Sequential Statement

คำสั่งแบบ Sequential

- IF Statements
  - IF-END
  - IF-ELSE
  - IF-ELSIF-ELSE
- CASE Statement
- LOOP Statement
  - FOR LOOP
  - WHILE LOOP
- WAIT Statements
  - WAIT FOR
  - WAIT ON
  - WAIT UNTIL
  - WAIT

การกำหนดค่าสัญญาณและตัวแปร

- ในการเขียนภาษา VHDL มักมีการใช้ SIGNAL และ VARIABLE
- VARIABLE สามารถใช้ได้ภายในคำสั่ง PROCESS เท่านั้น
- ยกเว้น SHARED VARIABLE ถูกนิยามขึ้นมาใหม่ใน VHDL'93 ที่หลายๆ Process สามารถใช้ร่วมกัน

SHARED VARIABLE shared_variable_name : shared_variable_type;

การกำหนดค่าสัญญาณ และตัวแปร

- การกำหนดค่าให้กับ VARLABLE จะใช้เครื่องหมาย `:=` หรือเรียกว่า Variable assignment
- การกำหนดค่าให้กับ SIGNAL จะใช้เครื่องหมาย `<=` หรือเรียกว่า Signal assignment
- การส่งผ่านค่าระหว่าง VARIABLE กับ SIGNAL สามารถทำได้แต่จะต้องเป็นข้อมูลชนิดเดียวกัน
  - Signal to variable
  - Variable to signal

- ตัวอย่าง Signal assignment

```
p1 : PROCESS
BEGIN
    WAIT FOR 10 ns;
        sum_a <= sum_a + 1 ;
        sum_b <= sum_a + 1;
END PROCESS p1;
```


- ตัวอย่าง Variable assignment

```
p2 : PROCESS
  VARIABLE sum_a , sum_b : INTEGER;
BEGIN
WAIT FOR 10 ns;
    sum_a := sum_a + 1 ;
    sum_b := sum_a + 1 ;
END PROCESS p2;
```


การใช้คำสั่ง Process

- ภายในคำสั่ง PROCESS จะบรรจุไปด้วยคำสั่งที่เป็น Concurrent statement ซึ่งเป็นคำสั่งที่ทำงานตามลำดับ
- มีเป็นประโยชน์มากในการออกแบบ ผู้ออกแบบสามารถเขียนในลักษณะแบบ อัลกอริทึมได้
- คล้ายคลึงกับการออกแบบโปรแกรมทางซอฟต์แวร์

คำสั่ง PROCESS มีรูปแบบการเขียนดังนี้ ในรูปแบบของ VHDL'93

```
optional_label :
PROCESS(optional_sentitivity_list) IS
    declaration
BEGIN
    sequential_statements;
    ...
END PROCESS optional_label;
```

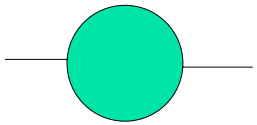
โดยรูปแบบของการเขียน Process จะมีสองรูปแบบ คือ

- Combinational process
- Clocked process

```
PROCESS(.. )                PROCESS(.. )
BEGIN                       BEGIN
...  ...  ...                    ...  ...  ...
...  ...  ...                    ...  ...  ...
END PROCESS;                END PROCESS;
```

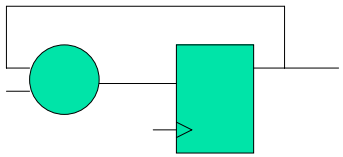

Combinational process

```
PROCESS(..  )
BEGIN
... ... ...
... ... ...
... ... ...
END PROCESS;
```



Clocked process

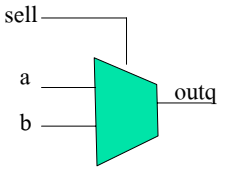
```
PROCESS (..  )
BEGIN___
... ... ...
... ... ...
END PROCESS;
```



การเขียน Combinational Process

- รูปแบบการเขียน Combinational Process ที่มี optional_sensitivity_list แบบสมบูรณ์

```
PROCESS (a, b, sell)
BEGIN
    IF (sell  =  '1')  THEN
       outq  <=  a;
    ELSE
       outq  <=  b;
    END  IF;
END  PROCESS;
```



Clock Process

การเขียนรูปแบบของ Process แบบสร้างส่วนของวงจรที่มีการทำงานขึ้นกับสัญญาณนาฬิกา

```
PROCESS (clk)
BEGIN
    IF (clk'event AND clk = '1') THEN
    -- All combinational logic
    END IF;
END PROCESS;
```

Sensitivity List มีเพียงสัญญาณ clk เท่านั้น


```
PROCESS (reset , clk)
BEGIN
    IF (reset = '1') THEN
    -- Reset all registers
    ELSIF ( clk'event AND clk = '1') THEN
    -- All combinational logic
    END IF;
END PROCESS;
```

Sensitivity List มีเพียงสัญญาณ reset,clk เท่านั้น

การใช้คำสั่ง IF

คำสั่ง IF ในภาษา VHDL มีหลายรูแปบบดังนี้

```
-- IF Statement
    IF condition THEN
        sequential_statements
    END IF;
```



