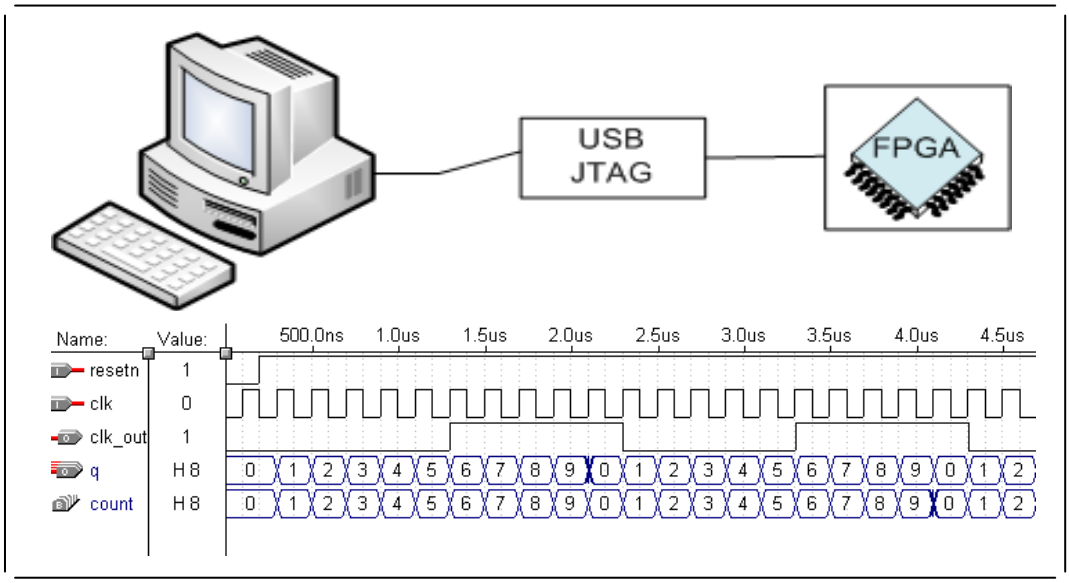
หนังสือเล่มนี้ผู้เขียนอนุญาตให้ทำการเผยแพร่ เพื่อการเรียน การสอน หรือพัฒนาเทคโนโลยีแก่ประเทศชาติ แต่ ห้ามคัดลอกหรือกระทำการใด ๆเพื่อการค้าหรือแสวงหาผลประโยชน์จากหนังสือเล่มนี้ จะมีความผิดตาม พรบ.

การออกแบบวงจรดิจิตอล และ การประยุกต์ใช้งานภาษา VHDL



เรียบเรียงโดย

วุฒิชัย สง่างาม

สาขาวิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์และสถาปัตยกรรมศาตร์
มหาวิทยาลัยเทคโนโลยีราชมงคลอีสาน นครราชสีมา

ตำรา

การออกแบบวงจรดิจิตอล และ การประยุกต์ใช้งานด้วยภาษา VHDL

วุฒิชัย สง่างาม

สาขาวิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์และสถาปัตยกรรมศาตร์
มหาวิทยาลัยเทคโนโลยีราชมงคลอีสาน นครราชสีมา

คำนำ

ตำราเล่มนี้จัดทำขึ้นเพื่อใช้ประกอบการเรียนในรายวิชาปฏิบัติการวงจรดิจิตอล การออกแบบวงจรดิจิตอลในขั้นสูง ระบบควบคุมดิจิตอล และปฏิบัติการระบบควบคุมดิจิตอล ที่ปัจจุบันจะต้องใช้องค์ความรู้ทางด้านการออกแบบวงจรดิจิตอลยุคใหม่ เมื่อมีการใช้ชิพไอซีตระกูลซีพีแอลดีหรือเอฟพีจีเอ โดยทำให้ผู้ออกแบบวงจรสามารถใช้ซอฟต์แวร์ช่วยออกแบบและดูผลจำลองการทำงาน เพื่อลดความผิดพลาดในขั้นตอนการออกแบบ สามารถเรียนรู้ได้ด้วยตนเอง

ดังนั้นเนื้อหาในตำราเล่มนี้จะมุ่งเน้นในการอธิบายโครงสร้างของภาษา VHDL เพื่อนำไปปฏิบัติในการออกแบบวงจรดิจิตอลได้ด้วยตนเอง โดยมีการจัดลำดับการเรียนออกเป็น 6 บทเรียนดังนี้

บทที่ 1 โครงสร้างภาษา VHDL
บทที่ 2 ประเภทของข้อมูล
บทที่ 3 ประเภทของตัวกระทำและตัวตรวจคุณลักษณะข้อมูล
บทที่ 4 ชุดคำสั่งชนิดแข่งขนาน
บทที่ 5 ชุดคำสั่งชนิดเรียงลำดับ
บทที่ 6 ไฟไนต์สเตตแมชชีน

ผู้เขียนขอขอบคุณเพื่อนร่วมงานในสาขาวิชาวิศวกรรมไฟฟ้าทุกท่านที่เป็นแรงผลักดันในการจัดทำตำราเล่มนี้จนสำเร็จ สุดท้ายนี้หากตำราเล่มนี้มีข้อผิดพลาดประการใด ผู้เขียนยินดีขอน้อมรับเพื่อจะได้นำไปปรับปรุงแก้ไขในครั้งต่อไป และหวังเป็นอย่างยิ่งว่าตำราเล่มนี้คงมีประโยชน์ต่อผู้อ่านทุกท่าน

วุฒิชัย สง่างาม
พฤษภาคม 2552
Email: wutichai@rmuti.ac.th

สารบัญ

บทที่ 1 โครงสร้างภาษา VHDL 1

1.1 โครงสร้างเบื้องต้นของภาษา VHDL 1

1.1.1 โครงสร้างการเขียนภาษา VHDL ในส่วนของ LIBRARY 1

1.1.2 โครงสร้างการเขียนภาษา VHDL ในส่วนของ ENTITY 2

1.1.3 โครงสร้างการเขียนภาษา VHDL ในส่วนของ ARCHITECTURE 3

1.2 ตัวอย่างการเขียนโปรแกรมภาษา VHDL 4

1.3 โจทย์ปัญหา 6

บทที่ 2 ประเภทของข้อมูล 7

2.1 ประเภทของข้อมูลที่ถูกกำหนดไว้แล้ว 7

2.2 ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง 11

2.2.1 ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง แบบ INTEGER 11

2.2.2 ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง แบบ ENUMERATED 12

2.3 ข้อมูลย่อย 12

2.4 ข้อมูลอะเรย์ 12

2.5 ข้อมูลเรคอร์ด 14

2.6 ประเภทข้อมูลที่คิดเครื่องหมายและ ไม่คิดเครื่องหมาย 14

2.7 การแปลงค่าข้อมูล 16

2.8 ตัวอย่างการเขียนโปรแกรมภาษา VHDL 17

2.9 โจทย์ปัญหา 19

บทที่ 3 ประเภทของตัวกระทำและตัวตรวจคุณลักษณะข้อมูล 20

3.1 ประเภทของตัวกระทำ 20

3.1.1 ตัวกระทำชนิดการส่งผ่านค่า 20

3.1.2 ตัวกระทำทางด้านลอจิก 21

3.1.3 ตัวกระทำทางด้านคณิตศาสตร์ 22

3.1.4 ตัวกระทำทางด้านเปรียบเทียบ 23

3.1.5 ตัวกระทำทางด้านการรวมข้อมูล 23

3.2 ตัวตรวจสอบคุณลักษณะข้อมูล 24

3.3 ตัวตรวจสอบคุณลักษณะของสัญญาณ 25

3.4 การประกาศใช้ GENERIC 26

3.5 ตัวอย่างการเขียนโปรแกรมภาษา VHDL 26

3.6 โจทย์ปัญหา 29

สารบัญ (ต่อ)

บทที่ 4 ชุดคำสั่งชนิดแข่งขนาน 30

4.1 กลุ่มคำสั่ง WHEN 30

4.1.1 รูปแบบ WHEN/ELSE 30

4.1.2 รูปแบบ WITH/SELECT/WHEN 33

4.2 ชุดคำสั่ง GENERATE 39

4.2.1 รูปแบบ FOR/GENERATE 39

4.2.2 รูปแบบ IF/GENERATE 40

4.3 โจทย์ปัญหา 41

บทที่ 5 ชุดคำสั่งชนิดเรียงลำดับ 42

5.1 ชุดคำสั่ง PROCESS 42

5.2 ตัวแปรประเภท VARIABLE และ SIGNAL 43

5.3 การใช้คำสั่ง IF ในชุดคำสั่ง PROCESS 44

5.4 การใช้คำสั่ง WAIT ในชุดคำสั่ง PROCESS 47

5.4.1 รูปแบบ WAIT ON 47

5.4.2 รูปแบบ WAIT UNTIL 47

5.4.3 รูปแบบ WAIT FOR 47

5.5 การใช้คำสั่ง CASE ในชุดคำสั่ง PROCESS 49

5.6 โจทย์ปัญหา 55

บทที่ 6 ไฟไนต์สเตตแมชชีน 56

6.1 รูปแบบมอร์สเตตแมชชีนสำหรับภาษา VHDL 57

6.2 รูปแบบเมลลี่สเตตแมชชีนสำหรับภาษา VHDL 64

6.3 โจทย์ปัญหา 68

ภาคผนวก ก. ประวัติความเป็นมาภาษา VHDL และข้อกำหนดในการตั้งชื่อตัวแปร 70

ภาคผนวก ข. คำสงวนในภาษา VHDL 73

ภาคผนวก ค. Standard Package ชื่อ STD_LOGIC_1164 74

ภาคผนวก ง. การใช้งานโปรแกรม Quartus-II เบื้องต้น สำหรับการเขียนโปรแกรม VHDL 77

บรรณานุกรม 85

1 โครงสร้างภาษา VHDL

Code Structure

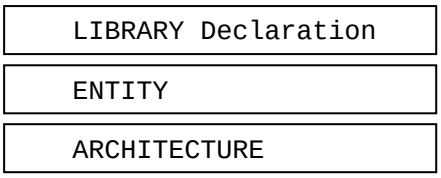
การเขียนโปรแกรมภาษา VHDL เป็นการเขียนบรรยายพฤติกรรมการทำงานของวงจรดิจิตอล จำเป็นอย่างยิ่งที่จะต้องเข้าใจถึงคุณลักษณะโครงสร้างของการเขียน การส่งผ่านค่า การใช้งานชุดคำสั่งต่างๆ รวมถึงการติดต่อส่วนอินพุทและเอาท์พุท ซึ่งจะได้อธิบายดังรายละเอียดต่อไปนี้

1.1 โครงสร้างเบื้องต้นของภาษา VHDL

โครงสร้างของภาษา VHDL จะมีองค์ประกอบที่สำคัญหรือสิ่งที่จำเป็นอย่างน้อย 3 ส่วนคือ

- ส่วนของ LIBRARY
- ส่วนของ ENTITY
- ส่วนของ ARCHITECTURE

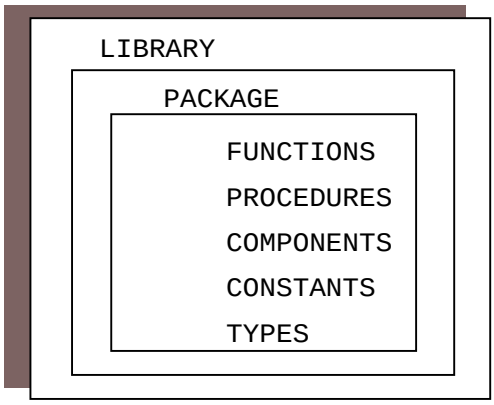
ดังรูปที่ 1.1 เป็นลักษณะองค์ประกอบของภาษา VHDL โดยแต่ละส่วนจะแตกต่างกัน ซึ่งสามารถอธิบายได้ดังนี้



รูปที่ 1.1 องค์ประกอบพื้นฐานของภาษา VHDL

1.1.1 โครงสร้างการเขียนภาษา VHDL ในส่วนของ LIBRARY

ในส่วนของ LIBRARY นี้จะทำหน้าที่เก็บ package ต่างๆไว้ โดยในแต่ละ package จะประกอบไปด้วยโปรแกรมย่อย (subprogram) ต่างๆและ package เหล่านี้สามารถนำไปใช้ในส่วนของ ENTITY และ ARCHITECTURE หรือ package ในชุดอื่นๆ ดังแสดงในรูปที่ 1.2



รูปที่ 1.2 องค์ประกอบของ LIBRARY

ในการประกาศการใช้งาน LIBRARY จะมีอยู่สองบรรทัดด้วยกันคือ

- บรรทัดแรกจะเป็นการบอกชื่อ LIBRARY ที่จะต้องการใช้ และ
- บรรทัดที่สองจะเป็นการระบุชื่อการใช้ package ที่ถูกเก็บไว้ใน LIBRARY นั้นๆ โดยใช้คำสั่ง USE ดังรูปที่ 1.3

```
1   LIBRARY library_name;
2   USE library_name.package_name.package_parts;
```

รูปที่ 1.3 รูปแบบการเขียนในส่วนของ LIBRARY

คำสั่ง USE เป็นชุดคำสั่งที่ทำหน้าที่ระบุการใช้งานขององค์ประกอบย่อยที่ถูกประกาศไว้ภายใน package นั้นๆ ซึ่งอาจจะเป็นชนิด TYPES CONSTANT SIGNAL FUNCTION หรือ PROCEDURE ก็ได้ เมื่อสิ้นสุดการใช้คำสั่ง USE จะต้องปิดท้ายด้วยเครื่องหมายอัฒภาค (;) ดังแสดงในรูปที่ 1.3

การเขียนโปรแกรม VHDL นั้น package ที่ถูกนำไปใช้งานบ่อยๆจะมีอยู่ด้วยกันสองชุด คือ

ชื่อ package	รายะเอียด
standard	ถูกเก็บไว้ใน LIBRARY ชื่อ std
std_logic_1164	ถูกเก็บไว้ใน LIBRARY ชื่อ ieee

จาก package ทั้งสองชุดนี้เมื่อนำมาเขียนในรูปแบบชุดคำสั่งของ LIBRARY จะสามารถเขียนได้ดังรูปที่ 1.4

```
1  LIBRARY std;
2  USE std.standard.all;
```

(ก)

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
```

(ข)

รูปที่ 1.4 (ก) การเรียกใช้ package ชื่อ standard จาก LIBRARY std
(ข) การเรียกใช้ package ชื่อ std_logic_1164 จาก LIBRARY ieee

การใช้งาน package ที่ชื่อ std สามารถเรียกใช้งานองค์ประกอบย่อยภายในได้ โดยไม่ต้องเขียนประกาศไว้ เนื่องจากเป็น package แฝง

1.1.2 โครงสร้างการเขียนภาษา VHDL ในส่วนของ ENTITY

เป็นส่วนมีไว้เพื่อสำหรับระบุช่องทางในการติดต่อ ระหว่างอุปกรณ์หรือวงจรที่จะสร้างขึ้นกับอุปกรณ์ภายนอก ถ้าพูดง่ายๆ ก็คือพอร์ตติดต่อนั่นเอง รูปแบบการเขียนของ ENTITY มีรูปแบบดังแสดงไว้ในรูปที่ 1.5

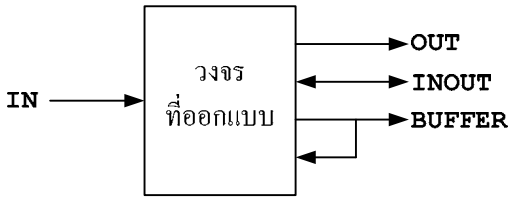
```
1   ENTITY entity_name IS
2       PORT (
3             port_name : signal_mode signal_type ;
4             port_name : signal_mode signal_type ;
5             ...
6             );
7   END entity_name;
```

รูปที่ 1.5 รูปแบบการเขียนชุดคำสั่งของ ENTITY

เมื่อ signal_mode จะเป็นตัวกำหนดทิศทางของสัญญาณ ซึ่งจะมีอยู่ด้วยกัน 4 ชนิดคือ IN OUT INOUT และ BUFFER ดังรูปที่ 1.6

signal_type จะเป็นบ่งบอกถึงประเภทของสัญญาณ ซึ่งมีหลากหลายประเภท บางประเภทสามารถใช้งานร่วมกันได้ บางประเภทก็ไม่สามารถใช้ร่วมกันได้ เช่น signal_type ประเภท BIT BOOLEAN STD_LOGIC INTEGER เป็นต้น ประเภทของสัญญาณนี้จะอธิบายรายละเอียดไว้ในบทที่ 2

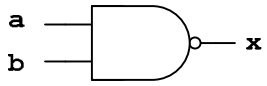
สุดท้าย การตั้งชื่อ entity_name และ port_name จะต้องเป็นไปตามกฎการตั้งชื่อในภาษา VHDL ดังแสดงไว้ในภาคผนวก ก.



IN เป็นตัวรับสัญญาณจากภายนอกเข้ามา
OUT เป็นตัวส่งสัญญาณออกไปภายนอก
INOUT เป็นตัวรับและส่งสัญญาณสองทิศทาง
BUFFER เป็นที่พักของสัญญาณ

รูปที่ 1.6 ลักษณะโครงสร้างของส่วน ENTITY

จากรูปที่ 1.7(ก) เป็นเกตแนนด์ เมื่อนำมาเขียนเป็นภาษา VHDL ในส่วนของ ENTITY จะต้องทราบว่ามีอินพุทและเอาท์พุทจำนวนเท่าใดและชื่ออะไร จากรูปเกตแนนด์นั้นมีอินพุท 2 ขา ชื่อ a และ b ส่วนเอาท์พุทมีขาเดียวชื่อ y ดังนั้นเราก็สามารถเขียนในส่วนของ ENTITY ได้ดังรูปที่ 1.7(ข)



(ก)

```
1   ENTITY nand_gate IS
2       PORT( a,b : IN BIT ;
3             x   : OUT BIT );
4   END nand_gate;
```

(ข)

รูปที่ 1.7 (ก) สัญลักษณ์แนนด์เกต 2 อินพุท
(ข) การกำหนดทิศทางเข้าและออกสัญญาณของแนนด์เกต

1.1.3 โครงสร้างการเขียนภาษา VHDL ในส่วนของ ARCHITECTURE

ในส่วน ARCHITECTURE จะเป็นส่วนที่มีไว้สำหรับเขียนบรรยายหรือกำหนดพฤติกรรมการทำงานของวงจรดิจิตอลที่ต้องการออกแบบใช้งาน โดยที่พฤติกรรมของวงจรที่เขียนขึ้นจะต้องสัมพันธ์กับทิศทางรูปแบบของสัญญาณ (signal_mode) ที่กำหนดไว้ในส่วนของ ENTITY ซึ่งในส่วนของ ARCHITECTURE มีรูปแบบในการเขียน ดังแสดงไว้ในรูปที่ 1.8

```
1   ARCHITECTURE architecture_name OF entity_name IS
2     [declarations option]
3   BEGIN
4     [ code ]
5   END architecture_name;
```

รูปที่ 1.8 รูปแบบการเขียนในส่วนของ ARCHITURE

จากรูปที่ 1.8 ส่วนที่อยู่ระหว่าง ARCHITECTURE และ BEGIN คือ declarations option เป็นส่วนเผื่อเลือกที่ประกาศไว้ สำหรับกำหนดค่าต่าง ๆที่จะนำไปใช้ภายใน ARCHITURE (โดยไม่ได้กำหนดไว้ในส่วนของ ENTITY) อาทิเช่น SIGNAL CONSTANT COMPONENT เป็นต้น และส่วนที่อยู่ระหว่าง BEGIN และ END ก็คือ code หรือชุดคำสั่งที่มีไว้สำหรับเขียนบรรยายหรือกำหนดพฤติกรรมการทำงานของวงจรดิจิตอลที่ต้องการออกแบบใช้งาน

จากรูปที่ 1.7(ก) เราสามารถเขียนบรรยายพฤติกรรมการทำงานของแนนด์เกต ไว้ในส่วนของ ARCHITECTURE ได้ดังรูปที่ 1.9

```
1    ARCHITECTURE behav OF nand_gate IS
2     BEGIN
3            y <= a nand b;
4    END behav;
```

รูปที่ 1.9 การเขียนบรรยายพฤติกรรมการทำงานของแนนด์เกต

จากรูปที่ 1.9 ในบรรทัดที่ 3 สัญญาณอินพุท a และสัญญาณอินพุท b จะกระทำการในลักษณะแบบแนนด์กัน แล้วผลลัพธ์ของการแนนด์จะถูกส่งผ่านค่า (โดยใช้เครื่องหมาย '<=') ไปยังสัญญาณเอาท์พุท y การส่ง ผ่านค่านั้นจะต้องเป็นประเภทข้อมูลเดียวกัน (จากรูปที่ 1.7(ข)นั้น อินพุท a อินพุท b และเอาท์พุท y เป็นข้อมูลประเภท BIT เหมือนกัน ดังนั้นการส่งผ่านค่าจึงสามารถกระทำได้)

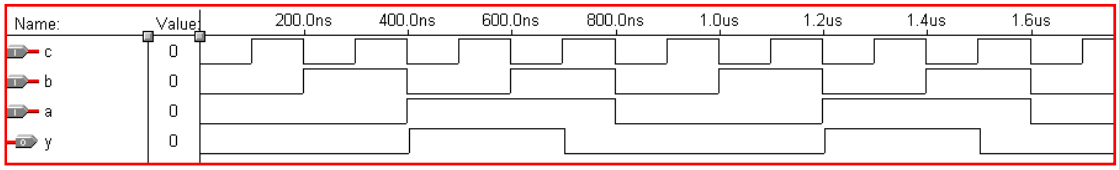
1.2 ตัวอย่างการเขียนโปรแกรมภาษา VHDL

<u>ตัวอย่างที่ 1.1</u> จากสมการพีชคณิตบูลีน y = ab'c + ac' ต้องการเขียนให้อยู่ในรูปแบบของภาษา VHDL

คำอธิบาย ก่อนอื่นต้องพิจารณาจากสมการพีชคณิต มีอินพุทจำนวน 3 ตัวคือตัวแปร a b และ c ส่วนเอาท์พุทมีตัวเดียวคือตัวแปร y ซึ่งสามารถเขียนในส่วนของ ENTITY ได้ตั้งแต่บรรทัดที่ 4 ถึง 7 และในส่วนของ ARCHITECTURE นั้นก็จะเขียนตามสมการพีชคณิตบูลีน ซึ่งจะได้ตั้งแต่บรรทัดที่ 9 ถึง 12 ส่วนบรรทัดที่ 1 และ 2 เป็นการเรียกใช้ package ชื่อ std_logic_1164 จาก LIBRARY ชื่อ ieee

```
1    LIBRARY ieee;
2    USE ieee.std_logic_1164.all;
3    ---------------------------------
4    ENTITY ex1 IS
5        PORT ( a,b,c  : IN BIT ;
6                   y   : OUT BIT );
7    END ex1;
8    --------------------------------
9    ARCHITECTURE behav OF ex1 IS
10   BEGIN
11           y <= (a and not(b) and c) or (a and not(c));
12   END behav;
```

เมื่อเขียนโปรแกรมภาษาบรรยายพฤติกรรมแล้วเสร็จ ต้องทำการบันทึกชื่อแฟ้มข้อมูลเป็นชื่อเดียวกับชื่อ entity_name ตามด้วยนามสกุลเป็น .vhd ซึ่งจะได้เป็น ex1.vhd หลังจากนั้น สามารถที่นำแฟ้มข้อมูลไปทำการจำลองดูรูปสัญญาณโดยใช้โปรแกรมช่วยต่าง ๆเช่น Quartus-II หรือ ModelSim ในที่นี้ผู้เขียนได้ใช้โปรแกรม Quartus-II ทำให้ได้รูปสัญญาณอินพุทและเอาท์พุท ดังแสดงในรูปที่ 1.10 เมื่ออินพุท a เป็นบิตสูงสุด (MSB) และอินพุท c เป็นบิตต่ำสุด(LSB)



รูปที่ 1.10 ผลการจำลองรูปคลื่นสัญญาณ ตัวอย่างที่ 1.1

<u>ตัวอย่างที่ 1.2</u> จากสมการพีชคณิตบูลีน y = ab’ + a’b และ z = (ab+bc)’ ให้เขียนอยู่ในรูปแบบของภาษา VHDL

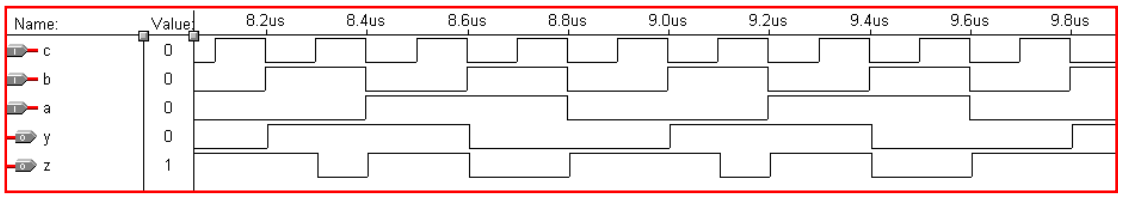
คำอธิบาย จากพิจารณาจากสมการพีชคณิตทั้ง 2 สมการ มีอินพุทจำนวน 3 ตัวคือ a b และ c ส่วนเอาท์พุทมีสองตัวคือตัวแปร y และ z สามารถเขียนเป็นโปรแกรมภาษา VHDL ได้เป็น

```
1    LIBRARY ieee;
2    USE ieee.std_logic_1164.all;
3    ---------------------------------
4    ENTITY ex2a IS
5    PORT ( a,b,c   : IN BIT ;
6           y,z     : OUT BIT );
7    END ex2a;
8    ---------------------------------
9    ARCHITECTURE behav OF ex2a IS
10    BEGIN
11      y <= (a and not(b)) or (not(a) and b);
12      z <= not ((a and b) or (b and c));
13   END behav;
```

หรือในบางครั้งเราสามารถที่จะเขียนสมการพีชคณิตของตัวแปร y ใหม่ เป็น

```
1    LIBRARY ieee;
2    USE ieee.std_logic_1164.all;
3    ---------------------------------
4    ENTITY ex2b IS
5    PORT ( a,b,c   : IN BIT ;
6           y,z     : OUT BIT );
7    END ex2b;
8    ---------------------------------
9    ARCHITECTURE behav OF ex2b IS
10    BEGIN
11      y <= a xor b;
12      z <= not ((a and b) or (b and c));
13   END behav;
```

เมื่อนำแฟ้มข้อมูลไปทำการจำลองดูรูปสัญญาณ จะทำให้ได้รูปสัญญาณอินพุทและเอาท์พุท ดังแสดงในรูปที่ 1.11 เช่นกัน



รูปที่ 1.11 ผลการจำลองรูปคลื่นสัญญาณ ตัวอย่างที่ 1.2

1.3 โจทย์ปัญหา

1. จากสมการพีชคณิตต่อไปนี้ จงเขียนให้อยู่ในรูปแบบของภาษา VHDL

   1.1) $Z = A + \overline{BC}$

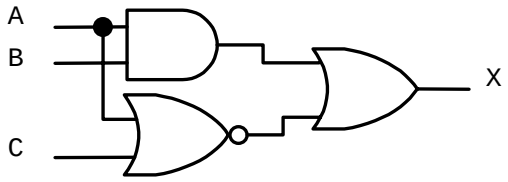
   1.2) $Y = \overline{A_1}A_2(\overline{A_1 + A_3})$

   1.3) $OUT = \overline{\overline{A}B + \overline{B}C + A\overline{C} + A\overline{B}C}$

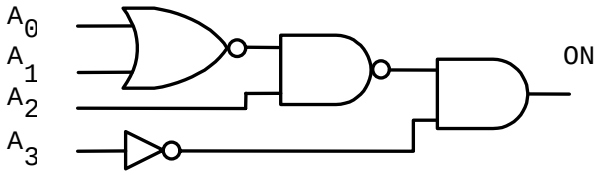
   1.4). $S = (\overline{\overline{W}_1 + W_2})(\overline{W_3 + \overline{W}_4})$

2. จากรูปวงจรลอจิกให้เขียนให้อยู่ในรูปแบบของภาษา VHDL

   2.1).



   2.2).



3. ถ้ากำหนดให้

- ดวงอาทิตย์สาดแสงตอนกลางวัน มีลอกจิก ‘0’ ส่วนตอนกลางคืนมีลอจิก ‘1’
- สายฝน เมื่อมีฝนตก มีลอจิก ‘1’ และเมื่อไม่มีฝน(ฝนไม่ตก) มีลอจิก ‘0’
- สายรุ้งจะปรากฏให้เห็น หรือมีลอจิก ‘1’ เมื่อมีดวงอาทิตย์สาดแสงและฝนตกเกิดขึ้นเท่านั้น

ให้เขียนบรรยายการทำงานของสายรุ้งอยู่ในรูปแบบของภาษา VHDL

4. จาก $Z = f(A, B, C) = \sum m(0, 2, 6, 7)$ ให้เขียนอยู่ในรูปแบบของภาษา VHDL

5. จาก $X = f(A, B, C, D) = \sum m(1, 5, 8, 10)$ให้เขียนอยู่ในรูปแบบของภาษา VHDL

6. จาก $Y = f(A, B, C) = \prod M(0, 1, 3, 4)$ ให้เขียนอยู่ในรูปแบบของภาษา VHDL

2 ประเภทของข้อมูล

Data Types

ในการเขียนโปรแกรมภาษาคอมพิวเตอร์ นอกจากจะทราบถึงโครงสร้างของภาษาแล้ว ยังจำเป็นที่จะต้องรู้ถึงประเภทของข้อมูลด้วย ภาษา VHDL ก็เช่นกันที่ผู้เรียนรู้จำเป็นจะต้องเรียนรู้ถึงประเภทของข้อมูลที่ถูกกำหนดไว้แล้วจากตัวของภาษา และผู้ใช้กำหนดขึ้นมาเอง พร้อมทั้งฟังก์ชั่นพิเศษที่มีมาพร้อมในตัวภาษา

ดังนั้นเนื้อหาต่อไปนี้จะได้กล่าวถึงประเภทของข้อมูลที่ถูกนำมาใช้งานบ่อยๆ

2.1 ประเภทของข้อมูลที่ถูกกำหนดไว้แล้ว (Pre-Defined Data Type)

กลุ่มประเภทข้อมูลนี้เป็นประเภทของข้อมูลขั้นพื้นฐานที่ถูกกำหนดไว้แล้ว ดังแสดงในภาคผนวก ข. ดังนั้นเมื่อต้องการนำไปใช้งานจะต้องมีการบอกถึงชื่อของ package และที่เก็บของ package หรือ LIBRARY ดังตารางที่ 2.1

ตารางที่ 2.1 รายละเอียดของ package ที่ถูกกำหนดไว้แล้ว

LIBRARY	Package	รายละเอียด
ieee	std_logic_1164	เป็น package มาตรฐานสำหรับภาษา VHDL ที่มีประเภทข้อมูล o STD_LOGIC o STD_LOGIC_VECTOR
	std_logic_arith	เป็น package ที่มีประเภทข้อมูล o SIGNED o UNSIGNED o ฟังก์ชันการแปลงค่าข้อมูล เช่น • conv_integer(p) • conv_unsinged(p,b) • conv_signed(p,b) • conv_std_logic_vector(p,b) o การกระทำทางคณิตศาสตร์และการเปรียบเทียบสำหรับ • SIGNED • UNSIGNED
	std_logic_unsigned	เป็น package ที่อนุญาตให้ใช้ประเภทข้อมูล o STD_LOGIC_VECTOR ถ้าหากมีการกำหนดประเภทของข้อมูลเป็น UNSIGNED
	std_logic_signed	เป็น package ที่อนุญาตให้ใช้ประเภทข้อมูล o STD_LOGIC_VECTOR ถ้าหากมีการกำหนดประเภทของข้อมูลเป็น SIGNED

ตารางที่ 2.1 รายละเอียดของ package ที่ถูกกำหนดไว้แล้ว(ต่อ)

LIBRARY	Package	รายละเอียด
ieee	numeric_std	เป็น package ที่ใช้สำหรับการกระทำทางคณิตศาสตร์สำหรับ ที่มีการอ้างถึงในหลาย package เช่น std_logic_arith std_logic_unsigned std_logic_signed
work	กำหนดจากผู้ใช้	เป็น package ที่กำหนดขึ้นโดยผู้ใช้

ตัวอย่างและรายละเอียดของกลุ่มของประเภทข้อมูลที่ถูกกำหนดไว้แล้ว

1. **BOOLEAN** เป็นประเภทของข้อมูลที่ให้ค่าเป็น FALSE และ TRUE

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_boolean IS
2     BEGIN
3         PROCESS(...)
4          BEGIN
5            IF resetn = ‘0’ THEN
6               ...
7            ELSE
8               ...
9            END IF;
10        END PROCESS;
11   END behav;
```

o เป็นการตรวจเงื่อนไขว่าตัวแปร resetn มีค่าเท่ากับ 0 หรือไม่ ถ้าเป็นจริงจะทำตามคำสั่งในบรรทัดที่ 6 แต่ถ้าไม่ใช่หรือเป็นเท็จ จะทำตามคำสั่งในบรรทัดที่ 8

```
1    ARCHITECTURE behav OF ex_boolean2 IS
2    ...
3     BEGIN
4         PROCESS(...)
5          BEGIN
6            IF a >= b THEN
7               ...
8            ELSE
9               ...
10           END IF;
11        END PROCESS;
12   END behav;
```

o เป็นการตรวจเงื่อนไขระหว่างตัวแปร a และ b ถ้าตัวแปร a มีค่ามากกว่าหรือค่าเท่ากับ b เป็นจริงจะทำตามคำสั่งในบรรทัดที่ 7 แต่ถ้าไม่ใช่หรือเป็นเท็จ จะทำตามคำสั่งในบรรทัดที่ 9 โดยทั้งตัวแปร a และ b จะต้องประเภทของข้อมูลกลุ่มเดียวกัน

