Synthesizable VHDL Design for FPGAs

Eduardo Augusto Bezerra
Djones Vinicius Lettnin

Universidade Federal de Santa Catarina
Florianópolis, Brazil

August 2013

Contents

Chapter 8. Using Finite State Machines as Controllers

In the previous chapter, FSMs have been used in the design of counters. This is a good example of FSM usage, where each state of the FSM provides as output a counting value. The present chapter introduces the design of FSMs targeting one of their main applications, which is the control of a sequence of events. At the end of the chapter, the students should be able:

- to understand the basic concepts of datapath and control unit;
- to design the FSM of a control unit;
- to implement a control module for the case study (calculator);
- to simulate and test the case study - calculator with control module;
- to prototype the case study using an FPGA board.

8.1 Designing an FSM Based Control Unit

As discussed in Chapter 1, and shown in Figure 1.1, digital systems in general have a module responsible for performing the processing, and another module used to control the sequence of operations. Figure 8.1 shows three examples of typical digital systems where there is a control unit and a datapath. In each design there is a control unit sending a sequence of "commands" to the operational module.

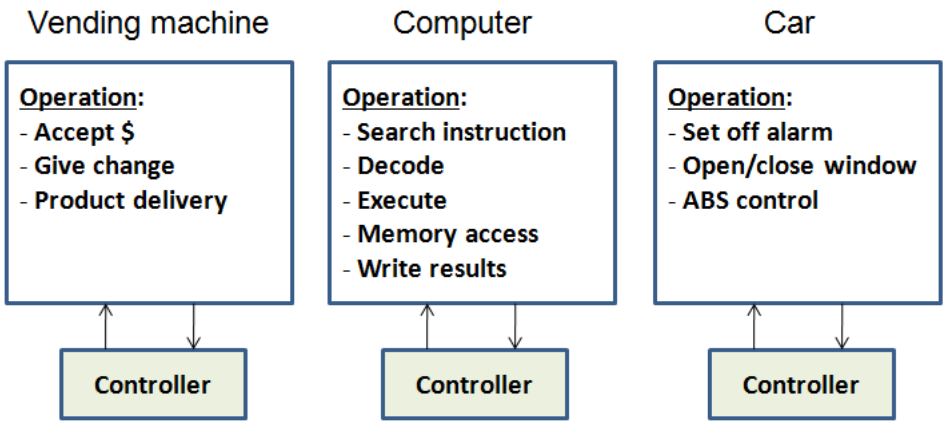

Figure 8.1. Simulation results showing the delay states of the FSM.

In Figure 8.1, FSMs can be used in the design of the control circuits. For instance, the Vending Machine controller has to perform all the necessary steps in order to deliver (or not) the selected product to the user. The controller's FSM should be designed con-

sidering that the operational part of the system has to perform the activities in a sequential order (i.e. payment selection; product selection; delivery product; …). The FSMs for the remaining controllers in Figure 8.1 should be designed following similar principles, always considering a very specific sequence of events to be performed.

Strategies for describing FSMs in VHDL have been discussed in Chapter 7. However, before writing the VHDL implementation for an FSM, the target circuit has to be properly designed. In Chapter 7, the FSM for the proposed counter case study has been written in VHDL in an empirical way, without following any design methodology.

In the present chapter, a controller for a new case study is implemented in VHDL, but this time a proper design methodology is followed.

The methodology adopted for the design of a controller in VHDL has the following steps:

STEP 1: Write a system description as complete as possible (requirements);

STEP 2: Prepare a graphical representation for the FSM, making as many versions as needed;

STEP 3: Write down a state transition table for the FSM, listing the inputs and outputs, including all the states (current and next);

STEP 4: Describe in VHDL the modeled FSM behavior, using the graph and the truth table as a guide, and following the examples provided in Chapter 7.

In the classical FSM controller design methodology used in digital systems in general (not in HDL based designs), the following additional steps are required:

STEP 5: Define the target circuit architecture which is, usually, represented by the block diagram in Figure 1.3;

STEP 6: Define a unique identifier (coding strategy) for each state;

STEP 7: Make the necessary changes in the truth table, in order to have only binary numbers (replace the state symbolic identifiers by the binary codes);

STEP 8: From the truth table, obtain the Boolean equations, and design the circuit for the combinational component of Figure 1.3.

It is important to notice that these additional steps are not used in a VHDL based design, as they are automatically performed by the synthesis tool. The synthesis tool not only extracts the required combinational logic from the VHDL FSM description, but also performs very efficient optimizations.

Next, a case study is presented and implemented in VHDL in order to demonstrate the FSM design methodology. The additional steps discussed are also performed for the case study, demonstrating the classical approach for the design of a FSM based digital system controller.

8.2 Case Study: Designing a Vending Machine Controller

In this case study you are asked to implement a digital controller for the vending machine shown in Figure 8.2.

