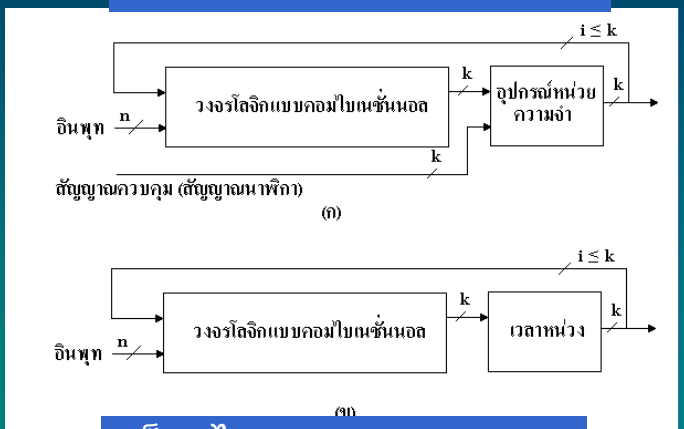
FSM (Finite-State Machine)

วงจรซีเควนเชียลแบบซิงโครนัส

(Synchronous Sequential Logic Circuit)

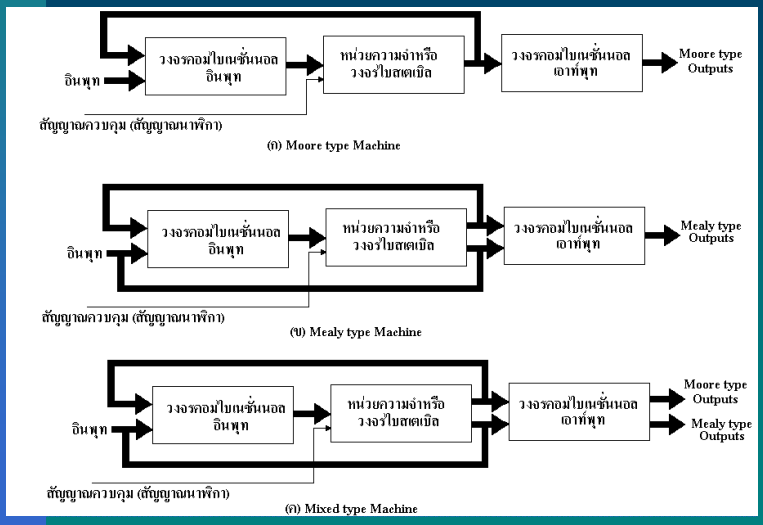
บทนำ

บล็อกไดอะแกรมของวงจรซิงโครนัส

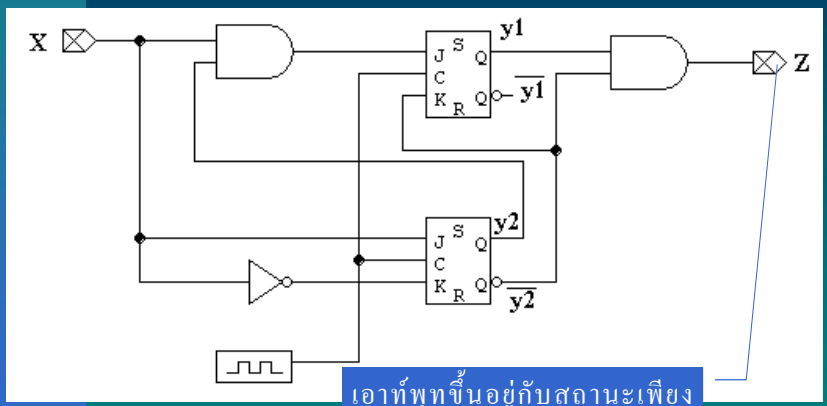


บล็อกไดอะแกรมของวงจรอะซิงโครนัส

ชนิดของวงจรซีเควนเชียลแบบซิงโครนัส

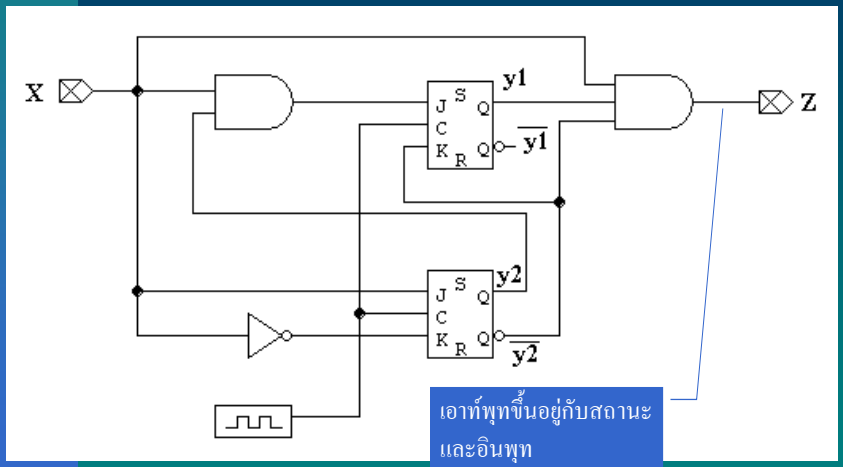


ตัวอย่างวงจร Moore type Machine



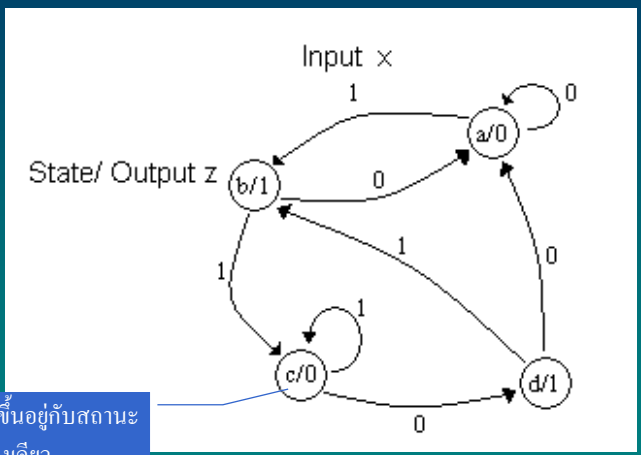
เอาท์พุทขึ้นอยู่กับสถานะเพียงอย่างเดียว

ตัวอย่างวงจร Mealy type Machine



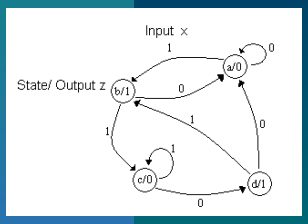
เอาท์พุทขึ้นอยู่กับสถานะ และอินพุท

ตัวอย่างสเตตไดอะแกรมแบบมัวร์ (Moore type State Diagram)



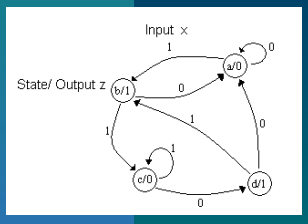
เอาท์พุทขึ้นอยู่กับสถานะเพียงอย่างเดียว

ตัวอย่างการเขียน VHDL แบบมัวร์




```
architecture moore_beh of moore is
type state_type is (a, b, c, d);
signal state: state_type;
begin
   state_proc: process (clk)
   begin
   …..
   end process;
   output_proce: process (state)
   begin
   ……
   end process;
end moore_beh;
```

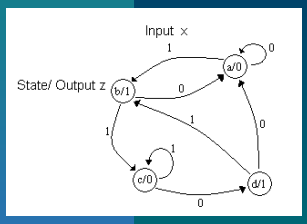
ตัวอย่างการเขียน VHDL แบบมัวร์ ส่วนสเตต




```
state_proc: 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;
```

