

CPE 426/526

Chapter 8 - HDL-Based Design Techniques

Dr. Rhonda Kay Gaede

UAH

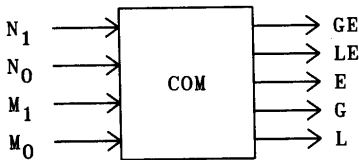
UAH

Chapter 8

CPE 426/526

8.1 Design of Combinational Logic Circuits (Array Method)

```
entity COM is
  generic (D:time);
  port (N1, N0, M1, M0: in BIT;
        GE, LE, E, G, L: out BIT);
end COM;
package TRUTH4x5 is
  constant NUM_OUTPUTS: INTEGER:=5;
  constant NUM_INPUTS: INTEGER:=4;
  constant NUM_ROWS: INTEGER:= 2 ** NUM_INPUTS;
  type WORD is array(NUM_OUTPUTS-1 downto 0) of BIT;
  type ADDR is array(NUM_INPUTS-1 downto 0) of BIT;
  type MEM is array (0 to NUM_ROWS-1) of WORD;
  constant TRUTH: MEM :=
    ("11100", "01001", "01001", "01001",
     "10010", "11100", "01001", "01001",
     "10010", "10010", "11100", "01001",
     "10010", "10010", "10010", "11100");
  function INTVAL(VAL:ADDR) return INTEGER;
end TRUTH4x5;
```



8.1 Design of Combinational Logic Circuits (Array Method)

```
-- Description of COM using table lookup.
use work.TRUTH4x5.all;
architecture TABLE of COM is
begin
  process (N1,N0,M1,M0)
    variable INDEX: INTEGER;
    variable WOUT: WORD;
  begin
    INDEX := INTVAL (N1&N0&M1&M0);
    WOUT := TRUTH (INDEX);
    GE <= WOUT(4) after D;
    LE <= WOUT(3) after D;
    E <= WOUT(2) after D;
    G <= WOUT(1) after D;
    L <= WOUT(0) after D;
  end process;
end TABLE;
```

N ₁	N ₀	M ₁	M ₀	GE	LE	E	G	L
0	0	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	1
0	0	1	0	0	1	0	0	1
0	0	1	1	0	1	0	0	1
0	1	0	0	1	0	0	1	0
0	1	0	1	1	1	1	0	0
0	1	1	0	0	1	0	0	1
0	1	1	1	0	1	0	0	1
1	0	0	0	1	0	0	1	0
1	0	0	1	1	0	0	1	0
1	0	1	0	1	1	1	0	0
1	0	1	1	0	1	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	0
1	1	1	0	1	0	0	1	0
1	1	1	1	1	1	1	0	0

--Figure 8.4 VHDL model for device COM using the ARRAY method.

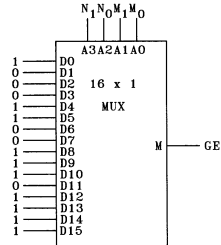
8.1 Design of Combinational Logic Circuits (Algorithmic Method)

```
entity COM_S is
  port (N1, N0, M1, M0: in std_logic;
        GE, LE, E, G, L: out std_logic);
end COM_S;
architecture ALG of COM_S is
begin
  process (N1,N0,M1,M0)
    variable right, left : std_logic_vector (1 downto 0);
  begin
    GE <= '0'; E <= '0'; LE <= '0'; G <= '0'; L <= '0';
    right := N1&N0;
    left := M1&M0;
    if (right >= left) then
      GE <= '1';
    elsif (right = left) then
      E <= '1';
    elsif (right <= left) then
      LE <= '1';
    elsif (right > left) then
      G <= '1';
    elsif (right < left) then
      L <= '1';
    end if;
  end process;
end ALG;
```