2. **BIT** เป็นประเภทของข้อมูลที่ให้ค่าเป็น ‘0’ และ ‘1’ เท่านั้น

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_bit IS
2    SIGNAL a,b : BIT;
3     BEGIN
4         PROCESS(...)
5          BEGIN
6            IF a >= b THEN
7               ...
8            ELSE
9               ...
10           END IF;
11        END PROCESS;
12   END behav;
```

o เป็นการกำหนดให้ตัวแปร a และ b เป็นตัวแปรประเภทข้อมูลแบบ BIT ที่มีขนาด 1 บิต

3. **BIT_VECTOR** เป็นกลุ่มของประเภทข้อมูลแบบ BIT ที่มีค่ามากกว่าหนึ่งบิตขึ้นไป

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_bit_vector IS
2    SIGNAL a : BIT_VECTOR(3 DOWNTO 0);
3    SIGNAL b : BIT_VECTOR(0 TO 7);
4     BEGIN
5            a <= “0110”;
6            b <= “01110000”;
7               ...
8    END behav;
```

o เป็นการกำหนดให้ตัวแปร a เป็นตัวแปรประเภทข้อมูลแบบ BIT_VECTOR มีขนาด 4 บิต โดยบิตสูงสุดอยู่ทางด้านซ้ายมือ และกำหนดให้ตัวแปร b เป็นตัวแปรประเภทข้อมูลแบบ BIT_VECTOR มีขนาด 8 แต่บิตสูงสุดอยู่ทางด้านขวามือ ในบรรทัดที่ 5 ตัวแปร a จะมีค่าเท่ากับ 0110_2 บรรทัดที่ 6 ตัวแปร b จะมีค่าเท่ากับ 00001110_2

<u>ข้อสังเกต</u> การเขียนข้อมูลขนาด 1 บิต จะใช้เครื่องหมายอัญประกาศครอบ (‘ ’) ส่วนการเขียนข้อมูลที่ขนาดมากกว่า 1 บิต จะใช้เครื่องหมายอัญประกาศสองตัวครอบ (“ ”)

4. **INTEGER** เป็นประเภทของข้อมูลที่ให้ค่าตัวเลขจำนวนต็มขนาด 32 บิต มีค่าอยู่ระหว่าง -2,147,483,647 ถึง +2,147,483,647

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_integer IS
2    SIGNAL a,b : INTEGER;
3     BEGIN
4            a <= 200;
5            b <= -100;
6               ...
7    END behav;
```

- เป็นการกำหนดตัวแปร a ให้มีค่าเท่ากับ 200 และตัวแปร b ให้ค่าเท่ากับ -100 โดยที่ตัวแปร a และ b สามารถรับค่าที่อยู่ในช่วงระหว่าง -2,147,483,647 ถึง +2,147,483,647

บางครั้งในการกำหนดค่าให้กับตัวแปรนั้นสามารถกำหนดให้เป็นเลขฐานสอง ฐานแปด ฐานสิบหก และฐานสิบก็ได้ ดังตัวอย่างต่อไปนี้

```
1    a_vect <= “1100_0011_0011_1100”;   --ระบุเลขฐานสอง หรือ
2    a_vect <= “1100001100111100”;      --ระบุเลขฐานสอง หรือ
3    a_vect <= X“C33C”;                 --ระบุเลขฐานสิบหก หรือ
4    a_vect <= 49980;                   --ระบุเลขฐานสิบ หรือ
5  a_vect <= X“141474”;                 --ระบุเลขฐานแปด
```

5. **NATURAL** เป็นประเภทของข้อมูลที่ให้ค่าตัวเลขจำนวนเต็มบวกเท่านั้น มีค่าอยู่ระหว่าง **0** ถึง 2,147,483,647

```
c <= 1234;
```

- ตัวแปร c จะถูกกำหนดให้มีค่าเท่ากับ 1234 โดยตัวแปร c เป็นประเภทข้อมูลแบบ NATURAL หรือ INTEGER ก็ได้

6. **REAL** เป็นประเภทของข้อมูลที่ให้ค่าที่อยู่ระหว่าง -1.0E38 ถึง +1.0E38

```
y <= 1.25E-5;
```

- ตัวแปร y จะถูกกำหนดให้มีค่าเท่ากับ 1.25E-5 แต่ค่าประเภทนี้ไม่สามารถนำไปสังเคราะห์ใช้งานได้

7. **STD_LOGIC** และ **STD_LOGIC_VECTOR** เป็นประเภทของข้อมูลที่ให้ค่าด้วยกัน 8 ค่า คือ (‘X’ ‘0’ ‘1’ ‘Z’ ‘W’ ‘L’ ‘H’ ‘-’)

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_std_logic IS
2    SIGNAL x : STD_LOGIC;
3    SIGNAL y : STD_LOGIC_VECTOR(3 DOWNTO 0):= “0101”;
4     BEGIN
5                ...
6    END behav;
```

- ตัวแปร x จะถูกกำหนดให้มีขนาดหนึ่งบิต เป็นประเภทข้อมูลแบบ STD_LOGIC ส่วนตัวแปร y จะถูกกำหนดให้มีขนาด 4 บิต โดยมีค่าเริ่มต้นเท่ากับ **“0101”** แต่เป็นประเภทข้อมูลแบบ STD_LOGIC_VECTOR โดยบิตสูงสุดอยู่ทางด้านซ้ายมือ

8. STD_ULOGIC และ STD_ULOGIC_VECTOR เป็นประเภทของข้อมูลที่ให้ค่าด้วยกัน 9 ค่า คือ (‘U’ ‘X’ ‘0’ ‘1’ ‘Z’ ‘W’ ‘L’ ‘H’ ‘-’)

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_std_logic IS
2    SIGNAL x : STD_ULOGIC;
3    SIGNAL y : STD_ULOGIC_VECTOR(3 DOWNTO 0):= “0101”;
4     BEGIN
5               ...
6    END behav;
```

- ตัวแปร x จะถูกกำหนดให้มีขนาดหนึ่งบิต เป็นประเภทข้อมูลแบบ STD_ULOGIC ส่วนตัวแปร y จะถูกกำหนดให้มีขนาด 4 บิต เป็นประเภทข้อมูลแบบ STD_ULOGIC โดยมีค่าเริ่มต้นเท่ากับ “0101”

9. TIME เป็นประเภทของข้อมูลที่ให้ค่าเป็นหน่วยเวลาที่มีค่าพื้นฐาน

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_time IS
2    ...
3     BEGIN
4       c <= a AFTER 10 ns WHEN sel_0 = ‘0’ ELSE
5             b AFTER 10 ns;
6    END behav;
```

- ตัวแปร c จะเท่ากับตัวแปร a เมื่อเวลาผ่านไป 10 นาโนวินาที หลังจากตัวแปร sel_0 มีค่าเท่ากับ ‘0’ และเท่ากับตัวแปร b เมื่อเวลาผ่านไป 10 นาโนวินาที หลังจากตัวแปร sel_0 มีค่าเท่ากับ ‘1’

10. CHARACTER เป็นประเภทของข้อมูลที่ให้ค่าเป็น พยัญชนะอักษร เครื่องหมายพิเศษ และอักษรพิเศษ

2.2 ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง (User-Defined Data Type)

ประเภทของข้อมูลที่ผู้ใช้กำหนดเองจะมีอยู่ด้วยกัน 2 แบบคือ integer และ enumerated การใช้งานจะต้องใช้ร่วมกับชุดคำสั่ง TYPE ประกอบด้วยเสมอ

2.2.1 ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง แบบ integer

ข้อมูลที่ผู้ใช้กำหนดในกลุ่มนี้ จะเป็นกลุ่มของตัวเลขที่ให้ค่าเป็นจำนวนเต็ม ที่มีค่าอยู่ในช่วง -2147483647 ถึง +2147483647 ซึ่งสามารถกำหนดช่วงในการใช้งานตามความเหมาะสม เช่น

```
1    ARCHITECTURE behav OF ex_integer IS
2    TYPE my_integer IS RANGE 0 to 255;
3    SIGNAL q : my_integer;
4     BEGIN
5               ...
6    END behav;
```

- ประเภทของข้อมูลชื่อ my_integer มีการกำหนดค่าใช้งานอยู่ระหว่าง 0 ถึง 255 ซึ่งตัวแปร q นี้เป็นข้อมูลย่อย(subtype) ของ my_integer อีกทีหนึ่ง

2.2.2 ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง แบบ enumerated

ข้อมูลที่ผู้ใช้กำหนดในกลุ่มนี้ จะเป็นลักษณะของกลุ่มคำหรือค่าที่ผู้ใช้ทำการแจกแจงไว้ เช่น

```
1    ARCHITECTURE behav OF ex_enum IS
2    TYPE state IS (idle,forward,backward,start,stop);
3    SIGNAL id : state;
4     BEGIN
5                ...
6    END behav;
```

- ประเภทของข้อมูลนี้ชื่อ state เป็นลักษณะของกลุ่มคำที่แสดงถึงสถานะของวงจร ซึ่งสามารถแจกแจงสถานะได้เป็น idle forward backward start และ stop

2.3 ข้อมูลย่อย (Subtypes)

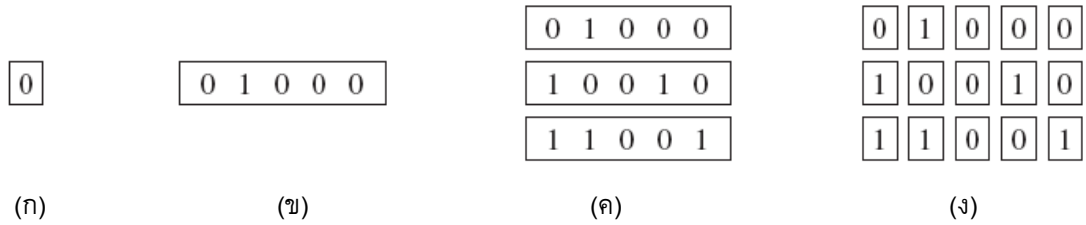
ในส่วนของ ประเภทของข้อมูลที่กำหนดไว้แล้วและ ผู้ใช้กำหนดขึ้นเอง สามารถที่จะแบ่งออกเป็นกลุ่มย่อยลงไปได้อีก ที่เรียกว่า ข้อมูลย่อยหรือ Subtype

```
1    ARCHITECTURE behav OF ex_sub IS
2    TYPE color IS (red,green,blue,white);
3    SUBTYPE my_color IS color RANGE red TO blue;
4    SIGNAL shirt_id : my_color;
5     BEGIN
6                ...
7    END behav;
```

- ข้อมูลย่อยชื่อ my_color จะทำการอ้างอิงค่าจาก ข้อมูลหลักที่มีชื่อว่า color ซึ่งมีค่าเป็น red green blue และ white แต่ข้อมูลย่อย my_color นี้มีการระบุค่าใช้งานที่อยู่ในช่วง red green และ blue เท่านั้น

2.4 ข้อมูลอะเรย์ (Array)

ข้อมูลอะเรย์เกิดขึ้นจากการนำเอาข้อมูลประเภทเดียวกัน มารวมเป็นกลุ่มของข้อมูล เพื่อประโยชน์ในการนำไปใช้งาน ซึ่งข้อมูลอะเรย์นี้จะมีทั้งที่เป็นกลุ่มข้อมูลอะเรย์หนึ่งมิติ (หนึ่งแถวหรือหนึ่งหลัก) ข้อมูลอะเรย์สองมิติ (เกิดจากหนึ่งแถวคูณด้วยหนึ่งหลัก) หรืออาจจะมีหลายๆมิติก็ได้ จากรูปที่ 2.1 จะเห็นว่ารูป(ก) เป็นข้อมูล scalar จำนวนหนึ่งค่า รูป(ข) เป็นข้อมูลขนาดหนึ่งมิติ รูป(ค) เป็นข้อมูลขนาดสองมิติ และสุดท้าย รูป(ง) เป็นข้อมูลอะเรย์แบบสองมิติเหมือนกัน เพียงแต่แตกต่างกันตรงที่เป็นการนำเอาข้อมูล scalar หลายๆค่ามาวางเรียงกัน



รูปที่ 2.1 ประเภทของข้อมูลอะเรย์

จากหัวข้อ 2.1 ข้อมูลที่เป็นสกาล่าร์ (scalar) จะได้แก่ข้อมูลประเภท

- BIT
- STD_LOGIC
- STD_ULOGIC
- BOOLEAN

ส่วนข้อมูลที่เป็นอะเรย์หรือเวกเตอร์(vector)ได้แก่

- BIT_VECTOR
- STD_LOGIC_VECTOR
- STD_ULOGIC
- SIGNED
- UNSIGNED
- INTEGER

ในการนำข้อมูลอะเรย์ไปใช้งาน เช่นข้อมูลสองมิตินั่นผู้ใช้จะต้องกำหนดขึ้นมาเอง โดยใช้คำสั่ง TYPE เป็นตัวกำหนด และในกรณีที่ต้องใช้ข้อมูลเวกเตอร์ใหม่ ก็ต้องกำหนดประเภทของข้อมูลพร้อมด้วยประเภทของตัวแปรสัญญาณ อันได้แก่ SIGNAL และ VARIABLE นั่นเอง (อธิบายไว้ในหัวข้อที่ 5.2) ลักษณะตัวอย่างการใช้ข้อมูลอะเรย์

```
1   ARCHITECTURE behav OF ex_array IS
2   TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
3   TYPE matrix IS ARRAY (0 TO 3) OF row;
4   SIGNAL x: matrix;
5    BEGIN
6   ...
```

o ตัวแปร row เป็นประเภทข้อมูลอะเรย์ขนาดหนึ่งมิติ
ตัวแปร matrix เป็นประเภทข้อมูลอะเรย์ขนาดสองมิติ
ตัวแปร x เป็นข้อมูลอะเรย์ขนาด หนึ่งมิติคูณหนึ่งมิติ โดยใช้ตัวแปรสัญญาณประเภท SIGNAL

```
1   ARCHITECTURE behav OF ex_array2 IS
2   TYPE mat IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR
                (7 DOWNTO 0);
3    BEGIN
4   ...
```

- o ตัวแปร mat เป็นข้อมูลอะเรย์ขนาด หนึ่งมิติคูณหนึ่งมิติ โดยทำการระบุหรือกำหนดมาจากประเภทของข้อมูลที่ถูกกำหนดไว้ ในหัวข้อที่ 2.1

⇨ การกำหนดค่าเริ่มต้นการใช้งานข้อมูลอะเรย์ สามารถกระทำได้ใช้ลักษณะการส่งผ่านค่าโดยใช้เครื่องหมาย **:=** หลังค่าตัวแปร

```
1   ARCHITECTURE behav OF ex_array3 IS
2   TYPE row1 IS ARRAY(3 DOWNTO 0)OF STD_LOGIC := “0001”;
3   TYPE row2 IS ARRAY(3 DOWNTO 0)OF STD_LOGIC :=
                              (‘0’,’0’,’0’,’1’);
4    BEGIN
5   ...
```

- o ตัวแปร row1 และ row2 เป็นประเภทข้อมูลอะเรย์ขนาดหนึ่งมิติ ที่มีค่าเริ่มต้นเป็น “0001” เหมือนกัน แต่มีลักษณะการเขียนที่แตกต่างกัน

2.5 ข้อมูลเรคอร์ด (Record)

ข้อมูลเรคอร์ด เป็นข้อมูลที่มีลักษณะคล้ายกับข้อมูลอะเรย์ แต่ภายในของเรคอร์ดสามารถที่จะมีประเภทของข้อมูลที่แตกต่างกันได้ดังตัวอย่างต่อไปนี้

```
1   ARCHITECTURE behav OF ex_rec IS
2      TYPE birthday IS RECORD
3        day: INTEGER RANGE 1 TO 31;
4        month: month_name;
5      END RECORD;
6    BEGIN
7   ...
```

- o ตัวแปร birthday เป็นประเภทข้อมูลเรคอร์ด ซึ่งภายในประกอบด้วยตัวแปร day เป็นประเภทข้อมูล INTEGER และตัวแปร month เป็นประเภทข้อมูล month_name (ประเภทของข้อมูลที่ผู้ใช้กำหนดเอง)

2.6 ประเภทข้อมูลที่คิดเครื่องหมาย (signed) และ ไม่คิดเครื่องหมาย (unsigned)

ประเภทของข้อมูลที่คิดเครื่องหมายและไม่คิดเครื่องหมาย เป็นประเภทข้อมูลที่จะใช้กับชุดคำสั่งที่กระทำกันทางคณิตศาสตร์ ลักษณะของประเภทข้อมูลที่คิดเครื่องหมายนั้น จะพิจารณาข้อมูลตัวทุกๆตัว ยกเว้นบิตสูงสุดของข้อมูล ซึ่งบิตสูงสุดนี้จะเป็นตัวบ่งบอกลักษณะค่าของตัวเลข ถ้าบิตสูงสุดมีค่าเป็น ‘1’ นั่นก็แสดงว่ากลุ่มตัวเลขนั้นๆมีค่าเป็นลบ และในทางตรงกันข้ามถ้าเป็น ‘0’ กลุ่มตัวเลขนั้นๆมีค่าเป็นบวก แต่ถ้าเป็นประเภทข้อมูลที่ไม่คิดเครื่องหมาย จะพิจารณาข้อมูลทุกบิต ซึ่งสามารถอธิบายได้ดังนี้ คือ

ถ้าข้อมูลมีค่า "0101" ข้อมูลนี้เมื่อคิดเครื่องหมายและไม่คิดเครื่องหมายจะค่าเท่ากัน คือ เท่ากับ $5_{(10)}$ ในทำนองเดียวถ้าข้อมูลมีค่า "1101" ข้อมูลนี้เมื่อคิดเครื่องหมายจะมีค่าเท่ากับ $-3_{(10)}$ (ต้องทำการคอมพลีเมนท์สองก่อน) และ ข้อมูลนี้เมื่อไม่คิดเครื่องหมายจะมีค่าเท่ากับ $13_{(10)}$

ลักษณะของการนำไปใช้งานของประเภทของข้อมูลที่คิดเครื่องหมายและไม่คิดเครื่องหมาย เมื่อต้องการใช้กับชุดคำสั่งที่กระทำกันทางคณิตศาสตร์ จะต้องมีการเรียกใช้ package ชื่อ std_logic_arith จาก LIBRARY ieee เสมอ ดังแสดงในตัวอย่าง

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_arith.all;
4   ...
5   ARCHITECTURE behav OF ex_sig1 IS
6   SIGNAL a: SIGNED (7 DOWNTO 0);
7   SIGNAL b: SIGNED (7 DOWNTO 0);
8   SIGNAL x: SIGNED (7 DOWNTO 0);
9   ...
10  BEGIN
11      v <= a + b;        -- ใช้งานได้
12      w <= a AND b;      -- ไม่สามารถใช้งานได้
13  END behav;
```

- จากข้อมูลตัวแปร a และ b เป็นประเภทของข้อมูลที่คิดเครื่องหมาย สามารถกระทำกันทางด้านคำสั่งคณิตศาสตร์ได้ แต่ไม่สามารถกระทำกันทางด้านตรรกะได้

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   ...
4   ARCHITECTURE behav OF ex_sig2 IS
5   SIGNAL a: STD_LOGIC_VECTOR (7 DOWNTO 0);
6   SIGNAL b: STD_LOGIC_VECTOR (7 DOWNTO 0);
7   SIGNAL x: STD_LOGIC_VECTOR (7 DOWNTO 0);
8   ...
9   BEGIN
10      v <= a + b;        -- ไม่สามารถใช้งานได้
11      w <= a AND b;      -- ใช้งานได้
12  END behav;
```

- จากข้อมูลตัว a และ b เป็นข้อมูลอะเรย์ นั่นก็แสดงว่าเป็นข้อมูลที่ไม่คิดเครื่องหมาย <u>ไม่สามารถกระทำกันทางด้านคำสั่งคณิตศาสตร์</u>ได้ เนื่องจากไม่มีการระบุการใช้งานของ package ทางด้าน คณิตศาสตร์แต่สามารถกระทำกันทางด้านตรรกะ

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_arith.all;
4   ...
5   ARCHITECTURE behav OF ex_sig3 IS
6   SIGNAL a: STD_LOGIC_VECTOR (7 DOWNTO 0);
7   SIGNAL b: STD_LOGIC_VECTOR (7 DOWNTO 0);
8   SIGNAL x: STD_LOGIC_VECTOR (7 DOWNTO 0);
9   ...
10  BEGIN
11      v <= a + b;        -- ใช้งานได้
12      w <= a AND b;      -- ใช้งานได้
13  END behav;
```

- จากข้อมูลตัวแปร a และ b เป็นข้อมูลอะเรย์ นั่นก็แสดงว่าเป็นข้อมูลที่ไม่คิดเครื่องหมาย สามารถกระทำกันทางด้านคำสั่งคณิตศาสตร์ได้ เนื่องจากมีการระบุการใช้งานของ package ทางด้าน คณิตศาสตร์ และสามารถกระทำกันทางด้านตรรกะ ได้ด้วย

2.7 การแปลงค่าข้อมูล (data conversion)

ขีดความสามารถของ VHDL เมื่อมีการกระทำกันไม่ว่าจะเป็น การกระทำทางด้านคณิตาสตร์ ทางด้านตรรกะ จะกระทำกันได้เฉพาะข้อมูลประเภทเดียวเท่านั้น ดังนั้นฟังก์ชันการแปลงค่าข้อมูลจึงจำเป็นอย่างยิ่งที่จะต้องมี และฟังก์ชั่นการแปลงค่าข้อมูลนั่นมีอยู่หลากหลายชนิดด้วยกัน อันได้แก่

- conv_integer(p)
- conv_unsigned(p,b)
- conv_signed(p,b)
- conv_std_logic_vector(p,b)

ตารางที่ 2.2 ฟังก์ชั่นของการแปลค่าข้อมูล

ฟังก์ชั่น	ความหมาย
conv_integer(p)	เป็นฟังก์ชั่นในการแปลงข้อมูล p ซึ่งเป็นประเภทข้อมูล SIGNED UNSIGNED และ STD_ULOGIC ให้เป็นประเภทข้อมูล INTEGER
conv_unsigned(p,b)	ฟังก์ชั่น เป็นฟังก์ชั่นในการแปลงข้อมูล p ซึ่งเป็นประเภทข้อมูล INTEGER SIGNED SIGNED และ STD_ULOGIC ให้เป็นประเภทข้อมูล UNSIGNED ตามขนาดจำนวน b บิต
conv_signed(p,b)	เป็นฟังก์ชั่นในการแปลงข้อมูล p ซึ่งเป็นประเภทข้อมูล INTEGER UNSIGNED และ STD_ULOGIC ให้เป็นประเภทข้อมูล SIGNED ตามขนาดจำนวน b บิต
conv_std_logic_vector(p,b)	เป็นฟังก์ชั่นในการแปลงข้อมูล p ซึ่งเป็นประเภทข้อมูล INTEGER UNSIGNED และ SIGNED ให้เป็นประเภทข้อมูล STD_LOGIC_VECTOR ตามขนาดจำนวน bบิต

ตัวอย่างการนำไปใช้งาน

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   ...
4   ARCHITECTURE behav OF ex_conv IS
5   TYPE long IS INTEGER RANGE -100 TO 100;
6   TYPE short IS INTEGER RANGE -10 TO 10;
7   SIGNAL x : short;
8   SIGNAL y : long;
9   BEGIN
10  ...
11     y <= 2*x + 5;            -- ไม่สามารถใช้งานได้
12  ...
13  END behav;
```

- จากข้อมูลตัวแปร x และ y ไม่สามารถนำผลลัพธ์ของ 2*x + 5 ส่งผ่านค่าไปยัง y เนื่องจากค่าของ x และ y เป็นคนละประเภทกัน

```
1    LIBRARY ieee;
2    USE ieee.std_logic_1164.all;
3    USE ieee.std_logic_arith.all;
4    ...
5    ARCHITECTURE behav OF ex_conv2 IS
6    SIGNAL a: UNSIGNED (7 DOWNTO 0);
7    SIGNAL b: UNSIGNED (7 DOWNTO 0);
8    SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);
9    BEGIN
10   ...
11      y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
12   ...
13   END behav;
12   ...
13   END behav;
```

- จากข้อมูลตัวแปร a และ b เป็นประเภทข้อมูล UNSIGNED ขนาด 8 บิต เมื่อนำมากระกระทำกันทางคณิตศาสตร์แล้ว จะส่งผ่านค่าไปยัง y โดยใช้ฟังก์ชั่น CONV_STD_LOGIC_VECTOR ช่วย

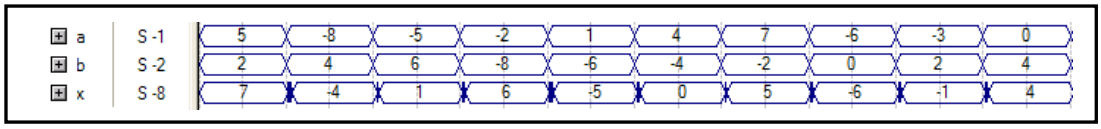
2.8 ตัวอย่างการเขียนโปรแกรมภาษา VHDL

<u>ตัวอย่างที่ 2.1</u> เป็นการออกแบบวงจรบวกเลขฐานสอง ขนาด 4 บิต โดยใช้ประเภทข้อมูลแบบ SIGNED

คำอธิบาย การบวกเลขขนาด 4 บิต จะกำหนดให้มีอินพุท 2 ตัวแปรคือ a และ b มีขนาด 4 บิต และเอาท์พุทของผลบวกเป็นตัวแปร x มีขนาด 4 บิต ซึ่งตัวแปรทั้งสามจะต้องเป็นประเภทข้อมูลแบบ SIGNED เหมือนกัน

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_arith.all;
4   ----------------------------------------
5   ENTITY adder1 IS
6   PORT ( a, b  : IN SIGNED (3 DOWNTO 0);
7              x : OUT SIGNED (3 DOWNTO 0));
8   END adder1;
9   ----------------------------------------
10  ARCHITECTURE behav OF adder1 IS
11  BEGIN
12     x <= a + b;
13 END behav;
```

เมื่อนำแฟ้มข้อมูลไปทำการจำลองดูรูปสัญญาณ จะทำให้ได้รูปสัญญาณอินพุทและเอาท์พุท ดังแสดงในรูปที่ 2.2 เช่นกัน



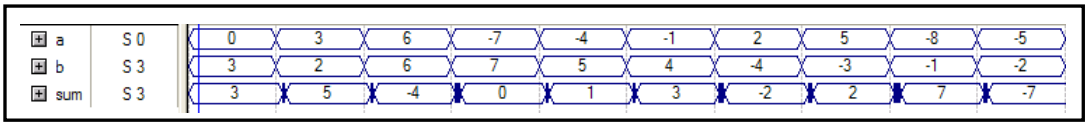
รูปที่ 2.2 ผลการจำลองรูปคลื่นสัญญาณ ตัวอย่างที่ 2.1

<u>ตัวอย่างที่ 2.2</u> เป็นการออกแบบวงจรบวกเลขฐานสอง ขนาด 4 บิต โดยใช้ประเภทข้อมูลแบบ SIGNED INTEGER ร่วมด้วยฟังก์ชั่นการแปลงข้อมูล CONV_INTEGER

คำอธิบาย การบวกเลขขนาด 4 บิต จะกำหนดให้มีอินพุท 2 ตัวแปรคือ a และ b มีขนาด 4 บิต เป็นประเภทข้อมูลแบบ SIGNED และเอาท์พุทของผลบวกเป็นตัวแปร sum มีขนาด 4 บิต แต่เป็นประเภทข้อมูลแบบ INTEGER

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_arith.all;
4   ----------------------------------------
5   ENTITY adder2 IS
6   PORT ( a, b  : IN SIGNED (3 DOWNTO 0);
7          sum : OUT INTEGER RANGE -16 TO 15);
8   END adder2;
9   ----------------------------------------
10  ARCHITECTURE behav OF adder2 IS
11  BEGIN
12    sum <= CONV_INTEGER(a + b);
13 END behav;
```

เมื่อนำแฟ้มข้อมูลไปทำการจำลองดูรูปสัญญาณ จะทำให้ได้รูปสัญญาณอินพุทและเอาท์พุท ดังแสดงในรูปที่ 2.2 เช่นกัน



