9 VHDL Design of Timed (Category 2) State Machines

9.1 Introduction

This chapter presents several VHDL designs of category 2 state machines. It starts by presenting two VHDL templates, for Moore- and Mealy-based implementations, which are used subsequently to develop a series of designs related to the examples introduced in chapter 8.

The codes are always complete (not only partial sketches) and are accompanied by comments and simulation results, illustrating the design's main features. All circuits were synthesized using Quartus II (from Altera) or ISE (from Xilinx). The simulations were performed with Quartus II or ModelSim (from Mentor Graphics). The default encoding scheme for the states of the FSMs was regular sequential encoding (see encoding options in section 3.7; see ways of selecting the encoding scheme at the end of section 6.3).

The same designs are presented in chapter 10 using SystemVerilog, so the reader can make a direct comparison between the codes.

Note: See suggestions of VHDL books in the bibliography.

9.2 VHDL Template for Timed (Category 2) Moore Machines

The template is presented below. Because it is an extension to the Moore template for category 1, described in section 6.3, a review of that template is suggested before this one is examined because only the differences are described.

The only differences are those needed for the inclusion of a timer (external to the FSM—see figure 8.2a). Recall, however, that the FSM itself is responsible for controlling the timer. For that purpose, two strategies were developed in chapter 8, being the first generic (section 8.5.2), and the second (section 8.5.3), non-generic. It is very important that the reader review those two sections before proceeding.

The first of the two templates that follow is for timed Moore machines with the timer implemented using strategy #1. The timer-related constants (T_1, T_2, . . .) can be declared either as *generic* constants (lines 6–7; see details in the template for category 1 in section 6.3), therefore in the entity, or in the declarative part of the architecture, as shown in lines 18–20. The signal *t* must obviously stay where it is (line 21). As seen in section 8.5.2, the timer must obey $t_{max} \geq \max(T_1, T_2, \ldots) - 1$.

The first process (lines 26–37) implements the timer. Note that it is a straight copy of the code presented in section 8.5.2.

The second process (line 40) implements the machine's state register, exactly as for category 1 Moore (section 6.3).

The third process (lines 43–71) implements the machine's combinational logic. It is the same as for category 1 except for the fact that *t* might appear in the conditions for *nx_state* (lines 50, 52, . . .). The use of $t \geq T - 1$ instead of $t = T - 1$ is required in the conditional-timed transitions with $T - 1 < t_{max}$. Note that t_{max} does not need to be defined in all states, which is not true for strategy #2.

The fourth and final process (line 74) implements the optional output register, exactly as for category 1.

Note: See also the comments in section 6.4 on template variations.

```
1   --Timed Moore machine with timer control strategy #1---------
2   library ieee;
3   use ieee.std_logic_1164.all;
4   ------------------------------------------------------------------
5   entity circuit is
6      generic (
7         (timer-related constants of lines 18-20 can go here)
8      port (
9         (same as for category 1 Moore, section 6.3)
10  end entity;
11  ------------------------------------------------------------------
12  architecture moore_fsm of circuit is
13
14     --FSM-related declarations:
15     (same as for category 1 Moore, section 6.3)
16
17     --Timer-related declarations:
18     constant T1: natural := <value>;
19     constant T2: natural := <value>;   ...
20     constant tmax: natural := <value>; --tmax≥max(T1,T2,...)-1
21     signal t: natural range 0 to tmax;
22
23  begin
24
25     --Timer (strategy #1, section 8.5.2):
26     process (clk, rst)
27     begin
28        if (rst='1') then
29           t <= 0;
```



```
30        elsif rising_edge(clk) then
31           if pr_state /= nx_state then
32              t <= 0;
33           elsif t /= tmax then
34              t <= t + 1;
35           end if;
36        end if;
37  end process;
38
39  --FSM state register:
40  (same as for category 1 Moore, section 6.3)
41
42  --FSM combinational logic:
43  process (all) --list proc. inputs if "all" not supported
44  begin
45     case pr_state is
46        when A =>
47           output1 <= <value>;
48           output2 <= <value>;
49           ...
50           if ... and t>=T1-1 then
51              nx_state <= B;
52           elsif ... and t>=T2-1 then
53              nx_state <= ...;
54           else
55              nx_state <= A;
56           end if;
57        when B =>
58           output1 <= <value>;
59           output2 <= <value>;
60           ...
61           if ... and t>=T3-1 then
62              nx_state <= C;
63           elsif ... then
64              nx_state <= ...;
65           else
66              nx_state <= B;
67           end if;
68        when C =>
69           ...
70     end case;
71  end process;
72
73  --Optional output register:
74  (same as for category 1 Moore, section 6.3)
75
76  end architecture;
77  ------------------------------------------------------------------
```

