 10300 ns, similarly with the data set of length 12 the average simulation time is 23040 ns. For both the cases clock cycle of 100 ns is used. Hence here it can be observed that simulation time is approx doubled if the length of data set is doubled. Test bench of the main entity also gives the same result. So, the design is also verified by the test bench. Whenever the data is changed (for same number of data) the simulation time is also varied. Because the number of steps taken to get the sorted data varies with the change in the nature of data and number of steps affects the total number of clock cycles taken.

After the simulation the design is synthesized for Spartan-2 XC2S00-5PQ208 of Xilinx family using ISE8.1i. Synthesis process generates a flat netlist of HDL with the synthesis report and device utilization summary. Figure 7 shows the RTL view of design entity.

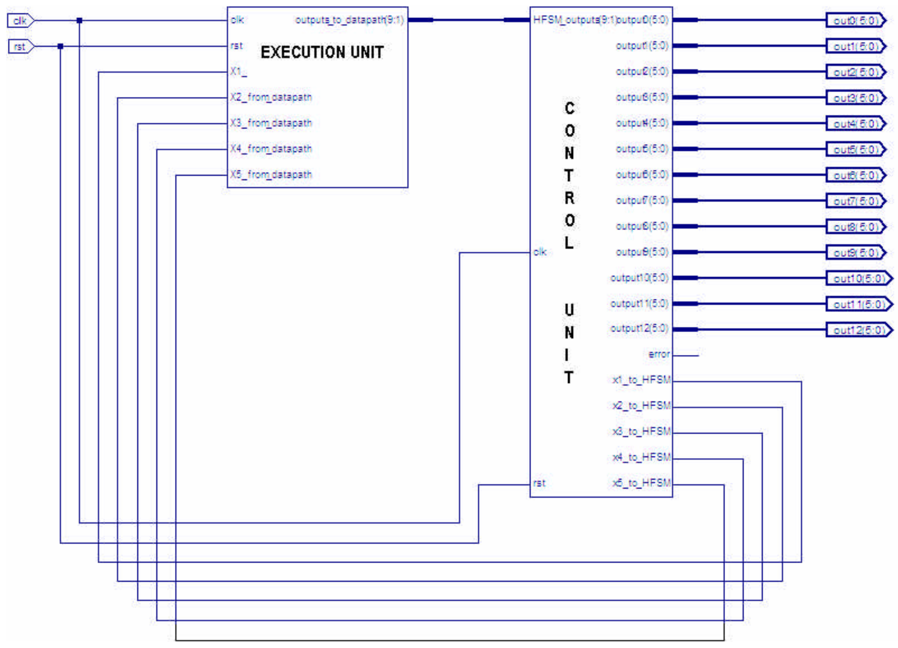
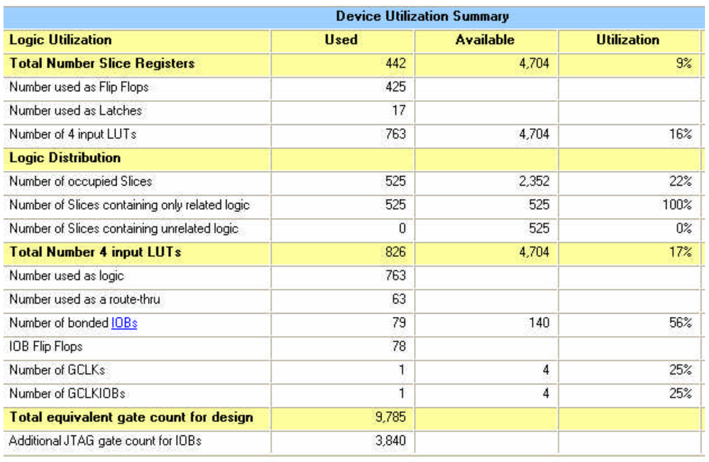


Figure 7: Top Level Schematic Diagram of Sorting Design

In Figure 7 clk, rst are the inputs port and out0, out1,……., out11 are the output ports to get the sorted output data. Error signal sets to show stack overflow. All other signals are the intermediate signals. Both the units work on the same clock and resets simultaneously.

The synthesis report for the design is generated in ISE8.1i. The synthesis report gives the details of hardware resources used and also timing analysis. Here synthesis report is partially presented. The design can operate with minimum period of 17.841ns (Maximum Frequency: 56.051MHz).

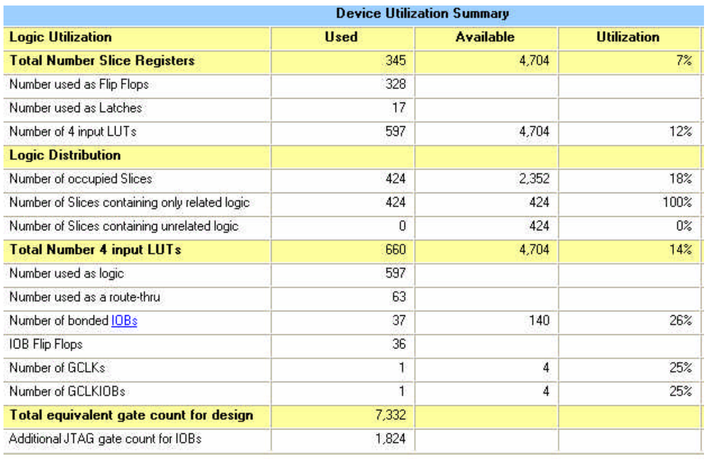
Figure 8 shows the device utilization report of data sorting design with 12 data. This report gives the information about total available hardware resources of the target device and used resources of the target device by the design. Total memory usage in this case is 110788 kilobytes.



