8-1-2006

A Study of Finite State Machine Coding Styles for Implementation in FPGAs

Nader I. Rafla
Boise State University

Brett L. Davis
Hewlett Packard, Boise, ID

A Study of Finite State Machine Coding Styles for Implementation in FPGAs

Nader I Rafla
Electrical and Computer Engineering
Boise State University
Boise, Idaho
nrafla@boisestate.edu

Brett LaVoy Davis
Hardware Design Enginner
Hewlett Packard
Boise, Idaho
brett.davis2@hp.com

Abstract— **Finite State Machines (FSM), are one of the more complex structures found in almost all digital systems today. Hardware Description Languages are used for high-level digital system design. VHDL (VHSIC Hardware Description Language) provides the capability of different coding styles for FSMs. Therefore, a choice of a coding style is needed to achieve specific performance goals and to minimize resource utilization for implementation in a re-configurable computing environment such as an FPGA. This paper is a study of the tradeoffs that can be made by changing coding styles. A comparative study on three different FSM coding styles is shown to address their impact on performance and resource utilization for the most commonly used encoding methods for FPGA designs. The results show that a particular coding style leads to a savings in resource utilization with a significant performance improvement over the others while the others pose a consistent performance regardless of the resource utilization outcome.**

I. INTRODUCTION

As with any programming language, regardless if it is a hardware description language such as VHDL or software programming language like C++, there will always be more than one way to write code to accomplish the same task [1, 2]. Each language provides several viable alternatives to the designer as to how to accomplish a given task.

The challenge facing a developer is to efficiently utilize these alternatives, while developing the code, within a constrained environment to accomplish the task at hand. However, there are no guidelines presented in the current literature as to how a hardware designer should develop code in a specific way to maximize performance or resource utilization when designing a digital hardware system using VHDL [3]. In this paper, we address this deficiency by evaluating three different methods of coding a finite state machine using two different state assignment-encoding schemes when implemented in an FPGA[4].

A finite state machine is treated as a hardware module that may have multiple inputs and can make decisions based on these inputs over time. There may be multiple paths that the machine can take based on the input values or transitions. The FSM outputs may depend on either the FSM state, often described as a Moore FSM, or a combination of the FSM state and the state of the inputs, described as a Mealy FSM. These are the only two types that are considered here since they are suitable for FPGA implementation.

II. CODING STYLES OF FINITE STATE MACHINES

There are three methods commonly used for coding a FSM: Combined Single Process (CSP), State Separated Combinatorial Outputs (SCO), and State Separated Registered Outputs (SRO) [5]. In the mean time, there are several methods used to encode the state registers within these FSMs. One-Hot, and Gray encoding are the most common choices used by the HDL designers targeting FPGAs [6]. A thorough discussion of the pros and cons of these coding styles along with their applications can be found in [7, 8]

A. Combined Single Process

This method uses a single process to control both the state transitions and outputs. It is claimed to be the most common method used for FSM design. An observed drawback to this method for coding a FSM is that synthesis software has difficulty identifying it as a finite state machine structure [9]. However, this may be overcome by having the outputs inferred into a clocked element, such as a flip-flop. Figure 1 shows a block diagram of the implementation of this type.

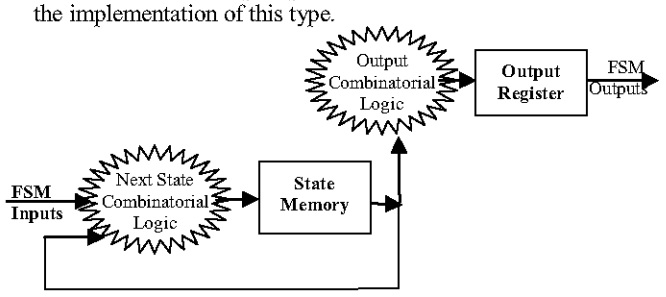

Figure 1. Combined single process

B. State separated combinatorial outputs

This coding style uses two processes, but has all of the output signals assigned in a combinatorial process, which infers that all

1-4244-0173-9/06/$20.00 ©2006 IEEE.

of the outputs are driven by combinatorial logic and are not registered as shown in Figure 2.

