การกำเนิดสัญญาณ PWM โดยใช้ FPGA

PWM GENERATION USING FPGA

วรนารถ แสงฉาย

WORRANART SANGCHAI

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2544

ISBN 974-7988-66-6

PWM GENERATION USING FPGA

WORRANART SANGCHAI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2001

ISBN 974-7988-66-6

หัวข้อวิทยานิพนธ์	การกำเนิดสัญญาณ PWM โดยใช้ FPGA
นักศึกษา	นายวรนารถ แสงฉาย
รหัสประจำตัว	38061257
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2544
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ. พิชิต ลำยอง

บทคัดย่อ

วิทยานิพนธ์นี้นำเสนอการประยุกต์ใช้งานวงจรรวมประเภท Field-Programmable Gate Array (FPGA) เพื่อใช้ในการกำเนิดสัญญาณ Pulse Width Modulation (PWM) สำหรับควบคุมความเร็วและแรงบิดของมอเตอร์เหนี่ยวนำแบบ 3 เฟส งานวิจัยนี้ได้นำหลักการ Space Vector Modulation (SVM) มาใช้ในการสร้างสัญญาณ PWM แก่ชุดอินเวอร์เตอร์เนื่องจากมีความเหมาะสมในการออกแบบสร้างวงจรประเภทดิจิตอล ชุดควบคุมที่นำเสนอมีข้อดีกว่าชุดควบคุมที่ใช้ไมโครโปรเซสเซอร์ตามแบบเดิมอยู่หลายประการ เช่น ให้ความถี่สวิตชิ่งที่สูงกว่า การออกแบบทำได้ง่ายและรวดเร็ว รวมทั้งสามารถเปลี่ยนแปลงการทำงานของระบบได้โดยไม่ต้องเปลี่ยนแปลงฮาร์ดแวร์ นอกจากนี้ยังสามารถออกแบบให้ระบบทั้งหมดอยู่ภายในวงจรรวมเพียงวงจรเดียวได้ ระบบโดยรวมจึงมีขนาดเล็ก วงจรรวมที่ออกแบบจะถูกเชื่อมต่อกับคีย์บอร์ดขนาด 4 x 4 และตัวแสดงผลแบบ 7 ส่วน เพื่อใช้กำหนดตัวแปรต่างๆ ของสัญญาณ เช่นดัชนีการมอดูเลท(Modulation Index), ความถี่ในการสวิตชิ่ง, ความถี่หลักมูล(Fundamental Frequency), ค่าเดดไทม์ เป็นต้น

Thesis Title	PWM generation using FPGA
Student	Mr. Worranart Sangchai
Student ID.	38061257
Degree	Master of Engineering
Program	Electrical Engineering
Year	2001
Thesis advisor	Asst. Prof. Pichit Lumyong

ABSTRACT

This thesis presents an application of Field – Programmable Gate Array (FPGA) to generation of Pulse Width Modulation (PWM) signals for controlling speed and torque of a three phases induction motor. In this research work, Space Vector Modulation (SVM) is employed as a means to generate PWM signals for an inverter. This is attributed to its compatibility with the digital circuit design .The proposed controller has several advantages over the traditional microprocessor-based design including a higher switching frequency, lower complexity and more flexibility. More importantly, it is possible to integrate the complete system on a single chip. To facilitate the set up of several parameters e.g. modulation index, switching frequency, fundamental frequency, dead time, the integrated circuit is accommodated with a 4x4 keyboard and a 7-segment display.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงลงได้ ด้วยคำแนะนำและคำปรึกษาจาก ผศ.พิชิต ล้ายอง ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกในความอนุเคราะห์จากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ ภาควิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีมหานคร ที่ให้ช่วยเหลือด้านอุปกรณ์ที่ใช้ในการทดลอง

วรนารถ แสงฉาย

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันสัญญาณ PWM ถูกนำมาใช้อย่างแพร่หลาย โดยเฉพาะอย่างยิ่งในการควบคุมความเร็วและแรงบิดของมอเตอร์เหนี่ยวนำ และจากการวิจัยในอดีตที่ผ่านมาจะเห็นได้ว่ารูปแบบของสัญญาณ PWM ที่จ่ายให้กับมอเตอร์จะมีผลโดยตรงต่อประสิทธิภาพในการควบคุมมอเตอร์ จึงได้มีการพัฒนาวิธีการกำเนิดสัญญาณ PWM กันอย่างต่อเนื่องและหลากหลายเพื่อให้การควบคุมมอเตอร์มีประสิทธิภาพดีที่สุด โดยในช่วงแรกจะใช้วิธีการทางอนาลอกโดย การนำสัญญาณที่ต้องการ (Modulating Signal) เช่น สัญญาณรูปซายนูซอยน์(Sinusoidal signal) มาเปรียบเทียบขนาดกับสัญญาณรูปฟันเลื่อย(Saw tooth) เป็นต้น ซึ่งวิธีนี้มีข้อเสียคือ มีความถูกต้องแม่นยำต่ำ, ในกรณีที่ใช้กับอินเวอร์เตอร์ 3 เฟสจะเกิดความไม่สอดคล้อง(Synchronization) กันของสัญญาณขึ้น เป็นต้น และในช่วงต่อมาได้มีการนำไมโครโปรเซสเซอร์มาควบคุมการกำเนิดสัญญาณ ส่งผลให้สามารถควบคุมการกำเนิดสัญญาณได้แม่นยำและมีประสิทธิภาพดีขึ้น รวมไปถึงวงจรโดยรวมมีขนาดเล็กลงและปรับเปลี่ยนรูปแบบการกำเนิดสัญญาณได้โดยง่าย ทำให้ไมโครโปรเซสเซอร์ได้รับความนิยมอย่างแพร่หลาย แต่อย่างไรก็ตามการกำเนิดสัญญาณ PWM โดยใช้ไมโครโปรเซสเซอร์ก็ยังไม่ใช่วิธีที่เหมาะสมนัก เนื่องจากผู้พัฒนาระบบต้องมีความรู้ความเข้าใจเกี่ยวกับไมโครโปรเซสเซอร์ตัวที่ต้องการใช้เป็นอย่างดี จึงจะสามารถออกแบบระบบและเขียนโปรแกรมควบคุมระบบที่มีเสถียรภาพที่ดีได้ และไมโครโปรเซสเซอร์ที่ต่างตระกูลกันจะมีโครงสร้างทางฮาร์ดแวร์และชุดคำสั่งที่แตกต่างกัน มีผลให้ถ้าผู้พัฒนาต้องการเปลี่ยนไปใช้ไมโครโปรเซสเซอร์ตระกูลใหม่ที่มีความสามารถมากขึ้น ก็จำเป็นต้องศึกษาโครงสร้างของไมโครโปรเซสเซอร์ตัวใหม่นี้ด้วย ทำให้ขาดความยืดหยุ่นและเกิดความล่าช้าในการพัฒนาระบบ นอกจากนี้ไมโครโปรเซสเซอร์ต้องใช้โปรแกรมเพื่อควบคุมการทำงาน จึงมีโอกาสที่สัญญาณรบกวนที่เกิดจากการสวิตชิ่งของชุดอินเวอร์เตอร์ จะทำให้ไมโครโปรเซสเซอร์กระโดดไปกระทำคำสั่งในตำแหน่งที่ไม่ถูกต้องได้ ระบบจึงมีความน่าเชื่อถือน้อยลง

1.2 วัตถุประสงค์ของการวิจัย

จากการศึกษาวิจัยในอดีตพบว่าการใช้ไมโครโปรเซสเซอร์ในการกำเนิดสัญญาณ PWM ยังไม่ใช่วิธีที่เหมาะสม เนื่องจากมีข้อจำกัดอยู่หลายประการ ดังนั้นในวิทยานิพนธ์ฉบับนี้จึงขอนำเสนอทางเลือกใหม่ในการพัฒนางานวิจัยที่เกี่ยวกับการกำเนิดสัญญาณ PWM คือการนำอุปกรณ์ประเภท

Programmable Logic Device เช่น FPGA (Field Programmable Gate Array) เป็นต้น มาประยุกต์ใช้ในการกำเนิดสัญญาณ PWM ซึ่งโครงสร้างภายในของ FPGA จะเสมือนกับประกอบด้วยลอจิกเกทจำนวนมาก โดยผู้ใช้สามารถกำหนดลักษณะการเชื่อมต่อกัน (Configuration) ของลอจิกเกทเหล่านี้ได้ จึงนำไปประยุกต์สร้างเป็นวงจรลอจิกต่างๆ ได้อย่างมากมาย โดยในวิทยานิพนธ์นี้ได้เลือกใช้วงจรรวม FPGA ของ ALTERA ในตระกูล FLEX (Flexible Logic Element Matrix) คือ FLEX10K30EQC208 ซึ่งวงจรภายในจะเสมือนกับประกอบด้วยลอจิกเกทประมาณ 30,000 ตัว ซึ่งสามารถนำไปออกแบบเป็นวงจรลอจิกขนาดใหญ่ได้ นอกจากนี้ภายในตัววงจรรวมยังประกอบด้วยส่วน EAB (Embedded Array Block) จำนวน 6 บล็อก แบ่งเป็นบล็อคละ 4096 บิท ซึ่งส่วน EAB นี้ สามารถออกแบบให้เป็นหน่วยความจำส่วนที่ใช้เก็บข้อมูลคงที่เพื่อใช้ในการคำนวณได้ มีผลให้ระบบที่ออกแบบไว้ไม่จำเป็นต้องต่อหน่วยความจำภายนอก ทำให้ประหยัดเนื้อที่ของแผ่นวงจรและมีคุณสมบัติความเป็นตัวควบคุมแบบชิปเดี่ยว(Single chip controller)

โดยในการออกแบบและพัฒนาวงจรภายในของวงจรรวมตระกูล FLEX10K นั้น จะใช้ชุดพัฒนา (Development Tool) ชื่อ Max Plus II ของบริษัท Altera ซึ่งสามารถกำหนดลักษณะการเชื่อมต่อกันของวงจรภายในตัว FLEX10K โดยใช้ภาษา AHDL (Altera High Description Language) ซึ่งโปรแกรม MaxPlus II นี้สามารถคอมไพล์ภาษา AHDL และจำลองการทำงาน (Simulation) ของวงจรที่ออกแบบไว้ ก่อนที่จะนำไปโปรแกรมลงในตัว FLEX10K ทำให้มีความยืดหยุ่นในการออกแบบสูง

