EE 595 FAD

State Machine Encoding Techniques
Synthesis

I. Introduction

There are several methods of encoding the state assignments when designing synchronous finite state machines (FSM). The decision on which method to use is a function of design constraints such as speed, area, power consumption, and the type of programmable logic device targeted to. This is an important decision to be made by the designer in order to satisfy the design requirements. There are some general guidelines one can use depending on design objectives. A wrong decision may result in a state machine that uses too much logic, too slow, or both.

Most research reports and other materials devoted to searching of the "optimal" coding of the internal states are based on the minimal number of internal states and sometimes also on the minimal number of flip-flops used in the hardware implementation. The only viable method as how to get the truly optimal results is to test for all possibilities [2]. Notice that availability of several synthesis tools for CPLDs and FPGAs have made it easier for designers to test all possible state encodings in order to satisfy all system requirement.

This document deals with some of the state encoding techniques used in synchronous finite state machine (FSM) design using VHDL synthesis tools; namely, the One-Hot Code, Binary/Sequential Code, and Gray Code state assignment. An example of state machine is implemented with VDHL using these three encoding schemes and the results are compared in terms of the circuit size (gate count and number of flip-flops) and performance. The synthesis engine used was FPGA Express integrated with XILINX FOUNDATION Series 2.1i.

II. One-Hot Encoding

In the one-hot encoding (OHE) only one bit of the state variable is "1" or "hot" for any given state. All other state bits are zero. (See Table 1) Therefore, one flip-flop (register) is used for every state in the machine i.e. n states uses n flip-flops. Using one-hot encoding, the next-state equations can be derived easily from state diagrams. State decoding is simplified, since the state bits themselves can be used directly to indicate whether the machine is in a particular state. In addition, with a one-hot encoded state machine, the inputs to the state bits are often simply the functions of other state bits [1]. Often times, no state decoding is necessary, and state encoding can only require the OR-ing of state bits.

	State Variables			
State	**One-Hot Code**	**Binary Code**	**Gray Code**	
S0	00001	000	000	
S1	00010	001	001	
S2	00100	010	011	
S3	01000	011	010	
S4	10000	100	110	

Table 1:An example of state Encoding for a 4 state Machine

IIa. Why use One Hot Code?

One-hot encoding (OHE) is better suited for use with the fan-in limited and flip-flop-rich architectures of the higher gate count filed-programmable gate arrays (FPGAs), such as offered by Xilinx, Actel, and others. OHE maps very easily in these architectures. [1] This is because OHE requires a larger number of flip-flops. It offers a simple and easy-to-use method of generating performance optimized state-machine designs because there are few levels of logic between flip-flops. One-hot state machines are typically faster. Speed is independent of the number of states, and instead depends only on the number of transitions into a particular state. A highly encoded machine may slow dramatically as more states are added. In such cases, one does not have to necessarily worry about finding an "optimal" state (assignment) encoding. This is particularly beneficial as the machine design is modified. What is "optimal" for one design may no longer be best (optimal) if a few states are added and some states are changed. One-hot is equally "optimal" for all machines. One-hot coded machines are easy to design. Schematics can be captured and HDL code can be written directly from the state diagram without first requiring the generation of a state table. Finally, they are easily synthesize-able using VHDL or Verilog.

In some cases, the one-hot method may not be the best encoding technique for a state machine implemented in an FPGA. For example, if the number of states is small, the speed advantages of using the minimum amount of combinatorial logic may be offset by delays resulting from inefficient CLB (configurable logic blocks) use, e.g. a Xilinx device. This assignment allows the designer to create state machine implementations that are more efficient in FPGA architectures in terms of area and logic depth (speed). FPGA have plenty of registers but the LUTs are limited to few bits wide. One-hot increases the flip-flop usage (one per state) and decreases the width of combinatorial logic. It makes it easy to decode the next state, resulting in large FSMs. [5] And finally, since one hot code state assignment reduces the area (area optimization) by using less logic gates, it consumes less power.

III. Binary Encoding

In a binary encoding scheme, the relationship between the number of state variables (q) and number of states (n) is given by the equation:

$$q=\log_2(n)$$

With this formula, one can easily determine the minimum number of state variables required for a binary-encoded state machine. Clearly, the number of flip-flops used is equal to the number of state variables (q). In this technique, the states are assigned in binary sequence where the states are numbered starting from binary 0 and up.