รูปที่ 2.2 ผลการจำลองรูปคลื่นสัญญาณ ตัวอย่างที่ 2.2

2.9 โจทย์ปัญหา

กำหนดให้ ประเภทของข้อมูลต่อไปนี้ สำหรับตอบคำถามในข้อ 1 และข้อ 2

```
TYPE array1 IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
TYPE array2 IS ARRAY (3 DOWNTO 0, 7 DOWNTO 0) OF STD_LOGIC;
TYPE array3 IS ARRAY (3 DOWNTO 0) OF array1;

SIGNAL a : BIT;
SIGNAL b : STD_LOGIC;:
SIGNAL x : array1;
SIGNAL y : array2;
SIGNAL w : array3;
SIGNAL z : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

1. ให้หาขนาดของข้อมูล จากประเภทของข้อมูลที่กำหนดให้ (ว่าเป็น สกาล่าร์ เวกเตอร์ขนาด 1มิติ 2มิติ หรือ ขนาด 1x1มิติ)
2. จากตาราง ให้หาคำตอบของตัวแปรดังนี้ และพิจารณาถึงความถูกต้องของการส่งผ่านค่าว่าถูกต้องหรือไม่

การส่งผ่านค่า	คำตอบของตัวแปร และ การส่งผ่านค่าถูกต้องหรือไม่

`a <= x(2);`
`b <= x(2);`
`b <= y(3,5);`
`b <= w(5)(3);`
`y(1)(0) <= z(7);`
`x(0) <= y(0,0);`
`x <= "1110000";`
`a <= "0000000";`
`y(1) <= x;`
`w(0) <= y;`
`w(1) <= (7=>'1', OTHERS=>'0');`
`y(1) <= (0=>'0', OTHERS=>'1');`
`w(2)(7 DOWNTO 0) <= x;`
`w(0)(7 DOWNTO 6) <= z(5 DOWNTO 4);`
`x(3) <= x(5 DOWNTO 5);`
`b <= x(5 DOWNTO 5);`
`y <= ((OTHERS=>'0'), (OTHERS=>'0'),`
`(OTHERS=>'0'), "10000001");`
`z(6) <= x(5);`
`z(6 DOWNTO 4) <= x(5 DOWNTO 3);`
`z(6 DOWNTO 4) <= y(5 DOWNTO 3);`
`y(6 DOWNTO 4) <= z(3 TO 5);`
`y(0, 7 DOWNTO 0) <= z;`
`w(2,2) <= '1';`	

3 ประเภทของตัวกระทำและตัวตรวจคุณลักษณะข้อมูล

Opertor and Attribute

ในการใช้งานภาษา VHDL นอกจากจะต้องเข้าโครงสร้างและประเภทข้อมูลแล้ว ยังต้องเข้าใจถึงลักษณะตัวกระทำข้อมูล และตัวตรวจคุณลักษณะข้อมูล ซึ่งถือว่าเป็นสิ่งที่สำคัญเช่นกัน ดังนั้นในบทนี้จะอธิบายถึงประเภทของตัวกระทำตัวตรวจสอบคุณลักษณะข้อมูล ตัวตรวจสอบคุณลักษณะของสัญญาณ และการกำหนดค่าพารามิเตอร์ของตัวแปร

3.1 ประเภทของตัวกระทำ(Operators)

ตัวกระทำเป็นประเภทของข้อมูลที่ถูกกำหนดไว้แล้วในภาษา VHDL สามารถแบ่งออกเป็น 5 ประเภทใหญ่ๆ คือ

- ตัวกระทำชนิดการส่งผ่านค่า
- ตัวกระทำทางด้านลอจิก
- ตัวกระทำทางด้านคณิตศาสตร์
- ตัวกระทำทางด้านการเปรียบเทียบ
- ตัวกระทำทางด้านการรวมข้อมูล

ตัวกระทำแต่ละประเภทจะมีการใช้งานที่แตกต่างกัน ดังนี้

3.1.1 ตัวกระทำชนิดการส่งผ่านค่า(assignment)

จะมีคุณลักษณะที่ใช้ระบุหรือกำหนดค่าให้กับตัวแปรสัญญาณ ซึ่งได้แก่สัญญาณประเภท

- SIGNAL
- VARIABLE
- CONSTANT

ตัวกระทำชนิดการส่งผ่านค่านี้จะมีคุณลักษณะของการกำหนดค่าตามเครื่องหมายที่ใช้ ดังนี้

เครื่องหมาย	ความหมาย
<=	ใช้สำหรับกำหนดค่าให้ตัวแปรสัญญาณชนิด SIGNAL
:=	ใช้สำหรับกำหนดค่าให้ตัวแปรสัญญาณชนิด VARIABLE และใช้กับตัวแปรสัญญาณ CONSTANT หรือ GENERIC เพื่อให้เป็นค่าเริ่มต้น
=>	ใช้สำหรับกำหนดค่าองค์ประกอบของเวกเตอร์ทีละตัวหรือตัวอื่นๆ

ตัวกระทำชนิดการส่งผ่านค่าทั้งสามลักษณะนี้ เมื่อมีการนำไปใช้จะแตกต่างกัน ดังตัวอย่างต่อไปนี้

```
1    ARCHITECTURE behav OF ex_41 IS
2    SIGNAL x : STD_LOGIC;
3    VARIABLE y : STD_LOGIC_VECTOR(3 DOWNTO 0);
4    SIGNAL w: STD_LOGIC_VECTOR(0 TO 7);
5     ...
6       x <= '1';
7       y := "0000";
8       w <= "10000000";
9       w <= (0 =>'1', OTHERS =>'0');
10      ...
11   END behav;
```

จากตัวอย่าง สามารถอธิบายได้คือ

- `x <= '1';`
  เป็นการกำหนดให้ข้อมูลที่มีค่า ‘1’ ส่งผ่านไปยังตัวแปร x ที่เป็นตัวแปรชนิด SIGNAL หรือพูดง่ายๆก็คือตัวแปร x มีข้อมูลค่าเท่ากับ ‘1’ นั่นเอง
- `y := "0000";`
  เป็นการกำหนดให้ข้อมูลที่มีค่าเท่ากับ "0000" ส่งผ่านไปยังตัวแปร y ซึ่งเป็นประเภทข้อมูลชนิด STD_LOGIC_VECTOR โดยที่บิตสูงสุดอยู่ทางซ้ายมือ
- `w <= "10000000";`
  เป็นการกำหนดให้ข้อมูลที่มีค่า "10000000" ส่งผ่านไปยังตัวแปร w ซึ่งเป็นประเภทข้อมูลชนิด STD_LOGIC_VECTOR โดยบิตสูงสุดอยู่ทางขวามือ
- `w <= (0 =>'1', OTHERS =>'0');`
  เป็นการกำหนดให้ข้อมูลที่มีค่า '1' ส่งผ่านไปยังตัวแปร w ที่ตำแหน่งบิต 0 และส่งผ่านข้อมูลที่มีค่า ‘0’ ให้กับตำแหน่งบิตอื่นๆที่เหลือ (บิต 1 ถึง 7)

3.1.2 ตัวกระทำทางด้านลอจิก (Logic)

ตัวกระทำทางลอจิกหรือตรรกะนี้ จะให้ผลลัพธ์จริงหรือเท็จ ซึ่งเป็นตัวกระทำเฉพาะ ดังนั้นจึงใช้ได้เฉพาะกับประเภทของข้อมูลที่เป็น

- BIT
- STD_LOGIC
- STD_ULOGIC
- BIT_VECTOR
- STD_LOGIC_VECTOR
- STD_ULOGIC_VECTOR

ตัวกระทำทางด้านลอจิกแต่ละตัวจะมีความหมายและตัวอย่างใช้งานดังนี้

ตัวกระทำการ	ความหมาย	ตัวอย่าง
AND	การกระทำแบบแอนด์	Z <= (A AND B)
OR	การกระทำแบบออร์	Z <= (A OR B)
NOT	การกระทำแบบน๊อท	Z <= NOT(A)
NAND	การกระทำแบบแนนด์	Z <= (A NAND B)
NOR	การกระทำแบบนอร์	Z <= (A NOR B)
XOR	การกระทำแบบเอ็กคลูซีพออร์	Z <= (A XOR B)
XNOR	การกระทำแบบเอ็กคลูซีพนอร์	Z <= (A XNOR B)

⇨ ในการใช้งานตัวกระทำ AND OR NAND NOR XOR ในภาษา VHDL ถือว่าเป็นตัวกระทำที่มีค่าความสำคัญเท่ากัน แต่ตัวกระทำ NOT ถือว่าเป็นตัวกระทำที่มีลำดับสำคัญมากกว่า นั่นก็หมายความว่าจะกระทำก่อน <u>ยกเว้น</u>เสียแต่ว่าจะกำหนดให้กระทำในเครื่องหมายวงเล็บก่อน

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_bit_vector IS
2    SIGNAL a,b,z     : BIT;
3    SIGNAL va,vb,vx  : BIT_VECTOR(0 TO 2);
4     BEGIN
5            z <= NOT a AND b;     -- หมายถึง a'.b
6            vx <= va NOR vb;      -- หมายถึง (va+vb)'
7                ...
8    END behav;
```

3.1.3 ตัวกระทำทางด้านคณิตศาสตร์ (Arithmetic)

การใช้งานตัวกระทำชนิดนี้ จะสามารถนำไปใช้กับประเภทของข้อมูลได้หลายประเภท ได้แก่

- INTEGER
- SIGNED
- UNSIGNED
- STD_LOGIC_VECTOR ♣

⇨ ♣<u>ข้อแม้</u> ของการใช้ STD_LOGIC_VECTOR คือจะต้องมีการใช้ package ชื่อ std_logic_singed และ std_logic_unsinged จาก LIBRARY ชื่อ ieee เมื่อมีตัวแปรประเภทของข้อมูลเป็น STD_LOGIC_VECTOR จึงจะสามารถที่กระทำการบวกและลบได้โดยตรง (ดังที่ได้อธิบายไว้แล้วในหัวข้อ 2.6)

ลักษณะของตัวกระทำชนิดนี้จะประกอบไปด้วย

ตัวกระทำการ	ความหมาย	ตัวอย่าง
+	การบวก	Z <= A+B
-	การลบ	Z <= A-B
*	การคูณ	Z <= A*B
/	การหาร	Z <= A/B
**	ยกกำลัง	Z <= 4**2
MOD	การหาร คิดเฉพาะเศษ	Z <= A MOD B
REM	การหาร คิดเฉาะส่วน	Z <= A REM B
ABS	ค่าจำนวนเต็ม	Z <= ABS(A)

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_bit_vector IS
2    SIGNAL a,b : INTEGER RANGE 0 TO 7;
3    SIGNAL z   : INTEGER RANGE 0 TO 15;
4     BEGIN
5            z <= a + b;
6    END behav;
```

3.1.4 ตัวกระทำทางด้านเปรียบเทียบ (Comparison)

ตัวกระทำชนิดนี้สามารถใช้ได้กับประเภทข้อมูลทุกประเภท จะให้ผลลัพธ์จริงหรือเท็จ ซึ่งใช้สำหรับตรวจสอบเงื่อนไข ลักษณะของตัวกระทำชนิดนี้จะประกอบไปด้วย

ตัวกระทำการ	ความหมาย	ตัวอย่าง
=	เท่ากัน	IF (a = b) THEN
/=	ไม่เท่ากัน	IF (a /= b) THEN
<	น้อยกว่า	IF (a < b) THEN
>	มากกว่า	IF (a > b) THEN
<=	มากกว่าหรือเท่ากัน	IF (a <= b) THEN
>=	น้อยกว่าหรือเท่ากัน	IF (a >= b) THEN

ตัวอย่างการใช้งานเช่น

```
1    ARCHITECTURE behav OF ex_comp IS
2    SIGNAL a,b,z     : BIT;
3     BEGIN
4         ...
5            IF A = B THEN
6               z <= '0';
7            ELSE
8               z <= '1';
9            END IF;
10              ...
11   END behav;
```

3.1.5 ตัวกระทำทางด้านการรวมข้อมูล (Concatenation)

เป็นตัวกระทำที่สามารถนำสัญญาณย่อยๆหรืออะเรย์ย่อยมาต่อกันหรือรวมกัน โดยใช้เครื่องหมาย &

ตัวกระทำการ	ความหมาย	ตัวอย่าง
&	เป็นการรวมข้อมูลเข้าด้วยกัน	Z <= A & B

ตัวอย่างการนำไปใช้งาน

```
1    ARCHITECTURE behav OF ex_conc IS
2    SIGNAL a,b,c     : BIT;
3    SIGNAL a_bus,b_bus : BIT_VECTOR(2 DOWNTO 0);
4    SIGNAL t_bus     : BIT_VECTOR(5 DOWNTO 0);
5     BEGIN
6            a_bus <= a & b & c;         -- รวมสัญญาณ
7            t_bus <= a_bus & b_bus;     -- รวมสัญญาณ
8               ...
9    END behav;
```

ตารางที่ 3.1 สรุปประเภทของตัวกระทำ

ชนิดตัวกระทำ	ตัวกระทำ(operator)	ประเภทของข้อมูล(data type)
ด้านลอจิก	NOT AND NAND OR NOR XOR XNOR	BIT BIT_VECTOR STD_LOGIC, STD_LOGIC_VECTOR STD_ULOGIC STD_ULOGIC_VECTOR
คณิตศาสตร์	+ - * / ** (mod, rem, abs)	INTEGER SIGNED UNSIGNED
การเปรียบเทียบ	= /= < > <= >=	ทุกประเภท
รวมข้อมูล	& (,,,)	คล้ายกับตัวกระทำการแบบตรรกะรวมไปถึง SIGNED และ UNSIGNED

3.2 ตัวตรวจสอบคุณลักษณะข้อมูล (Data Attributes)

ตัวตรวจสอบคุณลักษณะข้อมูล เป็นคุณลักษณะอย่างหนึ่งที่จำเป็นต้องทราบ เพื่อนำไปใช้กำหนดการทำงานของวงจรที่ต้องการออกแบบ ซึ่งจะประกอบด้วยตัวตรวจสอบคุณลักษณะดังนี้

- d’LOW
- d’High
- d’LEFT
- d’RIGHT
- d’LENGTH
- d’RANGE
- d’REVERSE_RANGE

ตัวตรวจสอบคุณลักษณะข้อมูลในแต่ละรูปแบบ จะมีความหมายดังนี้

	ความหมาย
d’LOW	เป็นตัวตรวจเพื่อชี้ตำแหน่ง ต่ำสุดของข้อมูล
d’HIGH	เป็นตัวตรวจเพื่อชี้ตำแหน่ง สูงสุดของข้อมูล
d’LEFT	เป็นตัวตรวจเพื่อชี้ตำแหน่ง ทางซ้ายมือสุดของข้อมูล
d’RIGHT	เป็นตัวตรวจเพื่อชี้ตำแหน่ง ทางขวามือสุดของข้อมูล
d’LENGTH	เป็นตัวตรวจเพื่อหา ขนาดความยาวของข้อมูล
d’RANGE	เป็นตัวตรวจเพื่อหา ช่วงหรือย่านของข้อมูล
d’REVERSE_RANGE	เป็นตัวตรวจเพื่อหา ช่วงหรือย่านของข้อมูล ที่ให้ค่าตรงข้ามกับ d’RANGE

ตัวอย่างการใช้งานเช่น

```
1   ARCHITECTURE behav OF ex_att IS
2   SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);
3      ...
4   END behav;
```

- จากตัวอย่าง ถ้าใช้ตัวตรวจสอบคุณลักษณะข้อมูล
- ถ้าใช้ d'LOW จะได้ค่าเป็น 0
- ถ้าใช้ d'HIGH จะได้ค่าเป็น 7
- ถ้าใช้ d'LEFT จะได้ค่าเป็น 7
- ถ้าใช้ d'RIGHT จะได้ค่าเป็น 0
- ถ้าใช้ d'LENGTH จะได้ค่าเป็น 8

- ถ้าใช้ d'RANGE จะได้ค่าเป็น 7 DOWNTO 0
- ถ้าใช้ d'REVERSE_RANGE จะได้ค่าเป็น 0 TO 7

โดยทั่วไปแล้วตัวตรวจสอบคุณลักษณะข้อมูล จะถูกนำไปใช้ในชุดคำสั่ง LOOP ดังตัวอย่างต่อไปนี้

```
1   ARCHITECTURE behav OF ex_att2 IS
2   SIGNAL x : STD_LOGIC_VECTOR (7 DOWNTO 0);
3      ...
4      FOR i IN RANGE(0 TO 7) LOOP ...
5      FOR i IN x'RANGE LOOP ...
6      FOR i IN RANGE (x'LOW TO x'HIGH) LOOP ...
7      FOR i IN RANGE (0 TO x'LENGTH-1) LOOP ...
8      ...
9   END behav;
```

o จากตัวอย่าง

บรรทัดที่ 4 เป็นการใช้งานปกติของชุดคำสั่ง LOOP

บรรทัดที่ 5 ถึง 7 เป็นการใช้งาน ชุดคำสั่ง LOOP ร่วมกับตัวตรวจสอบคุณลักษณะข้อมูล

3.3 ตัวตรวจสอบคุณลักษณะของสัญญาณ (Signal Attributes)

ตัวตรวจสอบคุณลักษณะของสัญญาณข้อมูล เป็นคุณลักษณะอย่างหนึ่งเหมือนกับตัวตรวจสอบคุณลักษณะข้อมูลที่จำเป็นต้องทราบ เพื่อนำไปใช้กำหนดการทำงานของวงจรที่ต้องการออกแบบ ซึ่งจะประกอบด้วย

- s'EVENT
- s'STABLE
- s'ACTIVE
- s'QUIET<time>
- s'LAST_EVENT
- s'LAST_ACTIVE
- s'LAST_VALUE

ตัวตรวจสอบคุณลักษณะของสัญญาณ ในแต่ละรูปแบบ จะมีความหมายดังนี้

	ความหมาย
s'EVENT	ให้ค่าเป็นจริง เมื่อตรวจสอบแล้วมีเหตุการณ์ของสัญญาณ s เกิดขึ้น
s'STABLE	ให้ค่าเป็นจริง เมื่อตรวจสอบแล้วไม่มีเหตุการณ์ของสัญญาณ s เกิดขึ้น หรือค่าความมีเสถียรภาพของสัญญาณ s
s'ACTIVE	ให้ค่าเป็นจริง เมื่อตรวจสอบแล้ว สัญญาณ s มีค่าเป็น '1'
s'QUIET<time>	ให้ค่าเป็นจริง เมื่อตรวจสอบแล้ว ไม่มีเหตุการณ์ในช่วงเวลาที่กำหนด
S'LAST_EVENT	ให้ค่าเป็นเวลา เมื่อตรวจสอบ เหตุการณ์ครั้งล่าสุดผ่านพ้นไป
S'LAST_ACTIVE	ให้ค่าเป็นเวลา เมื่อ สัญญาณ s มีค่าเป็น '1' ครั้งล่าสุดผ่านพ้นไป
s'LAST_VALUE	ให้ค่าของสัญญาณ s ก่อนเหตุการณ์ครั้งล่าสุด

โดยทั่วไปแล้วตัวตรวจสอบคุณลักษณะสัญญาณ จะถูกนำไปใช้ในชุดคำสั่ง IF และ WAIT ดังตัวอย่างต่อไปนี้

```
1    ARCHITECTURE behav OF ex_att3 IS
2       ...
3       IF (clk'EVENT AND clk = '1' THEN ...
4       IF (NOT clk'STABLE AND clk = '1' THEN ...
5       WAIT UNTIL (clk'EVENT AND clk = '1') ;
6       ...
7    END behav;
```

o จากตัวอย่าง

บรรทัดที่ 3 เป็นการตรวจสอบว่ามีเหตุการณ์ของสัญญาณ clk เกิดขึ้นหรือไม่

บรรทัดที่ 4 เป็นการตรวจสอบว่ามีสัญญาณ clk มีเสถียรภาพหรือไม่

บรรทัดที่ 5 เป็นการตรวจรอจนมีเหตุการณ์ของสัญญาณ clk เกิดขึ้น

3.4 การประกาศใช้ Generic

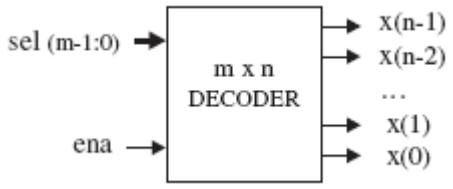
เป็นส่วนที่มีไว้สำหรับประกาศหรือกำหนดค่าพารามิเตอร์ของตัวแปร โดยที่ไม่ต้องทำการแก้ในส่วนของ ARCHITECTURE ดังนั้นส่วนที่การประกาศไว้ GENERIC จึงเป็นส่วนมีความยืดหยุ่น ที่สามารถแก้ไขได้เพิ่มเติมได้ภายหลัง โดยมีลักษณะกฎการเขียนดังนี้

```
ENTITY ...
GENERIC (parameter_name : parameter_type := parameter_value;
PORT ...
END ... ;
```

3.5 ตัวอย่างการเขียนโปรแกรมภาษา VHDL

<u>ตัวอย่างที่ 3.1</u> การออกแบบวงจร Decoder ขนาดเข้า 3 ออก 8

คำอธิบาย จากรูปเป็นวงจร Decoder มีขนาดอินพุท sel ขนาด 3 บิต และเอาท์พุท x ขนาด 8 บิต ส่วนอินพุท ena ขนาด 1บิต เป็นส่วนที่มีไว้สำหรับอนุญาตให้มีการ decoder หรือไม่



รูปที่ 3.1 โครงสร้างของวงจร Decoder ขนาดเข้า 3 ออก 8

```
1 ---------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY decoder IS
6     PORT ( ena : IN STD_LOGIC;
7            sel : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8            x : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9 END decoder;
10 ---------------------------------------------
11 ARCHITECTURE generic_decoder OF decoder IS
```

```
12 BEGIN
13    PROCESS (ena, sel)
14    VARIABLE temp1 : STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
15    VARIABLE temp2 : INTEGER RANGE 0 TO x'HIGH;
16    BEGIN
17          temp1 := (OTHERS => '1');
18          temp2 := 0;
19          IF (ena='1') THEN
20             FOR i IN sel'RANGE LOOP -- sel range is 2 downto 0
21                IF (sel(i)='1') THEN -- Bin-to-Integer conv.
22                      temp2:=2*temp2+1;
23                ELSE
24                      temp2 := 2*temp2;
25                END IF;
26             END LOOP;
27             temp1(temp2):='0';
28          END IF;
29          x <= temp1;
30    END PROCESS;
31 END generic_decoder;
32 ---------------------------------------------
```

o จากตัวโปรแกรมจะเห็นว่ามีการเลือกใช้

- ตัวกระทำ

  "+" ในบรรทัดที่ 22

  "*" ในบรรทัดที่ 22 และ 24

- ตัวส่งผ่านค่า

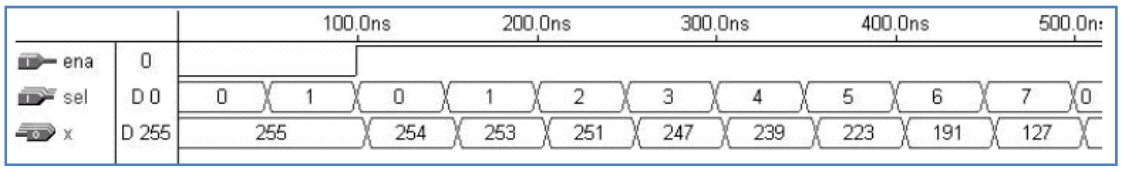
  ":=" ในบรรทัดที่ 17 18 22 24 และ 27

  "<=" ในบรรทัดที่ 29

  "=>" ในบรรทัดที่ 17

- ตัวตรวจสอบคุณลักษณะข้อมูล

  "x'HIGH" ในบรรทัดที่ 14 และ 15



รูปที่ 3.2 ผลการจำลองของตัวอย่างที่ 3.1

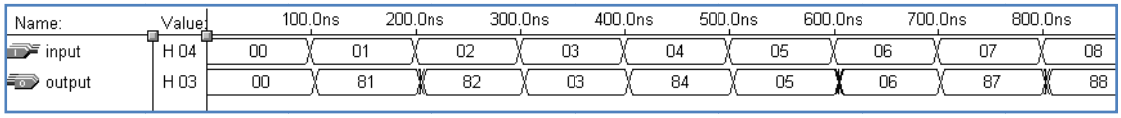
ในตัวอย่างนี้เมื่อเขียนเป็นภาษา VHDL แล้วนำไปจำลองดูผลจะได้ดังรูปที่ 3.2 จะพบว่าถ้าอินพุท sel = "000" (มีค่าเท่ากับ 0 ฐานสิบ) จะได้ x = "11111110" (มีค่าเท่ากับ 254 ฐานสิบ) และเช่นเดียวกัน ถ้าอินพุท sel = "010" (มีค่าเท่ากับ 2 ฐานสิบ) จะได้ x = "11111011" (มีค่าเท่ากับ 251 ฐานสิบ)

<u>ตัวอย่างที่ 3.2</u> การออกแบบวงจรกำเนิด parity

คำอธิบาย จากรูปเป็นวงจร parity มีขนาดอินพุท n-1 บิต และเอาท์พุท n บิต เมื่อใช้ GENERIC เราสามารถที่จะกำหนด n ที่เท่าใดก็ได้ ซึ่งจะทำให้ค่าตัว n ในทุกส่วนของโปรแกรมเปลี่ยนไปด้วย จึงทำให้ง่ายต่อการปรับปรุง

รูปที่ 3.3 โครงสร้างของวงจร parity

```
1 ------------------------------------------------
2  ENTITY parity_gen IS
3     GENERIC (n : INTEGER := 7);
4     PORT ( input: IN BIT_VECTOR (n-1 DOWNTO 0);
5            output: OUT BIT_VECTOR (n DOWNTO 0));
6  END parity_gen;
7 ------------------------------------------------
8  ARCHITECTURE Behavior OF parity_gen IS
9  BEGIN
10    PROCESS (input)
11    VARIABLE temp1: BIT;
12    VARIABLE temp2: BIT_VECTOR (output'RANGE);
13    BEGIN
14          temp1 := '0';
15          FOR i IN input'RANGE LOOP
16                temp1 := temp1 XOR input(i);
17                temp2(i) := input(i);
18          END LOOP;
19          temp2(output'HIGH) := temp1;
20          output <= temp2;
21    END PROCESS;
22 END Behavior;
```



รูปที่ 3.4 ผลการจำลองของตัวอย่างที่ 3.2

ในตัวอย่างนี้เมื่อเขียนเป็นภาษา VHDL แล้วนำไปจำลองดูผลจะได้ดังรูปที่ 3.4 จะพบว่าถ้าอินพุท input = “0000000” (มีค่าเท่ากับ 0 ฐานสิบหก) จะได้ output = “00000000” (มีค่าเท่ากับ 0 ฐานสิบหก) และเช่นเดียวกัน ถ้าอินพุท input = “0000001” (มีค่าเท่ากับ 1 ฐานสิบ) จะได้ output = “10000001” (มีค่าเท่ากับ 81 ฐานสิบหก)

3.6 โจทย์ปัญหา

กำหนดให้ ใช้ตัวแปรดังต่อไปนี้ สำหรับตอบคำถามในข้อ 1 ถึง 3

```
SIGNAL a : BIT := '1';
SIGNAL b : BIT_VECTOR (3 DOWNTO 0) := "1100";
SIGNAL c : BIT_VECTOR (3 DOWNTO 0) := "0010";
SIGNAL d : BIT_VECTOR (7 DOWNTO 0);
SIGNAL e : INTEGER RANGE 0 TO 255;
SIGNAL f : INTEGER RANGE -128 TO 127;
```

1. ให้หาคำตอบของตัวแปรดังต่อไปนี้ เมื่อผ่านตัวกระทำการ

```
x1 <= a & c;        คำตอบคือ  x1  ________
x2 <= c & b;        คำตอบคือ  x2  ________
x3 <= b XOR c;      คำตอบคือ  x3  ________
x4 <= a NOR b(3);   คำตอบคือ  x4  ________
x5 <= a AND NOT b(0) AND NOT c(1);
                    คำตอบคือ  x5  ________
d <= (5=>'0', OTHERS=>'1');   คำตอบคือ  ________
```

2. ให้หาคำตอบของตัวตรวจสอบคุณลักษณะของสัญญาณข้อมูล ดังต่อไปนี้

```
c'LOW             คำตอบคือ  ______
d'HIGH            คำตอบคือ  ______
c'LEFT            คำตอบคือ  ______
d'RIGHT           คำตอบคือ  ______
c'RANGE           คำตอบคือ  ______
d'LENGTH          คำตอบคือ  ______
c'REVERSE_RANGE   คำตอบคือ  ______
```

3. จากชุดคำสั่งด้านล่างต่อไปนี้ สามารถใช้งานได้หรือไม่

```
b(0) AND a
a + d(7)
NOT b XNOR c
c + d
e - f
IF (b<c) ...
IF (b>=a) ...
IF (f/=e) ...
IF (e>d) ...
e*3
5**5
f/4
e/3
d <= c
d(6 DOWNTO 3) := b
e <= d
f := 100
```

4. จากตัวอย่างที่ 3.1 ถ้าต้องการใช้ GENERIC อินพุทเป็น m บิต และเอาท์พุทเป็น $n = 2^m$ บิต จะต้องเขียนอย่างไร

4 ชุดคำสั่งชนิดแข่งขนาน

Concurrent Codes

ชุดคำสั่งชนิดแข่งขนาน (concurrent code) ในภาษา VHDLนี้ เป็นชุดคำสั่งที่ถูกนำไปใช้ในบรรยายถึงพฤติกรรมของวงจรดิจิตอล ประเภทคอมไบเนชั่น (combination logic) ซึ่งวงจรประเภทนี้มีเอาท์พุทเปลี่ยนแปลงตามอินพุทตลอดเวลา ดังนั้นการออกแบบวงจรที่ใช้ชุดคำสั่งแบบนี้ จะมีลักษณะทำงานพร้อมๆ กันและอิสระต่อกัน หรือมีผลกระทบถึงกันในเวลาเดียวกัน

โดยทั่วไปแล้ว ชุดคำสั่งชนิดแข่งขนานในภาษา VHDL จะมีการใช้ร่วมกับกลุ่มคำสั่งสองจำพวก อันได้แก่

- กลุ่มคำสั่ง WHEN
- กลุ่มคำสั่ง GENERATE

การใช้งานกลุ่มคำสั่งทั้งสองชนิดนี้ จะต้องใช้งานร่วมกับตัวกระทำการเสมอ ดังนั้นบทนี้จะนำเสนอถึงการใช้งานทั้งสองกลุ่มที่ถูกใช้งานบ่อยครั้ง ดังต่อไปนี้

4.1 กลุ่มคำสั่ง WHEN

เป็นกลุ่มคำสั่งที่มีการใช้งานค่อนข้างง่ายๆ ซึ่งใช้สำหรับเพื่อคอยตรวจสอบเงื่อนไข จะมีการใช้งานอยู่ 2 รูปแบบ คือ

- รูปแบบ WHEN/ELSE
- รูปแบบ WITH/SELECT/WHEN

4.1.1 รูปแบบ WHEN/ELSE

เป็นรูปแบบลักษณะการใช้ WHEN โดยมีกฎเกณฑ์การเขียนดังนี้

```
assignment WHEN condition ELSE
assignment WHEN condition ELSE
...;
```

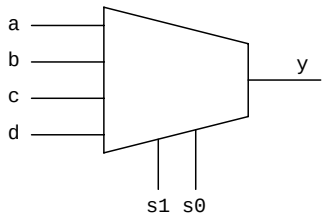
- โดยที่ assignment เป็นตัวกระทำการแบบส่งผ่านค่า (<=) เมื่อเงื่อนไขที่ตรวจสอบเป็นจริง ดังแสดงให้เห็นในตัวอย่างต่อไปนี้

