SYNTHESIS OPTIMIZATION FOR FINITE STATE MACHINE DESIGN IN FPGAS

R.UMA AND P. DHAVACHELVAN

Department of Computer Engineering, School of Engineering, Pondicherry University, Pondicherry, India
uma.ramadass1@gmail.com
dhavachelvan@gmail.com

ABSTRACT

Synthesis optimization plays a vital role in modern FPGAs in order to achieve high performance, in terms of resource utilization and reducing time consuming test process. Cell-based design techniques, such as standard-cells and FPGAs, together with versatile hardware synthesis are rudiments for a high productivity in ASIC design. As the capacity of FPGAs increases, synthesis tools and efficient synthesis methods for targeted device become more significant to efficiently exploit the resources and logic capacity. The synthesis tool provides the selection of different constraint to optimize the circuit. This paper presents the design and synthesis optimization constraints in FPGA for Finite state machine. The primary goal of this sequential logic design is to optimize the speed and area by choosing the proper options available in the synthesis tool. More over the work focuses the design of FSM with more processes operates at a faster rate and the number of slices utilized in an FPGA is also reduced when compare to single process. The module functionality are described using Verilog HDL and performance issues like slice utilized, simulation time, percentage of logic utilization, level of logic are analyzed at 90 nm process technology using SPARTAN6 XC6SLX150 XILINX ISE12.1 tool.

KEYWORDS

FPGA, FSM optimization, Synthesis constraints, State encoding, logic optimization

1. INTRODUCTION

Modern FPGAs became viable ASIC replacement because of very expensive fabrication process and time consuming test process. FPGAs are already fabricated allowing quick time-to-market. The amount of reconfigurable resources in a FPGA is fixed and limited. Within the available resources the application has to be embedded. Different ISE (Integrated development Environment) tools are available like Xilinx, Altera, Quartus etc. which provides various design constraints setting for various optimizations like speed and area to fulfill the requirement of design as well as the device selected in FPGAs. Generally hardware description language like VHDL and Verilog are used to develop the circuit descriptions. The synthesis tools optimize HDL code for both logic utilization and performance of an intended design. In FPGA each slices, LUT and register utilization are very important in order to accommodate larger design unit. Inefficient coding may also lead to adverse effect of synthesis to result in slow devices and occupies larger slices, LUT and registers.

In VLSI design flow, specifications are written first, specifications describe abstractly the functionality, interface and overall architecture of the digital circuit to be designed[2]. The circuit descriptions are written using HDL either VHDL or Verilog in terms of its behavior. The design at the behavioral level is to be elaborated in terms of known and acknowledge functional

DOI : 10.5121/vlsic.2012.3607

blocks[3]. Once the design is completed its functionality is tested using circuit simulators. After functionality test the RTL description are converted into gate-level netlist using logic synthesizer. A gate-level netlist is a description of the circuit in terms of gates and connections between them [4]. Synthesis is a process by which an abstract form of desired circuit behaviour (typically register transfer level (RTL)) is turned into a design implementation in terms of logic gates[5]. Logic synthesis tool ensure that the gate level netlist meets timing, area and power specifications. After several annotation if the expected output is derived then the final implementation is done through FPGA or ASIC.

Designing a synchronous Finite state machine (FSM) is a common task for a digital logic circuit. Sequential circuit optimization has been the subject of intensive investigation for several decades [1][12]. FPGA synthesis tool provides a variety of design constraints which essentially helps the designer to meet the design goal such as area and speed optimization to obtain the best implementation logic. This work focuses the selection of constrains and issues related to each constraint is elucidated. This paper details efficient Verilog coding styles to infer synthesizable state machines. HDL considerations such as advantages and disadvantages of one-always block FSMs Vs. two-always block FSMs are described.

The organization of the paper is as follows: Section 2 presents the implementation of Finite state machines with single and multiple processes. Section 3 focuses the performance issues of FSM with single and multiple processes. Section 4 presents the discussion. Finally the conclusion is presented in section 5.

2. FSM WITH SINGLE AND MULTIPLE PROCESSES

FPGA synthesis tool provides a variety of design constraints which essentially helps the designer to meet the design goal such as area and speed optimization to obtain the best implementation logic. This section describes the implementation of FSM with single and multiple processes and the important aspects of synthesis optimization like resource sharing for area, speed, latency and power. At the synthesis level, the high level description is converted into an optimized gate-level representation or RTL form. At this level of abstraction when global constraints of area or speed is set the synthesis tool will try to meet constraints, calculate cost of various implementation and try to generate best logic topology for given constraints, algorithm and target process. Normally synthesis tools use wire load models which statistically estimates the interconnect delay in the absence of physical layout data. For a wire delay with given fanout, the wire load model specifies the capacitance, resistance and area of the wire. Although the synthesis tool has complete control over the netlist, the resulting timing is greatly affected by the physical layout.

