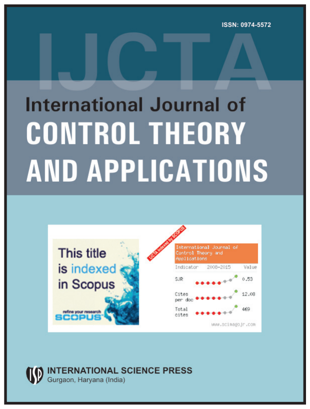


International Journal of Control Theory and Applications

ISSN : 0974-5572

© International Science Press

Volume 10 • Number 12 • 2017

DCAM: Discrete Control Based Automatic Management for Reconfiguring FPGA Applications

C. Subashini[a] and K. Senthil Kumar[b]

[a]*Research Scholar, Department of ECE, Dr. M.G.R Educational and Research Institute University, Chennai*
[b]*Professor, Department of ECE, Dr. M.G.R Educational and Research Institute University, Chennai*

Abstract: This paper focused on providing a general modeling framework consists of dynamic semi-reconfigurable hardware system based on FPGA. A control methodology is considered to model the entire behavior of the computing system to derive a control objective enforcement. Here the automatic computing of hardware level is lesser than the software level. To do this a discrete control model including labeled transition systems is investigated whereas the DC algorithm is used to generate enforcements for controlling the behavior of the system. Also it comprises of a set of reconfigurable areas which can be dynamically partially reconfigured to execute the tasks. The DC manager is encoded with respect to various constraints and multiple objectives like resource reuse, cost and time minimization. The entire framework model is simulated and experimented in Xilinx FPGA platform.

Keyword: Hardware Architecture, Discrete Control Algorithm, FPGA Platform, Hardware Programming, Reconfigurable FPGA.

1. INTRODUCTION TO RECONFIGURING FPGA

In spite of the fact that Gerald Estrin [1-3] had considered reconfigurable computing as early as 1960, the generally late advancements in reconfigurable computing have been filled by the accessibility of logic devices that can be rapidly and effectively programmed and reprogrammed to perform a large variety of functions. The primary devices of this type that attained sufficient density to perform large portions of a computation and that had critical accessibility are described as field-programmable door exhibits (FPGAs). These chips give the planner with arrays of simple logic functions and memories, (for example, flip-flops) that can be associated through programmable interconnection systems. In any case, the early FPGA gadgets from Xilinx, Altera, and others gave moderately little logic, yet later generations gave sufficient logic to analysts to consider what may be conceivable through direct implementation of computational algorithms in reconfigurable logic devices. The densities of today's FPGAs have surpassed 150,000 4-input lookup tables (LUTs) per gadget and some have formed into gadgets that can be utilized to construct complete systems on a programmable chip (SoPC), giving such specific elements as multi-gigabit serial I/O, embedded microprocessors, digital signal processing (DSP) blocks and embedded SRAM blocks of different sizes.

The objective of this section is to provide a short overview of what reconfigurable logic is and few important forms of reconfigurable logic which have been utilized in reconfigurable computing. The features of both fine- and coarse-grained reconfigurable logic devices are explored in this paper and also few significant examples of each are described.

2. FPGA AND RECONFIGURING FPGA

With their presentation in 1985, field-programmable gate arrays (FPGAs) have been an option for implementing digital logic in systems. In any case, FPGAs was utilized to give a denser answer for glue logic within systems, however now they have extended their applications to the point that it is normal to discover FPGAs as the central processing devices within systems. Contrasted with application-specific integrated circuits (ASICs) and mask-programmable gate arrays (MPGAs), FPGAs have a few preferences for their users, including: pre-tested silicon for use by the designer; quick time to market, being a standard product; no non-recurring engineering costs for fabrication; and re-programmability, permitting planners to overhaul or change logic through in-system programming. The design errors can be fixed and new features can be added or the function of the hardware can be entirely re-targeted to other applications by reconfiguring the device with a new circuit. Obviously, contrasted with ASICs and MPGAs, FPGAs are more expensive per chip to perform a particular function so they are bad for greatly high volumes. Likewise, an FPGA implementation of a function is found to be slower than the fixed silicon alternatives. After some time, however, the cost of doing custom silicon and the way that FPGAs now tend to utilize state-of-the-art CMOS processes imply that FPGAs are performing more of the functions that ASICs and MPGAs would have performed in numerous systems.

