บทที่ 7
การออกแบบวงจรดิจิตอลด้วย VHDL

7.1 บทนำ

การออกแบบวงจรดิจิตอลด้วย VHDL นี้ ได้มีการยกตัวอย่างเพื่อประกอบการอธิบายในหัวข้อต่างๆที่ผ่านมาบ้างแล้ว แต่ยังไม่ได้แสดงให้เห็นเป็นขั้นตอนที่ชัดเจน สำหรับในบทนี้ได้แสดงให้เห็นการใช้ความสามารถของ VHDL มาออกแบบวงจรดิจิตอลทั้งวงจรคอมไบเนชั่นนอล และวงจรซีเควนเชียล โดยจะใช้ออกแบบตั้งแต่ระดับอธิบายเป็นพฤติกรรมการทำงานของวงจร จนถึงระดับการเขียนด้วยสมการบูลีน การอธิบายนี้จะใช้วิธียกตัวอย่างวงจรด้วยมาตราฐานต่างๆเช่น วงจรเข้ารหัส วงจรถอดรหัส วงจรรีจิสเตอร์ วงจรนับ วงจรสเตทแมชชีน และวงจรหน่วยความจำ บางวงจรได้แสดงการออกแบบด้วยเทคนิคหลายๆแบบเพื่อให้เห็นถึงความสามารถของ VHDL และความแตกต่างของวงจรเมื่อสังเคราะห์ได้

7.2 การออกแบบวงจรคอมไบเนชั่นนอลโลจิกด้วย VHDL

วงจรคอมไบเนชั่นนอลโลจิกเป็นวงจรโลจิกที่สัญญาณเอาท์พุทขึ้นอยู่กับสัญญาณอินพุทเพียงอย่างเดียง การออกแบบวงจรคอมไบเนชั่นนอลโลจิกประกอบด้วยขั้นตอนหลักๆดังต่อไปนี้

ขั้นตอนที่ 1 กำหนดหน้าที่การทำงาน
ขั้นตอนที่ 2 กำหนดตัวแปร และค่าของตัวแปร
ขั้นตอนที่ 3 เขียนตารางการทำงาน
ขั้นตอนที่ 4 ลดทอนฟังก์ชั่นและเขียนเป็นฟังก์ชั่นบูลีน
ขั้นตอนที่ 5 เขียนโลจิกไดอะแกรม (Logic diagram)
ขั้นตอนที่ 6 ทดสอบการทำงาน

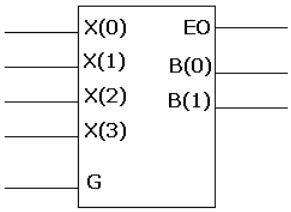
จากขั้นตอนการออกแบบที่กล่าวนี้ ขั้นตอนที่ 1 เป็นขั้นตอนที่ระบุหน้าที่การทำงานของวงจร ซึ่งอาจจะอยู่ในรูปของการอธิบายเป็นคำพูด หรือเป็นไดอะแกรมเวลาก็ได้ ส่วนขั้นตอนที่ 4 เป็นขั้นตอนที่จะนำไปสร้างเป็นวงจรนั้น ต้องผ่านขบวนการวิเคราะห์โดยผู้ออกแบบ แล้วนำมาเขียนเป็นตารางการทำงาน และจึงทำเป็นฟังก์ชั่นตามขั้นตอนที่ 4 ซึ่งจะเห็นได้ว่า ถ้าการทำงานของวงจรมีความซับซ้อนมาก ก็ต้องใช้เวลาในการเปลี่ยนจากขั้นตอนที่ 1 ไปเป็นขั้นตอนที่ 4 มาก และโอกาสผิดพลาดก็มากตามไปด้วย แต่ VHDL สามารถนำมาใช้ได้ตั้งแต่ขั้นตอนที่ 1 หรือจะใช้ในขั้นตอนที่ 3 หรือเป็นขั้นตอนที่ 4 ก็ได้ ทำให้ลดข้อผิดพลาดและลดเวลาในการออกแบบได้อย่างมาก ดังจะเห็นได้จากตัวอย่างต่างๆต่อไปนี้

7.2.1 วงจรเข้ารหัส (Encoder)

วงจรเข้ารหัสที่จะกล่าวถึงนี้เป็นวงจรเข้ารหัสแบบไบนารี (Binary Encoder) วงจรจะทำหน้าที่เข้ารหัสสัญญาณอินพุท ให้เป็นสัญญาณไบนารี โดยสัญญาณอินพุทจะเป็นสัญญาณอะไรก็ได้ แต่สัญญาณเอาท์พุทจะออกมาเป็นสัญญาณไบนารี จำนวนบิตของอินพุท จะมีค่าเท่ากับ 2^n และจำนวนบิตของเอาท์พุทจะมากกว่าหรือเท่ากับ n

ตัวอย่างที่ 7.1 วงจรเข้ารหัสแบบไพรออริตี้ขนาด 4 อินพุท

เป็นวงจรเข้ารหัสไบนารี ที่มีการจัดลำดับความสำคัญของสัญญาณอินพุทให้ไม่เท่ากัน ตามไดอะแกรมในรูปที่ 7-1 สัญญาณอินพุท X(3) เป็นสัญญาณที่มีลำดับความสำคัญสูงสุด ส่วนสัญญาณอินพุท X(0) จะมีลำดับความสำคัญต่ำสุด ดังนั้นถ้ามีสัญญาณอินพุทต้องการเข้ารหัสพร้อมกันมากกว่า 1 สัญญาณ สัญญาณที่ความสำคัญสูงกว่า จะได้รับการเข้ารหัส



รูปที่ 7-1 ไดอะแกรมวงจรเข้ารหัสแบบไพรออริตี้ขนาด 4 อินพุท

สัญญาณ X(3) X(2) X(1) และ X(0) เป็นสัญญาณอินพุท
G เป็นสัญญาณควบคุมทางอินพุท
EO เป็นสัญญาณสถานะของเอาท์พุท ใช้แสดงสถานะว่ามีการเข้ารหัส
B(1) และ B(0) เป็นสัญญาณเอาท์พุท

การทำงาน ถ้าสัญญานอินพุท G เป็น 1 สัญญาณเอาท์พุท B(1) B(0) จะให้ค่าเลขไบนารี่ ตามสถานะอินพุท X โดยถ้ามีอินพุท X เป็น 0 พร้อมกันมากกว่า 1 บิต จะให้ค่าไบนารีของอินพุท X ที่มีค่ามากกว่า เช่น ถ้า X(2) เป็น 0 พร้อม X(1) สัญญาณเอาท์พุท B จะได้เป็นเลข 2 คือ B(1) = 1 และ B(0) = 0 พร้อมกันนี้สัญญาณเอาท์พุท EO จะเป็น 1 เพื่อแสดงว่ามีการเข้ารหัส แต่ถ้าสัญญาณ G เป็น 0 สัญญาณเอาท์ B(1) B(0) และ EO จะเป็น 0 ทั้งหมด

การทำงานที่กล่าวถึงข้างต้นนี้ จัด อยู่ในขั้นตอนที่ 1 ของขั้นตอนการออกแบบ เมื่อกำหนดค่าตัวแปรต่างๆสามารถแปลงคำอธิบายการทำงานให้เป็น ตารางการทำงาน ตามขั้นตอนที่ 3 ได้ดังนี้

ตารางที่ 7-1 ตารางการทำงานของวงจรเข้ารหัสแบบไพรออริตี้ขนาด 4 อินพุท

Input					Output		
G	X(3)	X(2)	X(1)	X(0)	EO	B(1)	B(0)
0	x	x	x	x	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	0	1	0	0
1	1	1	0	x	1	0	1
1	1	0	x	x	1	1	0
1	0	x	x	x	1	1	1

x = don’t care

และสุดท้ายเมื่อนำค่าเอาท์พุทมาลดทอนฟังก์ชั่น สามาถเขียนเป็นฟังก์ชั่นบูลีนได้ ตามสมการ (7-1)

$$EO = G.\left(\overline{X(3).X(2).X(1).X(0)}\right)$$
$$B(1) = G.\left(\overline{X3}.\overline{X2}\right) \qquad (7\text{-}1)$$
$$B(0) = G.\left(\overline{X3} + X2.\overline{X1}\right)$$

การออกแบบด้วย VHDL แบบที่ 1 ออกแบบในระดับพฤติกรรมการทำงานของวงจร

จากการทำงานของวงจรซึ่งมีลักษณะเป็นเงื่อนไข ของสัญญาณอินพุท โดยถ้าเป็นจริงจะให้สัญญาณเอาท์พุทเป็นแบบหนึ่ง แต่ถ้าไม่จริงก็ให้สัญญาณเอาท์พุทเป็นอีกแบบหนึ่ง การเขียนเป็น VHDL จึงใช้คำสั่ง IF.... ELSE..... แต่คำสั่งนี้เป็นคำสั่งแบบลำดับ ต้องอยู่ใน PROCESS ดังนั้นสามารถเขียนเป็น VHDL ได้ดังนี้

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity encoder  is
  port (x : in std_logic_vector(3 downto 0);
         G : in std_logic;
         B : out std_logic_vector(1 downto 0);
       EO: out std_logic);
end encoder;

architecture Behavioral of encoder  is
begin
process (x, G)
  begin
     if (G = '0')  then
               EO <= '0';
               B <= "00";
     elsif  (x(3) = '0')  then
               EO <= '1';
               B  <= "11";
     elsif  (x(2) = '0')  then
               EO <= '1';
               B  <= "10";
     elsif  (x(1) = '0')  then
               EO <= '1';
               B  <= "01";
     elsif  (x(0) = '0')  then
               EO <= '1';
               B  <= "00";
     else
               EO <= '0';
               B  <= "00";
     end if;
  end process;
end Behavioral;
```

การออกแบบด้วย VHDL แบบที่ 2 ออกแบบในระดับตารางการทำงานของวงจร

แบบนี้ก็จัดอยู่ในรูปแบบพฤติกรรมการทำงานเช่นเดียวกับแบบแรก แต่วิธี อธิบายการทำงานเริ่มเป็นรูปธรรม คือเป็น ตารางการทำงาน คำสั่งที่เหมาะสมกับการเขียนแทนตารางการทำงานมีหลายแบบ ถ้าเป็นคำสั่งในกลุ่มคำสั่งแบบขนานก็คือคำสั่ง WITH ..... SELECT แต่ถ้าเป็นคำสั่งแบบลำดับก็เป็นคำสั่ง CASE ..... IS สำหรับตัวอย่างนี้ใช้ คำสั่ง WITH ..... SELECT ดังนี้

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity encoder  is
  port (x : in std_logic_vector(3 downto 0);
         G : in std_logic;
         B : out std_logic_vector(1 downto 0);
       EO: out std_logic);
end encoder;

architecture Behavioral2 of encoder  is
   signal sx: std_logic_vector(4 downto 0);
   signal sy: std_logic_vector(2 downto 0);
begin
   sx <= g&x;
   b <= sy(1 downto 0);
   EO <= sy(2);
  with sx select
      sy <= "111" when "10000",
                "111" when "10001",
                "111" when "10010",
                "111" when "10011",
                "111" when "10100",
                "111" when "10101",
                "111" when "10110",
                "111" when "10111",
                "110" when "11000",
                "110" when "11001",
                "110" when "11010",
                "110" when "11011",
                "101" when "11100",
                "101" when "11101",
                "100" when "11110",
                "000" when others;
end Behavioral2;
```

การออกแบบด้วย VHDL แบบที่ 3 ออกแบบจากฟังก์ชั่นบูลีน

สำหรับแบบที่ 3 นี้ แต่ผู้ออกแบบต้องออกแบบด้วยต้นเองมาก่อน จนได้เป็นฟังก์ชั่น ซึ่งทำให้มีโอกาสผิดพลาดได้มาก แต่ VHDL ที่เขียนมักจะสั้น และเขียนด้วยคำสั่งแบบขนานได้ดังนี้

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity encoder  is
  port (x : in std_logic_vector(3 downto 0);
         G : in std_logic;
         B : out std_logic_vector(1 downto 0);
       EO: out std_logic);
end encoder;

architecture dataflow of encoder  is

begin
   B(0) <= G and(not x(3) or (x(2) and not(x(1))));
   B(1) <= G and (not(X(3) and X(2)));
    EO <= G and (not (x(3) and x(2) and x(1) and x(0)));
end dataflow;
```

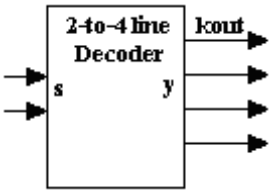
7.2.2 วงจรถอดรหัส (Decoder)

วงจถอดรหัสทำงานตรงข้ามกับวงจรเข้ารหัส สัญญาณอินพุทเป็นสัญญาณไบนารี ส่วนสัญญาณเอาท์พุทเป็นสัญญาณอะไรก็ได้ สัญญาณเอาท์พุทจะมีจำนวนน้อยว่าหรือเท่ากับ 2^n โดย n เป็นจำนวนสัญญาณอินพุท

ตัวอย่าง 10.2 วงจรถอดรหัส 2 ออก 4

สัญญาณ S1 และ S0 เป็นสัญญาณอินพุท

Y3 Y2 Y1 และ Y0 เป็นสัญญาณ์เอาท์พุท



รูปที่ 7-2 ไดอะแกรมวงจรถอดรหัส 2 ออก 4

การทำงาน ในแต่ละค่าของสัญญาณอินพุท จะมีสัญญาณเอาท์พุทเป็นโลจิก 0 (หมายถึงสัญญาณเอาท์พุทที่แอคทีฟ) เพียง 1 สัญญาณเท่านั้น ดังตารางที่ 7-2

ตารางที่7-2 ตารางการทำงานของวงจรถอดรหัส 2 ออก 4

S1	S0	Y3	Y2	Y1	Y0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

โมเดล VHDL แบบที่ 1 ออกแบบในระดับตารางการทำงานของวงจร

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder1 is
   Port ( s : in std_logic_vector(1 downto 0);
          y : out std_logic_vector(3 downto 0));
end decoder1;
architecture Behavioral of decoder1 is

begin
            process (s)
            begin
              case s is
                    when "00" =>
                                y <= “1110”;
                      when "01" =>
                                 y <= “1101”;
                      when "10" =>
                                y <= “1011”;
                      when others =>
                                 y <= “0111”;
                      end case;
            end process;
end Behavioral;
```

