Sequential Circuit Design using VHDL

Current FPGA architectures support only synchronous sequential circuits. The next pages give example VHDL descriptions of typical synchronous sequential circuits. These circuits are described using sequential assignment statements.

Registers

Figure 1 shows the VHDL code for a positive-edge-triggered D flip-flop.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity d_flip_flop is
5      Port(D   : in  STD_LOGIC;
6           CLK : in  STD_LOGIC;
7           Q   : out  STD_LOGIC);
8   end d_flip_flop;
9
10  architecture Behavioral of d_flip_flop is
11  begin
12     process(CLK)
13     begin
14        if CLK'EVENT and CLK='1' then
15           Q <= D;
16        end if;
17     end process;
18  end Behavioral;
```

Figure 1

The sensitivity list of the `process` in line `12` (**Figure 1**) includes only the `CLK` (clock) signal. This means that the `process` becomes active only when the value of `CLK` signal changes. The `'EVENT` (line `14`) is an attribute. The `CLK'EVENT` syntax represents a change in the value of the `CLK` signal. The `CLK'EVENT and CLK='1'` condition is true when a change in the value of the `CLK` signal occurs and the new value of the `CLK` signal is equal to `'1'`. This simply means “at the rising edge of the `CLK` signal”. So at the rising edge of the `CLK` signal, `Q` is assigned the value of `D`. When the condition is not true, `Q` is equal to the value that `D` had the last time the condition was true.

By replacing the condition in line `14` (**Figure 1**) with the `CLK'EVENT and CLK='0'` condition, a negative-edge-triggered D flip-flop is described. The `STD_LOGIC_1164` package defines two functions named `rising_edge()` and `falling_edge()`. They can be used as a short-form notation for the condition that checks for the occurrence of a clock edge [1]. For example the condition in line `14` (**Figure 1**) could be replaced by the `rising_edge(CLK)` expression [1], [2].

Figure 2 (a) shows the VHDL code for an 8-bit register with synchronous reset and **Figure 2 (b)** shows the VHDL code for an 8-bit register with clock enable and asynchronous reset. A register is defined as a group of D flip-flops that have the same properties.

```vhdl
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity d_reg_rset is
5     Port(CLK  : in   STD_LOGIC;
6          RSET : in   STD_LOGIC;
7          D    : in   STD_LOGIC_VECTOR(7 downto 0);
8          Q    : out  STD_LOGIC_VECTOR(7 downto 0));
9  end d_reg_rset;
10
11 architecture Behavioral of d_reg_rset is
12 begin
13    process(CLK)
14    begin
15       if CLK'EVENT and CLK='1' then
16          if RSET='1' then
17             Q <= (OTHERS => '0');
18          else
19             Q <= D;
20          end if;
21       end if;
22    end process;
23 end Behavioral;
```

(a)