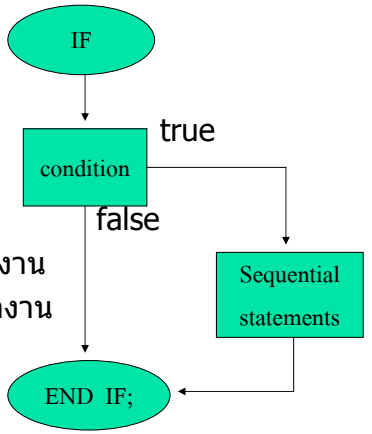
```
-- IF-ELSE Statements
IF condition THEN
    sequential_statement
ELSE
    sequential_statements
END IF;
```

```
- - IF -ELSIF  Statement
    IF  condition_1  THEN
       sequential_statements
    ELSIF  condition_2  THEN
       sequential_statements
     ELSIF  condition_3  THEN
       sequential_statements
    ...
    ELSE
       sequential_statements
    END  IF;
```


การใช้คำสั่ง IF-END

```
- -  IF  Statement
IF  condition  THEN
   sequential_Statements
END  IF;
```

CONDITION เป็นจริง(True)ทำงาน
CONDITION เป็นจริง(False)ทำงาน



การใช้คำสั่ง CASE

คำสั่ง CASE ในภาษา VHDL มีลักษณะการทำงานคล้ายกับคำสั่ง IF แต่ที่แตกต่าง คือ EXPRESSION ในคำสั่ง CASE ใช้กำหนดตัวเลือกที่จะนำมาเปรียบเทียบกับ Choice เพื่อทำ Sequential statements ต่างๆ มีความสำคัญเท่ากันโดยมีรูปแบบดังนี้

```
CASE  expression  IS
   WHEN  choice_1  =>
      sequential_statements
      ...
```



```
   WHEN  choice_2     choice_3  =>
      sequential_statements
      ...
   WHEN  choice_4  TO  choice_n  =>
      sequential_statements
      ...
   WHEN  OTHERS  =>
      sequential_statements
      ...
END  CASE;
```

ในรูปแบบของ VHDL'93 ในคำสั่ง CASE Statement จะมีส่วนของ Optional label เพิ่มขึ้นมา

```
Label_name  CASE expression IS
    WHEN  choice_1  =>
        sequential_statements
        ...
    WHEN  choice_2  choice_3  =>
        sequential_statements
        ...
```



```
    WHEN  choice_4  TO  choice_n  =>
        sequential_statements
        ...
    WHEN  OTHERS  =>
        sequential_statements
        ...
END  CASE  label_name;
```


กฎการใช้คำสั่ง CASE มีรูปแบบดังนี้

- ตัวเลือก (Choice) จะต้องเป็นแบบระบุตัวเลือกเป็นค่าที่แน่นอน (Single value)
- เลือกตัวใดตัวหนึ่งจากตัวเลือกมากกว่า 1 ตัวจะใช้เครื่องหมาย (" | " ซึ่งมีความหมาย = "or")
- การระบุตัวเลือกเป็นช่วงข้อมูล (Value range)โดยใช้คำสั่ง TO ได้
- "WHEN OTHERS" จะใช้เป็นตัวเลือกที่ครอบคลุมที่เหลือทั้งหมด

ข้อควรระวังในการใช้ CASE

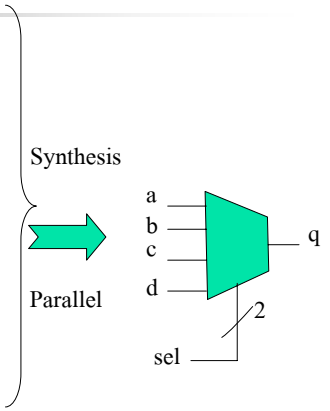
- เงื่อนไขต้องไม่ขัดแย้งกัน
- ต้องกำหนดตัวเลือกครบทุกกรณี

การทำงานของ CASE Statement และการสังเคราะห์วงจร

```
PROCESS  (sel, a, b, c, d)
BEGIN
  CASE  sel  IS
      WHEN "00"  =>
            q  <=  a;
      WHEN  "01"  =>
            q  <= b;
```



```
      WHEN  "10"  =>
            q  <=  c;
      WHEN  OTHERS  =>
            q  <=  d;
  END  CASE
END  PROCESS;
```

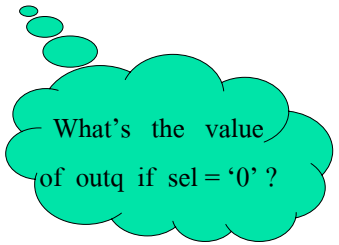


Combinational Process : Incomplete Assignment

```
LIBRARY  ieee;
USE  ieee.std_logic_list.ALL;
ENTITY  incomp_if  IS
  PORT  (a,  sel  :  IN  std_logic;
          ourq  :  out  std_logic  );
  END  incomp_if;
ARCHITECTURE  rtl  OF  incomp_if  IS
BEGIN
```



```
      PROCESS  (a,  sel )
      BEGIN
            IF  (sel  =  '1' ) THEN
                  outq  <=  a;
            END  IF;
      END  PROCESS;
END  rtl;
```