ส่วนวิธีการกำเนิดสัญญาณ PWM ที่นำมาใช้เป็นตัวอย่างในการออกแบบนั้น ได้เลือกใช้วิธี Space Vector Modulation เนื่องจากเป็นวิธีแบบดิจิตอล จึงเหมาะที่จะนำมาสร้างด้วยวงจรลอจิกบนตัว FPGA และวิธีนี้ยังเป็นที่นิยมในปัจจุบันเนื่องจากมีคุณสมบัติที่ดีกว่าวิธีเดิมๆ หลายประการ เช่น ให้แรงดันเอาท์พุทได้สูงกว่าวิธีทั่วไปเมื่อเทียบที่แรงดันไฟตรงค่าเท่ากัน, มีค่า Total Harmonic Distortion (THD) และ Torque Ripple ต่ำ นอกจากนี้ยังเป็นวิธีที่พิจารณาผลรวมของแรงดันไฟทั้ง 3 เฟสเป็นปริมาณเวคเตอร์เพียงค่าเดียว โดยประยุกต์ใช้ทฤษฎี Space Vector จึงตัดปัญหาเรื่องความไม่สอดคล้องกันของสัญญาณทั้ง 3 เฟสได้ เป็นต้น โดยวงจรรวมที่ออกแบบนี้จะสามารถสร้างสัญญาณ PWM ได้ 3 รูปแบบคือ Symmetric sequence, Alternating zero vector sequence และ Bus clamped โดยวงจรรวมจะเชื่อมต่อกับคีย์บอร์ดขนาด 4 x 4 เพื่อใช้กำหนดค่าตัวแปรต่างๆ ของสัญญาณ เช่น ดัชนีการมอดูเลท (Modulation index), ความถี่ในการสวิตชิ่ง, ความถี่หลักมูล และค่าเดดไทม์ เป็นต้น

ในส่วนของชุดอินเวอร์เตอร์แบบ 3 เฟสที่ใช้ในการทดลองนั้น ได้ใช้โมดูลสำเร็จรูปเบอร์ PM75RHA060 ซึ่งทนแรงดันไฟตรง(DC link) ได้ 600 โวลต์ กระแสสูงสุด 75 แอมป์ ทดสอบกับโหลดมอเตอร์เหนี่ยวนำ 3 เฟส ขนาด 1 แรงม้าที่แรงดันไฟตรง 310 โวลต์

1.3 ขั้นตอนการศึกษา

1. ศึกษาโครงสร้างของ FPGA ในตระกูล FLEX10K
2. ศึกษาวิธีการเขียนโปรแกรมโดยใช้ภาษา AHDL เพื่อใช้ออกแบบวงจรลอจิก
3. ศึกษาหลักการของ Space Vector Modulation
4. นำหลักการ Space Vector Modulation มาออกแบบเป็นวงจรลอจิก
5. นำวงจรลอจิกที่ออกแบบได้มาเขียนเป็นโปรแกรมภาษา AHDL
6. จำลองผลการทำงานของวงจรที่ออกแบบไว้ วิเคราะห์ผลและแก้ไขวงจร
7. นำผลที่ได้ไปโปรแกรมลงบน FPGA
8. วิเคราะห์สัญญาณจริงที่ FPGA กำเนิดออกมา
9. นำสัญญาณ PWM ที่ได้ไปทดลองขับมอเตอร์เหนี่ยวนำ 3 เฟส ผ่านชุดอินเวอร์เตอร์
10. สรุปผลการทดลอง

1.4 ขอบเขตของการศึกษา

วิทยานิพนธ์ฉบับนี้มีจุดมุ่งหมายหลักอยู่ที่การศึกษาถึงการนำวงจรรวม FPGA มาใช้ในการออกแบบวงจรลอจิกเพื่อใช้ในการกำเนิดสัญญาณ PWM ตลอดจนแสดงขั้นตอนต่าง ๆ นับจากการออกแบบวงจรด้วยภาษา AHDL การจำลองผลการทำงาน การโปรแกรมผลลงในตัว FPGA การวิเคราะห์ผลของสัญญาณจริงที่ได้จาก FPGA รวมไปถึงการนำสัญญาณ PWM ที่กำเนิดได้ไปขับมอเตอร์เหนี่ยวนำผ่านทางชุดอินเวอร์เตอร์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

การประยุกต์ใช้ FPGA เพื่อกำเนิดสัญญาณ PWM จะเป็นทางเลือกใหม่ในการพัฒนางานวิจัยที่เกี่ยวกับการกำเนิดสัญญาณ PWM เนื่องจากมีข้อดีกว่าการใช้ไมโครโปรเซสเซอร์ในแบบเดิมอยู่หลายประการ เช่น

1. การออกแบบวงจรทำได้ง่ายและรวดเร็วเนื่องจากไม่จำเป็นต้องต่อเป็นวงจรลอจิกจริง

2. เป็นการออกแบบวงจรลอจิกโดยการเขียนเป็นภาษาที่เรียกว่า Hardware Description Language (HDL) ซึ่งเป็นการออกแบบที่ไม่ขึ้นกับฮาร์ดแวร์ของ FPGA ซึ่งผู้ใช้สามารถนำภาษา HDL ออกแบบไว้ไปใช้กับ FPGA ได้ทุกตระกูล จึงมีความยืดหยุ่นในการออกแบบสูง โดยเฉพาะในกรณีที่ต้องการเปลี่ยนไปใช้ FPGA ตระกูลที่มีความสามารถสูงขึ้น

3. สัญญาณ PWM ที่ได้จากวงจรรวม FPGA เป็นสัญญาณที่เกิดจากการทำงานของวงจรลอจิกภายในตัว FPGA จึงไม่จำเป็นต้องใช้โปรแกรมเพื่อควบคุมการทำงาน ดังนั้นระบบโดยรวมจึงมีเสถียรภาพดีกว่าการใช้ไมโครโปรเซสเซอร์ในการกำเนิดสัญญาณ

4. การปรับเปลี่ยนวิธีการกำเนิดสัญญาณทำได้ง่าย

5. สามารถออกแบบให้วงจรโดยรวมมีขนาดเล็ก ซึ่งประกอบด้วยวงจรรวม FPGA เพียงตัวเดียวได้ ซึ่งแตกต่างจากการใช้ไมโครโปรเซสเซอร์ที่ต้องใช้โปรแกรมควบคุมการทำงาน ดังนั้นระบบจึงต้องประกอบด้วยวงจรรวมอย่างต่ำ 2 ตัว คือ ไมโครโปรเซสเซอร์และหน่วยความจำรอม

6.จากการทดลอง พบว่าใช้จำนวนลอจิกเกทภายในของ FLEX10K30EQC208 เพียง 65% ของจำนวนเกททั้งหมด จึงมีเกทเหลืออยู่อีกจำนวนมาก ซึ่งในอนาคตสามารถพัฒนาโดยการออกแบบ วงจรควบคุมต่าง ๆ ลงไปในตัว FPGA ได้อีก เช่น วงจรควบคุมกระแสแบบลูปปิด (Close loop current control) เป็นต้น

7.ถึงแม้ว่าในปัจจุบันราคาของวงจรรวม FPGA จะสูงกว่าไมโครโปรเซสเซอร์อยู่มากก็ตาม แต่จากข้อดีหลายๆประการ จึงพบว่า FPGA เริ่มเป็นที่นิยมมากขึ้น จึงมีแนวโน้มที่ราคาของชิป FPGA จะลดลงมาเป็นระดับเดียวกับไมโครโปรเซสเซอร์ได้ในอนาคต

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 บทนำ

ในส่วนของทฤษฎีที่เกี่ยวข้องกับวิทยานิพนธ์ฉบับนี้จะแบ่งออกเป็น 2 ส่วน โดยในบทที่ 2 นี้จะกล่าวถึงการประยุกต์ใช้ทฤษฎี Space Vector เพื่อนำมากำเนิดสัญญาณ PWM โดยเรียกวิธีนี้ว่า Space Vector Modulation และส่วนทฤษฎีในบทที่ 3 เกี่ยวกับโครงสร้างพื้นฐานของ วงจรรวม FPGA ในตระกูล FLEX10K

2.2 ทฤษฎี Space Vector

สำหรับมอเตอร์ไฟฟ้ากระแสสลับแบบ 3 เฟส ที่ถูกต่อแบบ star จะพบว่าปริมาณทางเฟสจะเป็นตัวแปรที่ไม่เป็นอิสระต่อกัน เช่น

$$V_{sa}(t) + V_{sb}(t) + V_{sc}(t) = 0 \tag{2.1}$$

โดยที่ $V_{sa}(t), V_{sb}(t)$ และ $V_{sc}(t)$ คือแรงดันเฟสของมอเตอร์
โดยสามารถแปลงค่าตัวแปรทั้งสามนี้ให้อยู่ในระบบ 2 แกน ได้เป็น

$$\begin{bmatrix} V_{sd}(t) \\ V_{sq}(t) \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & \cos\theta & \cos 2\theta \\ 0 & \sin\theta & \sin 2\theta \end{bmatrix}\begin{bmatrix} V_{sa}(t) \\ V_{sb}(t) \\ V_{sc}(t) \end{bmatrix} \tag{2.2}$$