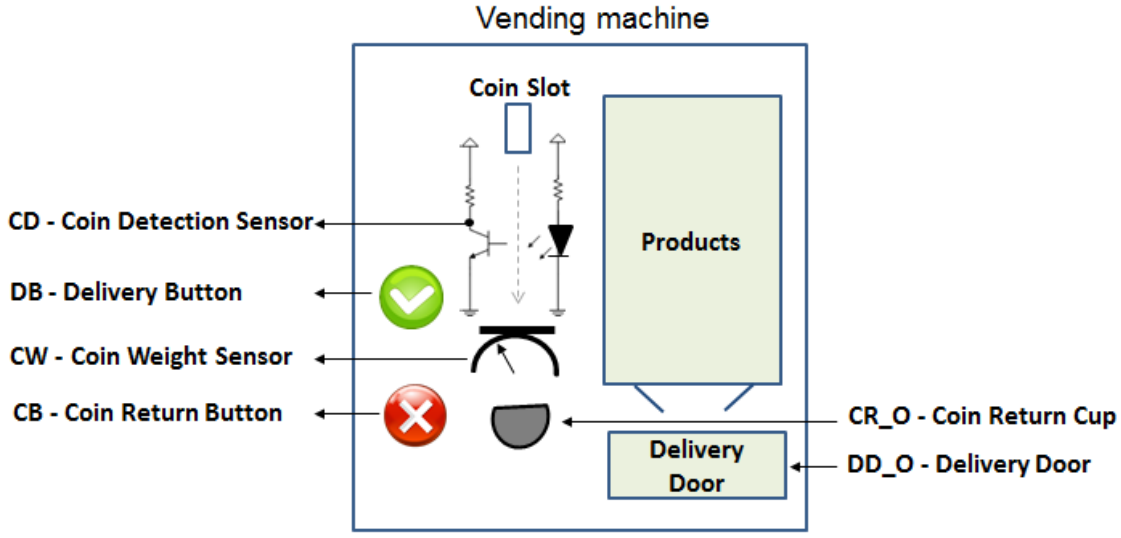

Figure 8.2. Block diagram of the vending machine.

STEP 1 – Textual description

The vending machine has the following features (textual description):

- The vending machine sells just one type of product, and each product costs $0.25;
- Only $0.25 coins are accepted;
- A valid $0.25 coin weights 5.670g;
- Invalid coins are rejected automatically, and the control circuit does not need to do anything about that;
- There is a slot to insert $0.25 coins in the vending machine;
- There is a coin detection circuit that outputs '1' every time a $0.25 coin is inserted in the slot, or '0' otherwise;
- There is a digital scale that outputs '1' when a weight of 5.670g (or more) is detected;
- There is a "delivery button" that outputs '1' every time it is pushed;
- There is a "coin return button" that outputs '1' every time it is pushed;
- There is a "coin return cup" to collect coins ejected by the "coin return button"(and also rejected coins);
- There is a door where the user can collect their product, after have pushed the "delivery button";
- When the vending machine runs out of products, it shuts itself, and the control circuit does not need to do anything about that.

The textual description together with the block diagram, comprises the Step 1 of the controller design methodology introduced in the previous section.

STEP 2 – FSM graphical representation

In Step 2, a graphical representation for the FSM is built, and shown in Figure 8.3. As shown in Figure 8.2, the following list of abbreviations has been adopted for the design:

CD – Coin Detection Sensor (input)
DB – Delivery Button (input)
CW – Coin weight Sensor (input)
CB – Coin Return Button (input)
CS – Current State (input)
NS – Next State (output)
CR_O – Coin Return Cup (output)
DD_O – Delivery Door (output)

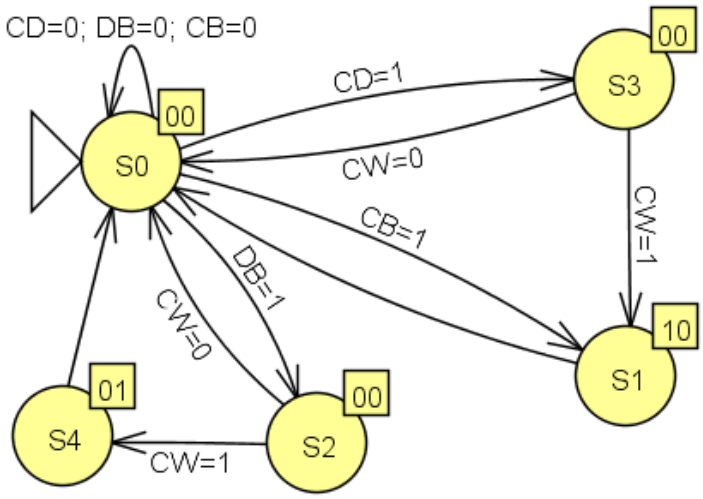

Figure 8.3. FSM graph for the vending machine controller.

In the graph diagram shown in Figure 8.3, it is possible to observe that:

- S0 is the initial state.
- The FSM remains in S0 waiting for a coin be inserted in the slot (CD = 1), or the delivery button is pushed (DB = 1), or the coin return button is pushed (CB = 1).
- The output signals are represented by the two bits inside the small box on the right top of each state. The most significant bit is the "coin return" (CR_O) output, and the less significant bit is the "delivery door" (DD_O) output.
- The "coin return" (CR_O) and the "delivery door" (DD_O) outputs are equal to 0 in all states but in S1 and S4. In S1, CR_O = 1, which means that a $0.25 coin is returned. In S4, DD_O = 1, in order to deliver a product to the customer.

- When a coin is inserted in the slot, the FSM goes to S3 state. If there was a coin already in the vending machine, the next state is S1, where the coin is returned to the customer (CR_O=1). Otherwise, the $0.25 coin is placed in the digital scale, and the FSM goes back to S0.
- When the delivery button is pushed, in the S2 state, the FSM goes to the S4 state to deliver a product only if the coin weight sensor indicates that there is a valid payment in the vending machine (CW = 1), otherwise, the FSM goes back to S0 state (CW=0).

STEP 3 – State transition table

In Step 3, a state transition table is built according to the FSM diagram. This sort of table is very similar to a truth table, and it has all the circuit's inputs and outputs. The main difference is the information regarding the "current state" and the "next state", even though they are also inputs and outputs for the circuit. Figure 8.4 shows the state transition table for the vending machine, where there are several "don't care" inputs. The table has been built considering that, for these inputs, the circuit has the same behavior when the respective input is high or low. For instance, on the first line of the table, it does not make any difference in the outputs if the CW input is '1' or '0'. This means that disregarding the CW input value, the outputs for this line will be NS = S0, CR_O = '0' and DD_O = '0'. The don't care inputs are very important in order to reduce the number of table lines.