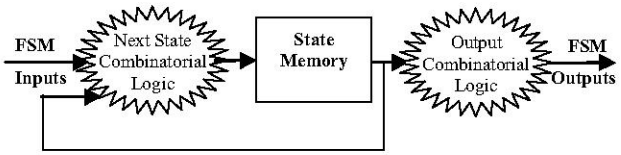

Figure 2. State separated combinatorial output

C. *State separated registered outputs*

The third coding style also employs two processes, but it provides a different method for registering the outputs. Each output signal is first inferred as a combinatorial logic then registered based upon the next state and not the current state as shown in Figure 3.

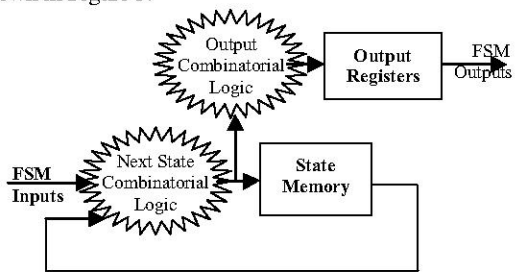

Figure 3. State separated registered outputs

III. STATE MACHINE ENCODING

There are many different methods available to encode the various states of a finite state machine [7]. The preferred method is entirely left up to the developer's preference. In our case, consideration of the encoding method must be taken into account because the type of encoding may have an impact on the target FPGA device resources. Highly encoded state assignment such as Gray encoding uses fewer flip-flops, but requires more combinatorial logic to encode and decode the state. On the other hand, One-Hot encoded state assignments use more flip-flops because each state is assigned a flip-flop, but utilize fewer logic resources, simplifying the decode process. Both of these techniques are evaluated.

IV. DESIGN OF THE EXPERIMENT

For our purpose of characterization, a constant environment must be maintained and limited to those parameters only affecting the coding styles. Thus, certain parameters remain unchanged throughout the evaluation process such as tool selection, tool configuration, target device for implementation, FSM function, and target device architecture.

Coding and implementation are conducted using the Xilinx tool suite, ISE 5.2i [10], targeting the FPGA device XC2S600E [11]. In order to focus only on changes in the FSM, common components are implemented separately and instantiated as needed by the FSM. In addition, a common component constraints file that defines the targeted device and other required timing parameters are utilized.

To ensure that the three coding styles operated identically, a common VHDL behavioral testbench was developed. The testbench modeled all possible combinations of input variations and monitored the outputs at all states. Modelsim XE 5.6b simulation tool [12] is then used to verify that the functions of the finite state machines developed remain the same for all tested coding styles.

Based upon this methodology, conclusions about the resource usage and performance of state machines can be drawn along with their dependency on the coding style and state encoding.

V. CASE STUDY

The finite state machine designed for this evaluation controls a unique serial protocol. The protocol consists of five signals, three inputs and two outputs as shown in Figure 4. For clarification, a signal name ending with an "_I" indicates an input and with an "_O" indicates an output. It is important to note that during the assertion of both nCTS_I and nRTS_O, multiple data byte sequences are possible.

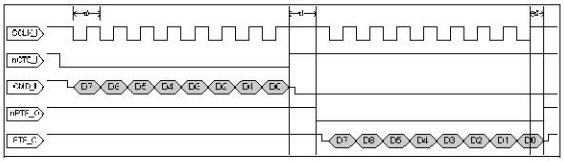

Figure 4. Typical serial protocol operation

In order to introduce additional complexity into the state machine design, an exception is added into the typical operation. This exception provides a means by which the receiving entity can signal the transmitting entity that data is ready to be sent but the transmitting entity needs to initiate the transfer. Figure 5 provides an illustration of the exception and how it is handled.

