ตัวขับมอเตอร์ไฟฟ้ากระแสตรงแบบ PWM ด้วย FPGA

PWM DC MOTOR DRIVER WITH FPGA

นายจิระวัฒน์ กิจค้า
นายสิริพงษ์ ศรีโสม
นายอัชฌา เล็กบำรุง

รายงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมการวัดคุม
ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

PWM DC MOTOR DRIVER WITH FPGA

JIRAWAT KIJCAR

SIRIPONG SRISOM

ATCHAR LAKBUMRUNG

A THESIS SUHBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION TECHNOLOGY

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2004

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	ตัวขับมอเตอร์ไฟฟ้ากระแสตรงแบบ PWM ด้วย FPGA	
นักศึกษาผู้จัดทำ	นายจิระวัฒน์ กิจค้า	รหัสประจำตัว 45015546
	นายสิริพงษ์ ศรีโสม	รหัสประจำตัว 45015581
	นายอัชฌา เล็กบำรุง	รหัสประจำตัว 45015586
ปริญญา	วิศวกรรมศาสตรบัณฑิต	
สาขาวิชา	วิศวกรรมการวัดคุม	
ปีการศึกษา	2547	

อาจารย์ผู้ควบคุมปริญญานิพนธ์	ลายมือชื่อ
ผศ.เชื้อ นกอยู่	
ผศ.ดร. สาธิต อินทจักร์	

วัน/เดือน/ปี ที่สอบ วันอังคารที่ 24 มีนาคม พ.ศ. 2548

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(รศ.ประสิทธิ์ จุลเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	ตัวขับมอเตอร์ไฟฟ้ากระแสตรงแบบ PWM ด้วย FPGA
	PWM DC MOTOR DRIVER WITH FPGA
นักศึกษาผู้จัดทำ	นายจิระวัฒน์ กิจค้า
	นายสิริพงษ์ ศรีโสม
	นายอัชฌา เล็กบำรุง
อาจารย์ที่ปรึกษา	ผศ.เชื้อ นกอยู่
	ผศ.ดร.สาธิต อินทจักร์
ปีการศึกษา	2547

บทคัดย่อ

ในปัจจุบันการควบคุมมอเตอร์กระแสตรงมีการใช้งานกันมากงานอุตสาหกรรม ด้วยความก้าวหน้าของเทคโนโลยีการสร้างแม่เหล็กถาวรที่มีคุณภาพสูง ทำให้มอเตอร์กระแสตรงเป็นสิ่งที่สำคัญในระบบควบคุมเกือบทุกชนิดในปัจจุบัน

ดังนั้นปริญญานิพนธ์เล่มนี้จึงได้นำเสนอวิธีการขับมอเตอร์กระแสตรงโดยใช้ FPGA (Field Programmable Gage Array) ที่จำลองด้วยโปรแกรมเพื่อนำไปใช้ในการขับมอเตอร์กระแสตรง ซึ่งสัญญาณของมอเตอร์กระแสตรงที่ได้มาจากการเขียนโปรแกรม VHDL จะถูกแสดงออกมาเป็น PWM เพื่อการควบคุมในลักษณะต่างๆ คือ ควบคุมความเร็วของมอเตอร์กระแสตรง (Speed) การเริ่มและหยุดของมอเตอร์กระแสตรง (Run and Stop) ควบคุมการหมุนซ้ายและหมุนขวาของมอเตอร์กระแสตรง (Turn right and Turn left) และควบคุมความเร็วให้คงที่

Thesis Title	PWM DC Motor Driver With FPG
Authors	Mr. Jirawat Kijcar
	Mr.Siripong Srisom
	Mr.Atchar Lakbumrung
Thesis Advisor	Asst.Prof.Chuae Nokyoo
	Asst.Prof.Dr.Satid Intrajak
Year	2004

ABSTRACT

Currently, DC motor control has widely used in the industrial. With the high technology of producing permanent magnet, the DC motor has also high quality to use as a power source.

In this thesis, the DC motor driving is proposed by using FPGA (Field Programmable Gage Array) to control in the mode of PWM. The motor control is simulated on VHDL program. In our application, the motor controller is evaluated in several applications as following: speed control, start/stop, rotating in right and left direction, and fixing speed.

กิตติกรรมประกาศ

รายงานฉบับนี้สำเร็จลุล่วงไปได้ด้วยดีก็เพราะได้รับการอบรมและสั่งสอนจากทางท่านอาจารย์เชื้อ นกอยู่ ซึ่งเป็นอาจารย์ที่ปรึกษาและเป็นอาจารย์ประจำภาควิชาวิศวกรรมการวัดคุม สถาบันเทคโนโลยีพระจอมเกล้าคุณทหารลาดกระบัง ซึ่งท่านให้คำแนะนำและสั่งสอนในการทำรายงานฉบับนี้และยังเอื้ออำนวยอุปกรณ์ในการศึกษาอย่างต่อเนื่อง ทางคณะกลุ่มผู้จัดทำรู้สึกดีใจและซาบซึ้ง จึงขอกราบขอบพระคุณท่านอาจารย์เป็นอย่างสูง

ขอขอบพระคุณ กับอาจารย์ประจำภาควิชาวิศวกรรมการวัดคุมทุกท่าน ที่ให้คำแนะนำและให้คำปรึกษามาตลอดเวลาในช่วงที่ได้ศึกษาอยู่ ณ ที่นี้เป็นอย่างมาก ขอบคุณครับ

และขอขอบคุณทางคุณพ่อและคุณแม่ที่เป็นกำลังใจและส่งเสียมาตลอดครับทางกลุ่มของกระผมหวังว่ารายงานเล่มนี้คงเป็นประโยชน์ต่อทุกท่านที่ได้ศึกษาจากรายงานฉบับนี้ครับ

คณะผู้จัดทำ

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง VII

สารบัญรูป VIII

บทที่ 1 บทนำ 1

1.1 ความเป็นมาและเหตุจูงใจ 1

1.2 วัตถุประสงค์ของปริญญานิพนธ์ 2

1.3 ขอบเขตของปริญญานิพนธ์ 2

1.4 ขั้นตอนการศึกษา 2

บทที่ 2 ความรู้เบื้องต้นเกี่ยวกับทฤษฎีและหลักการของเทคโนโลยี FPGA 3

2.1 เทคโนโลยี FPGA 3

2.1.1 Field –Programmable Gate Array (FPGA) 3

2.1.2 โครงสร้างภายในของ FPGA 4

2.1.3 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว 4

2.1.4 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์ 5

2.1.5 เครื่องมือสำหรับการออกแบบ FPGA 8

2.2 เทคโนโลยีการออกแบบวงจรลอจิก (VHDL : Logic Design Technology) 9

2.2.1 องค์ประกอบที่สำคัญของ VHDL 9

2.2.2 รูปแบบของภาษา VHDL 10

2.2.3 Sequential processing 14

2.2.4 Wait Statements 16

2.2.5 ชนิดของข้อมูล (Data types) 16

2.2.6 เทคนิคการใช้ VHDL 20

สารบัญ(ต่อ)

หน้า

2.2.8 การออกแบบ buffer แบบต่าง ๆ 22

2.2.9 ข้อดีของ VHDL 23

2.2.10 VHDL ช่วยในการออแบบได้อย่างไรบ้าง 24

2.3 หลักการของมอเตอร์ (Motor principle) 24

2.3.1 แรงดันไฟฟ้าต้านกลับ (Back e.m.f. or Counter) e.m.f.) 25

2.3.2 สมการของแรงดันฟ้าต้านกลับของมอเตอร์ 26

2.3.3 สภาวะการเกิดประสิทธิภาพสูงสุด (Condition for Maximum Power).... 27

2.3.4 แรงบิด (Torque) 27

2.3.5 ความเร็วรอบของมอเตอร์ไฟฟ้ากระแสตรง(Speed of d.c. motor)..... 29

2.3.6 คุณลักษณะของมอเตอร์ไฟฟ้ากระแสตรง(Characteristics of d.c. motor) 31

2.3.7 การสูญเสียประสิทธิภาพของมอเตอร์(Loss and Efficiency) 37

บทที่ 3 อธิบาย Block diagram และโปรแกรม

3.1 บล็อกการทำงาน 38

3.2 Dead time 39

3.3 acc_control 39

3.4 left_right_control 40

3.5 Rpm_ setup 40

บทที่ 4 อุปกรณ์และผลการทดลอง

4.1 รายการอุปกรณ์ 41

4.2 กราฟสัญญาณ 41

4.3 รูปอุปกรณ์การทำงาน 47

4.4 ตารางบันทึกผลของมอเตอร์ตัวที่1ขณะไม่มีโหลด 49

4.5 ตารางบันทึกผลของมอเตอร์ตัวที่1ขณะมีโหลด 50

4.6 ตารางบันทึกผลของมอเตอร์ตัวที่2ขณะไม่มีโหลด 51

4.7 ตารางบันทึกผลของมอเตอร์ตัวที่2ขณะมีโหลด 52

4.8 รูปแสดงผลการวัดความเร็วรอบ 53

สารบัญ(ต่อ)

หน้า

บทที่ 5 สรุปผลและวิจารณ์ 54

5.1 ประโยชน์ที่ได้รับ 54

5.2 ปัญหาและอุปสรรค 54

5.3 ข้อเสนอแนะ 54

ภาคผนวก

ภาคผนวก ก 55

สารบัญตาราง

ตารางที่ **หน้า**

4.4 ตารางบันทึกผลผลของมอเตอร์ที่ 1 ขณะไม่มีโหลด 49

4.5 ตารางบันทึกผลผลของมอเตอร์ที่ 1 ขณะมีโหลด 50

4.6 ตารางบันทึกผลผลของมอเตอร์ที่ 2 ขณะไม่มีโหลด 51

4.7 ตารางบันทึกผลผลของมอเตอร์ที่ 2 ขณะมีโหลด 52

สารบัญรูป

รูปที่	หน้า
2.1 โครงสร้างภายในของ FPGA ตระกูล FLEX 10K	4
2.2 การโปรแกรมลงในชิพ	5
2.3 แสดง D Filp-Flop	9
2.4 แสดง D Filp-Flop มี preset และ reset	21
2.5 แสดง Tri-State	22
2.6 แสดง Bi-Directional Buffer	23
2.7 แสดงขดลวดที่มีกระแสไฟฟ้าไหลและวางอยู่ในสนามแม่เหล็ก	24
2.8 แสดงทิศทางของแรงดันไฟฟ้าต้านกลับ	25
2.9 แสดงความสัมพันธ์ระหว่างแรงบิดกับกระแสในอาร์เมเจอร์	32
2.10 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับกระแสในอาร์เมเจอร์	33
2.11 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิด	33
2.12 แสดงความสัมพันธ์ระหว่างแรงบิดกับกระแสอาร์เมเจอร์	34
2.13 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับกระแสอาร์เมเจอร์	34
2.14 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิด	35
2.15 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิดและแรงบิดกับกระแสอาร์เมเจอร์	35
3.1 แสดงบล็อกการทำงาน	38
3.2 แสดงบล็อกการทำงาน dead time	39
3.3 แสดงบล็อกการทำงาน acc control	39
3.4 แสดงบล็อกการทำงานleft_right_control	40
3.5 แสดงบล็อกการทำงานrpm_actual	40
4.1 กราฟแสดงสัญญาณขณะที่มอเตอร์หยุดหมุน	41
4.2 แสดงการปรับค่าดิวตี้ไซเคิ้ลที่ 50% ให้กับสัญญาณที่ขั้ว A และ B	42
4.3 แสดงการปรับค่าดิวตี้ไซเคิ้ลที่ 75% ให้กับสัญญาณที่ขั้ว A และ B	42
4.4 แสดงการปรับค่าดิวตี้ไซเคิ้ลที่ 100% ให้กับสัญญาณที่ขั้ว A และ B	43
4.5 กราฟแสดงสัญญาณการควบคุมการหมุนทางขวาของมอเตอร์	43
4.6 กราฟแสดงสัญญาณการควบคุมการหมุนทางซ้ายของมอเตอร์	44

สารบัญรูป(ต่อ)

รูปที่ **หน้า**

4.7 การปรับแต่ง dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 1.0 ไมโครวินาที 44

4.8 การปรับแต่ง dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.8 ไมโครวินาที 45

4.9 การปรับแต่ง dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.6 ไมโครวินาที 45

4.10 การปรับแต่ง dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.4 ไมโครวินาที 46

4.11 การปรับแต่ง dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.2 ไมโครวินาที 46

4.12 กราฟแสดงสัญญาณควบคุมระหว่างขั้ว A และ B ที่ความถี่ 20 KHz 47

4.13 Bord Poweracex1K 47

4.14 มอเตอร์กระแสตรง 24 V 48

4.15 วงจรขับมอเตอร์ 48

4.16 ออสซิลโลสโคป 48

4.17 แสดงสัญญาณขณะมอเตอร์ไม่มีโหลด 49

4.18 แสดงสัญญาณขณะมอเตอร์มีโหลด 50

4.19 แสดงสัญญาณขณะมอเตอร์ไม่มีโหลด 51

4.20 แสดงสัญญาณขณะมอเตอร์มีโหลด 52

4.20 แสดงค่าความเร็วรอบ 53

4.22 แสดงมอเตอร์ในขณะที่ไม่มีโหลด 53

4.23 แสดงมอเตอร์ขณะตอนมีโหลด 53

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุจูงใจ

ในปัจจุบันนี้ เทคโนโลยีที่ถูกผลิตขึ้นมาในแต่ละวันนั้นมีความหลากหลายและทันสมัยมากยิ่งขึ้น ทั้งทางด้านอิเล็กทรอนิกส์และคอมพิวเตอร์ เทคโนโลยีการสื่อสารสารสนเทศรวมทั้งด้านโทรคมนาคมด้วย จึงจำเป็นต้องมีการควบคุมการทำงานที่หลากหลายแตกต่างกันออกไปและการควบคุมหรือระบบการควบคุม (Control System) จึงต้องมีการทำงานที่หลากหลายแตกต่างกันออกไปเพื่อให้เหมาะสมกับการนำไปใช้และนำไปทำงานในลักษณะของงานที่ซับซ้อนและยุ่งยากแตกต่างกันออกไป ในส่วนของทางด้านอิเล็กทรอนิกส์ได้มีการพัฒนาวงจรรวม Integrating Circuit or IC ขึ้นมาเพื่อถูกนำไปใช้ในงานด้านต่างๆและสะดวกสบายและรวดเร็วยิ่งขึ้นโดยการรวมวงจรที่ยุ่งยากและซับซ้อนเข้าด้วยกัน อย่างเช่น วงจรจำพวก LogicGateand Amplifier เป็นต้น และสามารถพัฒนาไอซีที่สามารถโปรแกรมได้ โดยใช้ภาษา Basicพื้นฐานทางด้านลอจิก เช่น IC ในตระกูล Z-80 ซึ่งเป็น IC Microprocressor และพัฒนาไปถึงไอซีในตระกูล MCS-51 ซึ่งเป็น ICMicrotroller ซึ่งทั้ง Z-80 และ MCS-51มีลักษณะการทำงานที่คล้ายกันคือสามารถเขียนโปรแกรมควบคุมการทำงานของระบบควบคุม (ControlSystem) ต่างๆ ตามวัตถุประสงค์ที่เราต้องการ ซึ่งในการศึกษาโปรแกรม FPGA ณ ที่นี้เป็นการศึกษาในการสร้างโปรแกรมขึ้นมาในรูปแบบของ Logic Gate เพื่อนำมาควบคุมการทำงานของมอเตอร์ไฟฟ้ากระแสตรง โดยจะสร้างโดย ProgramFPGA และต้องศึกษาโครงสร้างของโปรแกรม FPGAและภาษาVHDLในรูปแบบขอคำสั่งต่างๆกันไปตามลักษณะการนำโปรแกรมไปใช้ในการควบคุมในอุปกรณ์ไฟฟ้าทั่วไปสำหรับ FPGA แล้วนับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่างรวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTLโครงสร้างภายในของ FPGA ประกอบไปด้วยอะเรย์ของลอจิกเกจต่างๆ มากมาย ซึ่งในปัจจุบันความจุเกจภายในตัวชิพ FPGA ได้เพิ่มขึ้นจากระดับไม่กี่พันตัวจนถึงระดับล้านตัว ซึ่งสามารถรองรับวงจรดิจิตอลที่มีความสลับซับซ้อนได้เป็นอย่างดีนอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่าย ซึ่งในปัจจุบันการออกแบบวงจรโดยใช้ FPGA กำลังเป็นที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อยๆ

1.2 วัตถุประสงค์ของโครงงาน

เพื่อให้ได้ความรู้พื้นฐานเกี่ยวกับการทำงานของโปรแกรม FPGA และศึกษาเกี่ยวกับระบบโครงสร้างของโปรแกรม FPGA สามารถเขียนโปรแกรมใช้งานเกี่ยวกับ FPGA ในด้านการนำโปรแกรมไปควบคุมการ ทำงานของมอเตอร์ไฟฟ้ากระแสตรง เรียนรู้คำสั่งและโปรแกรมของ FPGA เช่น Asynchronous Counter Comparator Multiplex เรียนรู้ในส่วนของมอเตอร์ไฟฟ้ากระแสตรงและการใช้โปรแกรม MAX-PLUS 2 ในการสร้างวงจรทางด้าน Logic Gate และสามารถนำไปพัฒนาได้ศึกษาการต่อใช้งานร่วมกับอุปกรณ์อื่นได้ และนำไปประยุกต์และสามารถนำไปใช้งานในด้านการควบคุมอุปกรณ์อื่นๆได้โดยวิธีการแบบอื่น

1.3 ขอบเขตของปริญญานิพนธ์

จากการศึกษาโปรแกรม FPGA ในภาคเรียนนี้คือ Project เป็นการนำโปรแกรมFPGAมาใช้เพื่อทำการควบคุมการทำงานของมอเตอร์กระแสตรง โดยจะทำการควบคุมการหมุนซ้ายและหมุนขวาของมอเตอร์ (Turn right and Turn left) และการเพิ่มความเร็ว (Speed) ในการหมุนของมอเตอร์การหยุดและเพิ่ม-ลดและรักษาระดับความเร็วรอบให้คงที่การเริ่มหมุนและหยุดของมอเตอร์ (Start and Run)

1.4 ขั้นตอนการศึกษา

สำหรับการทำปริญญานิพนธ์นี้เริ่มจากศึกศึกษาค้นคว้าโครงสร้าง คำสั่งต่างๆ และการออกแบบโดยโปรแกรม FPGA โดยใช้สัญญาณ PWM เพื่อนำไปควบคุมการทำงานของมอเตอร์กระแสตรง ในรูปแบบของลอจิกเกจ

บทที่ 2
ความรู้เบื้องต้นเกี่ยวกับทฤษฎีและหลักการของเทคโนโลยีFPGA

2.1 เทคโนโลยีของ FPGA

2.1.1 Field-Programmable Gate Array (FPGA)

FPGA เป็นอุปกรณ์ในลักษณะของ Programmable Device ที่เราสามารถโปรแกรมตัวมันให้เป็นวงจรดิจิตอลใดๆก็ได้ สำหรับ FPGA แล้วนับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่าง รวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTL โครงสร้าง ภายในของ FPGA ประกอบไปด้วยอะเรย์ของลอจิกเกทต่างๆมากมาย

ซึ่งในปัจจุบันความจุเกทภายในตัวชิพ FPGA ได้เพิ่มขึ้น จากระดับไม่กี่พันตัวจนถึงระดับล้านตัวซึ่งสามารถรองรับวงจรดิจิตอลที่มีความสลับซับซ้อนได้เป็นอย่างดี นอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่ายซึ่งในปัจจุบัน การออกแบบวงจรโดยใช้ FPGA กำลังเป็น

ที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อยๆ ในปัจจุบันมี FPGA อยู่4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรมที่แตกต่างกันไป

นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบคือ การโปรแกรม โดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรม โดยการใช้หน่วยความจำ

1. การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

1.1 Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียวเท่านั้น ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อมต่อจะขาดจากกัน

1.2 Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรม แล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

2. การโปรแกรมโดยใช้หน่วยความจำ

2.1 EEPROM Based FPGA

FPGA ที่ใช้การโปรแกรมแบบนี้มักเรียกว่า CPLD ซึ่งเทคโนโลยีที่ใช้จะเหมือนกับ EEPROM ทำให้มีความจุของเกทต่ำโดยทั่วไปจะน้อยกว่า 20,000 เกทแต่ข้อดีของ EEPROM Based FPGA คือสามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง และในการโปรแกรมจะใช้ทรานซิสเตอร์ 1 ตัวต่อ 1 บิท ซึ่งการโปรแกรมสามารถทำได้ประมาณ 10,000 ครั้ง

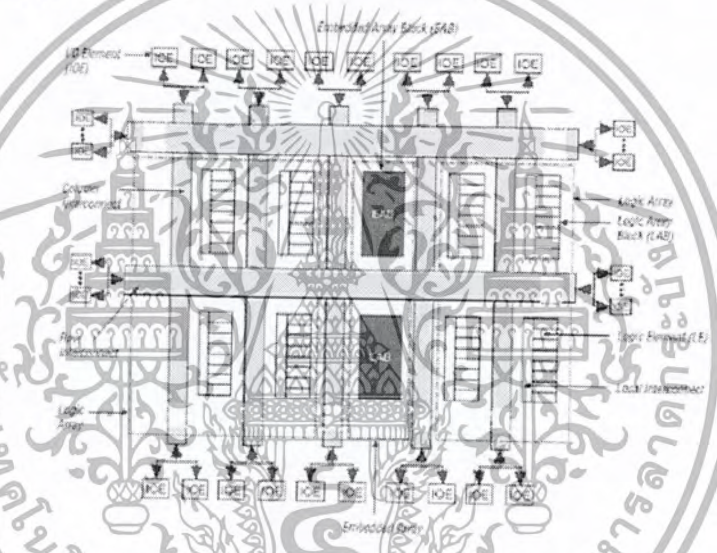
2.2 SRAM Based FPGA

FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM)