```vhdl
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity d_reg_rset_ce is
5     Port(CLK  : in   STD_LOGIC;
6          RSET : in   STD_LOGIC;
7          CE   : in   STD_LOGIC;
8          D    : in   STD_LOGIC_VECTOR(7 downto 0);
9          Q    : out  STD_LOGIC_VECTOR(7 downto 0));
10 end d_reg_rset_ce;
11
12 architecture Behavioral of d_reg_rset_ce is
13 begin
14    process(CLK,RSET)
15    begin
16       if RSET='1' then
17          Q <= (OTHERS => '0');
18       elsif CLK'EVENT and CLK='1' then
19          if CE='1' then
20             Q <= D;
21          end if;
22       end if;
23    end process;
24 end Behavioral;
```

(b)

Figure 2

In **Figure 2 (a)** the reset signal, `RSET`, is not included in the sensitivity list (line `13`). At the rising edge of the clock signal, `CLK`, if the condition `RSET='1'` is true all the D flip-flops that consist the register store the `'0'` logic value (line `17`). This syntax describes the synchronous reset property.

In **Figure 2 (b)** the reset signal, `RSET`, is included in the sensitivity list (line `14`). This means that the `process` becomes active at the change of the value of any signal that is included in the sensitivity list. A change in the value of the `RSET` signal activates the `process`. If the condition `RSET='1'` is true, all the D flip-flops that consist the register store the `'0'` logic value (line `17`). If the `CLK` and `RSET` signal values change simultaneously the process will be activated. If the `RSET='1'` condition is true, all the D flip-flops that consist the register store the `'0'` logic value because the `RSET='1'` condition has greater priority than the `CLK'EVENT and CLK='1'` condition. This syntax describes the asynchronous reset property. If the `RSET='1'` condition is not true at the rising edge of the clock signal `Q` is assigned the value of `D` only if the value of the clock enable signal, `CE`, is equal to `'1'`.

Shift Registers

Figure 3 shows the VHDL code for a N-bit Serial-In Parallel-Out (SIPO) shift register with clock enable (`CE`) and synchronous reset (`RSET`). The default value of the `N` parameter is set to `4` (line `5`). In line `14` the `tmp` signal is declared. This signal is necessary because the `Q` signal is an output and can't be used at the right side of the `<=` operator (line 22). Line `22` describes the main functionality of the shift register. Shifting is performed from right-to-left direction. After the shift, the least significant (right-most) bit (`tmp(0)`) is equal to the value that the serial input (`SIN`) had before the shift. In line `26` a simple signal assignment statement is used to describe the parallel-out property of the circuit. **Figure 4** shows an alternative VHDL description for the shift register that is described in **Figure 3**.

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity sh_reg_n is
 5     generic(N : integer := 4);
 6     Port(CLK  : in   STD_LOGIC;
 7          RSET : in   STD_LOGIC;
 8          EN   : in   STD_LOGIC;
 9          SIN  : in   STD_LOGIC;
10          Q    : out  STD_LOGIC_VECTOR(N-1 downto 0));
11  end sh_reg_n;
12
13  architecture Behavioral of sh_reg_n is
14     signal tmp : STD_LOGIC_VECTOR(N-1 downto 0);
15  begin
16     process(CLK)
17     begin
18        if CLK'EVENT and CLK='1' then
19           if RSET='1' then
20              tmp <= (OTHERS => '0');
21           elsif EN='1' then
22              tmp <= tmp(N-2 downto 0) & SIN;
23           end if;
24        end if;
25     end process;
26     Q <= tmp;
27  end Behavioral;
```

Figure 3

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity sh_reg_n is
 5     generic(N : integer := 4);
 6     Port(CLK  : in   STD_LOGIC;
 7          RSET : in   STD_LOGIC;
 8          EN   : in   STD_LOGIC;
 9          SIN  : in   STD_LOGIC;
10          Q    : out  STD_LOGIC_VECTOR(N-1 downto 0));
11  end sh_reg_n;
12
13  architecture Behavioral of sh_reg_n is
14     signal tmp : STD_LOGIC_VECTOR(N-1 downto 0);
15  begin
16     process(CLK)
17     begin
18        if CLK'EVENT and CLK='1' then
19           if RSET='1' then
20              tmp <= (OTHERS => '0');
21           elsif EN='1' then
22              tmp(0) <= SIN;
23              for i IN 1 to N-1 loop
24                 tmp(i) <= tmp(i-1);
25              end loop;
26           end if;
27        end if;
28     end process;
29     Q <= tmp;
30  end Behavioral;
```

Figure 4

The only difference between **Figure 3** and **Figure 4** is that line `22` in **Figure 3** is replaced by lines `22-25` in **Figure 4**. The `for loop` statement, that uses similar syntax to the `for generate` statement [1], can only be used in a `process` statement. The `for loop` statement is used to describe circuits that exhibit regularity in their function. The `for loop` statement that is used in lines `23-25` (**Figure 4**) is equal to the `tmp(N-1 downto 1) <= tmp(N-2 downto 0);` statement (or the subsequent statements `tmp(1) <= tmp(0);`, `tmp(2) <= tmp(1);`, `tmp(3) <= tmp(2);`, …, `tmp(N-1) <= tmp(N-2);`).

It is important to remember that when the value of a signal in the sensitivity list changes, the process becomes active. Once active, the statements inside the process are "evaluated" in sequential order. **Any signal assignments made in the process take effect only after all the statements inside the process have been evaluated**. **The signal assignment statements inside the process are scheduled and will take effect at the end of the process** [1]. **Figure 5** shows the architecture of a 4-bit shift register.