โมเดล VHDL แบบที่ 2 ออกแบบจากฟังก์ชั่นบูลีน

จากตารางที่ 7-2 สามารถเขียนเป็นฟังก์ชั่นบูลีน และ โมเด็ล VHDL ดังนี้

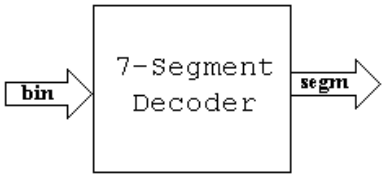
$$Y0 = S1+S0$$
$$Y1 = S1+\overline{S0} \qquad (7\text{-}2)$$
$$Y2 = \overline{S1}+S0$$
$$Y3 = \overline{S1}+\overline{S0}$$

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder1 is
   Port ( s : in std_logic_vector(1 downto 0);
          y : out std_logic_vector(3 downto 0));
end decoder1;
architecture Behavioral of decoder1 is

begin
          Y(0)<=S(1) or S(0);
          Y(1)<=S(1) or not S(0);
          Y(2)<=not S(1) or S(0);
          Y(3)<=not S(1) or not S(0);
end Behavioral;
```

ตัวอย่างที่ 7.3 วงจรถอดรหัสบีซีดี (BCD) เพื่อไปขับแอลอีดี 7 ส่วน

สัญญาณ bin3 bin2 bin1 และ bin0 เป็นสัญญาณอินพุท

Segm7 ถึง segm0 เป็นสัญญาณ์เอาท์พุท



รูปที่ 7-3 ไดอะแกรมวงจรถอดรหัสบีซีดี (BCD) เพื่อไปขับแอลอีดี 7 ส่วน

การทำงาน เมื่อป้อนรหัสบีซีดีเข้าที่อินพุท จะได้รับสัญญาณเอาท์พุทที่ทำให้แอลอีดี 7 ส่วนติดเป็นตัวเลขตามรหัสบีซีดีนั้นๆ สำหรับในตัวอย่างนี้ จะใช้สำหรับแอลอีดีแบบแอโหนดร่วม ดังนั้นแอลอีดีส่วนที่ติดต้องเป็นโลจิก 0 ตามตารางที่ 7-3

ตารางที่ 7-3 ตารางการทำงานของวงจรถอดรหัสบีซีดี (BCD) เพื่อไปขับแอลอีดี 7 ส่วน

bin3	bin2	bin1	bin0	segm7	segm6	segm5	segm4	segm3	segm2	segm1	segm0
0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	1
0	0	1	1	0	0	0	0	1	1	0	1
0	1	0	0	1	0	0	1	1	0	0	1
0	1	0	1	0	1	0	0	1	0	0	1
0	1	1	0	0	1	0	0	0	0	0	1
0	1	1	1	0	0	0	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	0	0	0	0	1	0	0	1
1	0	1	1	0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	0	1	0	0	1
1	1	0	1	0	0	0	0	1	0	0	1
1	1	1	0	0	0	0	0	1	0	0	1
1	1	1	1	0	0	0	0	1	0	0	1

โมเดล VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec2seg is
  port (bin : in std_logic_vector(3 downto 0);
              segm : out std_logic_vector(7 downto 0));
end dec2seg;

architecture Behavioral of dec2seg is
begin
        with bin select
                segm <= "00000011" when "0000",
                        "10011111" when "0001",
                        "00100101" when "0010",
                        "00001101" when "0011",
                        "10011001" when "0100",
                        "01001001" when "0101",
                        "01000001" when "0110",
                        "00011111" when "0111",
                        "00000001" when "1000",
                        "00001001" when others;
end Behavioral;
```

7.2.3 วงจรคณิตศาสตร์ (Arithmetic)

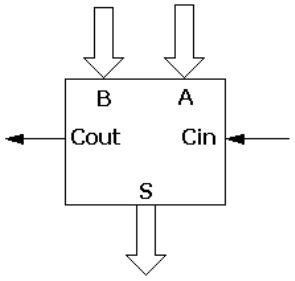
วงจรคณิตศาสตร์ถ้าแบ่งตามหน้าที่ก็มีหลายแบบเช่น วงจรบวก วงจรคูณ วงจรยกกำลัง และถ้าแบ่งตามวิธีการออกแบบก็มีหลายแบบเช่นกัน สามารถออกแบบเป็นวงจรแบบคอมไบเนชั่นนอลอย่างเดียวก็ได้ หรือออกแบบให้มีวงจรซีเควนเชียลผสมก็ได้ สำหรับในที่นี้จะยกตัวอย่างเพียงแบบแรกเท่านั้น

ตัวอย่างที่ 7.4 วงจรบวกเลข มีบิตตัวทดเข้าและบิตตัวทดออก

สำหรับตัวอย่างนี้ ต้องการแสดงให้เห็นวิธีการ ออกแบบวงจรบวกเลขที่เขียนจากพฤติกรรมการทำงานของวงจรโดยตรง ด้วยการใช้วิธีแปลงชนิดข้อมูลของสัญญาณอินพุทจากแบบ std_logic_vector เป็น integer แล้วใช้การบวกเลข (+) เพื่อให้ได้ผลลัพธ์ หลังจากนั้นจึงแปลงชนิดของผลลัพธ์จากแบบ integer

กลับเป็น std_logic_vector เพื่อส่งออกไปเป็นสัญญาณเอาท์พุท ฟังก์ชั่น ทั้งสองนี้อยู่ในไลบรารี std_logic_unsigned.vhdl นอกจากนี้แล้ว VHDL ที่ออกแบบยังสามารถปรับเปลี่ยนขนาดหรือจำนวนบิตของวงจรได้ โดยการใช้คำสั่ง generic เพื่อกำหนดจำนวนบิตไว้ในต้อนต้นของโมเดล ตามตัวอย่างได้กำหนดไว้เป็นแบบ 8 บิต

สัญญาณ A B Cin เป็นสัญญาณอินพุท ตัวตั้ง ตัวบวก ตัวทดเข้า ตามลำดับ
S Cout เป็นสัญญาณเอาท์พุท ผลบวก และตัวทดออก



รูปที่ 7-4 ไดอะแกรมวงจรบวก

การทำงาน สามารถเขียนเป็นสมการได้ดังนี้

Cout S = A + B + Cin — การบวกเลข

โมเดล VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity adder is
        generic (width: integer := 8);
        Port (A : in std_logic_vector(width-1 downto 0);
              B : in std_logic_vector(width-1 downto 0);
             Cin : in std_logic;
              S : out std_logic_vector(width-1 downto 0);
            Cout : out std_logic);
end adder;
architecture Behavioral of adder is
SIGNAL A_i, B_i : integer range 0 to 2**width;
SIGNAL S_i : integer range 0 to 2**(width+1);
SIGNAL S_s : std_logic_vector(width downto 0);

begin
-- convert from std_logic_vector to integer
        A_i <= (conv_integer(A));
        B_i <= (conv_integer(B));
        process(A_i,B_i,Cin)
        begin
                if(Cin='0') then
                        S_i <= A_i + B_i;
                else
                        S_i <= A_i + B_i + 1;
                end if;
        end process;
```

จำนวนบิต

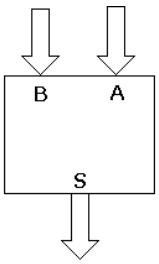
ใช้การบวกเลขโดยตรง ถ้ามีตัวทดก็บวกเพิ่มอีก 1

```
        -- convert from integer to a 8 bit std_logic_vector
        S_s <= (conv_std_logic_vector(S_i,width+1));

        S <= S_s(width-1 downto 0);
        Cout <= S_s(width);
end Behavioral;
```

ตัวอย่างที่ 7.5 วงจรคูณเลขจำนวนเต็ม

ตัวอย่างนี้ก็เช่นเดียวกับตัวอย่างที่ 7.4 ใช้วิธีการเขียนจากพฤติกรรมการทำงานของวงจรโดยตรง ขนาดหรือจำนวนบิตก็สามารถปรับเปลี่ยนได้ แต่จำนวนบิตของผลคูณจะเท่ากับผลรวมของจำนวนบิตตัวตั้งกับตัวคูณ ดังนั้นถ้าให้จำนวนบิตของตัวตั้งกับตัวคูณเท่ากัน จำนวนบิตของผลคูณก็เป็น 2 เท่าของจำนวนบิตตัวตั้งหรือตัวคูณ

สัญญาณ A เป็นสัญญาณอินพุท ตัวตั้ง
B เป็นสัญญาณอินพุท ตัวคูณ
S เป็นสัญญาณเอาท์พุท ผลคูณ



รูปที่ 7-5 ไดอะแกรมวงจรคูณ

โมเดล VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity multiplier is
        generic (width: integer :=8);
        Port (A : in std_logic_vector(width-1 downto 0);
                B : in std_logic_vector(width-1 downto 0);
               S : out std_logic_vector((2*width)-1 downto 0));
end multiplier;
architecture Behavioral of multiplier is
SIGNAL A_i, B_i : integer range 0 to 2**width;
SIGNAL S_i : integer range 0 to 2**(2*width);

begin
--          convert from std_logic_vector to integer
        A_i <= (conv_integer(A));
        B_i <= (conv_integer(B));
        S_i <= A_i*B_i;
--          convert from integer to a 8 bit std_logic_vector
        S <= (conv_std_logic_vector(S_i,2*width));
end Behavioral;
```

7.2.4 วงจรคอมไบเนชั่นนอลแบบอื่นๆ

นอกจากการออกแบบวงจรที่กล่าวมาข้างต้นแล้ว ยังมีวงจรอีกหลายแบบที่เป็นวงจรคอมไบเนชั่นนอล ซึ่งมีวิธีการออกแบบที่แตกต่างกันออกไป ในที่นี้จะขอยกตัวอย่างเพิ่มเติมอีก 2 วงจรคือวงจรตรวจสอบพาริตี้ และวงจรโลจิกแบบ 3 สถานะ

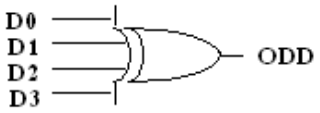
ตัวอย่างที่ 7.6 วงจรตรวจสอบค่าพาริตี้ (Parity) ของข้อมูลขนาด 4 บิต ถ้ามีพาริตี้เป็นคี่ (ODD) จะให้ค่าเอาท์พุทเป็น 1

สัญญาณ D3 – D0 เป็นสัญญาณอินพุท
ODD เป็นสัญญาณ์เอาท์พุท

การทำงาน สามารถเขียนเป็นโลจิกฟังก์ชั่นดังนี้

$$ODD = D3 \oplus D2 \oplus D1 \oplus D0$$

สำหรับตัวอย่างนี้ จะแสดงรูปแบบการเขียน 2 แบบ แบบแรก เหมือนกับการเขียน โลจิกไดอะแกรมตรงๆ แบบนี้การนำไปใช้ใหม่ต้องแก้ไขคำสั่งในบรรทัด ODD <= D(0) xor D(1) xor D(2) xor D(3); นี้ตลอด ถ้ายิ่งข้อมูลมีจำนวนมากบิต ก็ต้องแก้ไขมาก ส่วนอีกวิธีจะปรับเปลี่ยนจำนวนบิตได้ ตามตัวอย่างที่แสดงไว้ในแบบที่ 2

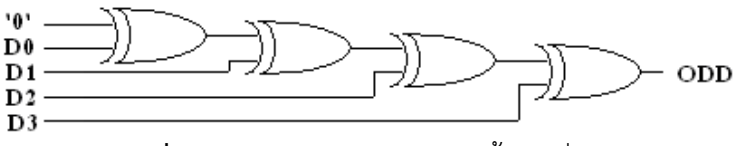


รูปที่ 7-6 วงจรตรวจสอบค่าพาริตี้แบบที่ 1

โมเดล VHDL

```
entity PARITY is
    port (D: in   bit_vector (3 downto 0);
          ODD : out bit);
end PARITY;
architecture RTL of PARITY is
begin
   ODD <= D(0) xor D(1) xor D(2) xor D(3);
end RTL;
```

แบบที่ 2 ทำเหมือนกับโลจิกไดอะแกรมในรูปที่ 7-5 โดยใช้คำสั่ง GENERIC เพื่อกำหนดจำนวนบิต และใช้คำสั่ง FOR LOOP ทำให้การเปลี่ยนแปลงจำนวนบิตข้อมูลทำเพียงแก้ไขค่าของ width เท่านั้น



รูปที่ 7-7 วงจรตรวจสอบค่าพาริตี้แบบที่ 2

โมเดล VHDL

```
entity PARITY is
generic (width: integer := 4);
    port (D: in   bit_vector (width-1  downto 0);
          ODD : out bit);
```