ทำให้สามารถ โปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 - 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการ โปรแกรมน้อย (ระดับ nSec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และเหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมในภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

2.1.2 โครงสร้างภายในของ FPGA

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อกลอจิกที่สามารถทำการโปรแกรมได้ดังรูปที่ 2.1



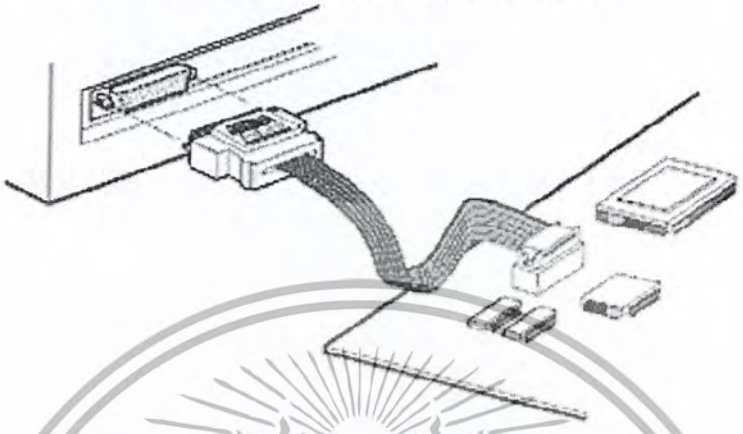
รูปที่ 2.1 โครงสร้างภายในของ FPGA ตระกูล FLEX 10K

2.1.3 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

1. ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้วต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2. มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language) เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตรฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3. การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นานเพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 2.2 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 2.2 การโปรแกรมลงในชิพ

2.1.4 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ ในกรณีของการออก แบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้วจึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำ ได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออก แบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี สำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

1. การสังเคราะห์วงจร(Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรม ของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์ นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น

FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตาม เทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบ ด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆจึงทำให้ผล ที่ได้มีประสิทธิภาพและในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดลแต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา(Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจร ที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับ เทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำ การสังเคราะห์วงจรเสร็จแล้ว ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้นเป็นอย่างไรเช่นมีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้น ตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตามที่กำหนด

2. การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำโดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิป-ฟลอบ (flip-flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ในอุปกรณ์ FPGAหลังจากทำขั้นตอน นี้เสร็จแล้วผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGAไปเท่าไร ส่วนข้อมูลทางเวลานั้นผู้ออกแบบจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความ หน่วงลอจิก(logic delay) ส่วนซอฟต์แวร์จะรวมเอาซอฟต์แวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement & Routing) เป็นไปอย่างต่อเนื่อง

3. การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควร จะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะ ได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี ความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด การวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กันโดยเฉพาะส่วนที่มีการเชื่อมต่อสัญญาณด้วยกันนอกจากนั้นการ

กำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ก็จะมีผลโดยตรงเลยคือซอฟต์แวร์จะวาง I/O ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่ง บางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขา I/O ควรกำหนดตำแหน่งให้เหมาะสมหรือไม่ก็ให้ซอฟต์แวร์จัดการเอง

4. การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ขั้นตอนนี้จะทำ ต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมด (เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกิน ค่าที่กำหนดในข้อบังคับ ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือผู้ออกแบบจะทำการ เชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับ ทางเวลา จะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

5. ความหน่วงด้านเวลา (Delay)

ในการทำFPGAนั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วง ที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์FPGAเองและความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay) เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการ เชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์และเชื่อมต่อสัญญาณผู้ออกแบบควรกำหนดข้อบังคับทางเวลาเพื่อให้ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อให้ได้ผลลัพธ์ที่ดีขึ้นค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความ หน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้น เป็นไปตามข้อกำหนด หรือไม่

6. การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการ ทำงานของโมเดลว่าถูกต้องหรือไม่มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น Model Sim ของบริษัท Model Technology หรือ Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดล เกิดขึ้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้ เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจรต้อง

ทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้นผู้ออกแบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจ สอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้วเพื่อตรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตาม ข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงานแล้วยังต้องตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวม สามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

7. การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement & Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้เป็นข้อมูลวงจร (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มี ฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบต้องการ ซึ่งในขั้นตอนนี้จะใช้วิธีที่แตกต่างกันออกไปสำหรับ อุปกรณ์ FPGA ของแต่ละบริษัทผู้ผลิตคือ ในกรณีที่เป็นอุปกรณ์ FPGA ชนิดที่ต้องโปรแกรมโดย วิธี SRAM นั้น ในการใช้งานผู้ออกแบบจะต้องเก็บข้อมูลวงจรไว้ในหน่วยความจำประเภท EPROM หรือ serial PROM ด้วยเพื่อจะใช้งานสะดวกขึ้น คือในการใช้งานโมเดลครั้งต่อไปไม่ ต้องดาวน์โหลดข้อมูลวงจรจากเครื่องคอมพิวเตอร์อีก เพราะมีข้อมูลวงจรเก็บอยู่ในหน่วยความจำอยู่ แล้ว แต่กรณีที่อุปกรณ์ FPGA เป็นชนิดที่โปรแกรมโดยใช้วิธี EPROM หรือ Anti fuse ก็ไม่จำเป็นต้องมีหน่วยความจำสำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูล วงจรลงไป ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออกแบบไว้ได้เลย

2.1.5 เครื่องมือสำหรับการออกแบบ FPGA

จะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่า ASIC มากเพราะใช้เวลาน้อยกว่ามากด้วย ส่วน สำคัญที่ใช้ในการทำ FPGA คือ ซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่วดาวน์โหลดลงในอุปกรณ์ FPGAซึ่งซอฟต์แวร์ที่ใช้ต้องเป็นซอฟต์แวร์ที่ทำงานต่อเนื่องกันได้สำหรับซอฟต์แวร์ที่ใช้ทำการจำลองการทำงานของวงจรนั้น ต้องสามารถใช้งาน

ต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบ เพราะโมเดลที่ได้จาก การทำขั้นตอนต่างๆ ด้วยซอฟต์แวร์ต่างๆ ต้องเอามาจำลองการทำงานได้และในการจำลองการทำงานของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดทั้งระบบ เพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่ จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ ที่ใช้บนพีซี (PC) มากขึ้นซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

2.2 เทคโนโลยีการออกแบบวงจรลอจิก (VHDL : Logic Design Technology)

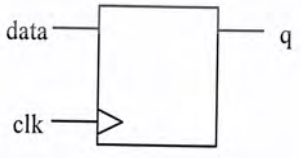
เทคโนโลยีของการออกแบบวงจรลอจิกได้มีการพัฒนาไปอยู่เสมอ จนได้มีการพัฒนาภาษาโปรแกรมที่ใช้ในการออกแบบวงจรลอจิกขึ้นมา VHDL เป็นหนึ่งในนั้นที่ได้มีการพัฒนาขึ้นมา VHDL ย่อมาจาก VHSIC Hardware Description Language (VHSIC =Very High Speed Integrated Circuit) มันก็คือภาษาที่ใช้ในการออกแบบวงจรลอจิกเพื่อผลิตเป็นชิพไอซีขึ้นมา VHDL เป็นภาษาระดับสูงที่ใช้ในการออกแบบระบบและวงจรลอจิกเป็นภาษาที่สามารถออกแบบวงจรได้ในระดับต่างๆ จะประกอบไปด้วยโครงสร้างต่างๆที่ถูกออกแบบขึ้นมา ในระดับที่สูงขึ้นตัวภาษานั้นสามารถที่จะออกแบบระบบโดยไม่คำนึงกระบวนการ หรือวิธีการของวงจรเพราะวงจรจะถูกสร้างในรูปแบบของฟังก์ชั่น เราจะพิจารณาเพียงจุดมุ่งหมายของการออกแบบวงจรเท่านั้น เป็นภาษามาตรฐานที่ IEEE ได้ให้การยอมรับ ข้อดีคือเป็นภาษาที่สามารถปรับเปลี่ยนโครงสร้างให้เข้ากับระบบฮาร์ดแวร์ต่างๆ ได้เป็นอย่างดี และอ่านเข้าใจได้ง่าย

2.2.1 องค์ประกอบที่สำคัญของ VHDL

VHDL มีองค์ประกอบที่สำคัญอยู่ 2 ส่วนคือ Entity และ Architecture

1 Entity เป็นเสมือน Block ที่เราสร้างขึ้นมาเพื่อจะบอกถึงจุดเชื่อมต่อภายนอก (I/O) ว่ามีอะไรบ้าง โดยไม่ต้องมีการอธิบายโครงสร้างภายใน

2 Architecture ใช้ในการอธิบายโครงสร้างภายในของ Entity



รูปที่ 2.3. แสดง D Filp-Flop

```
entity dff is
port (data,clk:in std_logic;
```

```
        q:out std_logic);
end dff;
architecture behav of dff is
 begin
  process (clk)
   begin
        if (clk'event and clk = '1') then
        q <= data;
        end if;
  end process;
end behav;
```

โปรแกรมนี้เป็นตัวอย่างของการเขียน โปรแกรมเพื่ออธิบายคุณสมบัติของ D F/F (รูปที่ 2.3) Entity มีการประกาศอินพุทอยู่ 2 ตัวคือ data กับ clk เอาท์พุทคือ q ส่วน Architecture มีการบอกการทำงานของ D F/F ว่าถ้าสัญญาณ clk มีการเปลี่ยนแปลงจาก 0 เป็น 1 ให้มีการส่งค่า data ไป q

2.2.2 รูปแบบของภาษา VHDL

- รูปแบบการเขียน Architecture ของ VHDL

VHDL มีรูปแบบการเขียนอยู่ 3 รูปแบบ

1 Structural Model การอธิบายวงจรโดยใช้โมดูลต่างๆมาเชื่อมต่อกันให้เห็นโครงสร้างภายใน

2 Behavioral Model การอธิบายการทำงานของวงจรในระดับลอจิกเกท

3 Sequential Model การอธิบายขั้นตอนการทำงานโดยใช้ Sequential Statement ทำงานทีละขั้นตอน

ต่อไปนี้จะเป็นตัวอย่างการเขียนโปรแกรมเพื่ออธิบายวงจร RS F/F โดยใช้ Architecture ที่แตกต่างกัน

Structural Model

```
u1: nand2 port map (set, q, qb);
u2: nand2 port map (reset, qb, q);
```

Behavioral Model

```
qb<=not (q and set) after 2 ns;
q <=not (qb and reset) after 2 ns;
```

Sequential Model

```
process (set,reset)
```

```
begin
        if set = '0' and reset = '1' then
          q <= '0' after 2 ns;
          qb <= '1' after 4 ns;
        if . . .
          . . .
end process;
```

สำหรับรูปแบบการเขียนที่เราจะเลือกใช้นั้น ขึ้นอยู่กับความเหมาะสมของฮาร์ดแวร์ เราสามารถที่จะใช้รูปแบบต่างๆมาร่วมกันได้อยู่ใน Architecture ตัวเดียวกันก็ขึ้นอยู่กับความเหมาะสมเช่นกัน จะเห็นว่าในแต่ละแบบก็มีจุดเด่นของตัวมันเอง แต่ทั้ง 3 แบบก็ให้ผลลัพธ์เหมือนกัน

Sequential มีการอธิบาย Architecture เป็นลำดับขั้นตอนที่ชัดเจน เข้าใจง่าย โดยปกติแล้วโปรแกรมที่อยู่ภายใน Architecture นั้นจะมีการ process พร้อมๆกันทุกบรรทัด (เหมือนวงจรจริงที่มีการเชื่อมต่อสัญญาณกันทั้งวงจร เมื่อมีสัญญาณ input ตัวใดเปลี่ยน output ก็จะมีการเปลี่ยนตามทันที จะมี delay ก็เพียงเล็กน้อย) แต่แบบ Sequential สามารถจะมองเป็นขั้นตอนได้โดยใส่ Process Statement คร่อมเข้าไปในโปรแกรม ส่วน Structural ก็มีการดึง Component NAND มาจะมองเห็นการเชื่อมต่อที่ชัดเจน ส่วน Behavioral นั้นมีการอธิบายคุณสมบัติระดับลอจิกเกท

- Behavioral Modeling

ในหัวข้อที่แล้วได้กล่าวถึง Structural Model ไปบ้างแล้วในหัวข้อนี้เป็น Behavioral Model ซึ่งจะกล่าวถึง Statement และฟังก์ชั่นต่างๆที่นิยมใช้กับ Behavioral Model

1. การใช้ Selected signal assignment และ Conditional signal assignment statement

Statement ที่นิยมใช้อีก 2 แบบสำหรับ Behavioral Model ก็คือ Conditional และ Selected signal assignment โปรแกรมต่อไปนี้จะแสดงตัวอย่างของการใช้ 2 Statement ดังกล่าว

```
with sel select
  q<=i0 after 10 ns when 0,
      i1 after 10 ns when 1,
      i2 after 10 ns when 2,
      i3 after 10 ns when 3,
     'x' after 10 ns when others;
```

```
sel<=   0 when a = '0' and b = '0' else
        1 when a = '1' and b = '0' else
        2 when a = '0' and b = '1' else
        3 when a = '1' and b = '1' else
        4;
```

โปรแกรมทางซ้ายคือ Selected signal assignment โดย q จะเลือกส่งค่า i0, i1, i2, i3 ออกไป

ขึ้นอยู่กับค่าของ sel (sel เป็น integer) ส่วนโปรแกรมทางขวาก็คือ Condition signal assignment โดยตัวแปร sel จะมีค่าเป็น 0, 1, 2, 3 หรือ 4 ขึ้นอยู่กับเงื่อนไขว่าเงื่อนไขใดถูกต้อง

2. การใช้ delay

การใช้ delay ใน VHDL นั้นเป็นการทำงานเลียนแบบการทำงานจริงๆของวงจรที่จะมี delay ในการทำงานอยู่เช่นกัน

a <= b; (ส่งค่า b ไป a ทันทีโดยไม่มี delay time)

a <= b after 10 ns; (ส่งค่า b ไป a หลังจากที่ผ่านไปแล้ว 10 ns)

VHDL จะมี delay อยู่ 2 แบบ

1 Inertial delay ก่อนจะส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณทางขวาจะต้องคงค่านั้นได้นานเท่ากับค่า delay ดังนั้น delay แบบนี้สัญญาณของตัวส่งกับตัวรับไม่จำเป็นจะต้องเหมือนกัน ดังตัวอย่างการใช้งาน

a <= b after 20 ns; (a จะได้รับ ค่าจาก b เมื่อค่าของ b ไม่เกิดการเปลี่ยนแปลงเป็นเวลา 20 ns)

2 Transport delay เป็นการส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณของทั้งตัวส่งและตัวรับจะเหมือนกัน แตกต่างกันก็ตรงที่สัญญาณตัวรับจะช้ากว่าเป็นเวลาเท่ากับที่ delay ไว้ ดังตัวอย่างการใช้งาน

a <= transport b after 20 ns; (สัญญาณของ b จะเหมือนกับ a แต่ b จะมีสัญญาณที่กว่า a อยู่ 20 ns)

-การใช้งาน Generics

Generics เป็นเครื่องมือที่ใช้ในการส่งข้อมูลต่างๆให้ Entity เช่น ค่า delay time, ค่าความจุ, ค่าความต้านทาน, ความกว้างของ data-path หรือว่า ความกว้างของ signal เป็นต้น ใช้งานได้โดยเราจะเพิ่มเติมส่วน Generic นี้แทรกเข้าไปใน Entity Block ดังตัวอย่างโปรแกรมดังนี้

```
entity buff is
  generic(delay:time;load:integer);
  port(a:in bit; b:out bit);
end buff;

architecture buff1 of buff is
  begin
    b<= a after(delay * load);
end buff1;
```

จากตัวอย่างโปรแกรมเป็น buffer ที่มีการส่งค่า delay และ load ให้กับตัว buffer โดยจะนำค่าเหล่านั้นไปใช้ใน architecture ดังตัวอย่าง ภายใน Architecture จะมีการส่งค่าจาก a ไปยัง b โดยมี delay หน่วงไว้โดยค่า delay นั้นเท่ากับ (delay * load) ซึ่งหลังจากที่เราได้เขียน entity buffer ตัวนี้ขึ้นมาแล้วก็สามารถจะนำไปใช้กับ entity ตัวอื่นได้เสมือนการเรียกใช้งาน function โดยเราจะใช้ entity buffer ตัวนี้ใน entity ตัวใดเราก็จะแทรก Component ไว้ใน Architecture ตัวนั้นโดยแทรกไว้

ก่อน begin ดังตัวอย่าง

```
architecture test1 of test is
component buff
        generic(delay : time; load : integer);
        port( a : in bit; b : out bit);
end component;
begin
```

เมื่อเราแทรก entity ตัวนั้นลงไปแล้วเราก็สามารถเรียกใช้ได้เสมือนเป็นฟังก์ชั่นตัวหนึ่ง และมีการส่งค่าให้ได้ด้วยดังนี้

u1: buff generic map (13 ns, 2) port map (ina, outb); (เป็นการเรียกใช้ในรูปแบบ Structure)

- การใช้งาน Block Statements

เป็น Statement ที่เข้ามาช่วยให้ Architecture ของเรานั้นดูเป็นระบบ และสัดส่วนมากขึ้น อีกทั้งยังทำให้เราสามารถประกาศตัวแปรแบบ Local ขึ้นมาใช้ได้ ทำให้ตัวแปรมีการแยกแยะใช้เป็นสัดส่วน ตัวแปรที่ประกาศไว้ใน Block ก็สามารถใช้ครอบคลุมได้ภายใน Block เท่านั้น (เหมือนในภาษาปาสคาล)

```
[label] : block [ ( condition )]
        [ประกาศตัวแปร]
begin
        -- statements
end block [label];
```

ในแต่ละ Architecture สามารถที่จะมีได้หลาย block ขึ้นอยู่กับผู้เขียนว่าจะแบ่งเป็นสัดส่วนเช่นใด และในแต่ละ block ก็สามารถมี block ซ้อนอยู่ข้างในได้ด้วย โดย condition ที่อยู่หลัง block นั้นคือคุณสมบัติของ Guarded Blocks ที่เพิ่มเติมเข้ามาเพื่อนำเงื่อนไขดังกล่าวไปใช้ใน statement ดูได้จากตัวอย่าง

```
architecture test1 of test is
begin
   g1 : block (clk = '1')
  begin
   q <= guarded d after 5 ns;
  end block g1;
```

```
end latch_guard;
```

การทำงานของโปรแกรมก็คือ จะเห็นว่าจะมี keyword GUARDED เพิ่มเข้ามาซึ่ง GUARDED ตัวนี้จะให้ค่าเป็นจริงเมื่อ clk = '1' นอกจากนี้จะเป็นเท็จ ดังนั้น statement d ที่ส่งค่าให้ q จะทำงานก็ต่อเมื่อ Guarded เป็นจริง

2.2.3 Sequential Processing

สำหรับการออกแบบวงจรที่มีความขนาดใหญ่และซับซ้อนนั้น รูปแบบของ Structure และ Behavioral Model นั้นยังไม่เพียงพอ รูปแบบของ Sequential เป็นรูปแบบที่มีความสำคัญมากในการใช้ เพราะมีรูปแบบอย่างเดียวกันกับภาษาระดับสูงอย่าง C หรือ ปาสคาลที่มีการทำงานแบบ Sequential Statement

- Process Statement

การที่จะทำให้ statement ใน VHDL นั้นสามารถทำงานแบบ Sequential ได้นั้นเราจะต้องใช้ statement ที่เรียกว่า Process เข้าไปช่วย ดังตัวอย่างการใช้งาน

```
architecture test1 of test is
begin
    process (a, b)
        variable temp : std_logic;
    begin
        -- sequential statement;
    end process;
    -- statement;
end test1;
```

จากตัวอย่างการใช้งานจะเห็นว่า Process ที่แทรกเข้าไปใน Architecture นั้นจะทำให้ในส่วน statement ที่ถูกคั่นด้วย begin และ end process นั้นจะมีการทำงานเป็น step ที่ละบรรทัดทันที ซึ่งก็คือรูปแบบของ Sequential Processing ส่วน statement ที่อยู่ใน architecture เดียวกันแต่อยู่นอก block process ก็ยังคงมีการทำงานแบบปกติก็คือ process พร้อมกันทุกบรรทัด นอกจากนี้จากโปรแกรมจะว่าข้างหลัง process ยังมี (a,b) ใส่ไว้ ในวงเล็บนั้นก็คือสัญญาณที่ใช้ใน architecture ตัว architecture จะเข้าไปทำใน block process นี้ก็ต่อเมื่อสัญญาณ a หรือ b มีการเปลี่ยนแปลงเท่านั้น หรือที่เรียกว่า Event ทำให้ในการ simulate นั้นโปรแกรมไม่ต้องเข้าไปทำงานใน process ตลอดเวลาโดยไม่จำเป็น