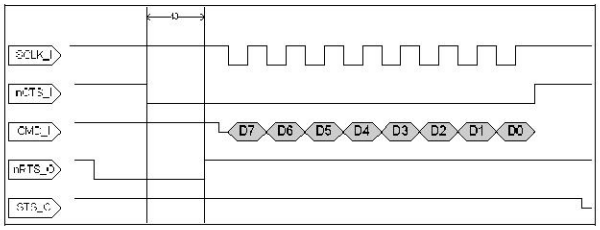

Figure 5. Serial protocol special case

A. *State Machine Challenges*

When creating the state machine to handle this protocol, the following challenges were identified:

- SCLK_I is not a free-running clock; therefore, it could not be used as the clock to move the state machine. Accordingly, an additional system clock is used for the whole system.
- nCTS_I is a required input because it's de-assertion initiates the start of time t1.
- nRTS_O and STS_O are the primary outputs that are monitored.
- Since control is only required for nRTS_O and STS_O, there is no need to monitor CMD_I.
- The dynamic amount of the data transmitted (single byte or multiple bytes) needs to be monitored.
- Active monitoring of the SCLK_I pulses is needed to observe the state machine state transitions.

- Management of timers t1, t2, and t3 is required to control movement of the state machine.

Additional components, such as programmable timers, a FIFO, specific counters, etc., necessary for the operation of the digital system are also designed and maintained the same. They are not discussed here since they are beyond the focus of this case study and do not affect the state machine performance nor its implementation.

With these challenges in mind, an appropriate state machine is developed. One important item to note is that with a significantly faster system clock than what the protocol is using, many states will have a significant amount of idle time prior to transitioning to the next state. This idle time must be managed such that no state machine movement occurs. Figure 6 shows the state diagram of the developed state machine used for controlling the serial protocol mentioned above. Each state manages particular signals in order to accurately depict the protocol management of nRTS_O and STS_O.

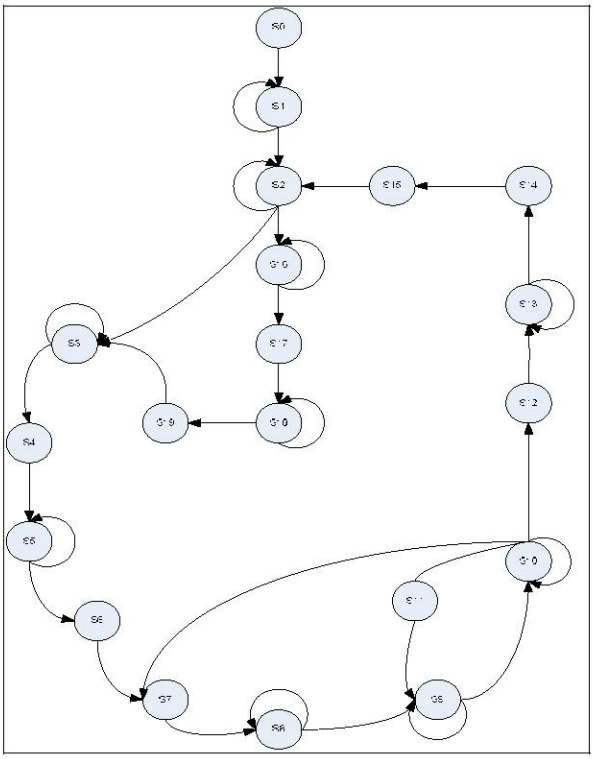

Figure 6. Case study state machine

VI. RESULTS

In order to analyze the differences between coding styles, data is collected from various reports generated by the Xilinx tools at several stages of the process as shown in Figure 7.

- **Synthesis Report:** This report provides detailed information as to how the HDL code is synthesized. It provides coarse device utilization information such as the number of flip-flops, LUTs, etc. utilized in the design. It also provides a coarse estimate as to what the theoretical maximum system clock speed the FSM can operate at. Note that synthesizers assume unlimited device resources and ideal routing availability.
- **Map Report:** This report provides accurate device utilization information without timing estimates. This step takes into account device architecture, slice structure and CLB information.

Place & Route Report: Upon completion of the place and route step, accurate timing information is provided. This step in the development cycle has taken into account pin locations on the device, internal routing delays, and CLB locations internal to the device as well as the physical architecture of the device.