The complexity of contemporary FPGA devices and the several existing texts that describe the FPGA architecture and design [4-6] are considered in this section to provide an overview of FPGA architecture and features as an underlying basis for future discussions about how FPGAs are utilized in reconfigurable computing. In this overview, the fundamental FPGA architecture is examined, including basic routing, logic and input/output (I/O) structures. Also, some of the more advanced features will be introduced which have been integrated into FPGAs since the late 1990's, that have included embedded arithmetic logic, high-speed serial I/O, embedded memory and embedded microprocessors. The reconfigurable logic and microprocessors are tightly integrated while introduced from the perspective of FPGAs.

It is seen that, on programming architecture, a few research FPGA gadgets [7-10] have extensible configuration in a novel path. These FPGAs bolsters switching between multiple configuration states that are stored on the FPGA. This permits the FPGA circuit to change quickly while the information in the FPGA either stays in place or it is swapped itself in and out of the array, hence, emulating a much bigger FPGA. Once the multiple, on-chip configuration memories are stacked, this extensively diminishes the external programming bandwidth that is needed to context switch among FPGA configurations since switching between gadget context (i.e., singular gadget designs) is altogether done through accesses to on-chip configuration memory.

The configurability of fine-grained reconfigurable logic devices permits designers to specialize their equipment down to the bit level, implying that, if an application prerequisites 7-or 17-bit arithmetic for an operation, the equipment can directly indulge what is required. The configurability makes gadgets, for instance FPGAs, reasonable to execute a huge variety of functions directly. This configurability comes at a noteworthy cost in terms of circuit power, speed and area. Each level of configurability needs more buffering, routing, multiplexing and/or memory, hence, requiring more transistors and their interconnections.

In perceiving these costs, numerous specialists [11-20] have examined how to utilize arrays of coarse grained reconfigurable administrators as the basis for reconfigurable computing machines. Other than the potential benefits in terms of circuit power, speed and area, numerous specialists have sought after the utilization of coarse-grained reconfigurable arrays (CGRAs) with the trust that they would be easier targets for higher level development tools. Notwithstanding, as pointed out in Hartenstein's brief review of 19 various coarse-grained designs [21], the provocation with utilizing CGRAs is that there is no general form that is successful for all applications. To be more proficient, CGRAs are optimized in terms of administrators and routing for some particular issue domain or set of domains. In this way, an application must be well matched with the array to discern dramatic improvements over other reconfigurable solutions. For instance, an application that performs numerous 8-bit operations squanders a lot of logic, if the CGRA utilizes 32-bit ALUs and operators. Considering the substantial number of CGRAs, we will quickly portray only a few of eminent research architectures and some of the recent commercial architectures to represent some fascinating past and momentum cases of CGRAs.

3. SYSTEM OBJECTIVES

The main objectives of the proposed system are defined in terms of functional and non-functional requirements. There are two different kinds of objectives are considered here as optimal and logical control objectives. The logical objectives are focused on non-related to states whereas the optimal objectives are focused on related to states associated with optimal costs. The following objectives are defined as logical and optimal objectives. They are:

(i) Logical Control Objectives

1. Constraint on resource utilization
2. Memory sharing
3. Energy consumption
4. Setting sleep mode during execution of a task
5. Setting active mode whenever required
6. Constraint on power peak
7. Reachability

(ii) Optimal Control Objectives

1. Reducing the Power peak in hardware
2. Reducing the WCET of system execution
3. Reducing least amount of energy consumption of system execution

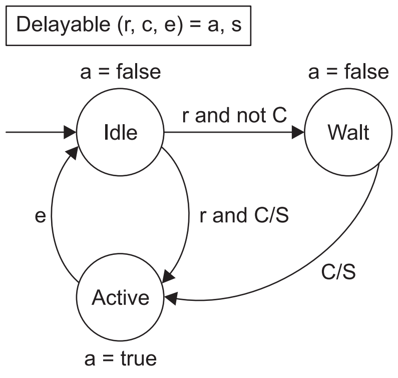
4. FPGA ARCHITECTURE

In order to do custom implementation to configure the hardware functions in FPGA is integrated with an array of cells and programmable channels to route. It is well known the fundamental cells in the FPGA are LUT is called as logic cells. LUTs are different size of memories utilized as devices for programming where it act as an intermediate device carry out various logic functions among various inputs and outputs. The state of a device between two sequence clock cycles is hold in D-Flip-Flop. Binary files are used to configure the basic unit LUT and switches in router, whereas the files comprises of a set of bit-streams generated by Xilinx Embedded Development Kit (XEDK). The tools available in this XEDK are used to design embedded architectures. In recent days the number of logic cells is huge for designing complex architectures. Large number of hardware accelerators and more number of processors are implemented to design multi-core architectures.