- Sequential Statements

นอกจาก process statement แล้วก็ยังมี statement ที่จำเป็นอยู่อีกมาก VHDL มี Sequential Statements ให้ใช้งานอยู่ 5 ตัวดังนี้ if, case, loop, assert, wait ให้เราได้เลือกใช้งานในงานต่างๆ มีรายละเอียดและรูปแบบดังต่อไปนี้

รูปแบบ IF statement

```
IF condition THEN
        Sequence Statements
[ELSEIF condition THEN Sequence Statements ]
[ELSE Sequence Statements ]
END IF;
```

if statement ใช้ในการตรวจสอบเงื่อนไขถ้า condition เป็นจริงก็จะทำใน sequential statement ของ then ถ้าไม่ก็จะไปทำใน sequential statement ของ else แทน ยกตัวอย่างการใช้งานง่ายๆ ถ้าเรามีเงื่อนไข ถ้า a = '1' ก็จะส่งค่า c ให้ b นอกจากนี้ก็จะส่งค่า d ให้ b เขียนเป็นโปรแกรมได้ดังนี้

```
if ( a = '1' ) then b <= c; else b <= d; end if;
```

รูปแบบ Case Statements

```
Case expression IS
  WHEN choices [| choices ] => sequence statements
  WHEN choices [| choices ] => sequence statements
  . . .
END CASE;
```

case statement ใช้ในการตรวจตรวจสอบว่า expression (ตัวแปรที่จะนำมาเปรียบเทียบ) นั้นมีค่าเท่ากับตัวเลือกใด โปรแกรมก็จะเข้าไปทำ statement ในตัวเลือกนั้น แตกต่างกับ if statement ตรงที่ case นั้นจะใช้กับกรณีที่เงื่อนไขนั้นเป็นการเปรียบเทียบกับตัวแปรตัวเดียวกัน และมีหลายเงื่อนไข

รูปแบบของ LOOP Statements

VHDL มี loop ให้ใช้งานอยู่ 2 แบบ คือ while และ for การใช้ loop นั้นมีประโยชน์มาก สำหรับข้อมูลที่เรามีหลายชุดเราไม่จำเป็นต้องเขียนโปรแกรมซ้ำๆกัน มีรูปแบบของการใช้งานดังนี้

```
WHILE condition LOOP
  Sequential Statements;
END LOOP
```

```
FOR identifier IN start_value TO finish_value LOOP
  Sequential Statements;
END LOOP;
```

ใน loop statement ยังมี statement อีก 2 ตัวที่ช่วยให้การทำงานของ loop นั้นมีความยืดหยุ่นขึ้นคือ next และ exit statement เมื่อเราเพิ่ม NEXT; เข้าไปใน loop เมื่อโปรแกรมได้ทำงานมาถึง statement ที่มี NEXT อยู่ โปรแกรมก็จะกระโดดกลับไปยังส่วนหัวของ loop ทันทีเพื่อทำงานใน step ถัดมาต่อ สำหรับ exit statement เมื่อเราเพิ่ม EXIT; เข้าไปเมื่อโปรแกรมทำงานมาเจอ EXIT แล้วก็จะทำให้หลุดออกจาก LOOP ทันทีไปทำงานนอก loop ต่อไป

2.2.4 Wait Statements

Wait statement คือ การหน่วงเวลาใน sequential statement ที่จะทำให้โปรแกรมนั้นหยุดทำงานอยู่ใน statement นั้นเป็นเวลาตามที่กำหนดไว้ หรือตามเงื่อนไขที่ตั้งไว้ มีรูปแบบการหน่วงให้ใช้อยู่ 4 แบบคือ wait on signal, wait until condition, wait for time_expression, และ multiple wait condition Statement เหล่านี้เป็น statement ที่มีประโยชน์มากอีก statement หนึ่งมีการทำงานของแต่ละแบบดังนี้

- Wait on signal

เมื่อโปรแกรมทำงานมาถึง statement นี้จะเป็นการหน่วงไว้กับที่ไปจนกว่าจะมีการเปลี่ยนแปลงของ signal ที่กำหนดให้ เช่น Wait on a,b; คือจะรออยู่กับที่จนกว่าค่าของ a หรือ b จะมีการเปลี่ยนแปลง

- Wait until condition

เมื่อโปรแกรมทำงานมาถึง statement นี้จะเป็นการหน่วงไว้กับที่ไปจนกว่าเงื่อนไขนั้นๆจะเป็นจริงก็จะหลุดจากการหน่วง เช่น Wait until ((a='1') and (b='1')); คือจะหน่วงไปเรื่อยๆ จนกว่า
a และ b จะมีค่า '1' ก็จะหลุดจากการหน่วง

- Wait for time_expression

เป็นการหน่วงตามเวลาที่เรากำหนดไว้ เช่น Wait for 10 ns; คือเป็นการหน่วงไว้เป็นเวลา 10 ns;

- Multiple Wait condition

เป็นการใช้การหน่วงทั้ง 3 แบบมาร่วมกันเช่นดังตัวอย่าง

Wait on a, b until c='1' for 2 usec;

คือจะหน่วงไว้จนกว่า a หรือ b จะมีการเปลี่ยนแปลงค่า และ c จะต้องเป็น 1 ถ้าไม่มีเงื่อนไขแบบนี้เกิดขึ้นก็จะหน่วงไว้นานสุดที่ 2 usec โดยการใช้ Multiple Wait นี้มีเงื่อนไขการใช้อยู่ด้วยคือสามารถจะนำมา wait แบบต่างๆมาใช้ร่วมกันได้ 4 แบบ โดยเวลาใช้ก็ต้องมีเรียงลำดับด้วยดังนี้ 1) on until for 2) on until 3) on for 4) until for

2.2.5 ชนิดของข้อมูล (Data types)

- Object Types

VHDL ก็มีคุณสมบัติต่างๆเช่นเดียวกับภาษาระดับสูง มีการแบ่งชนิดของตัวแปรให้ใช้ในภาษาอยู่ด้วย 3 ชนิดด้วยกันคือ

1 Signal ใช้เป็นสัญญาณเพื่อแสดงการเชื่อมต่อกันระหว่าง Component ต่างๆ

2 Variable ใช้เป็น Temp เก็บข้อมูลชั่วคราวภายใน entity โดยใช้ในการ process

3 Constant ชื่อที่มีการกำหนดค่าคงที่ให้ ไว้ใช้ใน program

signal มีรูปแบบการใช้งานดังนี้

```
SIGNAL signal_name : signal_type [:= initial_value];
```

การใช้งาน Signal นอกจากจะเป็นการประกาศชนิดของสัญญาณแล้ว ยังสามารถจะกำหนดค่าเริ่มต้นให้กับสัญญาณนั้นได้ด้วยดังตัวอย่างเป็นการประกาศสัญญาณ vcc ให้มีชนิดเป็น std_logic และมีค่าเริ่มต้นเป็น 1

```
SIGNAL vcc : std_logic := '1';
```

(* std_logic คือ ชนิดของตัวแปรอีกชนิดหนึ่งมีด้วยกันทั้งหมด 9 ค่าคือ U, X, 0, 1, Z, W, L, H, -)

Variables มีรูปแบบการใช้งานดังนี้

```
VARIABLE variable_name [,variable_name] : variable_type [:= value];
```

การใช้งาน Variables ก็มีลักษณะคล้ายกับ signal คือสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรได้ด้วย ดังตัวอย่าง กำหนดให้ delay เป็น time มีค่าเริ่มต้นที่ 2 ns

```
VARIABLE delay : time := 2 ns;
```

Constants มีรูปแบบการใช้งานดังนี้

```
CONSTANT constant_name [,constant_name] : type_name [:= value];
```

การใช้งาน Constant คือจะต้องมีการกำหนดค่าเริ่มต้นให้กับตัวแปรเพื่อเป็นค่าคงที่นำไปใช้ในโปรแกรมได้ ดังตัวอย่างเป็นการกำหนดตัวแปร pi มีชนิดเป็น real มีค่าคงที่เท่ากับ 3.1414

```
CONSTANT pi : real := 3.1414;
```

- ชนิดของข้อมูล (Data types)

ชนิดของตัวแปรที่เราใช้ประกาศใน Signal, Variable, หรือ Constant แบ่งออกมาได้เป็น 4 ประเภทใหญ่ๆ คือ Scalar, Composite, Access และ File Types

Scalar Types นั้นเป็นตัวแปรแบบพื้นฐานที่ใช้กันบ่อยเช่น integer, real เป็นต้น Composite เป็นชนิดที่ใช้สร้าง array และ record ส่วน Access types เป็นตัวแปรรูปแบบของ Pointer เหมือนกับภาษาโปรแกรมทั่วไป File เป็นตัวแปรที่เกี่ยวกับไฟล์

1. Scalar Types

Scalar Types แบ่งย่อยออกเป็น 4 ชนิดคือ Integer, Real, Enumerated, และ Physical types การใช้งานตัวแปรเหล่านี้จะต้องใส่ค่าให้ถูกต้องต้องใส่ค่าให้ถูกชนิดด้วยเช่น Integer ห้ามเอาค่า Real ใส่ให้เป็นต้น

Integer อ้างถึงข้อมูลตัวเลขจำนวนเต็มในช่วง -2,147,483,647 ถึง +2,147,483,647

Real อ้างถึงข้อมูลตัวเลขจำนวนจริงที่เป็นทศนิยมในช่วง -1.0E+38 ถึง +1.0E+38

Enumerated เป็นชนิดที่มีประสิทธิภาพมากอีกชนิดหนึ่ง เพราะเป็นชนิดที่มีการกำหนดขอบเขต หรือว่า set ของกลุ่มข้อมูลที่จะอ้างถึงขึ้นมาได้เองโดย Programmer ยกตัวอย่างถ้าเราต้องการสร้าง type ข้อมูลที่อ้างถึงข้อมูลได้ 3 ตัวคือ red, green และ yellow เพื่อใช้กับโปรแกรมไฟจราจรเพื่อทำให้มองโปรแกรมได้ง่าย

```
TYPE  t_state IS ( red, green, yellow );
```

Physical เป็นการกำหนดตัวแปรที่ประกอบด้วย 2 ส่วน ส่วนแรกกำหนดขอบเขตของตัวแปร ส่วนที่สองกำหนดหน่วยขึ้นมาใช้งาน ดังตัวอย่างจะเป็นการกำหนดชนิดข้อมูลที่ชื่อว่า current มีขอบเขตที่ 0 ถึง 1000000000 และมีหน่วยให้ใช้งานอยู่ 3 หน่วยคือ na, ua และ ma ตัวแปรแบบนี้เหมาะที่จะใช้กับข้อมูลที่มีความละเอียดสูง มีตัวเลขอยู่หลายหลัก เมื่อมาใช้หน่วยช่วยจะทำให้อ้างถึงข้อมูลเหล่านั้นได้ง่าย และ สั้นกว่า

```
Type  current is range 0 to 1000000000
  units
        na;
        ua = 1000 na;
        ma = 1000 ua;
end units;
```

2. Composite Types

เป็นชนิดที่มีการเก็บข้อมูลที่มีลักษณะเป็นกลุ่มจึงแยกย่อยได้เป็น 2 ชนิดคือ array และ record สำหรับข้อมูลชนิด array แล้วช่วยได้มากในการทำ ROM หรือ RAM

Array รูปแบบของ array มิติเดียว

```
TYPE  name  IS Array ( start  TO stop ) OF  ชนิดของตัวแปร
```

ตัวอย่างการประกาศ Array มิติเดียว data_bus เป็น type ที่มีขนาด 4 bit มีชนิด signal เป็น bit

```
TYPE data_bus IS Array ( 0 TO 3 ) OF bit;
```

ตัวอย่างการประกาศตัวแปรให้ x มีชนิดเป็น data_bus

VARIABLE x : data_bus;

ตัวอย่างการเข้าถึง Array แบบทีละตำแหน่ง x(1) := '1';

ตัวอย่างการเข้าถึง Array แบบเป็นกลุ่ม x := ('1', '0', '1', '0');

Multidimensional Arrays เป็นการใช้ array หลายมิติ Array 2 มิตินิยมใช้ในการออกแบบ rom และ ram

ตัวอย่างการประกาศ Array หลายมิติ

TYPE mem_data IS Array (0 TO 32, 0 TO 32) OF std_logic;

ตัวอย่างการเข้าถึง Array หลายมิติ X(1,1) := '1';

Record

Record คือการจะมองกลุ่มของข้อมูลชนิดต่างๆ เข้าด้วยกันเป็น Object เดียว ดังนั้น Record จะประกอบไปด้วยข้อมูลชนิดต่างๆอยู่ เราสามารถที่จะใช้ Record ซ้อน Record ได้เหมือนกับ Programming Language ทั่วๆไป

ตัวอย่างการประกาศ

```
type test_record is
    record
        d : integer;
        o : real;
    end record;
variable test : test_record;
```

เราสามารถอ้างถึงข้อมูลได้โดยใส่จุดไว้ข้างหลังตัวแปร และตามด้วยชื่อฟิลด์ใน record เช่น

```
test.d := 5;
test.o := 10.5;
```

3. Access Types

Access มีรูปแบบเหมือนกับ pointer ในภาษาปาสคาลสามารถนำ Pointer นั้นไป Link เชื่อมต่อกันเป็น Link List ได้ด้วย มี Function ที่ใช้กับ Access Types นี้อยู่ 2 ตัวคือ 1. NEW ใช้ในการ Allocate ตำแหน่งใน Memory 2. DEALLOCATE ใช้ในการยกเลิกการใช้งานของ Pointer

ตัวอย่างการใช้

```
TYPE fifo IS ARRAY (0 TO 3) OF std_logic;
TYPE fifo_access IS ACCESS fifo;
```

```
VARIABLE  fifo_ptr : fifo_access := NULL;
fifo_ptr := new fifo;
fifo_ptr.ALL := ('0','0','1','1');
```

4. File Types

เป็นข้อมูลที่ประกาศขึ้นเพื่อใช้ในการติดต่อกับไฟล์

รูปแบบการประกาศ

```
TYPE file_ name IS FILE OF file_ type;
```

file มี Function ให้ใช้งานอยู่ 3 Function

READ (file,data) (file คือชื่อ fileที่จะอ่านข้อมูล ; data คือตัวแปรที่จะรับค่าในไฟล์กลับ)

WRITE (file,data) (file คือชื่อ fileที่จะอ่านข้อมูล ; data คือตัวแปรที่เก็บค่าที่จะเขียนลงไปในไฟล์)

ENDFILE (file) (file คือชื่อ fileที่ปิด)

5. Subtypes

เป็นการนำชนิดของข้อมูลที่มีอยู่ดึงมาใช้เพียงบางส่วน มีข้อดีคือ ไม่ต้องกำหนด type ขึ้นมาใหม่, ใช้ขอบเขตของข้อมูลเท่าที่จำเป็น, เข้าใจขอบเขตการใช้งานได้ง่าย เช่นชนิดจำนวนเต็มบวก natural จาก type integer

```
SUBTYPE  natural  IS  integer  RANGE 0 to +2,147,483,647;
```

2.2.6 เทคนิคการใช้งาน VHDL

- การใช้งาน signal ชนิด std_logic

Std_logic เป็นสัญญาณอีกชนิดหนึ่งที่นิยมใช้ซึ่งมีอยู่ด้วยกันทั้งหมด 9 ค่าคือ (U, X, 0, 1, Z, W, L, H, -) ดังนั้นในการใช้งานเราต้องคำนึงอยู่เสมอว่าไม่ได้มีเพียงค่า 1 กับ 0 ดังนั้นในโปรแกรมของเรานั้นจะต้องมีการใช้ other อยู่เสมอ (others เป็น keyword ตัวหนึ่งที่จะให้ค่าที่เหลือที่เราไม่ได้อ้างถึง) ดังตัวอย่าง กำหนดให้ s เป็น array ขนาด 2 bit ชนิด std_logic

```
case  s  is
    when "00" => muxout <= c;
    when "01" => muxout <= d;
    when "10" => muxout <= e;
```

```
when others   => muxout <= f;
end case;
```

จากโปรแกรมถ้าค่า s ไม่ได้เท่ากับ “00” หรือ “01” หรือ “10” แล้ว โปรแกรมจะส่งค่า f ให้ muxout ทันที

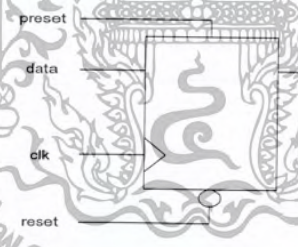
นอกจากการนำ others มาช่วยแล้ว เรายังสามารถใช้ don’t care เข้ามาช่วยได้อีกด้วยในกรณีที่เราสนใจข้อมูลเฉพาะบางบิท เช่น ถ้าเรามีข้อมูลอยู่ 6 บิท แต่ต้องการตรวจสอบเฉพาะ 2 บิท หลังว่าเป็น 1 ทั้งคู่หรือไม่ ก็จะต้องนำข้อมูลนั้นไปเปรียบเทียบกับ “----11” เครื่องหมายลบแทน don’t care การใช้งานอย่างนี้จะมีประโยชน์มากในการเปรียบเทียบเพราะเราไม่จำเป็นต้องแยกออกมาเปรียบเทียบกันทีละบิท

2.2.7 การใช้งาน VHDL กับ Flip-Flops

Flip-Flop จะทำงานได้นั้นจะต้องใช้ clk เข้ามาช่วยซึ่งการทำงานของ flip-flop นั้นอาศัยการทำงานของขอบขาขึ้น หรือ ขอบขาลงของ clk สามารถประยุกต์ใช้ได้ดังนี้

```
( clk'event   and   clk = '1')  แทนขอบขาขึ้นของ clk
( clk'event   and   clk = '0')  แทนขอบขาลงของ clk
```

ยกตัวอย่างการออกแบบ D Flip-Flop ที่มี preset และ reset



รูปที่ 2.4. แสดง D Flip-Flop มี preset และ reset

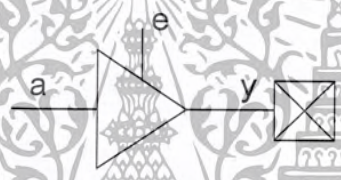
```
entity  dff_async  is     ( entity block )
  port (data, clk, reset, preset : in std_logic;
     q : out std_logic);
  end dff_async;

  architecture behav of dff_async is
  begin
  process (clk, reset, preset ) begin
```

```
if (reset ='0') begin            ( reset ค่า q เมื่อ reset เป็น '0' )
    q <= '0';
elsif (preset ='1') then                ( set ค่า q เมื่อ preset เป็น '1' )
    q <= '1';
elsif(clk'event and clk='1') then  (ส่ง data เมื่อมี clk เข้ามา )
    q <= data;
  end if;
end process;
end behav;
```

2.2.8 การออกแบบ buffer แบบต่างๆ

1. Tri-State



รูปที่ 2.5 แสดง Tri-State

การออกแบบ buffer ที่มี 3 สถานะ โดยมี enable เป็นตัวควบคุมสามารถเขียนเป็น entity ได้ดังนี้

```
entity  tristate  is
port  (e, a : in std_logic;
       y : out std_logic);
end tristate;
```

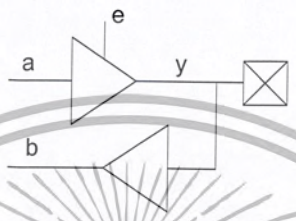
Architecture นั้นสามารถเขียนได้หลายแบบดังตัวอย่างทั้ง 2 แบบต่อไปนี้

```
architecture tri of tristate is
 begin
  process (e, a)
    begin
      if  e = '1' then
       y <= a;
```

```
architecture tri of tristate is
 begin
   y<=a when(e= '1') else 'z'
end tri;
```

```
    else
     y<= 'z';
    end if;
  end process;
end tri;
```

2. Bi-Directional Buffer



รูปที่ **2.6** แสดง Bi-Directional Buffer

เป็น buffer 2 ทิศทางสามารถเขียนเป็นโปรแกรมส่วน Architecture ได้ดังนี้

```
architecture bi of bidir is
 begin
  process (e, a)
   begin
     case e is
       when '1' => y <= a;
       when '0' => y <= 'z';
       when  other => y <= 'x';
     end case;
  end process;
  b <= y;
end bi;
```

2.2.9 ข้อดีของ VHDL

1. รูปแบบของภาษาที่เข้าใจได้ง่าย
2. มีโครงสร้างของภาษาที่สามารถปรับเปลี่ยนให้เข้ากับ hardware ได้ง่าย
3. มี statement ให้ใช้งานอยู่หลายตัว

4. สามารถออกแบบในรูปแบบ top-down design ได้

5. มี library ต่างๆให้เลือกใช้งานได้ทำให้ออกแบบได้ง่ายขึ้น เป็นประโยชน์ต่อ programmer

2.2.10 VHDL ช่วยในการออกแบบได้อย่างไรบ้าง

1. การออกแบบวงจรสามารถทำได้โดยง่าย

2. เมื่อมีการแก้ไข หรือ เปลี่ยนแปลงวงจรสามารถทำได้โดยง่าย เพียงโปรแกรมวงจรใหม่ลงไป ไม่ต้องมีการเปลี่ยนแปลงตัว hardware

3. การทดสอบไม่จำเป็นต้องทดลองกับวงจรจริง สามารถใช้การ simulate ทดลองดูผลลัพธ์ของวงจรที่เราออกแบบได้ว่าถูกต้องหรือไม่

2.3 หลักการของมอเตอร์ (Motor principle)

เมื่อมีกระแสไฟฟ้าไหลในตัวนำซึ่งอยู่ในสนามแม่เหล็กนั้น จะทำให้เกิดแรงขึ้นในตัวนำ

รูปที่ 2.7 แสดงขดลวดที่มีกระแสไฟฟ้าไหลและวางอยู่ในสนามแม่เหล็ก

แรงที่เกิดขึ้นนี้จะอยู่ในแนวที่ตั้งฉากกับเส้นแรงแม่เหล็กและกระแสที่ไหลผ่านในตัวนำนั้นๆ ดังนั้นเมื่อมีกระแสไหลในขดลวดตัวนำที่พันอยู่บนแกนเหล็กอาร์เมเจอร์ จะเกิดเส้นแรงแม่เหล็กขึ้นรอบๆ ตัวนำ และจะเกิดการทำปฏิกิริยากับเส้นแรงแม่เหล็กที่เกิดจากขั้วแม่เหล็กของมอเตอร์ ทำให้เกิดแรงผลักขึ้นบนตัวนำ จึงทำให้อาร์เมเจอร์หมุนไปได้

จากรูปที่ 2.7 แสดงขดลวดที่มีกระแสไฟฟ้าไหล และวางอยู่ในสนามแม่เหล็ก โดยตัวนำวางห่างจากจุดศูนย์กลางเป็นระยะ r และให้กระแสไฟฟ้าไหลเข้าขวดลวดที่ปลาย A และไหลออกที่ปลาย B จากคุณสมบัติของเส้นแรงแม่เหล็กที่ว่า เส้นแรงแม่เหล็กจะไม่ตัดผ่านซึ่งกันและกัน ดังนั้น ปริมาณของเส้นแรงแม่เหล็กจะมีจำนวนมากที่ด้านบนของปลาย A จึงทำให้เกิดแรงกดตัวนำ A ลงด้านล่าง และขณะเดียวกันที่ปลาย B นั้น เส้นแรงแม่เหล็กจะมีปริมาณมากที่ด้านล่าง ทำให้เกิดแรงดันตัวนำ B

เคลื่อนที่ขึ้นด้านบน ผลของแรงที่กระทำกับตัวนำ A และ B นี้ จะทำให้อาร์เมเจอร์ของมอเตอร์เกิดการเคลื่อนที่ไปได้

การกระทำของแรงที่เกิดขึ้นกับตัวนำที่มีกระแสไหลผ่านในขณะที่วางอยู่ในสนามแม่เหล็ก จะเป็นปฏิภาคโดยตรงกับความหนาแน่นของเส้นแรงแม่เหล็ก ความยาวของตัวนำ และค่าของกระแสไฟฟ้าที่ไหลผ่านตัวนำ ซึ่งสามารถเขียนเป็นสมการได้ดังนี้

$$F = Bli \qquad (2.1)$$

เมื่อ F คือ แรงที่เกิดขึ้นบนตัวนำ (N)
B คือ ความหนาแน่นของเส้นแรงแม่เหล็ก (Wb/m^2)
l คือ ความยาวของลวดตัวนำ (m)
i คือ กระแสที่ไหลในตัวนำ (A)

2.3.1 แรงดันไฟฟ้าต้านกลับ (Back e.m.f. or Counter e.m.f.)

