Designing State Machines for FPGAs

Introduction

The traditional methodology for designing state machines has been to draw a state diagram, map the states into the minimum number of register bits, and determine the next state function for each register bit. The minimum number of register bits needed can be determined by rounding up the natural log of the number of states. This methodology results in a minimum number of registers but usually requires wide gating and complicated logic to encode the next state bit. This scheme is necessary to implement state machines using programmable logic devices (PLDs) because of the PLDs inherent lack of registers. Because FPGAs do not have this limitation, other approaches are used for efficient state machines.

This application note will discuss techniques for efficiently implementing state machines by using one-hot-encoding for Actel FPGAs.

Sample One-Hot-Encoding

One-hot-encoding is an effective approach because it takes advantage of the abundance of registers. The state diagram of a sample state machine is illustrated in Figure 1. This state machine is the control section of a four-channel DMA controller for Actel FPGAs. The state machine contains six states, seven inputs, and five outputs. Each circle represents a different state, and each arrow represents a transition between states. Inputs that cause state transitions are listed adjacent to the state transition arrows. Control outputs are labeled along with the states inside the circles. For example, in state S2, the state machine asserts the CNTD and CMREQ outputs. If the MACK input is low, the state machine remains at state S2. If the MACK input is high, the state machine goes to state S3 and asserts the CE output in the process.

State Transition Equations

The next step is determining the logic to generate the state sequence. For one-hot-encoding implementations, assign each state to a separate register and write a state transition equation for each register. The state diagram in Figure 1 shows that state S0 can be realized when state S4 is asserted and the input CONT is low, or it remains at state S0 if all four inputs—A, B, C, and D—are low. Therefore, the transition equation of state S0 can be written as

S0 := /A*/B*/C*/D*S0 + /CONT*S4

Similarly, state S1 can be achieved when state S0 is asserted and any one of the four inputs—A, B, C, or D—is high, or it remains at the same state (S1) if the input PBGNT is low. The transition equation of state 1 can be derived as

S1 := (A+B+C+D)*S0 + /PBGNT*S1

The complete state transition equations for the state machine are listed in Table 1. Note that state S3 will go to state S4 unconditionally; therefore, the transition equation for state S3 can be written as S3 := S4.

Table 1 • *State Machine Transition Equations*

S0 := /A*/B*/C*/D*S0 + /CONT*S4;
S1 := (A+B+C+D)*S0 + /PBGNT*S1;
S2 := PBGNT*S1 + /MACK*S2;
S3 := /MACK*S2 + MACK*S3;
S4 := S3;
S5 := CONT*S4 + /MACK*S5;

Output Equations

Once the state transition equations are determined, the output equations can be written by encoding the states. In some cases, the output is active only in one state. Therefore, the output is simply a function of that state. For example, CE is active only in state S3, so the output equation for CE is simply CE = S3. Other output may be active in more than one state. The output equations can be written simply as functions of those states. For example, CMREQ is active in state S2 as well as state S5. Therefore, the output equation is

CMREQ = S2 + S5

Table 2 lists the output equations of the state machine.

Table 2 • *Output Equations*

PBREQ = S1;
CLD = S4;
CNTLD = S2;
CMREQ = S2 + S5;
CE = S3;

The state machine can be captured using schematic entry, or it can be automatically mapped by using synthesis tools such as Actel ACTmap VHDL Synthesis. The completed VHDL file is shown in Figure 2. A reset signal RST has been added to initialize the circuit.