During the run time some of the features are reset/reassigned/reconfigured for configuring the hardware is called as Reconfigurable FPGA. It also can be obtained by certain logical functions are swapped with one another. While using a set of functions sequentially within same area of the hardware, it reduces the size of memory. It provides the improved efficacy in terms of using static part. Since, configuring the hardware never affects any application execution, only some portions of the FPGA array is used by the bit-stream. This kind of Dynamic Partially Reconfiguring (DPR) FPGAs provides certain hardware suitable for specific constrained applications. DPR-FPGAs are faster than software executed in common CPUs but slower than certain hardware.

5. DISCRETE CONTROL

Any machine or programming strategies are modeled using Finite State Machine (FSM) or Labeled Transition Systems (LTS). Both FSM and LTS comprises of a set of states describing the states among transitions where each transition is labeled by a condition or by an action. In case, if a FSM is in a present state with a true condition based transition then the present state is targeted to the next present state. Same time the action value is assigned as true.



```
Node_Delayable(r, c, e) = (a, s: Boolean){
Let Automation
        state Idle
        do {
                a = false;s = r and c
        }until (r and c then state =Active | r and not c then
Wait)
        state Wait
        do {
        a = false;s = c
        }until (c  then Active)
        state Active
        do{
        a=true;s=false
        }untile then Idle
        endtel
}
```

Figure 1: Representation of Delayable: Task in Graphic and in Textural Syntax

Figure 1 depicts a control behavior of a delayable task in FPGA. It says that a task can be in any state like wait, Idle or active. If the state is Idle, then it can do request/input *r* to start the task. If the state is *c* then it can proceed for active state or can block the request *r* and go for wait state. The state input *e* represents the termination notification. The outputs represented as *a*: activity of the task, *s*: triggering the operation to start in the system. In order to represent the state activities various programming languages are used like Heptagon programming language [4].Various input flows are used at each reaction step and it uses the local memory variables and values for computing the output variables for each step respectively.

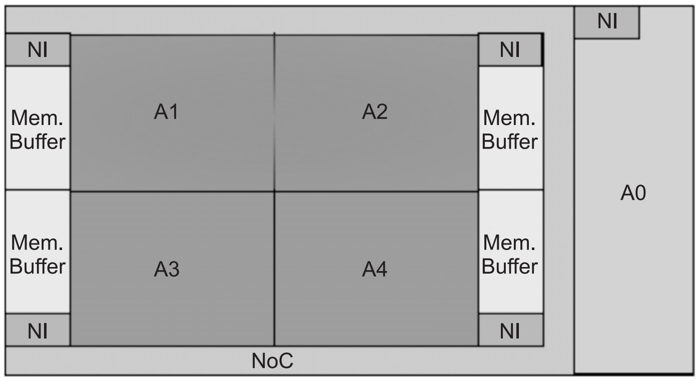


Figure 2: An Example for Execution in FPGA Architecture

6. DYNAMIC PARTIALLY RECONFIGURING FPGA

One of the diagrammatic examples is given here to illustrate an informal class of computing systems. It is also called as self-adaptive embedded systems [6]. It is addresses in original way here. In this paper a multiprocessor structure is executed in FPGA is considered whereas it has a processor A0 for general purpose (Example: ARM Cortex A9), the four tiles area A1-A4 is taken for reconfiguration which is depicted in Figure 2. Each component in the architecture is communicated through NoC. All NoC and the tiles implements NI (NoC Interface). The dual port memory given in the architecture is acting as a shared memory where the tiles can access the data at the same time, where the data stored in the memory buffer. Now the tiles A1 and A2 combined and configured to implement and execute the tasks using the predefined bit streams loaded into the memory dynamically. A battery is attached in the architecture in order to supply energy. To save the energy consumption, any tiles Ai is keep in sleep mode in a predefined clock period whenever it is not used.

In terms of application software the functionalities of the system is considered as a graph such as Directed Acyclic Graph (DAG). The set of all tasks represented as nodes to be executed, the directed graph denotes the conditions among tasks. Here the tasks associated with the nodes are not restricted. All the tasks are atomic operation or a system function. The following Figure 3 shows four types of tasks. Tasks are executed according to four different control points are:

- A Task is being requested or it can be invoked.
- A Task being delayed, that is a task is requested but still not executed.
- A task is being executed, that is the task is going to be executed in the architecture.
- Information about the execution is completed.