Inputs					Outputs		
CD	DB	CW	CB	CS	NS	CR_O	SD_O
0	0	X	0	S0	S0	0	0
0	0	X	1	S0	S1	0	0
0	1	X	0	S0	S2	0	0
1	0	X	0	S0	S3	0	0
X	X	X	X	S1	S0	1	0
X	X	0	X	S2	S0	0	0
X	X	1	X	S2	S4	0	0
X	X	0	X	S3	S0	0	0
X	X	1	X	S3	S1	0	0
X	X	X	X	S4	S0	0	1

Figure 8.4. State transition table for the vending machine.

STEP 4 – VHDL description

In this step, the FSM behavior is described in VHDL, as shown in Figure 8.5. The two processes approach shown in Figure 7.3 has been used in this implementation. The first process is used to model the state register, defining the next state, and the second process provides the controller outputs, and generates the next state, according to the input values.

```
1  entity VendingMachine is
2      port(clock, reset  : in std_logic;
3           CD, DB, CW, CB: in std_logic;
4           CR_O, DD_O    : out std_logic);
5  end;
6
7  architecture behv of VendingMachine is
8      type STATES is (S0, S1, S2, S3, S4);
9      signal CS, NS : STATES;
10 begin
11     process (clock, reset)
12     begin
13         if reset= '0' then
14             CS <= S0;
15         elsif rising_edge(clock) then
16             CS <= NS;
17         end if;
18     end process;
19
20     process(CS, CD, DB, CW, CB)
21     begin
22         case CS is
23             when S0 =>  CR_O <= '0'; DD_O <= '0';    -- wait for an event
24                         if CB='1' then NS <= S1; elsif DB='1' then NS <= S2;
25                         elsif CD='1' then NS <= S3; else NS <= S0;
26             when S1 =>  CR_O <= '1'; DD_O <= '0';    -- return a $0.25 coin
27                         NS <= S0;
28             when S2 =>  CR_O <= '0'; DD_O <= '0';    -- Is there a payment?
29                         if CW='0' then NS <= S0; else NS <= S4;
30             when S3 =>  CR_O <= '0'; DD_O <= '0';    -- Is it a valid coin?
31                         if CW='0' then NS <= S0; else NS <= S1;
32             when S4 =>  CR_O <= '0'; DD_O <= '1';    -- Product delivery
33                         NS <= S0;
34         end case;
35     end process;
36 end behv;
```

Figure 8.5. The vending machine controller described in VHDL.

IMPORTANT!!! Steps 5 through 8 are not used in VHDL based designs! They are implemented by synthesis tools, which are carefully designed in order to generate the best circuit according to the VHDL description provided in Step 4.

Steps 5 through 8 are included next just as an example of how this type of controller circuit can be manually implemented using logic gates and a register, with no synthesis tool hardware inference and automatic circuit optimizations.

As stated before, steps 1 to 4 are used by VHDL developers in the design of FSM based controllers, while steps 5 to 8 are used by synthesis tool developers. A synthesis tool developer is responsible for implementing all the algorithms used to obtain the most appropriate coding strategy for the FSM sates, the best optimization for the combinational circuit (Boolean equations), among other performance enhancement features.

STEP 5 – FSM controller architecture

The block diagram of the proposed controller is shown in Figure 8.6. All the input signals, including the "next state" are located on the left side of the diagram. The two output signals, and the "current state" information are located on the right side of the diagram. The register used to store the current state value is located in the bottom of the diagram. The combinational circuit is responsible for defining the outputs (DD_O and CR_O) and the next state, which will be stored as the FSM's current state (CS). The combinational circuit design will be provided in Step 8.

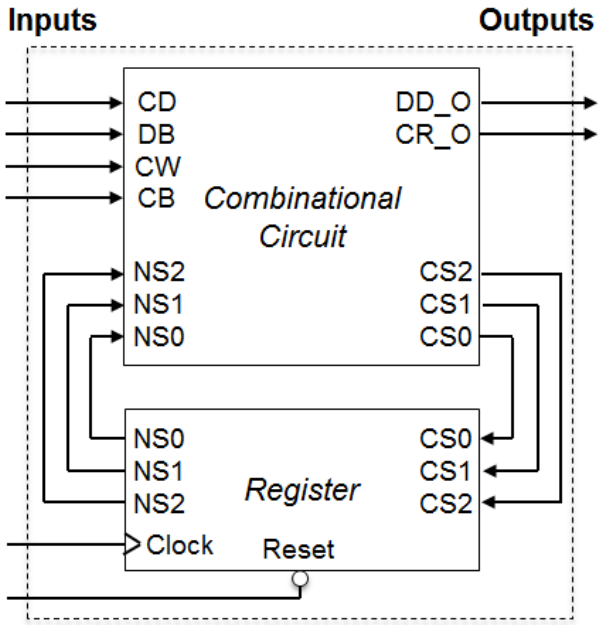

Figure 8.6. Block diagram of the architecture for the vending machine controller.

STEP 6 – States coding

During the synthesis process, all labels used to identify the FSM states are replaced by a binary code according to the available resources in the target hardware. For instance, FPGAs typically have a large amount of D flip-flops available and, in this case, the synthesis tools usually choose the "one-hot" coding strategy to identify the FSM states. In this strategy, for a group of bits that identify a state, only one bit is 1, and all the remain-

ing are 0. The one-hot strategy uses more storage resources, but on the other hand it does not need any special decoder circuits. An FSM's state is identified in the group of bits that represent a state, according to the position of the bit '1'. For instance, considering an FSM with four states, S0, S1, S2, and S3, in the one-hot strategy, these states are coded, respectively, as 0001, 0010, 0100, and 1000. Other options for coding the states of an FSM include the binary and the gray code. Figure 8.7 shows the first eight codes for each one of these strategies.

One-hot	Binary	Gray code
00000001	000	000
00000010	001	001
00000100	010	011
00001000	011	010
00010000	100	110
00100000	101	111
01000000	110	101
10000000	111	100

Figure 8.7. Coding strategies for FSM states.

