12 VHDL Design of Recursive (Category 3) State Machines

12.1 Introduction

This chapter presents several VHDL designs of category 3 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 11.

The codes are always complete (not only partial sketches) and are accompanied by comments and often also 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 13 using SystemVerilog, so the reader can make a direct comparison between the codes.

Note: See suggestions of VHDL books in the bibliography.

12.2 VHDL Template for Recursive (Category 3) Moore Machines

The template is presented below. Because it is an extension to the Moore templates for categories 1 and 2, described in sections 6.3 and 9.2, respectively, a review of those templates is suggested before this one is examined because only the differences are described. Review also some possible code variations in section 6.4.

The only differences are those needed for the inclusion of an auxiliary register, compulsory in category 3 machines. As seen in section 6.2, the architecture is composed of two parts, the declarative part (before **begin**) and the statements part (from **begin** on); both have new elements in order to accommodate the auxiliary register.

In the architecture's declarative part (lines 14–21), the difference is in line 21, in which two signals are created to deal with the auxiliary register. It is assumed that there is only one output and that it must be stored, but recall that the circuit might have several outputs, not all registered. The actual number of auxiliary registers is determined by the number of outputs that depend on past values.

In the architecture's statements part (lines 23–75), two differences are seen: the inclusion of a process to infer the auxiliary register and the replacement of *outp* with *outp_reg* on the right-hand side of the recursive equations. The latter removes the recursiveness, thus allowing the output to be computed by a combinational circuit.

Lines 29–36 show the process that implements the auxiliary register. If one prefers, this process can be combined with that for the FSM's state register (a shorter code results, but less didactic, with no effect on the result).

Lines 42–68 show the process that implements the machine's combinational logic section. The only difference here is that *outp_reg*, instead of *outp* itself, appears on the right-hand side of the (originally) recursive equations (lines 46 and 56).

As explained in section 11.3, an interesting aspect of category 3 FSMs is that the auxiliary register can also play the role of output register (for glitch-free and/or pipelined construction). To do so, we simply send *outp_reg* out instead of *outp* in line 73.

The code is concluded in line 73, in which the value of *outp* is passed to the actual output. In fact, the actual output could be used in lines 34, 46, and 56, in which case the mode of *output* (in the entity) should be changed from *out* to *buffer* (see example in section 12.5).

```
1   ------------------------------------------------------------
2   library ieee;
3   use ieee.std_logic_1164.all;
4   ------------------------------------------------------------
5   entity circuit is
6      generic (
7         (same as for category 2 Moore, section 9.2)
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     (same as for category 2 Moore, section 9.2)
19
20     --Auxiliary-register-related declarations:
```



```
21     signal outp, outp_reg: std_logic_vector(...);
22
23  begin
24
25     --Timer:
26     (same as for category 2 Moore, section 9.2)
27
28     --Auxiliary register:
29     process (clk, rst)
30     begin
31        if (rst='1') then
32           outp_reg <= <initial value>;
33        elsif rising_edge(clk) then
34           outp_reg <= outp;
35        end if;
36     end process;
37
38     --FSM state register:
39     (same as for category 2 Moore, section 9.2)
40
41     --FSM combinational logic:
42     process (all) --list proc. inputs if "all" not supported
43     begin
44        case pr_state is
45           when A =>
46              outp <= outp_reg;
47              tmax <= T1-1;
48              if <condition> then
49                 nx_state <= B;
50              elsif <condition> then
51                 nx_state <= ...;
52              else
53                 nx_state <= A;
54              end if;
55           when B =>
56              outp <= outp_reg + 1;
57              tmax <= T2-1;
58              if <condition> then
59                 nx_state <= C;
60              elsif <condition> then
61                 nx_state <= ...;
62              else
63                 nx_state <= B;
64              end if;
65           when C =>
66              ...
67        end case;
68     end process;
69
70     --Optional output register:
71     (same as for category 1 Moore, section 6.3)
72
73     output <= outp;
74
75  end architecture;
76  ------------------------------------------------------------
```

12.3 VHDL Template for Recursive (Category 3) Mealy Machines

The template is presented below. The only difference with respect to the Moore template just described is in the process for the combinational logic (lines 23–57) because the output is specified differently here. 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 value 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 template for category 2, in section 9.2.
—On the Mealy template for category 2, in section 9.3.
—Finally, on the Moore template for category 3, in section 12.2.

```
1   ------------------------------------------------------------
2   library ieee;
3   use ieee.std_logic_1164.all;
4   ------------------------------------------------------------
5   entity circuit is
6      (same as for Moore, section 12.2)
7   end entity;
8   ------------------------------------------------------------
9   architecture mealy_fsm of circuit is
10     (same as for Moore, section 12.2)
11  Begin
12
13     --Timer:
14     (same as for Moore, section 9.2)
15
16     --Auxiliary register:
17     (same as for Moore, section 12.2)
18
19     --FSM state register:
20     (same as for Moore, section 9.2)
21
22     --FSM combinational logic:
23     process (all) --list proc. inputs if "all" not supported
24     begin
25        case pr_state is
26           when A =>
27              if <condition> then
28                 outp <= outp_reg;
29                 tmax <= <value>;
30                 nx_state <= B;
31              elsif <condition> then
```



```
32                 outp <= outp_reg + 1;
33                 tmax <= <value>;
34                 nx_state <= ...;
35              else
36                 outp <= outp_reg;
37                 tmax <= <value>;
38                 nx_state <= A;
39              end if;
40           when B =>
41              if <condition> then
42                 outp <= outp_reg;
43                 tmax <= <value>;
44                 nx_state <= C;
45              elsif <condition> then
46                 outp <= outp_reg - 1;
47                 tmax <= <value>;
48                 nx_state <= ...;
49              else
50                 outp <= outp_reg;
51                 tmax <= <value>;
52                 nx_state <= B;
53              end if;
54           when C =>
55              ...
56        end case;
57     end process;
58
59     --Optional output register:
60     (same as for Moore, Section 12.2)
61
62     output <= outp;
63
64  end architecture;
65  ------------------------------------------------------------
```

12.4 Design of a Datapath Controller for a Multiplier

This section presents a VHDL-based design for the control unit introduced in section 11.7.5, which controls a datapath to produce a sequential add-and-shift multiplier. The Moore template for category 3 machines seen in section 12.2 is used to implement the FSM of figure 11.12b.

The entity, called *control_unit_for_multiplier*, is in lines 5–11. The number of bits (N) in the multiplier and multiplicand was entered as a generic parameter (line 6); a small value ($N = 4$) was used to ease the inspection of the simulation results. Note that all ports (lines 8–10) are of type *std_logic* or *std_logic_vector* (industry standard).

The architecture, called *moore_fsm*, is in lines 13–93. As usual, it contains a declarative part and a statements part, with three processes in the latter.

The declarative part of the architecture (lines 15–20) contains FSM- and auxiliary-register-related declarations. In the former the enumerated type *state* is created to represent the machine's present and next states. In the latter the signals *i* and *i_reg*