โดยที่ $\theta = \frac{2\pi}{3}$

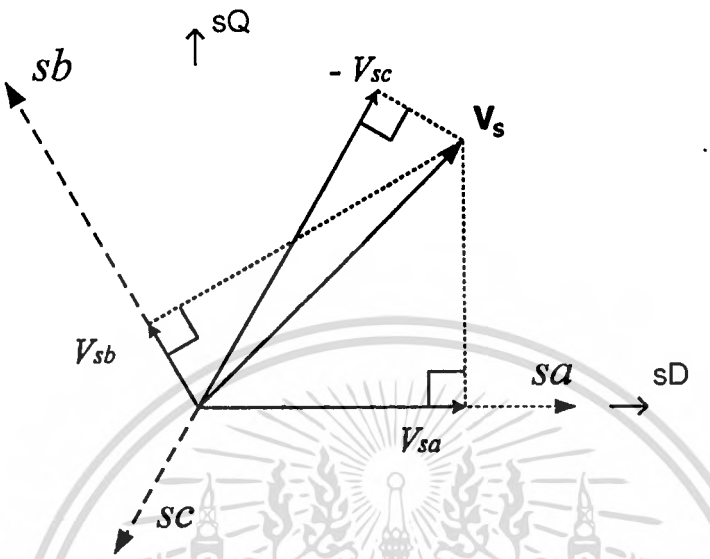
ถ้าสมมุติให้ $V_{sd}(t)$ มีทิศทางอยู่ในแกนจริง(Real axis,sD) และ $V_{sq}(t)$ อยู่ในแกนจินตภาพ(Imaginary axis,sQ) ของระนาบเชิงซ้อน จะสามารถนิยาม Space Vector ของแรงดันสเตเตอร์ ($\mathbf{V_s}(t)$) ให้อยู่ในรูปเชิงซ้อนได้เป็น

$$\mathbf{V_s}(t) = V_{sd}(t) + jV_{sq}(t) \tag{2.3}$$

และสามารถเขียน (2.3) ให้อยู่ในรูปแบบที่สัมพันธ์กับปริมาณใน 3 เฟสได้เป็น

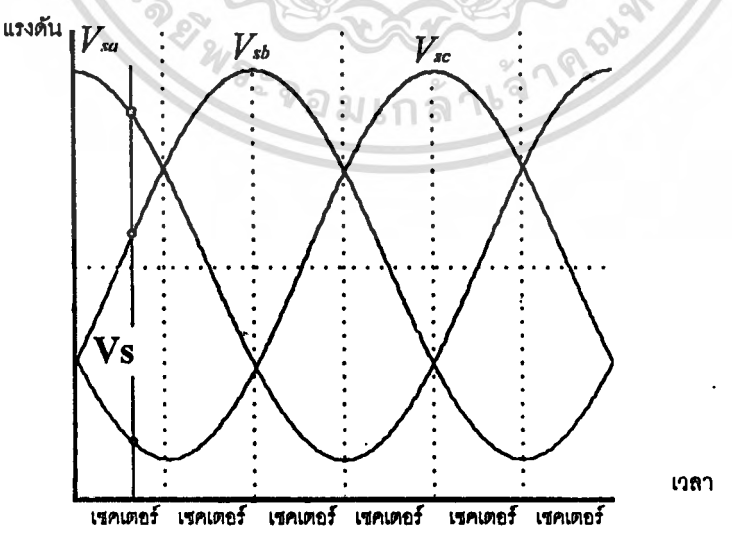
$$\mathbf{V_s}(t) = \frac{2}{3}\left(V_{sa}(t) + \gamma V_{sb}(t) + \gamma^2 V_{sc}(t)\right) \quad (2.4)$$

โดยที่ $\gamma = e^{2\pi j/3}$ และแสดงความสัมพันธ์ได้ดังรูปที่ 2.1



รูปที่ 2.1 ความสัมพันธ์ของ Space Vector $\mathbf{V_s}$ และแรงดันในแต่ละเฟส

โดยที่แกนจริงของระบบ 2 แกน จะถูกเลือกให้ตรงกับแกนของแรงดันเฟส a (*sa*) และค่าแรงดันในแต่ละเฟสของ Space Vector $\mathbf{V_s}$ ใดๆ ที่สัมพันธ์กับสมการ 2.4 หาได้จากค่าโปรเจคชั่น (Projection) ของเวคเตอร์ $\mathbf{V_s}$ บนแกนของแรงดันเฟสนั้นๆ ดังรูปที่ 2.1 ความสัมพันธ์เมื่อเทียบกับเวลาดังรูปที่ 2.2



รูปที่ 2.2 ความสัมพันธ์ของ Space Vector $\mathbf{V_s}$ และแรงดันในแต่ละเฟสเมื่อพิจารณาเทียบกับเวลา

และถ้าพิจารณากรณีที่แรงดันทั้ง 3 เฟสสมดุลย์ โดย

$$V_{sa}(t) = V_m \cos(\omega t)$$
$$V_{sb}(t) = V_m \cos(\omega t + \frac{2\pi}{3}) \quad (2.5)$$
$$V_{sc}(t) = V_m \cos(\omega t + \frac{4\pi}{3})$$

โดยที่ V_m คือขนาดของแรงดันในแต่ละเฟส

ω คือความเร็วเชิงมุมของสัญญาณมีหน่วยเป็นเรเดียนต่อวินาที

และถ้าแทนสมการ (2.5) ลงในสมการที่ (2.4) จะสามารถหา Space Vector ของแรงดัน $\mathbf{V_s}$ ได้เป็น

$$\mathbf{V_s}(t) = V_m e^{j\omega t} = V_m(\cos\omega t + j\sin\omega t) \quad (2.6)$$

พบว่าในระบบไฟฟ้า 3 เฟสที่สมดุลย์ ผลรวมของตัวแปรทั้ง 3 เฟส สามารถถูกแปลงให้เป็น Space Vector ที่หมุนวนอยู่ในระนาบเชิงซ้อนได้ โดยขนาดของ Space Vector จะเท่ากับขนาดของตัวแปรเฟสนั้น และความเร็วในการเคลื่อนตัวในระนาบเชิงซ้อนของ Space Vector ก็จะเท่ากับความเร็วเชิงมุมของตัวแปรเฟสนั้น

2.3 Space Vector ของแรงดันที่เกิดจากอินเวอร์เตอร์

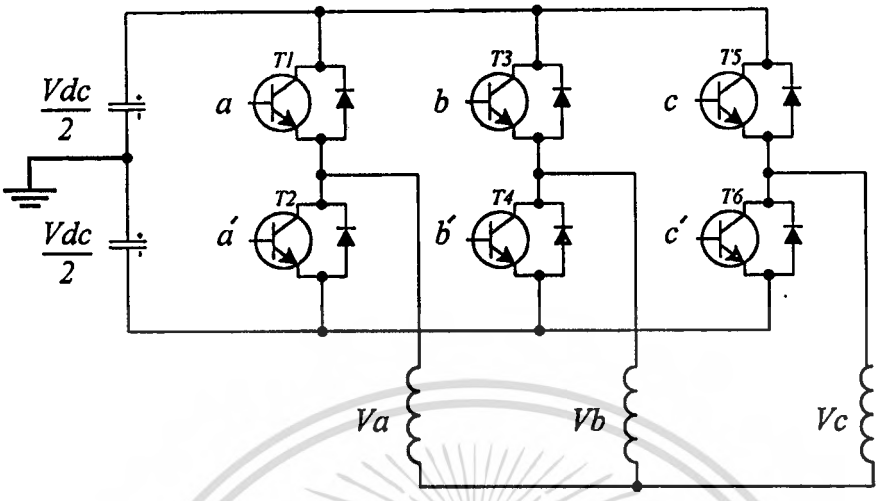
โครงสร้างของอินเวอร์เตอร์ชนิดแหล่งจ่ายแรงดันไฟฟ้า (Voltage Source Inverter) แบบ 3 เฟส แสดงดังรูปที่ 2.3

โดย V_a, V_b และ V_c คือแรงดันไฟฟ้าเอาต์พุทของอินเวอร์เตอร์

T_1 T_6 คือทรานซิสเตอร์กำลังที่ถูกควบคุมการนำกระแสหรือหยุดนำกระแสด้วยสัญญาณพัลซ์จาก $a, a', b, b' c, c'$

V_{dc} คือ แรงดันจากแหล่งจ่ายไฟตรง

ซึ่งความสัมพันธ์ระหว่างสถานะการสวิทชิ่งของทรานซิสเตอร์กับแรงดันเฟส (V_a, V_b, V_c) ที่เอาท์พุทของอินเวอร์เตอร์แสดงในรูปเมตริกซ์ดังสมการที่ 2.7



รูปที่ 2.3 โครงสร้างของอินเวอร์เตอร์ชนิดแหล่งจ่ายแรงดันไฟฟ้าแบบ 3 เฟส

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \frac{1}{3} V_{dc} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad (2.7)$$

โดย *a*, *b*, *c* แทนสถานะการสวิตชิ่งของทรานซิสเตอร์ในแต่ละเฟส

และจากเงื่อนไข 2 ประการคือ

1. ตัวแปร *a*, *b*, *c* แต่ละตัวจะมีค่าได้ 2 ค่าคือ

   - เป็น 1 แทนสถานะการนำกระแสของทรานซิสเตอร์ตัวบน

   - เป็น 0 แทนสถานะการหยุดนำกระแสของทรานซิสเตอร์ตัวบน

2. ทรานซิสเตอร์ที่อยู่ในกิ่งเดียวกัน เช่น $_1$ และ $_2$ จะได้รับสัญญาณควบคุม (a, a') ที่มีสถานะตรงข้ามกัน จึงนำกระแสไม่พร้อมกัน

ทำให้สามารถสร้างสถานะการสวิทชิ่งที่แตกต่างกันได้ 8 สถานะ โดยแต่ละสถานะจะให้แรงดันเอาท์พุทที่แตกต่างกัน แสดงได้ดังตารางที่ 2.1 โดยค่าที่แสดงจะเป็นค่าที่เทียบกับแรงดันที่บัสไฟตรง (V_{dc})

ตารางที่ 2.1 ความสัมพันธ์ระหว่างสถานะการสวิทชิ่งกับแรงดันเฟสของอินเวอร์เตอร์

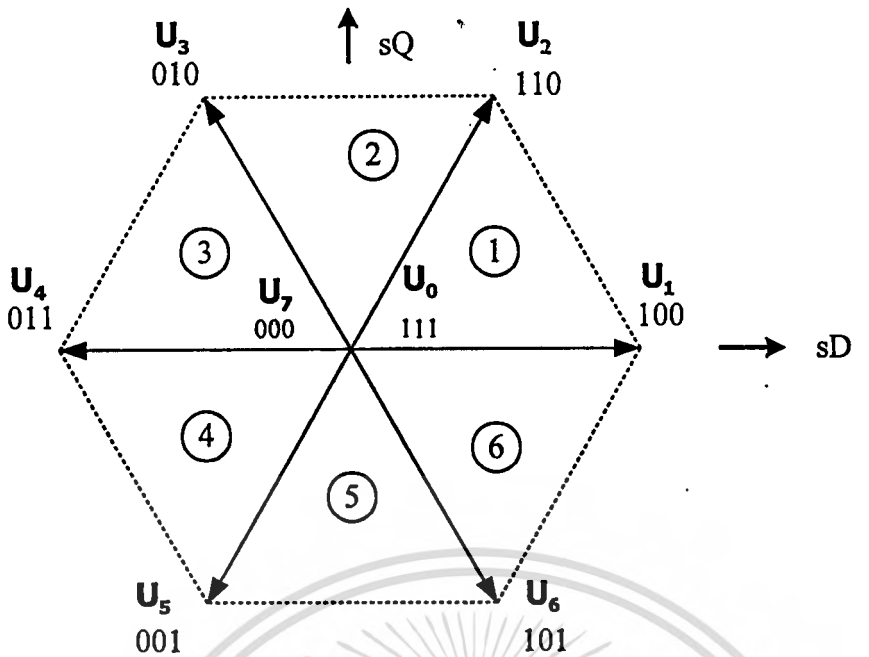
สถานะการสวิทชิ่ง			**แรงดันในแต่ละเฟสเทียบกับแรงดันที่บัสไฟตรง**		
a	b	c	Va	Vb	Vc
0	0	0	0	0	0
0	0	1	-1/3	-1/3	2/3
0	1	0	-1/3	2/3	-1/3
0	1	1	-2/3	1/3	1/3
1	0	0	2/3	-1/3	-1/3
1	0	1	1/3	-2/3	1/3
1	1	0	1/3	1/3	-2/3
1	1	1	0	0	0