```
architecture Behavioral of shift_4 is
   signal tmp : STD_LOGIC_VECTOR(3 downto 0);
begin
   process(CLK)
   begin
      if CLK'EVENT and CLK='1' then
         if RSET='1' then
            tmp <= (OTHERS => '0');
         elsif EN='1' then
            tmp(2) <= tmp(1);
            tmp(0) <= SIN;
            tmp(3) <= tmp(2);
            tmp(1) <= tmp(0);
         end if;
      end if;
   end process;
   Q <= tmp;
end Behavioral;
```

Figure 5

The order of the assignment statements that describe the shifting does not matter because any signal assignments made in the process do not take effect until the end of the process. Hence all flip-flops change their values at the same time, as required in the shift register.

Counters

Figure 6 shows a N-bit up counter with clock enable (`CE`) and synchronous reset (`RSET`). The default value of the `N` parameter is set to `4` (line `6`).

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.NUMERIC_STD.ALL;
 4
 5  entity count_n is
 6     generic(N : integer := 4);
 7     Port(CLK  : in   STD_LOGIC;
 8          RSET : in   STD_LOGIC;
 9          EN   : in   STD_LOGIC;
10          Q    : out  STD_LOGIC_VECTOR(N-1 downto 0));
11  end count_n;
12
13  architecture Behavioral of count_n is
14     signal count : unsigned(N-1 downto 0);
15  begin
16     process(CLK)
17     begin
18        if CLK'EVENT and CLK='1' then
19           if RSET='1' then
20              count <= (OTHERS => '0');
21           elsif EN='1' then
22              count <= count + 1;
23           end if;
24        end if;
25     end process;
26     Q <= std_logic_vector(count);
27  end Behavioral;
```

Figure 6

Line 3 (**Figure 6**) states that the NUMERIC_STD package is used. This statement is necessary in order to use the + operator. In line 14 the count signal is declared. This signal is necessary because the Q signal is an output and can't be used at the right side of the <= operator (line 22). The count signal is declared as of type unsigned. This is done to avoid the type conversions in lines 22 and 26. Specifically the declaration of the function "+" of the NUMERIC_STD package that is used in line 22 is shown in **Figure 7** (-- Id: A.5).

```
-- Id: A.3
function "+" (L, R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0).
-- Result: Adds two UNSIGNED vectors that may be of different lengths.

-- Id: A.4
function "+" (L, R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(MAX(L'LENGTH, R'LENGTH)-1 downto 0).
-- Result: Adds two SIGNED vectors that may be of different lengths.

-- Id: A.5
function "+" (L: UNSIGNED; R: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(L'LENGTH-1 downto 0).
-- Result: Adds an UNSIGNED vector, L, with a non-negative INTEGER, R.

-- Id: A.6
function "+" (L: NATURAL; R: UNSIGNED) return UNSIGNED;
-- Result subtype: UNSIGNED(R'LENGTH-1 downto 0).
-- Result: Adds a non-negative INTEGER, L, with an UNSIGNED vector, R.

-- Id: A.7
function "+" (L: INTEGER; R: SIGNED) return SIGNED;
-- Result subtype: SIGNED(R'LENGTH-1 downto 0).
-- Result: Adds an INTEGER, L(may be positive or negative), to a SIGNED
--         vector, R.

-- Id: A.8
function "+" (L: SIGNED; R: INTEGER) return SIGNED;
-- Result subtype: SIGNED(L'LENGTH-1 downto 0).
-- Result: Adds a SIGNED vector, L, to an INTEGER, R.
```

Figure 7

The left operand in line 22 (**Figure 6**) is of type unsigned and the right operand is of type natural (non-negative integer). The returned type is unsigned. So there is no need for type conversions. In line 26 type casting is used to convert the count signal of type unsigned to the std_logic_vector type [2].