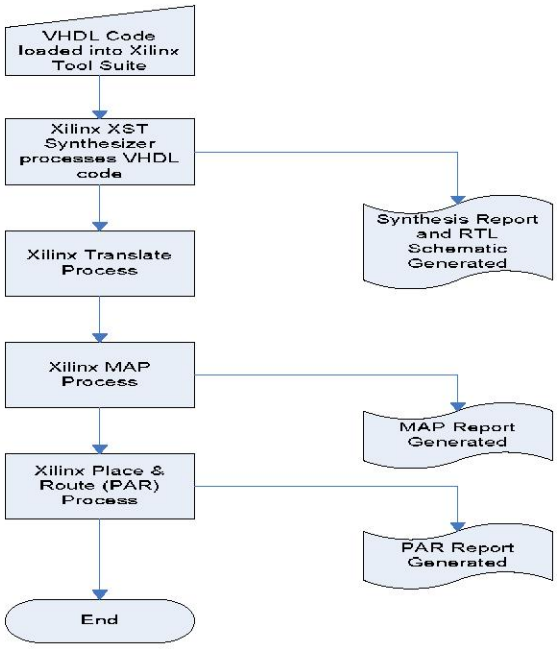

Figure 7. Xilinx tool suite process flow

A. *Synthesis Utilization*

Synthesis results are a key indicator of how the VHDL code is inferred into the FPGA fabric. As indicated above, the synthesizer assumes the best possible conditions, i.e. unlimited routing, signal fan-out, and logic resources. With this in mind, synthesis results are coarse compared to the actual resources used after further processing.

Key resource usage indicators are found in the synthesis report provided by the tools. These key indicators are total slice count, total slice flip-flops used, and total 4-input LUTs used. Table 1 summarizes the data extracted from the synthesis reports. Figure 8 presents these four indicators as a percentage of change from the median value of each category, allowing us to view the data in a manner illustrating how these indicators relate to each other.

Important items concerning the synthesis results are:

- Less than +/- 4% variation from the median values of total slices and 4-Input LUTs were evident across all combinations of coding styles and state assignment encoding.
- Total slice usage is driven by the total number of 4-Input LUTs needed, rather than flip-flops.

- Fewer flip-flops were needed when the state assignment encoding method was Gray, but in using this method the number of LUTs needed increased.

TABLE I. SYNTHESIS UTILIZATION RESULTS

	One Hot Encoding			Gray Encoding		
	CSP	*SCO*	*SRO*	*CSP*	*SCO*	*SRO*
Slices	244	242	238	248	243	247
Slice FF	90	77	89	79	66	78
4 input LUT	467	464	451	477	467	473

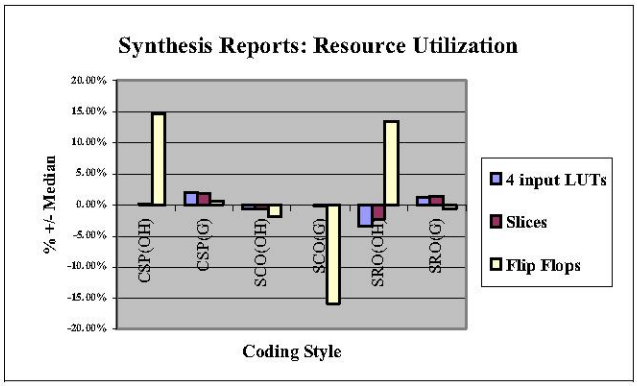

Figure 8. Synthesis resource utilization

B. *MAP Utilization*

MAP report results are similar to those found in the synthesis report, namely total slice count, total slice flip-flops used, and total 4-input LUTs used. Table 2 provides the data extracted from the MAP reports and Figure 9 presents these four indicators as a percentage of change from the median value of each category, allowing us to view the data in a manner illustrating how each indicator relates to each other.

Important items concerning the MAP results are:

- Less than +/- 4% variation from the median values of Total Slices and 4-Input LUTs is evident across all combinations of coding styles and state assignment encoding.
- When silicon architecture is taken into consideration, the resource numbers do change, but they also track the synthesis numbers.