STEP 7 – Edit the state transition table, and replace the symbolic identifiers by their binary coding representation

Inputs					Outputs		
CD	DB	CW	CB	$CS_{2..0}$	$NS_{2..0}$	CR_O	SD_O
0	0	X	0	0 0 0	0 0 0	0	0
0	0	X	1	0 0 0	0 0 1	0	0
0	1	X	0	0 0 0	0 1 0	0	0
1	0	X	0	0 0 0	0 1 1	0	0
X	X	X	X	0 0 1	0 0 0	1	0
X	X	0	X	0 1 0	0 0 0	0	0
X	X	1	X	0 1 0	1 0 0	0	0
X	X	0	X	0 1 1	0 0 0	0	0
X	X	1	X	0 1 1	0 0 1	0	0
X	X	X	X	1 0 0	0 0 0	0	1

Figure 8.8. State transition table, using binary codes instead of symbolic identifiers.

STEP 8 – Obtain the Boolean equations from the state transition table

Using the sum of products method, the following Boolean equations are obtained, which can be further optimized using methods as Boolean equations simplification, Karnahgk maps (in cases where there are up to five variables), or the Quine–McCluskey algorithm (prime implicants method):

```
DD_O = CS2 CS1' CS0'
CR_O = CS2' CS1' CS0
NS0 = CS2' CS1' CS0' CD' DB' CB +
      CS2' CS1' CS0' CD DB' CB' + CS2' CS1 CS0 CW
NS1 = CS2' CS1' CS0' CD' DB CB' +
      CS2' CS1' CS0' CD DB' CB'
NS2 = CS2' CS1 CS0' CW
```

Figure 8.9 shows the block diagram of the vending machine connected to proposed FSM based controller.

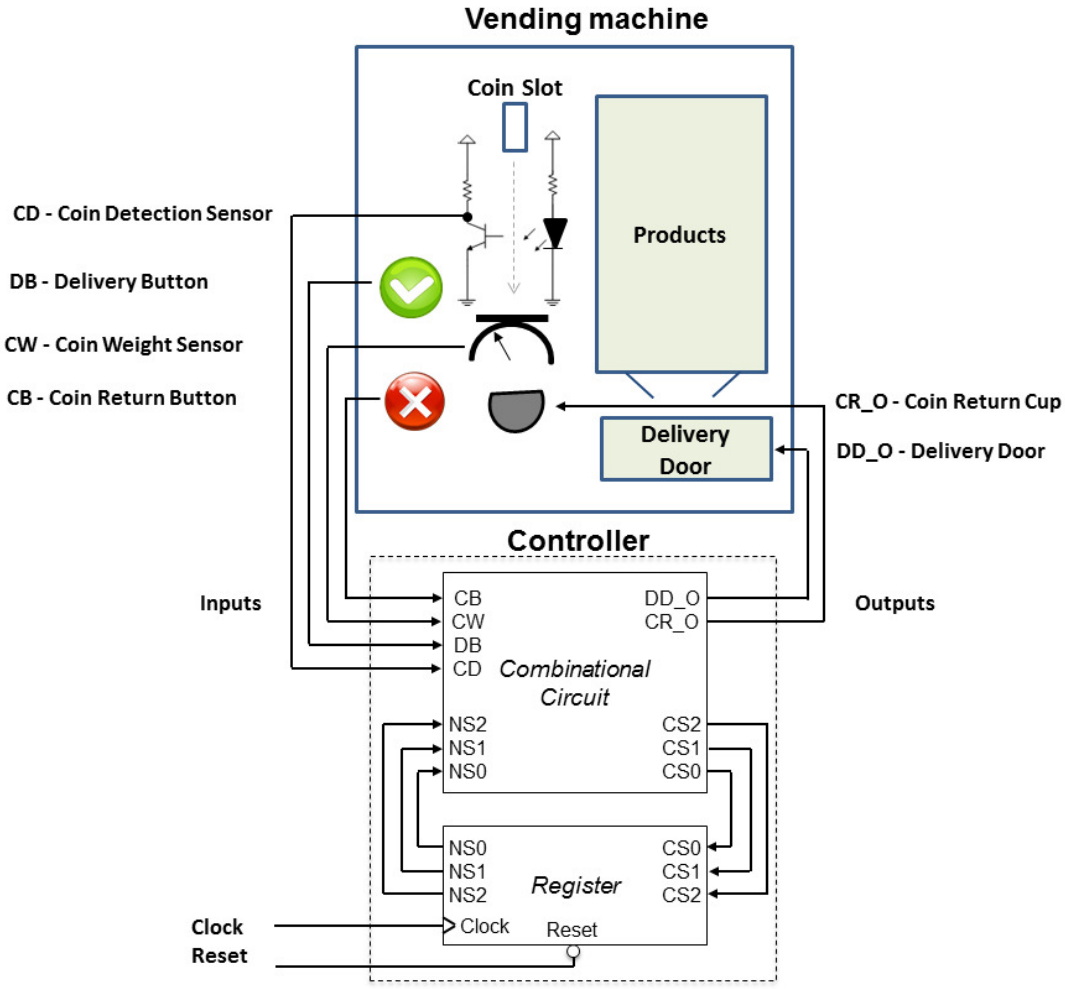

Figure 8.9. Block diagram of the designed FSM controller, showing the input and output connections.

In Figure 8.9, the Boolean equations should be placed in the "Combinational Circuit" box, and the register box is implemented using three D flip-flops. The controller inputs CB, CW, DB and CD are provided by the vending machine hardware. The controller outputs DD_O and CR_O are used to command the vending machine product delivery, and coin return.

8.3 Laboratory Assignment

The laboratory objectives are:

- to understand the use of FSMs as a control module for the operations flow of a combinational circuit;
- to design a VHDL based FSM to be used as a controller for the calculator implemented in previous chapters.

Problem definition: calculator with reduced data input signals

The calculator designed in previous chapter has the following inputs:

- $SW_{7..0}$ switches for operand A; and
- $SW_{15..8}$ switches for operand B.