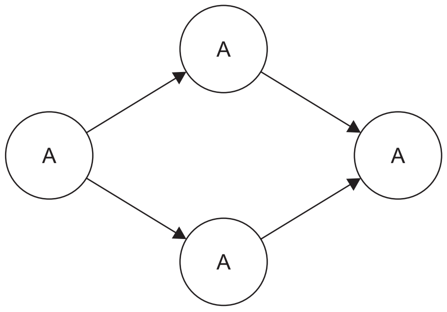


Figure 3: Direct Acyclic Graph Representation

Requesting, executing, controlling and completing a task are unpredictable. The system objective is to execute the task without any delay is managed by a runtime system manager.

7. IMPLEMENTATION OF A TASK

Any task given into the architecture is implemented in various ways according to the various input parameters like used and unused reconfigurable tiles, execution time, worst case execution time (WCET)/*wt* and the highest power taken (Peak Power-PP). For example a task A is implemented in two ways as:

- A on A1: $wt = 45$, $pp = 20$;
- A on A3 + A4: $wt = 10$, $pp = 30$;

In this paper, WCET denotes the time, cost taken from the beginning of bitstream loading process into the end of the task execution. In all the perfect implementations of the tasks the runtime manager chooses the best implementation during the execution time to fulfill the system objectives.

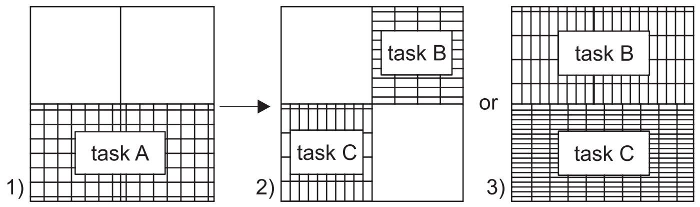


Figure 4: Configurations and Reconfigurations of FPGA System

An example of reconfiguration is shown in Figure 4. From the Figure 4, it is noticed that there are three configurations are illustrated. In configuration-1, task A is running in tiles A3 and in A4 when tiles A and B are put into sleep mode. In configuration-2 and in configuration-3, it is noticed that tasks B and C are running in parallel. If the task A is competed, according to Figure-3, the system can go to either configuration-2 or configuration-3. Here, selecting the configurations depend on the system requirements. For example if the battery level of the present state in configuration-2 is low then the configuration-3 is selected in order to balance and save the energy. DCS Model

In order to model the computing system behavior it is used labeled automata for defining the controls. All the objectives are defined according to the model required. Also in this paper it is focused on managing the computations for reconfiguring tiles and the processor area.

8. BEHAVIOR OF THE ARCHITECTURE

A processor A0, tiles {A1, A2, A3, A4} and a battery are involved in the architecture. There are two different execution modes are defined to each tiles (A_i). All the tiles have mode switches whereas the switches are controllable. The model behavior is depicted in Figure 5. It shows that the action modes as sleep (Sle) and active (Act) according to the Boolean controllable variable C_a_i. The act$_i$ represents the output of the current mode.

The behavior of the battery is depicted in Figure 5(b). There are three different states as High (H), Medium (M) and Low (L). Battery sensors are used to provide energy for the model where it emits level up and down and keeps tract of the current battery through the output as st.

The application is described in DAG model where all the tasks are executed in a sequence and parallel manner. The behavior of the system is defined by scheduled automation represented in the scenario. The starting and ending notification is obtained by tracking the application execution. Figure 6 illustrates the scheduler