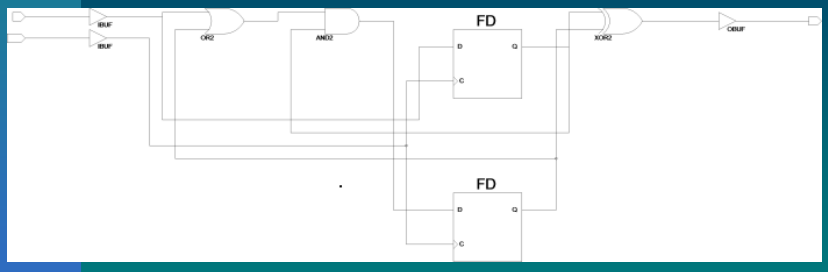
```
          when b => if x = '1' then
                              state <= c;
                    else
                              state <= a;
                    end if;
          when c => if x = '1' then
                              state <=c;
                    else
                              state <= d;
                    end if;
          when d => if x = '1' then
                              state <= b;
                    else
                              state <= a;
                    end if;
     end case;
    end if;
end process;
```

ตัวอย่างการเขียน VHDL แบบมัวร์ ส่วนเอาท์พุท

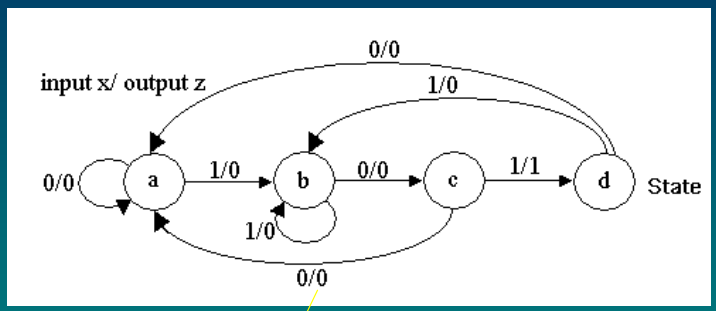



```
output_proce: process (state)
   begin
        case state is
                when a => z <= '0';
                when b => z <= '1';
                when c => z <= '0';
                when d => z <= '1';
        end case;
   end process;
end moore_beh;
```

Schematic ที่ได้จากการสังเคราะห์

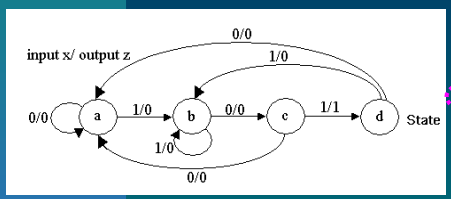


ตัวอย่างสเตต ไดอะแกรมแบบเมียลี่ (Mealy type State Diagram)



เอาท์พุทขึ้นอยู่กับสถานะและอินพุท

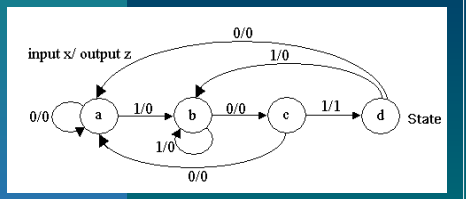
ตัวอย่างการเขียน VHDL แบบเมียลี่




```
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;
```

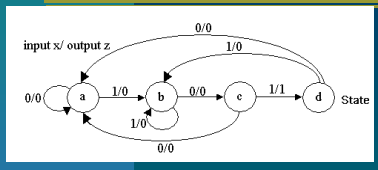
```
architecture mealy_beh of mealy is
type state_type is (a, b, c, d);
signal state: state_type;
begin
   state_proc: process (clk)
   begin
   ….
   end process;
   output_proc: process (state)
   begin
   …..
   end process;
end mealy_beh;
```