```
1   ARCHITECTURE behav OF ex_when1 IS
2   BEGIN
3       outp <=  "000" WHEN (inp='0' OR resetn='1') ELSE
4                "001" WHEN ctl ='1' ELSE
5                "010";
6             ...
7   END behav;
```

o จากตัวอย่าง ตัวแปร outp จะมีค่าเท่ากับ “000” เมื่อเงื่อนไขของ ตัวแปร inp มีค่าเท่ากับ ‘0’ หรือตัวแปร resetn มีค่าเท่ากับ ‘1’นั้นเป็นจริง ถ้าเป็นเท็จ ก็จะตรวจสอบเงื่อนไขถัดมา คือ ตัวแปร outp จะมีค่าเท่ากับ “001” เมื่อเงื่อนไขของ ตัวแปร ctl มีค่าเท่ากับ ‘1’เป็นจริง และถ้าเป็นเท็จ ตัวแปร outp จะมีค่าเท่ากับ “010” ซึ่งเป็นเงื่อนสุดท้าย

⇨ <u>ข้อสังเกต</u> ในการใช้คำสั่ง WITH/ELSE หลัง condition ELSE จะไม่มีเครื่องหมาย ;

<u>ตัวอย่างที่ 4.1</u> การออกแบบวงจร Multiplexer



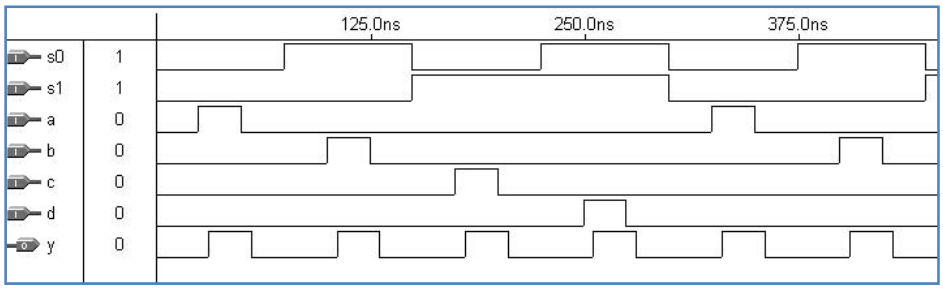
s1	s0	y
0	0	a
0	1	b
1	0	c
1	1	d

(ก) (ข)

รูปที่ 4.1 (ก) สัญลักษณ์ของวงจร Multiplexer (ข) ตารางความจริง

คำอธิบาย ในรูปที่ 4.1 เป็นสัญลักษณ์ของวงจร Multiplexer เข้า 4 ออก 1 พร้อมด้วยตารางความจริง จะมีอินพุท 4 ตัว คือ a b c และ d พร้อมด้วยขาเลือก (select)จำนวน 2 ตัว คือ s1 และ s0 ส่วนเอาท์พุทมี 1 ตัวคือ y ดังนั้นในส่วนของ ENITY ก็จะสามารถเขียนได้ดังบรรทัดที่ 5 ถึง 8 ส่วน ส่วนเอาท์พุทจะต้องเขียนให้เป็นสมการพีชคณิตก่อน จะได้ y = a.s1’.s0’+ b.s1’.s0 + c.s1.s0’+ d.s1.s0 ดังนั้นพฤติกรรมการทำงานของวงจร multiplexer ก็คือ สมการพีชคณิต y ซึ่งสามารถเขียนได้ดังบรรทัดที่ 10 ถึง 16

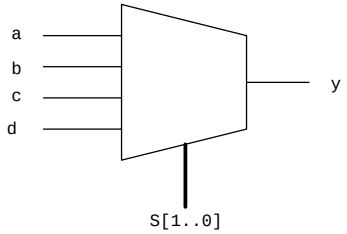
```
1  ----------------------------------------
2  LIBRARY  ieee;
3  USE  ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY  mux41  IS
6     PORT  (a,b,c,d,s0,s1 :  IN  STD_LOGIC;
7            y :  OUT  STD_LOGIC);
8  END  mux41;
9  ----------------------------------------
10  ARCHITECTURE  behave  OF  mux41 IS
11  BEGIN
12    y  <=(a  AND  NOT  s1  AND  NOT  s0)  OR
13         (b  AND  NOT  s1  AND  s0)  OR
14         (c  AND  s1  AND  NOT  s0)  OR
15         (d  AND  s1  AND  s0);
16  END  behave;
17  ----------------------------------------
```



รูปที่ 4.2 ผลการจำลองของตัวอย่างที่ 4.1

<u>ตัวอย่างที่ 4.2</u> การออกแบบวงจร Multiplexer ครั้งที่2 โดยใช้ WHEN/ELSE

ทำการเปลี่ยนแปลงขาตัวเลือกในสัญลักษณ์ของวงจร Multiplexer จากรูปที่ 4.1 (ก) ให้เป็นรูป ที่ 4.3(ก) แทน



(ก)

s[1..0]	y
00	a
01	b
10	c
11	d

(ข)

รูปที่ 4.3 (ก) สัญลักษณ์ของวงจร Multiplexer (ข) ตารางความจริง

คำอธิบาย จากรูปที่ 4.3 จะมีอินพุท 5 ตัว คือ a b c และ d พร้อมด้วยขาเลือก(select) จำนวน 2 บิตคือ s[1..0] ส่วนเอาท์พุทมี 1 ตัวคือ y ดังนั้นในส่วนของ ENITY ก็จะสามารถเขียนได้ดังบรรทัดที่ 5 ถึง 9 ส่วน ทางด้านเอาท์พุทจะใช้ชุดคำสั่ง WHEN บรรยายพฤติกรรมการทำงานของวงจรแทนสมการพีชคณิต y ซึ่งสามารถเขียนได้ดังบรรทัดที่ 11 เป็นต้นไป

```
1 ------------ with WHEN/ELSE ---------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------------
5 ENTITY mux41 IS
6     PORT ( a, b, c, d: IN STD_LOGIC;
7            sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8            y: OUT STD_LOGIC);
9 END mux41;
10 ---------------------------------------------
11 ARCHITECTURE behave1 OF mux41 IS
12 BEGIN
13    y <=  a WHEN sel="00" ELSE
14          b WHEN sel="01" ELSE
15          c WHEN sel="10" ELSE
16          d;
17 END behave1;
18 ---------------------------------------------
```

4.1.2 รูปแบบ WITH/SELECT/WHEN

เป็นอีกลักษณะหนึ่งของการใช้ WHEN ร่วมกับชุดคำสั่ง WITH/SELECT โดยมีกฎเกณฑ์การเขียนดังนี้

```
WITH identifier SELECT
assignment WHEN value,
assignment WHEN value,
...;
```

o โดยที่ identifier เป็นตัวแปรที่ใช้คอยตรวจสอบเงื่อนไข เมื่อมีค่าตรงตาม value จะทำให้ assignment ทำการส่งผ่านค่า ดังแสดงให้เห็นในตัวอย่างต่อไปนี้

```
1   ARCHITECTURE behav OF ex_when2 IS
2   BEGIN
3      WITH cont SELECT
4        outp <=  "000" WHEN “00”,
5                 "111" WHEN “01”,
6                 "ZZZ" WHEN OTHERS;
7              ...
8   END behav;
```

o จากตัวอย่าง ตัวแปร outp จะมีค่าเท่ากับ “000” เมื่อ ตัวแปร cont มีค่าเท่ากับ “00” เป็นจริง หรือ ตัวแปร outp จะมีค่าเท่ากับ “111” เมื่อ ตัวแปร cont มีค่าเท่ากับ “01”เป็นจริง หรือ ตัวแปร outp จะมีค่าเท่ากับ “ZZZ” เมื่อ ตัวแปร cont มีค่าไม่เท่ากับสองกรณีที่ผ่านมา หรือกรณีที่เหลือนั่นเอง

o

ในการใช้งานชุดคำสั่ง WHEN ค่าของ "... WHEN value" มีการใช้งานอยู่ 3 ลักษณะคือ

- WHEN value — ใช้กับ value ที่มีหนึ่งค่า
- WHEN value1 to value2 — ใช้ในช่วง value1 ถึง value2
- WHEN value1 | value2 — ใช้กับ value1 หรือ value2

⇨ <u>ข้อสังเกต</u> ในการใช้การคำสั่ง WITH/SELECT หลัง WHEN value จะใช้เครื่องหมาย , แทน ; ยกเว้นตำแหน่ง value สุดท้าย

ตัวอย่างที่ 4.3 การออกแบบวงจร Multiplexer ครั้งที่ 3 โดยใช้ WITH/SELECT/WHEN

```
1 -------------  with WITH/SELECT/WHEN  -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------------
5 ENTITY mux41 IS
6     PORT ( a, b, c, d: IN STD_LOGIC;
7            sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8            y: OUT STD_LOGIC);
9 END mux41;
10 ------------------------------------------
11 ARCHITECTURE beahve2 OF mux41 IS
12 BEGIN
13    WITH sel SELECT
14          y <=  a WHEN "00", -- notice "," instead of ";"
15                b WHEN "01",
16                c WHEN "10",
17                d WHEN OTHERS; -- cannot be "d WHEN "11" "
18 END behave2;
19 ------------------------------------------
```

จากบรรทัดที่ 7 ค่าของ sel เป็นข้อมูลประเภท STD_LOGIC_VECTOR ขนาด 2 บิต ถ้าถูกแทนด้วย ข้อมูลประเภท INTEGER และใช้ WHEN/ELSE จะทำให้เราเขียนโปรแกรม VHDL สะดวกและรวดเร็วขึ้น ดังแสดงให้ในตัวอย่างต่อไปนี้

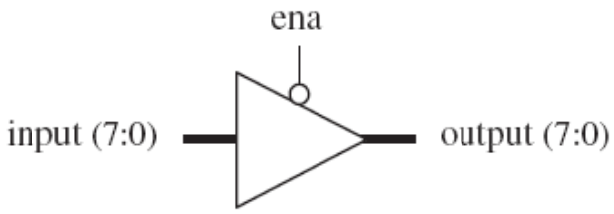
```
1 ----------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ----------------------------------------------
5 ENTITY mux41 IS
6     PORT ( a, b, c, d: IN STD_LOGIC;
7            sel: IN INTEGER RANGE 0 TO 3;
8            y: OUT STD_LOGIC);
9 END mux41;
10 ---- Solution 1: with WHEN/ELSE ---------------
11 ARCHITECTURE behave1 OF mux41 IS
12 BEGIN
13    y <=  a WHEN sel=0 ELSE
14          b WHEN sel=1 ELSE
15          c WHEN sel=2 ELSE
16          d;
17 END behave1;
```

เมื่อเปลี่ยนจาก WHEN/ELSE เป็น WITH/SELECT/WHEN ก็จะง่ายในการเขียนโปรแกรม

```
1 ----------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ----------------------------------------------
5 ENTITY mux41 IS
6     PORT ( a, b, c, d: IN STD_LOGIC;
7            sel: IN INTEGER RANGE 0 TO 3;
8            y: OUT STD_LOGIC);
9 END mux41;
10 ---- Solution 2: with WITH/SELECT/WHEN ---------------
```

```
11 ARCHITECTURE behave2 OF mux41 IS
12 BEGIN
13    y <=  a WHEN 0,
14          b WHEN 1,
15          c WHEN 2,
16          d WHEN 3; -- here, 3 or OTHERS are equivalent,
17 END behave2; -- for all options are tested anyway
```

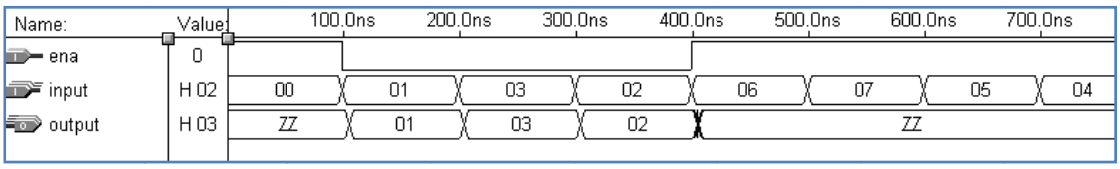
<u>ตัวอย่างที่ 4.4</u> การออกแบบวงจร Tri-state Buffer



รูปที่ 4.4 สัญลักษณ์วงจร Tri-state Buffer

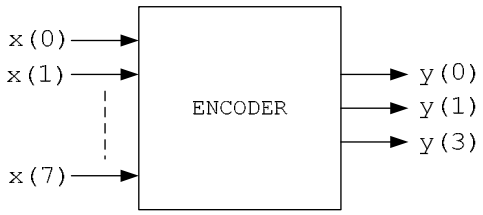
คำอธิบาย จากรูปที่ 4.4 เป็นวงจร Tri-state Buffer ขนาด 8 บิต ซึ่งมีลักษณะการทำงานดังนี้ สภาวะสัญญาณอินพุทไหลผ่านไปยังเอาท์พุท เมื่อสัญญาณควบคุมขา ena มีสถานะเป็นลอจิก '0' ในทางตรงกันข้ามถ้าสัญญาณควบคุมขา ena มีสถานะเป็นลอจิก '1' ที่เอาท์พุทจะมีสภาวะเปิดวงจรที่ให้ค่าความต้านทางสูง(high impedance) ในภาษา VHDL จะใช้ตัว Z แทนค่า จากลักษณะการทำงานของวงจร เราสามารถที่จะเขียนโปรแกรม VHDL บรรยายพฤติกรรม การทำงานของวงจรดังกล่าวได้ดังนี้

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 ----------------------------------------------
4 ENTITY tri_state IS
5     PORT ( ena: IN STD_LOGIC;
6            input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
8 END tri_state;
9 ----------------------------------------------
10 ARCHITECTURE behave OF tri_state IS
11 BEGIN
12    output <= input WHEN (ena='0') ELSE
13               (OTHERS => 'Z');
14 END behave;
15 ----------------------------------------------
```



รูปที่ 4.5 ผลการจำลองตัวอย่างที่ 4.4

ตัวอย่างที่ 4.5 การออกแบบวงจร Encoder



(ก)

x[7..0]	y[2..0]
00000001	000
00000010	001
00000100	010
00001000	011
00010000	100
00100000	101
01000000	110
10000000	111

(ข)

รูปที่ 4.6 (ก) สัญลักษณ์วงจร Encoder (ข) ตารางความจริง

คำอธิบาย คุณสมบัติของวงจร Encoder จะมีจำนวนขาอินพุท 2^n และเอาท์พุท m ขา จากรูปวงจรจากรูปที่ 4.6 เป็นสัญลักษณ์ของวงจร Encoder ที่จำนวนอินพุท 8 ขา และเอาท์พุท 3 ขา ลักษณะการทำงานของวงจร Encoder นี้ สามารถดูได้จากตารางความจริง จากพฤติกรรมการทำงาน ก็สามารถที่จะนำค่าจากตางความจริงมาเขียนเป็นโปรแกรมภาษา VHDL ได้สองรูปแบบดังนี้

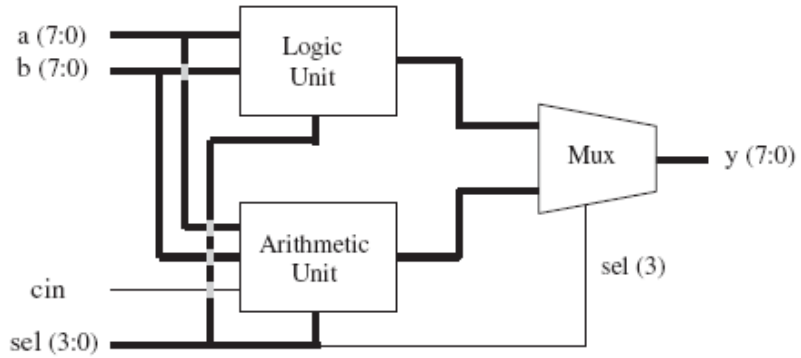
o รูปแบบการใช้ WHEN/ELSE

```
1 ---- Solution 1: with WHEN/ELSE -------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8 END encoder;
9 ---------------------------------------------
10 ARCHITECTURE encoder1 OF encoder IS
11 BEGIN
12    y <=  "000" WHEN x="00000001" ELSE
13          "001" WHEN x="00000010" ELSE
14          "010" WHEN x="00000100" ELSE
15          "011" WHEN x="00001000" ELSE
16          "100" WHEN x="00010000" ELSE
17          "101" WHEN x="00100000" ELSE
18          "110" WHEN x="01000000" ELSE
19          "111" WHEN x="10000000" ELSE
20          "ZZZ";
21 END encoder1;
22 ---------------------------------------------
```

o รูปแบบการใช้ WITH/SELECT/WHEN

```
1 ---- Solution 2: with WITH/SELECT/WHEN ------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8 END encoder;
9 ---------------------------------------------
10 ARCHITECTURE encoder2 OF encoder IS
11 BEGIN
12    WITH x SELECT
13          y <=  "000" WHEN "00000001",
14                "001" WHEN "00000010",
15                "010" WHEN "00000100",
16                "011" WHEN "00001000",
17                "100" WHEN "00010000",
18                "101" WHEN "00100000",
19                "110" WHEN "01000000",
20                "111" WHEN "10000000",
21                "ZZZ" WHEN OTHERS;
22 END encoder2;
23 ---------------------------------------------
```

<u>ตัวอย่างที่ 4.5</u> การออกแบบหน่วยประมวลทางคณิตศาสตร์และลอจิก (ALU)



รูปที่ 4.8 (ก)โครงสร้างการทำงานของ ALU

คำอธิบาย จากรูปที่ 4.8(ก) เป็นหน่วยประมวลทางคณิตศาสตร์และลอจิกขนาด 8 บิต ซึ่งจะประกอบด้วยกันสองหน่วยคือหน่วยกระทำทางด้านคณิตศาสตร์ และหน่วยกระทำทางด้านลอจิก การกระทำของแต่ละหน่วยจะถูกควบคุมจากตัวเลือกทีหนึ่ง ดังตารางการทำงานในรูปที่5.8(ข) แล้วส่งออกที่เอาท์พุท ในการกระทำกันทางด้านคณิศาสตร์ จะเป็นกระทำแบบไม่คิดเครื่องหมาย ดังนั้นจะทำการเรียกใช้ package ชื่อ std_logic_unsigned จาก LIBRARY ieee ดังนั้นเราก็สามารถที่จะเขียนพฤติกรรมการทำงานของ หน่วยประมวลทางคณิตศาสตร์และลอจิก ได้ดังนี้

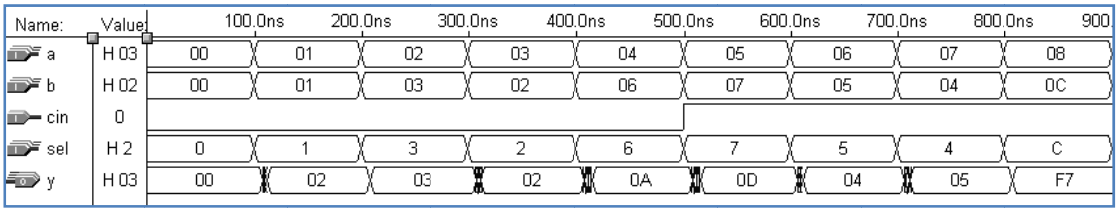
sel	Operation	Function	Unit
0000	y <= a	Transfer a	Arithmetic
0001	y <= a+1	Increment a	
0010	y <= a-1	Decrement a	
0011	y <= b	Transfer b	
0100	y <= b+1	Increment b	
0101	y <= b-1	Decrement b	
0110	y <= a+b	Add a and b	
0111	y <= a+b+cin	Add a and b with carry	
1000	y <= NOT a	Complement a	Logic
1001	y <= NOT b	Complement b	
1010	y <= a AND b	AND	
1011	y <= a OR b	OR	
1100	y <= a NAND b	NAND	
1101	y <= a NOR b	NOR	
1110	y <= a XOR b	XOR	
1111	y <= a XNOR b	XNOR	

(ข)

รูปที่ 4.8 (ข) ตารางการทำงาน

```
1 ----------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_unsigned.all;
5 ----------------------------------------------
6 ENTITY ALU IS
7     PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
9           cin: IN STD_LOGIC;
10          y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END ALU;
12 ----------------------------------------------
13 ARCHITECTURE dataflow OF ALU IS
14 SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16 ----- Arithmetic unit: ------
17    WITH sel(2 DOWNTO 0) SELECT
18          arith <= a   WHEN "000",
19                   a+1 WHEN "001",
20                   a-1 WHEN "010",
21                   b   WHEN "011",
22                   b+1 WHEN "100",
23                   b-1 WHEN "101",
24                   a+b WHEN "110",
25                   a+b+cin WHEN OTHERS;
26 ----- Logic unit: -----------
27    WITH sel(2 DOWNTO 0) SELECT
28          logic <= NOT a WHEN "000",
29                   NOT b WHEN "001",
30                   a AND b WHEN "010",
31                   a OR b WHEN "011",
32                   a NAND b WHEN "100",
33                   a NOR b WHEN "101",
34                   a XOR b WHEN "110",
35                   NOT (a XOR b) WHEN OTHERS;
```

```
36 -------- Mux: ---------------
37    WITH sel(3) SELECT
38          y <= arith WHEN '0',
39               logic WHEN OTHERS;
40 END dataflow;
41 ----------------------------------------------
```



รูปที่ 4.9 ผลการจำลองตัวอย่างที่ 5.5

4.2 ชุดคำสั่ง GENERATE

เป็นชุดคำสั่งที่มีลักษณะการทำงานแบบวนรอบซ้ำๆ ภายใต้เงื่อนไขกำหนดที่ต้องเป็นจริงเท่านั้น ซึ่งมักจะมีการใช้งานอยู่ 2 รูปแบบด้วยกันคือ

- FOR/GENERATE
- IF/GENERATE

4.2.1 รูปแบบ FOR/GENERATE

เป็นลักษณะการใช้ร่วมกับคำสั่ง FOR โดยมีกฎเกณฑ์การเขียนดังนี้

```
[label] : FOR identifier IN range GENERATE
            (concurrent assignment)
          END GENERATE;
```

o เมื่อ identifier เป็นตัวแปรที่ใช้คอยตรวจสอบค่า อยู่ในช่วงที่กำหนด(range)ไว้หรือไม่ ถ้าจริง จะกระทำในส่วนของการส่งผ่านค่าแบบแข่งขนาน โดยค่า identifier นี้จะทำการเพิ่มทีละหนึ่งไปเรื่อยๆจนกว่าเงื่อนไขจะเป็นเท็จจึงจะหยุด ส่วนป้ายชื่อ label จะบอกถึงตำแหน่งของชุดคำสั่ง FOR/GENERATE (ในกณีที่มีอยู่หลายชุด) ซึ่งจะมีหรือไม่มีก็ได้ ตัวอย่างลักษณะการเขียน FOR/GENERATE เช่น

```
1  ARCHITECTURE behav OF ex_U IS
2  SIGNAL x: BIT_VECTOR (7 DOWNTO 0);
3  SIGNAL y: BIT_VECTOR (15 DOWNTO 0);
4  SIGNAL z: BIT_VECTOR (7 DOWNTO 0);
5   ...
6   G1: FOR i IN x'RANGE GENERATE
7       z(i) <= x(i) AND y(i+8);
8       END GENERATE;
9    ...
10   OK: FOR i IN 0 TO 7 GENERATE
11       output(i)<='1' WHEN (z(i) AND y(i+1))='1' ELSE '0';
12       END GENERATE;
13   ...
14   END behave;
```

4.2.2 รูปแบบ IF/GENERATE

เป็นลักษณะการใช้ร่วมกับคำสั่ง IF ซึ่งการใช้งานลักษณะแบบนี้จะต้องกระทำอยู่ภายในส่วนของ FOR/GENERATE อีกทีหนึ่ง โดยมีกฎเกณฑ์การเขียนดังนี้

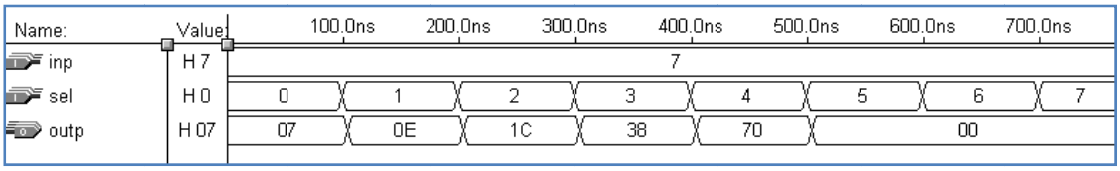
```
[label_1] : FOR identifier IN range GENERATE
            ...
          [label_2] : IF condition GENERATE
                      (concurrent assignment)
                      END GENERATE label_2;
          ...
           END GENERATE label_1;
```

<u>ตัวอย่างที่ 4.6</u> การออกแบบวงจรเลื่อนข้อมูล (vector shifter) โดยใช้ชุดคำสั่ง GENERATE

คำอธิบาย ต้องการออกแบบวงจรเลื่อนข้อมูลขนาด 4 บิต โดยเลื่อนไปทางซ้ายทีละบิตพร้อมทั้งเพิ่ม '0' เข้าไปในบิตสุดท้ายทุกๆครั้งที่มีการเลื่อน สมมติว่ามีข้อมูลมีค่าเป็น "1111" ดังนั้น ลักษณะลำดับการเลื่อนข้อมูลมีรายละเอียดดังนี้

row(0) = "00001111"
row(1) = "00011110"
row(2) = "00111100"
row(3) = "01111000"
row(4) = "11110000"

เมื่อทราบพฤติกรรมการทำงานแล้ว เราก็สามารถที่จะเขียนโปรแกรม VHDL โดยกำหนดอินพุทมีขนาด 4 บิต ส่วนเอาท์พุทมีขนาด 8 บิต และตัวควบคุมการเลื่อนข้อมูลจำนวน 5 ครั้ง (0 ถึง 4) ดังมีรายละเอียดการเขียนโปรแกรมดังต่อไปนี้



รูปที่ 4.10 ผลการจำลองตัวอย่างที่ 4.5

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 ENTITY shifter IS
4     PORT ( inp: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
5            sel: IN INTEGER RANGE 0 TO 4;
6            outp: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
7 END shifter;
8 ------------------------------------------------
9 ARCHITECTURE shifter OF shifter IS
10 SUBTYPE vector IS STD_LOGIC_VECTOR (7 DOWNTO 0);
11 TYPE matrix IS ARRAY (4 DOWNTO 0) OF vector;
12 SIGNAL row: matrix;
13 BEGIN
13    row(0) <= "0000" & inp;
14    G1: FOR i IN 1 TO 4 GENERATE
15          row(i) <= row(i-1)(6 DOWNTO 0) & '0';
16    END GENERATE;
17    outp <= row(sel);
18 END shifter;
```

4.3 โจทย์ปัญหา

1. ให้เขียนโปรแกรมของวงจร Multiplexer เข้า 8 ออก 1 โดยในแต่ละอินพุทมีขนาด 2 บิต
2. ให้เขียนโปรแกรมวงจร Priority Encoder ที่มีการทำงานดังตารางต่อไปนี้

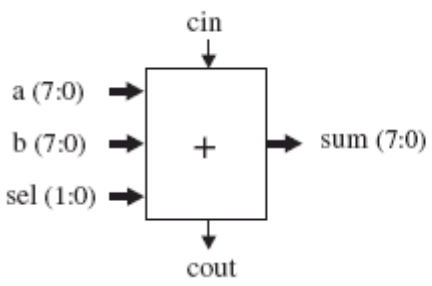
A[7..0]	W[3..0]
0000000	XXX0
00000001	0001
0000001X	0011
000001XX	0101
00001XXX	0111
0001XXXX	1001
001XXXXX	1011
01XXXXXX	1101
1XXXXXXX	1111

X=don't care

3. ให้เขียนโปรแกรมวงจร Binary decoder ดังตารางดังต่อไปนี้

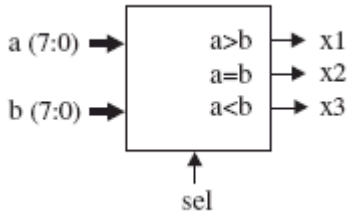
A[2..0]	Y[7..0]
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

4. ให้เขียนโปรแกรมบวกเลข ซึ่งมีการกระทำดังตารางต่อไปนี้



sel	operation
00	add unsigned
01	add signed
10	sub unsigned
11	sub signed

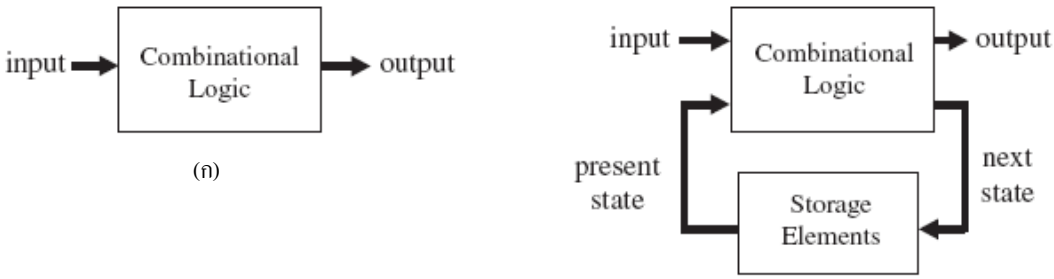
5. ให้เขียนโปรแกรมเปรียบเทียบ ดังรูปด้านล่างนี้



5 ชุดคำสั่งชนิดเรียงลำดับ

Sequential Codes

ชุดคำสั่งชนิดเรียงลำดับ (sequential code) เป็นชุดคำสั่งที่ถูกนำไปใช้สำหรับการออกแบบวงจรซีเควียนเชียล (sequential circuit) วงจรประเภทนี้จะแตกต่างจากวงจรคอมไบเนชั่น (ที่ผ่านมาเป็นเรากล่าวถึงวงจรประเภทนี้ทั้งหมด) กล่าวคือเอาท์พุทของวงจรประเภทนี้จะเปลี่ยนแปลงตามเหตุการณ์ของอินพุทในอดีตที่ผ่านมาและสภาวะการทำงานปัจจุบันของวงจร(present state)ดังนั้นวงจรประเภทนี้จะต้องมีหน่วยความจำสำหรับเก็บสภาวะการทำงานตลอดเวลา ซึ่งในส่วนของวงจรคอมไบเนชั่นจะไม่มีส่วนนี้ ดังแสดงในรูปที่ 5.1



(ก) (ข)

รูปที่ 5.1(ก) วงจรคอมไบเนชั่น และ (ข) วงจรซีเควียนเชียล

เมื่อนำชุดคำสั่งชนิดเรียงลำดับทีละคำสั่งมาเขียนในภาษา VHDL นั้น คำสั่งที่นำมาใช้จะต้องเขียนอยู่ภายในชุดคำสั่ง สามประเภท

- PROCESS
- FUNCTION
- PROCEDURE

ซึ่งภายในชุดคำสั่งเหล่านี้เราสามารถเรียกใช้คำสั่งจำพวก IF WAIT CASE และ LOOP ได้ แต่ในบทนี้จะขอนำเสนอการใช้งานเฉพาะชุดคำสั่ง PROCESS เท่านั้น ซึ่งเป็นชุดคำสั่งที่ถูกงานบ่อย

5.1 ชุดคำสั่ง PROCESS

คำสั่งประเภทนี้เป็นชุดคำสั่งพื้นฐานที่ถูกใช้สำหรับคำสั่งชนิดเรียงลำดับทีละคำสั่ง โดยมีกฎเกณฑ์การเขียนดังนี้

```
[label:] PROCESS (sensitivity list)
[VARIABLE variable_name : type [range] [:= initial_value;]]
BEGIN
(sequential code)
END PROCESS [label];
```

ชุดคำสั่ง PROCESS นี้ เวลานำไปใช้งานจะต้องระบุตัวกระตุ้นการทำงาน (sensitivity list) โดยตัวกระตุ้นการทำงานอาจมีหลายตัวหรือตัวเดียว ขึ้นอยู่พฤติกรรมของวงจรนั้นๆ เมื่อตัวกระตุ้นมีการเปลี่ยนแปลงเหตุการณ์หรือเปลี่ยนระดับสัญญาณ(อาจตัวเดียวหรือหลายตัว) ก็จะส่งผลทำให้คำสั่งภายใน PROCESS ทำงาน ตามลำดับบรรทัดของคำสั่งที่เขียนไว้ในส่วนของ sequential code

ชุดคำสั่งประเภทนี้ อาจจะมีการระบุป้ายชื่อ label ไว้ ในกรณีที่มีการใช้งานชุดคำสั่ง PROCESS หลายชุด และภายใต้ชุดคำสั่งประเภทนี้ อาจจะมีตัวแปรประเภท VARIABLE ด้วย (ขอยกไปอธิบายในหัวข้อ 5.3) เวลาสิ้นสุดการใช้งานชุดคำสั่ง PROCESS จะต้องมีคำว่า END PROCESS ตามด้วยป้ายชื่อ label (ในกรณีที่มีป้ายชื่อกำกับไว้)

ในบทนี้จะขอกล่าวถึงชุดคำสั่ง PROCESS นี้สามารถเรียกใช้คำสั่ง IF WAIT CASE และ LOOP ที่อยู่ระหว่าง BEGIN และ END PROCESS หรือ sequential code

5.2 ตัวแปรประเภท VARIABLE และ SIGNAL

ตัวแปรประเภท SIGNAL เป็นตัวแปรอีกประเภทหนึ่งของสัญญาณ ที่สามารถกำหนดค่าให้สัมพันธ์กับเวลา (สัญญาณนาฬิกา) กล่าวคือตัวแปรประเภทนี้สามารถรับค่าได้เพียงค่าเดียวในขณะเวลานั้นๆ และค่าของตัวแปรประเภทนี้สามารถเรียกใช้หรือถูกนำไปใช้ได้ทั้งตัวโปรแกรม ส่วนการส่งผ่านค่าของตัวแปรประเภทนี้จะใช้เครื่องหมาย **<=** โดยมีกฎเกณฑ์การเขียนดังนี้

```
[SIGNAL signal_name : type [range] [:= initial_value;]]
```

ตัวแปรประเภท VARIABLE ก็เป็นตัวแปรอีกประเภทหนึ่งที่สามารถกำหนดค่าให้ได้ แต่ค่านี้สามารถเปลี่ยนได้ตลอดเวลา กล่าวคือการเก็บค่าของตัวแปรประเภทนี้ ไม่ได้ขึ้นอยู่กับเวลาในขณะนั้น ตัวแปรชนิดนี้จะต้องประกาศไว้ใต้ชุดคำสั่ง PROCESS การเรียกใช้งานตัวแปรประเภทนี้จะสามารถเรียกใช้ได้เฉพาะในส่วนที่ถูกประกาศไว้เท่านั้น นั้นก็คือใช้ได้เฉพาะในส่วนของ sequential code เท่านั้น ส่วนการส่งผ่านค่าของตัวแปรประเภทนี้จะใช้เครื่องหมาย **:=** โดยมีกฎเกณฑ์การเขียนดังนี้