The synthesis based gate-level optimizations will include constraints like Finite State Encoding (FSM) algorithm (like auto, one-hot, compact, sequential, gray, Johnson, speed1), hierarchy setting (which allows MAP's physical synthesis options to optimize across hierarchical boundaries), logic duplication (avoids replication of logic), FSM style, register duplication and so on. By setting the required constraints the design can be optimized. As a general rule faster design requires parallelism at the expense of slice area, and minimize area design requires less logical depth.

This topic presents the issues in optimization of synthesis tool in sequential logic which is elucidated by the design of Finite State Machine (FSM) with different encoding algorithm constraints like gray, one-hot, sequential, Johnson, speed1 and auto. Finite state machine is a restricted class of sequential circuits called synchronous circuits which assumes the existence of a common global clock. An FSM is a discrete dynamic system that translates the sequence of input vectors into sequence of output vectors. States in an FSM specification can be either symbolic or

binary-encoded. To optimize the circuit at sequential level, state minimization and state assignment procedure are executed. State minimization reduces the redundant states and state assignment encodes the symbolic states into binary codes.

FSM with one Process

```
module fsm (clk, reset, x1, stateout);
  input clk, reset, x1;
  output stateout;
  reg stateout;
 reg [1:0] state;
parameter state1 = 2'b00;
parameter state2 = 2'b01;
parameter state3 = 2'b10;
 parameter state4 = 2'b11;
   always@(posedge clk or posedge
reset)
  begin
    if (reset)
      begin
        state = state1; stateout = 1'b1;
      end
    else
      begin
        case (state)
          state1: begin
                if (x1==1'b1) state = state2;
                else          state = state3;
                stateout = 1'b1;
              end
          state2: begin
                state = state4; stateout = 1'b1;
              end
          state3: begin
                state = state4; stateout = 1'b0;
              end
          state4: begin
                state = state1; stateout = 1'b1;
              end
        endcase
      end
  end
```

FSM with Two Processes

```
module fsm1 (clk, reset, x1, out);
  input clk, reset, x1;
  output out;
  reg out;
reg [1:0] state;
parameter state1 = 2'b00;
 parameter state2 = 2'b01;
parameter state3 = 2'b10;
 parameter state4 = 2'b11;

   always @(posedge clk or posedge
reset)
  begin
    if (reset)
      state = state1;
    else
      begin
        case (state)
          state1: if (x1==1'b1) state =
state4;
                else          state = state3;
          state2: state = state3;
          state3: state = state4;
          state4: state = state1;
        endcase
      end
  end

  always @(state)
  begin
        case (state)
          state1: out = 1'b1;
          state2: out = 1'b1;
          state3: out = 1'b0;
          state4: out = 1'b1;
        endcase
```

FSM with Three Processes

```
module fsm2 (clk, reset, x1, stateout);
  input clk, reset, x1;
  output stateout;
  reg stateout;
 reg [1:0] state;
reg [1:0] next_state;
parameter state1 = 2'b00;
 parameter state2 = 2'b01;
parameter state3 = 2'b10;
 parameter state4 = 2'b11;
   always @(posedge clk or posedge reset)
  begin
    if (reset)  state = state1;
    else        state = next_state;
  end

  always @(state or  x1)
  begin
        case (state)
          state1: if (x1==1'b1) next_state =
state2;
                else          next_state = state3;
          state2: next_state = state4;
          state3: next_state = state4;
          state4: next_state = state1;
        endcase
  end
   always @(state)
  begin
        case (state)
          state1: stateout = 1'b1;
          state2: stateout = 1'b1;
          state3: stateout = 1'b0;
          state4: stateout = 1'b1;
        endcase
  end
```

The following example presents the design of FSM with one, two and three processes providing the same specification with different encoding algorithm constraints. The state transition and output of the states are the same for all the different constructs using several processes. The HDL construct of this FSM with different process statement is shown in Fig 1. This is a FSM design for Moore machine with asynchronous "RESET". The state machine consist of four states namely state1, state2, state3 and state4 having 5 transitions with single input "x1" and single output "stateout". The state transition and state diagram is shown in Table 1 and Fig 2. In FSM with single process defines both the state transition and output of the state in single always statement. In FSM with two processes is designed in such a way that state transition in one always statement and the output of the state is defined in the second always statement. In FSM with three processes is designed in the manner that the first always statement initializes the state by enabling asynchronous reset signal, the second always statement is designed to provide state transition and the third always statement produces the state outputs. All the three FSM produces same state transition and same output. These FSM are synthesized by setting the global constraint for speed and area. The FSM are also synthesized by enabling different encoding schemes available in the Xilinx tool like auto, one-hot, gray, compact, sequential, speed1, user, Johnson and none options.