To decrease the amount of switches in the board layout, in this laboratory session the VHDL designer will have to use just one set of switches, $SW_{7..0}$, to input both operands A and B. In this new version, switches $SW_{17..16}$ are still used for the selection of the chosen operation. Figure 8.10 shows how the data input is performed in the DE2 board, considering the two calculator versions.

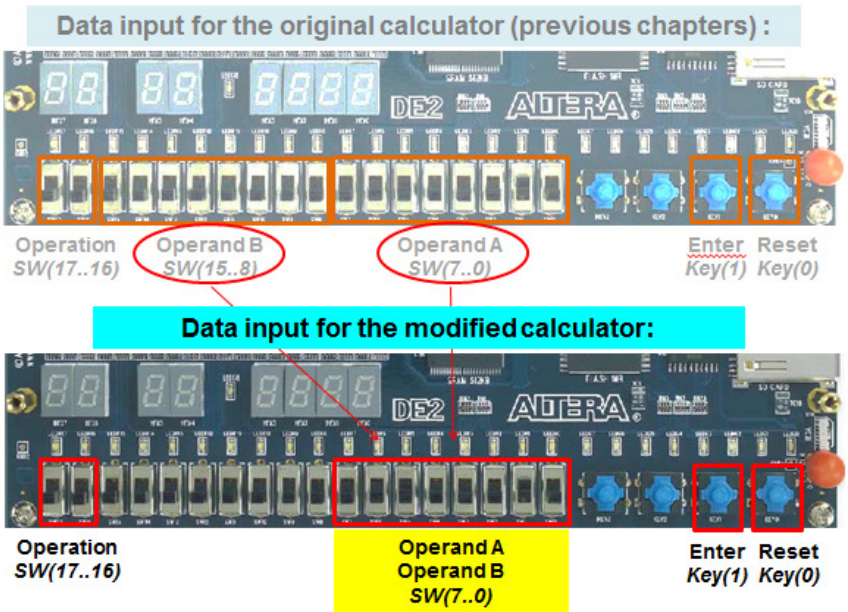

Figure 8.10. Data input for the calculator: previous and new version.

Laboratory Session

The tasks to be performed in this laboratory session are as follows:

- Add an input register to the calculator to hold the first operand;
- Design an FSM to control the sequence of operations in the new calculator;
- Add the FSM to the calculator design, simulate, and test the design in the FPGA board.

Adding a new input register to the calculator design

In Figure 8.11, a new register is added to the calculator design. The register is located in the upper left hand corner, and it is used to store the first operand for operations that employ two operands.

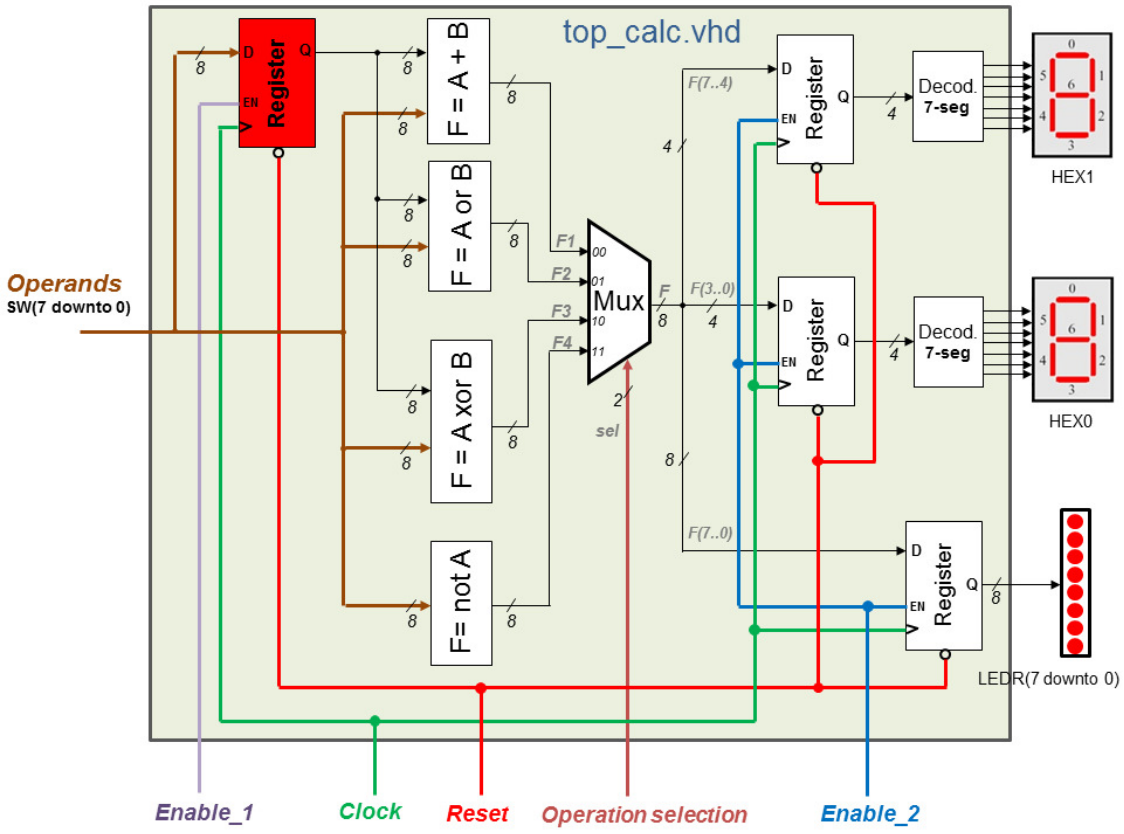

Figure 8.11. Inserting an input register to hold the first operand.

Some considerations:

- There is no need to make changes in the calculator components used in previous chapters;
- In the "top_calc.vhd" file, the new register is added just by including a new *port map* VHDL statement in the architecture. Notice that there are already two register components declared in the architecture, a four bits and an eight bits one;
- Some of the port map declarations should be changed in order to connect the new register to the remaining components, and also to the inputs;
- Another important remark is that all register enable signals, the multiplexor selector, among others, have been disconnected, as a controller FSM will be added to control the circuit operation sequence.