และถ้านำแรงดันเฟสของแต่ละเงื่อนไขแปลงให้อยู่ในรูป Space Vector **โดยการแทนลงใน** (2.4) **จะได้** Space Vector **ของแรงดันที่มีขนาดและทิศทางแสดงดังตารางที่** 2.2

ตารางที่ 2.2 ความสัมพันธ์ระหว่างสถานะการสวิทชิ่งกับ Space Vector($\mathbf{V_s}$)

a	b	c	$\mathbf{V_s}$	ชื่อ
0	0	0	0	$\mathbf{U_0}$
0	0	1	$\frac{2}{3}V_{dc}\angle 240^\circ$	$\mathbf{U_5}$
0	1	0	$\frac{2}{3}V_{dc}\angle 120^\circ$	$\mathbf{U_3}$
0	1	1	$\frac{2}{3}V_{dc}\angle 180^\circ$	$\mathbf{U_4}$
1	0	0	$\frac{2}{3}V_{dc}\angle 0^\circ$	$\mathbf{U_1}$
1	0	1	$\frac{2}{3}V_{dc}\angle 300^\circ$	$\mathbf{U_6}$
1	1	0	$\frac{2}{3}V_{dc}\angle 60^\circ$	$\mathbf{U_2}$
1	1	1	0	$\mathbf{U_7}$

เขียนแสดงได้ดังรูปที่ 2.4



รูปที่ 2.4 เวคเตอร์แรงดันที่ชุดอินเวอร์เตอร์สร้างขึ้นจากสถานะการสวิทชิ่งทั้ง 8 สถานะ

และเขียนสมการทั่วไปของเวคเตอร์แรงดันได้เป็น

$$\mathbf{U}_k = \begin{cases} \frac{2}{3} V_{dc}\, e^{\frac{j(k-1)\pi}{3}} & k = 1,2,\ldots 6 \\ 0 & k = 0,7 \end{cases} \qquad (2.8)$$

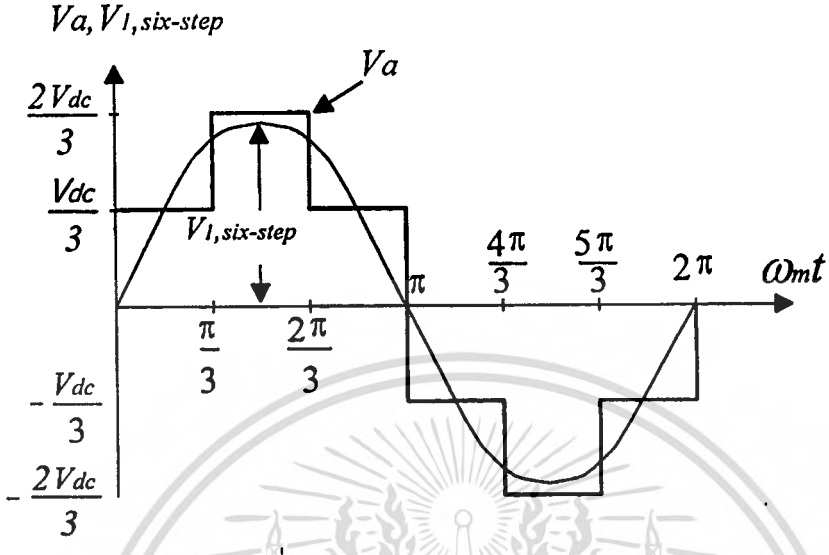
ซึ่งจะเห็นว่าอินเวอร์เตอร์ชนิดแหล่งจ่ายแรงดันแบบ 3 เฟส จะสามารถสร้างเวคเตอร์ของแรงดันได้ 8 เวคเตอร์ตามสถานะการสวิตชิ่ง โดยเวคเตอร์แรงดันทั้ง 8 นี้สามารถจำแนกออกได้เป็น 2 กรณี คือ

1. เวคเตอร์ที่มีขนาดเท่ากับ $\frac{2}{3}V_{dc}$ จำนวน 6 เวคเตอร์ ($\mathbf{U}_1$ - $\mathbf{U}_6$) โดยเวคเตอร์ทั้ง 6 นี้ จะแบ่งระนาบเชิงซ้อนออกเป็น 6 เซกเตอร์เท่า ๆ กัน และเส้นประในรูปที่ 2.4 ที่ประกอบกันเป็นรูป 6 เหลี่ยมนี้คือขนาดของเวคเตอร์แรงดันที่ใหญ่ที่สุดที่ชุดอินเวอร์เตอร์จะสามารถสร้างได้ ดังนั้นเวกเตอร์ใดๆ ที่ชุดอินเวอร์เตอร์สร้างขึ้นจะต้องมีขนาดอยู่ภายในพื้นที่รูปหกเหลี่ยมนี้ นอกจากนี้เวคเตอร์ที่มีทิศทางตรงข้ามกัน เช่น $\mathbf{U}_1$ และ $\mathbf{U}_4$ จะเป็นเวคเตอร์ที่เกิดจากสถานะการสวิตชิ่งที่ตรงข้ามกัน เช่น $\mathbf{U}_1$ เกิดจากสถานะการสวิตชิ่งของ abc เป็น 100 ส่วน $\mathbf{U}_4$ เกิดจากสถานะของ abc ที่เป็น 011 เป็นต้น

2. เวคเตอร์ที่มีขนาดเป็นศูนย์ (Zero Vector) จำนวน 2 เวคเตอร์ คือ $\mathbf{U}_0$ และ $\mathbf{U}_7$ ซึ่งมีตำแหน่งอยู่ที่จุดกำเนิด (Origin) ของระนาบเชิงซ้อน

2.4 ความสามารถของอินเวอร์เตอร์ในการสร้างเวคเตอร์แรงดัน

ค่าสูงสุดของแรงดันมูลฐาน(Maximum fundamental phase voltage) ที่ชุดอินเวอร์เตอร์จะสามารถสร้างได้ จะเกิดจากการจ่ายสัญญาณแบบ six-step ให้กับอินเวอร์เตอร์ดังรูปที่ 2.5



รูปที่ 2.5 ค่าสูงสุดของแรงดันมูลฐานที่สร้างโดยการจ่ายสัญญาณแบบ six-step

โดยที่องค์ประกอบหลัก(Fundamental component, $V_{1,SIX-STEP}$) ของแรงดันเฟสที่ได้จากการจ่ายสัญญาณแบบ six-step จะได้เท่ากับ

$$V_{1,SIX-STEP} = \frac{4}{\pi}\frac{V_{dc}}{2} \tag{2.9}$$

และถ้าพิจารณาแรงดันมูลฐานของสัญญาณ PWM ที่สร้างจากวิธี Sinusoidal PWM (SPWM) พบว่าค่าสูงสุดของแรงดันมูลฐานจะเท่ากับ

$$V_{1,SPWM} = \frac{V_{dc}}{2} \tag{2.10}$$

คือประมาณ 78.5% ของความสามารถที่อินเวอร์เตอร์จะจ่ายได้ (เมื่อเทียบกับค่าสูงสุดที่ได้จากการจ่ายแบบ six-step) ในขณะที่วิธี SVM ซึ่งจะรวมแรงดันทั้ง 3 เฟสเข้าด้วยกัน เพื่อให้เกิดเป็นเวคเตอร์แรงดันที่หมุนวนอยู่ในระนาบเชิงซ้อนดังสมการ

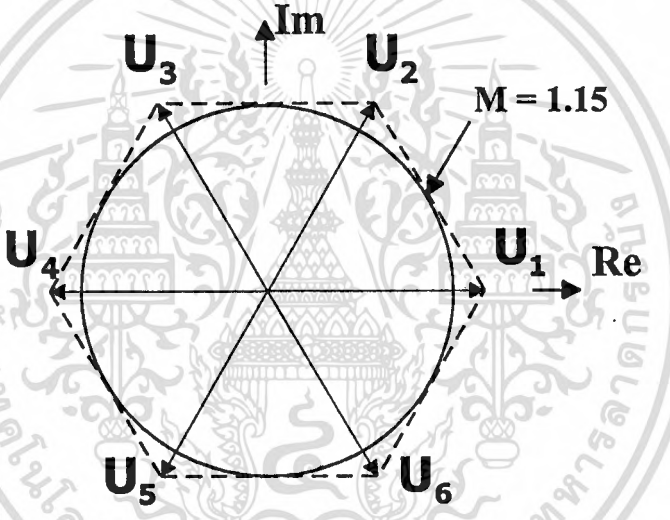
$$\mathbf{V} = V_m e^{j\omega_m t} \tag{2.11}$$

$$= M\frac{V_{dc}}{2}e^{j\omega_m t}$$

โดย M คือดัชนีการมอดูเลทและถ้าพิจารณาเฉพาะช่วงการมอดูเลทเชิงเส้น(Linear Modulation) จากรูปที่ 2.6 พบว่าเวคเตอร์แรงดันที่สร้างด้วยวิธี Space Vector Modulation ที่มีขนาดใหญ่ที่สุดจะเท่ากับรัศมีของวงกลมที่สัมผัสกับขอบในของรูปหกเหลี่ยมที่จุดกึ่งกลางของเส้นประ แสดงดังรูปที่ 2.6 ซึ่งในกรณีนี้จะให้ค่าดัชนีการมอดูเลทสูงสุดเป็น 1.15 และค่าสูงสุดของแรงดันมูลฐานจะเท่ากับ