FPGA synthesis tool supports different encoding options related to the state machine implementation. By setting the auto constraint option the synthesis tool will provide the best encoding algorithm for each FSM. This option is the default setting in the synthesis tool. In one-

hot encoding option each state is associated with one code bit and also one flip-flop to each state. By triggering the clock pulse only one state variable is asserted. When one-hot is used, the next-state equation for each flip-flop will contain one term for each link path leading into the corresponding state. On asynchronous reset only one flip-flop will be set to '1' instead of resetting all flip-flops to '0'. For this FSM design the encoding states are "0001, 0010, 0100 and 1000".

In compact encoding option the numbers of bits in the state registers are minimized using hypercube are also called code space immersion technique to minimize area. This hypercube technique uses code equivalence to fix the optimal state encoding (code) by assigning its true and complementary state assignment values like "00, 11, 01, 10" for four states.

Table 1 State Table

Present State	**Next State**		**State output**
	X1=0	X1=1	
State1	State3	State2	1
State2	State2	State4	1
State3	State3	State4	0
State4	State4	State1	1

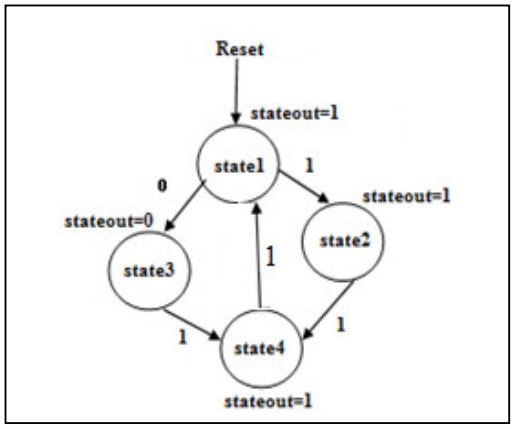
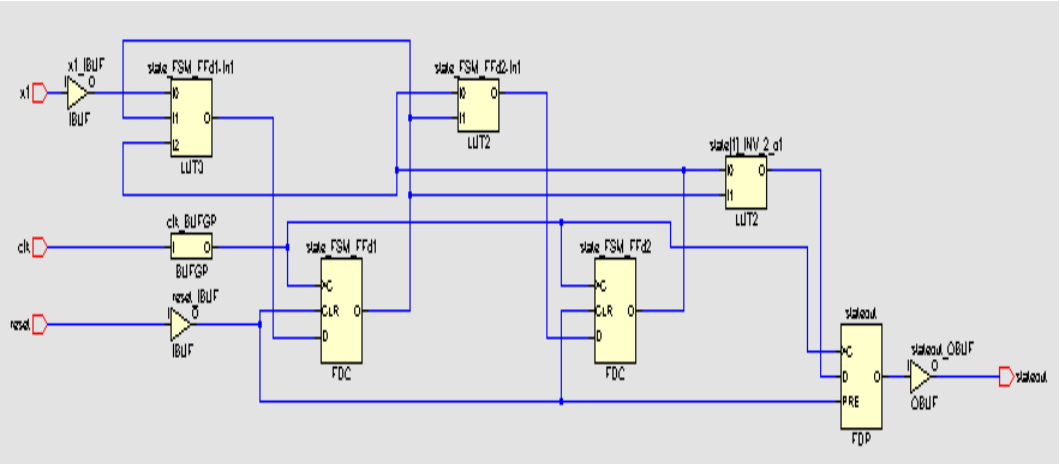