เมื่อมอเตอร์ทำงาน คือ อาร์เมเจอร์หมุน ตัวนำที่อาร์เมเจอร์จะตัดกับเส้นแรงแม่เหล็ก ซึ่งเป็นไปตามกฎของการเหนี่ยวนำแม่เหล็กไฟฟ้า (Electromagnetic induction) ดังนั้น แรงดันไฟฟ้าเหนี่ยวนำจึง

รูปที่ 2.8 แสดงทิศทางของแรงดันไฟฟ้าต้านกลับ

เกิดขึ้นในอาร์เมเจอร์ ทิศทางจะเป็นไปตามกฎมือขวาของเฟลมมิ่ง ซึ่งจะมีทิศทางไปในทางตรงกันข้าม กับแรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์ (Applied voltage) เพราะว่ามีทิศทางตรงกันข้ามกันจึงเรียกว่า แรงดันไฟฟ้าต้านกลับ (Back e.m.f. หรือ Counter e.m.f.) ใช้ตัวย่อ E_bหรือ E_cและวงจรเทียบเคียงของมอเตอร์ไฟฟ้ากระแสตรงสามารถเขียนได้ดังแสดงในรูปที่ 2.8

เมื่ออาร์เมเจอร์หมุนจะเกิดแรงดันไฟฟ้าต้านกลับออกมาซึ่งเปรียบเสมือนกับว่ามีแบตเตอรี่อยู่ภายในและจะเห็นได้ว่า

$$I_a = (V - E_b) / R_a \quad A$$

เมื่อ R_a คือ ความต้านทานในวงจรของอาร์เมเจอร์ (Ω)

แต่

$$E_b = (\Phi ZNP) / (60A) \quad V$$

เมื่อ Φ คือ เส้นแรงแม่เหล็กต่อขั้ว (Wb)

Z คือ จำนวนตัวนำในอาร์เมเจอร์ = จำนวนสล็อท × จำนวนตัวนำต่อสล็อท

P คือ จำนวนขั้วแม่เหล็ก

N คือ ความเร็วรอบของอาร์เมเจอร์ (rpm.)

A คือ จำนวนทางผ่านในอาร์เมเจอร์

E_b คือ แรงดันไฟฟ้าเหนี่ยวนำที่เกิดขึ้นในอาร์เมเจอร์ (V)

แรงดันไฟฟ้าต้านกลับนั้นขึ้นกับองค์ประกอบหลายอย่าง เช่นขึ้นอยู่กับความเร็วรอบของอาร์เมเจอร์ ถ้าความเร็วรอบสูงแรงดันไฟฟ้าต้านกลับก็จะมาก ดังนั้นกระแสที่ไหลในอาร์เมเจอร์ (I_a) จากสมการข้างบนนั้นจะมีค่าน้อย ถ้าความเร็วรอบต่ำแรงดันไฟฟ้าต้านกลับก็จะน้อย I_aก็จะมาก ดังนั้น แรงบิด (Torque) ที่เกิดขึ้นก็จะสูงกว่าเมื่อมีความเร็วรอบสูง

2.3.2 สมการของแรงดันไฟฟ้าต้านกลับในมอเตอร์

เมื่อ V คือ แรงดันไฟฟ้าที่จ่ายให้กับวงจรของมอเตอร์

E_b คือ แรงดันไฟฟ้าต้านกลับที่เกิดขึ้นในอาร์เมเจอร์

ซึ่ง

$$V = E_b + I_aR_a \quad (2.2)$$

เมื่อ I_aR_a คือ แรงดันไฟฟ้าตกคร่อมที่ความต้านทานของอาร์เมเจอร์

จากสมการที่ 2.2 ถ้าคูณด้วย I_a ตลอด จะได้

$$VI_a = E_bI_a + (I_a)^2 R_a \quad (2.3)$$

จะเห็นว่า VI_a คือ กำลังไฟฟ้าที่จ่ายให้กับอาร์เมเจอร์

E_bI_a คือ กำลังไฟฟ้าที่เกิดขึ้นในอาร์เมเจอร์ ซึ่งเทียบเท่ากับกำลังทางกลที่เกิดขึ้น

$(I_a)^2 R_a$ คือ กำลังสูญเสียในขดลวดทองแดงในอาร์เมเจอร์

2.3.3 สภาวะสำหรับการเกิดประสิทธิภาพสูงสุด (Condition for maximum Power)

กำลังทางกลที่เกิดขึ้นในมอเตอร์ คือ

$$P_m = VI_a - (I_a)^2R_a \qquad (2.4)$$

ดิฟเฟอร์เรนเชียลทั้งสองข้างเปรียบเทียบกับ I_a ซึ่งให้ทั้งหมด = 0 จะได้

$$dP_m / dI_a = V - 2I_aR_a = 0$$

ดังนั้น $V - 2I_aR_a = 0$

$$V = 2I_aR_a$$

นั่นคือ $I_aR_a = V/2$

แต่ $V = E_b + I_aR_a$

ดังนั้น $E_b = V / 2$

กำลังทางกลที่เกิดขึ้นในมอเตอร์จะมีค่าสูงสุดเมื่อแรงดันไฟฟ้าต้านกลับเท่ากับครึ่งหนึ่งของแรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์

2.3.4 แรงบิด (Torque)

จาก $T = F \times r$ N-m

งานที่ทำได้จากแรงใน 1 รอบของการหมุน = แรง × ระยะทาง

$= F \times 2\pi r$ จูล

กำลังที่เกิดขึ้น = $F \times 2\pi r \times N / 60$ จูล/วินาที

= $(F \times r) \times 2\pi N / 60$ จูล/วินาที

เมื่อ $2\pi N / 60$ = ความเร็วเชิงมุม คือ ω มีหน่วยเป็น เรเดียน/วินาที

และ $F \times r$ = แรงบิด (T)

ดังนั้น กำลังที่เกิดขึ้น = $T \times \omega$ จูล/วินาที หรือ วัตต์ (W)

แรงบิดในอาร์เมเจอร์ (Torque in armature, T_a)

แรงบิดที่เกิดขึ้นในอาร์เมเจอร์ขณะที่มอเตอร์กำลังหมุนอยู่ที่ N/60 รอบ/วินาที

ถ้า T_a เป็นนิวตัน – เมตร

ดังนั้น กำลังที่เกิดขึ้นจะมีค่า = $T_a \times 2\pi N / 60$ W (2.5)

แต่จากกำลังทางกลที่เกิดขึ้นในอาร์เมเจอร์ มีค่า = $E_b I_a$ W (2.6)

จากสมการที่ (2.5) และ (2.6) จะเห็นว่า

$T_a \times 2\pi N / 60 = E_b I_a$ (2.7)

เมื่อ $E_b = (\Phi ZN/60)(P/A)$ V

เพราะฉะนั้น $T_a \times 2\pi N / 60 = (\Phi ZN/60)(P/A) \times I_a$

หรือ $T_a = (1/2\pi)\, \Phi ZI_a(P/A)$ N-m

ดังนั้น $T_a = 0.159\, \Phi ZI_a(P/A)$ N-m

$= 0.0162\, \Phi ZI_a(P/A)$ kg-m

1 กิโลกรัม – น้ำหนัก (kg-wt) = 9.81 นิวตัน (N)

หมายเหตุ จากสมการข้างบนนี้จะเห็นว่า $T_a \sim \Phi I_a$

- ในกรณีของซีรีส์มอเตอร์ $\Phi \sim I_a$ ดังนั้น $T_a \sim (I_a)^2$
- ในกรณีของชั้นท์มอเตอร์ Φ ในทางปฏิบัติมีค่าคงที่ ดังนั้น $T_a \sim I_a$

จากสมการที่ (2.7) จะเห็นว่า

$$T_a = (1/2\pi)(E_b I_a)/(N/60) \quad \text{N-m}$$

$$= 0.159\, E_b I_a/(N/60) \quad \text{N-m}$$

ดังนั้น $T_a = 0.0162\, E_b I_a/(N/60)$ kg-m

แรงบิดที่เพลาของมอเตอร์ (Shaft torque, T_{sh})

แรงบิดที่อาร์เมเจอร์สามารถคำนวณหาได้ โดยใช้สมการที่ (2.7) คือ $T_a = (E_b I_a \times 60) / 2\pi N$ แต่ไม่สามารถนำมาใช้งานได้ เพราะว่ามีการสูญเสียที่แกนเหล็ก และการสูญเสียเนื่องจากความฝืดในมอเตอร์ด้วยแรงบิดที่นำมาใช้งานหรือใช้ประโยชน์คือแรงบิดที่เพลา (T_{sh}) เพราะว่าใช้ประโยชน์ ที่เพลา แรงม้า (Horse-power) ที่เกิดขึ้น นั้นคิดที่แรงบิดที่เพลา เรียกว่า ขนาดกำลังม้า (Brake horse power : B.H.P.) เพราะว่าขนาดกำลังม้าคิดที่ตำแหน่งเบรค

ขนาดกำลังม้า (B.H.P.) $= (T_{sh} \times 2\pi N/60) / 746$ กำลังม้า

แต่เพราะว่า $T = 746 \times \text{B.H.P.} \times (2\pi N/60)$ (N-m)

$=$ กำลังเอาต์พุต $/ (2\pi N/60)$ (N-m)

ความแตกต่างระหว่าง $T_a - T_{sh}$ เป็นแรงบิดที่สูญเสีย (Losses torque) มีหน่วยเป็น นิวตัน–เมตร (Newton – metre, N–m)

ดังนั้น แรงบิดที่สูญเสีย $=$ 0.159 $\times$ กำลังสูญเสียที่แกนเหล็กและกำลังสูญเสีย เนื่องจากความฝืด เป็น นิวตัน – เมตร (N–m)

$=$ 0.0162 $\times$ กำลังสูญเสียที่แกนเหล็กและกำลังสูญเสีย เนื่องจากความฝืด เป็น กิโลกรัม – เมตร (kg–m)

2.3.5 ความเร็วรอบของมอเตอร์ไฟฟ้ากระแสตรง (Speed of d.c. motor)

จากสมการของแรงดันไฟฟ้า หรือ Back e.m.f. ของมอเตอร์

$$E_b = V - I_aR_a$$

หรือ $(\Phi ZN/60)(P/A) = V - I_aR_a$

เพราะฉะนั้น $N = [(V - I_aR_a) / \Phi](60A/ZP)$ รอบต่อนาที (rpm.)

เมื่อ $E_b = V - I_aR_a$ โวลต์ (V)

เพราะฉะนั้น $N = (E_b/\Phi)(60A/ZP)$ รอบต่อนาที (rpm.)

เมื่อ $K = 60A/ZP$

เพราะฉะนั้น $N = K(E_b/\Phi)$ รอบต่อนาที (rpm.)

ดังนั้น จะเห็นได้ว่าความเร็วของมอเตอร์นั้นเป็นปฏิภาคโดยตรงกับแรงดันไฟฟ้าต้านกลับ และเป็นปฏิภาคส่วนกลับกับเส้นแรงแม่เหล็ก (Φ)

สำหรับซีรีส์มอเตอร์

ถ้าให้ N_1 คือ ความเร็วรอบในกรณีที่ 1

I_{a1} คือ กระแสในอาร์เมเจอร์กรณีที่ 1

Φ_1 คือ เส้นแรงแม่เหล็กต่อขั้วกรณีที่ 1

และ N_2, I_{a2}, Φ_2 คือ ค่าต่างๆ ในข้างบนนี้ แต่เป็นกรณีที่ 2

จากสมการความสัมพันธ์ข้างบนจะได้ว่า

$N_1 \sim E_{b1}/\Phi_1$ เมื่อ $E_{b1} = V - I_{a1}R_a$

$N_2 \sim E_{b2}/\Phi_2$ เมื่อ $E_{b2} = V - I_{a2}R_a$

จะได้ $N_2/N_1 = (E_{b2}/E_{b1})(\Phi_1/\Phi_2)$

ในขณะที่เส้นแรงแม่เหล็กยังไม่ถึงจุดอิ่มตัวจะได้ $\Phi \sim I_a$

ดังนั้น $N_2/N_1 = (E_{b2}/E_{b1})(I_{a1}/I_{a2})$

สำหรับชั้นท์มอเตอร์

ในกรณีเช่นนี้จะได้สมการเช่นเดียวกันกับของซีรีส์มอเตอร์ คือ

$$N_2/N_1 = (E_{b2}/E_{b1})(\Phi_1/\Phi_2)$$

ถ้า $\Phi_1 = \Phi_2$

ดังนั้น $N_2/N_1 = E_{b2}/E_{b1}$

เปอร์เซ็นต์การเปลี่ยนแปลงความเร็ว (Percent speed regulation) สามารถหาได้จากสมการ คือ ความแตกต่างระหว่างความเร็วรอบของมอเตอร์ในขณะที่ไม่มีโหลด กับในขณะที่มีโหลดนั้น เรียกว่า Speed regulation

ความเร็วที่เปลี่ยนแปลง = ความเร็วขณะไม่มีโหลด – ความเร็วขณะที่มีโหลดเต็มที่

ความเร็วที่เปลี่ยนแปลงนี้สามารถคิดเป็นเปอร์เซ็นต์ได้ โดยการเปรียบเทียบกับความเร็วขณะที่มีโหลดเต็มที่ คือ

$$\% \text{ Speed Ragulation} = [(\text{N.L. speed} - \text{F.L. speed})/\text{F.L. speed}] \times 100$$

เมื่อ % Speed Ragulation คือ เปอร์เซ็นต์การเปลี่ยนแปลงความเร็ว

N.L. speed คือ ความเร็วขณะที่ไม่มีโหลด

F.L. speed คือ ความเร็วขณะที่มีโหลดเต็มที่

2.3.6 คุณลักษณะของมอเตอร์ไฟฟ้ากระแสตรง (Characteristics of d.c. motor)

เส้นโค้งคุณลักษณะ (Characteristic curve) ของมอเตอร์นั้น เป็นเส้นโค้งแสดงความสัมพันธ์ระหว่างปริมาตรต่าง ๆ ดังต่อไปนี้

1. แรงบิดและกระแสในอาร์เมเจอร์ (T_g/I_a characteristic) เรียกว่า คุณลักษณะทางไฟฟ้า (Electrical Characteristic)

2. ความเร็วและกระแสในอาร์เมเจอร์ (N/I_a characteristic)

3. ความเร็วและแรงบิด (N/T_a characteristic) เราเรียกว่า คุณลักษณะทางกล ซึ่งสามารถหาได้จากคุณลักษณะของ T_a/I_a และ N/I_a คุณลักษณะของมอเตอร์ มีความสัมพันธ์ซึ่งกันและกัน ดังนี้คือ

$$T_a \sim \Phi I_a$$

และ

$$N \sim E_b/\Phi$$

คุณลักษณะของซีรีส์มอเตอร์ (Characteristics of series motor) มีดังนี้

1. คุณลักษณะของแรงบิดกับกระแสในอาร์เมเจอร์ (T_a/I_a characteristic) จะเห็นว่า $T_a \sim \Phi I_a$ ในกรณีเชนนี้ฟิลด์คอยล์มีกระแสไหลผ่านซึ่งเป็นกระแสไฟฟ้าที่ไหลผ่านอาร์เมเจอร์ และ $\Phi \sim I_a$ จนกระทั่งถึงจุดที่เส้นแรงแม่เหล็กอิ่มตัว (Magnetic saturation) ดังนั้นก่อนที่จะถึงจุดที่เส้นแรงแม่เหล็กอิ่มตัว จะได้

$$T_a \sim \Phi I_a$$

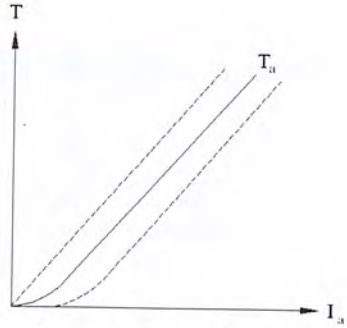
เพราะฉะนั้น

$$T_a \sim (I_a)^2$$

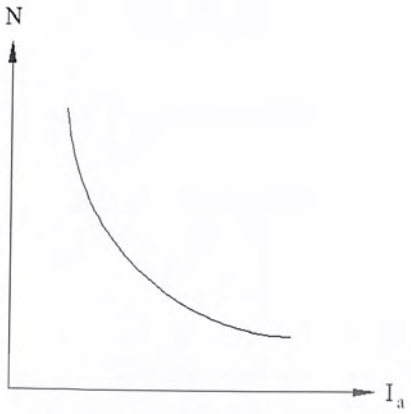
เมื่อโหลดน้อยๆ ค่าของ I_a และ Φ ก็จะมีค่าน้อย แต่ถ้า I_a เพิ่มขึ้น T_a ก็จะเพิ่มขึ้น เท่ากับกระแส $(I_a)^2$ ดังนั้น เส้นโค้ง T_a/I_a จึงเป็นพาราโบลา (parabola) ดังแสดงในรูปที่ 2.3

แต่หลังจากที่ถึงจุดที่เส้นแรงแม่เหล็กอิ่มตัวแล้ว Φ เกือบไม่ขึ้นอยู่กับค่า I_a ดังนั้น $T_a \sim I_a$ เพียงอย่างเดียว คุณลักษณะจึงเป็นเส้นตรง (Straight line)

แรงบิดที่เพลา (T_{sh}) จะน้อยกว่าแรงบิดที่อาร์เมเจอร์ (T_a) ซึ่งขึ้นอยู่กับการสูญเสียปลีกย่อย ดังแสดงดังรูปที่ 2.9 ด้วยเส้นประ



รูปที่ 2.9 แสดงความสัมพันธ์ระหว่างแรงบิดกับกระแสในอาร์เมเจอร์



รูปที่ 2.10 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับกระแสในอาร์เมเจอร์

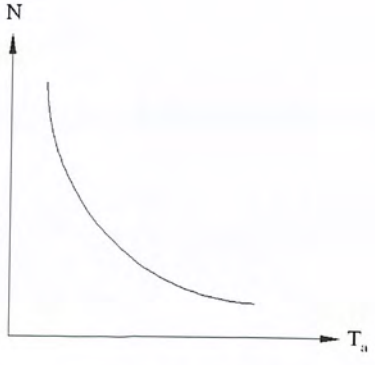
2. คุณลักษณะของความเร็วกับกระแสในอาร์เมเจอร์ (N/I_a characteristic) การเปลี่ยนแปลงของความเร็ว สามารถหาได้จากสมการดังนี้คือ

$$N \sim E_b/\Phi$$

เปลี่ยนแปลง E_b ได้โดยการเปลี่ยนแปลงโหลด จะทำให้กระแสในอาร์เมเจอร์เกิดการเปลี่ยนแปลง และเมื่อ I_a เพิ่มขึ้น Φ ก็จะเพิ่มขึ้นด้วย ดังนั้น ความเร็วจึงแปรผกผัน กับกระแสที่ไหลในอาร์เมเจอร์ ดังแสดงในรูปที่ 2.10