$$V_{1,SVM} = \frac{V_{dc}}{\sqrt{3}} \tag{2.12}$$

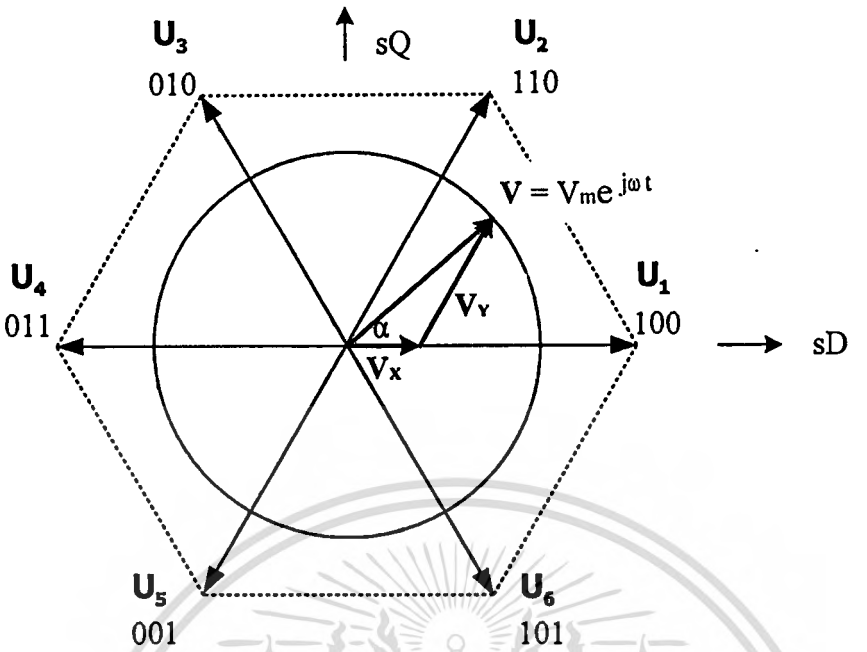
คือประมาณ 90.6% ของความสามารถที่อินเวอร์เตอร์จะจ่ายได้ และให้ค่าแรงดันสูงสุดมากกว่าวิธี SPWM อยู่ 15%



รูปที่ 2.6 เวคเตอร์แรงดันที่ใหญ่ที่สุดที่เกิดขึ้นเมื่อค่ามอดูเลชั่นอินเด็กเป็น 1.15

2.5 การมอดูเลตแบบเวคเตอร์ (Space Vector Modulation)

จากการประยุกต์ใช้ทฤษฎี Space Vector วิเคราะห์ระบบแรงดันไฟฟ้า 3 เฟส พบว่าเกิดเวคเตอร์แรงดันที่หมุนวนอยู่ในระนาบเชิงซ้อน โดยความเร็วในการเคลื่อนตัวของเวคเตอร์จะเท่ากับความเร็วเชิงมุม(ω)ของแรงดันไฟฟ้าและเพื่อสร้างเวคเตอร์แรงดันที่เคลื่อนตัวในระนาบเชิงซ้อนอย่างต่อเนื่อง ดังนั้นเวคเตอร์แรงดันที่ชุดอินเวอร์เตอร์ต้องจ่ายให้กับโหลดจึงมีจำนวนมากแต่ชุดอินเวอร์เตอร์สามารถสร้างเวคเตอร์แรงดันได้เพียง 8 รูปแบบเท่านั้น ดังนั้นการสร้างเวคเตอร์แรงดันที่ตำแหน่งใดๆบนระนาบเชิงซ้อนจึงต้องนำเวคเตอร์หลักเหล่านี้มาประกอบกันเพื่อให้ได้เวคเตอร์ใด ๆ ที่ต้องการ ซึ่งการมอดูเลตแบบเวคเตอร์เป็นการประยุกต์ใช้เวคเตอร์ข้างเคียง(Adjacent Vector) ในแต่ละเซกเตอร์มาประกอบกันเพื่อสร้างเป็นเวคเตอร์ใด ๆ ดังในรูปที่ 2.7



รูปที่ 2.7 การสร้างเวคเตอร์แรงดันใดๆ **V** จากเวคเตอร์หลัก

เช่นถ้าพิจารณาว่าเวคเตอร์แรงดันใดๆ ที่ต้องการสร้างคือ **V** มีขนาดและทิศทางคงที่ในคาบเวลาแซมปลิ้ง T_s ดังนั้นการสร้างเวคเตอร์แรงดัน **V** แสดงโดยสมการที่ (2.13)

$$\mathbf{V} = \frac{1}{T_s}(V_a t_a + V_b t_b + V_z t_z) \tag{2.13}$$

โดยที่ t_a, t_b, t_z คือช่วงเวลาการสร้าง V_a, V_b, V_z ตามลำดับ

V_a, V_b คือเวคเตอร์ข้างเคียงในแต่ละเซกเตอร์ เช่น ถ้า **V** อยู่ในเซกเตอร์ที่ 1 เวคเตอร์ข้างเคียงคือ $V_a = \mathbf{U}_1, V_b = \mathbf{U}_2$ เป็นต้น

V_z คือ เวคเตอร์ศูนย์ ($\mathbf{U}_0$ หรือ $\mathbf{U}_7$) เกิดขึ้นเนื่องจากผลรวมของ t_a และ t_b มักจะน้อยกว่าคาบเวลาแซมปลิ้ง ดังนั้นจึงต้องสร้างเวคเตอร์ศูนย์ขึ้น เพื่อรักษาคาบเวลาแซมปลิ้งให้คงที่ในทุกตำแหน่งของการสร้างเวคเตอร์แรงดัน ดังนั้น

$$T_s = t_a + t_b + t_z \tag{2.14}$$

ถ้าพิจารณาว่า $|V_z| = 0$ จาก (2.13) จะได้

$$\mathbf{V} = V_a \frac{t_a}{T_s} + V_b \frac{t_b}{T_s} \tag{2.15}$$

จากนั้น ถ้าพิจารณารูปที่ 2.7 จะได้ว่า

$$\mathbf{V} = \mathbf{V}_X + \mathbf{V}_Y \tag{2.16}$$

และจากการประยุกต์ใช้ทฤษฎีตรีโกณมิติ จะได้

$$\mathbf{V}_X = \mathrm{V_m} \cos\alpha - \frac{1}{\sqrt{3}} \mathrm{V_m} \sin\alpha \tag{2.17}$$

$$\mathbf{V}_Y = \frac{2}{\sqrt{3}} \mathrm{V_m} \sin\alpha \tag{2.18}$$

และถ้าพิจารณาเฉพาะขนาดของสมการ (2.15) และ (2.16) จะได้

$$\mathbf{V}_X = |V_a| \frac{t_a}{T_s} \tag{2.19}$$

$$\mathbf{V}_Y = |V_b| \frac{t_b}{T_s} \tag{2.20}$$

หรือ

$$t_a = \frac{|\mathbf{V}_x|}{|V_a|} T_s \tag{2.21}$$

$$t_b = \frac{|\mathbf{V}_y|}{|V_b|} T_s \tag{2.22}$$

ถ้า V_a และ V_b คือเวคเตอร์หลักของแต่ละสถานะการสวิทชิ่ง ดังนั้น

$$|V_a| = |V_b| = \frac{2}{3} V_{dc} \tag{2.23}$$

ดังนั้น ถ้าแทนสมการ (2.17), (2.18) และ (2.23) ลงในสมการ (2.21) กับ (2.22) ตามลำดับ จะได้

$$t_a = \frac{3}{4}M(\cos(\alpha) - \frac{1}{\sqrt{3}}\sin(\alpha))T_s \quad (2.24)$$

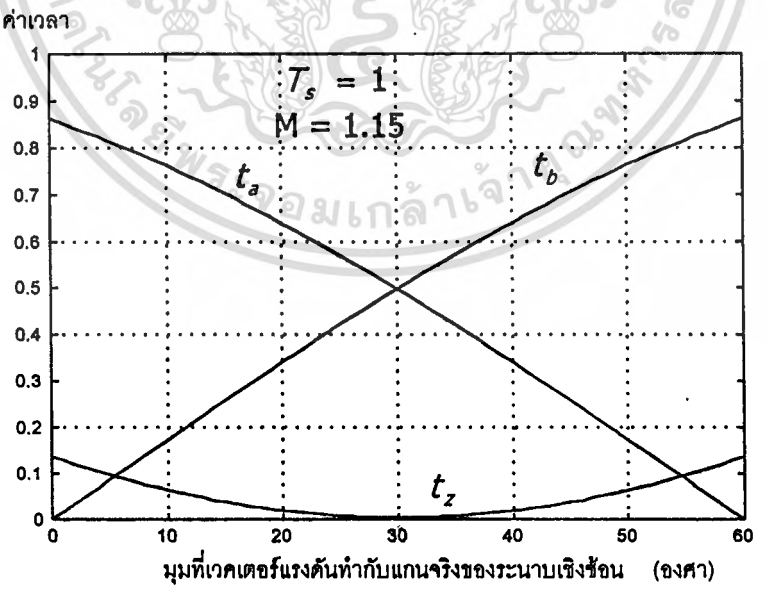
$$t_b = \frac{\sqrt{3}}{2}M\sin(\alpha)T_s \quad (2.25)$$

$$t_z = T_s - t_a - t_b \quad (2.26)$$

จากสมการที่ (2.24), (2.25) และ (2.26) พบว่าตัวแปรที่ใช้คำนวณค่าเวลา t_a, t_b และ t_z ของแต่ละเวคเตอร์แรงดันแบ่งได้เป็น

1. ค่าดัชนีการมอดูเลท (Modulation Index, M) เท่ากับ $\frac{V_m}{V_{dc}/2}$
2. ค่ามุมที่ **V** ทำกับแกนจริงของระนาบเชิงซ้อน (α) โดย $0 \le \alpha \le 60^\circ$
3. คาบเวลาแซมปลิ้ง T_s

เช่น ถ้ากำหนดค่าเวลาแซมปลิ้ง T_s = 1 และค่าดัชนีการมอดูเลท M = 1.15 เมื่อนำไปคำนวณ t_a, t_b และ t_z โดยใช้สมการที่ (2.24), (2.25) และ (2.26)และใช้ค่ามุม (α) เป็นตัวแปรอิสระจะได้ผลดังรูปที่ 2.8



รูปที่ 2.8 กราฟแสดงค่าเวลา t_a, t_b และ t_z เมื่อกำหนด T_s = 1 และ M = 1.15

2.6 องศาอิสระ(Degree of freedom) สำหรับการมอดูเลท

การสร้างเวคเตอร์อ้างอิง **V** ใดๆ จะประกอบด้วยเวคเตอร์หลัก 3 เวคเตอร์คือเวคเตอร์ข้างเคียงของแต่ละเซคเตอร์ 2 เวคเตอร์และเวคเตอร์ศูนย์อีก 1 เวคเตอร์และถ้าพิจารณาองศาอิสระ 3 ประการคือ