Figure 2. State Transition

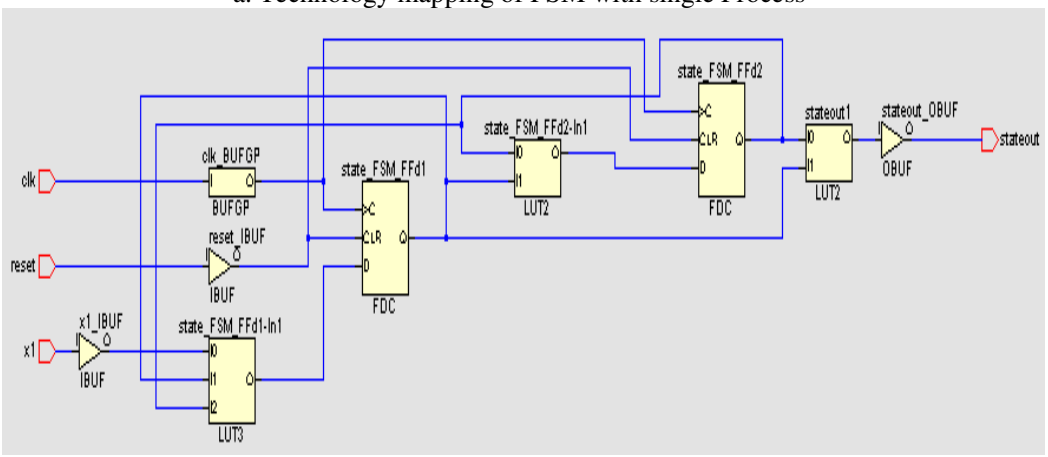
The gray code encode option allows only one bit difference in the adjacent codes to reduce the number of transition for low power purpose. Because of one bit transition unnecessary hazards and glitches can be minimized. Binary (sequential) encoding option encodes the states as consecutive binary numbers. More than one bit of the state register can change at a time; because more than one bit can be hot the value must be decoded to determine the state. The decoded value for this option will be "00, 01, 10 and 11". The user option will cause the synthesis tool to use the encoding defined in the source file. The Johnson encoding option is similar to gray code providing one bit transition, shows benefits with state machines containing long paths with no branch. Speed1 encoding option is specified for speed optimization. The state encoding depends on each FSM, normally the number of bits assigned for state register will be greater that of the FSM states. The none option will disables automatic FSM extraction.

The code generated for FSM with single and multiple processes is synthesized with various constraints for speed and area. The technology mapping of this FSM for single, two and three processes are shown in Fig 3. For single process the inputs X1 and reset are buffered. For clock generation the device utilizes the internal clock buffer called BUFGP. For this the state transition utilizes 3 LUTs and 3 internal registers. Two internal register have been designed with positive edge clock with its internal primitive named as FDC and the third one is designed with negative edge clock with its internal primitive named as FDP. For stateout output signal the device has utilized one output buffer OBUF. So the total number of input and output buffer utilized are 3 buffer, 3 LUT and 3 registers with positive and negative edge clock for single process. Similarly FSM designed with two and three utilizes two input buffer for X1 and reset, one output buffer for state output. This FSM uses only 3 LUTs and 2 internal registers with positive edge clock. So the total numbers of input and output buffer utilized are 3buffers, 3 LUT and 2 registers with positive

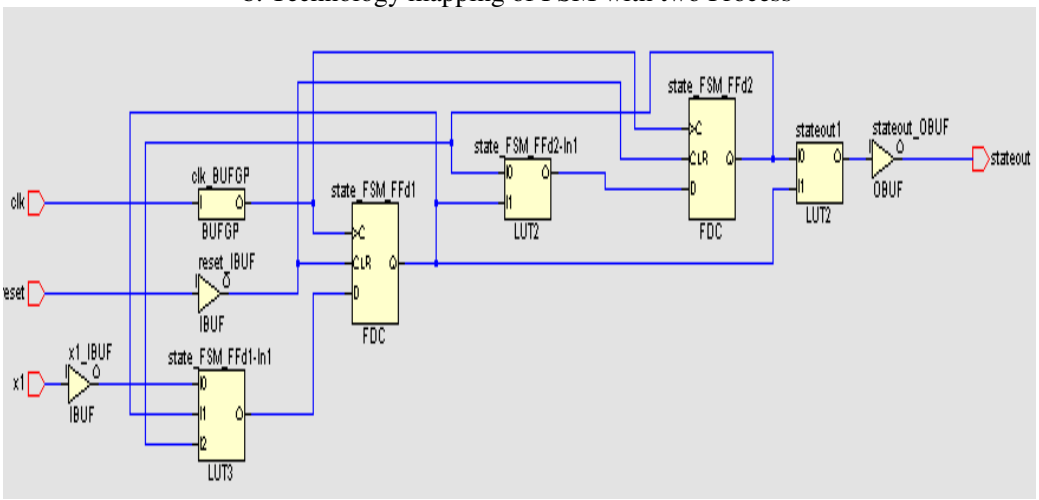
edge clock for two and three processes. Therefore for two and three process the utilization is less when compare to single process.