For example, in a 4-state machine, the state assignment is done as shown below;

	Y1	Y0
State1=	0	0
State2=	0	1
State3=	1	0
State4=	1	1

…
…

IIIa. Why binary encoding?

Binary encoding uses fewer flip-flops/registers than one-hot encoding. For example, binary encoding requires only seven (flip-flops) registers to implement a 100-state machine while a one-hot encoding needs 100 flip-flops. Binary encoding uses the minimum number of state variables (flip-flops) to encode a machine since the flip-flops are maximally utilized. As a result, it generally increases the amount of combinatorial logic because more combinatorial logic is required to decode each state. [4] Therefore, binary encoding is implemented more efficiently when using PLAs and CPLDs. These devices have wider gates and a large amount of combinatorial logic per register. It is usually the preferred method when implementing machines that are fewer than 8 states. Their wide 'AND-architecture' allows any number of state variable (bits)to be included in each product term with no speed (or area) penalty. [1]

The disadvantages of using a binary encoded FSM include the fact that more than one bit can flip at any time and can result in a glitch (hazard) specially in design of counters. It also requires a more complex decoding logic to determine the present state. If power consumption is an issue, then this technique may not be suitable since the more registers/flip-flop changes, the more power the device consumes.

IV.Gray Code Assignment

Gray code assignment is a state assignment in which consecutive states codes only differ by one bit (adjacent).

In other words, only one flip-flop changes at a time when changing consecutive states.

In this encoding, the number of flip-flops (register) used is also determined by:

$$\text{Number of state variables} = \text{number of flip-flops} = \log_2 (\text{number of states})$$

An example of a 3-bit gray code:

	Y3	Y2	Y1
State1=	0	0	0
State2=	0	0	1
State3=	0	1	1
State4=	0	1	0
State5=	1	1	0

…

…

IVa. Why gray code?

Gray code uses the same number of register (flip-flops) as the binary coding technique and the decoding logic can also be as complex if not more. Therefore, gray code assignment is also ideal for PLA and CPLD applications since they have wide gates and a large amount of combinatorial logic per register. [4] Gray code is highly recommended in PLA and CPLD applications when designing for low power requirement. One approach to low power design is to choose a state assignment that diminishes (minimizes) the switching activity of state transitions. The ideal case would be if only one state variable changes in each possible transition. Gray encoding has minimal switching between consecutive states. After choosing the gray state (assignment) encoding, then further improvements can be done in order to reduce the area of the combinatorial logic involved and thus minimizing power consumption. The resulting code determines the register configuration as well as size and structure of the combinational circuit.

In general, the performance of a highly encoded state machine (e.g. Gray or Binary encoded) implemented in an FPGA device drops as the number of states grows because of the wider and deeper decoding that is required for each additional state. CPLDs are less sensitive to this problem because they allow a higher fan-in. [3] [4].

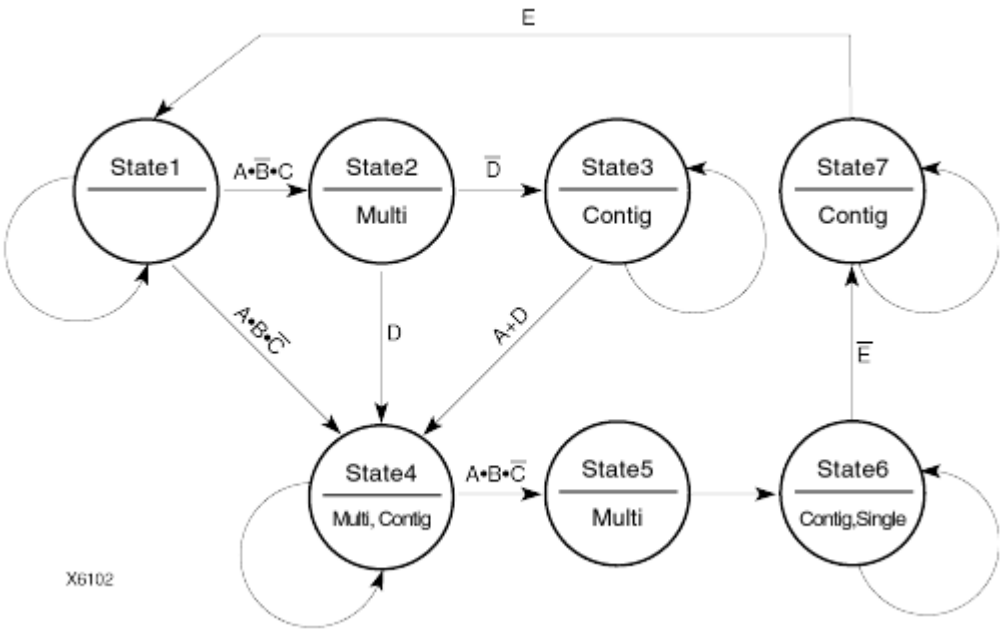
V. Experiments with different FSM VHDL codes