TABLE II. MAP UTILIZATION RESULTS

	One Hot Encoding			Gray Encoding		
	CSP	*SCO*	*SRO*	*CSP*	*SCO*	*SRO*
Slices	284	282	278	286	282	287
Slice FF	80	73	83	69	62	70
4 input LUT	427	425	412	437	428	434

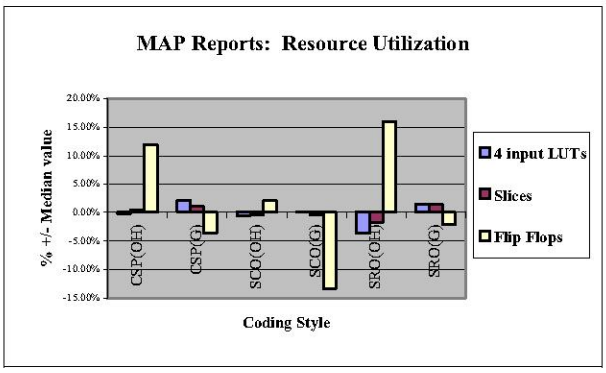

Figure 9. MAP resource utilization

C. *Place and Route (PAR) Results*

It is desired to evaluate performance based on how the logic is inferred into the device. An additional variable, clock period, is selected, added and varied, allowing the performance of the system to be affected.

By setting the system clock, net CLK_I, to a particular period length as a global constraint, the PAR process attempts to ensure that it will work at the associated frequency. In a synchronous design, the global constraint forces the PAR algorithms to try and route signals such that the time between the output of a FF to the input of the next FF is equal to or less than this constraint. A range of timing constraints was chosen such that the PAR process would continually improve and try to route the logic better. The constraint value varies between 8.5 ns to 30 ns.

Table 3 provides the timing results for the three coding styles using One-Hot state machine encoding. All values are in nano-seconds. The initial constraint of 30 ns provided a true starting point for the range of values. With the constraint set at 30.00 ns, the PAR process was able to produce results significantly better than what the constraint desired. Each subsequent constraint value was chosen to improve the previously obtained results. As the constraint continued to get smaller, the PAR process was pushed to improve its results until the constraint could not be met. Figure 10 is a graphical representation of the data shown in Table 3. All of the coding styles essentially had the same performance. The last constraint value where all coding styles met the constraint was 9.00 ns.

TABLE III. ONE-HOT ENCODING PAR TIMING

Constraint	*CSP*	*SCO*	*SRO*
30.00	14.78	13.52	15.32
16.00	13.09	12.47	12.19
14.00	10.76	10.51	11.94
12.00	11.11	11.37	11.27
10.00	9.94	9.94	9.99
9.00	8.98	8.84	8.92
8.50	8.78	8.68	9.37

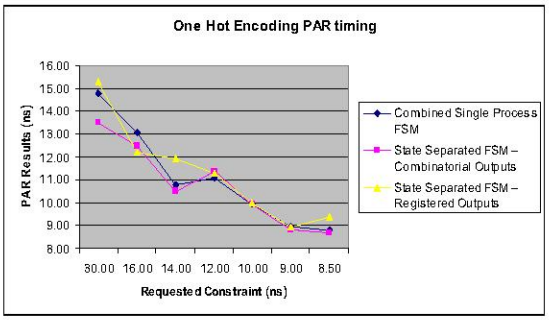

Figure 10. PAR timing for One-Hot encoding

Table 4 provides the timing results for the three coding styles using Gray encoding. All values are in nanoseconds using the same constraint as in the One-Hot encoding method.

Unlike the One-Hot encoding, there was more variance in the results between the three coding styles. Figure 11 utilizes the data from Table 4 to illustrate the performance for each coding style using Gray encoding. When comparing the two results, the State Separated Registered Outputs coding style stopped meeting the constraint at 10.00 ns. The Combined Single Process coding style continued meeting the constraint until it was 8.50 ns. Hence, the Combined Single Process coding style performed better.