a. Technology mapping of FSM with single Process



b. Technology mapping of FSM with two Process



c. Technology mapping of FSM with three Process

Figure 3. Technology mapping of single and multiple processes

3. PERFORMANCE AND SYNTHESIS RESULT OF FSM WITH SINGLE AND MULTIPLE PROCESSES

The synthesis report of finite state machine for single and multiple processes by setting different encoding scheme are listed in Table 2. This Table depicts the total number of slices occupied, the encoding style for different options like gray, on-hot, sequential etc., minimum period and maximum clock frequency (Input period / Input frequency), minimum input arrival time before clock (Min input) and maximum output required time after clock (Max output). This synthesis result shows that the auto encoding scheme has chosen gray code as the best encoding algorithm for this FSM. For one-hot option for both single and multiple processes the number of slices utilized is more when compare to other options, but the input period, maximum clock frequency and clock load are higher when compare to other encoding algorithm.

For compact and sequential options the slices utilization and clock load are seen to be the same but both these options uses different encoding algorithm and the input and output delays are different. For Johnson and gray the encoding method are the same and produces the same characteristics for both delay and slice utilization. For user and none option the slice utilization and delay are the same but the user option takes the encoding scheme as specified in the coding whereas the none option will not take any encoding algorithm. For speed1 option the slices utilized will be large with long input period and less clock frequency. This synthesis result presents the clock load for one-hot and speed1 are higher when comparing to other encoding algorithm. The maximum bit representations for encoding the states are occupied by one-hot and speed1 with a sequence of “0001, 0010, 0100, 1000” and “1000, 0011, 0010, 0101”.

Fig 4 shows the synthesis variation of FSM in terms of register utilization, LUT utilization, input period, maximum clock frequency, minimum input arrival time and maximum output required time for single and multiple processes. From register utilization graph it can be observe that one-hot and speed1 option utilizes 5 and 4 register slices for single and multiple processes whereas the remaining option occupies 3 and 2 register slices. So for minimum number of register utilization gray or compact or Johnson is suitable. From LUT utilization graph it shows that the maximum LUT utilization occurs for one-hot option with 4 LUTs and minimum utilization occurs for speed1 with 2 LUTs. So for minimum number of LUT usage speed1 is preferable.

The input period graph illustrates the longest input period occurs for user, compact and sequential option and the shortest occurs for one-hot. This parameter defines the speed optimization take place, if the input transition occurs for longer period. Therefore for high speed prerequisite speed1 and one-hot are appropriate. From the maximum clock frequency graph it can be seen that the highest frequency arise for one-hot and the lowest frequency arise for compact, sequential and none. So this parameter defines the significance of speed, illustrating that if the input period time is fast then the clock frequency will be maximum. The minimum input arrival time graph exemplify the arrival time is longer for one-hot and speed1, and shorter for gray, Johnson and compact. Maximum output required time graph shows no significant changes but shows variation for single and multiple processes.

From this analysis the significance of each encoding algorithm is very comprehensible. Finite state machines designed with one-hot algorithm are typically faster. In this encoding method speed is independent of the number of states, though this scheme utilizes more next-state equations; the algorithm depends only on the number of transitions into particular state. The disadvantage is that one-hot encoding typically requires many registers. For sequential encoding algorithm the major merit is that the number of states examined is independent of multiple input constraints. The limitation in this algorithm is that this encoding method will depend on all flip-flops in the state representation, and thus a state-decode will be indispensable.