In this section, an example of a finite state machine is synthesized using Xilinx Foundation 2.1i with VHDL codes. The machine was coded using Binary, Gray, and One-Hot encoding. In each instance, the code is synthesized targeting an FPGA (XILINX Virtex V150FG256) and a CPLD (XILINX XC9500 95108PC84) device. In addition, the synthesis tool is set to optimize for area only. (Note: the version that is used had some software problems with the simulator; therefore, optimizing for speed could not be realized if I could not simulate the circuit!)

The objectives in this case, were;

(1) Generate the circuit using the same VHDL code for each state encoding
(2) Synthesize each encoding technique using an FPGA and a CPLD
(3) Compare the gate and logic counts for each implementation.

Example finite state machine: (Moore machine)



Using the above example of a finite state machine, the following sets of test data/report were compiled.

Each set contains : (i) the vhdl code listing
(ii) the schematic generated when CPLD was used
(iii) the post-synthesis report when CPLD was used
(iv) the schematic generated when an FPGA was used
(v) the post-synthesis report when an FPGA was used

A. Binary Encoding State Assignment:

	DEVICE		
COMPONENTS USED	VIRTEX V150FG256 (FPGA)	XILINX XC9500 (CPLD)	
NO. OF FLIP-FLOPS	3	3	
NO. OF OTHER GATES	13	49	
NO. OF LUT'S	19	N/A	

Table II: Synthesis Results – Hardware Implementation

B. Gray Encoding State Assignment:

	DEVICE		
COMPONENTS USED	VIRTEX V150FG256 (FPGA)	XILINX XC9500 (CPLD)	
NO. OF FLIP-FLOPS	3	3	
NO. OF OTHER GATES	13	45	
NO. OF LUT'S	15	N/A	

C. One-Hot State Assignment:

	DEVICE		
COMPONENTS USED	VIRTEX V150FG256 (FPGA)	XILINX XC9500 (CPLD)	
NO. OF FLIP-FLOPS	7	7	
NO. OF OTHER GATES	17	49	
NO. OF LUT'S	13	N/A	

VI. Conclusion

Ideally, a good state machine design must optimize the amount of combinatorial logic, the number of fan-ins to each flip-flop, the number of flip-flops (registers), and the propagation delay between registers. However, these factors are interrelated, and compromises between them may be necessary. For example, to increase speed, levels of logic must be reduced. However, fewer levels of logic result in wider combinatorial logic, creating a higher fan-ins than can be efficiently implemented given the limited number of fan-ins allowed by the FPGA architecture. Another issue is the design for low power applications. The minimization of state (flip-flops) switching activity and area minimization are just some of the issues encountered when designing for low power. [6]

The availability of different synthesis tools for programmable logic devices such as FPGA and CPLD have made it easier for designers to realize their designs. By simply changing some parameters in their codes and design constraints on the synthesis tool, they can experiment with different circuit designs to get different results in order to satsfy their system requirements.

References

[1] Golson, Steve, "One-hot state machine design for FPGA's", Trilobyte Systems, 3rd PLD Design Conference, Santa Clara, CA, March 30, 1993.

[2] Kubatova, Hana, "Implementation of the FSM into FPGA", The International Workshop of Discrete-Event System Design, Pryztok, Poland, June 27-29,2001.

[3] Application Brief No. 131, Altera Corporation, May 1994, version 1.

[4] Foundation Series 2.1i User's Guide, Xilinx, Inc.

[5] Sutter, G., Todorovich, E., Lopez-Buedo, S., and Boemo, E., "Low-Power FSMs in FPGA: Encoding Alternatives", INCA, Universidad Nacional del Centro, Tandil, Argentina and Computer Engineering School, Universidad Autonoma de Madrid, Spain.

[6] Koegst, M., Franke, G., and Feske, K., "State Assignment for FSM Low Power Design", FhG IIS Erlangen – Department EAS Dresden, EURO-DAC '96 with EURO-VHDL '96.