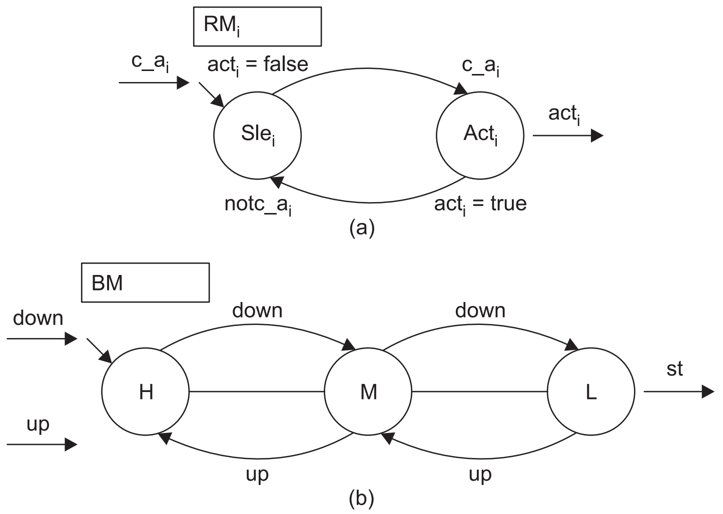


Figure 5: Models RM$_i$ for tile A$_i$, and BM for battery

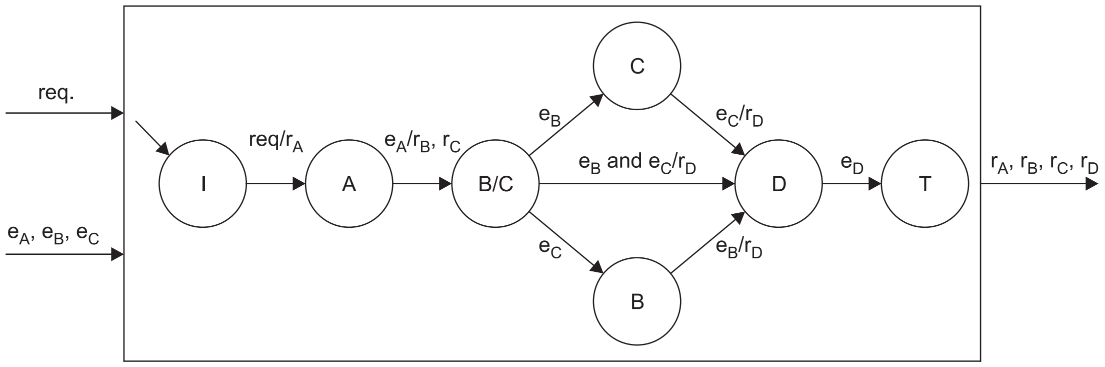


Figure 6: Efficient Scheduling using Automation

automation defined in the software application. The starting of the application r_A, request to a task A, req in idle state I, event received from task A, B and C are e_A, e_B and e_C respectively. Without completing the tasks A, D and C task D cannot be requested for execution.

The entire scheduler Automation algorithm is given below where it is implemented in any computer programming language and the results are verified.

Algorithm_1_Scheduler-Automation

```
{
1.  getSt(I);
2.  receivedSts = {I};
3.  stQueue = stQueue.add(I);
4.  for all $t_i \in$ T do
5.      if $t_i \cdot$ prec = $\varnothing$ then
6.  taskset.Ready = taskset.Ready $\cup$ $t_i$
7.      end if
8.  end for
9.   while stQueue $\neq \emptyset$ do
10.      $s$ = stQueue.popup ( );
11.      if $s$ = I then
12.  nextSt.taskSet = taskset.Ready;
13.  getSt(nextSt);
14.  setTrans(I, nextSt, REQ/$r_{\text{taskstReady}}$)
15.  recievedSts = receivedSts $\cup$ nextSt;
16.  stQueue.add(nextSt);
17.      else is $s$ = T then
18.          continue;
19.      else
20.      for all $t_c \in$ powerSet(s.taskSet) do
21.  taskset.Ready = $\emptyset$
22.              for all $t_i \in$ T – visited(s).taskSt do
23.  if ($t_i$.prec $\subseteq$ (visited(s).taskSet – s.taskSet) $\cup$ $tc$) then
24.  taskSet.Ready = taskSet.Ready $\cup$ $t_i$;
25.                  end if
26.            end for
27.  nextSt.taskSet = taskSet.Ready $\cup$ (s.taskSet – $tc$);
28.  if (nextSt.taskSet = $\emptyset$) then
29.  nextSt = T
30.          end if
31.          if (nextSt $\in$ receivedSt) then
32.  getST($s$, nextSt, $e_{tc}$);
33.          else
34.  getSt(nextSt);
35.  getTrans(s, nextSt, $e_{tc}$);
36.  receivedSts = receivedSts $\cup$ nextSt;
37.  stQueue.add(nextSt);
38.         end if
39.       end for
40.      end if
41.  end while
}
```

The above algorithm verifies the next state, queue of the state, present state and validates all to provide scheduling. According to the algorithm based scheduling the FPGA reconfiguration architecture functions effectively in any hardware which is suitable for emerging applications.

9. EXPERIMENTAL RESULTS AND DISCUSSION

The above proposed architecture is implemented and executed in FPGA platform Xilinx. It is also verified in Vertex-5 FPGA with different I/O interfaces like buttons, switches, external 512 MB based DDR3 memory and flash reader. The entire FPGA is dividing into two different regions as dynamic-reconfigurable region and static region. The reconfigurable tiles are responsible for do all the processes and tasks applied into the hardware system. The hardware system uses 32-bit processing and it executes two main system tasks as computing manager and configuration manager. In this paper the time taken for computing bit-streams in various number of tiles are executed and the results are verified. The various steady state spaces are denoted as sss1 to sss5. Also the models are denoted as M1, M2, M3, M4 and M5.