ตัวอย่างการเขียน VHDL แบบเมียลี่ ส่วนสเตต



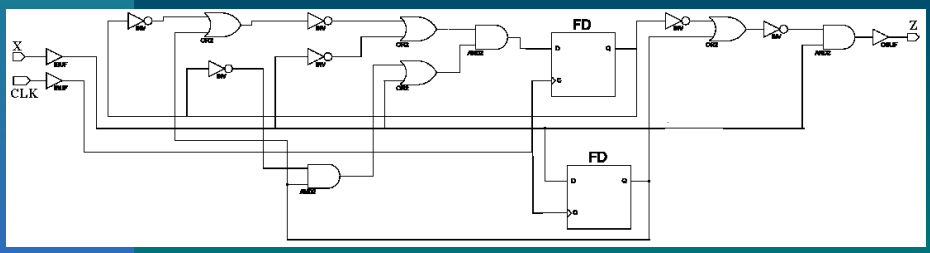

```
state_proc: process(clk)
 begin
  if clk = '1' and clk'event then
     case state is
        when a => if  x = '1' then state <= b;
                  else state <= a;
                  end if;
        when b => if x = '1' then state <= b;
                  else state <= c;
                  end if;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

ตัวอย่างการเขียน VHDL แบบเมียลี่ ส่วนเอาท์พุท

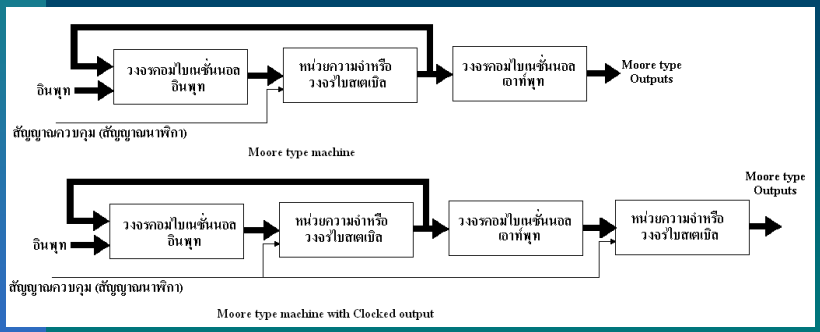



```
output_proc: process(state)
  begin
     case state is
           when a => if x = '1' then z <= '0';
                     else    z <= '0'; end if;
           when b => if x = '1' then z <= '0';
                     else    z <= '0'; end if;
           when c => if x = '1' then z <= '1';
      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

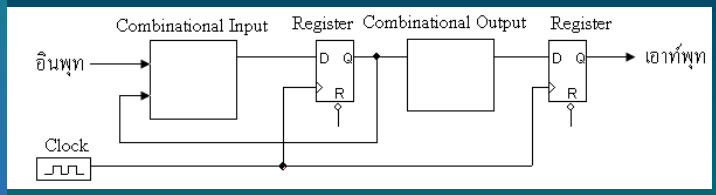
Schematic ที่ได้จากการสังเคราะห์



Moore type Machine with Clocked Output

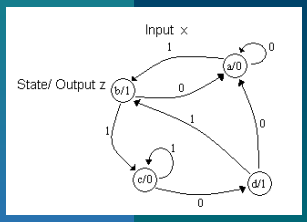


Moore type Machine with Clocked Output



ค่าของสัญญาณเอาท์พุททำงานสอดคล้องกับ (Synchronized) สัญญาณนาฬิกา และสามารถเคลียร์ค่าเอาท์พุทได้

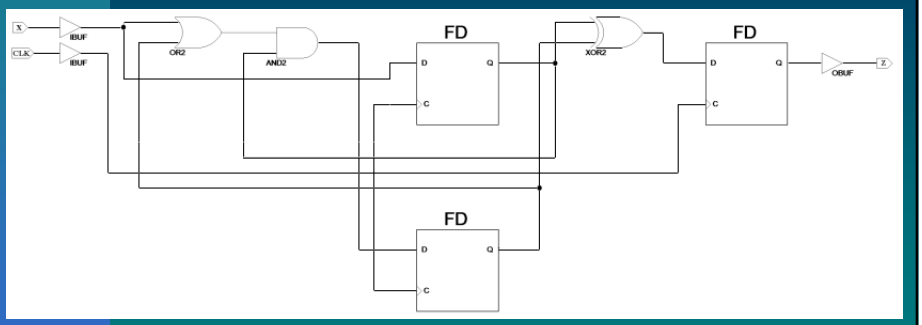
ตัวอย่างการเขียน VHDL แบบมัวร์ แบบมี Clock ที่เอาท์พุท