จำนวนบิตข้อมูล

```
end PARITY;
architecture RTL of PARITY is
begin
    process (D)
      variable TMP : bit;
    begin
      TMP := `0`;

      for I in  D`low to D`high  loop
        TMP := TMP xor D(I);
      end loop;
      ODD <= TMP;
    end process;
end RTL;
```

ตัวอย่างที่ 7.7 อุปกรณ์โลจิกแบบแบบ 3 สถานะ

สัญญาณ a เป็นสัญญาณอินพุท

e เป็นสัญญาณควบคุม ถ้ามีค่าเป็น ‘0’ เอาท์พุทจะเป็น ‘Z’ (High impedance)

y เป็นสัญญาณ์เอาท์พุท

โมเดล VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity highz is
   Port ( a, e : in  std_logic;
          y : out  std_logic);
end highz;

architecture Behavioral of highz is

begin
            with e select
              y <= 'Z' when '0',
                   a when others;
end Behavioral;
```

High impedance

ถ้ากรณีมีอินพุทและเอาท์พุทมากกว่า 1 แต่ใช้สัญญาณควบคุมอันเดียวกันสามารถเขียนได้ดังนี้

โมเดล VHDL

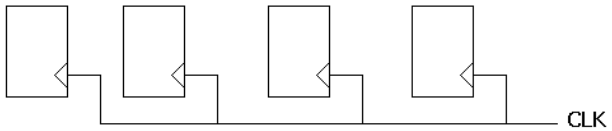
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bufferz is
   Port ( a : in std_logic_vector(7 downto 0);
            e : in  std_logic;
           y : out  std_logic_vector(7 downto 0););
end bufferz;
```

```
architecture Behavioral of bufferz is
begin
        with e select
          y <= “ZZZZZZZZ” when '0',
               a when others;
end Behavioral;
```

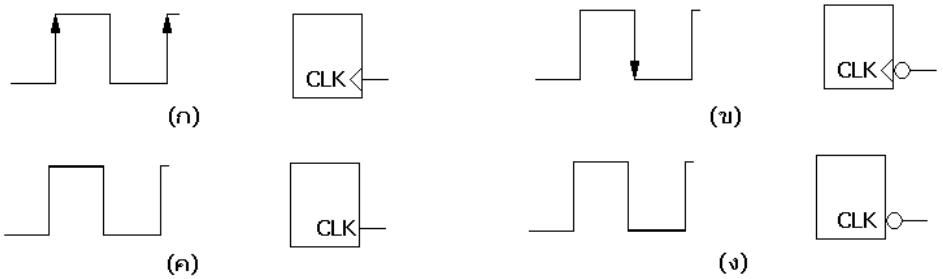
7.3 การออกแบบวงจรซีเควนเชียลด้วย VHDL

การทำงานของวงจรซีเควนเชียลนั้น สัญญาณเอาท์พุทจะขึ้นอยู่กับสัญญาณอินพุท และค่าสัญญาณเอาท์พุทเดิม วงจรซีเควนเชียลแบ่งได้เป็น 2 ประเภทคือ หนึ่งวงจร ซิงโครนัส (Synchronous) หรือ Clock mode สองวงจรอะซิงโครนัส (Asynchronous) สำหรับการออกแบบด้วย VHDL สามารถทำได้ทั้งสองแบบ แต่แบบ ซิงโครนัสจะเหมาะสมกว่า ดังนั้นตัวอย่างที่จะนำมากล่าวทั้งหมดต่อไปนี้จะเป็นวงจรแบบซิงโครนัส

ลักษณะสำคัญของวงจรซีเควนเชียลแบบซิงโครนัสคือต้องประกอบด้วยฟลิบฟลอบ จะมีจำนวนกี่ตัวก็แล้วแต่ แต่ทั้งหมดจะถูกกระตุ้นให้ทำงานพร้อมๆกัน ด้วยสัญญาณนาฬิกา ดังรูปที่ 7.8 และสัญญาณนาฬิกาที่กระตุ้นให้ฟลิบฟลอบทำงานก็มีด้วยกัน 4 แบบ คือ ขอบบวก ขอบลบ โลจิก 1 และ โลจิก 0



รูปที่ 7-8 ลักษณะการต่อสัญญาณนาฬิกาของวงจรซีเควนเชียลแบบซิงโครนัส



รูปที่ 7-9 ลักษณะของสัญญาณนาฬิกา (ก) ขอบบวก (ข) ขอบลบ (ค) โลจิก 1 และ (ง) โลจิก 0

การสร้างสัญญาณใน VHDL ใช้คำสั่ง IF.....THEN ดังนี้

D ฟลิบฟลอบทำงานด้วยสัญญาณนาฬิกาขอบบวก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
 entity D_Flipflop is
   Port (D : in std_logic;
         CLK : in std_logic;
           Q : Buffer std_logic);
end D_Flipflop;

architecture Behavioral of D_Flipflop is
begin
          PROCESS (clk)
```

```
        BEGIN
                IF(CLK'EVENT and CLK ='1') THEN
                        Q <= D;
                ELSE
                        Q <= Q;
                END IF;
        END PROCESS;
end Behavioral;
```

D ฟลิบฟลอบทำงานด้วยสัญญาณนาฬิกาขอบลบ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity D_Flipflop is
  Port (D : in std_logic;
                        CLK : in std_logic;
      Q : Buffer std_logic);
end D_Flipflop;

architecture Behavioral of D_Flipflop is
begin
        PROCESS (clk)
        BEGIN
                IF(CLK'EVENT and CLK = '0') THEN
                        Q <= D;
                ELSE
                        Q <= Q;
                END IF;
        END PROCESS;
end Behavioral;
```

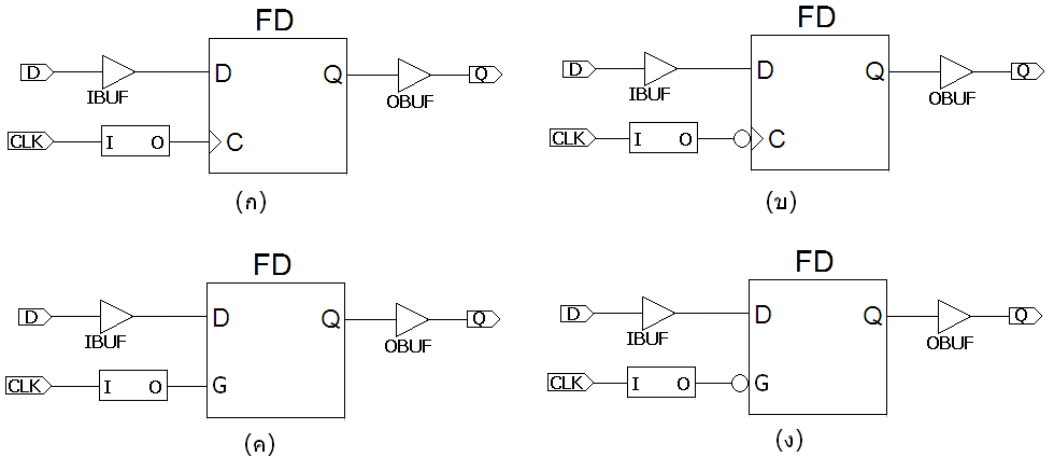
D ฟลิบฟลอบทำงานด้วยสัญญาณนาฬิกาโลจิก 1 (แสดงเฉพาะส่วน Process)

```
        PROCESS (clk, D)
        BEGIN
                IF(CLK = '1') THEN
                        Q <= D;
                ELSE
                        Q <= Q;
                END IF;
        END PROCESS;
```

D ฟลิบฟลอบทำงานด้วยสัญญาณนาฬิกาโลจิก 0 (แสดงเฉพาะส่วน Process)

```
        PROCESS (clk, D)
        BEGIN
                IF(CLK = '0') THEN
                        Q <= D;
                ELSE
                        Q <= Q;
                END IF;
        END PROCESS;
```

วงจรที่สังเคราะห์ได้ด้วย FPGA แสดงอยู่ในรูปที่ 7-10

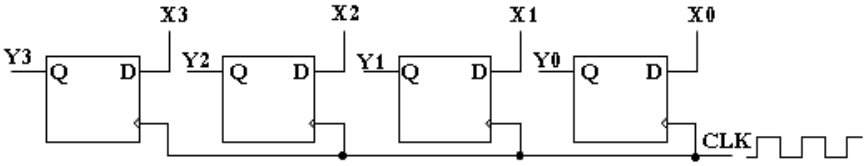


รูปที่ 7-10 ลักษณะวงจร D ฟลิบฟลอบ ที่สังเคราะห์ได้บน FPGA
(ก) แบบสัญญาณนาฬิกาขอบบวก (ข)แบบสัญญาณนาฬิกาขอบลบ
(ค) สัญญาณนาฬิกาแบบโลจิก 1 (ง) สัญญาณนาฬิกาแบบโลจิก 0

7.3.1 รีจิสเตอร์

รีจิสเตอร์เป็นหน่วยเก็บข้อมูลชั่วคราว ได้จากการรวมกลุ่มของส่วนความจำ ใช้เป็นหน่วยความจำชั่วคราวที่สามารถจะเก็บรักษาข้อมูลได้ และใช้เป็นตัวเลื่อนข้อมูล(shifting)ไปทางซ้ายหรือทางขวาได้

ตัวอย่างที่ 7.8 บัฟเฟอร์รีจิสเตอร์
สัญญาณ X3 – X0 เป็นสัญญาณอินพุท
CLK เป็นสัญญาณนาฬิกา
Y3 – Y0 เป็นสัญญาณ์เอาท์พุท



รูปที่ 7-11 วงจรบัฟเฟอร์รีจิสเตอร์

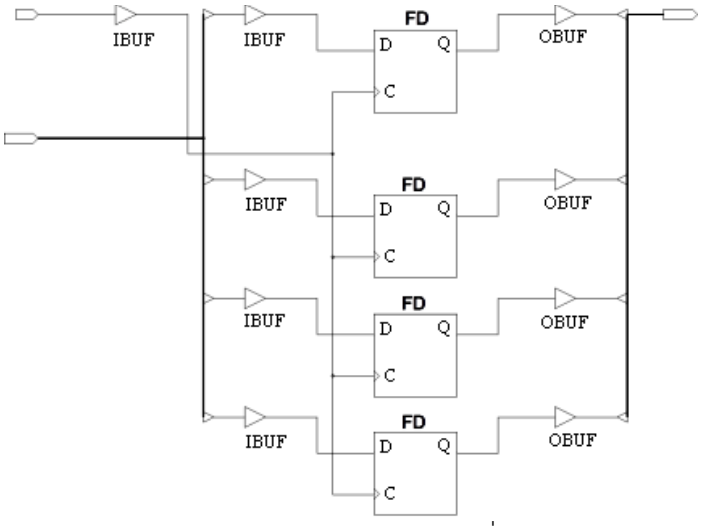
การทำงาน

X_3 X_2 X_1 X_0 เป็นอินพุทของวงจร เมื่อป้อนข้อมูลที่ต้องการเก็บเข้าที่อินพุทนี้ แล้วป้อนสัญญาณขอบบวกเข้าที่ CLK สัญญาณ Y_3 Y_2 Y_1 Y_0 จะเท่ากับ X_3 X_2 X_1 X_0 และจะมีค่าคงเดิมตลอดไปตราบเท่าที่ยังไม่มีสัญญาณขอบบวกมาที่ขา CLK อีก

โมเดล VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity buf_reg is
   Port ( clk : in  STD_LOGIC;
```