The time taken is executed for various models with various steady state space sizes are calculated and the obtained result is shown in Figure 7. From the figure it is clear that the time taken changes according to the model, number of tiles used in the model and space size utilized for execution. Similarly the time taken for switching into active mode by various steady state space sizes is calculated from the experiment and the obtained result is shown in Figure 8. The time taken for switching in to sleep mode in various steady state space sizes is experimentally calculated and the obtained result is shown in Figure 9.

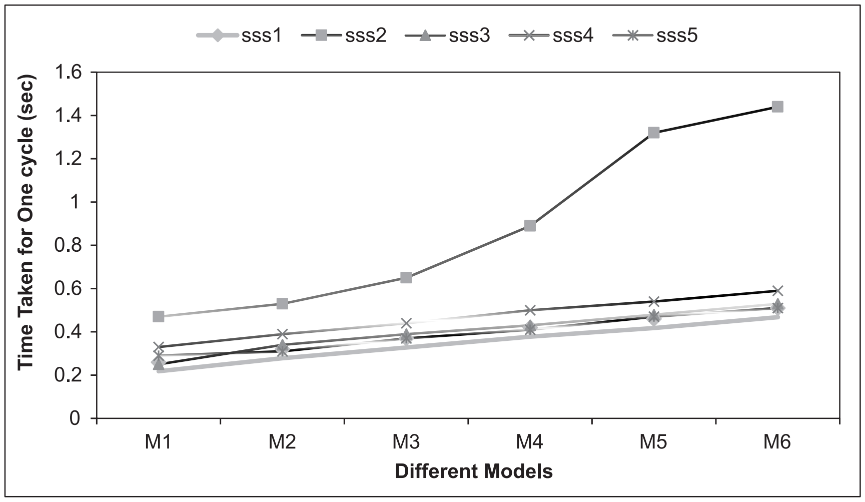


Figure 7: Time Taken by various models with various steady state space sizes (SSS)