Finite State Machines

The general form of a Finite State Machine (FSM) is shown in **Figure 8**. The flip-flops that save the current state Q consist the state register. The combinational circuit on the left calculates the next state using the current state, Q, and the inputs, W. The combinational circuit on the right calculates the outputs, Z, using as inputs the current state and the inputs, W. This is a Mealy type FSM because the outputs depend directly on the primary inputs. If the magenta line is ignored, **Figure 8** describes a Moore type FSM [1].

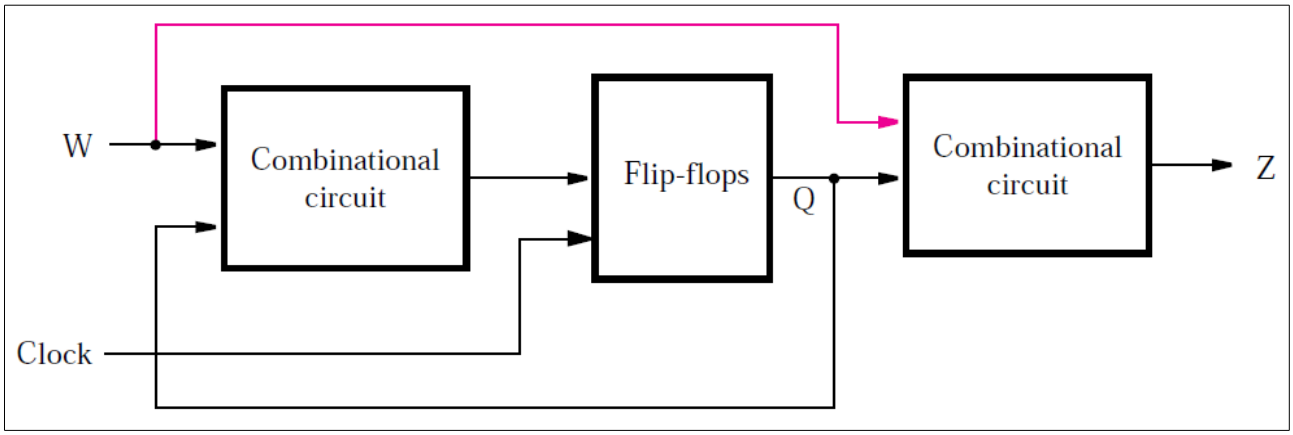

Figure 8

Figure 9 shows a state diagram of a Moore FSM.