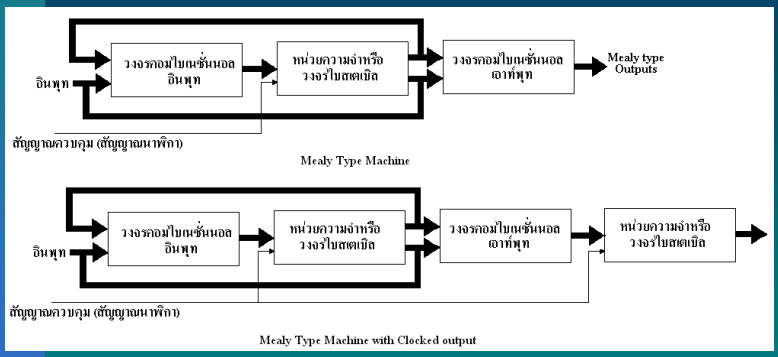

```
state_proc: process (clk)
   begin
    if clk = '1' and clk'event then
     case state is
         when a => if x = '1' then
                              state <= b;
                   else
                              state <= a;
                   end if;
                    z <= '0';
```

```
         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;
```

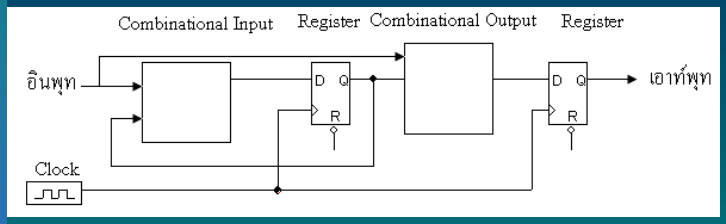
Schematic ที่ได้จากการสังเคราะห์



Mealy type Machine with Clocked Output

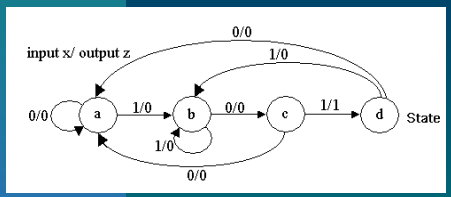


Mealy type Machine with Clocked Output



ค่าของสัญญาณเอาท์พุททำงานสอดคล้องกับ (Synchronized) สัญญาณนาฬิกา และสามารถเคลียร์ค่าเอาท์พุทได้

ตัวอย่างการเขียน VHDL แบบเมียลี่ แบบมี Clock ที่เอาท์พุท




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

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

การเข้ารหัสสเตต (State Encoder)

นอกจากนี้
รหัสที่กำหนดโดยซอฟต์แวร์สังเคราะห์วงจร
ผู้ใช้งานกำหนดเอง

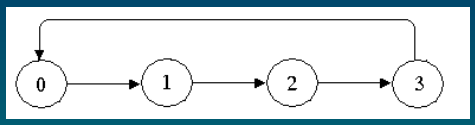
การเข้ารหัสสเตต (State Encoder)

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

การเข้ารหัสสเตต แบบผู้ใช้กำหนดขึ้นเอง

```
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;
```

วงจรนับเลขไบนารีขนาด 2 บิต แบบไม่มีสัญญาณควบคุม เขียนแบบสเตทแมชชีน

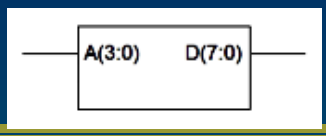



```
entity binco is
    port (clk : in std_logic;
          y : out std_logic_vector(1 downto 0));
end binco;
```

```
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;
```

หน่วยความจำ ROM



การออกแบบ ใช้วิธีสร้าง อะเรย์ (Array) สำหรับเก็บค่าคงที่

1. 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);
```

2. กำหนดค่าคงที่ชื่อ rom เป็นข้อมูลชนิดอะเรย์ rom_array และให้เก็บค่า X”15” ….