Next, the FSM controller is partially designed, according to the methodology described before. In Step 1, a textual description is provided for the calculator controller. In Step 2, a graphical representation for the FSM is built, according to the textual description discussed in Step 1. The proposed graphical representation is incomplete, as there are some states where all the possible outputs are not provided. Several graphical representations should be made, until one of the versions can be considered the "final" one. Step 3 is not used in this design, as the VHDL description can be extracted from the graphical view. In Step 4, a partial VHDL description for the FSM is provided. The students should use the information in steps 3 and 4, in order to complete calculator's controller design.

Designing an FSM based controller for the calculator

The design methodology described before is used in the calculator FSM design. In this case, the four steps are as follows:

STEP 1: Describing the calculator's sequence of operations and control flow

To operate the calculator the user should:

1. Select the chosen operation using switches $SW_{17..16}$;
2. Provide operand A using switches $SW_{7..0}$;
3. Press *Enter* – KEY_1 button is '0' when pressed.
4. For the *"not A"* operation, the result is shown on the 7-segments displays and LEDs;
5. For the remaining operations (*add*, *xor* and *or*), provide the second operand using switches $SW_{7..0}$, and press *Enter* again;
6. After the result is presented, this sequence is restarted from step 1.

Figure 8.12 shows the block diagram of the single input calculator, with the proposed FSM controller.

Figure 8.12As can be observed from Figure 8.12, the new top_calc.vhd file has 12 inferred components (using port map). These 12 components are inferred from nine components described in VHDL, and saved in the respective files as listed next:

- 2 x 8 bits registers – file *reg8bits.vhd*;
- 2 x 4 bits registers – file *reg4bits.vhd*;
- 1 x adder component – file *c1.vhd*;
- 1 x OR component – file *c2.vhd*;
- 1 x XOR component – file *c3.vhd*;
- 1 x NOT component – file *c4.vhd*;
- 1 x 4 input multiplexer – file *mux4x1.vhd*;
- 2 x 7-segment decoders – file *decod7seg.vhd*;
- 1 x FSM controller – file *FSMctrl.vhd*.

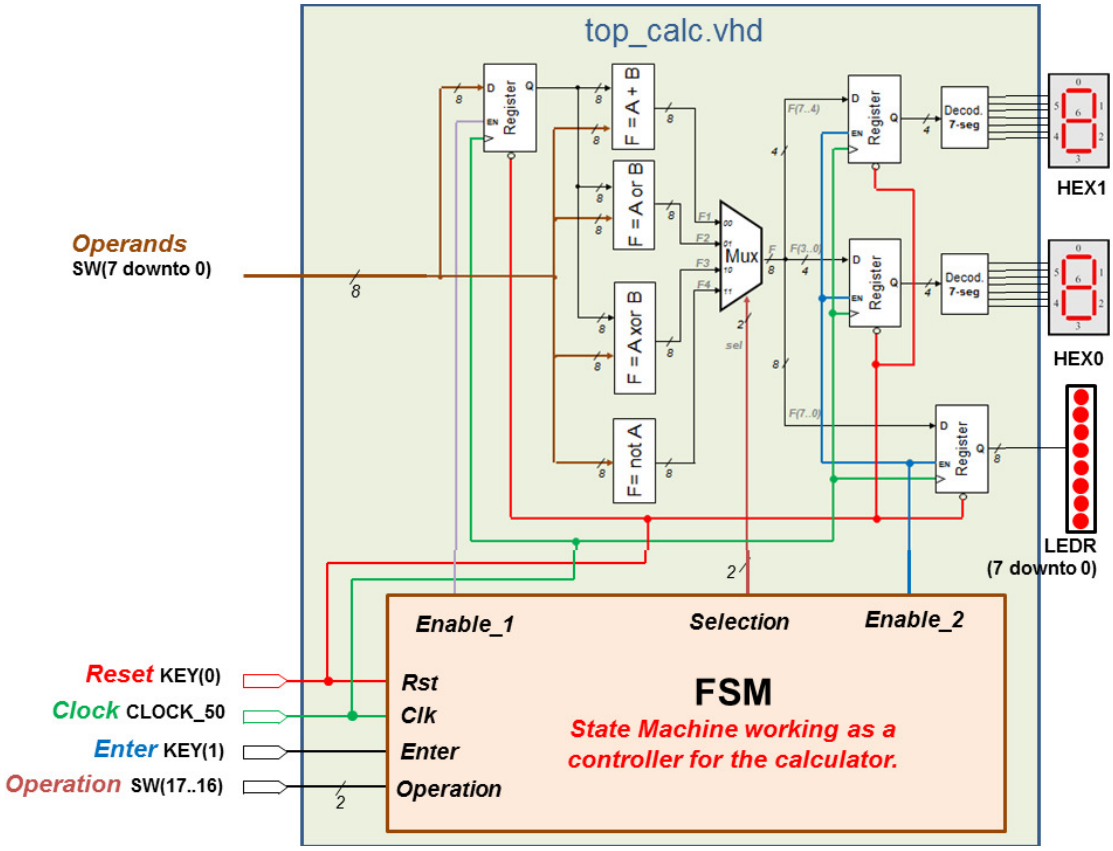

Figure 8.12. Adding an FSM to control the calculator sequence of operations.

STEP 2: FSM graphical representation

The graphical view in Figure 8.13 was built according to the description in step 1.