การใช้คำสั่ง NULL

- คำสั่ง NULL หมายถึง ไม่กระทำการใดๆโดยคำสั่งนี้
- ใช้เป็นค่า Default ของสัญญาณที่ใช้ในคำสั่ง CASE ในกรณีที่ไม่ต้องการเปลี่ยนแปลงค่าใดในกรณี OTHERS


```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY case_ex2 IS
PORT (a : IN std_logic_vector(1 DOWNTO 0);
        q1, q2, q3, : OUT std_logic ) ;
END case_ex2;
ARCHITECTURE behavioral OF case_ex2 IS
BEGIN
    p1 : PROCESS(a)
    BEGIN
        q1 <= '0'; - - Initial default value
        q2 <= '0';
        q3 <= '0';
```



```
        CASE a IS
            WHEN "01" => q1 <= '1';
            WHEN "10" => q2 <= '1';
            WHEN "11" => q3 <= '1';
            WHEN OTHERS => NULL;
        END CASE;
    END PROCESS p1;
END behavioral;
```

Do nothing

การใช้คำสั่ง Attributes

- ชนิดของข้อมูล (Type) หรือวัตถุ (Object) ต่างๆ ภายในภาษา VHDL สามารถอ้างอิงถึงคุณสมบัติภายในของชนิดของข้อมูลหรือวัตถุ
- เรียกว่า Attributes
- การเขียนอ้างอิงถึง Attributes จะใช้เครื่องหมาย Notation (')

- มีรูปแบบการเขียนดังนี้

  Object'attribute_name

  - X'high
  - X'low
  - X'length
  - X'event

การใช้คำสั่ง LOOP

คือการทำงานในลักษณะเป็นวงรอบ (Loop) โดยในภาษา VHDL มี 2 รูปแบบ คือ

- For Loop Statement
- While Loop Statement

การใช้คำสั่ง FOR LOOP

คำสั่ง For Loop ในภาษา VHDL มีรูปแบบดังนี้

```
optional_label : FOR parameter IN
discrete_range LOOP
    sequential_statement
    ...
END LOOP optional_label;
```


การใช้คำสั่ง WHILE LOOP

- คำสั่ง While Loop ไม่สามารถเขียนในโมเดลเพื่อนำไปสังเคราะห์เป็นวงจรได้
- นิยมเขียนคำสั่งนี้ในโมเดลสำหรับใช้ในการทดสอบวงจรเท่านั้น เช่น การอ้างข้อมูลจากไฟล์ หรือ ใช้สร้างสัญญาณนาฬิกา เป็นต้น รูปแบบการเขียน

```
WHILE condition LOOP
    sequential_statements
END LOOP;
```

การใช้คำสั่ง WAIT

คำสั่ง WAIT เป็นคำสั่งที่ใช้หยุดการทำงานภายใน Process มี 4 แบบดังนี้

```
WAIT FOR specific_time;
WAIT ON signal_list;
WAIT UNTIL condition;
WAIT;
```


การใช้คำสั่ง WAIT FOR

```
WAIT FOR specific_time;
```

เป็นการสั่งให้ Process หยุดรอเป็นช่วงเวลา เช่น

```
WAIT FOR 10 ns;
```

หมายถึงการสั่งให้ Process หยุดรอเป็นเวลา 10 ns

การใช้คำสั่ง WAIT ON

```
WAIT ON signal_list;
```

เป็นการสั่งให้ Process หยุดรอจนกว่าจะเกิด Event ของสัญญาณ เช่น

```
WAIT ON a;
```

หมายถึงสั่งให้ Process หยุดการทำงานจนกว่าจะเกิด Event ของสัญญาณ a

การใช้คำสั่ง WAIT UNTIL

```
WAIT UNTIL condition;
```

เป็นการสั่งให้ Process หยุดการทำงานจนกว่าเงื่อนไขที่กำหนดใน Process จะเป็นจริง เช่น

```
WAIT UNTIL clk'event AND clk = '1';
```

หมายถึงให้ Process หยุดการทำงานจนกว่าจะเกิด Event ของสัญญาณ clk และสัญญาณ clk มีค่าเท่ากับ '1' คือ (ขอบขาขึ้นของสัญญาณนาฬิกา)

การใช้คำสั่ง WAIT

มีรูปแบบการเขียนดังนี้

WAIT;

เป็นการสั่งให้ Process หยุดรอไม่มีที่สิ้นสุด

คำสั่ง WAIT vs Sensitivity List

- การเขียนรูปแบบคำสั่ง WAIT ON แทนการเขียน Sensitivity list

```
PROCESS                                  PROCESS (a, b, sel)
BEGIN                                    BEGIN
  IF (sel = '1') THEN                      IF (sel = '1')
  THEN
      outq <= a;                               outq <= a;
  ELSE                                     ELSE
      outq <= b;                               outq <= b;
  END IF;                                  END IF;
  WAIT ON (a, b, sel)                    END PROCESS;
END PROCESS;
```