8.1 Design of Combinational Logic Circuits - CASE Statement

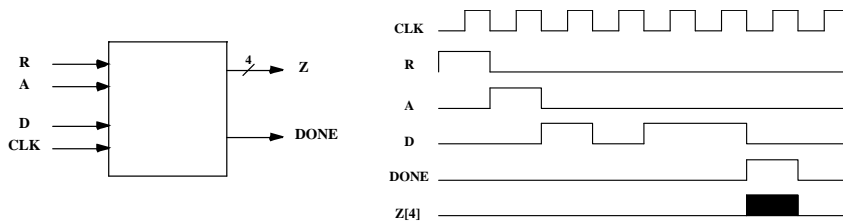
```

architecture MUX of COM is
begin
  process(N1,N0,M1,M0)
  variable aggregate : BIT_VECTOR (3 downto 0);
  begin
    aggregate := N1&N0&M1&M0;
    case aggregate is
      when "0000" => GE <= '1' after D; LE <= '1' after D;
        E <= '1' after D; G <= '0' after D; L <= '0' after D;
      when "0001" => GE <= '0' after D; LE <= '1' after D;
        E <= '0' after D; G <= '0' after D; L <= '1' after D;
      when "0010" => GE <= '0' after D; LE <= '1' after D;
        E <= '0' after D; G <= '0' after D; L <= '1' after D;
      when "0011" => GE <= '0' after D; LE <= '1' after D;
        E <= '0' after D; G <= '0' after D; L <= '1' after D;
      when "0100" => GE <= '1' after D; LE <= '0' after D;
        E <= '0' after D; G <= '1' after D; L <= '0' after D;
      when "0101" => GE <= '1' after D; LE <= '1' after D;
        E <= '1' after D; G <= '0' after D; L <= '0' after D;
      .
      .
      when "1111" => GE <= '1' after D; LE <= '1' after D;
        E <= '1' after D; G <= '0' after D; L <= '0' after D;
    end case;
  end process;
end MUX;
  
```



8.3 Design of Sequential Logic Circuits - Example

Design a serial to parallel converter. Input CLK is the clock that controls all operations in the system. Reset signal R is synchronous. If R = '1' at the end of any clock period, the device must enter the reset state. Input A is asserted for exactly one clock period prior to the arrival of serial data on input D. For the next four clock periods, data arrives serially on line D. The device must collect the four bits of serial data and output them in parallel at output Z, which is a 4-bit vector. During the clock period when the parallel data is present at Z, signal DONE is asserted. The outputs Z and DONE must remain asserted for one full clock period. DONE alerts the destination device that data is present on Z. During the clock period when the parallel data is present on Z, the device may receive another pulse on line A indicating that new data will be arriving on line D during the following clock period. If so, it must be prepared to receive that data. If not, the device goes to the reset state after sending out the parallel data and waits for new data to arrive.



8.3 Design of Sequential Logic Circuits - State Diagram

8.3 Design of Sequential Logic Circuits - Control and Data Style

```
-- Serial to Parallel Converter
entity STOP is
  port (R, A, D, CLK: in BIT;
        Z: out BIT_VECTOR(3 downto 0);
        DONE: out BIT);
end STOP;
architecture FSM_RTL of STOP is
  type STATE_TYPE is (S0, S1, S2, S3, S4, S5);
  signal STATE: STATE_TYPE;
  signal SHIFT_REG: BIT_VECTOR (3 downto 0);
begin
  NEXT_STATE: process (CLK)
  begin
    if CLK='1' then
      case STATE is
        when S0 =>
          if R='1' or A='0' then
            STATE <= S0;
          elsif R='0' and A='1' then
            STATE <= S1;
          end if;
        when S1 =>
          SHIFT_REG <= D & SHIFT_REG(3 downto 1);
          if R='0' then
            STATE <= S2;
          elsif R='1' then
            STATE <= S0;
          end if;
      end case;
    end if;
  end process;
end;
```