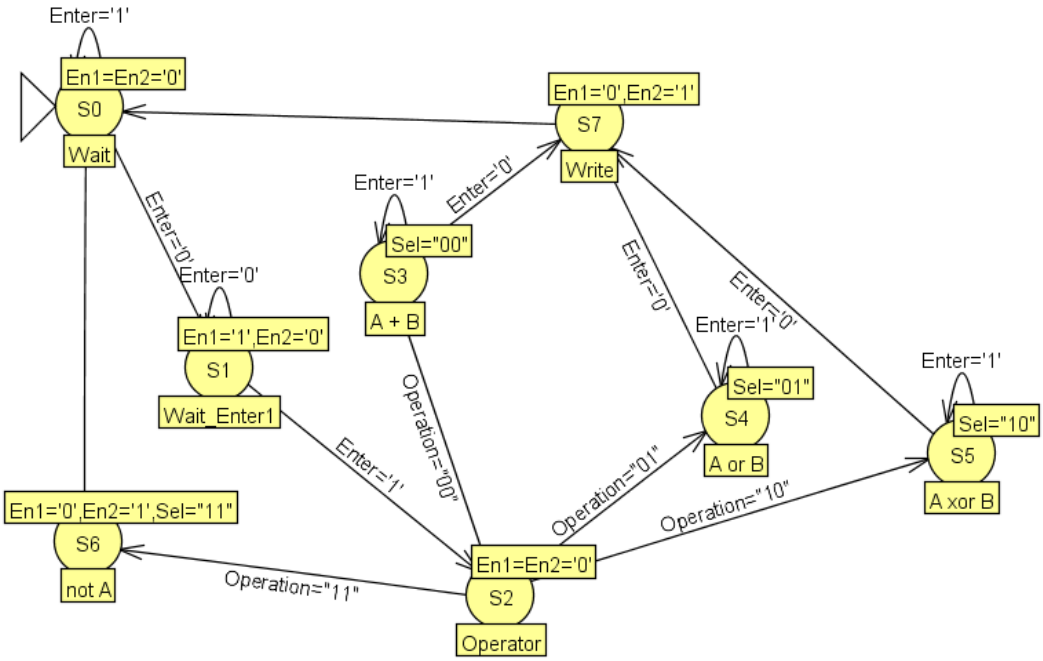

Figure 8.13. FSM controller graphical view – incomplete version.

The FSM has the following functionality:

- In the initial state "Wait", FSM outputs are disabled (*Enable_1*='0', *Enable_2*='0'), ensuring that there are no activities in the calculator, while the user does not provide the input data;
- When *Enter* is '0' (Key_1 is pressed), the FSM goes to the next state, and stays waiting until *Enter* goes back to '1' (push button is released);
- In the *Operator* state, according to the operation, there is a transition to the appropriate next state;
- For the "*not A*" operation (only one operand), the result is stored in the output registers (*Enable_2='1'*) and the FSM goes back to the initial state ("Wait");
- For the remaining operations, the second operator is read in an additional state - *Enable_2='0'* in states *S3*, *S4*, and *S5*, followed by *Enable_2='1'* in the *Write* state. In these operations, when Enter is pressed for the second time (Key_1 = *'0'*), the result is written in the output registers, and the FSM goes back to the initial state.

STEP 3: FSM state transition table

The state transition table can be obtained straight from the FSM graphical view, and it can be used to better visualize the inputs and outputs in each state.

Inputs				Outputs			
Reset	Enter	Operation	CS	NS	Enable_1	Selection	Enable_2
0	X	XX	S0	S0	0	00	0
1	1	XX	S0	S0	0	00	0
1	0	XX	S0	S1	0	00	0
1	0	XX	S1	S1	1	00	0
1	1	XX	S1	S2	1	00	0
1	X	00	S2	S3	0	00	0
1	X	01	S2	S4	0	00	0
1	X	10	S2	S5	0	00	0
1	X	11	S2	S6	0	00	0
1	0	XX	S3	S7	0	00	0
1	1	XX	S3	S3	0	00	0
1	0	XX	S4	S7	0	01	0
1	1	XX	S4	S4	0	01	0
1	0	XX	S5	S7	0	10	0
1	1	XX	S5	S5	0	10	0
1	X	XX	S6	S0	0	11	1
1	X	XX	S7	S0	0	00	1

Figure 8.14. State transition table for the proposed FSM.

STEP 4: VHDL description for the modeled FSM behavior

```
1  library  ieee;
2  use  ieee.std_logic_1164.all;
3  entity FSMctrl is
4    port ( Clk, Rst, Enter : in std_logic;
5           Operation: in std_logic_vector(1 downto 0);
6           Selection: out std_logic_vector(1 downto 0);
7           Enable_1, Enable_2: out std_logic);
8  end FSMctrl;
9  architecture FSM_beh of FSMctrl is
10   type states is (S0, S1, S2, S3, S4, S5, S6);
11   signal CS, NS: states;
12   signal clock, reset: std_logic;
13 begin
14   clock <= Clk;    reset <= Rst;
15 process (clock, reset)
16 begin
17   if reset = '0' then
18     EA <= S0;
19   elsif clock'event and clock = '1' then
20     EA <= PE;
21   end if;
22 end process;
23 process (EA, Enter)
24 begin
25   case EA is
26     when S0 =>
27       if Enter = '1' then PE <= S0; else PE <= S1;
28       end if;
29       Enable_1 <= '0'; Enable_2 <= '0';
30     when S1 =>      -- ... to be done
31     when S2 =>      -- Operator
32       Enable_1 <= '0'; Enable_2 <= '0';
33       if Operation = "00" then
34         PE <= S3;  -- add
35       elsif Operation = "01" then
36         PE <= S4;  -- OR
37       elsif         -- ... to be done
38    end case;
39 end process;
40 end FSM_beh;
```

Figure 8.15. Partial VHDL description for the proposed FSM.

Another option is to create a new top file, to connect together the FSM and the calculator components. Figure 8.16 shows this alternative implementation, where all calculator components are connected together in a top file called "top_calc.vhd", and the FSM implementation is in the "FSM.vhd" file. Figure 8.17 shows a simplified view of this alternative implementation.