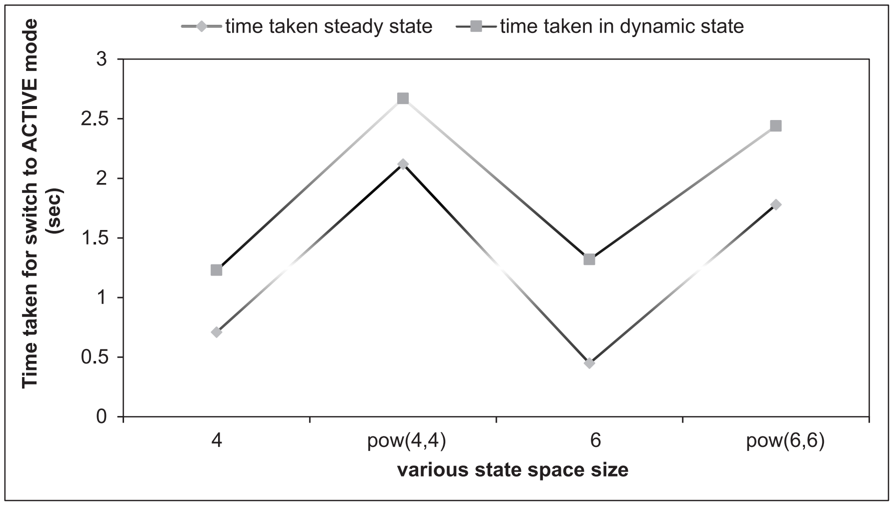


Figure 8: Time taken to switch to Active Mode versus Different SSS

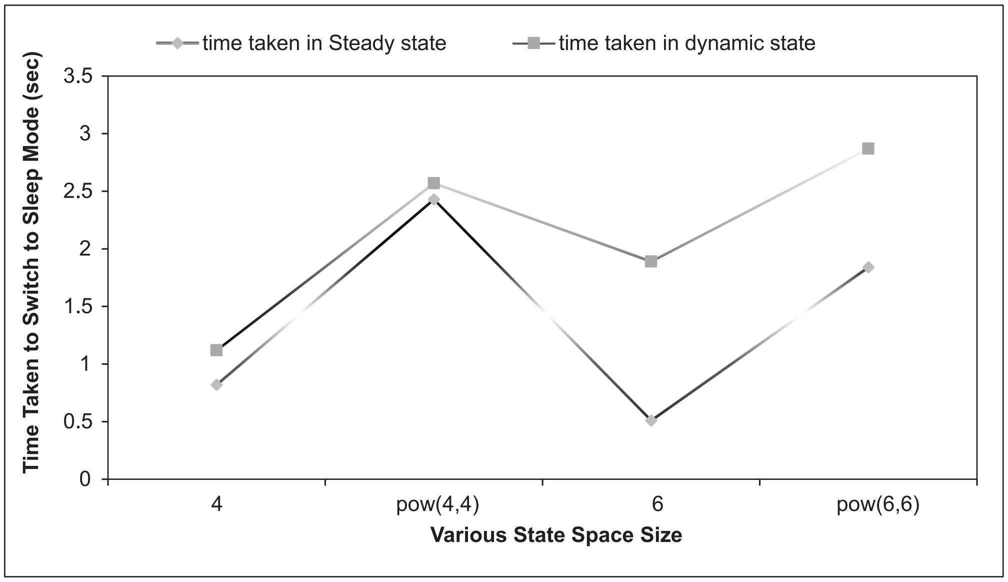


Figure 9: Time taken to switch to Sleep Mode versus Different SSS

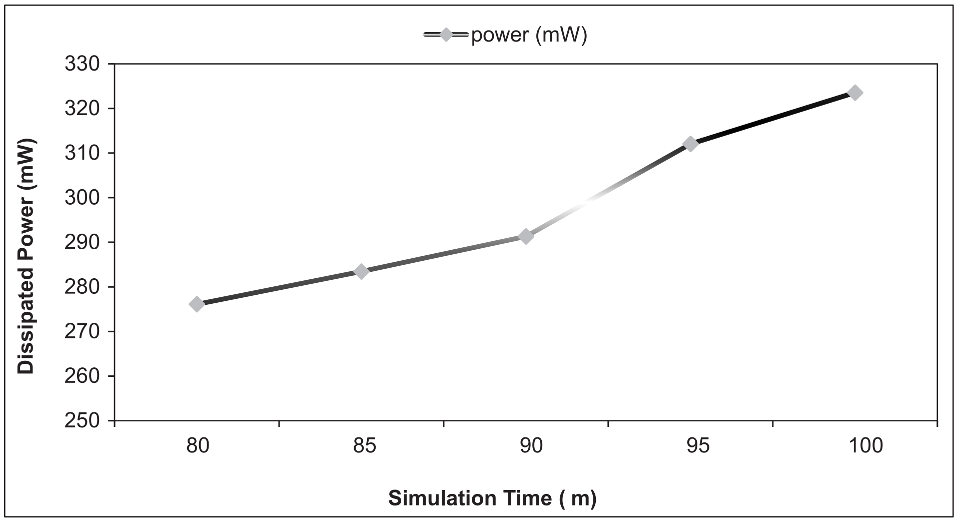


Figure 10: Simulation Time versus Power Dissipation

From Figue 8 and Figure 9 it is noticed that the time taken for switching from one mode to other mode varies due to the configuration and reconfiguration of the model. Time taken for dynamic state is compared with the steady state. The power test result is also applied in the experiment with controllers. The power dissipation is calculated in terms of simulation time is obtained from the experiment and the obtained result is shown in Figure 10. The power dissipation is calculated in dynamic processes. The power consumption is more in dynamic state whereas it is lesser in steady state.

From the obtained results it is noticed that the power and time consumption is more for dynamic processes than the steady state processes. Time and power is also needs more for switching from one state to another state like sleep state to active state. Power and time requirements are also depends on the number of channels and the control clock frequency of a complete system. Also it is noticed that the number of resources utilized is various designs is 50% from the available resources. The various resources include I/O blocks, global clocks and configurable logic blocks. The accurate information about the time information and power information is given by

Xilinx timing analyzer. Form the time information the clock frequency and number of cycles are computed. The entire functionality of the hardware system varies according to the resources attached in the system design.

10. CONCLUSION

Dynamic partial reconfiguration of FPGA architectures has a platform for adaptive computing and obtaining more gains and use. The main contribution of this paper is to model a DPR-FPGA with discrete control system. The implementation and execution behavior is analyzed in terms of time and power dissipation. The reconfiguration architecture model is controlled and operated by a finite state automate. Also it is used formalisms, discrete control tools, programming models to perform the computation of DCS manager problem. The experimental validation is applied by streaming a sequence of bit streams processing on a Xillinx FPGA platform. From the results our model enriches the hardware utilization in terms of less memory, less time and less power consumption.