```
        x : in STD_LOGIC_vector(3 downto 0);
        y : out STD_LOGIC_vector(3 downto 0));
end buf_reg;

architecture Behavioral of buf_reg is
begin
    process (clk)
    begin
        if CLK='1' and CLK'event then
                y <= x;
        end if;
    end process;
end Behavioral;
```



รูปที่ 7-12 โลจิกไดอะแกรมวงจร บัฟเฟอร์รีจิสเตอร์ที่ได้จากการสังเคราะห์

บัฟเฟอร์รีจิสเตอร์ที่มีการควบคุม

เนื่องจากวงจรแบบแรกนั้น ใช้สัญญาณ CLK เป็นสัญญาณควบคุมการเก็บข้อมูล ดังนั้นเมื่อนำวงจรนี้ไปใช้ร่วมกับวงจรอื่นๆ อาจทำให้การทำงานของวงจรไม่สอดคล้องกั บวงจรเหล่านั้น วงจรแบบใหม่นี้กำหนดให้ CLK ต่ออยู่กับสัญญาณนาฬิกาซึ่งวิ่งอยู่ตลอดเวลา และให้มีสัญญาณควบคุมการเก็บขึ้นต่างหากอีกหนึ่งสัญญาณ ดังนั้นเมื่อนำวงจรนี้ไปใช้งานร่วมกับวงจรอื่นการทำงานก็จะสอดคล้องกับวงจรอื่นๆ เพราะใช้สัญญาณนาฬิกาควบคุมการทำงานร่วมกัน

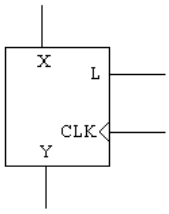
ตัวอย่างที่ 7.9 บัฟเฟอร์รีจิสเตอร์ที่มีการควบคุม

สัญญาณ X3 – X0 เป็นสัญญาณอินพุท

CLK เป็นสัญญาณนาฬิกา

Y3 – Y0 เป็นสัญญาณ์เอาท์พุท

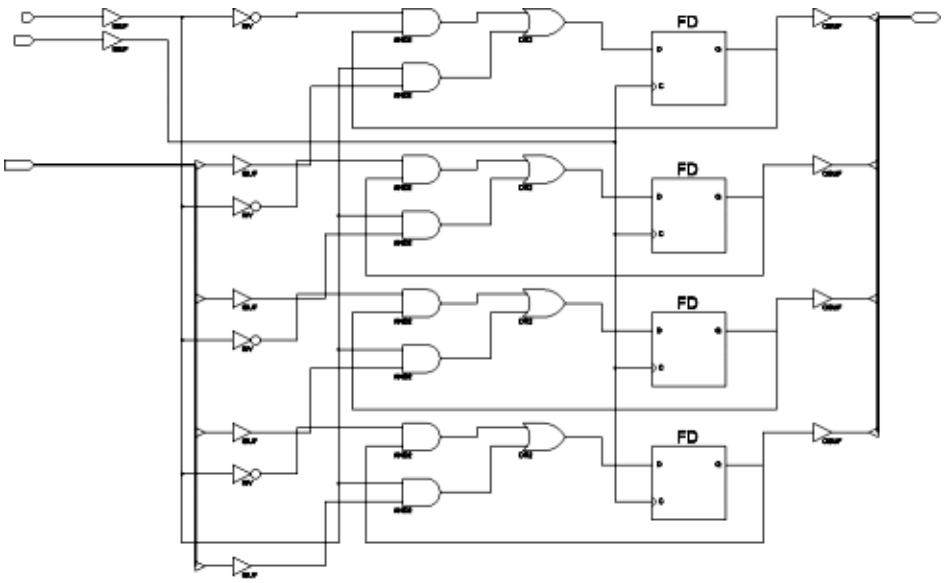
L เป็นสัญญาณควบคุมการเก็บ กำหนดให้ ถ้า L = 1 ให้ Y = X เป็นการโหลดข้อมูลเก็บ แต่ถ้า L = 0 ให้ Y คงเดิมไม่เปลี่ยนแปลง



รูปที่ 7-13 แผนผังบล็อกของบัฟเฟอร์รีจิสเตอร์ที่มีสัญญาณควบคุม

โมเดล VHDL

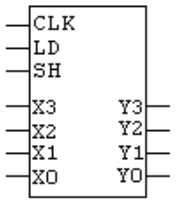
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity cont_reg is
   Port ( clk : in  STD_LOGIC;
                l : in  STD_LOGIC;
               x : in STD_LOGIC_vector(3 downto 0);
               y : inout STD_LOGIC_vector(3 downto 0));
end cont_reg;
architecture Behavioral of cont_reg is
begin
          process (clk)
          begin
                    if CLK='1' and CLK'event then
                              if l='1' then
                                        y <= x;
                              else
                                        y <= y;
                              end if;
                    end if;
          end process;
end Behavioral;
```



รูปที่ 7-14 โลจิกไดอะแกรมวงจร บัฟเฟอร์รีจิสเตอร์ที่ได้จากการสังเคราะห์

ชิฟท์รีจิสเตอร์ (Shift Register)

ชิฟท์รีจิสเตอร์ เป็นรีจิสเตอร์ที่ทำหน้าที่เลื่อนบิตข้อมูลไปทางซ้ายหรือมาทางขวา รูปที่ 7-10 เป็นแผนผังบล็อกของชิฟท์รีจิสเตอร์ที่มีการควบคุมการเลื่อนข้อมูลและโหลด ทุกครั้งที่มีสัญญาณนาฬิกา และสัญญาณ LD ข้อมูลเป็น '1' Y3 - Y0 จะเท่ากับ X3 – X0 แต่ถ้า LD เป็น '0' และ SH เป็น '1' Y3 – Y2 เท่ากับ Y2 – Y0 และ Y0 เท่ากับ X0



รูปที่ 7-15 แผนผังบล็อกของชิฟท์รีจิสเตอร์ที่มีการควบคุมการเลื่อนและการโหลดข้อมูล

ตัวอย่างที่ 7.10 ชิฟท์รีจิสเตอร์ที่มีการควบคุมการเลื่อนและการโหลดข้อมูล

โมเดล VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shift_reg is
   Port ( clk : in  STD_LOGIC;
          ld : in  STD_LOGIC;
          sh : in  STD_LOGIC;
           x : in STD_LOGIC_vector(3 downto 0);
           y : inout STD_LOGIC_vector(3 downto 0));
end shift_reg;

architecture Behavioral of shift_reg is

begin
        process (clk)
        begin
                if CLK='1' and CLK'event then
                        if ld='1' then
                                y <= x;
                        else
                                if sh='1' then
                                        y(3 downto 1)<=y(2 downto 0);
                                        y(0) <= x(0);
                                else
                                        y <= y;
                                end if;
                        end if;
                end if;
        end process;
end Behavioral;
```

7.3.2 วงจรนับ

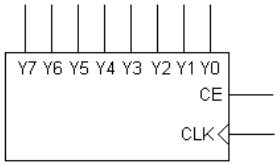
การออกแบบวงจรนับด้วย VHDL วิธีที่สะดวกที่สุดคือการเขียนจากพฤติกรรมการทำงานของวงจรโดยตรง ดังตัวอย่างวงจรนับขึ้นต่อไปนี้

ตัวอย่างที่ 7.11 วงจรนับเลขไบนารี (Binary counter) ขนาด 8 บิต

สัญญาณ Y7 ถึง Y0 เป็นเอาท์พุท

CE สัญญาณควบคุมการนับ

CLK เป็นสัญญาณนาฬิกา



รูปที่ 7-16 บล็อกไดอะแกรมวงจรนับเลขไบนารี

การทำงาน ทุกครั้งที่สัญญาณนาฬิกาเปลี่ยนจาก ‘0’ เป็น ‘1’ และสัญญาณ CE เท่ากับ 1 จะนับขึ้น แต่ถ้า CE เท่ากับ 0 จะหยุดนับ เอาท์พุท Y ค้างค่าเดิม

โมเดล VHDL แบบที่ 1 วงจรนับขึ้นที่มีสัญญาณควบคุมการนับ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity coup is
    port(CLK: in STD_LOGIC;
                CE: in STD_LOGIC;
              y: inout INTEGER range 255 downto 0);
end coup;

architecture Behavioral of coup is
begin

process (CLK)
begin
  if CLK='1' and CLK'event then
     if CE = '1' then
       y <= y + 1;
     else
       y <= y;
     end if;
  end if;
end process;

end Behavioral;
```

แบบที่ 2 ตัวอย่างวงจรนับ coup ตามแบบที่ 1 นั้น บางครั้งไม่สามารถแปลงเป็นวงจรจริงๆได้ ใช้ได้เพียงการจำลองการทำงานเท่านั้น ดังนั้นถ้าต้องการให้สามารถแปลงเป็นวงจรจริงได้ จะต้องใช้ ฟังก์ชั่นในการแปลงเหมือนตัวอย่างที่ 7.4 แต่สำหรับตัวอย่างนี้ได้แสดงการสร้างโมดูล VHDLสำหรับการแปลงประเภทของสัญญาณจาก Integer เป็น Standard_logic_vector ชื่อ int2bit8 ส่วนโมดูลหลัก coup2 จะเรียกใช้

คอมโปเนนต์ coup เพื่อทำหน้าที่เป็นวงจรนับ และคอมโปเนนต์ int2bit8 เป็นส่วนแปลงสัญญาณเพื่อส่งออกไปยังสัญญาณเอาท์พุท

โมดูล int2bit8

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity int2bit8 is
   port (x : in integer range 255 downto 0;
         z : out std_logic_vector(7 downto 0));
end int2bit8;
architecture beh of int2bit8 is
begin
  process(x)
  variable i : integer range 0 to 7;
  begin
    for i in 0 to 7 loop
      if ((x/(2**i) ) mod 2 = 1) then
         z(i) <= '1';
      else
         z(i) <= '0';
                end if;
    end loop;
  end process;
end beh;
```

โมดูลหลัก

```
library IEEE;
use IEEE.std_logic_1164.all;
entity coup2 is
   port (pclk, pce : in std_logic;
              py : out  std_logic_vector (7 downto 0));
end coup2;

architecture coup2_arch of coup2 is

component int2bit8
   port (x : in integer range 255 downto 0;
       z : out std_logic_vector(7 downto 0));
end component;

component coup     port(CLK, CE: in STD_LOGIC;
        y: inout integer range 255 downto 0);
end component;
signal bus1 : INTEGER range 255 downto 0;
begin
  c1: coup port map(pclk, pce, bus1);
  c2: int2bit8 port map(bus1, py);
end coup2_arch;
```

ข้อมูลเป็น std_logic_vector

ข้อมูลเป็น integer

ส่วนนับ

ส่วนแปลงประเภทสัญญาณ

แบบที่ 3 สำหรับตัวอย่างนี้ใช้การแปลงประเภทสัญญาณเช่นเดียวกับแบบที่ 2 แต่ฟังก์ชั่นการแปลงได้สร้างอยู่ใน Package

Package ที่สร้างขึ้นใหม่ ภายในมีเพียง 1 ฟังก์ชั่น

```
-- i2bv : Integer to Bit_vector.
-- In :  Integer, Value and width.
-- Return : std_logic_vector, with left bit is the most significant bit.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
package my_pack is
    function i2bv (val : integer) return std_logic_vector;
end my_pack;
package body my_pack is
    function i2bv (val : integer) return std_logic_vector is
    variable result : std_logic_vector(7 downto 0) := (others => '0');
    begin
        for i  in 0 to 7 loop
                                if ( (val/(2**i) ) mod 2 = 1) then
                                result(i) := '1';
                                end if;
        end loop;
        return (result);
    end i2bv;
end my_pack;
```

ส่งคืนค่าผลลัพธ์

โมดูลหลัก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.my_pack.all;
entity coup3 is
  port(dout : inout std_logic_vector(7 downto 0);
               clk, ce  : in STD_LOGIC);
end coup3;
architecture coup3_beh of coup3 is
begin
  counter: process(clk, ce)
  variable cnum : integer range 255 downto 0;
           begin
           if clk='1' and clk'event then
           if ce = '1' then
           cnum:= cnum+1;
           end if;
           end if;
           dout <= i2bv(cnum);
           end process;
end coup3_beh;
```

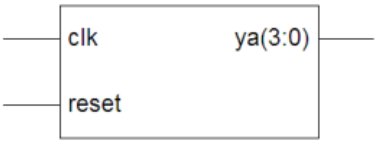
เรียกใช้ฟังก์ชั่น

ตัวอย่างที่ 7.12 วงจรนับเลขฐานสิบขนาด 1 หลัก

สัญญาณ clk เป็นสัญญาณอินพุท

reset เป็นสัญญาณควบคุม

ya เป็นสัญญาณเอาท์พุท



รูปที่ 7-17 บล็อกไดอะแกรมวงจรนับเลขฐานสิบ

การทำงาน ถ้าสัญญาณ reset เป็น ‘0’ สัญญาณเอาท์พุทเป็น “0000” แต่ถ้า reset เป็น ‘1’ เมื่อมีสัญญาณนาฬิกามากระตุ้น(สัญญาณ clk เปลี่ยนจาก ‘0’ เป็น ‘1’) วงจรนับจะนับขึ้นที่ละ 1 แบบเลขฐานสิบ

สำหรับเทคนิคที่ใช้ในตัวอย่างนี้ ต้องการแสดงให้เห็นถึงการเขียนวงจรนับเพื่อนับเลขฐานสิบ และวิธีการแปลงสัญญาณอีกรูปแบบหนึ่งซึ่งต่างจากตัวอย่างที่ 7.11 ซึ่งเทคนิคแบบนี้เหมาะกับการนำไปใช้แปลงรหัสแบบต่างๆได้ดี เขียนเป็นคำสั่ง VHDL ได้ง่าย