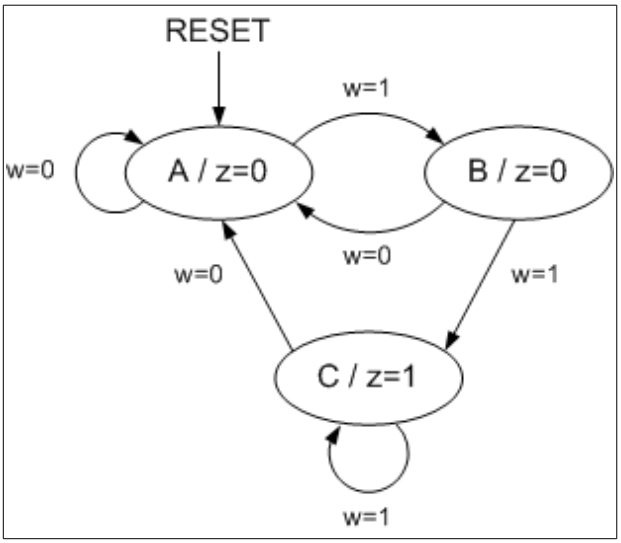

Figure 9

Figure 10 shows the VHDL code for the state diagram that is shown in **Figure 9**.

```
 1        library IEEE;
 2        use IEEE.STD_LOGIC_1164.ALL;
 3
 4        entity MOORE_1 is
 5           port(CLOCK, RESET, w : in STD_LOGIC;
 6                z : out STD_LOGIC);
 7        end MOORE_1;
 8
 9        architecture BEHAVIORAL of MOORE_1 is
10           type STATE_TYPE is (A, B, C);
11           signal y_present, y_next : STATE_TYPE;
12        begin
13           -- Combinational logic for next state calculation.
14           process(w, y_present)
15           begin
16              case y_present is
17                 when A =>
18                    if w = '0' then
19                       y_next <= A;
20                    else
21                       y_next <= B;
22                    end if;
23                 when B =>
24                    if w = '0' then
25                       y_next <= A;
26                    else
27                       y_next <= C;
28                    end if;
29                 when C =>
30                    if w = '0' then
31                       y_next <= A;
32                    else
33                       y_next <= C;
34                    end if;
35              end case;
36           end process;
37           -- State register.
38           process(CLOCK, RESET)
39           begin
40              if RESET = '1' then
41                 y_present <= A;
42              elsif CLOCK'EVENT and CLOCK = '1' then
43                 y_present <= Y_next;
44              end if;
45           end process;
46           --Combinational logic for output calculation.
47           z <= '1' when y_present = C else '0';
48        end BEHAVIORAL;
```

Figure 10

In line `10` (**Figure 10**) the enumeration type `STATE_TYPE` is declared. In an enumeration type declaration all the possible values that a signal of that type (in this case the type `STATE_TYPE`)

can have are designer specified [1]. In line `11` signals `y_present` and `y_next` of type `STATE_TYPE` are declared. These signals can have one of the values declared in line `10 (A, B, C)`. In lines `14-36` the combinational circuit used for next state calculation is described (the combinational circuit on the left in **Figure 8**). In lines `38-45` the state register is described and in line `47` the combinational circuit used for output calculation is described (the combinational circuit on the right in **Figure 8**).

The combinational circuit that is used for next state calculation and the state register can be described in a single process. **Figure 11** shows an alternative description for the architecture of the FSM described in **Figure 10**.

```
architecture BEHAVIORAL of MOORE_1 is
   type STATE_TYPE is (A, B, C);
   signal y : STATE_TYPE;
begin
   -- State register with the combinational
   -- logic for next state calculation.
   process(CLOCK, RESET)
   begin
      if RESET = '1' then
         y <= A;
      elsif CLOCK'EVENT and CLOCK = '1' then
         case y is
            when A =>
               if w = '0' then
                  y <= A;
               else
                  y <= B;
               end if;
            when B =>
               if w = '0' then
                  y <= A;
               else
                  y <= C;
               end if;
            when C =>
               if w = '0' then
                  y <= A;
               else
                  y <= C;
               end if;
         end case;
      end if;
   end process;
   --Combinational logic for output calculation.
   z <= '1' when y = C else '0';
end BEHAVIORAL;
```

Figure 11

In **Figure 11** only one signal is declared, signal `y`. This signal represents the current state of the FSM because is the output of the state register.