เมื่อมอเตอร์มีโหลดมากๆ I_a ก็จะมีปริมาณมากด้วย ดังนั้น ความเร็วรอบของมอเตอร์จึงต่ำ แต่เมื่อโหลดน้อย I_a ก็จะมีค่าลดน้อยลงความเร็วรอบของมอเตอร์จะสูงขึ้นมาก และอาจเป็นอันตรายแก่มอเตอร์ได้ ดังนั้น ซีรีส์มอเตอร์จึงไม่สามารถเริ่มเดินเมื่อปราศจากโหลดได้ เส้นโค้งคุณลักษณะแสดงได้ดังรูปที่ 2.10

3. คุณลักษณะของความเร็วกับแรงบิดหรือคุณลักษณะทางกล (Mechanical characteristic) ซึ่ง



รูปที่ 2.11 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิด

สามารถหาได้ดังที่กล่าวมาแล้วในตอนต้น ความเร็วสูงแรงบิดจะมีค่าต่ำ ความสัมพันธ์ระหว่างความเร็วรอบ และแรงบิด แสดงได้ดังในรูปที่ 2.11

คุณลักษณะของชั้นท์มอเตอร์ (Characteristics of shunt motor) มีดังนี้

1. คุณลักษณะของแรงบิดกับกระแสในอาร์เมเจอร์ (T_a/I_a characteristic) สมมติให้ Φ ในทางปฏิบัติมีค่าคงที่ (เมื่อมีโหลดมากๆ Φ จะลดลง เพราะอาร์เมเจอร์รีแอคชั่นเพิ่มขึ้น) เราจะพบว่า

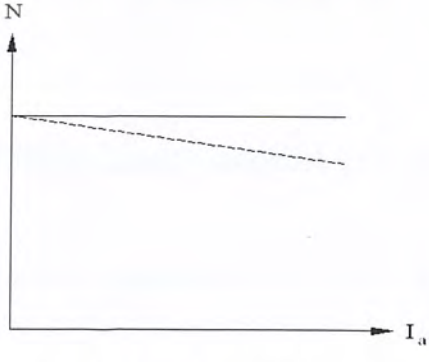
$$T_a \sim I_a$$

ดังนั้น คุณลักษณะทางไฟฟ้าจึงแสดงได้ดังรูปที่ 2.6 ในทางปฏิบัตินั้นจะเป็นเส้นตรงโดยเริ่มจากจุดศูนย์หรือจุดเริ่มต้น (Origin) ส่วนแรงบิดที่เพลานั้นแสดงด้วยเส้นประ ในขณะที่เริ่มเดินมอเตอร์นั้น ถ้ามีโหลดมากๆ จะต้องใช้กระแสจำนวนมากเพื่อใช้ในการเริ่มเดินมอเตอร์ ดังนั้นชั้นท์มอเตอร์จึงไม่สามารถที่จะเริ่มเดินในขณะที่มีโหลดจำนวนมากต่ออยู่ได้

2. คุณลักษณะของความเร็วกับกระแสในอาร์เมเจอร์ (N/I_a characteristic) ถ้าสมมติให้ Φ มี

รูปที่ 2.12 แสดงความสัมพันธ์ระหว่างแรงบิดกับกระแสในอาร์เมเจอร์

ค่าคงที่ ดังนั้น $N \sim E_b$ และ E_b ในทางปฏิบัติถือว่ามีค่าคงที่ ดังนั้น ความเร็วรอบของมอเตอร์จึงมีค่าคงที่ไปด้วยและสามารถเขียนเส้นโค้งได้ดังแสดงในรูปที่ 2.13



รูปที่ 2.13 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับกระแสในอาร์เมเจอร์

แต่ที่แท้จริงแล้ว E_b และ Φ ลดลงเมื่อเพิ่มโหลด ถึงอย่างไรก็ตาม E_b จะลดลงน้อยกว่า Φ ดังนั้น ความเร็วจึงลดลงไปบ้าง ความเร็วจะลดลงไปประมาณ 5-15 เปอร์เซ็นต์ของความเร็ว เมื่อมีโหลดเต็มที่ ซึ่งขึ้นอยู่กับการอิ่มตัวของสนามแม่เหล็ก อาร์เมเจอร์รีแอคชั่น และตำแหน่งของแปรงถ่าน (Brush position) ดังนั้น เส้นโค้งของความเร็วขณะใช้งานจริง (Actual speed curve) จึงลดลง ดังแสดงด้วยเส้นประ แต่ในทางปฏิบัติแล้ว ชั้นท์มอเตอร์นั้นถือว่าเป็นมอเตอร์ที่ให้ความเร็วคงที่ (Constant speed)

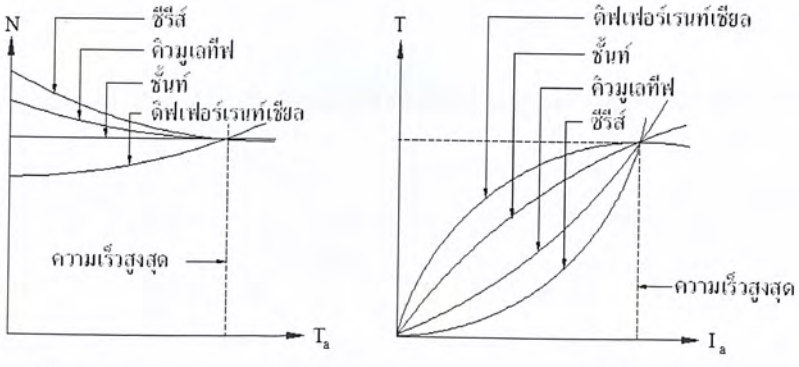
3. คุณลักษณะของความเร็วรอบกับแรงบิด (N/T_a characteristic) สามารถหาได้ จาก T_a/I_a และ

รูปที่ 2.14 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิด

คุณลักษณะของ N/I_a ซึ่งสามารถเขียนเส้นโค้งได้ดังแสดงในรูปที่ 2.8

คุณลักษณะของคอมปาวด์มอเตอร์ (Characteristics of compound motor)

มอเตอร์ชนิดนี้มีทั้งซีรีส์ฟิลด์และชั้นท์ฟิลด์ ถ้าการกระตุ้นที่ซีรีส์ฟิลด์ (Series field excitation) ช่วยการกระตุ้นที่ชั้นท์ฟิลด์ และเส้นแรงแม่เหล็กที่เกิดขึ้นที่ซีรีส์ฟิลด์ (Series flux) มีทิศทางเดียวกันกับเส้นแรงแม่เหล็กที่เกิดขึ้นที่ชั้นท์ฟิลด์ (Shunt flux)มอเตอร์ตัวนั้น เรียกว่า คิวมูเลทีฟคอมปาวด์มอเตอร์ (Cumulative compound motor) แต่ถ้าเส้นแรงแม่เหล็กที่เกิดขึ้น ที่ซีรีส์ฟิลด์มีทิศทางตรงกันข้ามกับเส้นแรงแม่เหล็กที่เกิดขึ้นที่ชั้นท์ฟิลด์แล้ว มอเตอร์ตัวนั้นเรียกว่า ดิฟเฟอร์



รูปที่ 2.15 แสดงความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิดและแรงบิดกับกระแสในอาร์เมเจอร์
(a) ความเร็วรอบกับแรงบิด
(b) แรงบิดกับกระแสในอาร์เมเจอร์

เรนท์เชียลคอมปาวด์มอเตอร์ (Differential compound motor)

คุณลักษณะของคอมปาวด์มอเตอร์นี้ ได้นำเอาคุณสมบัติของชั้นท์มอเตอร์และซีรีส์มอเตอร์มารวมเข้าด้วยกัน ซึ่งเขียนเป็นเส้นโค้งได้ดังแสดงในรูปที่ 2.15 (a) และ 2.15 (b)

1. คิวมูเลทีฟคอมปาวด์มอเตอร์ (Cumulative compound motor) เป็นเครื่องกลไฟฟ้าที่ใช้คุณสมบัติของซีรีส์มอเตอร์และชั้นท์มอเตอร์ เพื่อนำไปใช้กับเครื่องมือที่ต้องการใช้กำลังขับสูงๆ โดยต่อซีรีส์ฟิลด์ให้มีทิศทางเสริมกับชั้นท์ฟิลด์ เมื่อมอเตอร์มีโหลดมากๆ จะทำให้เส้นแรงแม่เหล็กที่ซีรีส์ฟิลด์มีมากเพิ่มขึ้น จึงทำให้แรงบิดสูงหรือมากขึ้นกว่าชั้นท์มอเตอร์ แต่การเพิ่มขึ้นของเส้นแรงแม่เหล็กนี้ จะทำให้ความเร็วลดลงไปอย่างเร็วมากกว่าชั้นท์มอเตอร์ และมอเตอร์นี้มีแรงบิด ที่อาร์เมเจอร์สูงในขณะที่ได้รับโหลดทันทีทันใด แต่ในขณะที่ไม่มีโหลดก็จะไม่มีความเร็วสูงเกินไป

2. ดิฟเฟอร์เรนท์เชียลคอมปาวด์มอเตอร์ (Differential compound motor) มอเตอร์ชนิดนี้เป็นการต่อซีรีส์ฟิลด์ให้มีทิศทางตรงกันข้ามกับชั้นท์ฟิลด์ จะทำให้เส้นแรงแม่เหล็กมีค่าลดลงเมื่อมีโหลดเพิ่มขึ้น ดังนั้น จะทำให้ความเร็วมีค่าคงที่ ($N \sim E_b/\Phi$) ความเร็วจะเพิ่มขึ้นเมื่อโหลดเพิ่มขึ้นอีก แต่ถ้าโหลดเพิ่มมากเกินไปอาจเป็นอันตรายได้ จึงไม่ค่อยจะนิยมใช้

ข้อดีและข้อเสียของชั้นท์มอเตอร์

1. ชั้นท์มอเตอร์ให้ความเร็วที่คงที่
2. ชั้นท์มอเตอร์จะให้แรงบิดขณะเริ่มเดินไม่สูง เมื่อเปรียบเทียบกับซีรีส์มอเตอร์ ดังนั้นจึงใช้ชั้นท์มอเตอร์ ดังนี้

2.1 เมื่อต้องการให้ความเร็วคงที่จากขณะไม่มีโหลดจนถึงเมื่อมีโหลดเต็มที่

2.2 เมื่อต้องการใช้กับโหลดที่มีการเปลี่ยนแปลงความเร็วซึ่งสะดวกต่อการควบคุมความเร็วและประหยัด

ข้อดีและข้อเสียของซีรีส์มอเตอร์

1. ให้แรงบิดขณะเริ่มเดินสูง
2. มีความเร็วต่ำเมื่อโหลดมากๆ และความเร็วสูงเมื่อโหลดน้อยๆ ซึ่งจะทำให้มอเตอร์ได้รับอันตรายได้ ดังนั้น จึงใช้ซีรีส์มอเตอร์เมื่อต้องการ ดังต่อไปนี้

2.1 เมื่อต้องการแรงบิดขณะเริ่มเดินสูง

2.2 เมื่อมอเตอร์สามารถต่อ (Coupling) โดยตรงกับโหลด

2.3 ถ้าความเร็วคงที่นั้นไม่เป็นปัจจัยสำคัญเลย ดังนั้น จึงลดความเร็วโดยการเพิ่มโหลด

2.4 ซีรีส์มอเตอร์ไม่สามารถใช้เมื่อมีโหลดจำนวนน้อยๆ ต่ออยู่

2.3.7 การสูญเสียและประสิทธิภาพของมอเตอร์ (Losses and Efficiency of d.c. motor)

การสูญเสียในมอเตอร์ไฟฟ้ากระแสตรงนี้ สามารถหาได้เช่นเดียวกับในเครื่องกำเนิดไฟฟ้ากระแสตรงคือ

- การสูญเสียกำลังในลวดทองแดง (Copper losses)
- การสูญเสียกำลังเนื่องจากแม่เหล็ก (Magnetic losses)
- การสูญเสียกำลังทางกล (Mechanical losses)

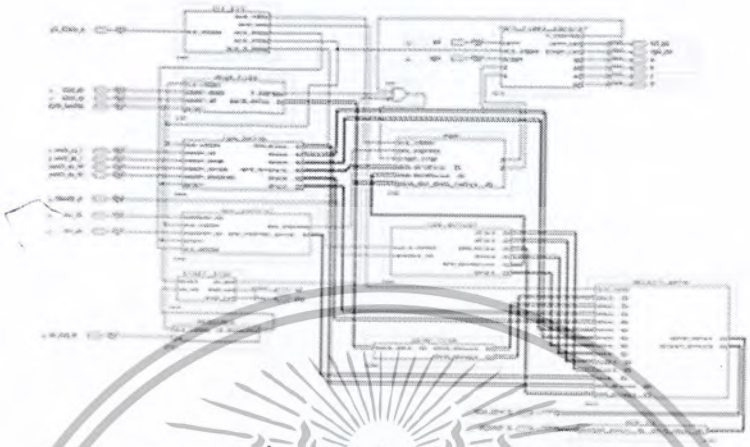
สภาวะที่เกิดกำลังสูงสุดในมอเตอร์ คือ

$$I_aR_a = V/2 = E_b$$

สภาวะที่เกิดประสิทธิภาพสูงที่สุด คือ การสูญเสียกำลังที่ขดลวดอาร์เมเจอร์ เท่ากับการสูญเสียกำลังที่มีค่าคงที่ (Armature copper losses = Constant losses) เช่นเดียวกัน

บทที่ 3

อธิบาย Block diagram



รูปที่ 3.1 บล็อกการทำงานย่อย

3.1 บล็อกการทำงาน

Clk_div จากบล็อกการทำงานจะทำหน้าที่หารความถี่ที่ป้อนเข้ามาในวงจร osc 20MHz ให้มีค่าที่ตรงตามความต้องการเพื่อที่นำไปใช้ในวงจรควบคุมมอเตอร์กระแสตรง

Dead time เป็นการควบคุมการทำงานซึ่งอยู่ในส่วนของการเลือกค่า dead time ซึ่งมีสามารถเลือกได้ดังนี้ 0.2uS, 0.4uS, 0.6uS,0.8uS และ 1.0uS เพื่อนำไปใช้ตามความเหมาะสมในวงจร

Rpm_setup เป็นการแสดงถึงการควบคุมความเร็วรอบของมอเตอร์กระแสตรงและทำหน้าที่กำหนดค่า rpm ที่ต้องการ ซึ่งสามารถเลือกได้ในช่วง 100-3000 rpm และ ±1 rpm/step

Acc_control เป็นการควบคุมอัตราการเร่งให้คงที่ของมอเตอร์กระแสตรงและจะทำหน้าที่เลือกอัตราการเร่งของมอเตอร์ซึ่ง สามารถเลือกได้ 5 ระดับ

Start_stop จะทำหน้าที่ควบคุมการหมุนของมอเตอร์ให้มีการหมุนหรือหยุดได้ตามต้องการโดยมีสวิตซ์เป็นตัวควบคุม

Debounce จากบล็อกการทำงานจะทำหน้าที่แก้ไขการ bounce ของสัญญาณเกิดจากหน้าสัมผัสของสวิตช์

Left_right_control จะทำหน้าที่ควบคุมทิศทางการหมุนของมอเตอร์กระแสตรงให้หมุนซ้ายขวาตามความต้องการ โดยจะใช้สัญญาณเป็นตัวควบคุม

PWM เป็นสัญญาณที่ถูกสร้างขึ้นโดยจะทำหน้าที่สร้างสัญญาณ PWM เพื่อส่งไปยังวงจรขับมอเตอร์กระแสตรงเพื่อควบคุมการทำงานในส่วนต่างๆ

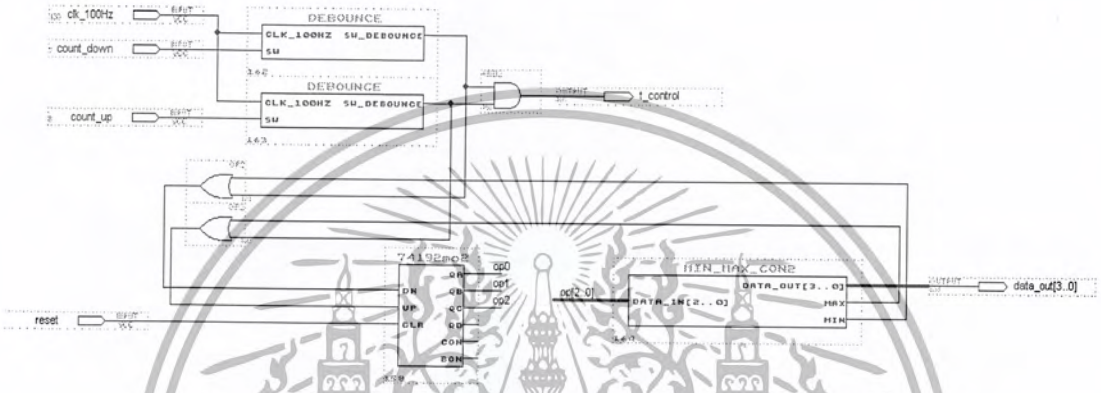
Rpm_actual ทำหน้าที่ตรวจจับรอบการทำงานจริงของมอเตอร์กระแสตรง

Dead_tcon ทำหน้าที่แปลงค่าของ deadtime เพื่อส่งต่อไปยังส่วนแสดงผล

Select_data ทำหน้าที่เลือกข้อมูลต่างๆ ที่ส่งไปยังส่วนแสดงผลเพื่อทำการแสดงค่าที่ได้ออกมา

Bcd_cu ทำหน้าที่แปลงค่า binary เป็นเลขฐานสิบ

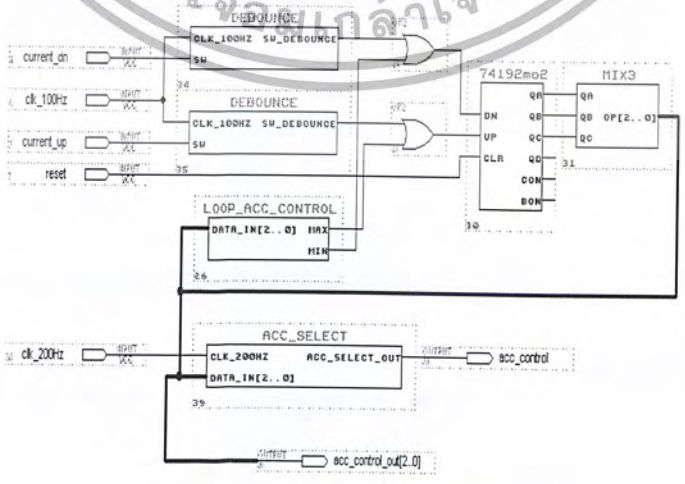
3.2 บล็อกการทำงาน dead time จะมีบล็อกการทำงานย่อยดังนี้



รูปที่ 3.2 บล็อกการทำงานย่อย

Dead time ควบคุมในส่วนของการเลือกค่า dead time ซึ่งมีสามารถเลือกได้ดังนี้ 0.2uS, 0.4uS, 0.6uS,0.8uS และ 1.0uS

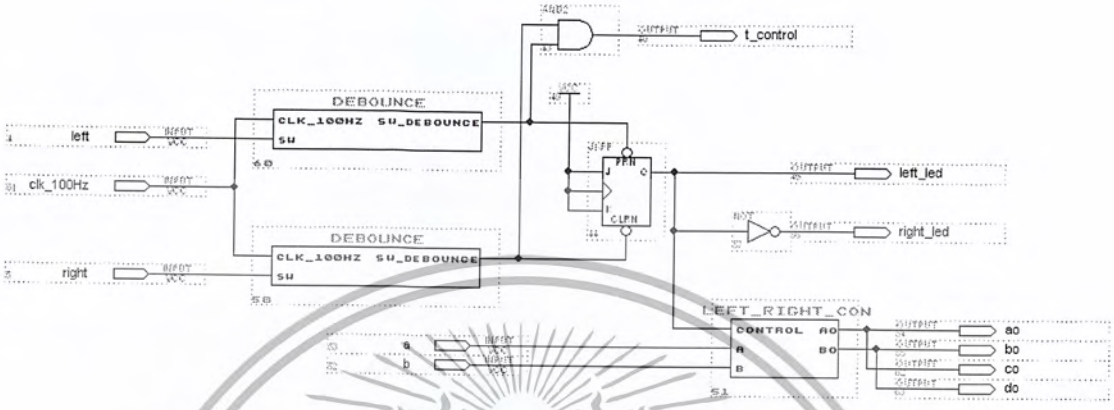
3.3 บล็อกการทำงาน acc_control จะมีบล็อกการทำงานย่อยดังนี้