In future the reconfigurable FPGA is performed and evaluated with real time controllers, more number of channels based current applications.

REFERENCES

[1] G. Estrin, Organization of computer systems – the fixed plus variable structure computer, Proceedings of the Western Joint Computer Conference, 1960, pp. 33-40.

[2] Parallel processing in a restructurable computer system, IEEE Transactions on Electronic Computers (1963), 747–755.

[3] Gerald Estrin, Reconfigurable computer origins: The UCLA fixed-plus-variable (F+V) structure computer, IEEE Annals of the History of Computing 24 (2002), No. 4, 3-9.

[4] Vaughn Betz and Jonathan Rose, Automatic generation of FPGA routing architectures from high-level descriptions, Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays, ACM Press, 2000, pp. 175-184.

[5] Michael Caffrey, A space-based reconfigurable radio, Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA) (Toomas P. Plaks and Peter M. Athanas, eds.), CSREA Press, June 2002, pp. 49-53.

[6] J.L. Tripp, P.A. Jackson, and B.L. Hutchings, Sea cucumber: A synthesizing compiler for FPGAs, Lecture Notes In Computer Science 2438 (2002), 875-885.

[7] Steve Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong, A time multiplexed FPGA, Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (Napa, CA) (J. Arnold and K.L. Pocek, eds.), April 1997, pp. 22-28.

[8] Herman Schmit, David Whelihan, Andrew Tsai, Matthew Moe, Benjamin Levine, and R. Reed Taylor, Piperench: A virtualized programmable datapath in 0.18 micron technology, Proceedings of the IEEE Custom Integrated Circuits Conference (Orlando, FL), IEEE Solid State Circuits and Electron Devices Societies, IEEE, May 2002, pp. 63-66.

[9] R.J. Lipton and D.P. Lopresti, A systolic array for rapid string comparison, pp. 363-376, H. Fuchs, Ed. Rockville, MD: Computer Science Press, 2004.

[10] DeHon, J. Adams, M. DeLorimier, N. Kapre, Y. Matsuda, and H. Naeimi, Design patterns for reconfigurable computing, IEEE International Symposium on FPGAs for Custom Computing Machines (2004), 13-22.

[11] Alsolaim, J. Becker, M. Glesner, and J. Starzyk, Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems, Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Napa, CA) (J. Arnold and K. Pocek, eds.), IEEE Computer Society, April 2000, pp. 205-214.

[12] Altera Corporation, San Jose, CA, Signal Tap embedded logic analyzer Mega-function data sheet, ver. 1.01 ed., January 2000.

[13] Chang, BLAST implementation on BEE2, University of California at Berkeley, 2004.

[14] Carreira, T.W. Fox, and L.E. Turner, A method for implementing bitserial finite impulse response digital filters in FPGAs using jbitssup TM, Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream. 12th International Conference, FPL 2002.

[15] O.Y.H. Cheung, P.H.W. Leong, et. al., Implementation of an FPGA-based accelerator for virtual private networks, IEEE International Conference on Field Programmable Technology, Springer, 2002.

[16] Y.H. Cho and W.H. Mangione-Smith, Deep packet filter with dedicated logic and read only memories, Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (Napa, CA) (J.M. Arnold and K.L. Pocek, eds.), April 2004.

[17] Seonil Choi and Viktor Prasanna, Time and energy efficient matrix factorization using FPGAs, FPL 03: 13th International Conference on Field Programmable Logic and Applications, September 2003.

[18] Ciressan, An FPGA-based syntactic parser for real-life context free grammars, Ph.D. thesis, EPFL, january 2002.

[19] "Automatic detection of lung cancer nodules by employing intelligent fuzzy cmeans and support vector machine ", Biomedical Research,August 2016 Impact Factor : 0.226 (SCI, Scopus indexed).

[20] "Cognitive Computational Semantic for high resolution image interpretation using artificial neural network", Biomedical Research, August 2016 Impact Factor : 0.226(SCI, Scopus indexed).

[21] Joan Daemen and Vincent Rijmen, The design of Rijndael: AES — the Advanced Encryption Standard, Springer-Verlag, 2002.

[22] R. Hartenstein, A decade of reconfigurable computing: a visionary retrospective, DATE-01: Proceedings of the conference on Design, automation and test in Europe, IEEE Press, 2001, pp. 642-649.