FSM Type	Process	Slices Occupied				Encoding Style		Input period/Input Frequency ns/MHz	Delay (ns)	
		No of Slice Register	No of Slice LUT	No of Bonded I/Os	No of BUFG	State	Encoding		Min input	Max output
Auto	single	3	3	4	1	00 | 00 10 | 01 01 | 11 11 | 10		1.608 621.794	2.34	3.819
	Two	2	3	4	1			1.608 621.794	2.34	4.828
	Three	2	3	4	1			1.608 621.794	2.34	4.828
One-hot	single	5	4	4	1	00 | 0001 10 | 0010 01 | 0100 11 | 1000		1.574 635.223	2.40	3.819
	Two	4	4	4	1			1.537 650.682	2.37	4.794
	Three	4	4	4	1			1.537 650.682	2.37	4.794
Compact	single	3	3	4	1	00 | 00 10 | 11 01 | 01 11 | 10		2.083 480.054	2.34	3.819
	Two	2	3	4	1			1.574 635.223	2.34	4.724
	Three	2	3	4	1			1.574 635.223	2.34	4.724
Sequential	single	3	3	4	1	00 | 00 10 | 01 01 | 10 11 | 11		2.083 480.054	2.38	3.819
	Two	2	3	4	1			1.574 635.223	2.38	4.724
	Three	2	3	4	1			1.574 635.223	2.38	4.724
Gray	single	3	3	4	1	00 | 00 10 | 01 01 | 11 11 | 10		1.608 621.794	2.34	3.819
	Two	2	3	4	1			1.608 621.794	2.34	4.828
	Three	2	3	4	1			1.608 621.794	2.34	4.828
Johnson	single	3	3	4	1	00 | 00 10 | 01 01 | 11 11 | 10		1.608 621.794	2.34	3.819
	Two	2	3	4	1			1.608 621.794	2.34	4.828
	Three	2	3	4	1			1.608 621.794	2.34	4.828
User	single	3	3	4	1	00 | 00 10 | 10 01 | 01 11 | 11		2.083 480.054	2.38	3.819
	Two	2	3	4	1			1.574 635.223	2.38	4.724
	Three	2	3	4	1			1.574 635.223	2.38	4.724
Speed1	single	5	2	4	1	00 | 1000 10 | 0011 01 | 0010 11 | 0101		2.012 497.092	2.40	3.819
	Two	4	2	4	1			1.574 635.223	2.37	4.653
	Three	4	2	4	1			1.574 635.223	2.37	4.653
None	single	3	3	4	1	No encoding		2.083 480.054	2.38	3.819
	Two	2	3	4	1			1.574 635.223	2.38	4.724
	Three	2	3	4	1			1.574 635.223	2.382	4.724