Figure 12 shows the architecture of the FSM of **Figure 10** with designer defined state encoding.

```
architecture BEHAVIORAL of MOORE_1 is
   signal y_present, y_next : STD_LOGIC_VECTOR(1 downto 0);
   constant A : STD_LOGIC_VECTOR(1 downto 0) := "00";
   constant B : STD_LOGIC_VECTOR(1 downto 0) := "01";
   constant C : STD_LOGIC_VECTOR(1 downto 0) := "11";
begin
   -- Combinational logic for next state calculation.
   process(w, y_present)
   begin
      case y_present is
         when A =>
            if w = '0' then
               y_next <= A;
            else
               y_next <= B;
            end if;
         when B =>
            if w = '0' then
               y_next <= A;
            else
               y_next <= C;
            end if;
         when C =>
            if w = '0' then
               y_next <= A;
            else
               y_next <= C;
            end if;
         when others =>
            y_next <= A;
      end case;
   end process;
   -- State register.
   process(CLOCK, RESET)
   begin
      if RESET = '1' then
         y_present <= A;
      elsif CLOCK'EVENT and CLOCK = '1' then
         y_present <= Y_next;
      end if;
   end process;
   --Combinational logic for output calculation.
   z <= '1' when y_present = C else '0';
end BEHAVIORAL;
```

Figure 12

Using enumeration type signals for state encoding the designer gives the synthesis tool the flexibility to choose the state encoding. The synthesis tool takes into account the design constrains (e.g. area, speed) in order to set the state encoding. In **Figure 12** two signals of `STD_LOGIC_VECTOR` type are used for current and next state representation. The length of the

`y_next` and `y_present` signals is 2 because at least 2 bits of information are needed to code 3 states. Three constants are declared for `A`, `B` and `C` state representation. The only difference in the process that describes the combinational circuit for next state calculation between **Figure 12** and **Figure 10** is that in **Figure 12** a `when others` syntax is used because the `y_present` signal is of `STD_LOGIC_VECTOR` type [1].

Figure 13 shows a state diagram of a Mealy FSM.

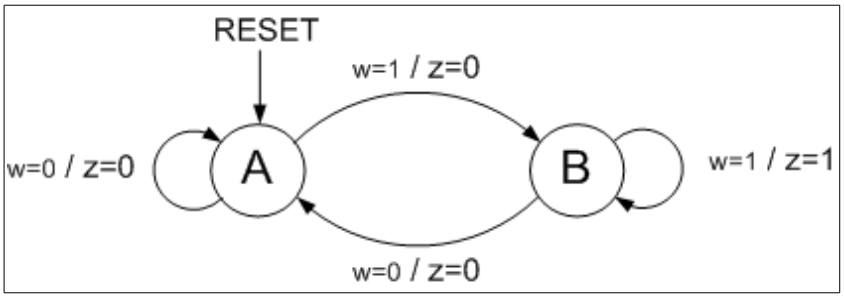

Figure 13

Figure 14 shows the VHDL code of the Mealy FSM shown in **Figure 13**.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity MEALY is
5      port(CLOCK, RESET, w : in STD_LOGIC;
6           z : out STD_LOGIC);
7   end MEALY;
8
9   architecture BEHAVIORAL of MEALY is
10     type STATE_TYPE is (A, B);
11     signal y : STATE_TYPE;
12  begin
13     -- State register with the combinational
14     -- logic for next state calculation.
15     process(CLOCK, RESET)
16     begin
17        if RESET = '1' then
18           y <= A;
19        elsif CLOCK'EVENT and CLOCK = '1' then
20           case y is
21              when A =>
22                 if w = '0' then
23                    y <= A;
24                 else
25                    y <= B;
26                 end if;
27              when B =>
28                 if w = '0' then
29                    y <= A;
30                 else
31                    y <= B;
32                 end if;
33           end case;
34        end if;
35     end process;
36     --Combinational logic for output calculation.
37     z <= '0' when y = A else w;
38  end BEHAVIORAL;
```

Figure 14

References and Further Reading

[1] Stephen Brown and Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd Edition, McGraw-Hill, 2009

[2] Edwin Naroska, comp.lang.vhdl Frequently Asked Questions And Answers, http://www.vhdl.org/comp.lang.vhdl/