The next template is for timed Moore machines employing strategy #2 to implement the timer.

The first difference is in line 18, which now includes also t_{max}.

The second difference is in the process for the timer (lines 23–34), which is a copy of the code presented in section 8.5.3.

The third and final difference is in the process for the combinational logic block (lines 40–70), which requires now the value of t_{max} to be specified in each state (lines 47, 59, . . .), even if the state is untimed (t_{max} = 0). This code can obviously be simplified in several ways when there are no conditional-timed transitions and/or t_{max} is the same in all or most states.

```
1    --Timed Moore machine with timer control strategy #2-----------
2    library ieee;
3    use ieee.std_logic_1164.all;
4    ---------------------------------------------------------------
5    entity circuit is
6          (same as template above)
7    end entity;
8    ---------------------------------------------------------------
9    architecture moore_fsm of circuit is
10
11      --FSM-related declarations:
12      (same as for category 1 Moore, section 6.3)
13
14      --Timer-related declarations:
15      constant T1: natural := <value>;
16      constant T2: natural := <value>; ...
17      constant tmax_timer: natural := <value>; -- ≥max(T1,T2,...)-1
18      signal t, tmax: natural range 0 to tmax_timer;
19
20   begin
21
22      --Timer (strategy #2, section 8.5.3):
23      process (clk, rst)
24      begin
25         if (rst='1') then
26               t <= 0;
27         elsif rising_edge(clk) then
28               if t < tmax then
29                  t <= t + 1;
30               else
31                  t <= 0;
32               end if;
33         end if;
34      end process;
35
36      --FSM state register:
37      (same as for category 1 Moore, section 6.3)
38
39      --FSM combinational logic:
40      process (all) --list proc. inputs if "all" not supported
41      begin
42         case pr_state is
43            when A =>
44               output1 <= <value>;
45               output2 <= <value>;
46               ...
47               tmax <= T1-1;
48               if ... and t=tmax then
49                  nx_state <= B;
```



```
50                elsif ... then
51                   nx_state <= ...;
52                else
53                   nx_state <= A;
54                end if;
55             when B =>
56                output1 <= <value>;
57                output2 <= <value>;
58                ...
59                tmax <= T2-1;
60                if ... and t=tmax then
61                   nx_state <= C;
62                elsif ... then
63                   nx_state <= ...;
64                else
65                   nx_state <= B;
66                end if;
67             when C =>
68                ...
69          end case;
70       end process;
71
72       --Optional output register:
73       (same as for category 1 Moore, section 6.3)
74
75    end architecture;
76    ---------------------------------------------------------------
```

9.3 VHDL Template for Timed (Category 2) Mealy Machines

The template is presented below, using strategy #1 to implement the timer. The only difference with respect to the Moore template (section 9.2) is in the process for the combinational logic block (lines 20–60) because the outputs are specified differently here (see the template for category 1 Mealy machines in section 6.5). Recall that in a Mealy machine the output depends not only on the FSM's state, but also on the input, so **if** statements are expected for the output in one or more states because the output values might not be unique.