TABLE IV. PAR TIMING FOR GRAY ENCODING

Constraint	*CSP*	*SCO*	*SRO*
30.00	15.56	14.06	13.68
16.00	12.02	12.07	13.42
14.00	10.93	11.33	13.41
12.00	10.49	10.47	11.82
10.00	9.63	10.02	11.10
9.00	8.99	9.13	11.44
8.50	8.72	10.31	12.27

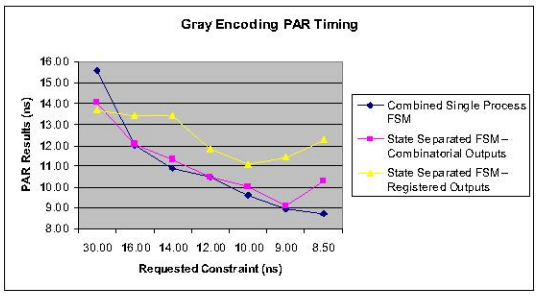

Figure 11. PAR timing for Gray encoding

VII. CONCLUSIONS

Resource utilization for any HDL design implemented in FPGAs is an important factor that can affect design performance. Resources are divided into two groups, combinatorial, in terms of LUTS and slice counts, and sequential in terms of flip-flops. Analyzing the results we can conclude that the State Separated Registered Outputs (SRO) coding style utilized fewer LUTs and overall slices than the other two coding styles with a slight increase in the number of flip-flips. Since it is always recommended that FPGA outputs be registered, this increase in flip-flop count is necessary and can be considered as a design requirement.

Performance is subjective in nature. With regards to this study, the determination as to which coding style performs the best is based on how fast the system clock can operate without adverse affects on the overall function. With this in mind, performance variations among the coding styles were observed as follows: The State Separated Combinatorial Outputs (SCO) implemented using One-Hot encoding, achieved the best performance with a system clock of 8.68 ns, as indicated in Table 3. This method is not a recommended implementation due to the outputs being driven by combinatorial logic. Therefore, the Combined Single Process (CSP) implemented using Grey encoding, achieved the best performance with a system clock of 8.72 ns, as indicated in Table 4. This method is the recommended implementation due to the outputs being driven by registered logic.

VIII. REFERENCES

[1] IEEE Standard VHDL Reference Manual, IEEE *Standard 1076, 1987 and 2001.*

[2] H. Sutter, and A. Alexanderscu, C++ Coding Standards, *Addison Wesly, 2005.*

[3] P. Robinson, Tai-Chi Lee and E. Henne, Erik, "Framework for executing VHDL code on FPGA", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'04*, v3, *PDPTA'04*, 2004, p 1296-1299.

[4] K. Kuusilinna, V.Lahtinen, and J.Saarinen, "Finite State Machine Encoding for VHDL Synthesis," *IEE Proc.-Comput. Digit. Tech*, Vol 148, No. 1, January 2001.

[5] Ben. Cohen, *VHDL Coding Styles and Methodologies,* Second Edition, Kluwer Academic Publishers, Boston, MA, 1999

[6] Peter J. Ashenden, *The Designer's Guide to VHDL,* Second Edition, San Francisco, CA, Morgan Kaufmann Publishers, 2001.

[7] Wen-Tsong Shiue "Novel state minimization and state assignment in finite state machine designs", *Integration, the VLSI Journal*, v 38, n 4, April, 2005, p 549-570.

[8] Stephen L. Wasson, "High-Speed State Machine Design", Integrated *System Design Magazine*, July 1995.

[9] Richard Sandige, "Synchronous State Machine design methodologies with VHDL and implementations using CAD tools", *Proceedings of the IASTED International Conference on Modeling, Simulation, and Optimization*, 2003, p 29-32

[10] ISE5.2 software manual, Xilinx Inc., http://toolbox.xilinx.com/docsan/xilinx5/pdf/ manuals.pdf

[11] Spartan-IIE data sheet, *ds077.pdf, Xilinx inc, www.xilinx.com*

[12] Modelsim XE 5.6b User's guide and software manuals, Mode Technology, http://www.model.com/fpga/documents/ xilinxfnd.pdfModelsim