```
[VARIABLE variable_name : type [range] [:= initial_value;]]
```

จากกฎเกณฑ์การเขียนทั้งในส่วนของ VARIABLE และ SIGNAL ค่า variable_name หรือ signal_name เป็นชื่อตัวแปรประเภท VARIABLE และ SIGNAL ตามลำดับ ส่วน type เป็นประเภทของข้อมูลที่ตัวแปรระบุการใช้งาน และ initial_value เป็นการกำหนดค่าเริ่มต้นในการใช้งาน ดังแสดงในตัวอย่างการใช้งานต่อไปนี้

```
1   ----------EXAMPLE SIGNAL----------------------------
2   SIGNAL  sel : BIT_VECTOR (2 DOWNTO 0) := “000”;
3   SIGNAL  count,temp : INTEGER RANGE 0 TO 59;
4   ...
5     count <= count + 1 AFTER 5 ns;
6     temp <= temp – 1 AFTER 10 ns;
7   ...
8   -----------EXAMPLE VARIABLE--------------------------
9   VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := “0000”;
10  VARIABLE freq,direction : INTEGER;
11  ...
12    freq := clk + direction;
13  ...
14  ----------------------------------------------------
```

5.3 การใช้คำสั่ง IF ในชุดคำสั่ง PROCESS

คำสั่ง IF เป็นคำสั่งที่มีไว้คอยตรวจสอบเงื่อนไข (เหมือนคำสั่ง WHEN)เพื่อคอยตัดใจกระทำบางสิ่งบางอย่างเมื่อเงื่อนเป็นจริง โดยมีกฎเกณฑ์การเขียนดังนี้

```
IF conditions THEN assignments;
ELSIF conditions THEN assignments;
...
ELSE assignments;
END IF;
```

เมื่อมีการใช้คำสั่ง IF ที่มีเงื่อนไขการตัดสินใจ 2 ทาง จะไม่มีการใช้ ELSIF ดังตัวอย่างต่อไปนี้

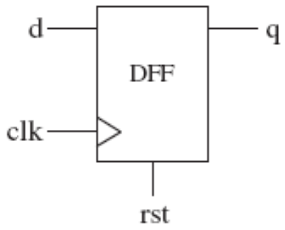
```
1   ------- IF/THEN/ELSE----------------
2   VARIABLE count1,count2 : STD_LOGIC_VECTOR( 3 DOWNTO 0);
3   ... PROCESS(clk)
4   ...
5     IF (clk = ‘1’) then
6           count1 := count1 + 1;
7     ELSE
8           count2 := count2 – 1;
9     END IF;
10  ...
11  -----------------------------------
```

เมื่อมีการใช้คำสั่ง IF ที่มีเงื่อนไขการตัดสินใจมากกว่า 2 ทาง จะมีการใช้ ELSIF ดังตัวอย่างต่อไปนี้

```
1   ------- IF/THEN/ELSIF/ELSE----------------
2   VARIABLE cnt1,cnt2 : STD_LOGIC_VECTOR( 3 DOWNTO 0);
3   VARIABLE sel : STD_LOGIC_VECTOR( 1 DOWNTO 0);
4   ... PROCESS(clk,sel)
5   ...
6   IF sel = “00” then  count1 := count1 + 1;
7   ELSIF sel = “01” then  count1 := count1 – 1;
8   ELSIF sel = “10” then  count2 := count2 + 1;
9   ELSE  count2 := count2 – 1;
10  END IF;
11  ...
12  -----------------------------------
```

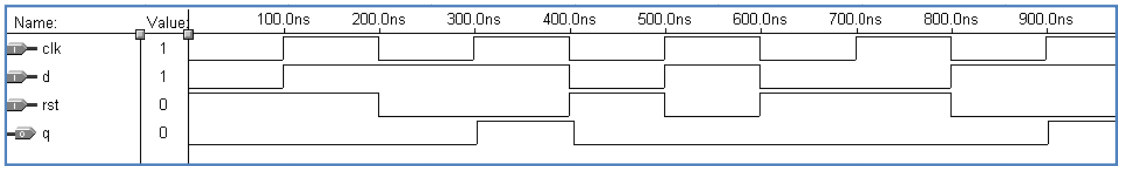
ตัวอย่างที่ 5.1 การออกแบบวงจร ดีฟลิบฟลอบ เมื่อขารีเซตทำงานโดยไม่สนใจสัญญาณนาฬิกา

คำอธิบาย จากรูปเป็นสัญลักษณ์ของวงจรดีฟลิบฟลอบ ถ้าต้องการให้ขารีเซตของฟลิบฟลอบนี้ทำงานทุกครั้งเมื่อมีการกระตุ้น(โดยไม่สนใจสัญญาณนาฬิกา) เราสามารถที่เขียนบรรยายพฤติกรรมการทำงาน โดยใช้ชุดคำสั่ง PROCESS ร่วมด้วย IF โดยที่ตัวกระตุ้นที่ใช้ในส่วนของชุดคำสั่ง PROCESS จะประกอบด้วย clk และ rst ดังนี้



รูปที่ 5.1 สัญลักษณ์ของวงจรดีฟลิบฟลอบ

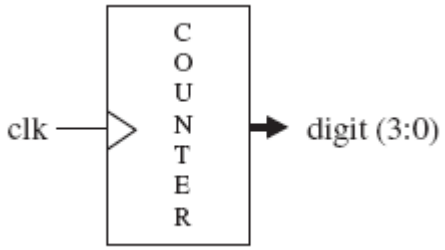
```
1 ---------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8 END dff;
9 ---------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12 PROCESS (clk, rst)
13    BEGIN
14          IF (rst='1') THEN
15              q <= '0';
16          ELSIF (clk'EVENT AND clk='1') THEN
17              q <= d;
18          END IF;
19 END PROCESS;
20 END behavior;
21 ---------------------------------------
```



รูปที่ 5.2 ผลการจำลอง ตัวอย่างที่ 5.1

ตัวอย่างที่ 5.2 การออกแบบวงจรนับ 0 ถึง 9 ขนาด 1 หลัก

คำอธิบาย เป็นการออกแบบวงจรนับ 1 หลัก ฐานสิบ ซึ่งมีการนับตั้งแต่ 0 ถึง 9 แล้วย้อนกลับมานับใหม่ ดังนั้น เอาท์พุทของวงจรจะมีขนาด 4 บิต (9_{10} = 1001_2) ส่วนอินพุทมีเฉพาะสัญญาณนาฬิกาเท่านั้น (ดังแสดงในบรรทัดที่ 5 ถึง 8)โดยสัญญาณนาฬิกาจะเป็นตัวกระตุ้นในชุดคำสั่ง PROCESS ทำงาน พร้อมทั้งประกาศตัวแปร temp เป็นตัวแปรประเภท VARIABLE ที่มีไว้สำหรับเพิ่มค่าการนับเมื่อสัญญาณนาฬิกามีการเปลี่ยนแปลง บรรทัดที่ 20 เป็นการส่งผ่านค่าจากตัวแปร temp ออกมายังภายนอกชุดคำสั่ง PROCESS โดยที่มีตัวแปรเอาท์พุท digit คอยรอรับค่าสัญญาณ



รูปที่ 5.3 รูปสัญลักษณ์ของวงจรนับ ขนาด 1 หลัก

```
1 ---------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY counter IS
6     PORT (clk : IN STD_LOGIC;
7           digit : OUT INTEGER RANGE 0 TO 9);
8 END counter;
9 ---------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    count: PROCESS(clk)
13    VARIABLE temp : INTEGER RANGE 0 TO 10;
14    BEGIN
15          IF (clk'EVENT AND clk='1') THEN
16                temp := temp + 1;
17                IF (temp=10) THEN temp := 0;
18                END IF;
19          END IF;
20          digit <= temp;
21    END PROCESS count;
22 END counter;
23 ---------------------------------------------
```

5.4 การใช้คำสั่ง WAIT ในชุดคำสั่ง PROCESS

คำสั่ง WAIT เมื่อนำมาภายใต้ชุดคำสั่ง PROCESS จะทำหน้าที่คอยควบคุมการทำงานตัวกระตุ้น กล่าวคือ ในชุดคำสั่ง PROCESS หนึ่งชุด สามารถที่จะมีตัวกระตุ้นได้หลายตัว ถ้าตัวกระตุ้นตัวใดตัวหนึ่งมีการเปลี่ยนแปลงเกิดขึ้น คำสั่งหรือกระบวณการทำงานที่อยู่ใต้ชุดคำสั่ง PROCESS จะมีการทำงานแบบลำดับทีละบรรทัดจนจบ ซึ่งในบางบรรทัดอาจจะไม่ต้องการเปลี่ยนแปลงค่า เนื่องจากไม่ได้เกิดจากตัวกระตุ้นในส่วนที่ต้องการ จุดนี้จึงเป็นข้อเสียของชุดคำสั่ง PROCESS ที่มีตัวกระตุ้นหลายตัวแต่ไม่อาจสามารถควบคุมได้ ดังนั้นคำสั่ง WAIT จึงถูกนำมาใช้ระงับการทำงานหรือกระบวณการทำงานจนกว่าจะมีตัวกระตุ้นที่ต้องการ มีการเปลี่ยนแปลงเกิดขึ้น การทำงานจึงจะถูกทำงานต่อไป

การใช้งานคำสั่ง WAIT นี้สามารถกำหนดตำแหน่งได้หลายๆตำแหน่ง ขึ้นอยู่กับความต้องการของผู้เขียน โดยมีลักษณะการใช้งานอยู่สามรูปแบบ ดังต่อไปนี้

5.4.1 รูปแบบ WAIT ON

เป็นคำสั่งที่หยุดรอ จนกว่าจะมีตัวกระตุ้นที่ต้องการ โดยมีกฎการเขียนและตัวอย่างดังนี้

```
WAIT ON signal1,[signal2,...];
```

```
1   ---------------------------------------------
2   PROCESS (clk,rst,sel)
3   BEGIN
4   WAIT ON clk, rst;
5   IF (rst='1') THEN
6      output <= "00000000";
7   ELSIF (clk'EVENT AND clk='1') THEN
8      output <= input;
9   END IF;
10 END PROCESS;
   11 ---------------------------------------------
```

5.4.2 รูปแบบ WAIT UNTIL

เป็นคำสั่งที่หยุดรอ จนกว่าเงื่อนไขเป็นจริง โดยมีกฎการเขียนดังนี้

```
WAIT UNTIL condition;
```

5.4.3 รูปแบบ WAIT FOR

เป็นคำสั่งที่หยุดรอ จนกว่าจะได้เวลาตามที่ต้องการ โดยมีกฎการเขียนดังนี้

```
WAIT FOR time;
```

```
1   ---------------------------------------------------
2   PROCESS      -- no sensitivity list
3   BEGIN
4     WAIT UNTIL (clk'EVENT AND clk='1');
5     IF (rst='1') THEN
6           WAIT FOR 10 ns;
7           output <= "00000000";
8     ELSIF (clk'EVENT AND clk='1') THEN
9           WAIT FOR 10 ns;
10          output <= input;
11    END IF;
12  END PROCESS;
13  ---------------------------------------------------
```

ตัวอย่างที่ 5.4 การออกแบบวงจร ดีฟลิบฟลอบ เมื่อขารีเซตทำงานโดยไม่สนใจสัญญาณนาฬิกา ครั้งที่2

คำอธิบาย จากตัวอย่างที่ 5.1 จะทำการดัดโปรแกรม โดยนำเอาชุดคำสั่ง WAIT ON เข้าไปเขียนแทนดังนี้

```
1 --------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 --------------------------------------
5 ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8 END dff;
9 --------------------------------------
10 ARCHITECTURE dff OF dff IS
11 BEGIN
12    PROCESS
13    BEGIN
14          WAIT ON rst, clk;
15          IF (rst='1') THEN
16                q <= '0';
17          ELSIF (clk'EVENT AND clk='1') THEN
18          q <= d;
19          END IF;
20    END PROCESS;
21 END dff;
22 --------------------------------------
```

ตัวอย่างที่ 5.5 การออกแบบวงจรนับ 0 ถึง 9 ขนาด 1 หลัก ครั้งที่ 2

คำอธิบาย จากตัวอย่างที่ 5.2 จะทำการดัดแปลงโปรแกรม โดยนำเอาชุดคำสั่ง WAIT UTIL เข้าไปเขียนแทนดังนี้

```
1 ---------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY counter IS
6     PORT (clk : IN STD_LOGIC;
7           digit : OUT INTEGER RANGE 0 TO 9);
8 END counter;
9 ---------------------------------------------
```

```
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12 PROCESS -- no sensitivity list
13    VARIABLE temp : INTEGER RANGE 0 TO 10;
14    BEGIN
15          WAIT UNTIL (clk'EVENT AND clk='1');
16          temp := temp + 1;
17          IF (temp=10) THEN temp := 0;
18          END IF;
19          digit <= temp;
20    END PROCESS;
21 END counter;
22 ---------------------------------------------
```

จากลักษณะคำสั่ง WAIT ทั้งสามรูปแบบคือ ON WAIT และ FOR เราสามารถที่จะนำทั้งสามรูปแบบมาผสมกันได้ เพื่อให้เกิดประสิทธิภาพต่อการใช้งาน ได้ดังนี้

- การใช้ ON ร่วมกับคำสั่ง UTIL เป็นการหยุดรอ จนกว่าจะมีตัวกระตุ้นที่ต้องการ <u>และ</u> เงื่อนไขเป็นจริง จึงจะทำงานต่อไป
- การใช้ ON ร่วมกับคำสั่ง FOR เป็นการหยุดรอ จนกว่าจะมีตัวกระตุ้นที่ต้องการ <u>หรือ</u> ได้เวลาตามที่ต้องการ จึงจะทำงานต่อไป
- การใช้งานคำสั่ง UTIL ร่วมกับคำสั่ง FOR เป็นการหยุดรอ จนกว่าเงื่อนไขเป็นจริง <u>หรือ</u> ได้เวลาตามที่ต้องการ จึงจะทำงานต่อไป

5.5 การใช้คำสั่ง CASE ในชุดคำสั่ง PROCESS

ลักษณะการทำงานของคำสั่ง CASE จะมีการใช้งานร่วมกับคำสั่ง WHEN ดังนั้นลักษณะการทำงานจะใกล้เคียงกันกับคำสั่ง WITH/SELECT/WHEN (หัวข้อ 4.3) ซึ่งคำสั่ง CASE ก็เป็นคำสั่งที่ทำหน้าเลือกช่องทางปฏิบัติที่ตรงตามเงื่อนไขที่กำหนดให้ โดยมีกฎการเขียนดังนี้

```
CASE identifier IS
      WHEN  value => assignment1, assignment2;
      WHEN  value => assignment1, assignment2;
      ...
END CASE;
```

จากกฎการเขียนของคำสั่ง CASE เมื่อ identifier เป็นตัวแปรที่คอยตรวจสอบค่า ถ้าตรงกันกับค่า value ตัวใดตัวหนึ่งหรือไม่ ก็จะกระทำการในส่วนของการส่งผ่าน(assignment)นั้นๆ โดยในส่วนของการส่งผ่านค่าอาจจะมีการส่งผ่านค่าหลายๆตัวก็ได้ และคำสั่ง CASE นี้สามารถเรียกใช้งานภายใต้ชุดคำสั่ง PROCESS เดียวกันได้หลายๆครั้ง ดังแสดงในตัวอย่างย่อยต่อไปนี้

```
1 --- sequential code --------
2 ----CASE/WHEN---------------
3 CASE control IS
4     WHEN "00" => x<=a; y<=b;
5     WHEN "01" => x<=b; y<=c;
6     WHEN OTHERS => x<="0000";
7                    y<="ZZZZ";
8 END CASE;
------------------------------
```

```
1------  concurrent code ---------
2----------WITH/SELECT/WHEN-------
3 WITH control SELECT
4           x <= a WHEN "00",
5          b WHEN "01",
6     "0000" WHEN OTHERS;
7  ...
```

ในการใช้งานคำสั่งระหว่าง CASE/WHEN กับ WITH/SELECT/WHEN จะมีข้อแตกต่างกัน ดังตารางที่ 5.1 เพราะฉะนั้นจึงต้องทำความเข้าใจถึงข้อแตกต่างก่อนนำไปใช้งาน

ตารางที่ 5.1 ความแตกต่างระหว่าง CASE/WHEN กับ WITH/SELECT/WHEN

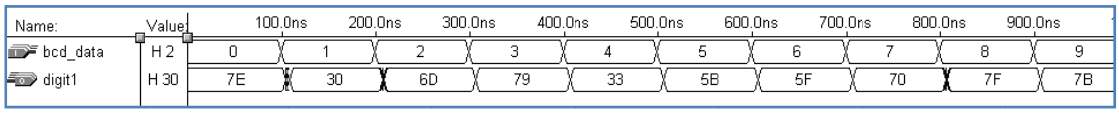
	WITH/SELECT/WHEN	CASE/WHEN
-ลักษณะคำสั่ง	เป็นชุดคำสั่งชนิดแข่งขนาน	เป็นชุดคำสั่งชนิดเรียงลำดับทีละคำสั่ง
-การใช้งาน	ใช้งานนอกชุดคำสั่ง PROCESS FUNCTION และ PROCEDURE เท่านั้น	ใช้งานภายในชุดคำสั่ง PROCESS FUNCTION และ PROCEDURE เท่านั้น
-การส่งผ่านค่า	ส่งผ่านค่าได้ตัวแปรเดียว	ส่งผ่านค่าได้มากกว่าหนึ่งตัวแปร
-จำนวนครั้งที่ถูกใช้	ใช้ได้ เพียงครั้งเดียว	ใช้ได้ไม่จำกัดจำนวนครั้ง

ตัวอย่างที่ 5.6 การออกแบบวงจรขับส่วนแสดงผลหลอดเจ็ดส่วน (7-segment display)

คำอธิบาย วงจรที่ทำหน้าที่ทำให้หลอดแสดงผลเจ็ดส่วนทำงาน ให้แสดงผลเป็นตัวเลขตั้งแต่ 0 ถึง 9 เราเรียกว่าวงจร BCD to seven segment display ดังรูปที่ 5.4 เราเรียกวงจรลักษณะนี้ว่าวงจรถอดรหัส (Decoder) ส่วนตัวแสดงผลเจ็ดส่วนนี้ เมื่อนำมาใช้งานจะต้องทราบประเภทของขาร่วม(common) โดยทั่วไปแล้วจะมีอยู่ประเภทคือ ขาร่วมแบบแอโนด(common anode) และขาร่วมแบบแคโทด(common cathode)หรือเรียกง่ายๆว่า ขาร่วมแบบบวก และขาร่วมแบบลบ ในตัวอย่างนี้จะขอใช้ขาร่วมแบบลบ ดังนั้นเราจะต้องส่งสัญญาณที่เป็นบวกหรือลอจิก '1' ให้กับส่วนแสดงผลเจ็ดส่วน ตั้งแต่ส่วน (segment)a ถึงส่วน g ซึ่งจะทำให้มีการแสดงเป็นตัวเลขออกในรูปที่ 5.4 การเขียนโปรแกรม VHDL นั้นส่วนอินพุทจะมี 4 ขาคือ และเอาท์พุทมีอยู่กัน 7 ขา (โดยไม่นับรวมขา dot) ดังแสดงในส่วนของ ENTITY ส่วนพฤติกรรมการทำงานของวงจรนี้จะต้องทราบว่าตัวเลขแต่ลัวที่แสดง เกิดจากส่วนบ้างเช่น ถ้าเป็นเลข 0 ก็จะป้อนสัญญาณลอจิก '1' ให้กับส่วนแสดงผลเจ็ดส่วนมีส่วน a ถึง f ซึ่งจะได้เป็น 111 1110 เป็นต้น ดังนั้นเราก็สามารถที่จะทำการเขียนการแสดงผลตัวเลขแต่ละตัวได้ ดังในบรรทัดที่ 16 ถึง 26

รูปที่ 5.4 ลักษณะของส่วนแสดงผลเจ็ดส่วน และ การแสดงเป็นตัวเลข

```
1 ---------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------------
5  ENTITY bcd_ssd IS
6     PORT (bcd_data : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
7           digit1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
8  END bcd_ssd;
9  ---------------------------------------------------
10 ARCHITECTURE behave OF bcd_sssd IS
11 BEGIN
12    PROCESS(bcd_data)
13    BEGIN
14 ---- BCD to seven segment display conversion: --------
15          CASE bcd_data IS
16                WHEN "0000" => digit1 <= "1111110"; --7E
17                WHEN "0001" => digit1 <= "0110000"; --30
18                WHEN "0010" => digit1 <= "1101101"; --6D
19                WHEN "0011" => digit1 <= "1111001"; --79
20                WHEN "0100" => digit1 <= "0110011"; --33
21                WHEN "0101" => digit1 <= "1011011"; --5B
22                WHEN "0110" => digit1 <= "1011111"; --5F
23                WHEN "0111" => digit1 <= "1110000"; --70
24                WHEN "1000" => digit1 <= "1111111"; --7F
25                WHEN "1001" => digit1 <= "1111011"; --7B
26                WHEN OTHERS => NULL;
27          END CASE;
28    END PROCESS;
29 END behave;
30 ---------------------------------------------------
```



รูปที่ 5.5 ผลการจำลองของตัวอย่างที่ 5.6

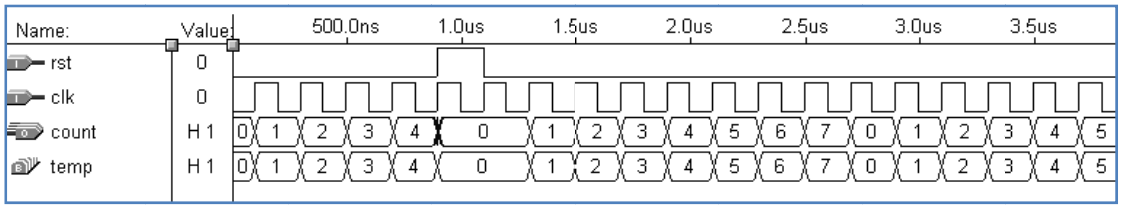
<u>ตัวอย่างที่ 5.7</u> การออกแบบวงจรนับ 0 ถึง 7

คำอธิบาย การออกแบบวงจรนับ 0 ถึง 7 หรือเป็นการออกแบบวงจรนับขนาด 3 บิต สามารถเขียนบรรยายพฤติกรรมการทำงานโดยใช้ชุดคำสั่ง ได้หลายวิธี ดังต่อไปนี้

```
1 ------ Solution 1: With a VARIABLE --------
2 ENTITY counter IS
3     PORT ( clk, rst: IN BIT;
4            count: OUT INTEGER RANGE 0 TO 7);
5 END counter;
6 ---------------------------------------------
7 ARCHITECTURE counter OF counter IS
8 BEGIN
9 PROCESS (clk, rst)
10    VARIABLE temp: INTEGER RANGE 0 TO 7;
11    BEGIN
12          IF (rst='1') THEN
13                temp:=0;
14          ELSIF (clk'EVENT AND clk='1') THEN
15                temp := temp+1;
```

```
16          END IF;
17          count <= temp;
18    END PROCESS;
19 END counter;
20 ---------------------------------------------
```

```
1 ------ Solution 2: With SIGNALS only -------
2 ENTITY counter IS
3     PORT ( clk, rst: IN BIT;
4            count: BUFFER INTEGER RANGE 0 TO 7);
5 END counter;
6 ---------------------------------------------
7 ARCHITECTURE counter OF counter IS
8 BEGIN
9     PROCESS (clk, rst)
10    BEGIN
11          IF (rst='1') THEN
12                count <= 0;
13          ELSIF (clk'EVENT AND clk='1') THEN
14                count <= count + 1;
15          END IF;
16    END PROCESS;
17 END counter;
18 ---------------------------------------------
```



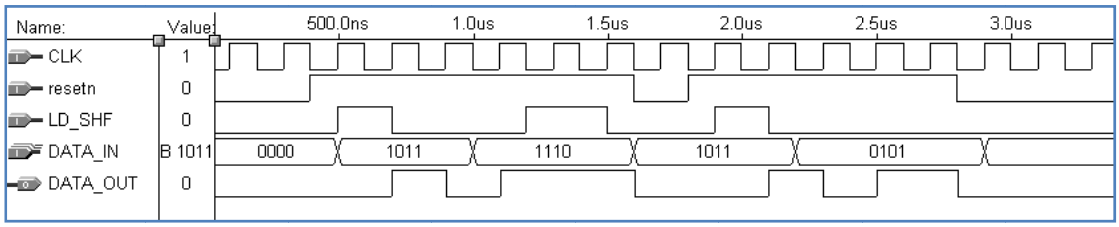
รูปที่ 5.6 ผลการจำลองของตัวอย่างที่ 5.7

<u>ตัวอย่างที่ 5.8</u> การออกแบบวงจรเลื่อนเก็บข้อมูลแบบ เข้าขนานออกอนุกรม (PISO)

คำอธิบาย วงจรเลื่อนเก็บข้อมูล แบบเข้าขนานออกอนุกรม ขนาด 8 บิต มีพฤติกรรมการทำงานคือ ขั้นตอนแรกคือต้องทำการนำ(load)เข้าข้อมูลมาเก็บไว้ก่อน หลังจากนั้นข้อมูลจะมีการเลื่อนไปทางซ้ายทีละบิต(เลื่อนไปทางบิตสูงสุด) พร้อมทั้งเติม '0'ลงในบิตต่ำสุดด้วย ดังนั้นเราก็สามารถเขียนเป็นชุดคำสั่งได้ดังนี้

```
1  ---------------------------------------------------
2  LIBRARY ieee;
3  USE iee.std_logic_1164.all;
4  ----------------------------------------------------
5  ENTITY piso IS
6  PORT(clk,resetn,ld_shf : in std_logic; --
clock,resetn,load/shift
7       data_in   : in std_logic_vector(4 downto 1);  --Data
input
8       data_out  : out std_logic                     --Data
output
9        );
10 END piso;
11 ----------------------------------------------------
```

```
12 ARCHITECTURE behavioral of piso IS
13 CONSTANT reset_active : std_logic := '0';
14 BEGIN
15    PROCESS(    clk, resetn)
16    VARIABLE temp_a,temp_b : std_logic_vector(4 downto 1);
17    BEGIN
18 -------clear output register-----------------------
19          IF(resetn = reset_active) THEN
20                temp_a := (OTHERS => '0');
21 ------ On rising edge of clock, load/shift in data
22          ELSIF (clk'EVENT and clk = '1') THEN
23 -------load data ----------------------------------
24                IF(ld_shf = '1') THEN
25                      temp_a := data_in;
26 ----------shift data------------------------------
27                ELSE
28                      temp_b := temp_a;
29                      temp_a (4) := temp_b (3);
30                      temp_a (3) := temp_b (2);
31                      temp_a (2) := temp_b (1);
32                      temp_a (1) := '0';
33                END IF;
34          END IF;
35          data_out <= temp_a(4);
36    END PROCESS;
37 END behavioral;
38 -----------------------------------------------------
```



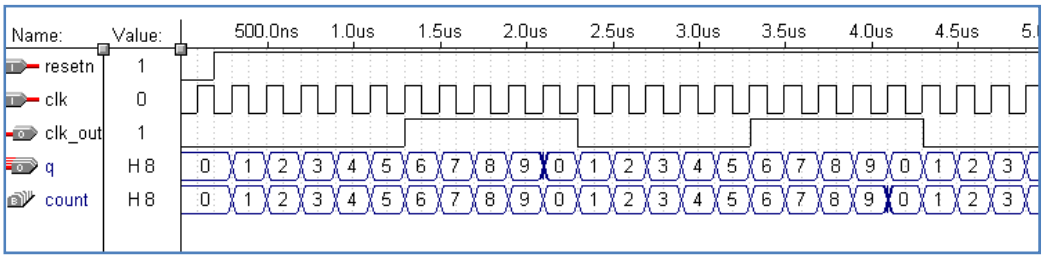
รูปที่ 5.7 ผลการจำลองของตัวอย่างที่ 5.8

ตัวอย่างที่ 5.9 การออกแบบวงจรหารความถี่ (Frequency divider)จาก 1Mhz เป็น 100 khz

คำอธิบาย วงจรหารความถี่ ถือว่าเป็นวงจรที่มีความสำคัญอีกอย่าง เนื่องจากความถี่ที่ผลิตจากตัวกำเนิดความถี่ (Osillator) ไม่ได้ตรงตามความต้องของผู้ใช้

หลักการของวงจรหารความถี่ ก็คือวงจรนับนั่นเอง โดยที่ค่าของการนับหาได้จาก
n = 1,000,000/100,000 = 10

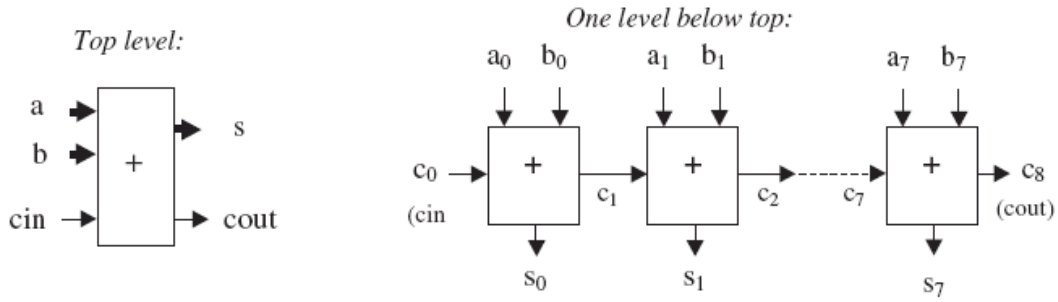
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.All;
3   USE ieee.std_logic_unsigned.All;
4
5   ENTITY freq_div IS
6   PORT(clk,resetn : IN STD_LOGIC;
7   clk_out : OUT STD_LOGIC;
8    q : OUT INTEGER RANGE 0 TO (10-1)     );
9   END freq_div;
10
11  ARCHITECTURE Behav OF freq_div IS
12  SIGNAL count : INTEGER RANGE 0 TO (10-1) := 0;
13  SIGNAL temp : STD_LOGIC;
14  BEGIN
15  PROCESS (clk,resetn)
16     BEGIN
17     IF (resetn = '0' OR count > (10-1)) THEN
18           count <= 0;
19     ELSIF (clk'EVENT AND clk = '1') THEN
20           count <= count + 1;
21           IF count > ((10/2)-1) THEN
22                 temp <= '1';
23           ELSE temp <= '0';
24           END IF;
25     END IF;
26           q <= count;
27           clk_out <= temp;
28     END PROCESS;
29 END Behav;
```



รูปที่ 5.8 ผลการจำลองของตัวอย่างที่ 5.9

5.6 โจทย์ปัญหา

1. ให้เขียนโปรแกรมสำหรับวงจรบวกเลขประเภท Carry Ripple Adder ขนาด 8 บิต แบบไม่คิดเครื่องหมาย ในแต่ละบิตที่ทำการบวกจะได้ผลลัพธ์ ดังนี้
   $s_j = a_j$ XOR b_j XOR c_j
   $c_{j+1} = (a_j$ AND $b_j)$OR$(a_j$ AND $c_j)$OR$(b_j$ AND $c_j)$
   กำหนดให้ใช้ชุดคำสั่ง GENERIC ร่วมกับ FOR/LOOP



2. จากตัวอย่างที่ 5.5 ให้ปรับปรุงตัวโปรแกรมเพื่อให้นับเลข 00 ถึง 99 ขนาด 2 หลัก
3. จากตัวอย่างที่ 5.6 ให้ปรับปรุงตัวโปรแกรมสำหรับขับส่วนแสดงผลหลอดเจ็ดส่วน โดยให้แสดงผลตั้งแต่เลข 0 ถึง F จำนวน 1 หลัก
4. จากตัวอย่างที่ 5.9 ให้ปรับปรุงตัวโปรแกรมเพื่อให้หารความถี่จากความถี่ 50hz ให้เหลือเพียง 1hz
5. ให้เขียนโปรแกรม เพื่อทำเป็นนาฬิกา 12 ชั่วโมง โดยใช้ความถี่ 50 hz (จากแหล่งจ่ายกำเนิดที่ได้มาจากการไฟฟ้าส่วนภูมิภาค)