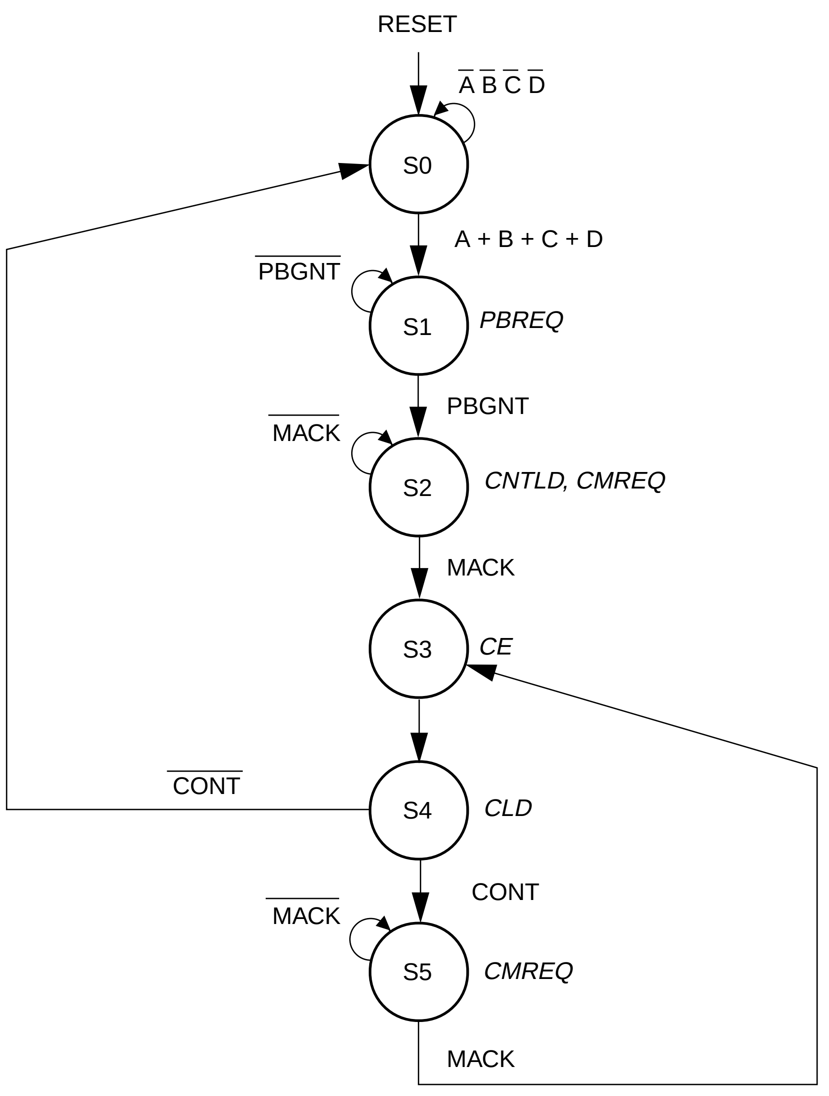

Figure 1 • *Four-Channel DMA Controller State Diagram (Control Section)*

Figure 3 shows the schematic of the state machine implementation using the one-hot-encoding approach in the ACT 3 family. The circuit requires 18 logic modules after combining timing for the A1425A-2, as shown in Figure 4.

Summary

A summary of the bit-per-state methodology is as follows:

1. Draw a state diagram.
2. Assign each state to a separate register.
3. Write a state transition equation for each register.
4. Derive output equations based on active states.

Larger state machines can be implemented using this technique by distributing control to several smaller state machines and using a single master machine to coordinate activities among the state machines. This usually results in higher-performance designs. It is also easier to design and debug simpler and smaller state machines.

Shift Register Design

Shift registers with serially controlled data inputs and parallel outputs can be used as powerful controlled sequence generators. As a result, shift registers can be used in high-speed state machine designs.