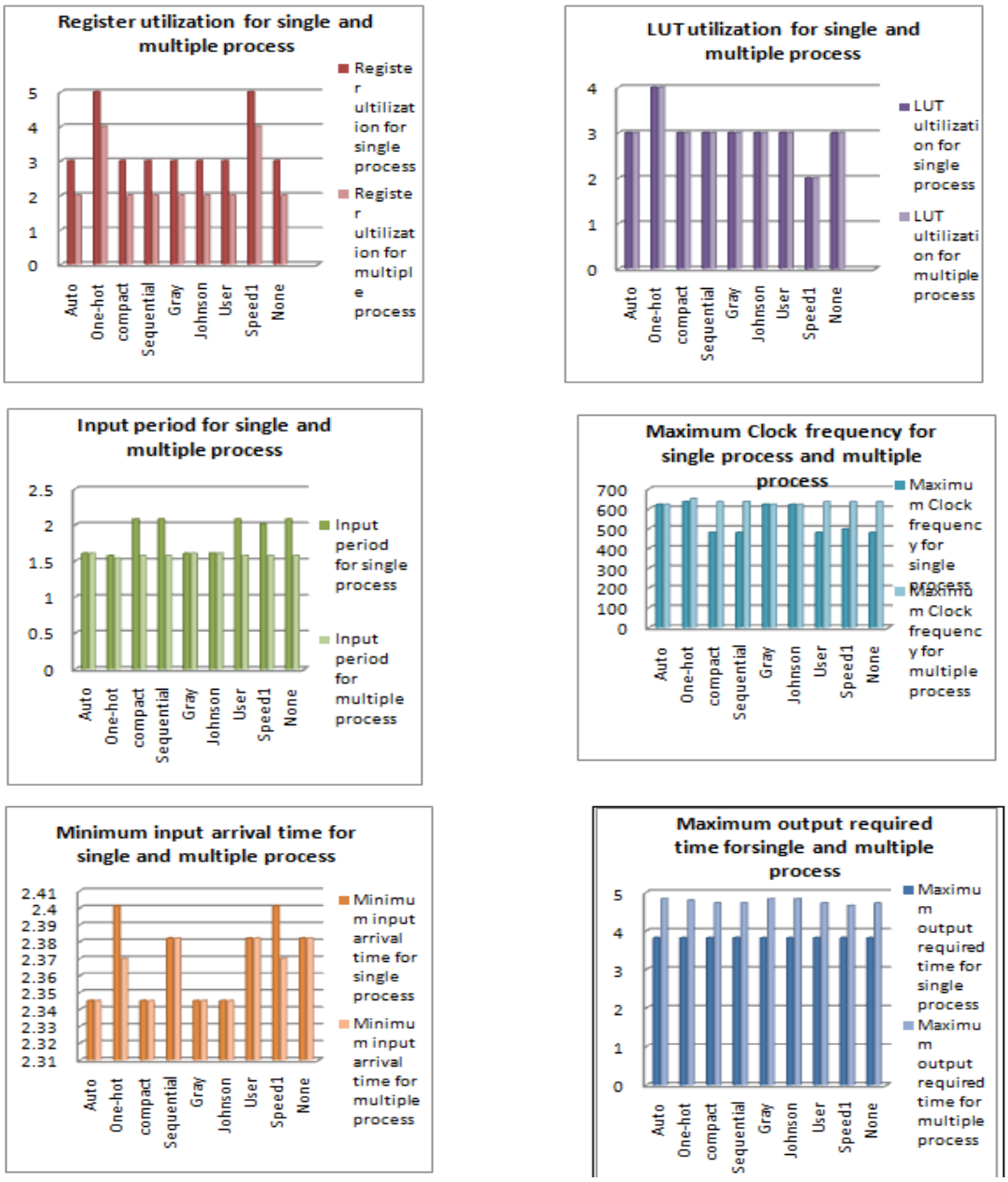


Figure 4. Synthesis variation of FSM in terms of register utilization, LUT utilization, input period, maximum clock frequency, minimum input arrival time and maximum output required time for single and multiple processes

The main significant of gray code is that codes assigned to two adjacent states have minimal possible Hamming distance equal to 1. Therefore glitches at the combinational FSM outputs are reduced. The number of state register flip-flops is also minimal. The disadvantage of this scheme is that both the next state logic as well as output logic depends on all values of all state register. Johnson encoding scheme allows codes in the adjacent states have the hamming distance of their codes equal to one, therefore reducing the combinational glitches at the output. The main drawback of this scheme is that the number of flip-flops in the state register is equal to N/2, where N is the number of states. Speed1 encoding is oriented for speed optimization. The number of bits

for state register depends on each FSM, but in general case it is greater than the number of FSM states.

To consolidate this analysis, for the design of finite state machine for constraint fixed for speed, then one-hot and speed1 encoding are appropriate though it utilizes more number of registers. This encoding provides minimum input period and maximum clock frequency. So the FSM synthesized with one-hot and speed1 converges faster when compare to other encoding algorithms. A finite state machine designed for the constraint fixed for area, then gray code, Johnson, compact and sequential are suitable. This encoding method provides minimum number of state registers and the adjacent states will have Hamming code distance equal to 1, thereby eliminating the combinational glitches at the output. Also this encoding scheme provides minimum register and LUT utilization with minimum delay. From this analysis it is also observed that the FSM designed with multiple process have utilized less register and LUT when compare to the FSM designed with single process. For multiple process the clock frequency as well as the input period time is larger when compare to single process. So to conclude these issues, design FSM with multiple processes to reduce register and LUT utilization. For speed optimizations prefer one-hot and speed1 option and for area optimization chose gray or Johnson or sequential encoding scheme.

4. DISCUSSION

With increasing complexity and demand for high performance FPGA circuit design, there is greater need for synthesis optimization for proper resource utilization and logic that are embedded in the targeted device. The focus has been to minimize the active area, speed and resources in a FPGA target device. The synthesis optimization in FSM design with single and multiple processes were presented. From this analysis the significance of each encoding algorithm is very comprehensible. A FSM designed for high speed target then one-hot and speed1is more preferable. A FSM designed for the constraint fixed for area, then gray code, Johnson, compact and sequential are suitable. This encoding method requires minimum number of state registers and it eliminates the combinational glitches at the output. From this analysis it is also observed that the FSM designed with multiple process have utilized less register and LUT when compare to the FSM designed with single process. For multiple process the clock frequency as well as the input period time is larger when compare to single process. So to conclude these issues, design FSM with multiple processes to reduce register and LUT utilization. For speed optimizations prefer one-hot and speed1 option and for area optimization chose gray or Johnson or sequential encoding scheme.