Please review the following comments, which can be easily adapted from the Moore case to the Mealy case:

—On the Moore template for category 1, in section 6.3, especially comment 10.
—On the *enum_encoding* and *fsm_encoding* attributes, also in section 6.3.
—On possible code variations, in section 6.4.
—On the Mealy template for category 1, in section 6.5.
—On the Moore templates for category 2, in section 9.2.

```
1    --Timed Mealy machine with timer control strategy #1----
2    library ieee;
3    use ieee.std_logic_1164.all;
4    ---------------------------------------------------------
```

```
5    entity circuit is
6       (same as for Moore, Section 9.2));
7    end entity;
8    ------------------------------------------------------------
9    architecture mealy_fsm of circuit is
10      (same as for Moore, section 9.2)
11   begin
12
13      --Timer (using timer control strategy #1):
14      (same as for Moore, section 9.2)
15
16      --FSM state register:
17      (same as for Moore, section 9.2)
18
19      --FSM combinational logic:
20      process (all)
21      begin
22         case pr_state is
23            when A =>
24               if ... and t>=T1-1 then
25                  output1 <= <value>;
26                  output2 <= <value>;
27                  ...
28                  nx_state <= B;
29               elsif ... and t>=T2-1 then
30                  output1 <= <value>;
31                  output2 <= <value>;
32                  ...
33                  nx_state <= ...;
34               else
35                  output1 <= <value>;
36                  output2 <= <value>;
37                  ...
38                  nx_state <= A;
39               end if;
40            when B =>
41               if ... and t>=T3-1 then
42                  output1 <= <value>;
43                  output2 <= <value>;
44                  ...
45                  nx_state <= C;
46               elsif ... then
47                  output1 <= <value>;
48                  output2 <= <value>;
49                  ...
50                  nx_state <= ...;
51               else
52                  output1 <= <value>;
53                  output2 <= <value>;
54                  ...
55                  nx_state <= B;
56               end if;
57            when C =>
58               ...
59         end case;
60      end process;
```



```
61
62       --Optional output register:
63       (same as for Moore, section 9.2)
64
65   end architecture;
66   ------------------------------------------------------------
```

9.4 Design of a Light Rotator

This section presents a VHDL-based design for the light rotator introduced in section 8.11.2. The Moore template of section 9.2 is used to implement the FSM of figure 8.14b. Either strategy #1 (section 8.5.2) or #2 (section 8.5.3) can be used to build the timer (both templates were presented in section 9.2); the former is employed in the code below, while the latter is explored in exercise 9.1.

The entity, called *light_rotator*, is in lines 5–9. All ports are of type *std_logic* or *std_logic_vector* (industry standard).

The architecture, called *moore_fsm*, is in lines 11–105. As usual, it contains a declarative part and a statements part, with three processes in the latter.

The declarative part of the architecture (lines 13–21) contains FSM- and timer-related declarations. In the former, the enumerated type *state* is created to represent the machine's present and next states. In the latter, the values chosen for T_1 and T_2 are such that 120 ms and 35 ms result, respectively, assuming f_{clk} = 50 MHz.

The first process (lines 26–37) implements the timer (using strategy #1). Except for the inclusion of *stp* (lines 26 and 30), this code is exactly as in the template.

The second process (lines 40–47) implements the FSM's state register, exactly as in the template.

The third and final process (lines 50–103) implements the entire combinational logic section. It is just a list of all states (indeed, because this code is repetitive, some of the states were not detailed in order to save some space), each containing the output (*ssd*) value and the next state. Note that in each state the output value is unique because in a Moore machine the output depends only on the state in which the machine is.

In this kind of application the "–1" term present in the determination of the total time (lines 20, 55, 62, . . .) does not make any difference, but it was maintained as a reminder of the accurate value.

Observe the correct use of registers and the completeness of the code, as described in comment 10 of section 6.3. Note in particular the following:

1) Regarding the use of registers: The circuit is not overregistered. This can be observed in the **elsif rising_edge(clk)** statement of line 44 (responsible for the inference of flip-flops), which is closed in line 46, guaranteeing that only the machine state (line