For this application, a shift register sequentially shifts a single logic high signal while the rest of the output bits are low. This function can be implemented by simply connecting the output of the one flip-flop to the input of the next flip-flop and the output of the last flip-flop back to the input of the

```
--library ieee;
--use ieee.std_logic_1164.all;
entity DMASM is
        port (A,B,C,D:  in bit;
              PBGNT, MACK, CONT :  in bit;
              RST, CLK :  in bit;
              PBREQ, CMREQ, CE, CNTLD, CLD :  out bit);
        end DMASM;
architecture BEHAVE of DMASM is
        type STATE is (S0, S1, S2, S3, S4, S5);
        signal CURRENT_STATE, NEXT_STATE: STATE;
begin
SEQ: process (RST, CLK)
begin
        if (RST = '0') then
                CURRENT_STATE <= S0;
        elsif (CLK' event and CLK = '1' ) then
                CURRENT_STATE <= NEXT_STATE;
        end if;
end process;
COMB: process (CURRENT_STATE, A, B, C, D, PBGNT, MACK, CONT)
begin
  PBREQ <= '0';
  CMREQ <= '0';
  CE    <= '0';
  CNTLD <= '0';
  CLD   <= '0';
case CURRENT_STATE is
    when S0 =>
        if (A = '1' or B = '1' or C = '1' or D = '1') then
        NEXT_STATE <= S1;
        else
        NEXT_STATE <= S0;
        end if;
    when S1 => PBREQ <= '1';
        if (PBGNT = '1') then
        NEXT_STATE <= S2;
        else
        NEXT_STATE <= S1;
        end if;
    when S2 => CNTLD <= '1'; CMREQ <= '1';
        if (MACK = '1') then
        NEXT_STATE <= S3;
        else
        NEXT_STATE <= S2;
        end if;
    when S3 => CE <= '1';
        NEXT_STATE <= S4;
    when S4 => CLD <= '1';
        if (CONT = '1') then
        NEXT_STATE <= S5;
        else
        NEXT_STATE <= S0;
        end if;
    when S5 => CMREQ <= '1';
        if (MACK = '1') then
        NEXT_STATE <= S3;
        else
        NEXT_STATE <= S5;
        end if;
    end case;
  end process;
end BEHAVE;
```

Figure 2 • *Complete State Machine VHDL file*

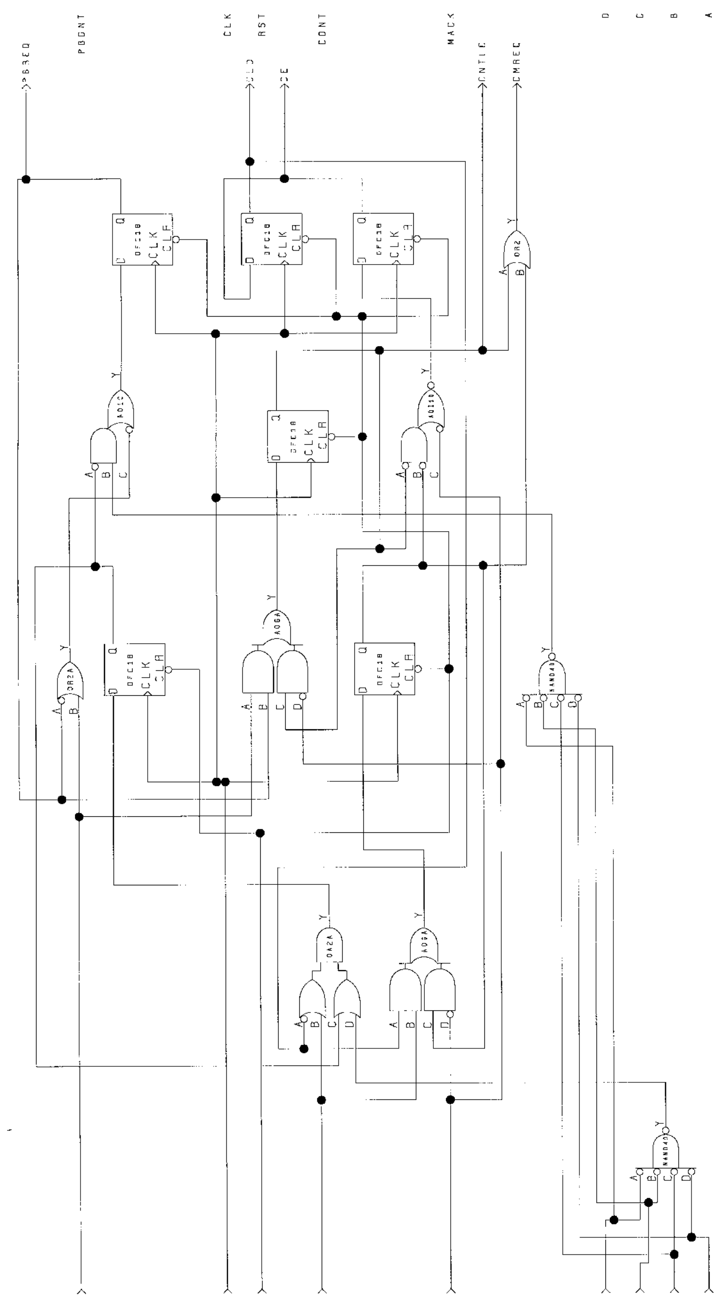

Figure 3 • *State Machine Schematic in ACT 3 Family*

Worst-Case Path (from A, B, C, D Registered Inputs)