8.3 Design of Sequential Logic Circuits - Control and Data Style

```
when S2 =>
    SHIFT_REG <= D & SHIFT_REG(3 downto 1);
    if R='0' then
        STATE <= S3;
    elsif R='1' then
        STATE <= S0;
    end if;
when S3 =>
    SHIFT_REG <= D & SHIFT_REG(3 downto 1);
    if R='0' then
        STATE <= S4;
    elsif R='1' then
        STATE <= S0;
    end if;
when S4 =>
    SHIFT_REG <= D & SHIFT_REG(3 downto 1);
    if R='0' then
        STATE <= S5;
    elsif R='1' then
        STATE <= S0;
    end if;
when S5 =>
    if R='0' and A='1' then
        STATE <= S1;
    elsif R='1' or A='0' then
        STATE <= S0;
    end if;
end case;
end if;
end process NEXT_STATE;
Electrical and Computer Engineering
```

8.3 Design of Sequential Logic Circuits - Control and Data Style

```
--
-- Output process
--
OUTPUT: process (STATE)
begin
    case STATE is
        when S0 to S4 =>
            DONE <= '0';
        when S5 =>
            DONE <= '1';
            Z <= SHIFT_REG;
        end case;
    end process OUTPUT;
end FSM_RTL;
```

8.3 Design of Sequential Logic Circuits - State Table Style

```

entity TWO_CONSECUTIVE is
  port(CLK,R,X: in BIT; Z: out BIT);
end TWO_CONSECUTIVE;
architecture FSM of TWO_CONSECUTIVE is
  type STATE is (S0,S1,S2);
  signal FSM_STATE: STATE := S0;
  type TRANSITION is record
    OUTPUT: BIT;
    NEXT_STATE: STATE;
  end record;
  type TRANSITION_MATRIX is array(STATE,BIT) of TRANSITION;
  constant STATE_TRANS: TRANSITION_MATRIX :=
    (S0 => ('0' => ('0',S1), '1' => ('0',S2)),
     S1 => ('0' => ('1',S1), '1' => ('0',S2)),
     S2 => ('0' => ('0',S1), '1' => ('1',S2)));
begin
  process(R,X,CLK,FSM_STATE)
  begin
    if R = '0' then -- Reset
      FSM_STATE <= S0;
    elsif CLK'EVENT and CLK = '1' then -- Clock event
      FSM_STATE <= STATE_TRANS(FSM_STATE,X).NEXT_STATE;
    end if;
    if FSM_STATE'EVENT or X'EVENT then -- Output Function
      Z <= STATE_TRANS(FSM_STATE,X).OUTPUT;
    end if;
  end process;
end FSM;

```

Page 11 of 12

8.3 Design of Sequential Logic Circuits - Moore State Table

```

architecture TABLE_MOORE of TWO_CONSECUTIVE is
  type STATE is (S0, S1, S2, S3, S4);
  signal FSM_STATE: STATE := S0;
  type TRANSITION_MATRIX is array (STATE, BIT) of STATE;
  type OUTPUT_MATRIX is array (STATE) of BIT;
  constant STATE_TRANS: TRANSITION_MATRIX :=
    (S0 => ('0' => S1, '1' => S3),
     S1 => ('0' => S2, '1' => S3),
     S2 => ('0' => S2, '1' => S3),
     S3 => ('0' => S1, '1' => S4),
     S4 => ('0' => S1, '1' => S4));
  constant OUTPUT : OUTPUT_MATRIX :=
    (S2 => '1', S4 => '1', OTHERS => '0');
begin
  process(R, CLK, FSM_STATE)
  begin
    if (R = '0') then
      FSM_STATE <= S0;
      Z <= '0';
    elsif (CLK'EVENT and CLK = '1') then -- Clock event
      FSM_STATE <= STATE_TRANS(FSM_STATE, X);
    end if;
    Z <= OUTPUT(FSM_STATE);
  end process;
end TABLE_MOORE;

```

Page 12 of 12