แบบที่ 1 ใช้การแปลงสัญญาณเช่นเดียวกับตัวอย่าง 7.7

โมเดลหลัก

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bcd1 is
   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          y : out  STD_LOGIC_VECTOR (3 downto 0));

end bcd1;

architecture Behavioral of bcd1 is
signal ya : INTEGER range 0 to 15;

component int2std_logic is
  port (bin : in integer range 0 to 15 ;
                y : out std_logic_vector(3 downto 0));
end component;
begin
            process (clk, reset)
            begin
                        if Reset='0' then
                                    ya <= 0;
                        else
                                    if CLK='1' and CLK'event then
                                                if ya >= 9 then
                                                            ya <= 0;
                                                else
                                                            ya <= ya + 1;
                                                end if;
                                    else
                                                ya <= ya;
                                    end if;
                        end if;
            end process;
            c1: int2std_logic port map(ya, y);
end Behavioral;
```

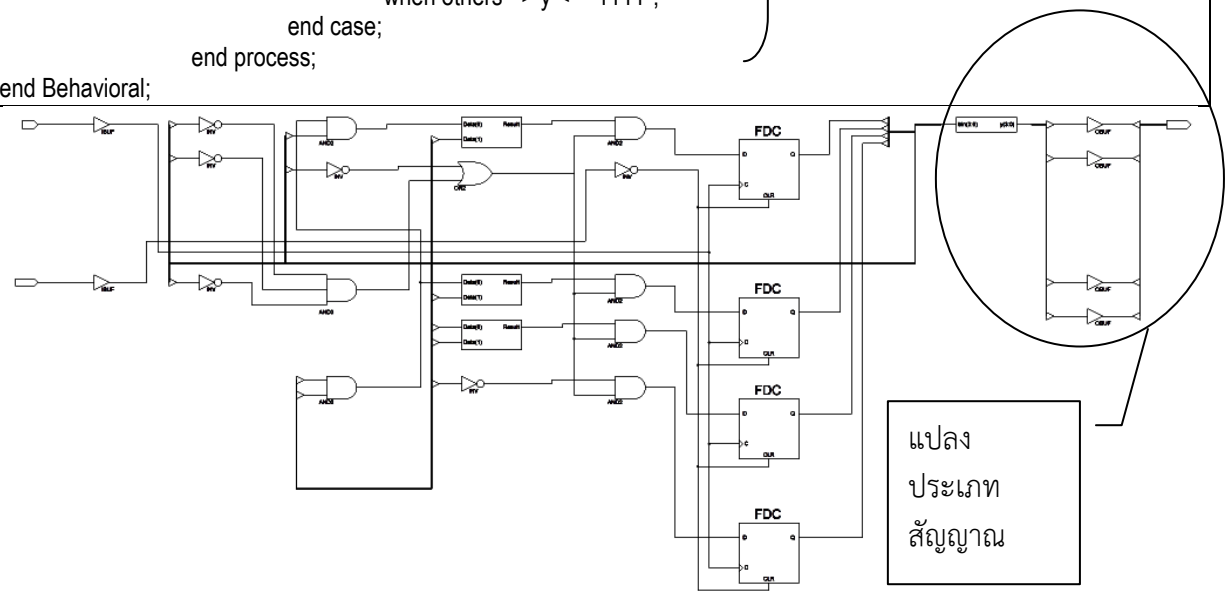
สถานะรีเซ็ต

นับเลขฐานสิบ

โมดูลแปลงสัญญาณ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity int2std_logic is
  port (bin : in integer range 0 to 15 ;
              y : out std_logic_vector(3 downto 0));
end int2std_logic;
architecture Behavioral of int2std_logic is
        begin
                int2bv:process(bin)
                begin
                        case bin is
                                when 0 => y <= "0000";
                                when 1 => y <= "0001";
                                when 2 => y <= "0010";
                                when 3 => y <= "0011";
                                when 4 => y <= "0100";
                                when 5 => y <= "0101";
                                when 6 => y <= "0110";
                                when 7 => y <= "0111";
                                when 8 => y <= "1000";
                                when 9 => y <= "1001";
                                when 10 => y <= "1010";
                                when 11 => y <= "1011";
                                when 12 => y <= "1100";
                                when 13 => y <= "1101";
                                when 14 => y <= "1110";
                                when others => y <= "1111";
                        end case;
                end process;
end Behavioral;
```

ตารางแปลงค่า

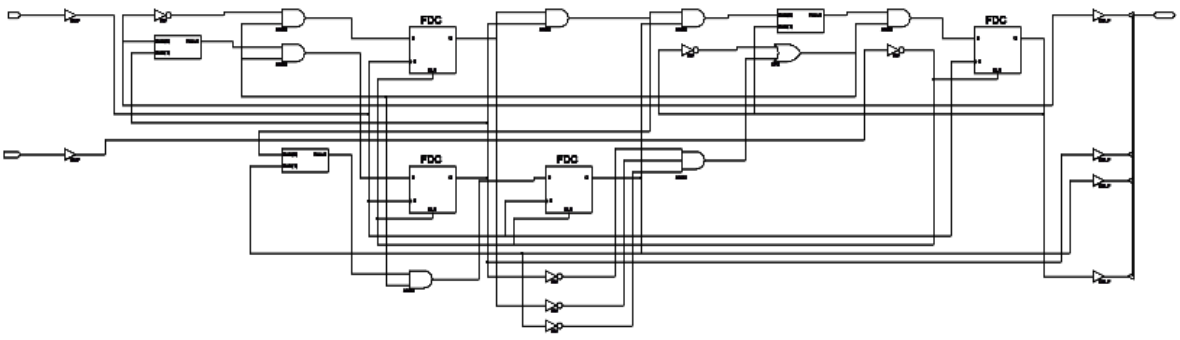


รูปที่ 7-18 โลจิกไดอะแกรมวงจรนับเลขฐานสิบที่ได้จากการสังเคราะห์

แบบที่ 2 ไม่ใช้การแปลงสัญญาณ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity bcd1 is
   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
            ya : inout  integer range 0 to 15 );
end bcd1;

architecture Behavioral of bcd1 is

begin
          process (clk, reset)
          begin
                    if Reset='0' then
                              ya <= 0;
                    else
                              if CLK='1' and CLK'event then
                                        if ya >= 9 then
                                                  ya <= 0;
                                        else
                                                  ya <= ya + 1;
                                        end if;
                              else
                                        ya <= ya;
                              end if;
                    end if;
          end process;
end Behavioral;
```

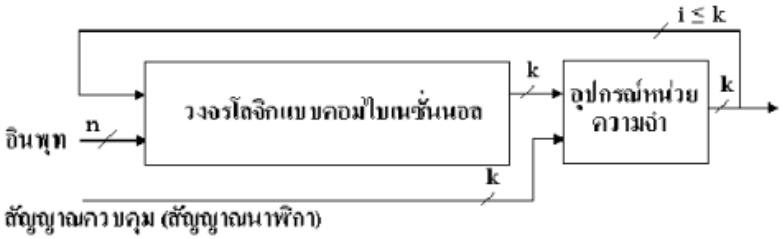


รูปที่ 7-19 โลจิกไดอะแกรมวงจรนับเลขฐานสิบที่ได้จากการสังเคราะห์

7.3.3 สเตทแมชชีน (State Machine)

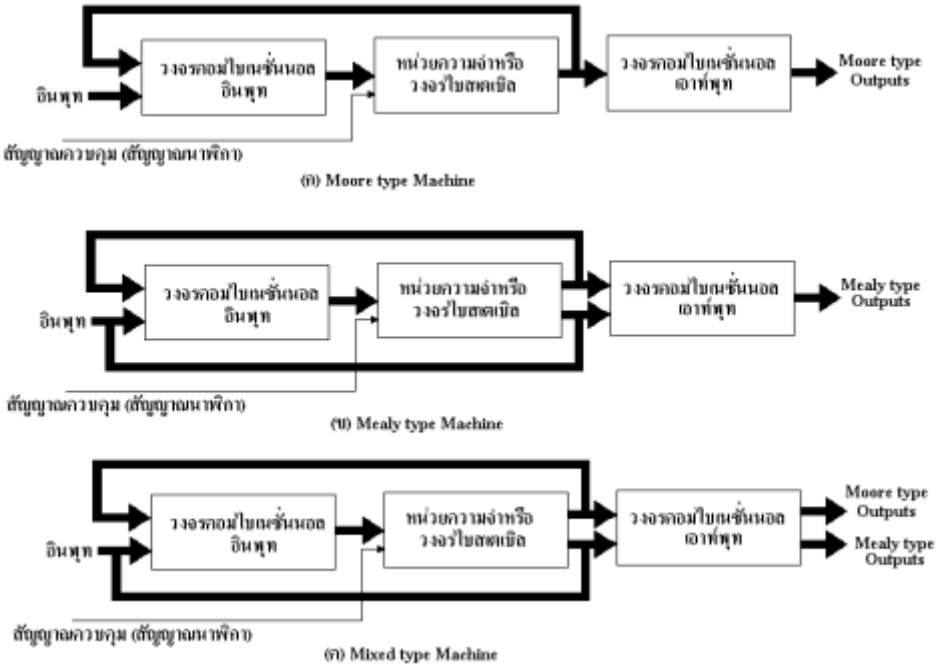
วงจรโลจิกแบบซีเควนเชียลมีอีกชื่อหนึ่งว่า “สเตทแมชชีน (State machine)” และเนื่องจากการใช้อุปกรณ์มีจำนวนที่แน่นอนดังนั้นบางทีก็เรียกว่า “Finite State Machine (FSM)” สำหรับในที่นี้จะกล่าวเฉพาะการออกแบบวงจรซีเควนเชียลแบบซิงโครนัส (Synchronous Sequential Logic Circuit) เท่านั้น เพื่อให้สอดคล้องกับการสร้างลงบนอุปกรณ์ CPLD หรือ FPGA ซึ่งส่วนใหญ่ใช้สัญญาณนาฬิกาจากแหล่งเดียวกัน

วงจรซีเควนเชียลประกอบด้วยส่วนสำคัญ 2 ส่วน คือวงจรโลจิกแบบคอมไบเนชั่นนอล และอุปกรณ์หน่วยความจำหรือฟลิปฟลอป ซึ่งใช้ทำหน้าที่กำหนดสเตทหรือสถานะของวงจร สำหรับวงจรซีเควนเชียลแบบซิงโครนัสนั้นฟลิปฟลอปทุกตัวในวงจร จะถูกกระตุ้นให้ทำงานพร้อมๆกัน ด้วยวิธีการต่อสัญญาณนาฬิกาของฟลิปฟลอปทุกตัวให้รับสัญญาณนาฬิกาจากแหล่งเดียวกัน ตามรูปที่ 7-20



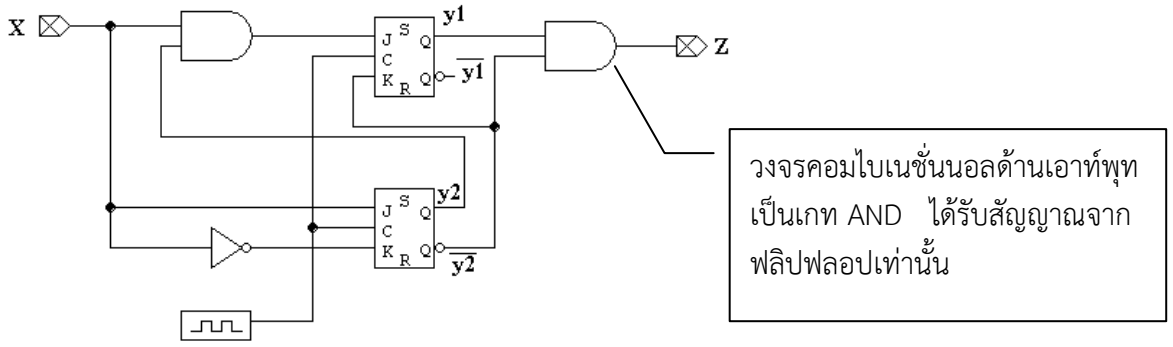
รูปที่ 7-20 แผนผังบล็อกของวงจรซิงโครนัส

วงจรซีเควนเชียลแบบซิงโครนัสจัดแบ่งตามลักษณะของเอาท์พุทได้ 3 รูปแบบ แบบแรกคิดค้นโดย E.F.Moore ชื่อว่าแบบ “ มัวร์ (Moore)” ลักษณะของภาคเอาท์พุทจะขึ้นอยู่กับสถานะปัจจุบันของวงจรและไม่ขึ้นกับสัญญาณอินพุทจากภายนอก ลักษณะบล็อกไดอแกรมจะเป็นไปตามรูปที่ 7-21(ก) แบบที่สองคิดค้นโดย G.H.Mealy ได้ชื่อว่าแบบ “เมียลี(Mealy)” ลักษณะของภาคเอาท์พุทนอกจากจะขึ้นอยู่กับสถานะปัจจุบันของวงจรแล้ว ยังขึ้นอยู่กับสัญญาณอินพุทจากภายนอกด้วยตามรูปที่ 7-21(ข) ส่วนแบบสุดท้ายเป็นการผสมทั้งสองแบบมัวร์และแบบเมียลีเข้าด้วยกัน ตามรูปที่ 7-21(ค) และตัวอย่างวงจรทั้งสองแบบอยู่ในรูปที่ 7-22

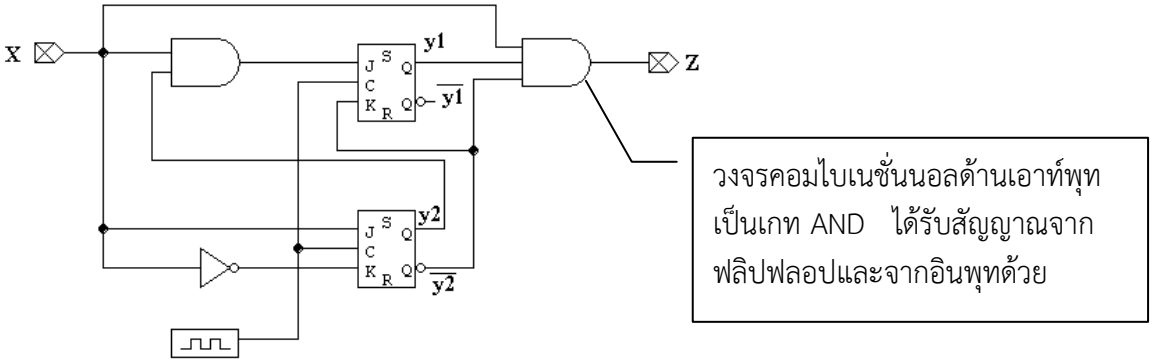


C

รูปที่ 7-21 แผนผังบล็อกของวงจรซิงโครนัสทั้ง 3 แบบ



(ก)



(ข)

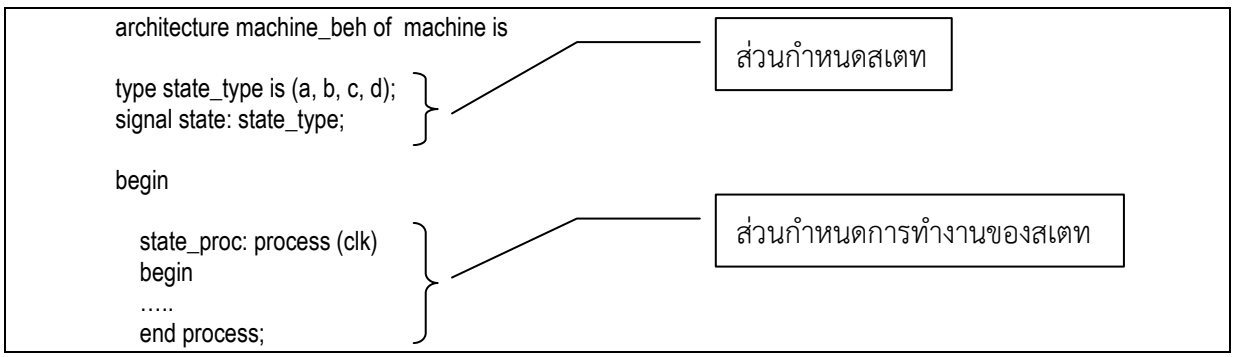
รูปที่ 7-22 ตัวอย่างวงจรซิงโครนัส (ก) แบบมัวร์ (ข)แบบเมียลี

การออกแบบวงจรซีเควนเชียลแบบซิงโครนัส หรือสเตทแมชชีน ด้วย VHDL เริ่มจากการกำหนดขอบเขตของงานออกมาในรูปของ สเตทไดอะแกรม (State diagram) แล้วนำสเตทไดอะแกรมนี้มาเขียนด้วยภาษา VHDL เมื่อได้เป็นรหัสคำสั่งของ VHDL แล้วการสังเคราะห์เป็นวงจรจริงก็ให้เป็นหน้าที่ของเครื่องมือหรือซอฟต์แวร์ทูล (Software tools) ต่อไป

โครงสร้างของคำสั่ง VHDL สำหรับสเตทไดอะแกรม ประกอบด้วยส่วนสำคัญ 3 ส่วน ได้แก่

- ส่วนกำหนดสเตท ส่วนนี้เขียนด้วยคำสั่ง TYPE เพื่อเป็นการสร้างชนิดข้อมูลใหม่ขึ้นมา
- ส่วนกำหนดการทำงานของสเตท ส่วนนี้เขียนอยู่ใน PROCESS
- ส่วนกำหนดการทำงานของเอาท์พุท ส่วนนี้เขียนอยู่ใน PROCESS เช่นเดียวกัน

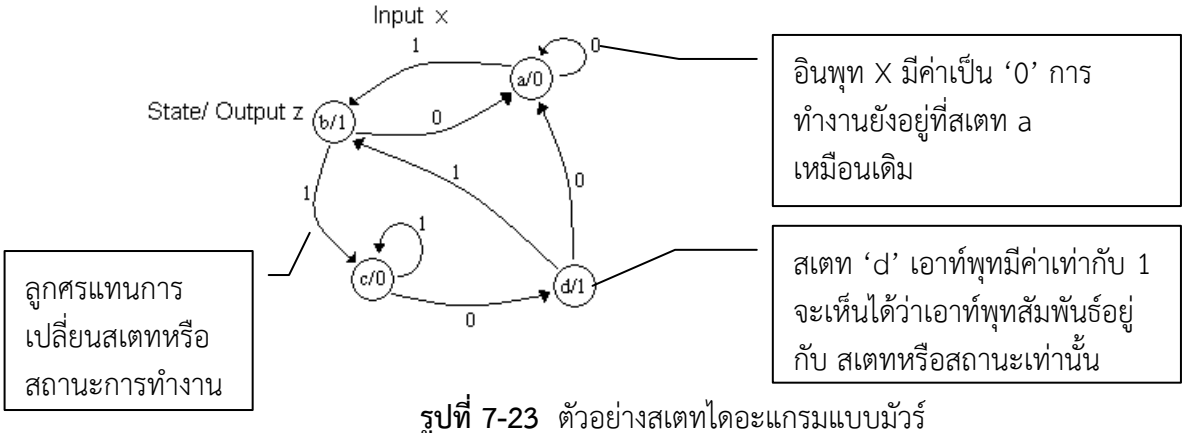
ซึ่งเขียนเป็นโครงร่างคร่าวๆได้ดังนี้