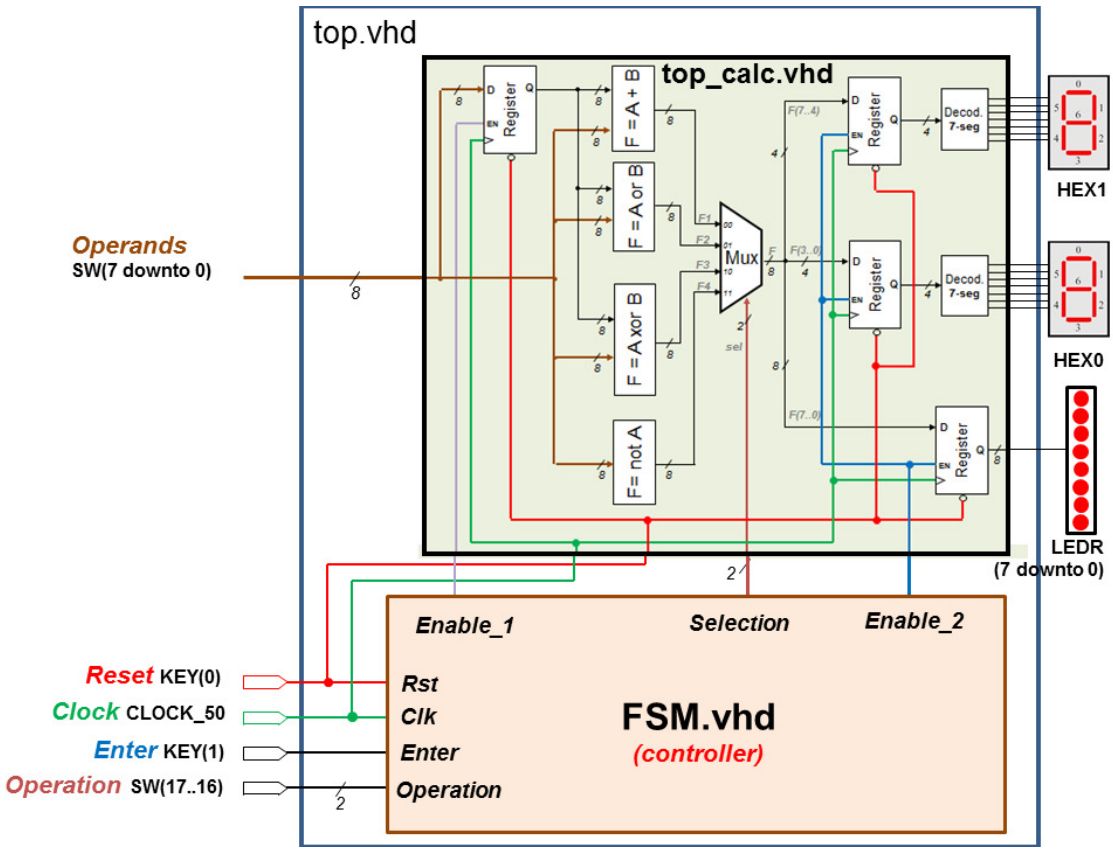

Figure 8.16. Block diagram for the alternative implementation.

It is important to notice that in this alternative implementation it may be interesting to rename all signals listed in the "top_calc.vhd" entity (component interface), in order to avoid using DE2 labels SW, LEDR, HEX0, and HEX1. These labels are used in the entity of the new "top.vhd" file. In Figure 8.17, the labels Operands, BIN, DISP0 and DISP1 were used instead.

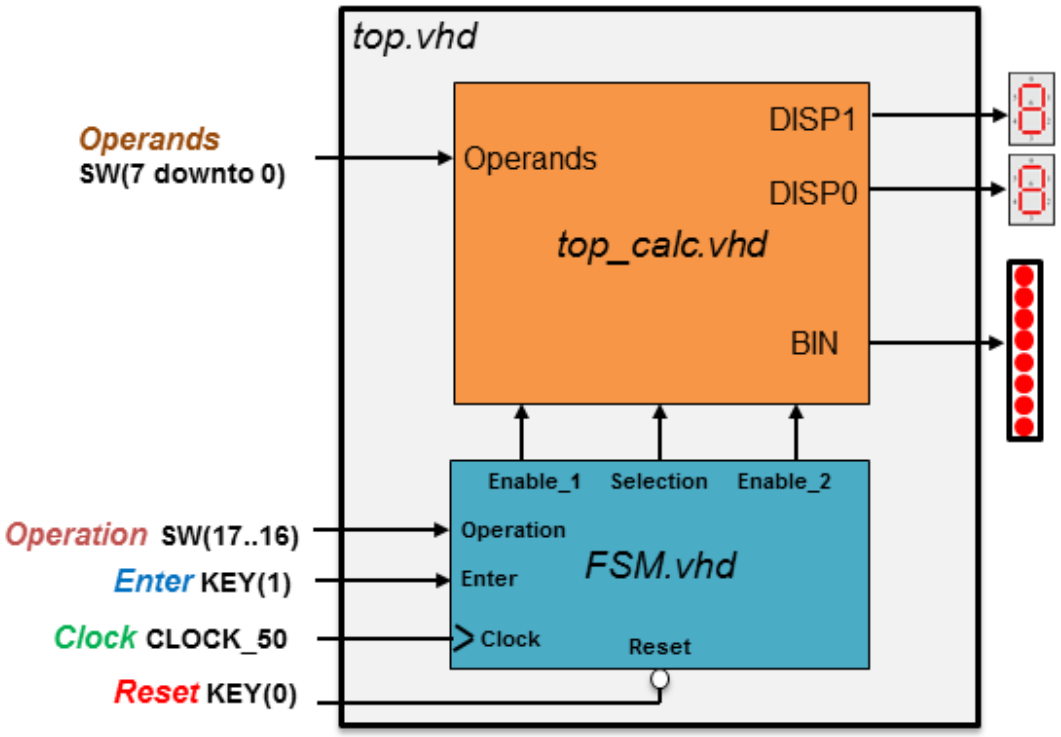

Figure 8.17. Simplified view of the block diagram for the alternative implementation.