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	442	4,704	9%
Number used as Flip Flops	425		
Number used as Latches	17		
Number of 4 input LUTs	763	4,704	16%
Logic Distribution			
Number of occupied Slices	525	2,352	22%
Number of Slices containing only related logic	525	525	100%
Number of Slices containing unrelated logic	0	525	0%
Total Number 4 input LUTs	826	4,704	17%
Number used as logic	763		
Number used as a route-thru	63		
Number of bonded IOBs	79	140	56%
IOB Flip Flops	78		
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%
Total equivalent gate count for design	9,785		
Additional JTAG gate count for IOBs	3,840		

Figure 8: Device Utilization with 12 Data Sets

Figure 8 shows that the numbers of slices occupied are 22% of total available slices and total gate count for the design is 9,785. Fig. 9 shows the device utilization for complete data sorting design with data size equal to 6. Total memory usage in this case is 108740 kilobytes.



Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	345	4,704	7%
Number used as Flip Flops	328		
Number used as Latches	17		
Number of 4 input LUTs	597	4,704	12%
Logic Distribution			
Number of occupied Slices	424	2,352	18%
Number of Slices containing only related logic	424	424	100%
Number of Slices containing unrelated logic	0	424	0%
Total Number 4 input LUTs	660	4,704	14%
Number used as logic	597		
Number used as a route-thru	63		
Number of bonded IOBs	37	140	26%
IOB Flip Flops	36		
Number of GCLKs	1	4	25%
Number of GCLKIOBs	1	4	25%
Total equivalent gate count for design	7,332		
Additional JTAG gate count for IOBs	1,824		

Figure 9: Device Utilization with 6 data sets

Figure 9 shows that the total numbers of slices occupied by the design are 18% of total available number of slices and total gate count for the design is 7,332. This shows that when the data is increased by 100% then there is only 33.45% increment in total gate count. Similarly occupied slices are incremented by 22.22%.

In both the cases hardware resources used are less than the available hardware resources on the Spartan- 2 so, synthesized designs are configured on Spartan-2 FPGA.

The total hardware resources should neither be very much less nor more. It should be optimum. Because if we are configuring only a single design onto the FPGA and the design is coded into VHDL such as it consumes very less hardware resources then the rest of the available hardware resources of target device are wasted. But in case if multiple design entries are configured on the same FPGA and each design it taking more hardware resources then it will not be possible to configure all the designs on FPGA. Further also there is a memory and time trade off. So all these constraints must be taken into consideration while designing any entity.

CONCLUSIONS

Recursive algorithms permit complex solutions to be specified in relatively simple manner. Although recursion takes more time to get executed but it can be reduced by implementing on recursion on hardware. FPGAs do not use operating system; minimize reliability concerns with true parallel executions and deterministic hardware dedicated to every task.

This paper covers a way to implement recursive algorithm on hardware with an example of sorting of numeric data. The complete unit is been designed in VHDL and verified by the testbench in Modelsim 6.0d and is implemented in an FPGA of the Spartan-2E family with ISE 8.0i project navigator. It is also verified that when the data is increased by 100% then there is only 33.45% increment in total gate count. Similarly occupied slices are incremented by 22.22%. Because there is time-memory tradeoff so, the model should be designed such as the hardware resources used by the design should be optimum.

All the modules developed for the discrete steps for the translation of recursive procedure to non-recursive procedures are reusable for different sizes of the same problem. Further the design can be made dynamic and accurate timing analysis can be performed by logic analyzer. Further work can be done in many more application involving recursive algorithms such as data compression and optimization problems.

REFERENCES

http://www.xilinx.com (Last viewed on 27/6/09)

Lipschutz, S. (2002) *Theory and problems of data structures,* New Delhi, Tata McGraw-Hill.

Ninos, S. and Dollas, A. (2008) ‘Modelling recursion data structures for FPGA-based implementation’, *paper presented at International Conference on Field- Programmable Logic and Applications,* September.

Roth, C.H. (2001) *Digital system design using VHDL* (3rd edn.), PWS publishing company.

Sklyarov, V. (1999) 'Hierarchical finite-state machines and their use for digital control', *IEEE Transactions on VLSI Systems,* 7: 2, 222–228.

Sklyarov, V. (2004) 'FPGA-based implementation of recursive algorithm', *Microprocessors and Microsystems,* 28, 197-211.

Sklyarov, V. (2005) 'Hardware implementation of hierarchical FSMs', *ACM International Conference Proceeding, Proceedings of the 4th international symposium on Information and communication technologies,* Cape Town, SA, p. 148–153.

Sklyarov, V. and Skliarova, I. (2006) 'Reconfigurable Hierarchical Finite State Machines', *Proceedings of the 3rd International Conference on Autonomous Robots and Agents, Palmerston North, p. 599-604.*

Sklyarov, V., Skliarova, I. and Pimentel, B. (2005) 'FPGA- Based implementation and comparison of recursive and iterative algorithms', *Proceedings of the 15th International Conference on Field- Programmable Logic and Application,* Finland, p. 235-240.

Sklyarove, V., Skliarova, I. and Ferrari, A.B. (nd) 'Hierarchical Specification and Implementation of Combinatorial Algorithms based on RHS Model' (online) (cited 27th June 2009). Available from http://www.ieeta.pt/~iouliia/Papers/2001/p13005.pdf**.**

Megha Agarwal is Sudent in Department of Electrical Engineering, Indian Institute of Technology, Roorkee, Uttarakhand, India.

Dr. Indra Gupta is Associate Professor in Department of Electrical Engineering, Indian Institute of Technology, Roorkee, Uttarakhand, India.

Chitkara University
Saraswati Kendra, Plot 11-12, Dainik Bhaskar Building
Sector 25-D, Chandigarh-160014, India
Email: journal@chitkarauniversity.edu.in
Website: www.chitkara.edu.in/journal/index.php