```
        output_proce: process (state)
        begin
        ……
        end process;

    end machine_beh;
```

ส่วนกำหนดการทำงานของ

ตัวอย่างที่ 7.13 สเตทแมชชีนแบบมัวร์ (Moore) การทำงานเป็นไปตามสเตทไดอะแกรม ในรูปที่ 7-19



อินพุท X มีค่าเป็น ‘0’ การทำงานยังอยู่ที่สเตท a เหมือนเดิม

สเตท ‘d’ เอาท์พุทมีค่าเท่ากับ 1 จะเห็นได้ว่าเอาท์พุทสัมพันธ์อยู่กับ สเตทหรือสถานะเท่านั้น

ลูกศรแทนการเปลี่ยนสเตทหรือสถานะการทำงาน

รูปที่ 7-23 ตัวอย่างสเตทไดอะแกรมแบบมัวร์

โมเดล VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.ALL;
entity moore is
   port (x, clk : in std_logic;
         z : out std_logic);
end moore;
architecture moore_beh of moore is
type state_type is (a, b, c, d);
signal state: state_type;
begin
 process (clk)
 begin
  if clk = '1' and clk'event then
   case state is
        when a =>  if x = '1' then
                        state <= b;
                   else
                        state <= a;
                   end if;
        when b =>  if x = '1' then
                        state <= c;
                   else
                        state <= a;
                   end if;
        when c =>  if x = '1' then
                        state <=c;
                   else
                        state <= d;
                   end if;
        when d =>  if x = '1' then
                        state <= b;
```

ส่วนกำหนดสเตท

เมื่อมีสัญญาณนาฬิกามากระตุ้น

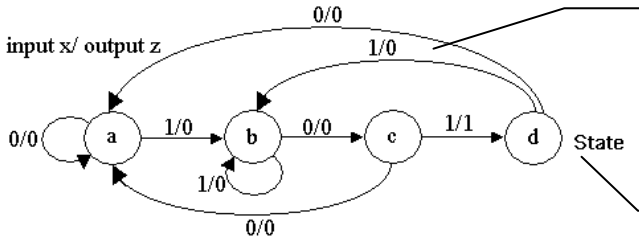
Process สำหรับ กำหนดการทำงานของสเตท

ความหมาย- ขณะทำงานที่สเตท a ถ้าอินพุท X เป็น ‘1’ การทำงานจะไปเป็นสเตท b แต่ถ้าอินพุท X เป็น ‘0’ การทำงานยังคงอยู่สเตทเดิมคือ a

```
                else
                    state <= a;
                end if;
            end case;
    end if;
end process;
process (state)
  begin
            case state is
                    when a => z <= '0';
                    when b => z <= '1';
                    when c => z <= '0';
                    when d => z <= '1';
            end case;
  end process;
end moore_beh;
```

Process สำหรับ กำหนดการทำงานของเอาท์พุท (ไม่มีค่าอินพุทและสัญญาณนาฬิกามากำกับโดยตรง)

ตัวอย่างที่ 7.14 สเตทแมชชีนแบบเมียลี (Mealy) การทำงานเป็นไปตามสเตทไดอะแกรม ในรูปที่ 7-24



ขณะทำงานสเตท d อินพุท X เป็น ‘1’ ทำให้ได้เอาท์พุท Z เป็น ‘0’ และการทำงานเปลี่ยนเป็นสเตท b

สเตท ‘d’ (ไม่มีค่าเอาท์พุท)

รูปที่ 7-24 ตัวอย่างสเตทไดอะแกรมแบบเมียลี

โมเดล VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.ALL;
entity mealy is
   port (x, clk : in std_logic;
         z : out std_logic);
end mealy;
architecture mealy_beh of mealy is
type state_type is (a, b, c, d);
signal state: state_type;
begin
 process (clk)
 begin
  if clk = '1' and clk'event then
    case state is
          when a => if x = '1' then        state <= b;
                    else       state <= a;            end if;
          when b => if x = '1' then        state <= b;
                    else       state <= c;            end if;
          when c => if x = '1' then        state <=d;
                    else state <= a; end if;
          when d => if x = '1' then        state <= b;
                    else state <= a;       end if;
          end case;
   end if;
end process;
```

ส่วนกำหนดสเตท

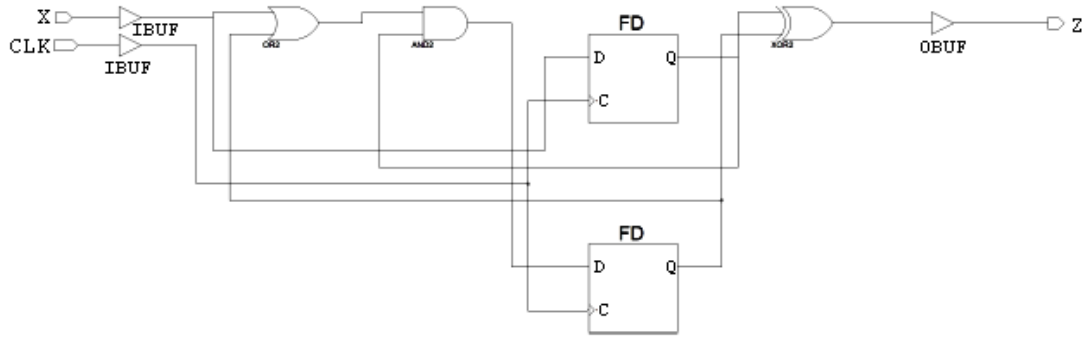
เมื่อมีสัญญาณนาฬิกามากระตุ้น

Process สำหรับกำหนดการทำงานของสเตท

```
process (state)
  begin
        case state is
              when a =>     if x = '1' then z <= '0';
                            else      z <= '0';   end if;
              when b =>     if x = '1' then z <= '0';
                            else      z <= '0';   end if;
              when c =>     if x = '1' then z <= '1';
                            else      z <= '0';   end if;
              when d =>     if x = '1' then z <= '0';
                            else      z <= '0';   end if;
        end case;
  end process;
end mealy_beh;
```

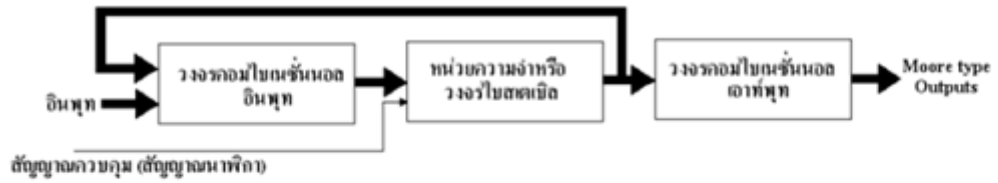
Process สำหรับกำหนดการทำงานของเอาท์พุท ซึ่งขึ้นอยู่กับค่าสเตทและสัญญาณอินพุท(แต่ไม่มีสัญญาณนาฬิกามากำกับโดยตรง)

จากตัวอย่างที่ 7.13 และ 7.14 เป็นการเขียนรหัสคำสั่งโดยแยกส่วนการทำงานของสเตทและส่วนเอาท์พุทออกเป็น 2 โปรเซส โดยส่วนของเอาท์พุทไม่มีสัญญาณนาฬิกา CLK มากำกับการทำงาน ทำให้วงจรในส่วนเอาท์พุทเป็นเพียงวงจรคอมไบเนชั่นนอลเท่านั้น ดังรูปที่ 7.25 ซึ่งเป็นโลจิกไดอะแกรมที่ได้จากการสังเคราะห์ตัวอย่างที่ 7.13

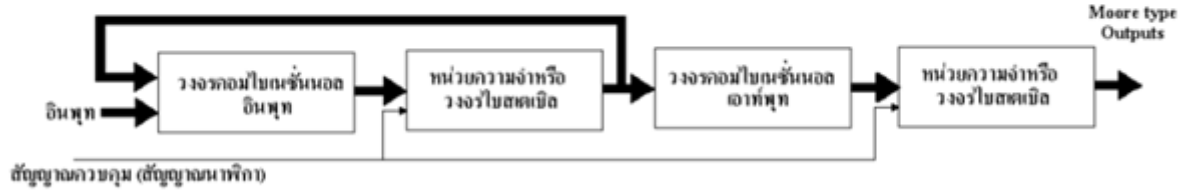


รูปที่ 7-25 โลจิกไดอะแกรมที่สังเคราะห์ได้จากตัวอย่างที่ 7.13

แต่ถ้าต้องการให้สัญญาณเอาท์พุททำงานสอดคล้องกับสัญญาณนาฬิกา CLK ก็สามารถทำได้โดยการรวมโปรเซสทั้งสองเข้าด้วยกัน เขียนเป็นแผนผังบล็อกได้ตามรูปที่ 7.26



(ก) วงจรมัวร์แบบไม่มีสัญญาณนาฬิกาที่เอาท์พุท



(ข) วงจรมัวร์แบบมีสัญญาณนาฬิกาที่เอาท์พุท

รูปที่ 7-26 เปรียบเทียบแผนผังบล็อกของแบบมัวร์ทั้ง 2 แบบ

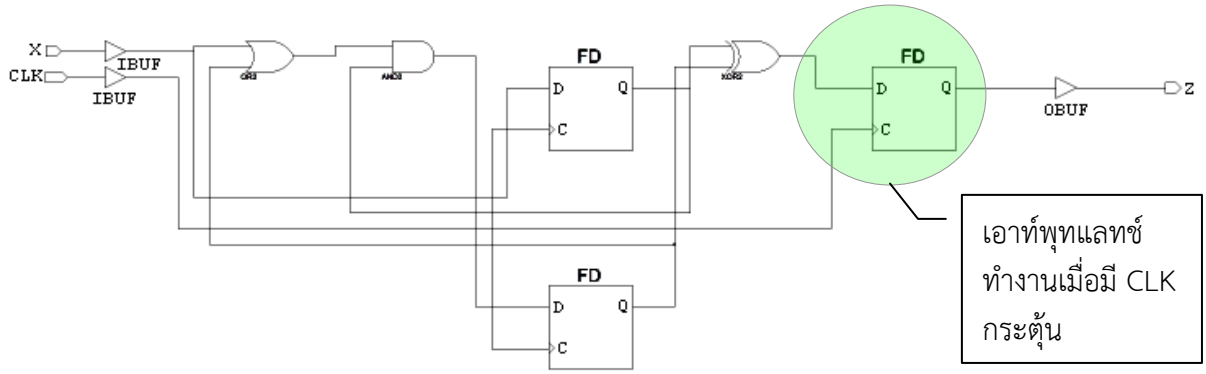
ตัวอย่างที่ 7.15 จากตัวอย่างที่ 7.13 เมื่อเขียนแบบสัญญาณเอาท์พุททำงานสอดคล้องกับสัญญาณนาฬิกา โมเดล VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.ALL;
entity moore_clk is
   port (x, clk : in std_logic;
         z : out std_logic);
end moore_clk;
architecture moore_beh of moore_clk is

type state_type is (a, b, c, d);
signal state: state_type;

begin

      process (clk)
      begin
            if clk = '1' and clk'event then
                  case state is
                        when a => if x = '1' then
                                          state <= b;
                                    else
                                          state <= a;
                                    end if;
                                    z <= '0';
                        when b =>  if x = '1' then
                                          state <= c;
                                    else
                                    state <= a;
                                    end if;
                                    z <= '1';
                        when c =>      if x = '1' then
                                          state <=c;
                                    else
                                          state <= d;
                                    end if;
                                    z <= '0';
                        when d =>      if x = '1' then
                                          state <= b;
                                    else
                                          state <= a;
                                    end if;
                                    z <= '1';
                  end case;
            end if;
      end process;

end moore_beh;
```