6 ไฟไนต์สเตตแมชชีน

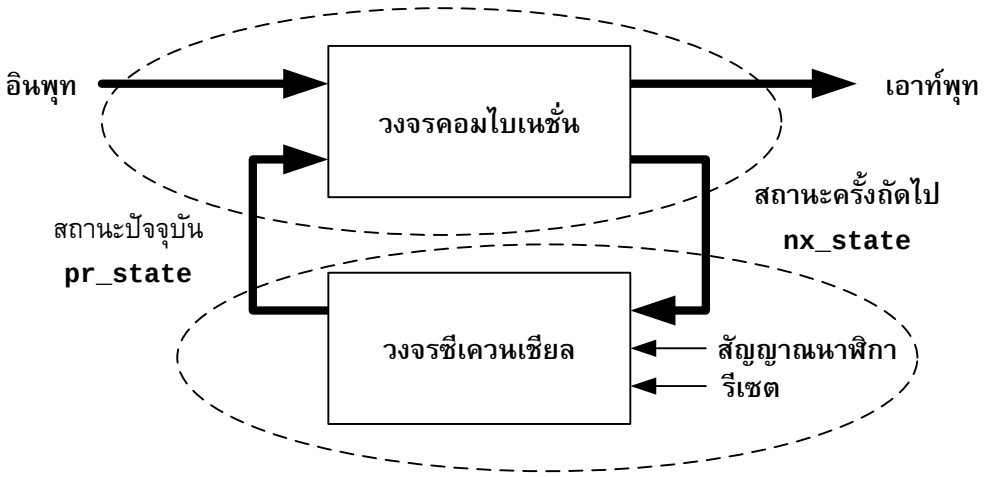
Finite–State Machine

การออกแบบวงจรดิจิตอลในรูปแบบ*ไฟไนต์สเตตแมชชีน*เป็นการออกแบบเชิงลำดับ ในลักษณะที่บ่งบอกถึงสถานะ(state)ของการทำงานและให้ค่าฟังก์ชั่นเอาท์พุท การออกแบบวงจรในรูปแบบนี้เหมาะสำหรับลักษณะงานที่ต้องมีการทำตามสถานะและสถานะนั้นมีการเปลี่ยนแปลงไป-มา

การทำงานของวงจรไฟไนต์สเตตแมชชีน ในวงจรหนึ่งอาจมีจำนวนสถานะ หลาย ๆสถานะ และในแต่ละสถานะก็อาจส่งผลต่อค่าฟังก์ชั่นเอาท์พุทและค่าสถานะครั้งถัดไป

โดยทั่วไปของการออกแบบวงจรดิจิตอลในลักษณะนี้ จะมีองค์ประกอบสองส่วน (ดังรูปที่ 6.1)คือ

1. ส่วนของวงจรคอมไบเนชั่น จะมีอินพุทอยู่ 2 ชุดคือ ชุดของขาอินพุท (สำหรับรับข้อมูลจากภายนอก) และชุดของขาสถานะปัจจุบัน (pr_state) ซึ่งได้จากเอาท์พุทของวงจรซีเควน-เชียล ส่วนเอาท์พุทจะมีอยู่ 2 ชุดเช่นกันคือ ชุดของขาเอาท์พุท และชุดของขาสถานะครั้งถัดไป (nx_state)
2. ส่วนของวงจรซีเควนเชียล จะมีอินพุทจำนวน 3 ชุดคือ ขารีเซต ขาสัญญาณนาฬิกา และชุดของขาสถานะครั้งถัดไปที่ได้มาจากเอาท์พุทของวงจรคอมไบเนชั่น ส่วนเอาท์พุทจะมีอยู่ 1 ชุดคือชุดของขาสถานะปัจจุบัน สำหรับส่งไปยังวงจรคอมไบเนชั่น



รูปที่ 6.1 ลักษณะองค์ประกอบของการออกแบบวงจรดิจิตอลในลักษณะเชิงสถานะ

ในการออกแบบวงจรไฟไนต์สเตตแมชชีนนี้ เมื่อแบ่งตามค่าฟังก์ชั่นเอาท์พุทที่ได้ จะสามารถแบ่งออกได้ 2 ประเภท ได้แก่

1. ค่าของฟังก์ชั่นเอาท์พุทที่ได้ในขณะนั้น ได้มาจากค่าของฟังก์ชั่นสถานะปัจจุบันและค่าของฟังก์ชั่นอินพุทในขณะนั้นด้วย บางครั้งเรียกการออกแบบนี้ว่าเมลลี่สเตตแมชชีน (Mealy state machine)
2. ค่าของฟังก์ชั่นเอาท์พุทที่ได้ในขณะนั้น ได้มาจากค่าของฟังก์ชั่นสถานะปัจจุบันในขณะนั้นอย่างเดียว บางครั้งเรียกการออกแบบนี้ว่ามอร์สเตตแมชชีน (Moore state machine)

6.1 รูปแบบมอร์สเตตแมชชีนสำหรับภาษา VHDL

ลักษณะการเขียนโปรแกรมภาษา VHDL สำหรับรูปแบบมอร์สเตตแมชชีน จะแบ่งการเขียนออกเป็นสองส่วน คือ

ส่วนที่ 1 : เป็นส่วนของวงจรคอมไบเนชั่น

ส่วนที่ 2 : เป็นส่วนของวงจรซีเควนเชียล

ส่วนที่ 1: ไฟไนต์สเตตแมชชีนในส่วนของวงจรคอมไบเนชั่น

การเขียนโปรแกรมไฟไนต์สเตตแมชชีนในส่วนของวงจรคอมไบเนชั่น สามารถเขียนโดยใช้ชุดคำสั่งแข่งขนาน (ในบทที่4) หรือชุดคำสั่งชนิดเรียงลำดับ(ในบทที่5)ก็ได้ แต่เพื่อความสะดวกและง่ายต่อการเขียนจึงนิยมใช้ชุดคำสั่งชนิดเรียงลำดับ ประเภทชุดคำสั่ง PROCESS ที่มีการเรียกใช้คำสั่ง CASE/WHEN ดังตัว อย่างแม่แบบการเขียน ต่อไปนี้

```
1    ---------- part 1 : combinational ----------------
2    PROCESS (pr_state)
3    BEGIN
4        CASE pr_state IS
5            WHEN state0 =>  output <= <value>;
6            WHEN state1 =>  output <= <value>;
7            ...
8        END CASE;
9    END PROCESS;
```

จากตัวอย่างแม่แบบการเขียนโปแกรมนี้ เป็นลักษณะเขียนที่ง่ายต่อความเข้าใจ ที่เลือกใช้ชุดคำสั่ง PROCESS ที่มีการระบุตัวกระตุ้นการทำงาน จำนวนเพียงตัวเดียวคือ pr_state เมื่อตัวกระตุ้นมีการเปลี่ยนแปลงเหตุการณ์หรือเปลี่ยนระดับสัญญาณ จะส่งผลทำให้คำสั่งภายใน PROCESS ทำงาน ตามลำดับตั้งแต่บรรทัดที่4 เป็นต้นไป กล่าวคือเมื่อค่า pr_state ตรงกับสถานะนั้นๆก็จะทำให้ได้ค่าเอาท์พุท ณ ตำแหน่งสถานะนั้นๆ

ส่วนที่ 2: ไฟไนต์สเตตแมชชีนในส่วนของวงจรซีเควนเชียล

ในส่วนของการเขียนโปรแกรมไฟไนต์สเตตแมชชีนของวงจรซีเควนเชียลนั้น จะเป็นลักษณะเหมือนกับของหน่วยความจำหรือฟลิบฟลอบ โดยจะต้องใช้ชุดคำสั่งชนิดเรียงลำดับประเภทชุดคำสั่ง PROCESS จำนวนสองชุด กล่าวคือ

ชุดคำสั่ง PROCESS ชุดแรก จะทำหน้าที่ ตรวจสอบสถานะ(สเตต)และอินพุทขณะนั้น แล้วให้ค่าสถานะครั้งถัดไป

ชุดคำสั่ง PROCESS ชุดที่สอง จะทำหน้าที่ตรวจสอบอินพุทขารีเซตและขาสัญญาณนาฬิกา แล้วให้ค่าสถานะครั้งถัดไปสัมพันธ์กับสัญญาณนาฬิกา หรือเป็นจุดเริ่มต้นสถานะใหม่(state0)เมื่อขารีเซตมีค่าเป็นลอจิก ‘1’ ดังตัวอย่างแม่แบบการเขียน ต่อไปนี้

```
1   ---------- part 2 : sequential ------------------
2   PROCESS(input,pr_state)
3   BEGIN
4      CASE pr_state IS
5           WHEN state0 =>
6              IF (input = ...) THEN
7                nx_state <= state1;
8              END IF;
9           WHEN state1 =>
10             IF (input = ...) THEN
11               nx_state <= state2;
12             END IF;
13            ...
14        END CASE;
15     END IF;
16  END PROCESS;
17  ------------ Update state -----------------------
18  PROCESS(reset,clock)
19  BEGIN
20     IF (reset='1') THEN
21         pr_state <= state0;
22     ELSIF (clock'EVENT AND clock='1') THEN
23        pr_state <= nx_state;
24     END IF;
25  END PROCESS;
```

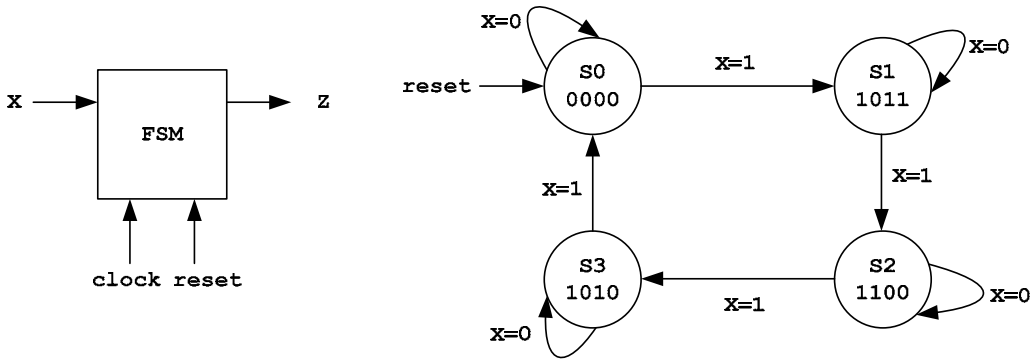
เมื่อนำแม่แบบการเขียนโปรแกรมในส่วนที่ 1 และ 2 มารวมกันจะได้ โครงสร้างของไฟไนต์สเตตแมชชีนสำหรับภาษา VHDL ในรูปแบบมอร์สเตตแมชชีน ดังนี้

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   -------------------------------------------------------
4   ENTITY <entity_name> IS
5   PORT ( input: IN <data_type>;
6       reset, clock: IN STD_LOGIC;
7       output: OUT <data_type>);
8   END <entity_name>;
9   -------------------------------------------------------
10  ARCHITECTURE <arch_name> OF <entity_name> IS
11  TYPE state IS (state0, state1, state2, state3, ...);
12  SIGNAL pr_state,nx_state : state;
13  BEGIN
14  ---------- part 2 section: ------------------------
15  PROCESS(input,pr_state)
16  BEGIN
17     CASE pr_state IS
18          WHEN state0 =>
19             IF (input = ...) THEN
20               nx_state <= state1;
21             END IF;
22          WHEN state1 =>
23             IF (input = ...) THEN
24               nx_state <= state2;
25             END IF;
26            ...
27        END CASE;
28     END IF;
```

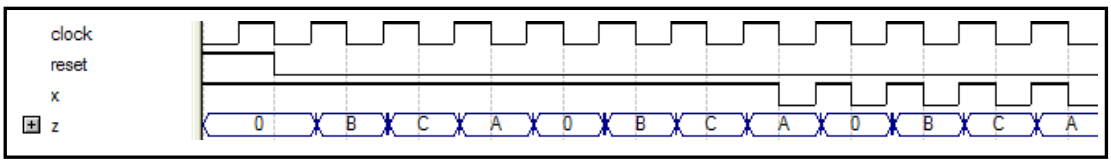
```
29   END PROCESS;
30   -------------- Update state --------------------
31   PROCESS(reset,clock)
32   BEGIN
33      IF (reset='1') THEN
34          pr_state <= state0;
35      ELSIF (clock'EVENT AND clock='1') THEN
36         pr_state <= nx_state;
37      END IF;
38   END PROCESS;
39   ---------- part 1 section: ------------------------
40    PROCESS (pr_state)
41     BEGIN
42      CASE pr_state IS
43          WHEN state0 =>  output <= <value>;
44          WHEN state1 =>  output <= <value>;
45          ...
46       END CASE;
47    END PROCESS;
48  END <arch_name>;
```

ตัวอย่างที่ 6.1 การออกแบบวงจรไฟไนต์สเตตแมชชีนอย่างง่าย ดังรูปที่ 6.2

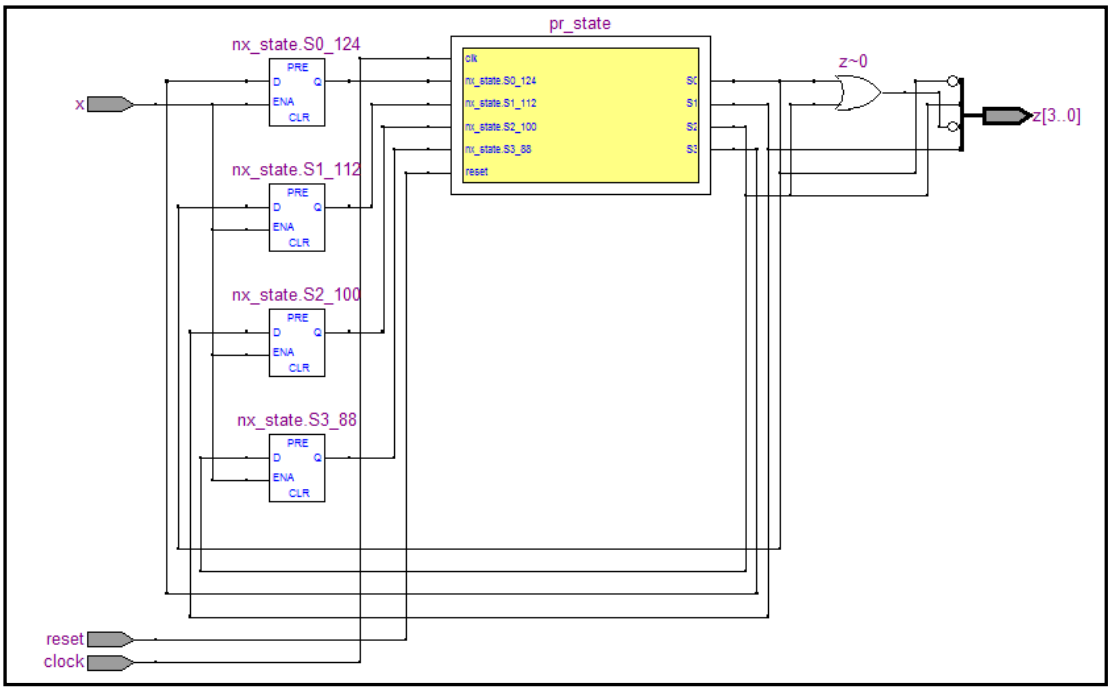
คำอธิบาย จากรูปในระบบจะมีจำนวนสถานะเพียงสี่สถานะคือ S0 S1 S2 และ S3 โดยที่สถานะจะเปลี่ยนนั้นจะเกิดขึ้นในกรณีที่อินพุท x มีค่าลอจิกเป็น ‘1’ เท่านั้น และค่าของเอาท์พุทจะมีค่าเท่ากับ “0000” “1011” “1100” และ “1010” เมื่อสถานะอยู่ที่ S0 S1 S2 และ S3 ตามลำดับ ในการเริ่มต้นการทำงานหรือมีการรีเซตระบบ จะต้องเริ่มต้นที่สถานะ S0 เสมอ



รูปที่ 6.2 รูปแบบสเตตแมชชีนของตัวอย่างที่ 6.1



รูปที่ 6.3 ผลของการจำลองของตัวอย่างที่ 6.1



รูปที่ 6.4 ผลของการสังเคราะห์วงจรลอจิกเกตของตัวอย่างที่ 6.1

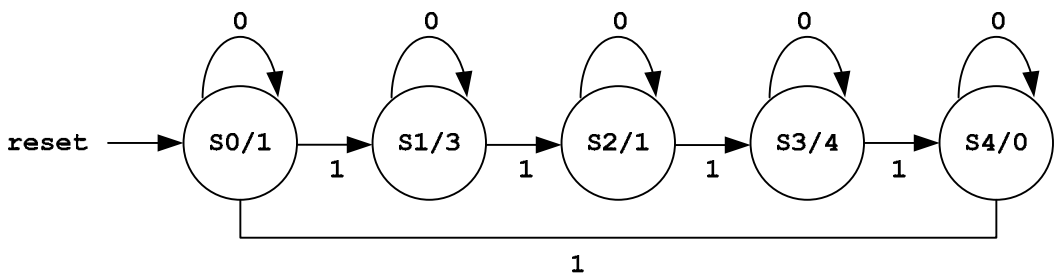
เมื่อนำไปเขียนโปแกรมก็จะอาศัย โครงสร้างของไฟไนต์สเตตแมชชีนสำหรับภาษา VHDL โดยมีชื่อสถานะในบรรทัดที่ 11 ส่วนในบรรทัดที่ 15 ถึง 44 เป็นส่วนของวงจรซีเควนเชียล และในบรรทัดที่ 46 ถึง 54 เป็นส่วนของวงจรคอมไบเนชั่น เมื่อนำผลไปทำการจำลองจะได้ดังรูปที่ 6.3 และเมื่อนำไปสังเคราะห์เป็นวงจรลอจิกเกตจะได้ดังรูปที่ 6.4

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   -------------------------------------------------------
4   ENTITY exam_fsm_moore1 IS
5   PORT ( x: IN STD_LOGIC;
6          reset, clock: IN STD_LOGIC;
7          z: OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
8   END exam_fsm_moore1;
9   -------------------------------------------------------
10  ARCHITECTURE fsm OF exam_fsm_moore1 IS
11  TYPE state IS (S0, S1, S2, S3);
12  SIGNAL pr_state,nx_state : state;
13  BEGIN
14  ---------- part 2 section: ------------------------
15         PROCESS (reset, clock)
16         BEGIN
17               IF (reset='1') THEN
18                     pr_state <= S0;
19               ELSIF (clock'EVENT AND clock='1') THEN
20                     pr_state <= nx_state;
21               END IF;
22         END PROCESS;
23  ---------------------------------------------------
24         PROCESS (x,pr_state)
25         BEGIN
26               CASE pr_state IS
27                     WHEN S0 =>
```

```
28                          IF (x = '1') THEN
29                                nx_state <= S1;
30                          END IF;
31                    WHEN S1 =>
32                          IF (x = '1') THEN
33                                nx_state <= S2;
34                          END IF;
35                    WHEN S2 =>
36                          IF (x = '1') THEN
37                                nx_state <= S3;
38                          END IF;
39                    WHEN S3 =>
40                          IF (x = '1') THEN
41                                nx_state <= S0;
42                          END IF;
43                    END CASE;
44              END PROCESS;
45  ---------- part 1 section: ------------------------
46    PROCESS (pr_state)
47    BEGIN
48              CASE pr_state IS
49                    WHEN S0 =>  z <= "0000";
50                    WHEN S1 =>  z <= "1011";
51                    WHEN S2 =>  z <= "1100";
52                    WHEN S3 =>  z <= "1010";
53               END CASE;
54         END PROCESS;
55  END fsm;
56  ---------------------------------------------
```

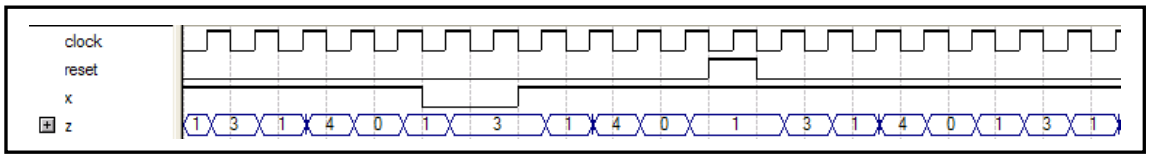
ตัวอย่างที่ 6.2 การออกแบบวงจรนับเพื่อให้ได้ค่า 13140 โดยใช้รูปแบบไฟไนต์สเตตแมชชีน

คำอธิบาย ในการออกแบบวงจรนับแบบนี้จะมีการนับค่าทั้งหมดห้าครั้ง ดังนั้นก็จะมีจำนวนสถานะทั้งหมด 5 สถานะ คือ S0 S1 S2 S3 และ S4 โดยแต่ละสถานะก็จะให้ค่าเอาท์พุทออกมาเป็น 1 3 1 4 และ 0 ดังรูปที่ 6.5 การเปลี่ยนจากสถานะหนึ่งไปยังอีกสถานะหนึ่งจะต้องให้อินพุท x มีค่าเป็นลอจิก ‘1’ เท่านั้น และเมื่อขาอินพุท reset มีค่าเป็นลอจิก ‘1’ จะทำให้ระบบเริ่มต้นใหม่ที่สถานะ S0 ทันที และเมื่อมีการเปลี่ยนแปลงของสถานะถึงสถานะ S4 ระบบก็จะกลับมาเริ่มต้นใหม่ที่สถานะ S0



รูปที่ 6.5 แสตตไดอะแกรมของลำดับการนับ 13140

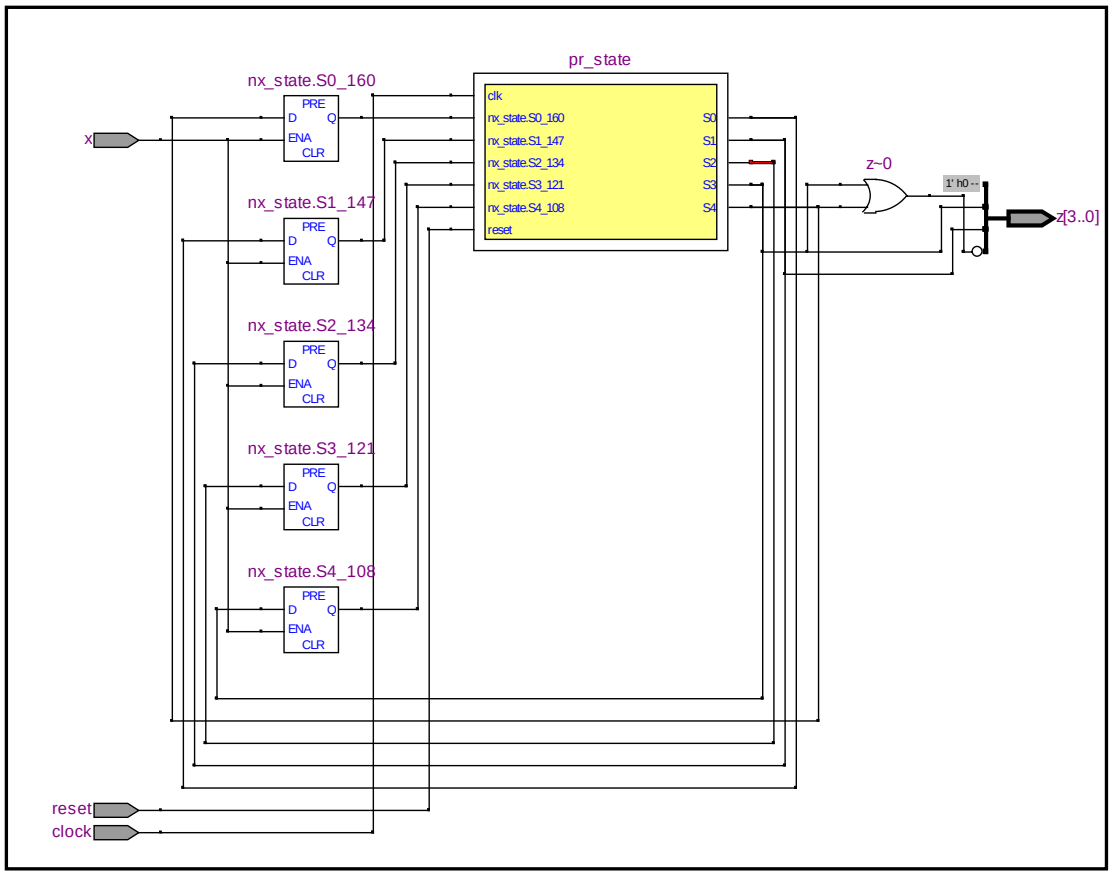
เมื่อนำไปเขียนโปแกรมก็จะอาศัย โครงสร้างของไฟไนต์สเตตแมชชีนสำหรับภาษา VHDL เช่นกัน โดยชื่อสถานะจะเพิ่มในบรรทัดที่ 11 ส่วนในบรรทัดที่ 15 ถึง 48 เป็นส่วนของวงจรซีเควนเชียล และในบรรทัดที่ 50 ถึง 59 เป็นส่วนของวงจรคอมไบเนชั่น เมื่อนำผลไปทำการจำลองจะได้ดังรูปที่ 6.6 และเมื่อนำไปสังเคราะห์เป็นวงจรลอจิกเกตจะได้ดังรูปที่ 6.7



รูปที่ 6.6 ผลของการจำลองของตัวอย่างที่ 6.2

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   -----------------------------------------------------
4   ENTITY exam_fsm_moore_13140 IS
5   PORT ( x: IN STD_LOGIC;
6         reset, clock: IN STD_LOGIC;
7         z: OUT INTEGER RANGE 0 TO 9 );
8   END exam_fsm_moore_13140;
9   -----------------------------------------------------
10  ARCHITECTURE fsm OF exam_fsm_moore_13140 IS
11  TYPE state IS (S0, S1, S2, S3, S4);
12  SIGNAL pr_state,nx_state : state;
13  BEGIN
14  ---------- part 2 section: ------------------------
15        PROCESS (reset, clock)
16        BEGIN
17              IF (reset='1') THEN
18                    pr_state <= S0;
19              ELSIF (clock'EVENT AND clock='1') THEN
20                    pr_state <= nx_state;
21              END IF;
22        END PROCESS;
23  ---------------------------------------------------
24        PROCESS (x,pr_state)
25        BEGIN
26              CASE pr_state IS
27                    WHEN S0 =>
28                          IF (x = '1') THEN
29                                nx_state <= S1;
30                          END IF;
31                    WHEN S1 =>
32                          IF (x = '1') THEN
33                                nx_state <= S2;
34                          END IF;
35                    WHEN S2 =>
36                          IF (x = '1') THEN
37                                nx_state <= S3;
38                          END IF;
39                    WHEN S3 =>
40                          IF (x = '1') THEN
41                                nx_state <= S4;
42                          END IF;
43                    WHEN S4 =>
```

```
44                          IF (x = '1') THEN
45                                nx_state <= S0;
46                          END IF;
47                END CASE;
48          END PROCESS;
49  ---------- part 1 section: ------------------------
50     PROCESS (pr_state)
51           BEGIN
52                CASE pr_state IS
53                      WHEN S0 =>  z <= 1;
54                      WHEN S1 =>  z <= 3;
55                      WHEN S2 =>  z <= 1;
56                      WHEN S3 =>  z <= 4;
57                      WHEN S4 =>  z <= 0;
58                 END CASE;
59           END PROCESS;
60  END fsm;
```



รูปที่ 6.7 ผลของการสังเคราะห์วงจรลอจิกเกตของตัวอย่างที่ 6.2

6.2 รูปแบบเมลลี่สเตตแมชชีนสำหรับภาษา VHDL

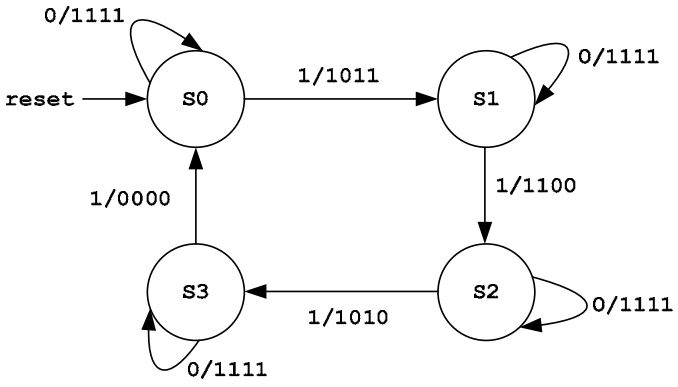
จากหัวข้อที่ผ่านมาเป็นการออกแบบวงจรในรูปแบบมอร์สเตตแมชชีน ที่ซึ่งค่าเอาท์พุทขณนั้นจะได้จากสถานะขณะนั้นเท่านั้น แต่การออกแบบในรูปแบบเมลลี่สเตตแมชชีนแล้วค่าของเอาท์พุทที่ได้จะขึ้นอยู่กับสถานะปัจจุบันและอินพุทปัจจุบัน ดังนั้นเมื่อนำไปทำเป็นแม่แบบการเขียนโปรแกรมภาษา VHDL จะปรับปรุงแม่แบบการเขียนของมอร์สเตตแมชชีน เฉพาะในส่วนที่ 1 เท่านั้น ดังต่อไปนี้

```
1    LIBRARY ieee;
2    USE ieee.std_logic_1164.all;
3    ----------------------------------------------------
4    ENTITY <entity_name> IS
5    PORT ( input: IN <data_type>;
6        reset, clock: IN STD_LOGIC;
7        output: OUT <data_type>);
8    END <entity_name>;
9    ----------------------------------------------------
10   ARCHITECTURE <arch_name> OF <entity_name> IS
11   TYPE state IS (state0, state1, state2, state3, ...);
12   SIGNAL pr_state,nx_state : state;
13   BEGIN
14   ---------- part 2 section: ------------------------
15   PROCESS(input,pr_state)
16   BEGIN
17      CASE pr_state IS
18           WHEN state0 =>
19              IF (input = ...) THEN
20                nx_state <= state1;
21              END IF;
22           WHEN state1 =>
23              IF (input = ...) THEN
24                nx_state <= state2;
25              END IF;
26            ...
27         END CASE;
28      END IF;
29   END PROCESS;
30   -------------- Update state --------------------
31   PROCESS(reset,clock)
32   BEGIN
33      IF (reset='1') THEN
34          pr_state <= state0;
35      ELSIF (clock'EVENT AND clock='1') THEN
36         pr_state <= nx_state;
37      END IF;
38   END PROCESS;
39  ---------- part 1 section: ----------------------
40    PROCESS (input,pr_state)
41    BEGIN
42      CASE pr_state IS
43         WHEN state0 =>
44              IF (input = ...) THEN
45                output <= <value>;
46              ELSE
47                output <= <value>;
48              END IF;
49         WHEN state1 =>
50              IF (input = ...) THEN
51                output <= <value>;
52              ELSE
```