$$= \overbrace{tCO + tRD2}^{\text{DF1}} + \overbrace{tPD + tRD1}^{\text{NAND4D}} + \overbrace{tPD + tRD1}^{\text{AOI}} + tSUD$$

$$= 2.3 + 1.4 + 2.3 + 1.0 + 2.3 + 1.0 + 0.6$$

$$= 10.9 \text{ ns}$$

Figure 4 • *State Machine Timing Using A1425A-2* first flip-flop. Table 3 shows the function table of an 8-bit serial shift register. The width of the shift register is expandable by serially adding more flip-flops to the last stage. Figure 5 shows the schematic of an 8-bit shift register. Note that the shift register is designed so that it can be set to a known state with a reset signal.

Table 3 • *8-Bit Shift Register Function Table*

RST	CLK	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	X	1	0	0	0	0	0	0	0
1	≠	0	1	0	0	0	0	0	0
1	≠	0	0	1	0	0	0	0	0
1	≠	0	0	0	1	0	0	0	0
1	≠	0	0	0	0	1	0	0	0
1	≠	0	0	0	0	0	1	0	0
1	≠	0	0	0	0	0	0	1	0
1	≠	0	0	0	0	0	0	0	1
1	≠	1	0	0	0	0	0	0	0

State Machine Implementation

The state machine implementation is best illustrated by an example. The sample state machine has six states with three output bits. The sequence is organized such that only one output bit changes state for every clock pulse. Figure 6 shows the state diagram of the state machine.