1.การเลือกใช้เวคเตอร์ศูนย์ว่าจะเลือก $\mathbf{U}_0$ หรือ $\mathbf{U}_7$ ในการสร้างเวคเตอร์แรงดันอ้างอิงแต่ละครั้ง

2. ลำดับในการสร้างเวคเตอร์ข้างเคียง

3. การแบ่งครึ่งค่าเวลา t_a, t_b และ t_z ของเวคเตอร์อ้างอิงแต่ละเวคเตอร์

จะทำให้สามารถสร้างรูปแบบการมอดูเลท (Modulation scheme) ที่แตกต่างกันได้หลายรูปแบบคือ

1. Symmetric sequence
2. Alternating zero vector sequence
3. Bus clamped

2.7 รูปแบบการมอดูเลท (Modulation scheme)

1. Symmetric sequence รูปที่ 2.9 แสดงรูปสัญญาณในแต่ละเซคเตอร์ของการมอดูเลท พบว่าในแต่ละคาบเวลาของการสุ่ม(Sampling period, T_s) จะเริ่มต้นและลงท้ายด้วย $\mathbf{U}_0$ เสมอและการสร้างเวคเตอร์ในแต่ละเซคเตอร์จะมีลำดับที่แน่นอน โดยจะมีการแบ่งครึ่งค่าเวลา t_a, t_b และ t_z เพื่อให้ได้สัญญาณที่มีความสมมาตร ซึ่งจาก [1] พบว่าความสมมาตรของสัญญาณในการสุ่มแต่ละครั้งจะมีผลให้ Total Harmonic Distortion (THD) ของสัญญาณมีค่าต่ำ นอกจากนี้ในแต่ละคาบเวลาของการสุ่มจะประกอบด้วยลำดับของการสร้างเวคเตอร์จำนวน 7 ลำดับ ดังนั้นวงจรที่สร้างโดยใช้วิธีการนี้จึงมีความซับซ้อนมากกว่าวิธีอื่นที่มีลำดับของการสร้างเวคเตอร์น้อยกว่า อีกทั้งยังมีการสวิทชิ่ง 6 ครั้งต่อการสุ่ม ดังนั้นความสูญเสียจากการสวิทชิ่งจึงสูงกว่าวิธีอื่นด้วย

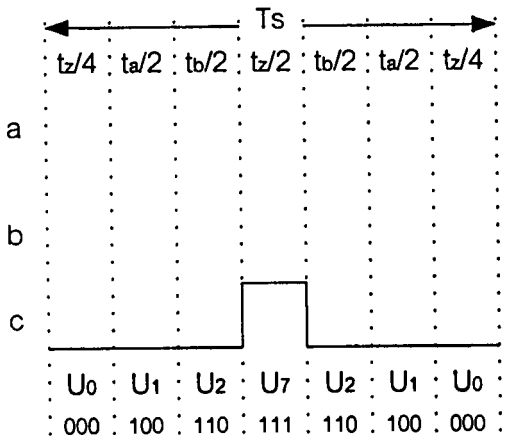
2. Alternating zero vector sequence การสร้างเวคเตอร์อ้างอิงในแต่ละแซมเปิ้ล จะใช้ $\mathbf{U}_0$ สลับกับ $\mathbf{U}_7$ ดังนั้นคาบเวลาการสวิตชิ่ง(Switching period, T_{sw}) จะเป็นครึ่งหนึ่งของคาบเวลาของการสุ่ม ทำให้สัญญาณที่สร้างมีความละเอียดในการสุ่มสูง แต่มีความถี่ในการสวิตชิ่งต่ำ (Hi sampling Low switching) ได้ โดยสัญญาณในการสุ่มแต่ละครั้งจะมีการสวิตชิ่ง 3 ครั้ง มีผลให้ความสูญเสียจากการสวิตชิ่ง (Switching loss) น้อยกว่าวิธีการมอดูเลทแบบแรกประมาณ 50 % แต่จะปรากฏฮาร์โมนิคที่ความถี่ในการสวิตชิ่งขึ้น ดังนั้นค่า THD ก็จะสูงกว่าวิธีแรก รูปสัญญาณในแต่ละเซคเตอร์แสดงดังรูปที่ 2.10

3. Bus clamped เนื่องจากจำนวนครั้งในการสวิตชิ่งจะมีผลโดยตรงต่อความสูญเสียที่เกิดขึ้นในอุปกรณ์สวิตชิ่ง ดังนั้นถ้าลดจำนวนครั้งของการสวิตชิ่งลง ก็จะลดความสูญเสียจากการสวิตชิ่งลงได้ และพบว่าเวคเตอร์ข้างเคียงในแต่ละเซคเตอร์จะมีสถานะการสวิตชิ่งที่เปลี่ยนไปเพียงสถานะเดียว ดัง

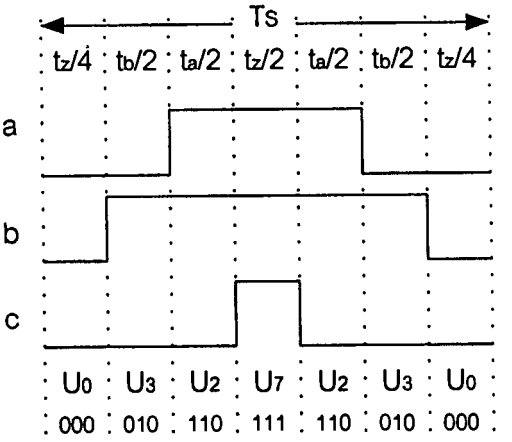
นั้นถ้าเลือกใช้ U_0 หรือ U_7 อย่างเหมาะสม ก็จะทำให้สัญญาณในบางแขนแนลไม่เกิดการสวิตชิ่งตลอดช่วงเซคเตอร์นั้น มีผลให้สามารถลดจำนวนครั้งในการสวิตชิ่งลงได้ รูปสัญญาณในแต่ละเซคเตอร์แสดงดังรูปที่ 2.11

2.8 มอดูเลติ้งฟังก์ชั่นของสเปซเวคเตอร์

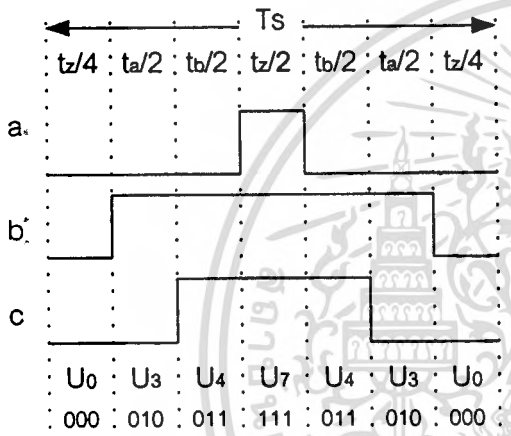
เมื่อนำค่า t_a, t_b และ t_z ที่คำนวณได้จากสมการที่ (2.24) - (2.26) ไปสร้างเป็นสัญญาณ PWM ด้วยรูปแบบการมอดูเลทต่างๆ ที่กล่าวมา แล้วนำค่าความกว้างของสัญญาณ (Pulse width) ในการสุ่มแต่ละครั้งไปพล็อตเป็นกราฟเทียบกับมุม จะได้กราฟดังรูปที่ 2.12, 2.13 และ 2.14 ซึ่งรูปสัญญาณตามกราฟนี้ จะใช้แทนมอดูเลติ้งฟังก์ชั่น (Modulating function) ของสัญญาณ โดยรูป (ก) แทนสัญญาณในแต่ละเฟสเทียบกับกราวด์ ส่วนรูป (ข) เป็นสัญญาณแบบเฟสต่อเฟส โดยจะเห็นได้ว่ามอดูเลติ้งฟังก์ชั่นของวิธี Symmetric sequence และ Alternating zero vector sequence จะไม่ใช่รูปซายน์นูซอยที่สมบูรณ์ แต่จะเป็นสัญญาณที่มีช่วงกลางของลูกคลื่นแบนลง ซึ่งมีผลมาจากองค์ประกอบของฮาร์โมนิคอันดับที่ 3 มีผลให้ค่าแรงดันเอาต์พุตสูงกว่าวิธี Sinusoidal PWM (SPWM) อยู่ 15 เปอร์เซ็นต์ [2] ที่ค่ามอดูเลชั่นอินเด็กเดียวกัน ส่วนมอดูเลติ้งฟังก์ชั่นของวิธี Bus clamped นั้นจะมีบางช่วงของสัญญาณที่ไม่เกิดการสวิตชิ่ง พิจารณาจากสัญญาณในบางช่วงจะเป็น 0 หรือ 1 ซึ่งเป็นข้อดีของวิธีการนี้ คือการที่มีช่วงเวลาการหยุดสวิตช์ใน 1 รอบการทำงานทำให้ความสูญเสียในการสวิตช์ลดลง แต่อย่างไรก็ตามมอดูเลติ้งฟังก์ชั่นของสัญญาณเมื่อวัดแบบเฟสต่อเฟสจะเป็นรูปซายน์นูซอยที่สมบูรณ์ เนื่องจากฮาร์โมนิคที่ไม่ต้องการจะหักล้างกันหมดไป แสดงดังรูปที่ 2.12 (ข), 2.13 (ข) และ 2.14 (ข)