สัญญาณนาฬิกา

กำหนดการทำงานของสเตท

สัญญาณเอาท์พุท



รูปที่ 7-27 โลจิกไดอะแกรมที่สังเคราะห์ได้จากตัวอย่างที่ 7.15

การเข้ารหัสสเตท (State Encoding)

การเข้ารหัสสเตทหมายถึงการกำหนดค่าให้กับสเตทที่สร้างขึ้นใช้งาน จากตัวอย่างที่ 7.13 ถึง 7.15 การกำหนดสเตทเขียนได้ดังนี้

```
type state_type is (a, b, c, d);
signal state: state_type;
```

การกำหนดลักษณะนี้ไม่ได้กำหนดรหัสให้กับสเตท แต่ให้เครื่องมือหรือซอฟท์แวร์ ทูลเป็นผู้กำหนดให้เอง ซอฟท์แวร์ทูลทั่วๆไปจะมีรหัสให้เลือกหลายแบบเช่น Binary Gray Johnson และ One-Hot หรือจะให้ซอฟท์แวร์ทูลเลือกเองโดยอัตโนมัติ หรือกำหนดให้เลือกเพื่อเน้นเรื่องความเร็วการทำงานของวงจร หรือเน้นให้ใช้ทรัพยากรของชิพที่ประหยัด หรือสุดท้ายให้ผู้ใช้กำหนดขึ้นเองก็ได้ โดยเขียนดังนี้

```
type state_type is (a, b, c, d);
attribute enum_encoding : string;
attribute enum_encoding of state_type : type is "0001 0011 0111 1111";
signal state: state_type;
```

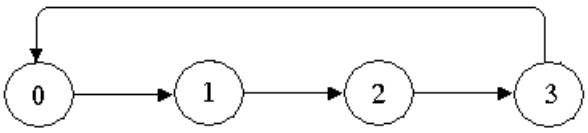
รหัสที่กำหนดให้

ตารางที่7-4 ตัวอย่างรหัสแบบต่างๆ

State	Binary	Gray	Johnson	One-hot
S0	000	000	0000	00000001
S1	001	001	0001	00000010
s2	010	011	0011	00000100
S3	011	010	0111	00001000
S4	100	110	1111	00010000
S5	101	111	1110	00100000
S6	110	101	1100	01000000
S7	111	100	1000	10000000

ตัวอย่างที่ 7.16 ตัวอย่างนี้เป็นการออกแบบวงจรนับเลขไบนารีขนาด 2 บิต แบบไม่มีสัญญาณควบคุม เขียนแบบสเตทแมชชีน

สัญญาณ CLK เป็นสัญญาณนาฬิกา ใช้สำหรับการนับ
Y1 และ Y0 เป็นสัญญาณเอาท์พุท
การทำงาน ตามสเตทไดอะแกรมในรูปที่ 7-24



รูปที่ 7-28 สเตทไดอะแกรมวงจรนับแบบไบนารีขนาด 2 บิต

โมเดล VHDL เขียนแบบสเตท

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.ALL;
entity binco is
   port (clk : in std_logic;
         y : out std_logic_vector(1 downto 0));
end binco;
architecture binco_beh of binco is
type state_type is (a, b, c, d);
signal state: state_type;
begin
  process (clk)
  begin
    if clk = '1' and clk'event then
      case state is
            when a =>  state <= b;
            when b =>  state <= c;
            when c =>  state <= d;
            when d =>  state <= a;
      end case;
    end if;
end process;
process (state)
   begin
     case state is
             when a => y <= "00";
             when b => y <= "01";
             when c => y <= "10";
             when d => y <= "11";
     end case;
   end process;
end binco_beh;
```

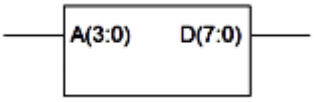
7.4 หน่วยความจำ

หน่วยความจำเป็นอุปกรณ์สำคัญในระบบคอมพิวเตอร์ ใช้เก็บข้อมูลและคำสั่งหรือโปรแกรมของคอมพิวเตอร์ หน่วยความจำแบ่งเป็น 2 ชนิด เรียกว่า รอม (ROM หรือ Read Only Memory) และแรม (RAM หรือ Random Access Memory)

ตัวอย่างที่ 7.17 ออกแบบหน่วยความจำรอม แบบ 8 บิต ขนาด 16 ตำแหน่ง

สัญญาณ A3 – A0 เป็นสัญญาณแอดเดรส ใช้ระบุตำแหน่งข้อมูล

D7 – D0 เป็นสัญญาณข้อมูล



รูปที่ 7-29 แผนผังบล็อกของรอม ROM16x8

การทำงาน รอมในตัวอย่างนี้เป็นแบบง่าย มีเฉพาะสัญญาณแอดเดรส A(3:0) และสัญญาณข้อมูล D(7:0) เท่านั้น ดังนั้นเพียงแต่ระบุตำแหน่งที่ A(3:0) ก็จะปรากฏข้อมูลที่ D(7:0) บัสข้อมูล D(7:0) จะมีข้อมูลอยู่ตลอดเวลา และเป็นบัสทิศทางเดียว การออกแบบ ใช้วิธีสร้าง อะเรย์ (Array) สำหรับเก็บค่าคงที่ ดังนี้

- กำหนดข้อมูลชนิดอะเรย์ ชื่อ rom_array สำหรับเก็บ std_logic_vector (7 downto 0);
  type rom_array is array (0 to 15) of std_logic_vector (7 downto 0);
- กำหนดค่าคงที่ชื่อ rom เป็นข้อมูลชนิดอะเรย์ rom_array และให้เก็บค่า X”15” ....
  constant rom : rom_array := (X"15", ...);

โมเดล VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity rom16x8 is
port (A : in std_logic_vector (3 downto 0);
    D : out std_logic_vector (7 downto 0));
end rom16x8;
architecture rom16x8_beh of rom16x8 is
        type rom_array is array (0 to 15) of std_logic_vector (7 downto 0);
        constant rom : rom_array := (
        X"15", X"24", X"40", X"F0",
        X"01", X"02", X"03", X"04",
        X"05", X"06", X"07", X"08",
        X"09", X"0A", X"0B", X"0C");

begin
        D <= rom(to_integer(unsigned(A)));
end rom16x8_beh;
```

สำหรับฟังก์ชั่น

ข้อมูลที่เก็บอยู่ใน ROM

แปลงข้อมูลจาก std_logic_vector เป็น Integer

สำหรับการออกแบบแรมก็ใช้ อะเรย์ (Array) สำหรับเก็บข้อมูลเช่นเดียวกับรอม เพียงแต่ข้อมูลนั้นรับมาจากภายนอก ไม่ได้กำหนดไว้เป็นค่าคงที่เหมือนกับรอม ดังนั้นบัสข้อมูลของแรมต้องเป็นบัสอินพุทและบัสเอาท์พุท ซึ่งการสร้างบัสลักษณะนี้ทำได้ 2 แบบ แบบแรกทำเป็น 2 บัสแยกกัน บัสหนึ่งทำหน้าที่รับข้อมูลเข้าไปเก็บในแรม ส่วนอีกบัสสำหรับปล่อยข้อมูลออกมาใช้งานภายนอก โครงสร้างลักษณะนี้ใช้ได้กับอุปกรณ์ซีพีแอลดีและเอฟพีจีเอ เพราะไม่ต้องการบัสแบบ 3 สถานะ (3-state)

ส่วนแบบที่สองเหมาะสำหรับการสร้างลงบนอุปกรณ์ที่โครงสร้างภายในมีบัสแบบ 3 สถานะเช่นเอฟพีจีเอ เพราะแรมแบบที่สองนี้ บัสข้อมูลมีเพียงบัสเดียว เป็นทั้งบัสอินพุทและบัสเอาท์พุท ส่วนซีพีแอลดีนั้นไม่เหมาะกับโครงสร้างเช่นนี้

ตัวอย่างที่ 7.18 การออกแบบหน่วยความจำแรม แบบบัสข้อมูลทิศทางเดียว

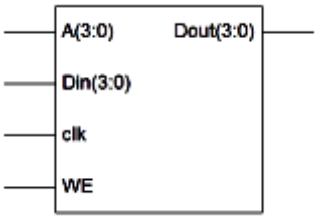
สัญญาณ A3 – A0 เป็นบัสแอดเดรส ใช้ระบุตำแหน่งข้อมูลได้ 16 ตำแหน่ง

Din3 – Din0 เป็นบัสข้อมูลเข้า

Dout3 – Dout0 เป็นบัสข้อมูลออก

CLK เป็นสัญญาณนาฬิกา ใช้ควบคุมการทำงานของแรม

WE เป็นสัญญาณควบคุมการเขียนข้อมูล



รูปที่ 7-30 แผนผังบล็อกของแรมแบบบัสข้อมูลทิศทางเดียว

การทำงาน สัญญาณ CLK ใช้ควบคุมการทำงานของแรม แรมจะทำงานได้ต้องมีสัญญาณนาฬิกาตลอด

เมื่อต้องการบันทึกข้อมูลไว้ในแรม

- ให้ป้อนค่าตำแหน่งที่ต้องการเก็บข้อมูลที่บัสแอดเดรส A(3:0)
- ป้อนข้อมูลเข้าที่ Din(3:0)
- ให้สัญญาณ WE เป็น ‘0’

เมื่อต้องการอ่านข้อมูลจากแรม

- ให้ป้อนค่าตำแหน่งที่ต้องการอ่านข้อมูลที่บัสแอดเดรส A(3:0)
- ให้สัญญาณ WE เป็น ‘1’
- จะได้ข้อมูลออกทาง Dout(3:0)

โมเดล VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RAM16x4 is
        port (A : in std_logic_vector(3 downto 0);
                Din : in std_logic_vector(3 downto 0);
                WE, clk : in std_logic;
                Dout :out std_logic_vector(3 downto 0));
end  RAM16x4;

architecture RTL of RAM16x4 is

begin
        process (clk)
        type ram_array is array (0 to 15) of std_logic_vector(3 downto 0);
        variable memory : ram_array;
        begin
                if clk = '1' and clk'event then
                        if WE = '0' then
                                memory(to_integer(unsigned(A))) := Din;
                        end if;
```

บันทึกข้อมูล