รูปที่ 3.3 บล็อกการทำงานย่อย

acc_control ทำหน้าที่เลือกอัตราการเร่งของมอเตอร์ สามารถเลือกได้ 5 ระดับ

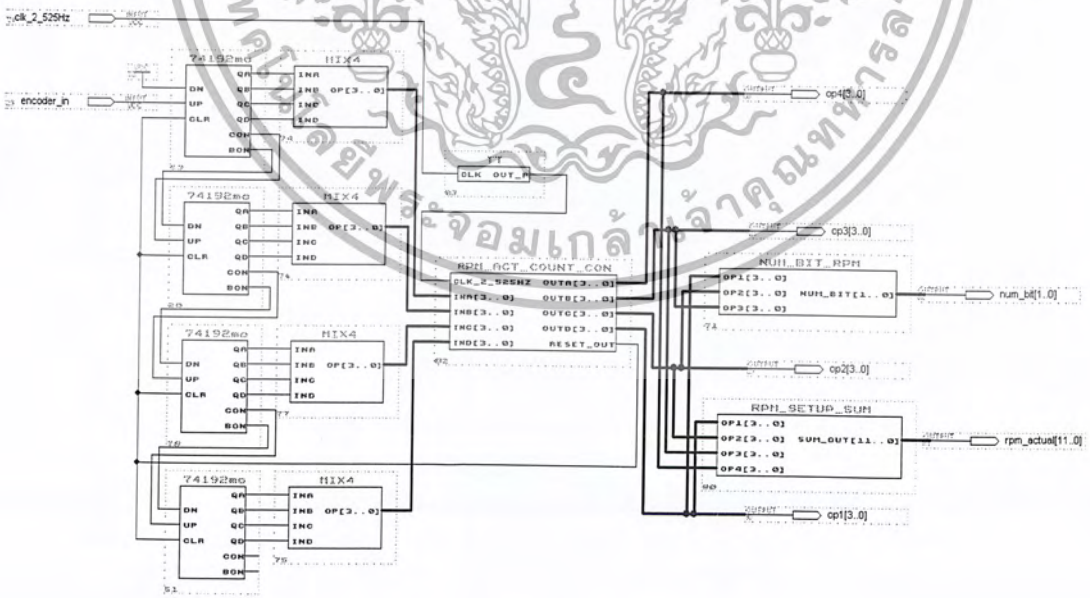
3.4 บล็อกการทำงาน left_right_control จะมีบล็อกการทำงานย่อยดังนี้



รูปที่ 3.4 บล็อกการทำงานย่อย

left_right_control ทำหน้าที่ควบคุมทิศทางการหมุนของมอเตอร์

3.5 บล็อกการทำงาน rpm_actual จะมีบล็อกการทำงานย่อยดังนี้



รูปที่ 3.6 บล็อกการทำงานย่อย

Rpm_act_count_con rpm_actual ทำหน้าที่ตรวจจับรอบการทำงานจริงของมอเตอร์

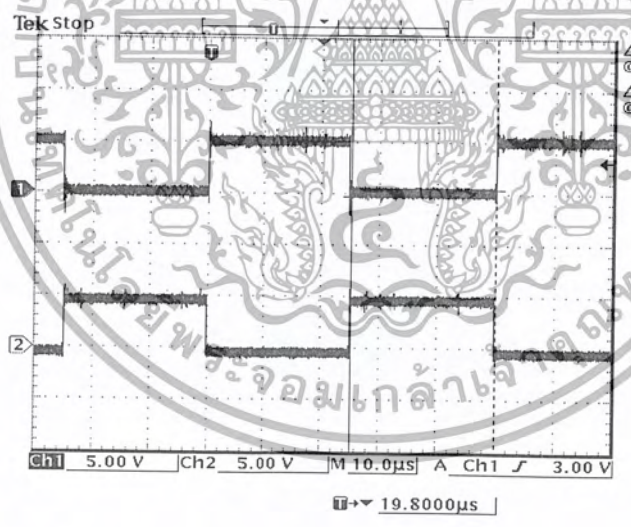
บทที่ 4

ผลการทดลอง

4.1 รายการอุปกรณ์

1. ออสซิลโลสโคป รุ่น Tektronix TDS3012
2. แหล่งจ่ายไฟกระแสตรง (Power supply)
3. Bord Poweracex1K
4. มอเตอร์กระแสตรง 24 V
5. วงจรขับมอเตอร์
6. ชุดวงจรควบคุม
7. คอมพิวเตอร์

4.2 กราฟสัญญาณ

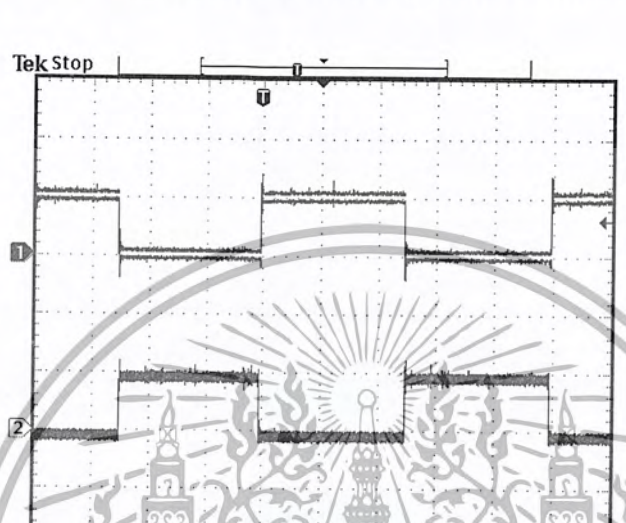


รูปที่ 4.1 กราฟแสดงสัญญาณที่ขณะที่มอเตอร์หยุดหมุน(Break)

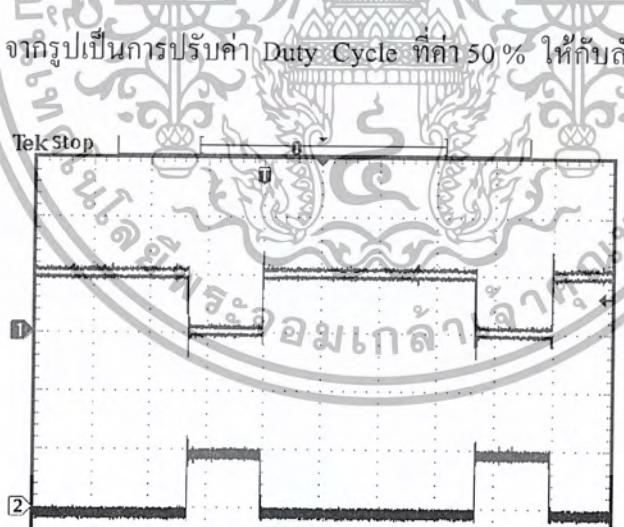
รูปที่ 4.1 Break A=B เป็นกราฟแสดงสัญญาณ ที่ระหว่างขั้ว A และ ขั้ว B ซึ่งเป็นสัญญาณควบคุม (Control) ที่เราป้อนให้กับภาคขับมอเตอร์ (Driver Motor) ซึ่งกราฟในตอนนี้เป็นสัญญาณขณะที่มอเตอร์หยุดหมุนจะสังเกตได้ว่ากราฟในขณะนี้มีค่าความกว้างพัลซ์ (Duty Cycle) จะมีค่าอยู่ที่ 50 % ทั้งทางด้านบวกและด้านลบมีค่าความกว้างพัลซ์เท่ากัน ซึ่งในขณะนี้ทางภาคขับ

(Driver Motor) ยังคงทำงานอยู่แต่ค่าเฉลี่ยทางด้านเอาท์พุตจะมีค่าเท่ากับศูนย์โวลท์ (0 v) จึงทำให้มอเตอร์หยุดหมุน.

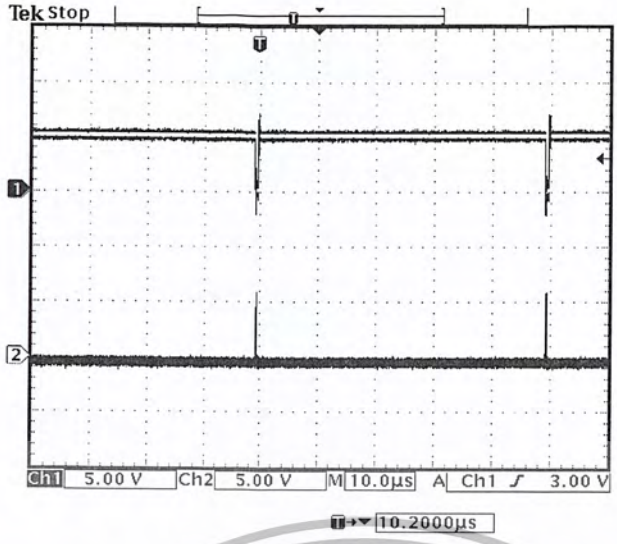
รูปที่ 4.2 4.3 4.4 ค่า Duty Cycle นั้นคือค่าอัตราส่วนของสัญญาณ คาบเวลาระหว่างค่าที่เป็นสัญญาณ ซีกบวกเทียบกับคาบเวลาทั้งหมดในช่วงเวลา 1 Cycle คิดเป็นเปอร์เซนต์ การปรับค่า Duty Cycle ในการทดลองนี้นั้นเพื่อทำการควบคุมความเร็วมอเตอร์ที่ค่าต่างๆ



รูปที่ 4.2 จากรูปเป็นการปรับค่า Duty Cycle ที่ค่า 50 % ให้กับสัญญาณที่ขั้ว A และ B

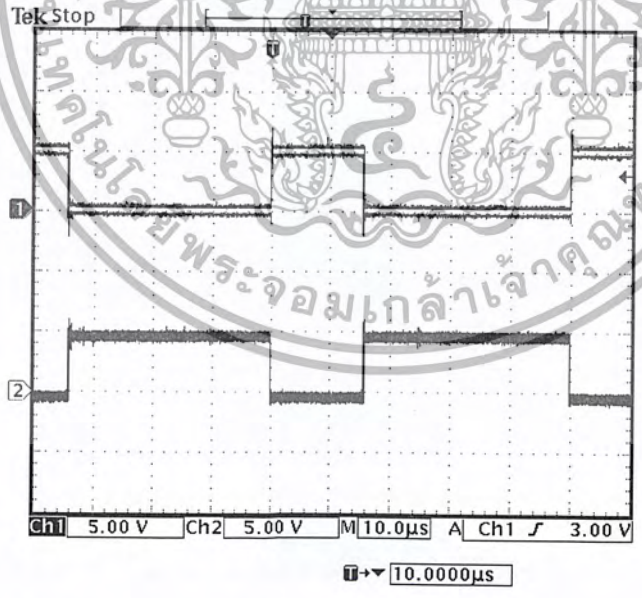


รูปที่ 4.3 จากรูปเป็นการปรับค่า Duty Cycle ที่ค่า 75 % ให้กับสัญญาณที่ขั้ว A และ B



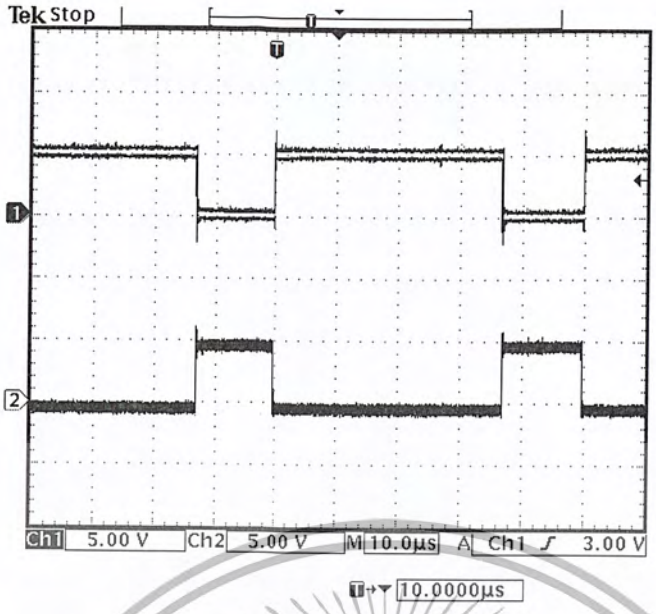
รูปที่ 4.4 จากรูปเป็นการปรับค่า Duty Cycle ที่ค่า 100 % ให้กับสัญญาณที่ขั้ว A และB

จากรูป 4.4 จะสังเกตได้ว่าเมื่อเราปรับค่า Duty Cycle มากๆแล้วก็จะทำให้ระดับของแรงดันไฟจะมีระดับเป็นไฟฟ้ากระแสตรงมากขึ้นเพราะฉะนั้นจะทำให้แรงดันเฉลี่ยกระแสตรงทางด้านเอาท์พุเพิ่มมากขึ้นก็จะทำให้ความเร็วมอเตอร์เพิ่มขึ้นตามไปด้วย เมื่อค่าของ Duty Cycle เพิ่มขึ้นและเข้าใกล้ 100% ก็จะยิ่งทำให้มอเตอร์หมุนเร็วขึ้นตามไปด้วยนั่นเอง



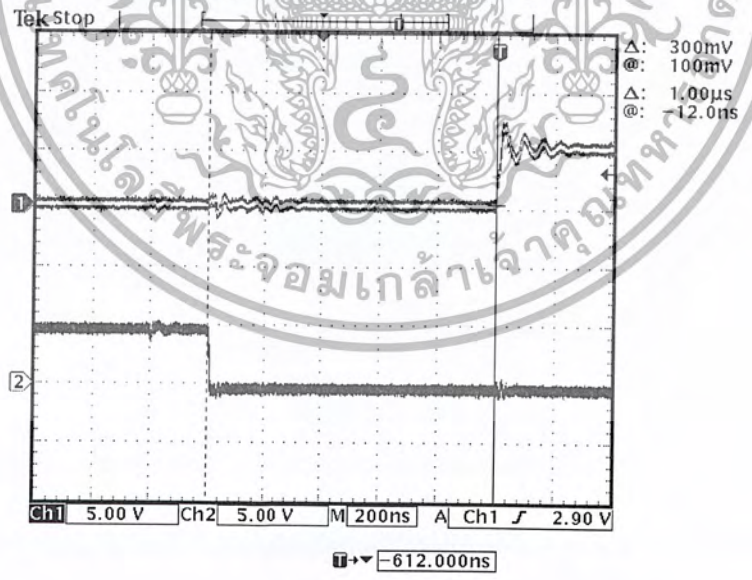
รูปที่ 4.5 Motor แสดงสัญญาณการควบคุมการหมุนทางขวาของมอเตอร์

จากรูป 4.5 จะสังเกตได้ว่าระดับแรงดันทางซีกบวกของสัญญาณ B ซึ่งจะอยู่ทางด้านล่างมีค่ามากกว่าระดับซีกบวกขอสัญญาณ A ซึ่งอยู่ทางด้านบนจึงทำให้ค่าเฉลี่ยของแรงดันของสัญญาณ B มีค่ามากกว่าจึงทำให้มอเตอร์หมุนไปทางขวามือ



รูปที่ 4.6 กราฟแสดงสัญญาณการควบคุมการหมุนทางซ้ายของมอเตอร์

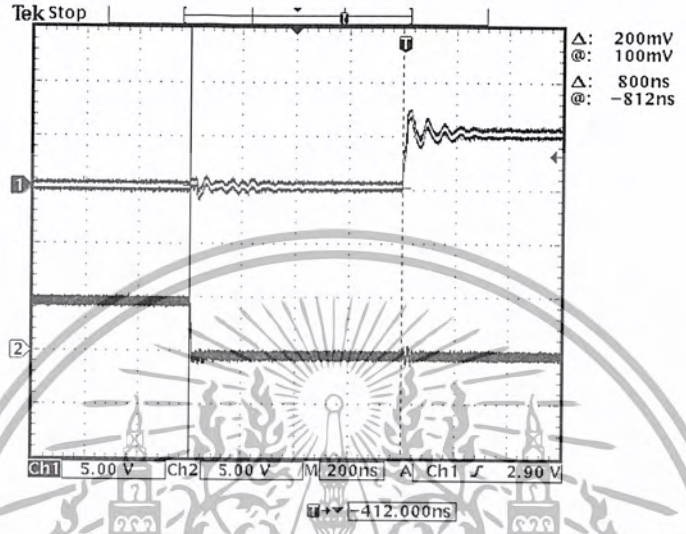
จากรูป 4.6 จะสังเกตได้ว่าระดับแรงดันทางซีกบวกของสัญญาณ B ซึ่งจะอยู่ทางด้านล่างมีค่าน้อยกว่าระดับซีกบวกของสัญญาณ A ซึ่งอยู่ทางด้านบนจึงทำให้ค่าเฉลี่ยของแรงดันของสัญญาณ B มีค่าน้อยกว่าจึงทำให้มอเตอร์หมุนไปทางซ้าย



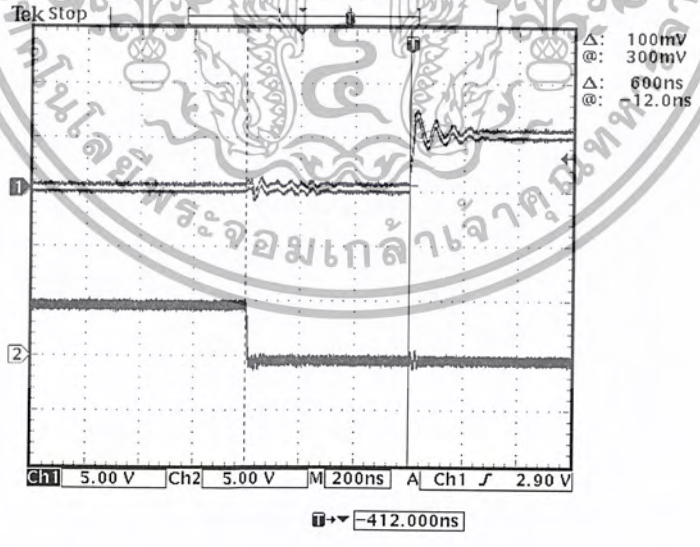
รูปที่ 4.7 Dead Time for adjust 0.2-1.0 micro(sec)

รูปที่ 4.7 การปรับ Dead Time ที่ค่าต่างๆที่ได้โปรแกรมไว้ใน FPGA จากรูปด้านล่างเป็นการปรับค่า Dead Time ที่ค่าต่างๆ กันตั้งแต่มากสุดคือ 1 ไมโครวินาที จนถึงน้อยสุดคือ 0.2

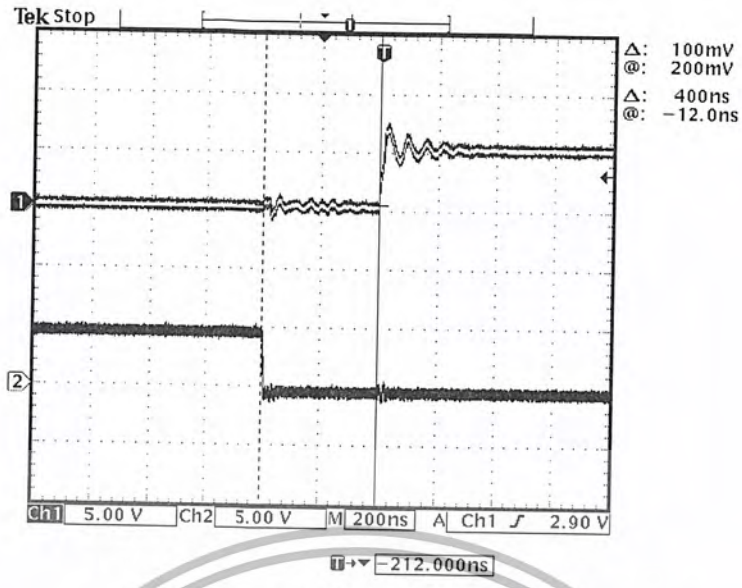
ไมโครวินาที ประโยชน์ของการปรับค่า Dead Time นั้นคือเพื่อหน่วงเวลาในการเปลี่ยนสถานะในการทำงานระหว่างสัญญาณ ที่ขั้ว A และ B คือช่วงเวลาระหว่างขอบขาขึ้นและขอบขาลงของ A และ B ตามลำดับการปรับแต่ง Dead timeระหว่างขั้วสัญญาณ AและBที่คาบเวลา 1.0 ไมโครวินาที ระยะห่างระหว่างช่วงพัลซ์ ขอบขาขึ้น และ ขอบขาลงของ A และ B ตามลำดับ



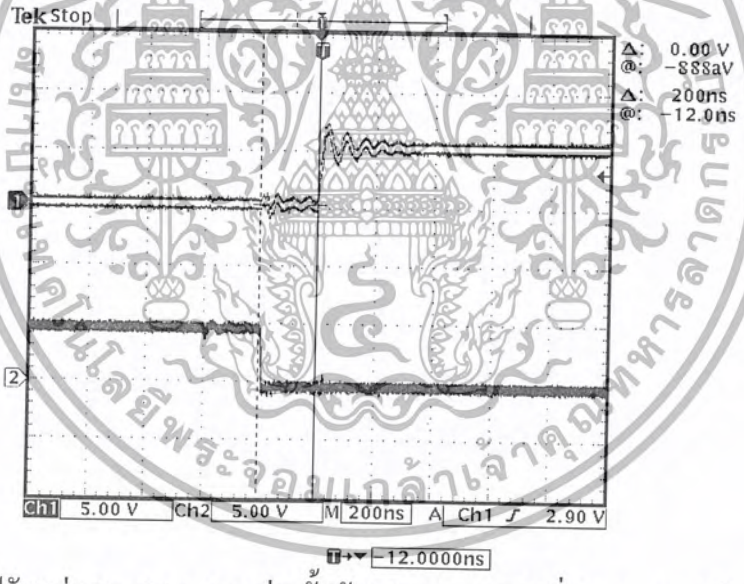
รูปที่ 4.8 การปรับแต่ง Dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.8 ไมโครวินาที ระยะห่างระหว่างช่วงพัลซ์ ขอบขาขึ้นและขอบขาลงของ A และ B ตามลำดับ