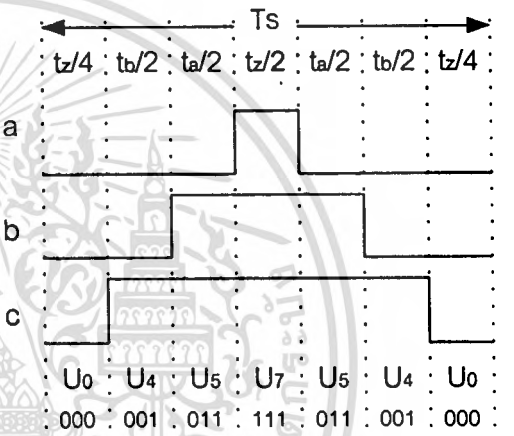
รูปสัญญาณในเซคเตอร์ 1



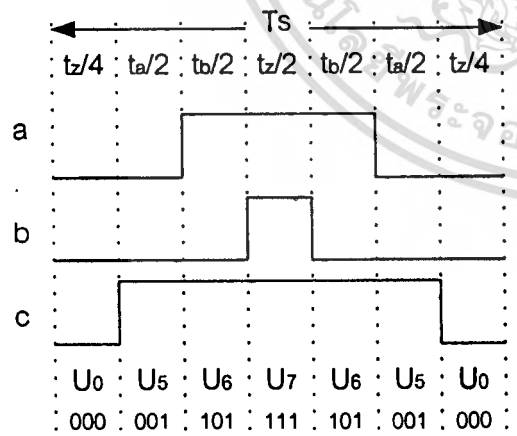
รูปสัญญาณในเซคเตอร์ 2



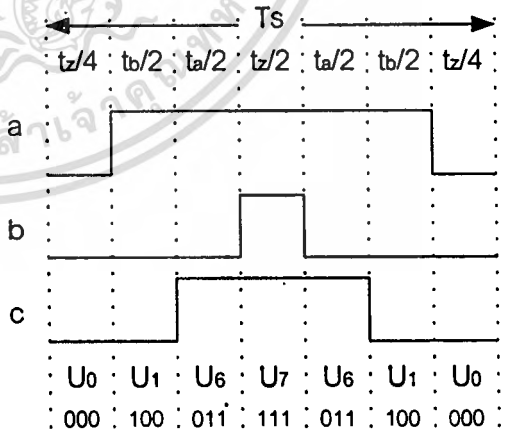
รูปสัญญาณในเซคเตอร์ 3



รูปสัญญาณในเซคเตอร์ 4

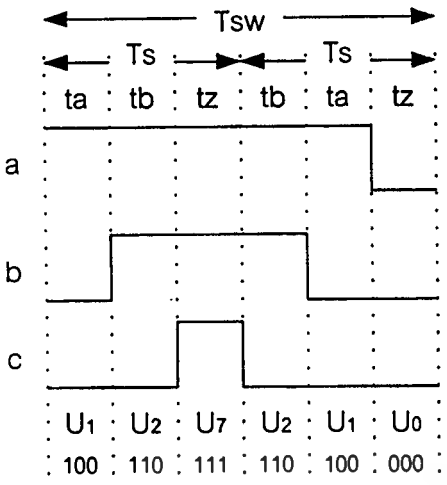


รูปสัญญาณในเซคเตอร์ 5

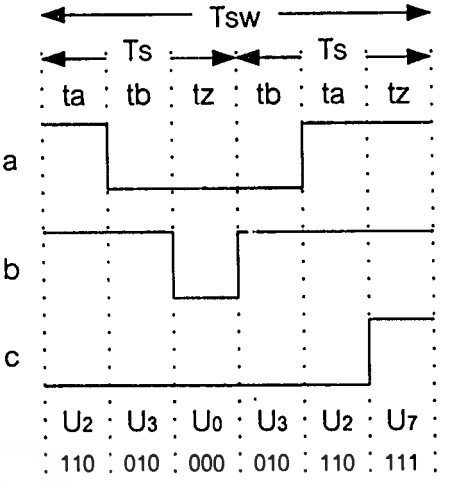


รูปสัญญาณในเซคเตอร์ 6

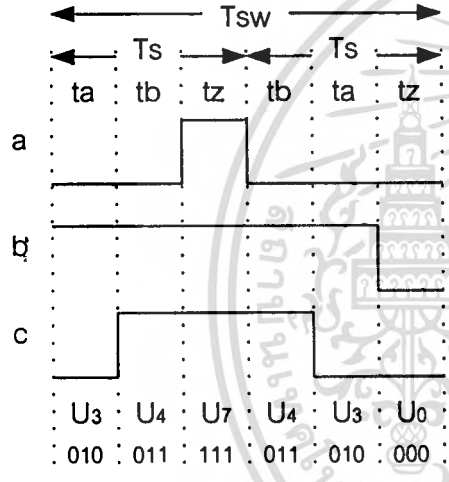
รูปที่ 2.9 รูปสัญญาณ PWM ที่สร้างโดยวิธี Symmetric sequence



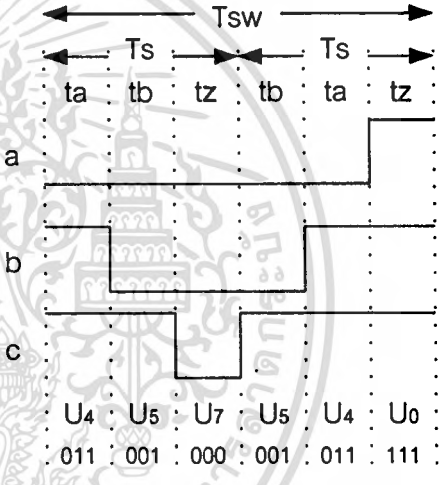
รูปสัญญาณในเซคเตอร์ 1



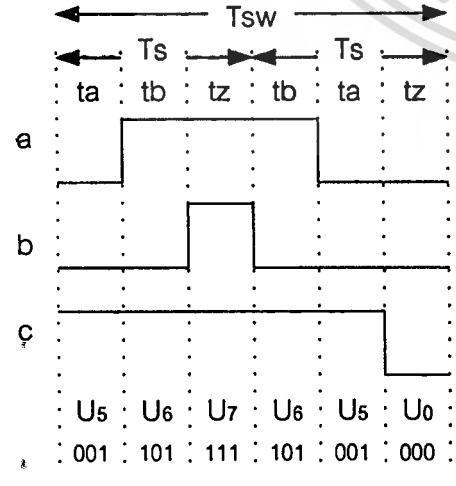
รูปสัญญาณในเซคเตอร์ 2



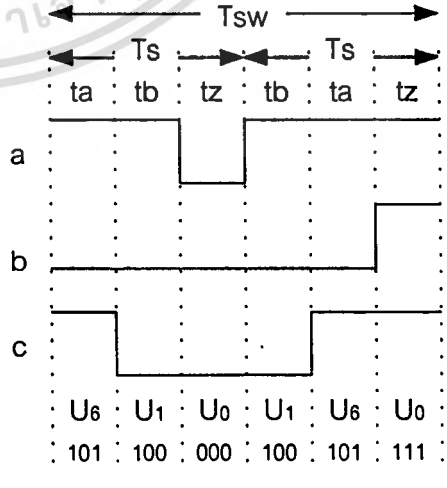
รูปสัญญาณในเซคเตอร์ 3



รูปสัญญาณในเซคเตอร์ 4

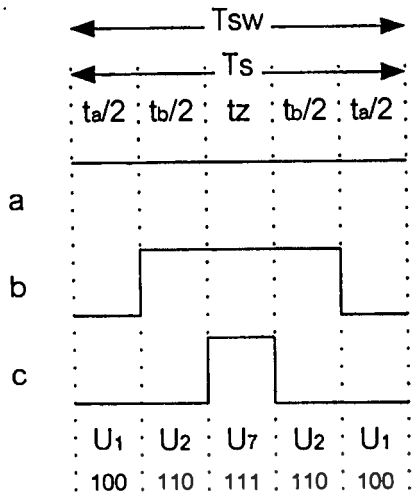


รูปสัญญาณในเซคเตอร์ 5

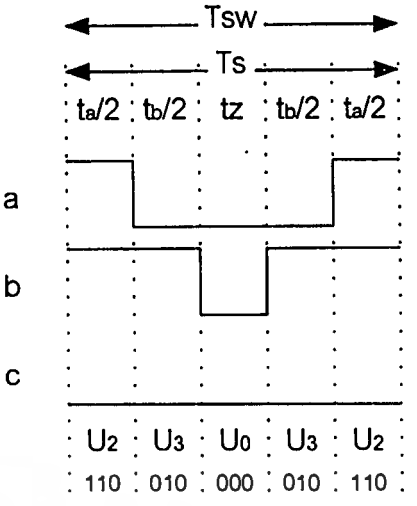


รูปสัญญาณในเซคเตอร์ 6

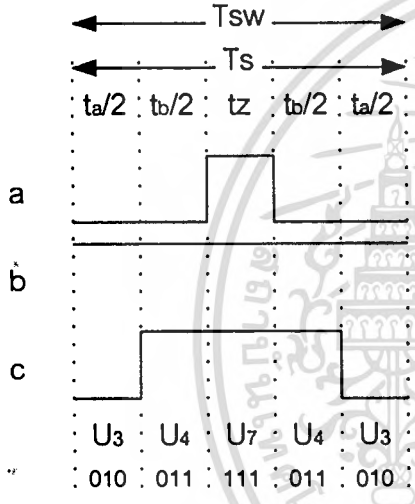
รูปที่ 2.10 รูปสัญญาณ PWM ที่สร้างโดยวิธี Alternating zero vector sequence