```
                    Dout <= memory(to_integer(unsigned(A)));
                end if;
            end process;
end RTL;
```

อ่านข้อมูล

ตัวอย่างที่ 7.19 การออกแบบหน่วยความจำแรม แบบมีบัสข้อมูลทิศทางเดียว

สัญญาณ A3 – A0 เป็นบัสแอดเดรส ใช้ระบุตำแหน่งข้อมูลได้ 16 ตำแหน่ง

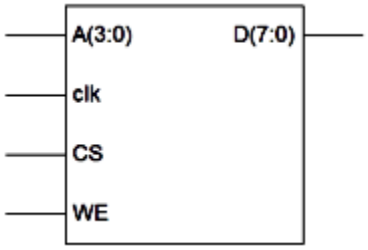
D3 – D0 เป็นบัสข้อมูล เป็นทั้งบัสอินพุทและบัสเอาท์พุท

CLK เป็นสัญญาณนาฬิกา ใช้ควบคุมการทำงานของแรม

WE เป็นสัญญาณควบคุมการเขียนข้อมูล

CS เป็นสัญญาณควบคุมบัสข้อมูล ถ้าสัญญาณนี้เป็น '1' บัสข้อมูลจะเป็น High impedance

SRAM สองทิศทาง



รูปที่ 7-31 แผนผังบล็อกของแรมแบบบัสข้อมูลสองทิศทาง

การทำงาน สัญญาณ CLK ใช้ควบคุมการทำงานของแรม แรมจะทำงานได้ต้องมีสัญญาณนาฬิกาตลอด

เมื่อต้องการบันทึกข้อมูลไว้ในแรม

- ให้ป้อนค่าตำแหน่งที่ต้องการเก็บข้อมูลที่บัสแอดเดรส A(3:0)
- ป้อนข้อมูลเข้าที่ D(3:0)
- ให้สัญญาณ CS เป็น '0'
- ให้สัญญาณ WE เป็น '0'

เมื่อต้องการอ่านข้อมูลจากแรม

- ให้ป้อนค่าตำแหน่งที่ต้องการอ่านข้อมูลที่บัสแอดเดรส A(3:0)
- ให้สัญญาณ CS เป็น '0'
- ให้สัญญาณ WE เป็น '1'
- จะได้ข้อมูลออกทาง D(3:0)

ถ้า CS เป็น '1' D(3:0) = "ZZZZZZZZ"

โมเดล VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RAM16x8Z is
        port (A : in std_logic_vector(3 downto 0);
                WE,CS, clk : in std_logic;
                D :inout std_logic_vector(7 downto 0));
end  RAM16x8Z;

architecture RTL of RAM16x8Z is

begin
        process (clk)
        type ram_array is array (0 to 15) of std_logic_vector(7 downto 0);
        variable memory : ram_array;
        variable control : std_logic_vector(1 downto 0);
        begin
                control := CS&WE;
                if clk = '0' and clk'event then
                        case control is
                                when "00" => memory(to_integer(unsigned(A))) := D;
                                when "01" => D <= memory(to_integer(unsigned(A)));
                                when others => D <= "ZZZZZZZZ";
                        end case;
                end if;
        end process;
end RTL;
```

แบบฝึกหัด

7.1 จากนิพจน์ต่อไปนี้ จงเขียนโมเดล VHDL

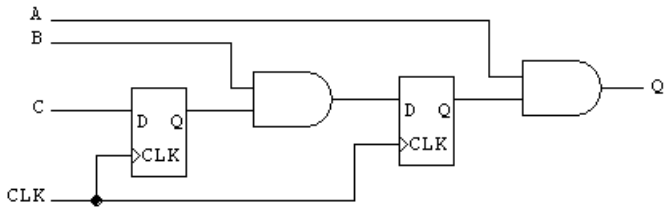
(ก) $F(A,B,C,D,E) = \Sigma m(0,1,4,10,17,20)$

(ข) $F(A,B,C,D,E) = \Pi M(7,13,18,21,26,31)$

(ค) $F(A,B,C,D,E) = (\overline{C}+D)(\overline{A}+B+C+\overline{D}+E)$

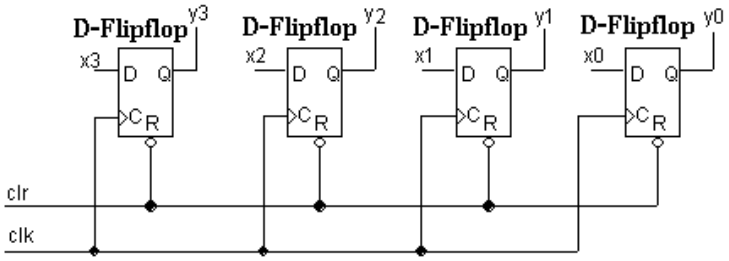
7.2 จากโลจิกไดอะแกรมต่อไปนี้ จงเขียนโมเดล VHDL

(ก)



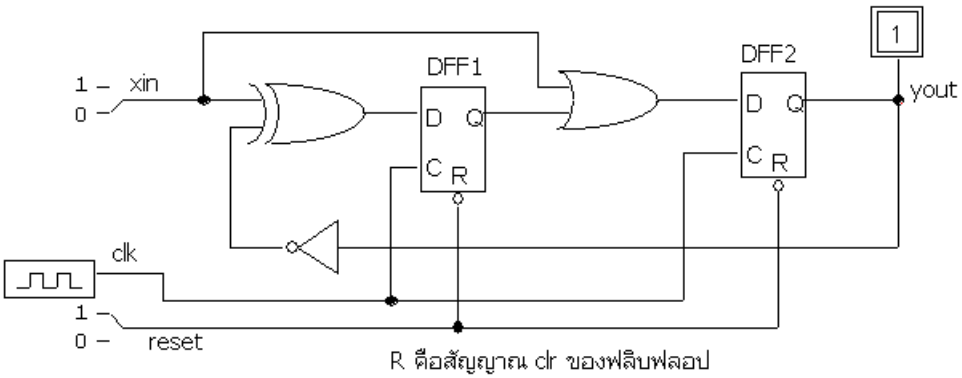
รูปที่ 7-32 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.2 (ก)

(ข)



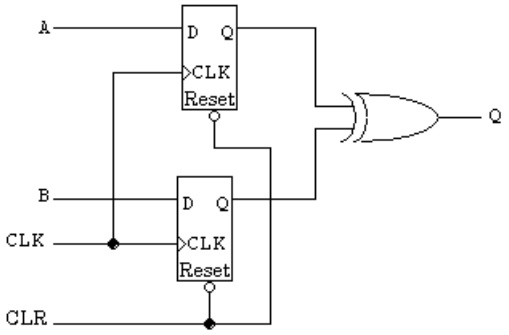
รูปที่ 7-33 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.2 (ข)

(ค)



รูปที่ 7-34 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.2 (ค)

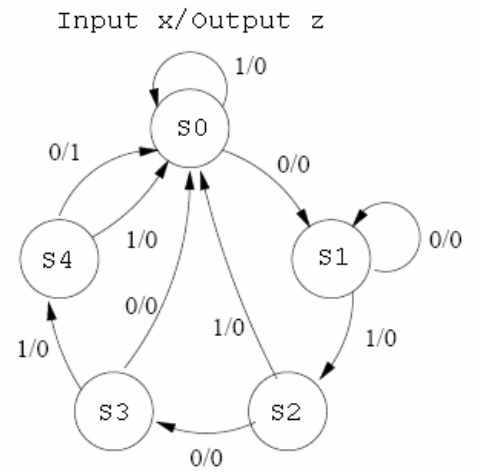
(ง)



รูปที่ 7-35 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.2 (ง)

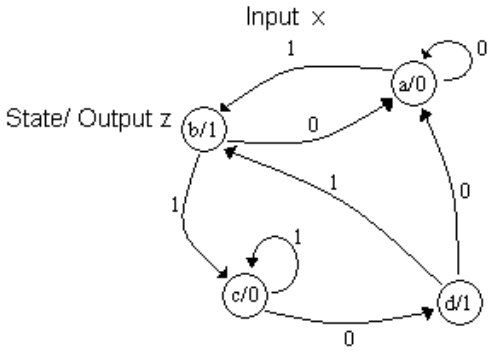
7.3 จากสเตทไดอะแกรมต่อไปนี้ จงเขียนโมเดล VHDL แบบเอาท์พุทไม่มีสัญญาณนาฬิกา

(ก)



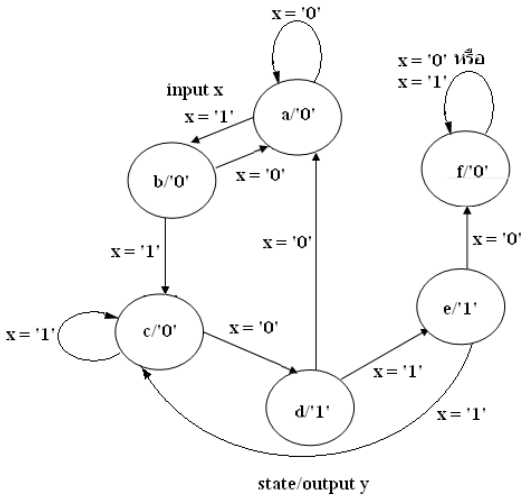
รูปที่ 7-36 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.3 (ก)

(ข)



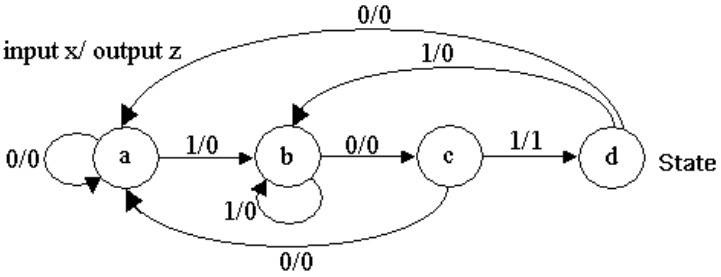
รูปที่ 7-37 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.3 (ข)

(ค)



รูปที่ 7-38 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.3 (ค)

(ง)



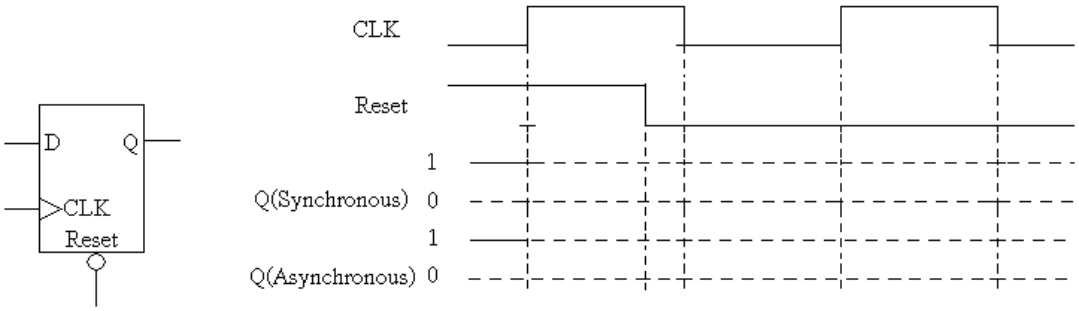
รูปที่ 7-39 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 7.3 (ง)

7.4 จากข้อ 7.3 จงเขียนโมเดล VHDL แบบเอาท์พุทมีสัญญาณนาฬิกา

7.5 จาก D Flipflop ในรูปที่ 7-36 สัญญาณ Reset ทำงานที่โลจิก 0 และ CLK ทำงานที่ขอบขาขึ้น

(ก) ถ้าสัญญาณ Reset เป็นแบบ Synchronous (ทำงานเมื่อมีสัญญาณนาฬิกามากระตุ้น) จงเขียนสัญญาณ Q(Synchronous) โดยต้องสอดคล้องกับ CLK และ Reset

(ข) ถ้าสัญญาณ Reset เป็นแบบ Asynchronous (เมื่อสัญญาณ Reset active ฟลิบฟลอปจะทำงานทันที) จงเขียนสัญญาณ Q(Asynchronous)โดยต้องสอดคล้องกับ CLK และ Reset



รูปที่ 7-40

7.6 จากข้อ 7.5 จงเขียน VHDL Code ของ D Flipflop เฉพาะในส่วนของ architecture มาทั้ง แบบ Synchronous และ Asynchronous

ตั้งแต่ข้อ 7.17 ถึง 7.11 ให้ออกแบบวงจรโดยเขียนเป็น โมเดล VHDL

7.7 จงเขียน VHDL Code สำหรับ ซิงโครนัสสเตทแมชชีน ที่มีอินพุท 2 อินพุทคือ X1 X2 และเอาท์พุท z โดยเอาท์พุท z จะเป็นโลจิก 1 ทุกครั้งที่อินพุท X1X2 เป็น “10” หรือ X1X2 เป็น “01” ติดต่อกันตั้งแต่ 2 สัญญาณนาฬิกาขึ้นไป และเอาท์พุท z จะค้างอยู่ที่โลจิก 1 นี้จนกว่า อินพุททั้งสองจะเป็น 0 ทั้งคู่ เอาท์พุท z จึงจะกลับมาเป็น 0

7.8 จากวงจรนับเลขไบนารีในตัวอย่างที่ 7.14 ให้เพิ่มสัญญาณควบคุมการนับ Count นับ ถ้าเป็นโลจิก ‘1’ จะนับสัญญาณนาฬิกา ถ้าเป็นโลจิก ‘0’ จะหยุดนับและคงค่าสถานะเดิม

7.9 จงออกแบบวงจรนับให้นับเลขฐานสิบเรียงลำดับดังนี้ 3 7 2 5 3 7 2 5

7.10 จากวงจรเลื่อนข้อมูลในตัวอย่างที่ 7.10 จงออกแบบวงจรให้เลื่อนทางซ้าย (จากบิตที่ 3 ไปบิตที่ 2)

7.11 จากข้อ 7.10 จงออกแบบวงจรให้มีสัญญาณควบคุมที่สามารถเลื่อนทางซ้ายหรือขวาก็ได้

7.12 วงจรชิพรีจิสเตอร์ สัญญาณเข้าแบบขนาน สัญญาณออกแบบอนุกรม