รูปที่ 4.9 การปรับแต่ง Dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.6 ไมโครวินาที ระยะห่างระหว่างช่วงพัลซ์ ขอบขาขึ้น และ ขอบขาลงของ A และ B ตามลำดับ

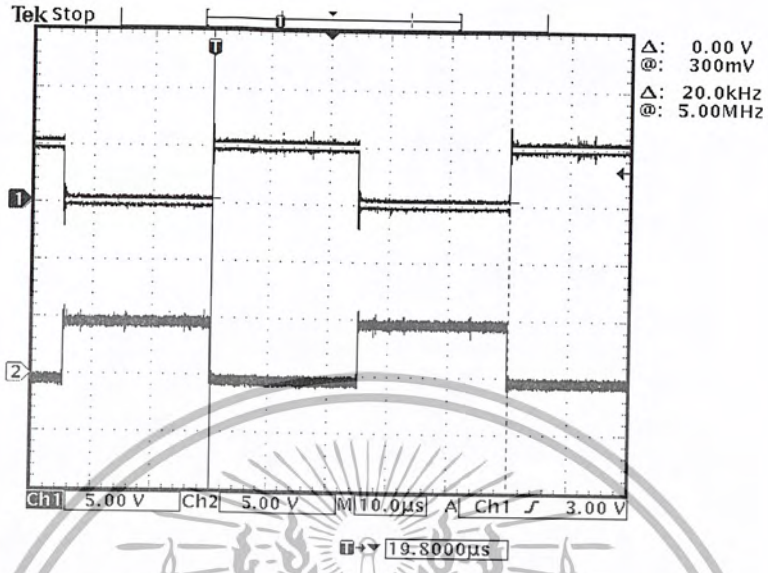


รูปที่ 4.10 การปรับแต่ง Dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.4 ไมโครวินาที ระยะห่างระหว่างช่วงพัลซ์ ขอบขาขึ้น และ ขอบขาลงของ A และ B ตามลำดับ



รูปที่ 4.11 การปรับแต่ง Dead time ระหว่างขั้วสัญญาณ A และ B ที่คาบเวลา 0.2 ไมโครวินาที

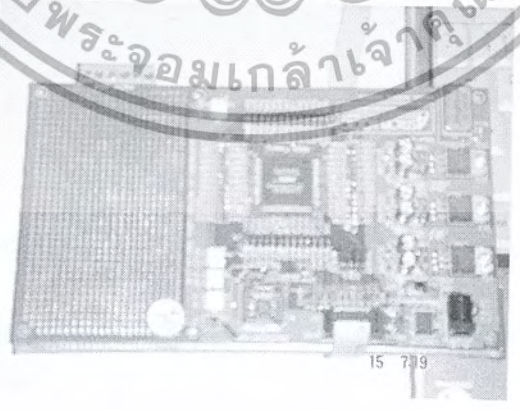
ระยะห่างระหว่างช่วงพัลซ์ ขอบขาขึ้น และ ขอบขาลงของ A และ B ตามลำดับจะสังเกตได้ว่าเมื่อเมื่อเราปรับค่าช่วงเวลาที่แตกต่างกันก็จะทำให้สัญญาณ A และB เกิดในระยะต่างๆ โดยถ้าคาบเวลาที่เราปรับมีค่ามากก็จะทำให้ระยะห่างระหว่าง A และ B มากตามไปด้วย ถ้าเราปรับค่าคาบเวลาน้อยก็จะทำให้ระยะห่างน้อยลงโดยจะแปรผันตรงตามกัน การปรับค่านี้มีผลต่อความเร็วของมอเตอร์เพราะเป็นการปรับสัญญาณควบคุมที่ค่าต่างกันโดยถ้าคาบเวลาน้อยลงจะทำให้ประสิทธิภาพในการหมุนดีขึ้น ทั้งนี้ขึ้นอยู่กับอุปกรณ์ของภาคขับเองด้วย



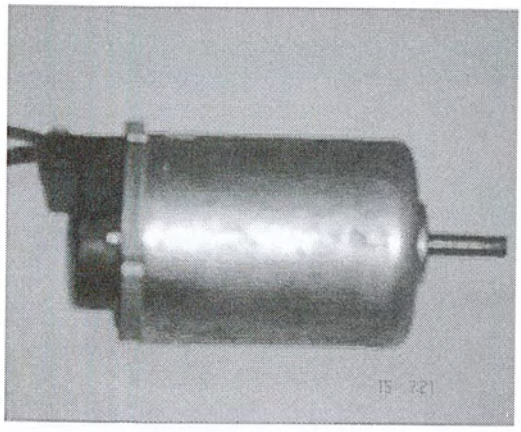
รูปที่ 4.12 Frequency

รูปที่ 4.12 กราฟแสดงสัญญาณควบคุมระหว่างขั้วA และ B เลือกใช้ความถี่ที่ 20 KHzในการเลือกใช้ความถี่นั้นเราต้องคำนึงถึงความสามารถของมอเตอร์ว่ารองรับความถี่ได้แค่ไหนในการเลือกความถี่ที่ 20KHz นี้นั้นเป็นการเลือกความถี่ให้เหมาะสมกับตัวมอเตอร์ที่เรานำมาใช้ในการทดลองนั้นมีขนาด 24V 3Aและความถี่ที่ 20KHz

4.3 รูปอุปกรณ์การทำงาน

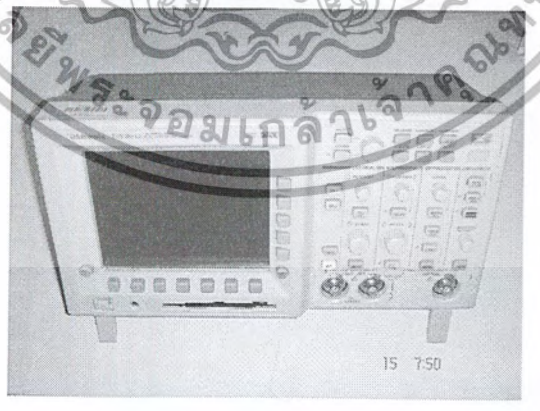


รูปที่ 4.13 Bord Poweracex1K



รูปที่ 4.14 มอเตอร์กระแสตรง 24 V

รูปที่ 4.15 วงจรขับมอเตอร์



รูปที่ 4.16 ออสซิลโลสโคบ

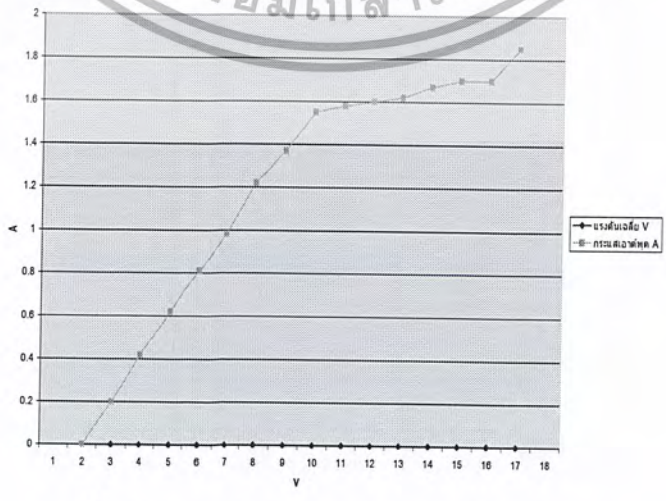
4.4 ตารางบันทึกผลของมอเตอร์ตัวที่1ขณะไม่มีโหลด

ตารางที่ 4.1 การบันทึกผลของมอเตอร์ตัวที่1ขณะไม่มีโหลด

มอเตอร์ขนาด 24V 3A

ความเร็วรอบมาตรฐาน rpm.	แรงดันเฉลี่ย V	กระแสเอาต์พุต A	ความเร็วรอบจริง Techo-Gen	ความเร็วรอบจริง Encoder	error
0	0.33mv	0	0	0	0%
200	2.05v	0.2	198	201	0.50%
400	3.30v	0.42	309	403	0.75%
600	5.00v	0.62	596	603	0.50%
800	6.85v	0.81	811	804	0.50%
1000	8.02v	0.98	996	1002	0.20%
1200	9.72v	1.22	1197	1207	0.58%
1400	10.85v	1.37	1408	1411	0.78%
1600	12.83v	1.55	1602	1609	0.56%
1800	14.47v	1.58	1807	1798	-0.11%
2000	15.97v	1.6	1996	2011	0.55%
2200	17.63v	1.62	2205	2197	-0.14%
2400	19.25v	1.67	2411	2409	0.38%
2600	20.80v	1.7	2590	2615	0.58%
2800	22.40v	1.7	2837	2812	0.43%
3000	24.23v	1.85	3025	3023	0.76%

ตารางแสดงค่าของมอเตอร์ขณะไม่มีโหลด (No Load)



รูปที่ 4.17 แสดงสัญญาณขณะมอเตอร์ไม่มีโหลด

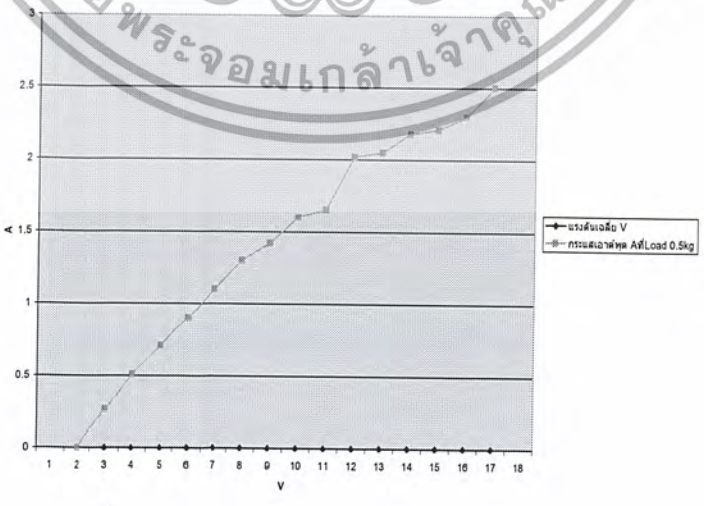
4.5 ตารางบันทึกผลของมอเตอร์ตัวที่1ขณะมีโหลด

ตารางที่ 4.2 การบันทึกผลของมอเตอร์ตัวที่1ขณะมีโหลด

มอเตอร์ขนาด 24V 3A

ความเร็วรอบมาตรฐาน rpm.	แรงดันเฉลี่ย V	กระแสเอาต์พุต A ที่Load 0.5kg	ความเร็วรอบจริง Techo-Gen	ความเร็วรอบจริง	error
0	0.33mv	0	0	0	0%
200	2.00v	0.27	196	199	-0.50%
400	3.25v	0.51	307	400	0%
600	5.01v	0.71	600	595	-0.83%
800	6.90v	0.9	805	811	1.38%
1000	8.05v	1.1	990	999	-0.10%
1200	9.80v	1.3	1192	1203	0.25%
1400	10.8v	1.42	1403	1407	0.50%
1600	12.90v	1.6	1598	1605	0.31%
1800	15.00v	1.65	1807	1803	0.16%
2000	16.01v	2.02	1999	2009	0.45%
2200	17.85v	2.05	2198	2200	0%
2400	19.30v	2.18	2400	2407	0.29%
2600	20.75v	2.21	2590	2612	0.46%
2800	22.50v	2.3	2837	2812	0.43%
3000	24.03v	2.5	3020	3019	0.63%

ตารางแสดงค่าของมอเตอร์ขณะมีโหลด (Load)



รูปที่ 4.18 แสดงสัญญาณขณะมอเตอร์มีโหลด

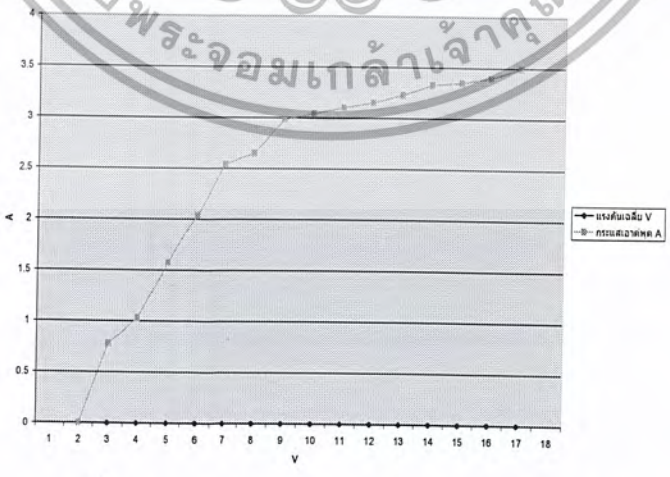
4.6 ตารางบันทึกผลของมอเตอร์ตัวที่2ขณะไม่มีโหลด

ตารางที่ 4.3 การบันทึกผลของมอเตอร์ตัวที่2ขณะไม่มีโหลด

มอเตอร์ขนาด 24V 8A

ความเร็วรอบมาตรฐาน rpm.	แรงดันเฉลี่ย V	กระแสเอาต์พุต A	ความเร็วรอบจริง Techo-Gen	ความเร็วรอบจริง Encoder	error
0	0	0	0	0	0%
200	6.05v	0.78	197	206	3%
400	7.85v	1.03	309	403	0.75%
600	9.05v	1.57	597	613	2.16%
800	9.12v	2.04	811	804	0.50%
1000	9.35v	2.54	997	1002	0.20%
1200	10.03v	2.65	1187	1207	0.58%
1400	11.23v	2.98	1408	1411	0.07%
1600	12.53v	3.04	1602	1610	0.625%
1800	15.47v	3.10	1817	1797	-0.16%
2000	15.97v	3.15	1998	2009	0.45%
2200	17.63v	3.23	2204	2198	-0.1%
2400	19.25v	3.33	2413	2411	0.45%
2600	20.80v	3.35	2590	2612	0.46%
2800	23.40v	3.4	2835	2814	0.5%
3000	24.23v	3.5	3012	3015	0.5%

ตารางแสดงค่าของมอเตอร์ขณะไม่มีโหลด (No Load)



รูปที่ 4.19 แสดงสัญญาณขณะมอเตอร์ไม่มีโหลด

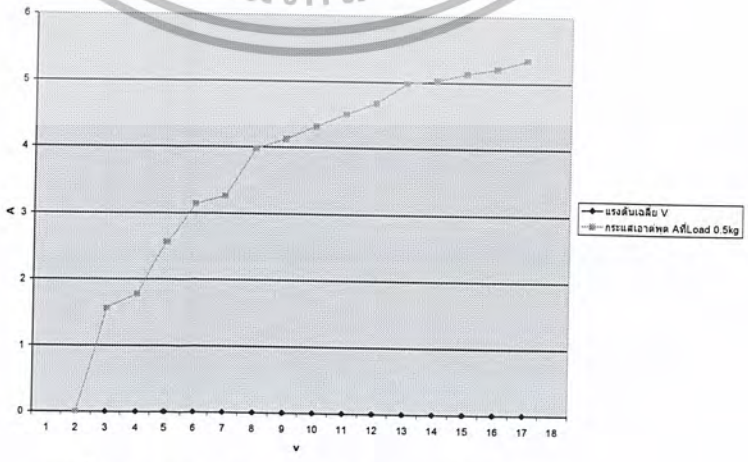
4.7 ตารางบันทึกผลของมอเตอร์ตัวที่2ขณะมีโหลด

ตารางที่ 4.4 การบันทึกผลของมอเตอร์ตัวที่2ขณะมีโหลด

มอเตอร์ขนาด 24V 8A

ความเร็วรอบมาตรฐาน rpm.	แรงดันเฉลี่ย V	กระแสเอาต์พุต Aที่Load 0.5kg	ความเร็วรอบจริง Techo-Gen	ความเร็วรอบจริง	error
0	0	0	0	0	0%
200	3.00v	1.56	198	199	-0.50%
400	3.45v	1.78	306	400	0%
600	5.01v	2.56	601	595	-0.83%
800	7.00v	3.14	805	811	1.38%
1000	8.05v	3.26	990	999	-0.10%
1200	9.80v	3.98	1195	1203	0.25%
1400	10.78v	4.13	1403	1407	0.50%
1600	12.90v	4.32	1597	1605	0.31%
1800	15.03v	4.51	1807	1803	0.16%
2000	16.01v	4.68	1997	2009	0.45%
2200	17.95v	4.98	2198	2200	0%
2400	19.75v	5.03	2401	2407	0.29%
2600	21.75v	5.13	2590	2612	0.46%
2800	22.47v	5.21	2835	2812	0.43%
3000	24.13v	5.34	3024	3019	0.63%

ตารางแสดงค่าของมอเตอร์ขณะมีโหลด (Load)



รูปที่4.20 แสดงสัญญาณขณะมอเตอร์มีโหลด

4.8 รูปแสดงผลการวัดค่าความเร็วรอบ

รูปที่ 4.21 แสดงค่าความเร็วรอบ

รูปที่ 4.22 แสดงมอเตอร์ในขณะที่ไม่มีโหลด

รูปที่ 4.23 แสดงมอเตอร์ขณะตอนมีโหลด

บทที่ 5
สรุปผลและวิจารณ์

จากผลการทดลองจะพบว่าการควบคุมมอเตอร์กระแสตรงโดยPWMที่สร้างขึ้นจากกาเขียนโปรแกรมซึ่งสัญญาณที่ได้มาจะใช้ในการควบคุมการทำงานของมอเตอร์กระแสตรงซึ่งอยู่ใรูปแบบการควบคุมความเร็วของมอเตอร์กระแสตรง (Speed) การเริ่มและหยุดของมอเตอร์กระแสตรง (Run and Stop) ควบคุมการหมุนซ้ายและหมุนขวาของมอเตอร์กระแสตรง (Turn right and Turn left) และควบคุมความเร็วให้คงที่ของมอเตอร์กระแสตรงโดยสัญญาณที่ได้มาจากการเขียนโปรแกรมซึ่งถูกแสดงออกมาเป็นPWMมาควบคุมรวมรวมถึงการเรียนรู้ถึงอุปกรณ์ต่างๆ และศึกษาถึงโครงสร้างของโปรแกรมและภาษาของ VHDL จนถึงการพัฒนาทางด้านของโปรแกรม

5.1 ประโยชน์ที่ได้รับ

1. รู้จักการทำงานของ FPGA ได้มากยิ่งขึ้น
2. สามารถเขียนโปรแกรมควบคุมการทำงาน FPGA ได้
3. สามารถนำ FPGA ไปใช้ควบคุมในงานด้านมอเตอร์กระแสตรง
4. สามารถนำ FPGA ไปประยุกต์ใช้งานได้หลากหลายเช่น ควบคุมมอเตอร์กระแสตรง

5.2 ปัญหาและอุปสรรค

1. เนื่องจาก FPGA ยังไม่ค่อยมีผู้นิยมใช้กันมากในประเทศ ดังนั้นข้อมูลหรือตำราจึงหาได้ไม่สะดวกและจำกัดอยู่ในวงแคบ

2. ขาดบุคลากรที่มีความรู้ในเรื่อง FPGA ในการให้คำปรึกษาเพราะยังไม่แพร่หลายในประเทศ

5.3 ข้อเสนอแนะ

สำหรับผู้ที่เริ่มศึกษาเกี่ยวกับตัวโปรแกรมของ FPGA นั้น ควรจะหาข้อมูลและตัวอย่างจากผู้ที่ชำนาญแล้ว เพราะจะเป็นประโยชน์ในการปะยุกต์ไปใช้งานได้อย่างหลากหลายมากยิ่งขึ้น

ภาคผนวก

ภาคผนวก ค

Source code

Clk_div ทำหน้าที่หารความถี่ที่ป้อนเข้ามา osc 20MHz ให้มีค่าที่ตรงตามความต้องการ

```
library ieee;
use ieee.std_logic_1164.all;
entity clk_div is
port(clk_20MHz
                :in std_logic;
        clk_10MHz,clk_1kHz,clk_200Hz,clk_100Hz,clk_2_525Hz                :out
std_logic
        );
end clk_div;
architecture rtl of clk_div is
signal temp_1:integer range 0 to 9999;
signal temp_2:integer range 0 to 4;
signal temp_3:integer range 0 to 98;
signal
temp_10MHz,temp_1kHz,temp_500Hz,temp_200Hz,temp_100Hz,temp_2_525Hz:std_logic;
begin
 process(clk_20MHz)
        begin
                if clk_20MHz'event and  clk_20MHz='0' then
                        temp_10MHz<=not temp_10MHz;                        -- 10MHz
                        if temp_1=9999 then                                1kHz
                                temp_1<=0;
                                temp_1kHz<=not temp_1kHz;
                        else
                                temp_1<=temp_1+1;
                        end if;
        end if;
                                clk_10MHz<=temp_10MHz;
                                clk_1kHz<=temp_1kHz;
end process;
```