```
constant rom : rom_array := ( X"15", ……);
```

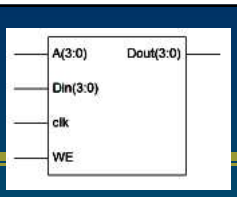
```
use ieee.numeric_std.all;
entity rom16x8 is
port (A : in std_logic_vector (3 downto 0);
      D : out std_logic_vector (7 downto 0));
end rom16x8;
architecture rom16x8_beh of rom16x8 is
type rom_array is array (0 to 15) of std_logic_vector (7 downto 0);

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;
```

หน่วยความจำ RAM

สำหรับการออกแบบแรมก็ใช้ อะเรย์ (Array) สำหรับเก็บข้อมูล เช่นเดียวกับรอม เพียงแต่ข้อมูลนั้นรับมาจากภายนอก ไม่ได้กำหนดไว้เป็นค่าคงที่เหมือนกับรอม ดังนั้นบัสข้อมูลของแรมต้องเป็นบัสอินพุท และบัสเอาท์พุท ซึ่งการสร้างบัสลักษณะนี้ทำได้ 2 แบบ

แบบแรกทำเป็น 2 บัสแยกกัน บัสหนึ่งทำหน้าที่รับข้อมูลเข้าไปเก็บในแรม ส่วนอีกบัสสำหรับปล่อยข้อมูลออกมาใช้งานภายนอก โครงสร้างลักษณะนี้ใช้ได้กับอุปกรณ์ซีพีแอลดีและเอฟพีจีเอ เพราะไม่ต้องการบัสแบบ 3 สถานะ (3-state) ส่วนแบบที่สองเหมาะสำหรับการสร้างลงบนอุปกรณ์ที่โครงสร้างภายในมีบัสแบบ 3 สถานะเช่นเอฟพีจีเอ เพราะแรมแบบที่สองนี้บัสข้อมูลมีเพียงบัสเดียว เป็นทั้งบัสอินพุทและบัสเอาท์พุท ส่วนซีพีแอลดีนั้นไม่เหมาะกับโครงสร้างเช่นนี้

หน่วยความจำแรม แบบบัสข้อมูลทิศทางเดียว



การทำงาน สัญญาณ CLK ใช้ควบคุมการทำงานของแรม แรมจะทำงานได้ต้องมีสัญญาณนาฬิกาตลอด

เมื่อต้องการบันทึกข้อมูลไว้ในแรม

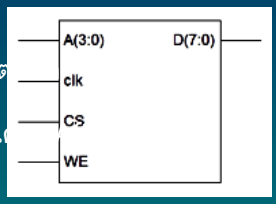
- ให้ป้อนค่าตำแหน่งที่ต้องการเก็บข้อมูลที่บัสแอดเดรส A(3:0)
- ป้อนข้อมูลเข้าที่ Din(3:0)
- ให้สัญญาณ WE เป็น '0'

เมื่อต้องการอ่านข้อมูลจากแรม

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

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

หน่วยความจำแรม แบบมีบัสข้อมูลสองทิศทาง



การทำงาน สัญญาณ CLK ใช้ควบคุมการทำงาน
ของแรม แรมจะทำงานได้ต้องมีสัญญาณนาฬิกา

เมื่อต้องการบันทึกข้อมูลไว้ในแรม

- ให้ป้อนค่าตำแหน่งที่ต้องการเก็บข้อมูลที่บัสแอดเ
- 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"

```
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;
```