```
53                output <= <value>;
54              END IF;
55              ...
56        END CASE;
57     END PROCESS;
58  END <arch_name>;
```

ตัวอย่างที่ 6.3 การออกแบบเมลลี่สเตตแมชชีน ดังรูปที่ 6.8

คำอธิบาย จากรูปเมื่ออินพุทเปลี่ยนแปลงจะทำให้สถานะปัจจุบันและเอาท์พุทเปลี่ยนแปลงตาม โดยมีจำนวนสถานะสี่สถานะคือ S0 S1 S2 และ S3 จากรูปแบบการเขียนโปรแกรมของเมลลี่สเตตแมชชีน จะมีการระบุชื่อของสถานะในบรรทัดที่ 11 และส่วนที่ 2 มีการตรวจค่ากระตุ้นการทำงานของค่าสถานะปัจจุบันและอินพุทเพื่อให้ได้ค่าเอาท์พุท

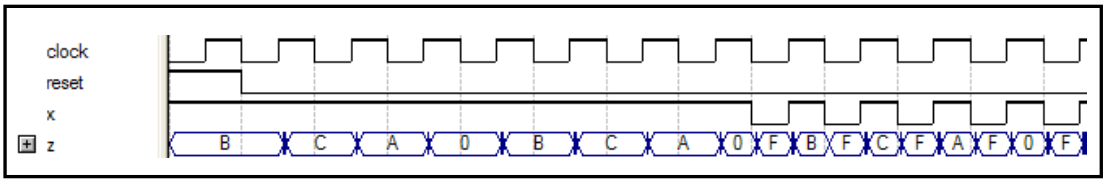


รูปที่ 6.8 รูปแบบสเตตแมชชีนของตัวอย่างที่ 6.3

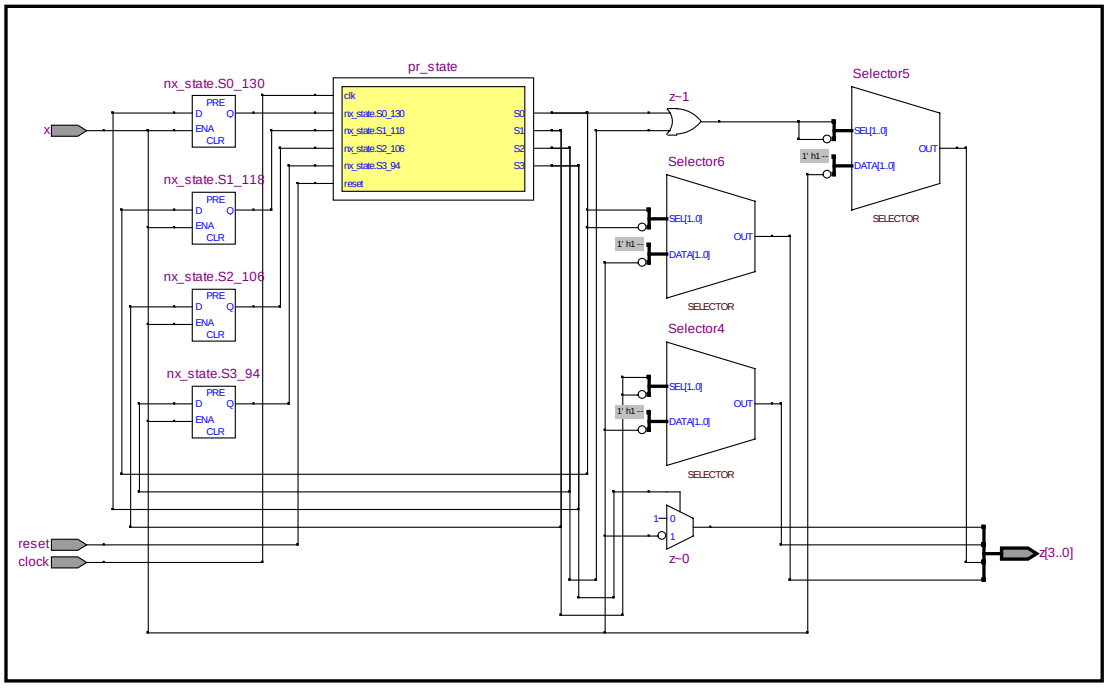
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  -----------------------------------------------------
4  ENTITY exam_fsm_mealy_1 IS
5  PORT ( x: IN STD_LOGIC;
6         reset, clock: IN STD_LOGIC;
7         z: OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
8  END exam_fsm_mealy_1;
9  -----------------------------------------------------
10 ARCHITECTURE fsm OF exam_fsm_mealy_1 IS
11 TYPE state IS (S0, S1, S2, S3);
12 SIGNAL pr_state,nx_state : state;
13 BEGIN
14 ---------- part 2 section: ------------------------
15        PROCESS (reset, clock)
16        BEGIN
17              IF (reset='1') THEN
18                    pr_state <= S0;
19              ELSIF (clock'EVENT AND clock='1') THEN
20                    pr_state <= nx_state;
21              END IF;
22        END PROCESS;
23 ---------------------------------------------------
24        PROCESS (x,pr_state)
25        BEGIN
26              CASE pr_state IS
```

```
27                  WHEN S0 =>
28                        IF (x = '1') THEN
29                              nx_state <= S1;
30                        END IF;
31                  WHEN S1 =>
32                        IF (x = '1') THEN
33                              nx_state <= S2;
34                        END IF;
35                  WHEN S2 =>
36                        IF (x = '1') THEN
37                              nx_state <= S3;
38                        END IF;
39                  WHEN S3 =>
40                        IF (x = '1') THEN
41                              nx_state <= S0;
42                        END IF;
43                  WHEN S4 =>
44                        IF (x = '1') THEN
45                              nx_state <= S0;
46                        END IF;
47                  END CASE;
48            END PROCESS;
49 ---------- part 1 section: ----------------------
50    PROCESS (x,pr_state)
51        BEGIN
52            CASE pr_state IS
53                  WHEN S0 =>
54                        IF (x='1') THEN
55                              Z <= "1011";
56                        ELSE
57                              Z <= "1111";
58                        END IF;
59                  WHEN S1 =>
60                        IF (x='1') THEN
61                              Z <= "1100";
62                        ELSE
63                              Z <= "1111";
64                        END IF;
65                  WHEN S2 =>
66                        IF (x='1') THEN
67                              Z <= "1010";
68                        ELSE
69                              Z <= "1111";
70                        END IF;
71                  WHEN S3 =>
72                        IF (x='1') THEN
73                              Z <= "0000";
74                        ELSE
75                              Z <= "1111";
76                        END IF;
77             END CASE;
78       END PROCESS;
79  END fsm;
```

เมื่อนำเขียนโปแกรมไปทำการจำลองจะได้ดังรูปที่ 6.9 และเมื่อนำไปสังเคราะห์เป็นวงจรลอจิกเกตจะได้ดังรูปที่ 6.10 และจากผลการจำลองจะเห็นว่าเอาท์พุทที่ได้จะแตกต่างจากตัวอย่างที่ 6.1



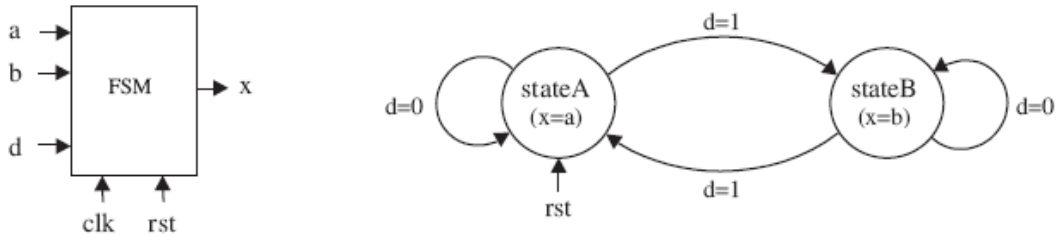
รูปที่ 6.9 ผลของการจำลองของตัวอย่างที่ 6.3



รูปที่ 6.10 ผลของการสังเคราะห์วงจรลอจิกเกตของตัวอย่างที่ 6.3

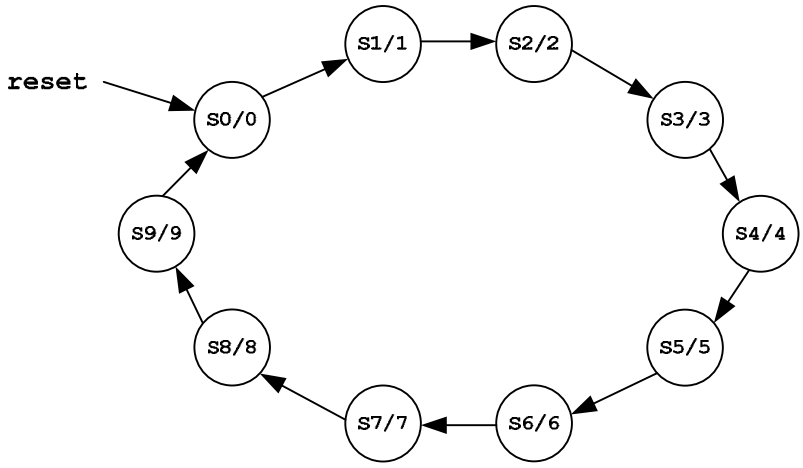
6.3 โจทย์ปัญหา

1. จากรูปที่ บผ6.1 ให้เขียนบรรยายเป็นภาษา VHDL



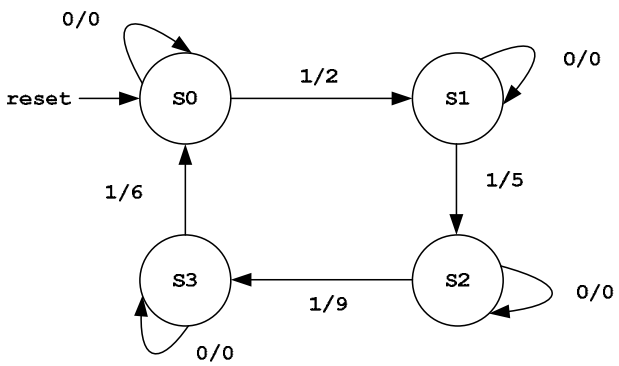
รูปที่ บผ6.1

2. จากรูปที่ บผ6.2 เป็นการออกแบบวงจรนับ BCD counter ให้เขียนบรรยายเป็นภาษา VHDL



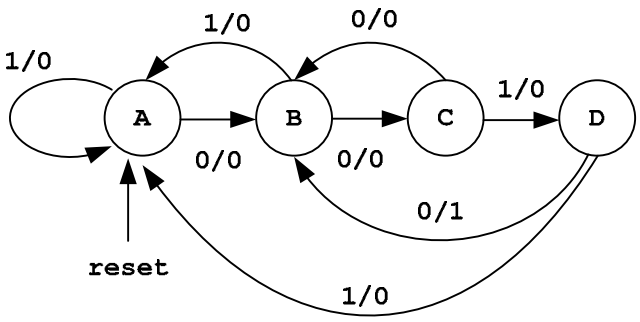
รูปที่ บผ6.2

3. จากรูปที่ บผ6.3 ให้เขียนบรรยายเป็นภาษา VHDL



รูปที่ บผ6.3

4. จากรูปที่ บผ6.4 ให้เขียนบรรยายเป็นภาษา VHDL



รูปที่ บผ6.4

ก

ภาคผนวก ก.

ประวัติความเป็นมาของภาษา VHDL และข้อกำหนดในการตั้งชื่อตัวแปร

VHDL คือะไร

VHDL ย่อมาจาก **V**HSIC **H**ardware **D**escription **L**anguage (VHIC : Very High speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ใช้กำหนดและบรรยายพฤติกรรมฟังก์ชัน การทำงานของฮาร์ด-แวร์ในระบบดิจิตอล ภาษา VHDL นี้ถูกกระทรวงกลาโหมของสหรัฐอเมริกา เริ่มพัฒนาตั้งแต่ปี ค.ศ. 1981 จนกระทั่งปี ค.ศ. 1985 เทคโนโลยีการออกแบบวงจรดิจิตอลด้วยภาษาVHDL ก็ได้ถูกนำมาใช้งานโดยทั่วไป หลังจากนั้นไม่นานทางสมาคม IEEE จึงได้รับภาษานี้เข้ามาศึกษา จนกระทั่งปี ค.ศ. 1987 ได้ยอมกำหนดมาตรฐานของภาษา VHDL โดยใช้ชื่อว่า IEEE 1076-1987 และได้มีการปรับปรุงมาตรฐานใหม่ในปี ค.ศ.1993 โดยใช้ชื่อว่า IEEE 1076-1993

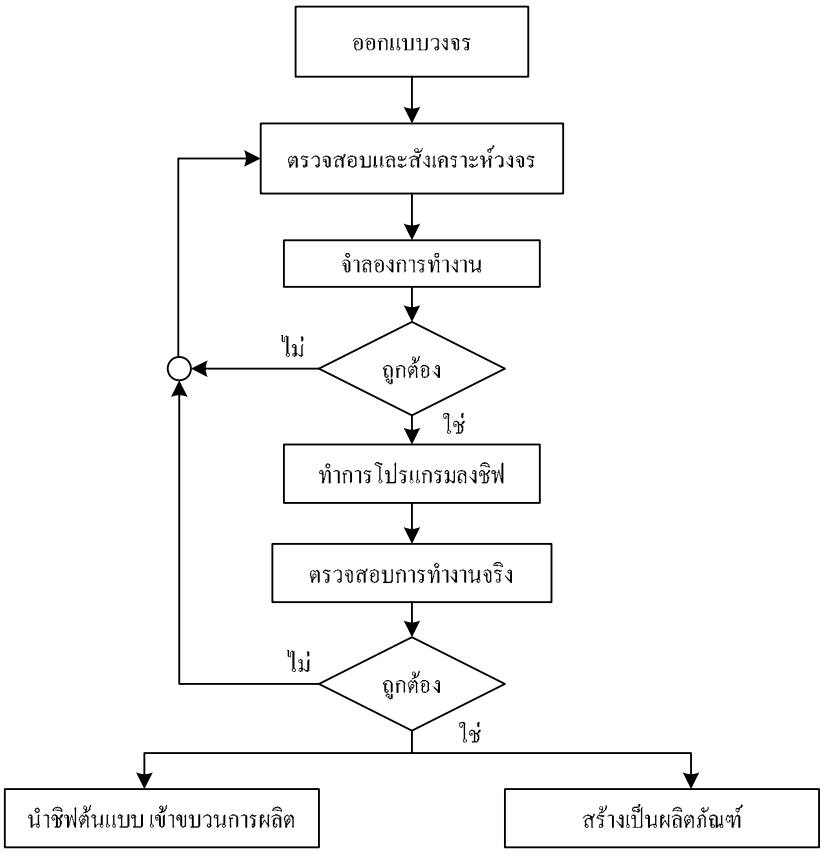
ข้อดีของภาษา VHDL

- เป็นมาตรฐานของ IEEE ทำให้มีเครื่องมือต่างๆและบริษัทที่สนับสนุนการทำงานมากมาย นอกจากนี้วงจรที่ออกแบบโดย VHDL ก็จะใช้งานได้นานเนื่องจากมีความเข้ากันได้ของภาษากับวงจรที่ได้รับการออกแบบใหม่
- ภาคอุตสาหรกรรมสนับสนุน เนื่องจากภาษา VHDL เป็นภาษาที่เป็นมาตรฐานของ IEEE จึงมีอุตสาห-กรรมจำนวนมากที่รองรับการออกแบบที่ใช้ภาษา VHDL
- การออกแบบโดยใช้ภาษา VHDL สามารถนำไปจำลองการทำงานหรือสังเคราะห์ ด้วยซอฟท์แวร์ตัวใดก็ได้ที่รองรับภาษาVHDL จึงทำให้การออกแบบด้วยภาษา VHDL จึงเป็นการออกแบบที่ไม่ยึดติดกับซอฟท์แวร์ที่ใช้ในการออกแบบ
- ผู้ออกแบบวงจรสามารถออกแบบวงจรโดยใช้ภาษา VHDL ได้หลายระดับตั้งแต่ระดับ Electronic boxes ถึงระดับทรานซิสเตอร์ และสามารถออกแบบวงจรที่มีความซับซ้อนสูงและมีขนาดใหญ่มากได้
- วงจรที่ออกแบบโดยภาษา VHDL สามารถนำกลับมาใช้ใหม่ได้ง่าย เนื่องจากสามารถเปลี่ยนแปลงแก้ไขวงจรได้ง่าย
- เป็นภาษาในรูปแบบบรรยายพฤติกรรม ทำให้เราสามารถอธิบายการทำงานของวงจรภายในการออกแบบได้ทัน

ขั้นตอนการออกแบบ

ขั้นตอนการออกแบบจะเริ่มต้นจากศึกษาขั้นตอนหรือพฤติกรรมการทำงานของวงจรที่ต้องจะสังเคราะห์มาใช้งาน หลังจากนั้นทำการกำหนดจำนวนพร้อมขนาดของสัญญาณที่เข้าไปและสัญญาณหลังจากมีการทำงานแล้ว หลังจากนั้นก็เลือกใช้ชุดคำสั่งที่เหมาะสม หลังจากนั้นโปรแกรมภาษา VHDL ที่ถูกเขียนขึ้นจะถูกคอมไพล์ เมื่อผ่านขั้นตอนนี้ ก็จะทำเข้าสู่ขั้นตอนการจำลองผล ซึ่งเป็นตอนที่สำคัญขั้นตอนหนึ่ง โดยผลการจำลองที่ได้ตรงตามความต้องการหรือไม่ ถ้าไม่ก็ต้องออกแบบใหม่ แต่ถ้าถูกต้องก็ทำการอัดโค๊ดโปรแกรมที่ได้จากการคอมไพล์ลงสู่ชิพไอซี ซึ่งได้แก่ชิพไอซีประเภท CPLD หรือ FPGA ที่ซึ่งพร้อมจะถูกนำไปใช้งานจริง ดังแสดงแสดงในรูปที่ ผ.1

จากขั้นดังกล่าวผู้ออกแบบ สามารถที่จะออกแบบ ดูผลการจำลอง พร้อมผลิตชิพไอซีที่ต้องการ ซึ่งจะใช้เวลาไม่มาก และก็ไม่ต้องสิ้นเปลืองเวลาในการต่อสายวงวงจร เหมือนเมื่อก่อน



รูปที่ ผ.1 ขั้นตอนการออกแบบวงจรดิจิตอลยุคใหม่

ข้อกำหนดการตั้งชื่อตัวแปรในด้วยภาษา VHDL

การเขียนโปรแกรมด้วยภาษา VHDL จะมีข้อกำหนดเฉกเช่นเดียวกันกับการเขียนโปรแกรมด้วยภาษาอื่นๆ ดังนั้นจึงต้องพยายามทำความเข้าใจข้อกำหนดต่างๆก่อนลงมือเขียนโปรแกรม เพื่อที่จะทำให้การเขียนโปรแกรมเป็นไปด้วยดี

การตั้งชื่อตัวแปร (Object Name)

- ชื่อจะประกอบด้วยตัวหนังสือ(พยัญชนะและตัวเลข)ในภาษาอังกฤษ ได้แก่ A-Z , a-z , 0-9, และ _ (underscore)
- ชื่อจะต้องขึ้นต้นตัวอักษรเสมอ
- ชื่อสามารถประกอบด้วยพยัญชนะ ตัวเลข และเครื่องหมายขีดล่างจำนวนไม่จำกัด
- การใช้เครื่องขีดล่าง(_)ทุกครั้งจะต้องนำหน้าด้วยพยัญชนะ หรือตัวเลขและตามด้วยพยัญชนะหรือตัวเลข
- ชื่อที่ใช้ด้วยพยัญชนะตัวใหญ่หรือตัวเล็กไม่มีความแตกต่างกัน(case insensitive)
- ชื่อที่ใช้จะต้องไม่ซ้ำกับคำสงวน
- ชื่อของแฟ้มข้อมูลจะต้องตรงกันชื่อ entity_name

หนังสือเล่มนี้ผู้เขียนอ้างมาตราฐานภาษา VHDL ปี 87 ซึ่งเป็มาตราฐานของภาษา VHDL ยุคเริ่มแรก ซึ่งข้อแตกต่างระหว่าง VHDL ปี 87 ปี 93 และปี 2002 นั้นแตกต่างไม่มากเท่าไร ผู้อ่านสามารถหาอ่านได้ ในเครื่อข่ายอินเตอร์เนท และในภาคผนวก ข. ของชุดคำสงวนก็มีข้อแตกต่างในงานบางคำสั่ง ยกตัวอย่างเช่น XNOR ไม่มีบรรจุอยู่ใน VHDL ปี 87 แต่มีใน VHDL ปี 93

ข

ภาคผนวก ข.

คำสงวนในภาษา VHDL

From VHDL 87:
 	
ABS
ACCESS
AFTER
ALIAS
ALL
AND
ARCHITECTURE
ARRAY
ASSERT
ATTRIBUTE
BEGIN
BLOCK
BODY
BUFFER
BUS
CASE
COMPONENT
CONFIGURATION
CONSTANT
DISCONNECT
DOWNTO
ELSE
ELSIF
END
	ENTITY
	EXIT
	FILE
	FOR
	FUNCTION
	GENERATE
	GENERIC
	GUARDED
	IF
	IN
	INOUT
	IS
	LABEL
	LIBRARY
	LINKAGE
	LOOP
	MAP
	MOD
	NAND
	NEW
	NEXT
	NOR
	NOT
	NULL
	OF
	ON
		OPEN
		OR
		OTHERS
		OUT
		PACKAGE
		PORT
		PROCEDURE
		PROCESS
		RANGE
		RECORD
		REGISTER
		REM
		REPORT
		RETURN
		SELECT
		SEVERITY
		SIGNAL
		SUBTYPE
		THEN
		TO
		TRANSPORT
		TYPE
		UNITS
		UNTIL
		USE
		VARIABLE
			WAIT
			WHEN
			WHILE
			WITH
			XOR
			
			From VHDL 93:
			GROUP
			IMPURE
			INERTIAL
			LITERAL
			POSTPONED
			PURE
			REJECT
			ROL
			ROR
			SHARED
			SLA
			SLL
			SRA
			SRL
			UNAFFECTED
			XNOR

ค

ภาคผนวก ค.

Standard Package ชื่อ STD_LOGIC_1164

```
-- This package shall be compiled into a design library
-- symbolically named IEEE.

package STD_LOGIC_1164 is
-----------------------------------------------------------------------------
-- logic State System (unresolved)
-----------------------------------------------------------------------------
type STD_ LOGIC is    (          'U',      -- Uninitialized
                                 'X',      -- Forcing Unknown
                                 '0',      -- Forcing    0
                                 '1',      -- Forcing    1
                                 'Z'       -- High Impedance
                                 'W',      -- Weak Unknown
                                 'L',      -- Weak 0
                                 'H',      -- Weak 1
                                 '-'       -- don't care
                                 );
-----------------------------------------------------------------------------
-- Unconstrained array of std_ulogic for use with the
-- resolution function
-----------------------------------------------------------------------------
type STD_ULOGIC_VECTOR is array ( NATURAL range <> ) of STD-ULOGIC;
-----------------------------------------------------------------------------
-- resolution function
-----------------------------------------------------------------------------
-- function RESOLVED ( S : STD_ULOGIC_VECTOR )  return STD_ULOGIC;
-----------------------------------------------------------------------------
-- *** industry standard logic type ***
-----------------------------------------------------------------------------
subtype STD-LOGIC is RESOLVED STD_ULOGIC;
-----------------------------------------------------------------------------
-- Unconstrained array of std_logic for use in declaring
-- signal arrays
-----------------------------------------------------------------------------
type STD_LOGIC_VECTOR is array ( NATURAL range <> )of STD_LOGIC;
-----------------------------------------------------------------------------
-- common subtypes
-----------------------------------------------------------------------------
subtype X01 is RESOLVED STD_ULOGIC range 'X' to '1';        -- ('X', '0', '1')
subtype X01Z is RESOLVED STD_ULOGIC range 'X' to 'Z';       -- ('X', '0', '1', 'Z;)
subtype UX01 is RESOLVED STD_ULOGIC range 'U' to '1';       -- ('U', 'X', '0', '1')
subtype UX01Z is RESOLVED STD_ULOGIC range 'U' to 'Z';      -- ('U', 'X', '0', '1', 'Z')
-----------------------------------------------------------------------------
-- overloaded logical operators
```

```
-----------------------------------------------------------------------------
function "and" ( L : STD_ULOGIC; R : STD_ULOGIC ) return UX01;
function "nand" (L : STD_ULOGIC; R : STD_ULOGIC ) return UX01;
function "or" (L : STD_ULOGIC; R : STD_ULOGIC ) return UX01;
function "xor" (L : STD_ULOGIC; R : STD_ULOGIC ) return UX01;
function "xnor" (L : STD_ULOGIC; R : STD_ULOGIC ) return UX01;
function "not" (L : STD_ULOGIC ) return UX01;
-----------------------------------------------------------------------------
-- vectorized overloaded logical operators
-----------------------------------------------------------------------------
function "and" ( L, R : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "and" ( L, R : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
function "nand" ( L, R : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "nand" ( L, R : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
function "or" ( L, R : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "or" ( L, R : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
function "nor" ( L, R : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "nor" ( L, R : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
function "xor" ( L, R : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "xor" ( L, R : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
function "xnor" ( L, R : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "xnor" ( L, R : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
function "not" ( L  : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function "not" ( L   : STD_ULOGIC_VECTOR ) return STD_ULOGIC_VECTOR;
-----------------------------------------------------------------------------
-- conversion functions
-----------------------------------------------------------------------------
function TO_BIT ( S : STD_ULOGIC; XMAP : BIT :='0') return BIT;
function TO_BITVECTOR (S : STD_LOGIC_VECTOR;  XMAP : BIT :='0') return BIT_VECTOR
function TO_BITVECTOR (S : STD_ULOGIC_VECTOR;XMAP : BIT :='0') return BIT_VECTOR
function TO_STDULOGIC( B : BIT ) return STD_ULOGIC;
function TO_STDLOGICVECTOR(B : BIT_VECTOR) return STD_LOGIC_VECTOR;
function TO_STDLOGICVECTOR(S : STD_ULOGIC_VECTOR)return STD_LOGIC_VECTOR;
function TO_STDULOGICVECTOR(B : BIT_VECTOR) return STD_ULOGIC_VECTOR;
function TO_STDULOGICVECTOR(S : STD_LOGIC_VECTOR) return STD_ULOGIC_VECTOR;
-----------------------------------------------------------------------------
-- strength strippers and type converters
-----------------------------------------------------------------------------
function TO_X01  ( S : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function TO_X01  ( S : STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function TO_X01  ( S : STD_ULOGIC ) return X01;
function TO_X01  ( B : BIT_VECTOR ) return STD_LOGIC_VECTOR;
function TO_X01  ( B : BIT_VECTOR ) return STD_ULOGIC_VECTOR;
function TO_X01  ( B : BIT ) return X01;
function TO_X01Z  ( S : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function TO_X01Z  ( S : STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function TO_X01Z  ( S : STD_ULOGIC     ) return X01Z;
function TO_X01Z  ( B : BIT_VECTOR     ) return STD_LOGIC_VECTOR;
function TO_X01Z  ( B : BIT_VECTOR     ) return STD_ULOGIC_VECTOR;
function TO_X01Z  (B : BIT ) return X01Z;
function TO_UX01  ( S : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR;
function TO_UX01  ( S : STD_ULOGIC_VECTOR) return STD_ULOGIC_VECTOR;
function TO_UX01  ( S : STD_ULOGIC     ) return UX01;
function TO_UX01  ( B : BIT_VECTOR     ) return STD_LOGIC_VECTOR;
function TO_UX01  ( B : BIT_VECTOR     ) return STD_ULOGIC_VECTOR;
function TO_UX01  ( B : BIT  )  return UX01;
-----------------------------------------------------------------------------
-- edge detection
-----------------------------------------------------------------------------
function RISING_EDGE (signal S : STD_ULOGIC) return BOOLEAN;
```

```
function FALLING_EDGE (signal S : STD_ULOGIC)  return BOOLEAN;
-----------------------------------------------------------------------------------
-- object contains an unknown
-----------------------------------------------------------------------------------
function IS_X ( S : STD_ULOGIC_VECTOR ) return BOOLEAN;
function IS_X ( S : STD_LOGIC_VECTOR ) return BOOLEAN;
function IS_X ( S : STD_ULOGIC      ) return BOOLEAN;

end STD_LOGIC_1164;
```

ง

ภาคผนวก ง.