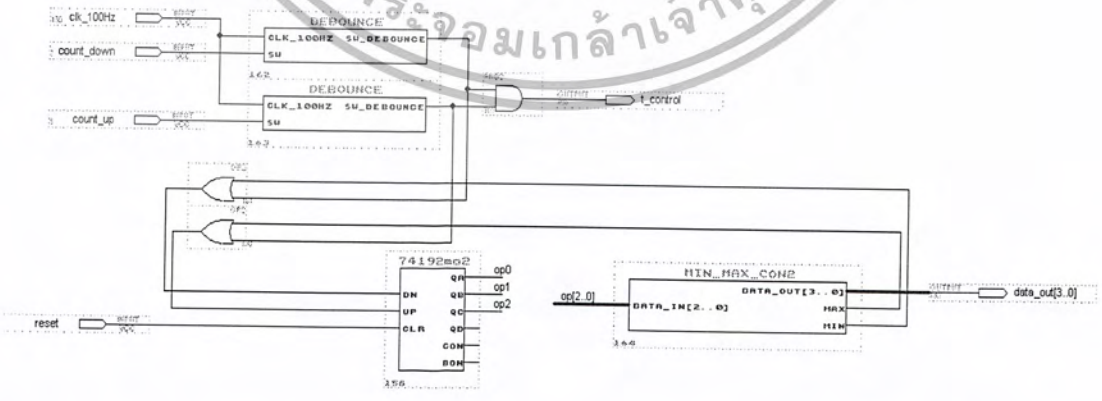
```
process(temp_1kHz)
 begin
                if temp_1kHz'event and temp_1kHz='0' then
                        temp_500Hz<=not temp_500Hz;                    --
500Hz
        end if;
 end process;
process(temp_1kHz)
        begin
                if temp_1kHz'event and temp_1kHz='0' then
                        if temp_2<4 then
        -- 200Hz
                                temp_2<=temp_2+1;
                        else
                                temp_2<=0;
                        end if;
                        case temp_2 is
                                when 0 to 2 => temp_200Hz<='1';
                                when others     => temp_200Hz<='0';
                        end case;
                                clk_200Hz<=temp_200Hz;
                end if;
end process;

process(temp_200Hz)
                begin
                if temp_200Hz'event and temp_200Hz='0' then
                        temp_100Hz<=not temp_100Hz;                    --
100Hz
                end if;
```

```
                clk_100Hz<=temp_100Hz;
        end process;
        process(temp_500Hz)
        begin
                if temp_500Hz'event and temp_500Hz='0' then
                        if temp_3<98 then
                -- 500/198 (0-98) = 2.525Hz
                                temp_3<=temp_3+1;
                        else
                                temp_3<=0;
                                temp_2_525Hz<=not temp_2_525Hz;
                        end if;
                                clk_2_525Hz<=temp_2_525Hz;
                end if;
        end process;
end rtl;
```

Dead time ควบคุมในส่วนของการเลือกค่า dead time ซึ่งมีสามารถเลือกได้ดังนี้ 0.2uS, 0.4uS, 0.6uS,0.8uS และ 1.0uS

บล็อกการทำงาน dead time จะมีบล็อกการทำงานย่อยดังนี้



Debounce ทำหน้าที่แก้ไขการ bounce ของสัญญาณเกิดจากหน้าสัมผัสของสวิตช์

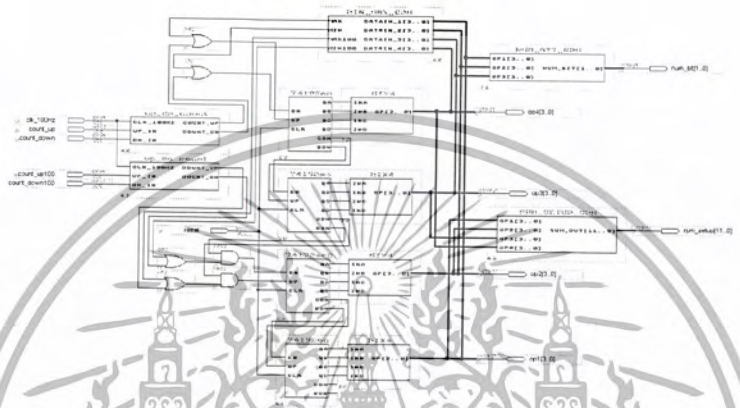
```
library IEEE;
```

```
use ieee.std_logic_1164.all;
entity Debounce is
port( Clk_100Hz          : in std_logic;
      SW                 : in std_logic;
      SW_Debounce : out std_logic
          );
end Debounce;
architecture rtl of debounce is
signal shift_sw : std_logic_vector(2 downto 0);
begin
 process(Clk_100Hz)
      begin
          if (Clk_100Hz'Event) and (Clk_100Hz = '1') then
             shift_sw(1 downto 0) <= shift_sw(2 downto 1);
             shift_sw(2) <= not sw;
          end if;
end process;
          SW_Debounce<= '1' when shift_sw(2 downto 0) = "000" else '0';
end rtl;
min_max_con2
library ieee;
use ieee.std_logic_1164.all;
entity min_max_con2 is
port(data_in              :in  integer range 0 to 7;
      data_out            :out integer range 0 to 15;
      max,min             :out std_logic
     );
end min_max_con2;
architecture rtl of min_max_con2 is
begin
  max<= '1' when data_in=5 else
        '0';
```

```
        min<= '1' when data_in=1 else '0';
data_out<=data_in*2;
end rtl;
```

Rpm_setup ทำหน้าที่กำหนดค่า rpm ที่ต้องการ ซึ่งสามารถเลือกได้ในช่วง 100-3000 rpm และ ±1 rpm/step

บล็อกการทำงาน Rpm_setup จะมีบล็อกการทำงานย่อยดังนี้



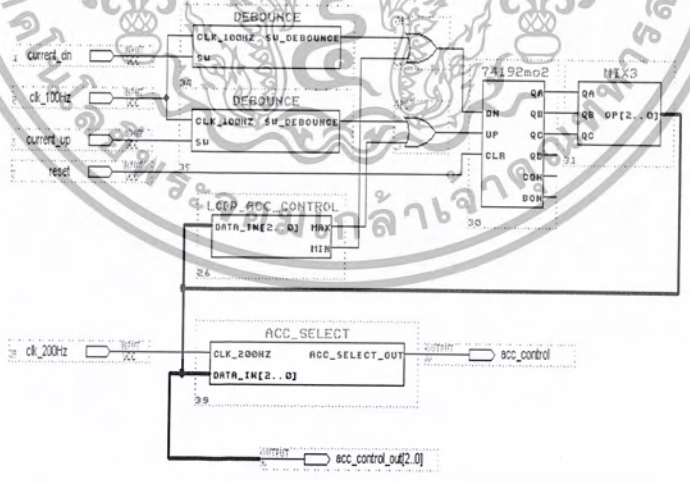
```
Rpm_setup_sum
ibrary ieee;
use ieee.std_logic_1164.all;
entity rpm_setup_sum is
port(op1,op2,op3,op4    :in integer range 0 to 15;
        sum_out                    :out integer range 0 to 3000);
end rpm_setup_sum;
architecture rtl of rpm_setup_sum is
begin
sum_out<=((op1*1000)+(op2*100)+(op3*10)+op4);
end rtl;
num_bit_rpm
library ieee;
use ieee.std_logic_1164.all;
entity num_bit_rpm is
port(op1,op2,op3                    :in integer range 0 to 15;
        num_bit                     :out integer range 0 to 3
```

```
        );
end num_bit_rpm;
architecture rtl of num_bit_rpm is
begin
        num_bit <= 0 when op1=0 and op2=0 and op3=0 else
                        1 when op1=0 and op2=0 else
                        2 when op1=0 else
                        3 ;
end rtl;
min_max_con
library ieee;
use ieee.std_logic_1164.all;
entity min_max_con is
port(datain_1,datain_2,datain_3,datain_4  :in  integer range 0 to 15;
        max,min,max100,min100                          :out std_logic
        );
end min_max_con;
architecture rtl of min_max_con is
signal temp:integer range 0 to 31;
begin
  max<= '1' when datain_1>=3  else
                '0';
      min<= '1' when datain_1=0 and datain_2<=1 and datain_3=0 and datain_4=0 else
                '0';
      max100<='1' when datain_1>=3 and datain_2>=0 and datain_3=0 and datain_4=0 else
                 '1' when datain_1=2 and datain_2=9 and datain_3>0 and datain_4>0 else
                 '0';
      min100<='1' when datain_1=0 and datain_2<=1 and datain_3>=0 and datain_4>=0 else
                '0';
end rtl;
mix4
library ieee;
```

```
use ieee.std_logic_1164.all;
entity mix4 is
port(ina,inb,inc,ind :in std_logic;
        op                          :out std_logic_vector(3 downto 0)
       );
end mix4;
architecture rtl of mix4 is
begin
        op(3)<=ind;
        op(2)<=inc;
        op(1)<=inb;
        op(0)<=ina;
end rtl;
```

acc_control ทำหน้าที่เลือกอัตราการเร่งของมอเตอร์ สามารถเลือกได้ 5 ระดับ

บล็อกการทำงาน acc_control จะมีบล็อกการทำงานย่อยดังนี้



Loop_acc_control

```
library ieee;
use ieee.std_logic_1164.all;
entity loop_acc_control is
```

```
port(data_in                    :in  integer range 0 to 7;
     max,min                          :out std_logic
    );
end loop_acc_control;
architecture rtl of loop_acc_control is

begin
   max<= '1' when data_in=5  else
              '0';
      min<= '1' when data_in=1  else
              '0';
end rtl;
acc_select
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity acc_select is
port(clk_200Hz                  :in std_logic;
     data_in                    :in integer range 0 to 7;
     acc_select_out             :out std_logic
    );
end acc_select;
architecture rtl of acc_select is
signal temp1:integer range 0 to 7;
signal temp2:integer range 0 to 7;
signal temp3:std_logic;
begin
temp2<=5-data_in;
process(clk_200Hz)
 begin
      if clk_200Hz'event and clk_200Hz='0' then
          if temp1<temp2 then
```

```
                temp1<=temp1+1;
            else temp1<=0;
                temp3<= not temp3;
            end if;
        end if;
    end process;
    acc_select_out<=temp3;
    end rtl;
    mix3
    library ieee;
    use ieee.std_logic_1164.all;
    entity mix3 is
    port(qa,qb,qc     :in std_logic;
         op                    :out std_logic_vector(2 downto 0)
         );
    end mix3;
    architecture rtl of mix3 is
    begin
        op(2)<=qc;
        op(1)<=qb;
        op(0)<=qa;
    end rtl;
```

start_stop ทำหน้าที่ควบคุมการหมุนของมอเตอร์

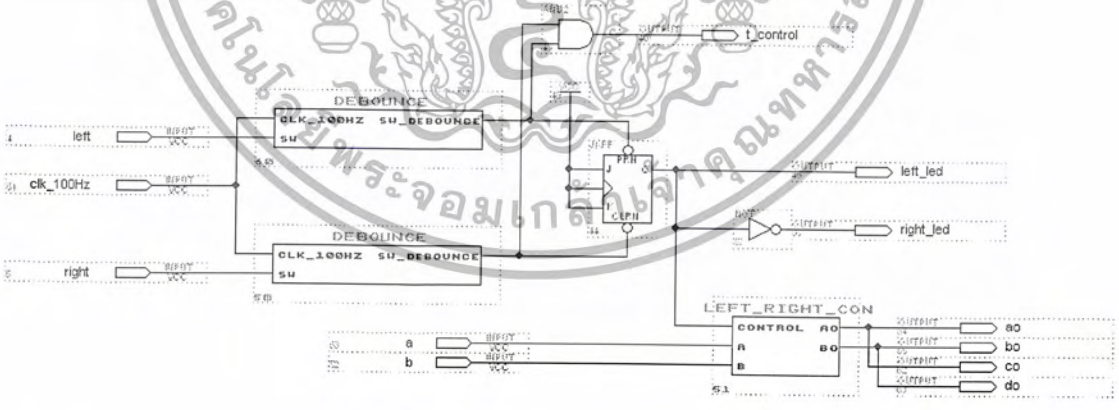
```
library ieee;
use ieee.std_logic_1164.all;

entity start_stop is
port(reset,ss_in                    :in std_logic;
     ss_out,run_led,stop_led   :out std_logic
    );
end start_stop;
```

```
architecture rtl of start_stop is
signal temp:std_logic;
begin
process(ss_in,reset)
 begin
        if reset='0' then
                temp<='0';
        elsif ss_in'event and ss_in='0' then
                temp<=not temp;
        end if;
end process;
        ss_out<=temp;
        run_led<=temp;
        stop_led<= not temp;
end rtl;
```

left_right_control ทำหน้าที่ควบคุมทิศทางการหมุนของมอเตอร์

บล็อกการทำงาน left_right_control จะมีบล็อกการทำงานย่อยดังนี้



Left_right_con

```
library ieee;
use ieee.std_logic_1164.all;
entity left_right_con is
port(control,a,b :in std_logic;
```

```
        ao,bo                :out std_logic
      );
end left_right_con;
architecture rtl of left_right_con is
begin
 ao<= a when control='0' else b;
 bo<= b when control='0' else  a;
end rtl;
```

PWM ทำหน้าที่สร้างสัญญาณ PWM เพื่อส่งไปยังวงจรขับมอเตอร์

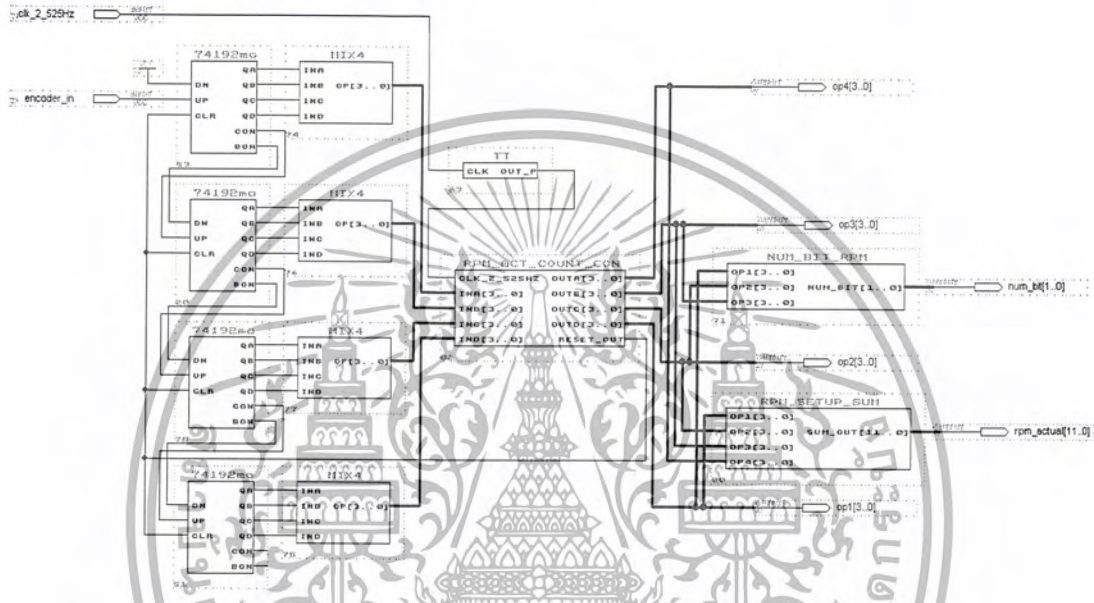
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity pwm is
port(clk_10MHz,acc_control,start_stop          :in std_logic;
        rpm_setup,rpm_actual                          :in integer range 0 to 3000;
        data_out_dead_time                            :in integer range 0 to
15;
        a,b                                           :out
std_logic
      );
end pwm;
architecture rtl of pwm is
signal count_temp:integer range 0 to 500;
signal a_temp:integer range 0 to 500;
signal b_temp:integer range 0 to 500;
signal c_temp:integer range 0 to 500;
signal d_temp:integer range 0 to 500;
signal e_temp:integer range 0 to 500;
begin
 process(clk_10MHz)
 begin
        if clk_10MHz'event and clk_10MHz='0' then
```

```
                if count_temp<499 then
                        count_temp<=count_temp+1;
                else count_temp<=0;
                end if;
        end if;
end process;
a<= '1' when count_temp>=0 and count_temp<=a_temp else
        '0';
b<= '1' when count_temp>c_temp and count_temp<=b_temp else
        '0';
process(start_stop,acc_control)
begin
        if acc_control'event and acc_control='0' then
        if start_stop='0' then
                case data_out_dead_time is
                        when 2 =>       a_temp <=247; b_temp<=497;
d_temp<=493 ;e_temp<=248;
                        when 4 =>       a_temp <=245; b_temp<=495;
d_temp<=487 ;e_temp<=246;
                        when 6 =>       a_temp <=243; b_temp<=493;
d_temp<=481 ;e_temp<=244;
                        when 8 =>       a_temp <=241; b_temp<=491;
d_temp<=475 ;e_temp<=242;
                        when others =>  a_temp <=239; b_temp<=489;
d_temp<=469 ;e_temp<=240;
                end case;
        else
                        if rpm_actual<rpm_setup and a_temp<d_temp then
                                a_temp<=a_temp+1;
                        elsif rpm_actual>rpm_setup and a_temp>=e_temp then
                                a_temp<=a_temp-1;
                        end if;
```

```
    end if;
                    c_temp<=a_temp+data_out_dead_time;
        end if;
 end process;
end rtl;
```

Rpm_act_count_con rpm_actual ทำหน้าที่ตรวจจับรอบการทำงานจริงของมอเตอร์

บล็อกการทำงาน rpm_actual จะมีบล็อกการทำงานย่อยดังนี้




```
library ieee;
use ieee.std_logic_1164.all;
entity rpm_act_count_con is
port(clk_2_525Hz                    :in std_logic;
     ina,inb,inc,ind          :in integer range 0 to 15;
     outa,outb,outc,outd      :out integer range 0 to 15;
     reset_out                          :out std_logic
    );
end rpm_act_count_con;
architecture rtl of rpm_act_count_con is
begin
 process(clk_2_525Hz)
 begin
```

```
        if clk_2_525Hz'event and clk_2_525Hz='0' then
                        outa<=ina;
                        outb<=inb;
                        outc<=inc;
                        outd<=ind;
        end if;
        reset_out<=clk_2_525Hz;
 end process;
end rtl;
num_bit_rpm
library ieee;
use ieee.std_logic_1164.all;
entity num_bit_rpm is
port(op1,op2,op3                :in integer range 0 to 15;
        num_bit                 :out integer range 0 to 3
        );
end num_bit_rpm;
architecture rtl of num_bit_rpm is
begin
        num_bit <= 0 when op1=0 and op2=0 and op3=0 else
                        1 when op1=0 and op2=0 else
                        2 when op1=0 else
                        3 ;
end rtl;
```

Dead_tcon ทำหน้าที่แปลงค่าของ deadtime เพื่อส่งต่อไปยังส่วนแสดงผล

```
library ieee;
use ieee.std_logic_1164.all;
entity dead_tcon is
port(data_in                    :in integer range 0 to 15;
        data_out1,data_out2     :out integer range 0 to 15
        );
end dead_tcon;
```

```
architecture rtl of dead_tcon is
begin
 data_out2<= 0 when data_in<10 else
                              1;
 data_out1<=data_in when data_in<10 else
                              0;
end rtl;
```

Select_data ทำหน้าที่เลือกข้อมูลต่างๆ ที่ส่งไปยังส่วนแสดงผล

```
library ieee;
use ieee.std_logic_1164.all;
entity select_data is
port(clk_1KHz              :in std_logic;
        a2,a3,a4,a5,a6,a7,a8,a9,a10,a11  :in std_logic_vector(3 downto 0);
        b1                                    :in std_logic_vector(2 downto 0);
        num_bit1,num_bit2          :in  integer range 0 to 3;
        data_out                          :out std_logic_vector(3 downto 0);
        select_bit                        :out std_logic_vector(10 downto 0)
       );
end select_data;
architecture rtl of select_data is
signal temp:integer range 0 to 10;
begin
 process(clk_1KHz)
        begin
              if clk_1KHz'event and clk_1KHz='0' then
               if temp<10 then
                     temp<=temp+1;
               else temp<=0;
               end if;
              end if;
              case temp is
              when 0 => data_out <= '0' & b1;
```

```
                    select_bit<= "11111111110";
when 1 => data_out <= a2;
                    select_bit<= "11111111101";
when 2 => data_out <= a3;
                    select_bit<= "11111111011";
when 3 =>  if num_bit1=2 then
                                select_bit<= "11111111111";
                            else
                                data_out <= a4;
                                select_bit<= "11111110111";
                            end if;
when 4 => data_out <= a5;
                    select_bit<= "11111101111";
when 5 => data_out <= a6;
                    select_bit<= "11111011111";
when 6 => data_out <= a7;
                    select_bit<= "11110111111";
when 7 =>       if num_bit2<3 then
                                select_bit<= "11111111111";
                            else
                                data_out <= a8;
                                select_bit<= "11101111111";
                            end if;
when 8 =>       if num_bit2<2 then
                                select_bit<= "11111111111";
                            else
                                data_out <= a9;
                                select_bit<= "11011111111";
                            end if;
when 9 =>       if num_bit2<1 then
                                select_bit<= "11111111111";
                            else
```

```
                                        data_out <= a10;
                                        select_bit<= "10111111111";
                                    end if;
                when 10 => data_out <= a11;
                                    select_bit<= "01111111111";
                when others=>null;
                end case;
        end process;
end rtl;
```

Bcd_cu ทำหน้าที่แปลงค่า binary เป็นเลขฐานสิบ

```
library  ieee;
use ieee.std_logic_1164.all;
entity bcd_cu is
port(data_in                    : in integer range 0 to 15;
     segment_out                : out std_logic_vector(7 downto 0)
     );
end bcd_cu;
architecture rtl of bcd_cu is

begin
        with data_in select
        segment_out<= "00000011" when 0, --0
                      "10011111" when 1, --1
                      "00100101" when 2, --2
                      "00001101" when 3, --3
                      "10011001" when 4, --4
                      "01001001" when 5, --5
                      "01000001" when 6, --6
                      "00011111" when 7, --7
                      "00000001" when 8, --8
                      "00001001" when 9, --9
                      "11111111" when others;
end rtl;
```