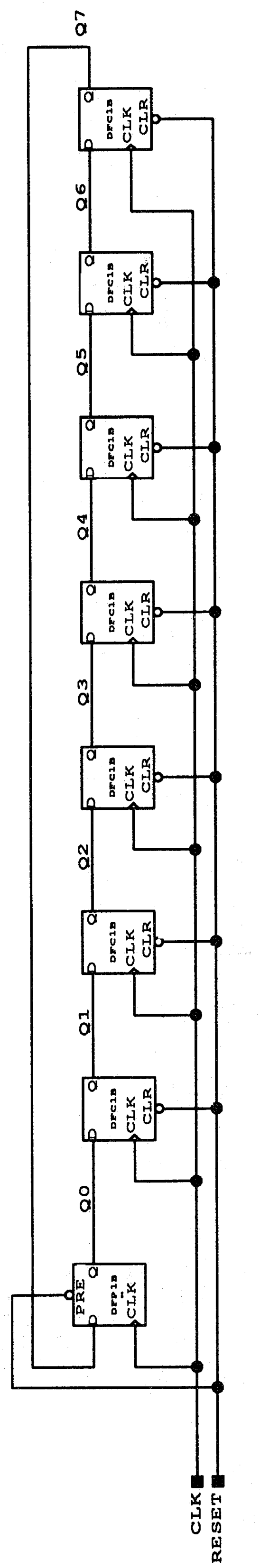

Figure 5 • *Eight-Bit Shift Register Schematic*

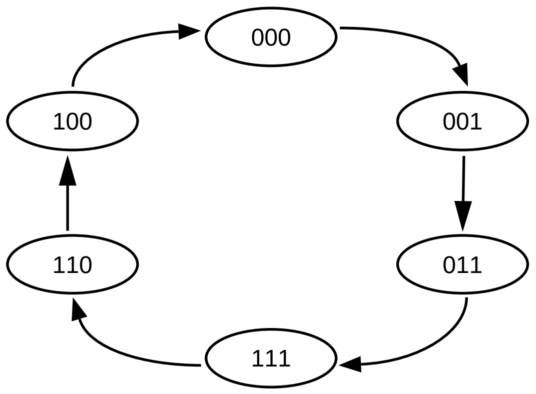

Figure 6 • *State Diagram of State Machine*

A six-state state machine requires a six-bit shift register with one register per state. Using this shift register determines the state sequence map of the state machine. The state machine creates the three output bits based on decoding the outputs of the shift register. The decoding logic is greatly minimized because there is only one output bit asserted in any state. Table 4 shows the state table of the state machine. When the outputs of the state machine are 001, the shift register outputs are 000010. In this state, Q5, Q4, Q3, Q2, and Q0 are all low and Q1 is high. Therefore, the state machine uses only Q1 for the decoding logic. The logic for the state machine outputs is based on the shift register outputs. Out0 is high when Q1 or Q2 or Q3 is high, Out1 is high when Q2 or Q3 or Q4 is high, and Out2 is high when Q3 or Q4 or Q5 is high. The complete decoding logic equations are the following:

Out0: Q1 + Q2 + Q3

Out1: Q2 + Q3 + Q4

Out2: Q3 + Q4 + Q5

Table 4 • *State Table for Example State Machine*

Shift Register Outputs						State Machine Outputs		
Q5	Q4	Q3	Q2	Q1	Q0	Out2	Out1	Out0
0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	1
0	1	0	0	0	0	1	1	0
1	0	0	0	0	0	1	0	0

Figure 7 shows the schematic diagram of the state machine using ACT 2 or ACT 3 macros. Note that the decoding logic requires only one level of logic to implement. Thus, using a shift register to implement state machines improves performance significantly.

Conclusion

To achieve the high performance possible with FPGAs, new register-rich techniques are needed. Two effective approaches include bit-per-state and shift register decoding. The user can manually implement these techniques or select FPGA-friendly algorithms with optimization tools such as the Actel ACTmap VHDL Synthesis tool.

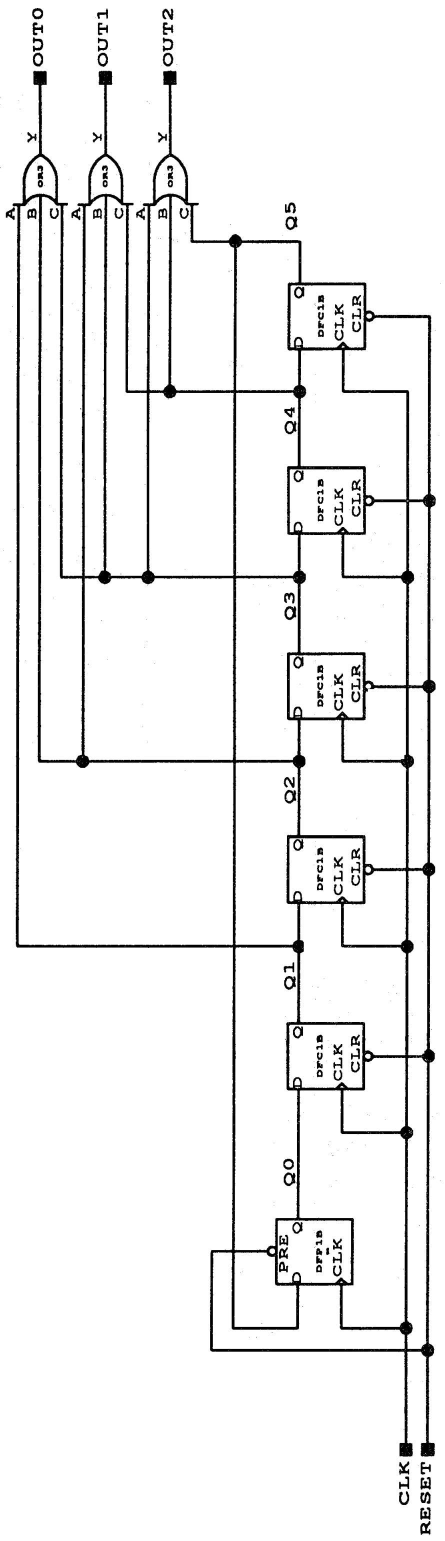

Figure 7 • *State Machine Schematic*