5. CONCLUSION

This paper presents the synthesis optimization for various constraints to minimize the resource utilization and logic density in the design of FSM. To design a finite state machine for constraint fixed for speed, then one-hot and speed1 encoding are appropriate though it utilizes more number of registers. This encoding provides minimum input period and maximum clock frequency. This algorithm has longest minimum input arrival time and has moderate maximum output required time. So the FSM synthesized with one-hot and speed1 converges faster when compare to other encoding algorithms. A finite state machine designed for the constraint fixed for area, then gray code, Johnson, compact and sequential are suitable. This encoding method provides minimum number of state registers and the adjacent states will have Hamming code distance equal to 1, thereby eliminating the combinational glitches at the output. Also this encoding scheme provides minimum register and LUT utilization with minimum delay. From this analysis it is also observed that the FSM designed with multiple process have utilized less register and LUT when compare to the FSM designed with single process. For multiple process the clock frequency as well as the input period time is larger when compare to single process. So to conclude these issues, design FSM with multiple processes to reduce register and LUT utilization. For speed optimizations

prefer one-hot and speed1 option and for area optimization chose gray or Johnson or sequential encoding scheme.

REFERENCES

[1] R.Uma, "Qualitative Analysis of Hardware Description Languages: VHDL and Verilog" (IJCSIS) International Journal of Computer Science and Information Security, Vol. 9, No. 4, pp-127-135, April 2011.

[2] JingXia Wang, Sin Ming Loo, "Case Study of Finite Resource Optimization in FPGA Using Genetic Algorithm", IJCA, Vol. 17, No.2, June 2010

[3] Alberto Sangiovanni-Vincentelli, Abbas El Gamal And Jonathan Rose, "Synthesis Methods for Field Programmable Gate Arrays" Proceedings of the IEEE, VOL. 81, NO. 7, JULY 1993

[4] Matthew French, Li Wang, Tyler Anderson, Michael Wirthlin "Post Synthesis Level Power Modeling of FPGAs" Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05) 2005 IEEE

[5] Maico Cassel, Fernanda Lima Kastensmidt "Evaluating One-Hot Encoding Finite State Machines for SEU Reliability in SRAM-based FPGAs" Proceedings of the 12th IEEE International On-Line Testing Symposium (IOLTS'06)

[6] abrizio Ferrandi, Pier Luca Lanzi, Gianluca Palermo, Christian Pilato, Donatella Sciuto, Antonino Tumeo Politecnico di Milano "n Evolutionary Approach to Area-Time Optimization of FPGA designs" 2007 IEEE

[7] St anisław Deni ziak, Mar iu s z WiĞniews ki "A Symboli c RTL Sy nt hesis for LUT- based FPGAs 2009 IEEE

[8] Jason Cong Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang "IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, VOL. 30, NO. 4, April 2011

[9] Digital System Design Using VHDL, Charles H. Roth, Jr., PWS Publishing Company, 1998, pp 25-27

[10] Steve Golson "One-hot state machine design for FPGAs" March 30, 1993 -- 3rd PLD Design Conference, Santa Clara CA

[11] P. Dhavachelvan, G.V. Uma and V.S.K.Venkatachalapathy (2006),"*A New Approach in Development of Distributed Framework for Automated Software Testing Using Agents*",International Journal on Knowledge –Based Systems, Elsevier, Vol. 19, No. 4, pp. 235-247, August-2006.

[12] Jian Li , Ra jesh K. Gupta "HDL Code Restructuring Using Timed Decision Tables" http://www.ics.uci.edu/iesag

Authors

R.Uma received her B.E. (EEE) degree from Bharathiyar University Coimbatore in the year 1998, Post graduated in M.E (VLSI Design) from Anna University Chennai in the year 2004. Currently she has been working as Assistant Professor in Electronics and Communication Engineering, Rajiv Gandhi College of Engineering and Technology, Puducherry. She authored books on VLSI Design. She has published several papers on national and International journal and conferences. She is the guest faculty for Pondicherry University for M.Tech Electronics. She has received the best teacher award for the year 2006 and 2007. Her research interests are Analog VLSI Design, Low power VLSI Design, Testing of VLSI Circuits, Embedded systems and Image processing. She is a member of ISTE. Perusing her Ph.D. from Pondicherry University in the Department of Computer Science.

Dr. P. Dhavachelvan is working as a Professor in the Department of Computer Science, Pondicherry University, India. He obtained his B.E. in the field of Electrical and Electronics Engineering from University of Madras, India. He pursued his M.E. and Ph.D. in the field of Computer Science and Engineering from Anna University, Chennai, India. He has about 15 years of experience as an academician and his research areas include Software Engineering & Standards and Web Service Computing. In his credit, he has more than 125 research papers published in reputed International and National Journals and Conferences. He also obtained Patents and proposed Standards in the domain of Software Engineering.