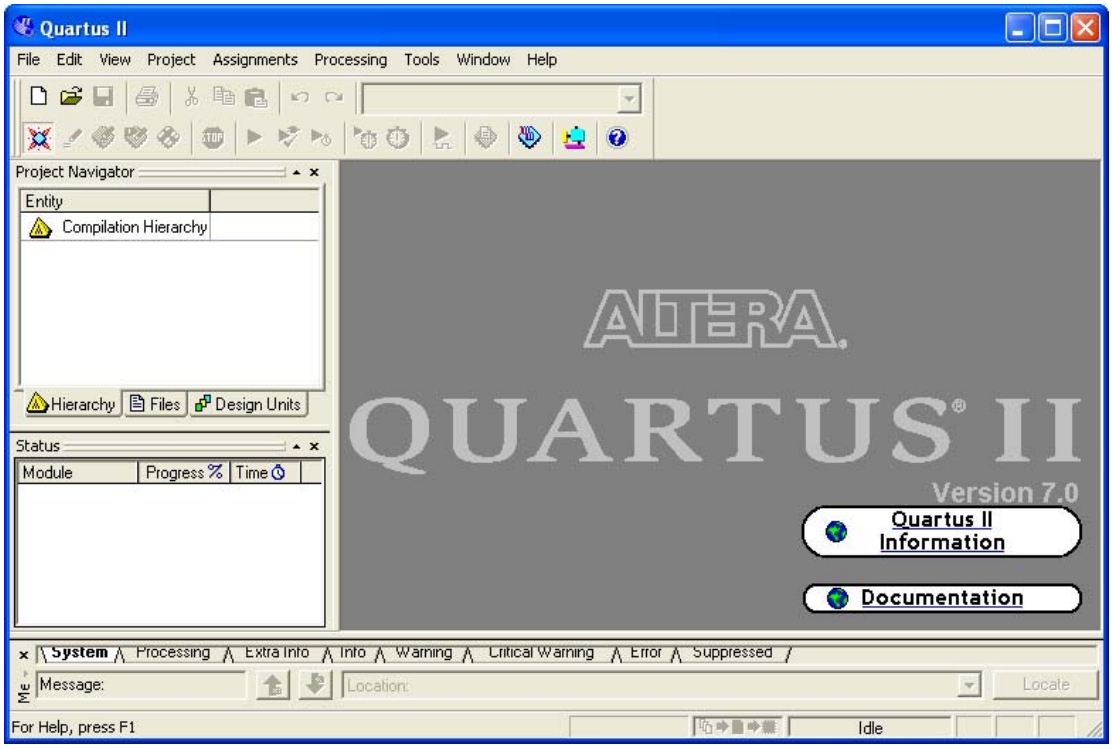
การใช้งานโปรแกรม Quartus II เบื้องต้น สำหรับการเขียนโปรแกรม VHDL

โปรแกรม Quartus II สร้างจากบริษัท Altera เป็นโปรแกรมที่ช่วยในออกแบบและสังเคราะห์วงจรดิจิตอลลงสู่ชิพซีพีแอลดี (CPLD) หรือ เอฟพีจีเอ (FPGA) ตามความต้องการของผู้ใช้งาน โปรแกรมนี้จะมีลักษณะการใช้งานอยู่ 2 รูปแบบคือ

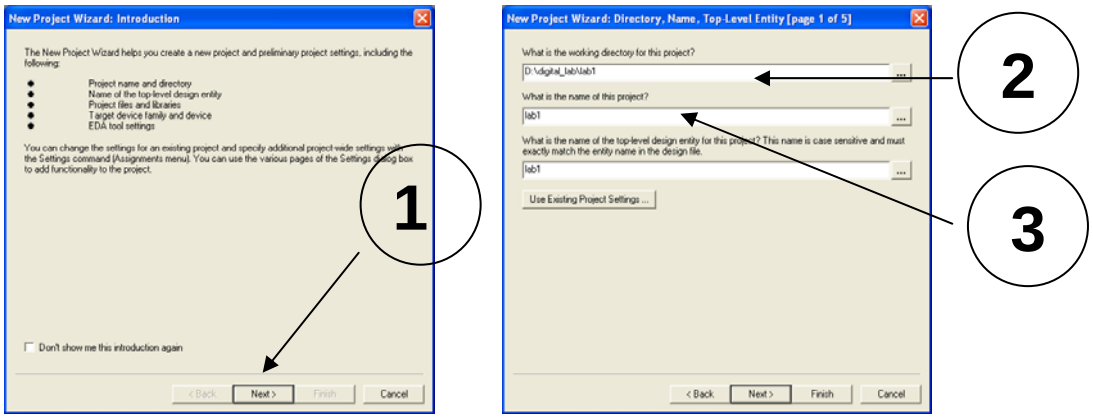
- รูปแบบ Schemetic หรือ Block diagam เป็นลักษณะไฟล์กราฟฟิก ซึ่งสามารถนำเกตประเภทต่างๆใน Library หรือโมเดลที่เราเป็นผู้สร้างขึ้น มาวางต่อกัน ให้เกิดเป็นรูปวงจรลอจิก
- รูปแบบ Text เป็นลักษณะการเขียนโปรแกรมที่บรรยายถึงลักษณะการทำงานของวงจรดิจิตอลที่จะนำไปใช้งาน ภาษาที่ใช้ได้แก่ ภาษาวีเอชดีแอล (VHDL) ภาษาเวอริลอก (Verilog) เป็นต้น

ในภาคผนวกนี้จะขอนำเสนอเฉพาะรูปแบบเป็นโปรแกรมภาษา VHDL ซึ่งมีขั้นตอนดังต่อไปนี้

1. เปิดโปรแกรม Quartus II ซึ่งจะมีลักษณะดังรูป



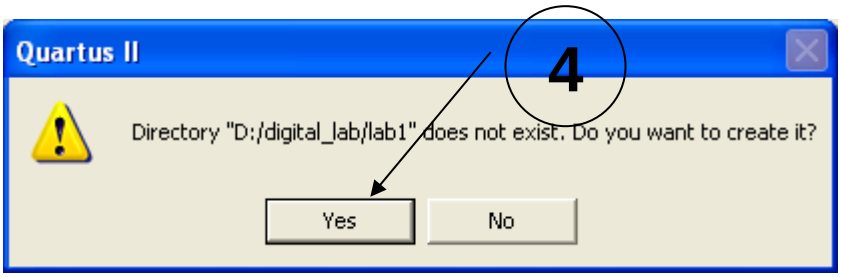
2. การใช้งาน Quartus II ทุกครั้งจะต้องตั้งชื่อโปรเจ็ค โดยคลิกที่ **File / New Project Wizard** จากนั้น ตัวโปรแกรมจะมีตัวช่วยหรือ Wizard ช่วยในการจัดเก็บและตั้งชื่อโปรเจ็ค ดังแสดงในรูป

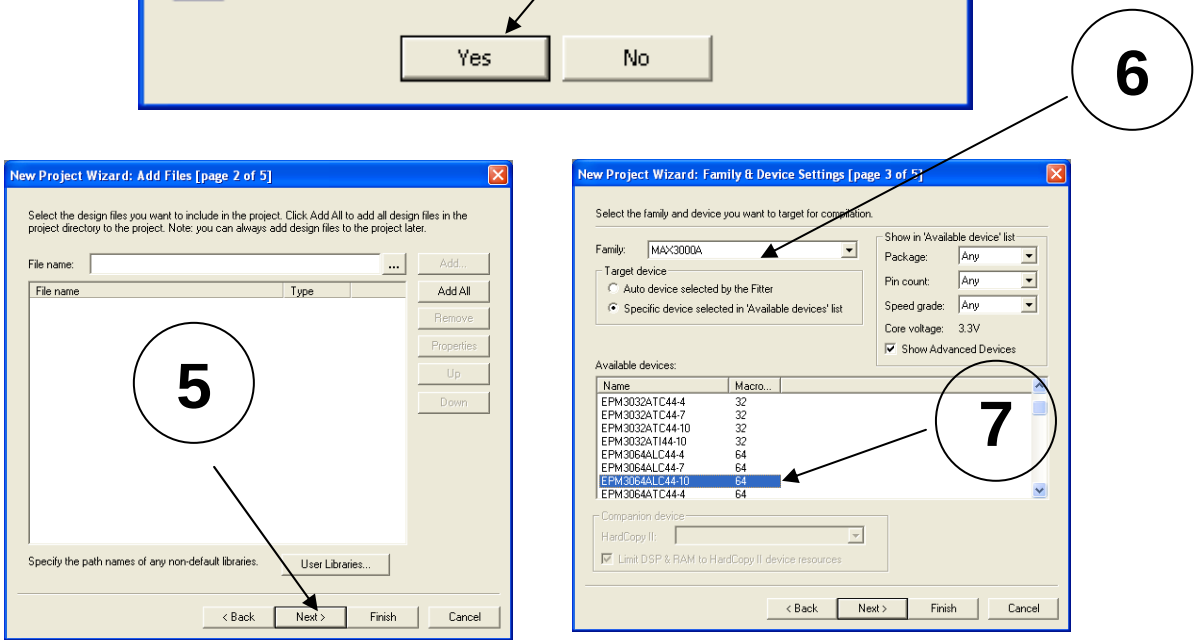


(1) คลิกที่ Next (2) คือเลือก Folders หรือ Directory ที่จะเก็บตัวโปรเจ็ค

(3) ตั้งชื่อตัวโปรเจ็ค (4) Directory ที่จะเก็บโปรเจ็ค ต้องการที่จะสร้างหรือไม่

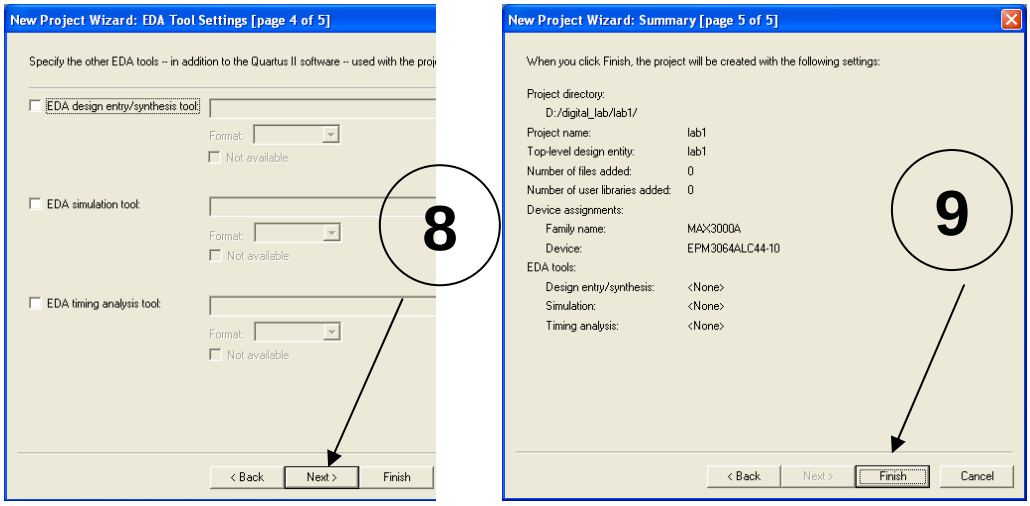
(5) คลิกที่ Next





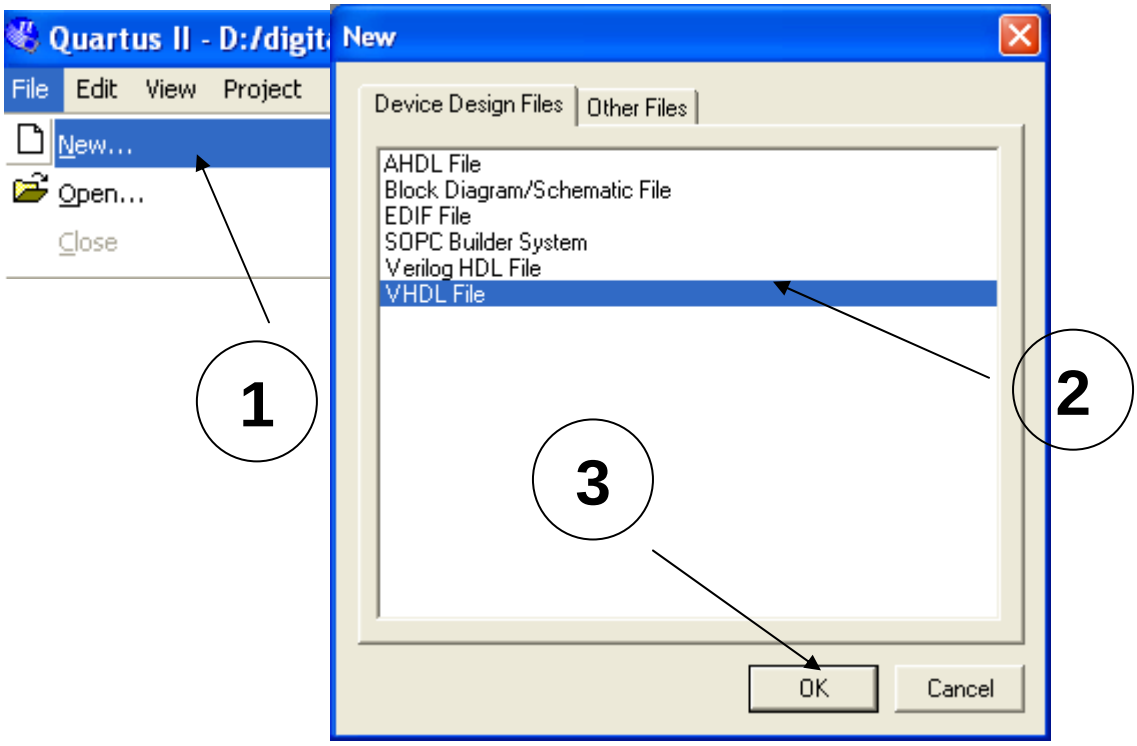
(6) เลือกตระกูลของเบอร์ชิพ

(7) เลือกหมายเลขของเบอร์ชิพที่จะใช้งาน



(8) คลิกที่ Next (9) คลิกที่ Finish

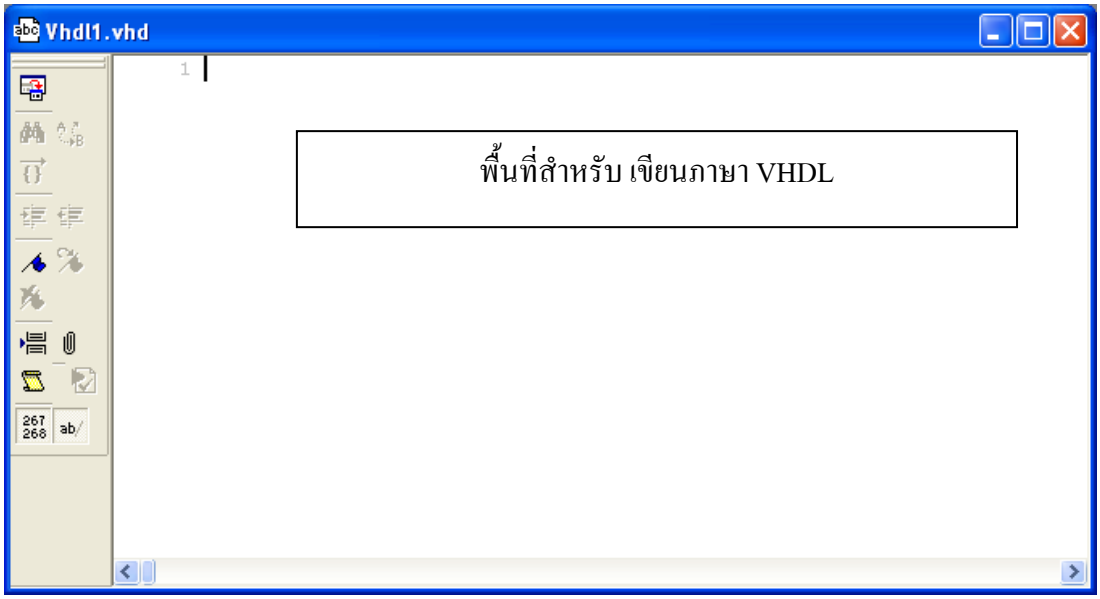
3. ทำการเลือกประเภทของไฟล์ที่จะใช้งาน โดยคลิกที่ **File / New** หลังจากนั้นเลือกชนิดของไฟล์ เป็น VHDL File



(1) คลิกที่ New (2) คลิกเลือก VHDL File

(3) คลิก OK

หลังจากนั้นจากนั้นจะปรากฏหน้าต่างของ **VHDL1.vhd** เกิดขึ้นดังแสดงในรูป ซึ่งเป็นพื้นที่สำหรับเขียนบรรยายหารทำงานวงจรดิจิตอลที่ต้องการออกแบบ



4. ในการใช้งานขั้นแรกนี้จะใช้สมการพีชคณิตง่ายๆ ประกอบคำอธิบาย นั่นก็คือ

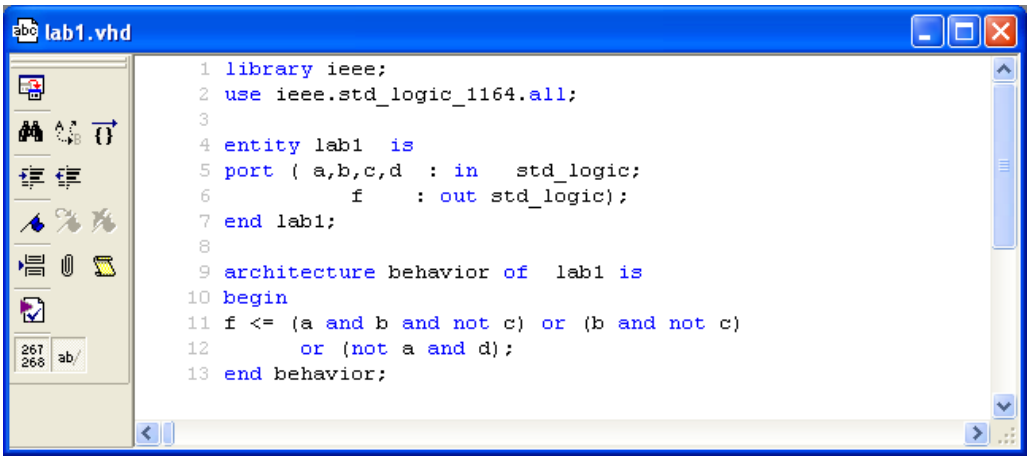
$$F = ab\overline{c} + b\overline{c} + \overline{a}d$$

ดังนั้นพื้นที่สำหรับเขียน โปรแกรมภาษา VHDL จะเขียนเป็น

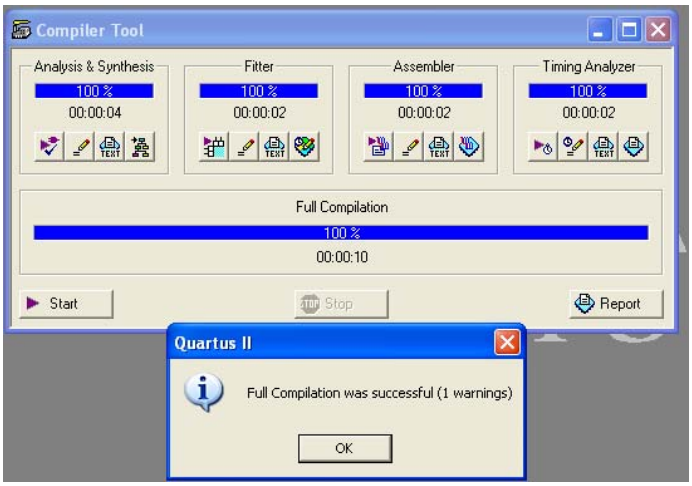
```
library ieee;
use ieee.std_logic_1164.all;

entity lab1  is
      port ( a,b,c,d  : in   std_logic;
             f     : out std_logic);
end lab1;

architecture behavior of  lab1 is
begin
      f <= (a and b and not c) or (b and not c)
             or (not a and d);
end behavior;
```

จากนั้นให้ทำการบันทึกโปรแกรมที่เขียนขึ้น โดยคลิกที่ **File / Save as** ให้ใส่ชื่อไฟล์เป็น **lab1.vhd**

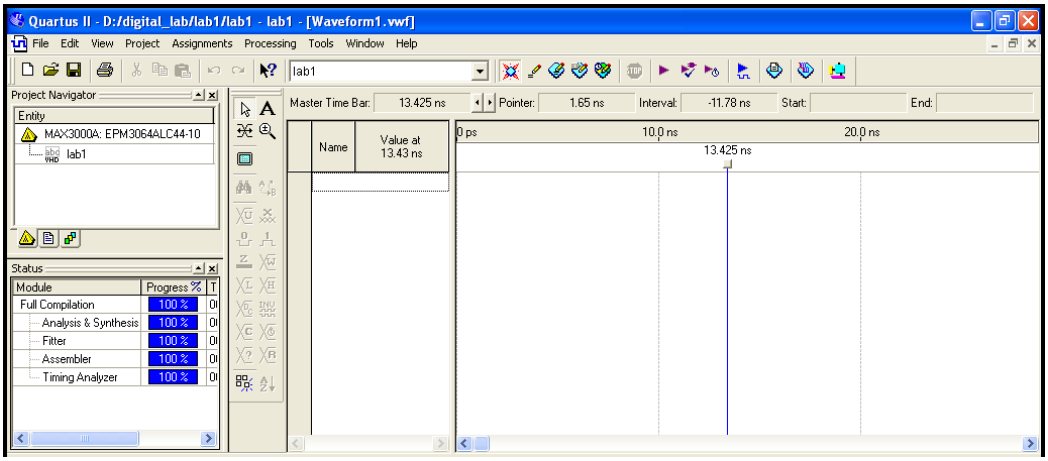
(โดยปกติแล้ว ตัวโปรแกรมฯจะทำการบันทึกชื่อแฟ้มข้อมูลเป็นชื่อเดียวกับชื่อโปรเจ็คเสมอ)



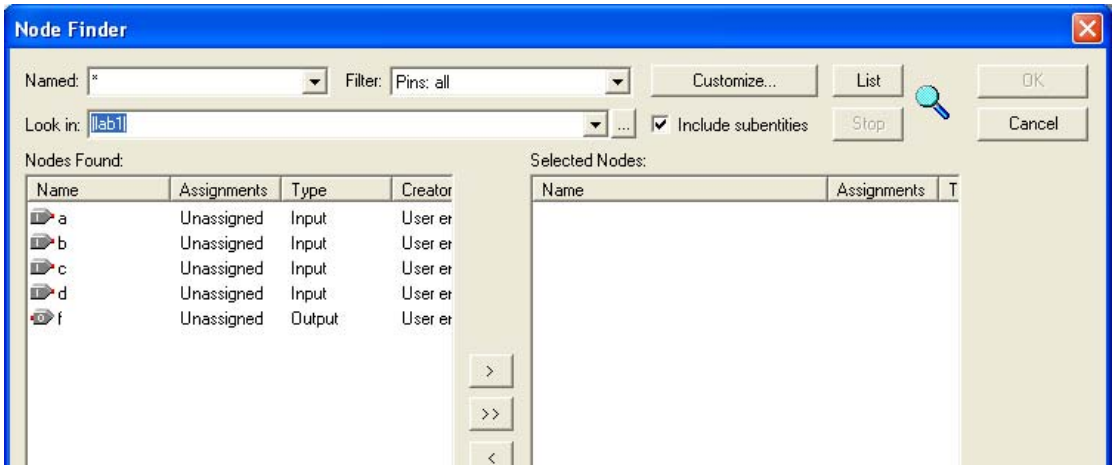
5. ทำการคอมไฟล์ โปรแกรมที่เขียนขึ้นมา โดยคลิกที่ **Processing / Complier Tool** จากนั้นคลิกที่ **Start** เมื่อคอมไฟล์เสร็จจะมีหน้าต่างรายงานผลการคอมไฟล์ error และ warning ดังรูป หากมีข้อผิดพลาดเกิดขึ้นจะมีข้อความสีแดงบอกว่า error เนื่องจากสาเหตุใด



6. ทำการจำลองผลหรือ simulate จากการเขียนบรรยายพฤติกรรมวงจร เริ่มต้นจะต้องสร้างหรือป้อนสัญญาณที่ต้องการให้วงจรก่อนโดยคลิกที่ **File / New** จะปรากฏหน้าต่างใหม่ขึ้นมาให้เลือก Other Files แล้วคลิกเลือก Vector Waveform File จะได้ดังรูปด้านล่าง

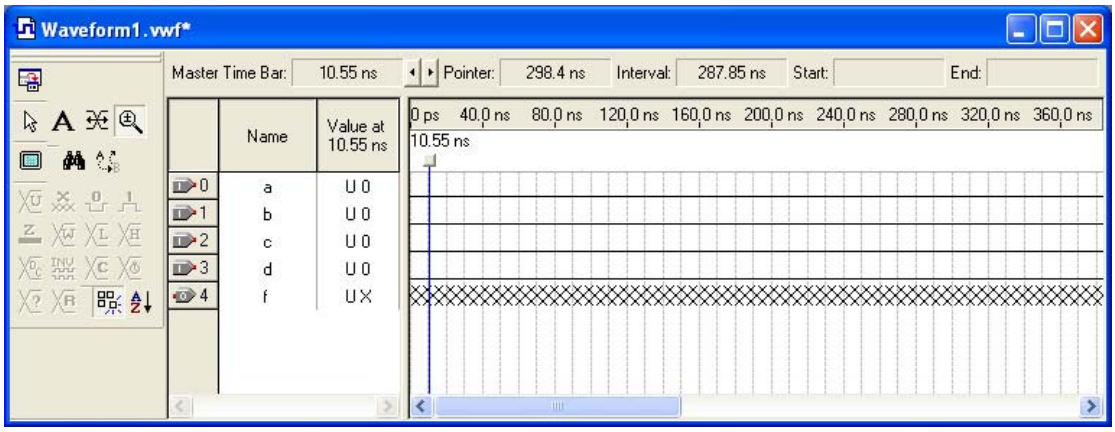


7. ทำการโหลด Node ต่างๆเข้ามา โดยคลิกที่ **Edit / Insert / Insert Node or Bus** จะปรากฏหน้าต่างเกิดขึ้นดังรูป หลังจากนั้นให้คลิกที่ Node Fider จะปรากฏหน้าต่างของ Node Finder ที่ตำแหน่งช่อง Filter : ให้เลือก **Pins : all** จากนั้นให้คลิกที่ปุ่ม **List**



จะพบขาอินพุทหรือเอาท์พุทที่อยู่ในวงจรปรากฏขึ้นมา(ในช่องทางซ้ายมือ) จากนั้นเลือก Node ทางซ้ายที่ต้องการ Simulate ไปไว้ช่องทางซ้ายมือ (ในตัวอย่างนี้ให้เลือกทั้งหมด) พร้อมคลิกปุ่ม >>

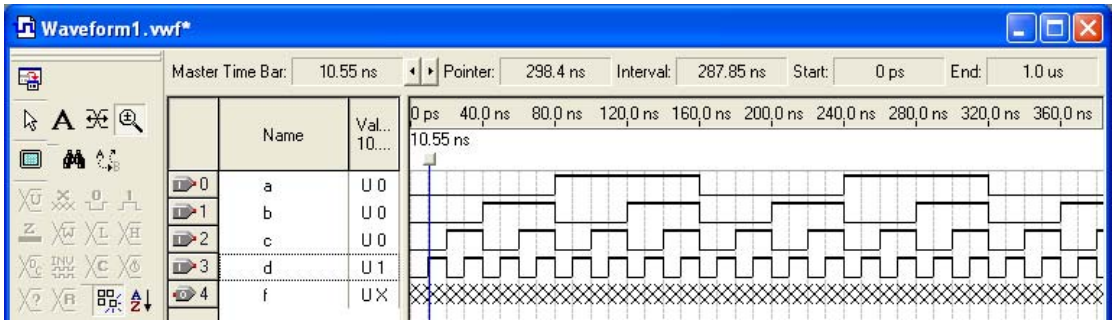
จะปรากฏหน้าต่างของ อินพุทและเอาท์พุท โดยที่อินพุทนั้นจะ เราจะต้องป้อนสัญญาณให้เอง ส่วนเอาท์พุทจะเป็นผลที่ได้จากการ Simulate จากสัญญาณอิพุทที่ป้อนให้ ดังรูปด้านล่าง



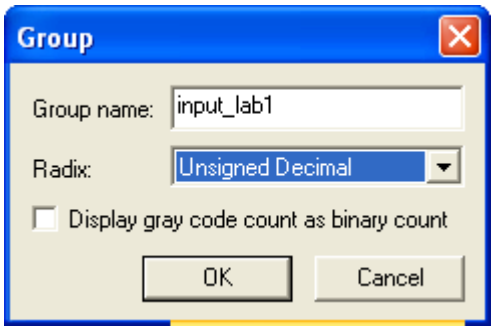
8. กำหนดเวลาสิ้นสุดในการจำลองผล โดยคลิกที่ **Edit / End Time** จากนั้นให้ใส่ค่าเวลาที่ต้องการ ในตัวอย่างนี้เลือก **1.0 us**

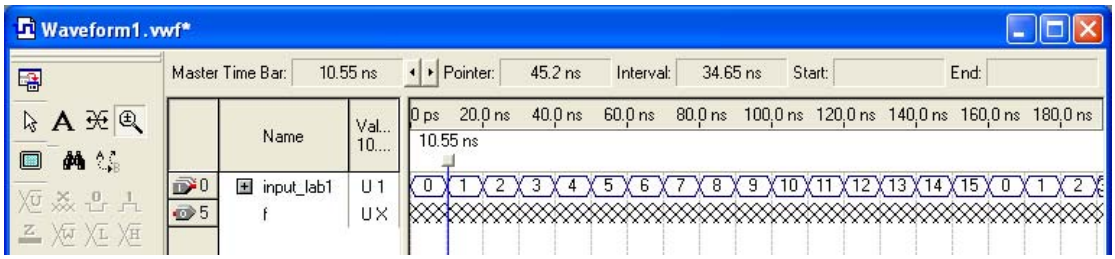
9. กำหนดขนาดของกริด โดยคลิกที่ **Edit / Grid size** ในตัวอย่างนี้เลือกขนาดกริดเท่ากับ **100 ns**

ทำการกำหนดรูปแบบของสัญญาณให้กับ Node อินพุท โดยให้เมาส์คลิกที่อินพุท b หรือ a ดังรูป ซึ่งจะปรากฏแถบสีฟ้าขึ้นมา จากนั้นกำหนดรูปแบบสัญญาณให้กับอินพุทดังกล่าว ซึ่งมีลักษณะเป็นพัลซ์ โดยคลิกที่ **Edit / Value / Clock** จะปรากฏหน้าต่างขึ้นมา ซึ่งจะมีช่อง Period ให้ใส่ 100 และ 200 (สำหรับการใส่ค่า Period นี้ ให้ใส่เป็นจำนวน 2^n เท่าของ Grid size เรียงจาก บิตต่ำสุดเป็นต้นไป)



เพื่อความสวยงามและการแสดงผลที่เข้าใจง่ายขึ้น เมื่อมีสัญญานอินพุทและเอาท์ทพุทที่มีมากกว่า 1 ตัว ให้ทำการจัดกลุ่มหรือ Group สัญญาณดังกล่าว ก่อนทำการ Group จะต้องเรียงลำดับบิตของสัญญาณ จากบิตสูงสุดจนถึงบิตต่ำสุด จากนั้นทำการลากเมาส์ครอบกลุ่มสัญญาณที่ต้องการจะ Group (จะปรากฏเป็นแถบสีฟ้า ดังรูป) จากนั้นให้คลิกที่เมนู **Edit / Grouping / Group** ใส่ชื่อ Group name และเลือก Radix

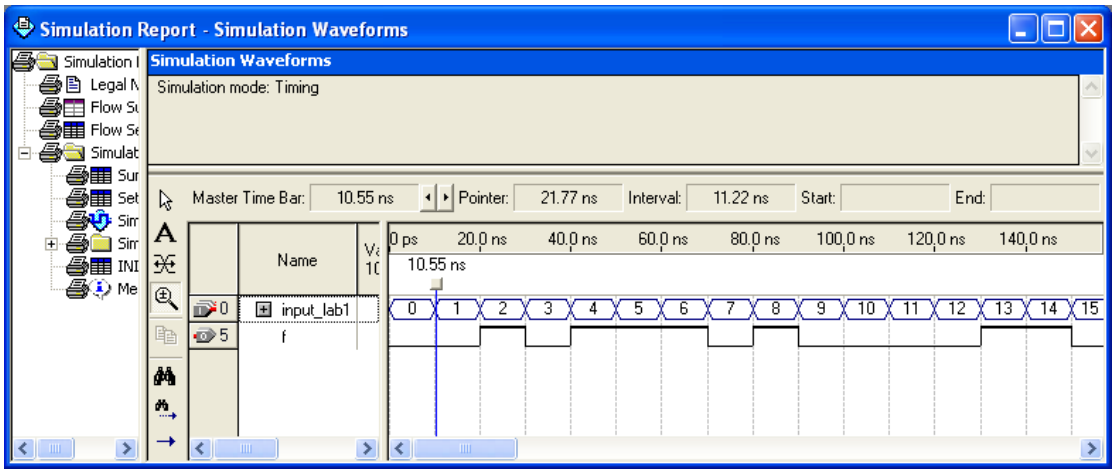




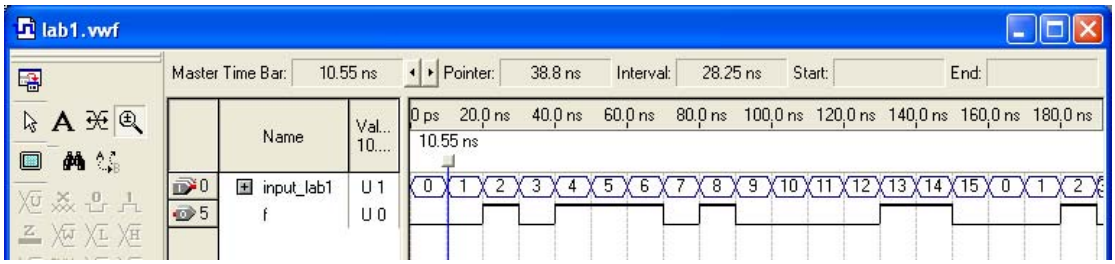
10. บันทึกไฟล์ Waveform โดยคลิกที่ **File / Save as** ชื่อไฟล์ที่จะบันทึกเป็นชื่อเดียวกับชื่อโปรเจ็ค คือ **Lab1.vwf**

11. ทำการจำลองผลการทำงานของวงจรที่สร้างขึ้น โดยคลิกที่ **Process / Simulator Tool** แล้วคลิก **Start**

- เมื่อทำการจำลองผลเสร็จ จะปรากฏหน้าต่างดังนี้



- นำผลการจำลองจาก Simulation Report บันทึกลงใน Waveform ของโปรเจ็คปัจจุบัน โดยคลิกที่ **Processing / Simulation Debug / Overwrite Vector Inputs with Simulation Outputs** จะได้ดังรูปด้านล่าง ซึ่งเป็นอันสิ้นสุดในการออกแบบวงจรดิจิตอลด้วยการบรรยายพฤติกรรมการทำงานและตรวจการออกแบบได้จากผลการจำลอง



บรรณานุกรม

1. นอ. ชาติชาย ดิษฐกุล ,เอกสารประกอบการสอนภาษา VHDL,โรงเรียนนายเรืออากาศ
2. Armstrong J. R. and F. G. Gray, VHDL Design Representation and Synthesis, Englewood Clis, NJ: Prentice Hall, 2nd Edition, 2000.
3. Bhasker J., VHDL Primer, Englewood Clis, NJ: Prentice Hall,3rd Edition, 1999.
4. Chang K. C., Digital Systems Design with VHDL and Synthesis—An Integrated Approach, Los Alamitos,CA: IEEE Computer Society Press, 1999.
5. Hamblen J. and M. Furman, Rapid Prototyping of Digital Systems, Boston: Kluwer Academic Publisher,2nd Edition, 2001.
6. Naylor D. and S. Jones, VHDL: A Logic Synthesis Approach, London: Chapman & Hall, 1997.
7. Navabi Z., VHDL Analysis and Modeling of Digital Systems, New York: McGraw-Hill, 1993.
8. Pellerin D. and D. Taylor, VHDL Made Easy, Englewood Cliffs, NJ: Prentice Hall, 1997.
9. Perry D. L., VHDL, New York: McGraw-Hill, 2nd Edition, 1994.
10. Yalamanchili S., Introductory VHDL from Simulation to Synthesis, Englewood Clis, NJ: Prentice Hall,2001.
11. Yalamanchili S., VHDL Starter’s Guide, Englewood Cliffs, NJ: Prentice Hall, 1998.
12. Volnei A. Pedroni.,Circuit Design with VHDL,Massachusetts,MIT Press,2004.