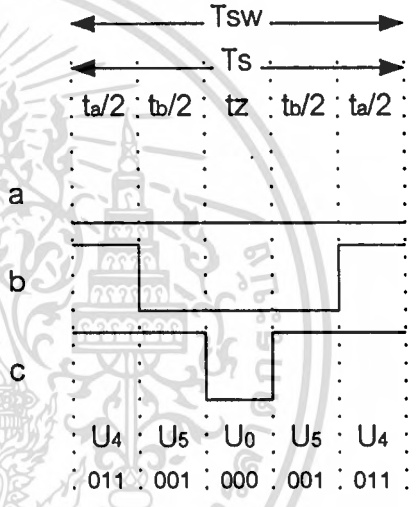
รูปสัญญาณในเซคเตอร์ 1



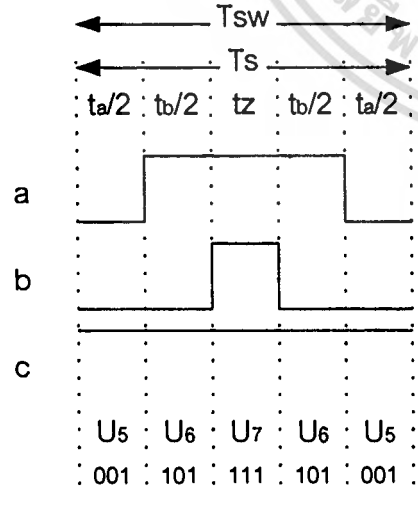
รูปสัญญาณในเซคเตอร์ 2



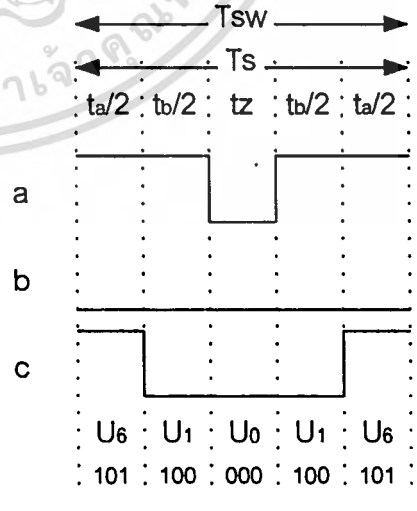
รูปสัญญาณในเซคเตอร์ 3



รูปสัญญาณในเซคเตอร์ 4

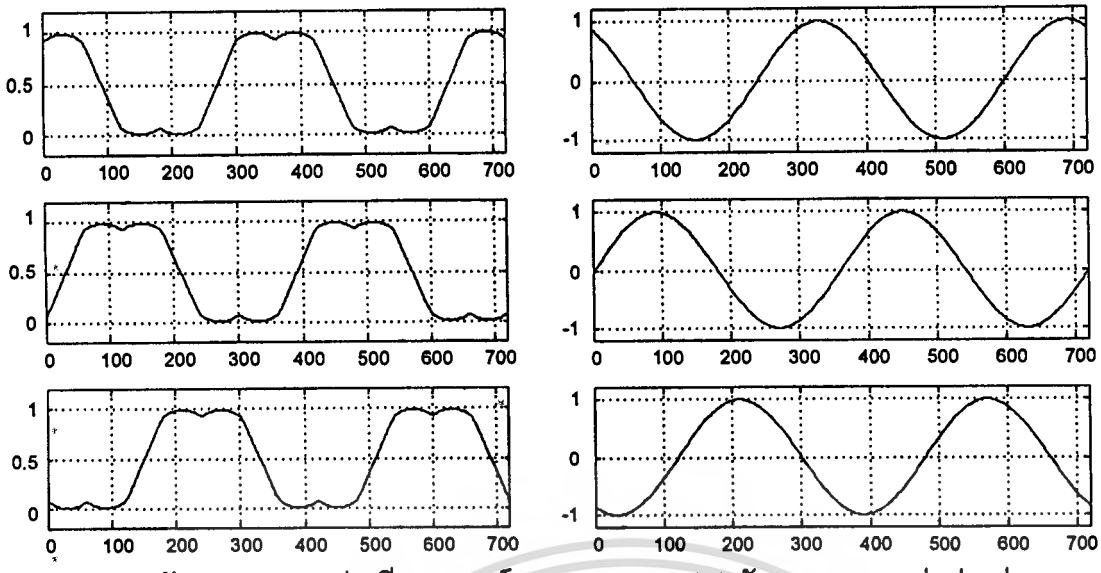


รูปสัญญาณในเซคเตอร์ 5



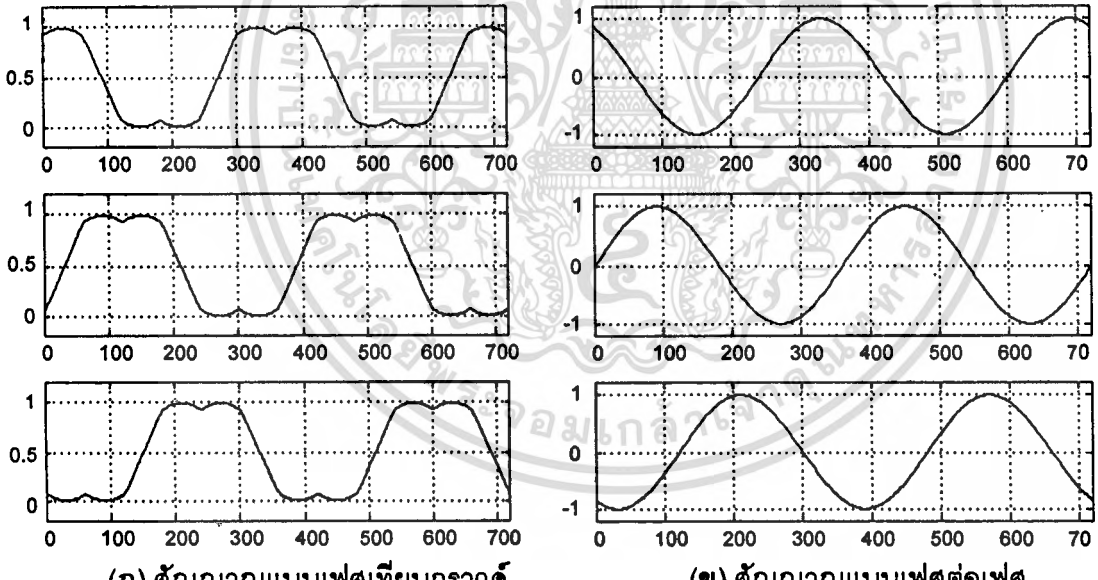
รูปสัญญาณในเซคเตอร์ 6

รูปที่ 2.11 รูปสัญญาณ PWM ที่สร้างโดยวิธี Bus clamped



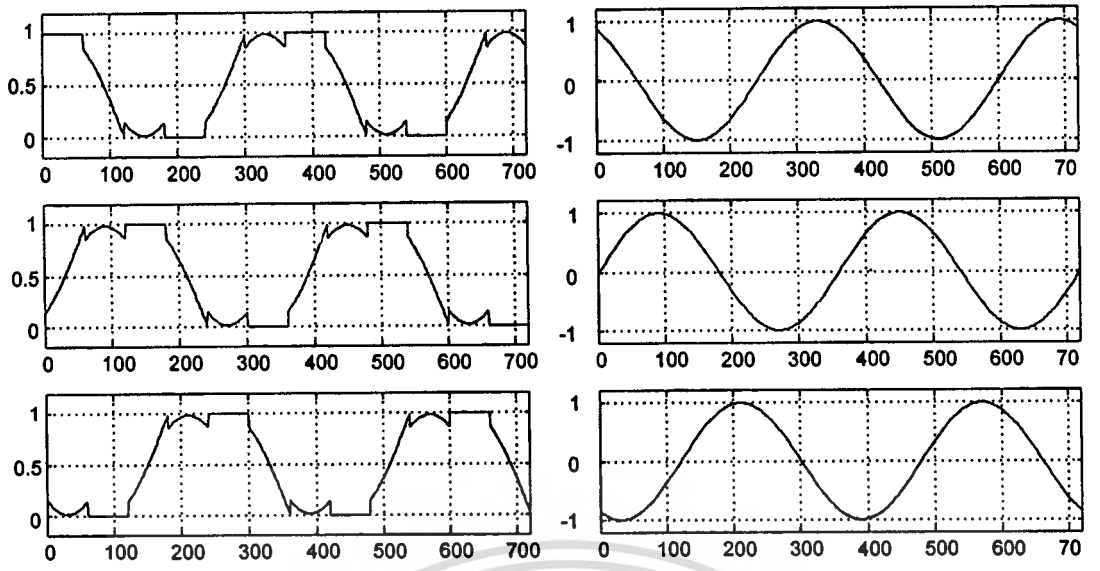
(ก) สัญญาณแบบเฟสเทียบกราวด์ (ข) สัญญาณแบบเฟสต่อเฟส

รูปที่ 2.12 มอดูเลติ้งฟังก์ชั่นของวิธี Symmetrical sequence



(ก) สัญญาณแบบเฟสเทียบกราวด์ (ข) สัญญาณแบบเฟสต่อเฟส

รูปที่ 2.13 มอดูเลติ้งฟังก์ชั่นของวิธี Alternating zero vector sequence



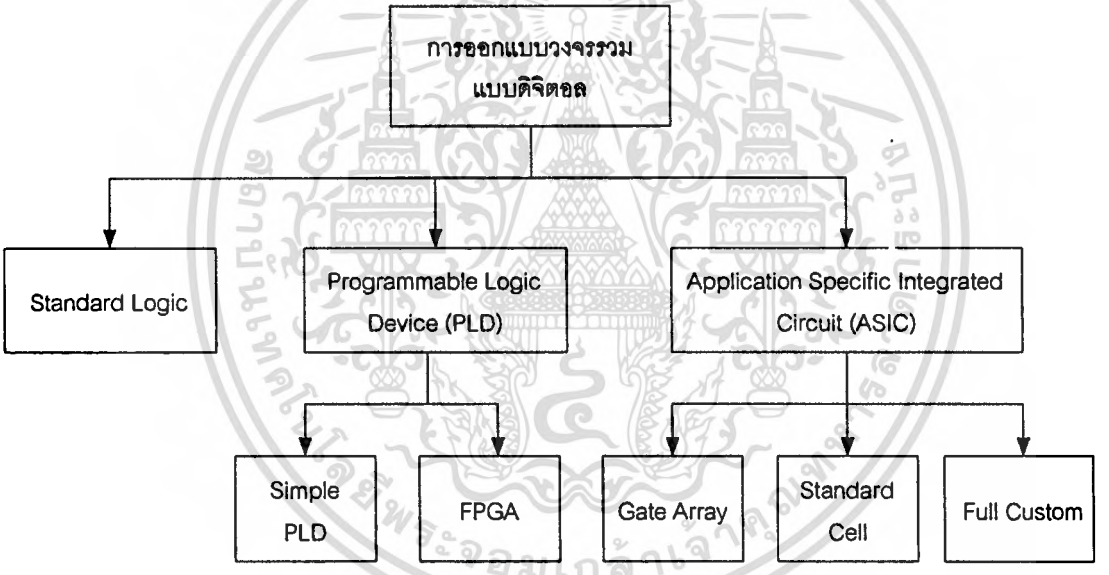
(ก) สัญญาณแบบเฟสเทียบกราวด์ (ข) สัญญาณแบบเฟสต่อเฟส

รูปที่ 2.14 มอดูเลติ้งฟังก์ชั่นของวิธี Bus clamped

บทที่ 3
เทคโนโลยีการออกแบบวงจรรวมดิจิตอล

3.1 เทคโนโลยีของวงจรรวมแบบดิจิตอล

จากความก้าวหน้าของอุตสาหกรรมอิเลคทรอนิกส์ในปัจจุบัน ทำให้เกิดการพัฒนาความสามารถของอุปกรณ์ดิจิตอลขึ้นอย่างมากมาย มีผลให้สามารถลดค่าใช้จ่าย การสิ้นเปลืองพลังงานและขนาดของวงจรที่ต้องการออกแบบลงได้และในขณะเดียวกันก็สามารถเพิ่มประสิทธิภาพและระดับความน่าเชื่อถือของวงจรให้สูงขึ้นอย่างเห็นได้ชัด ซึ่งถ้าพิจารณาจากแนวทางในการออกแบบ วงจรดิจิตอลแล้วจะสามารถจำแนกเทคโนโลยีในการออกแบบได้ 3 ประเภท ดังแผนภาพในรูปที่ 3.1 คือ



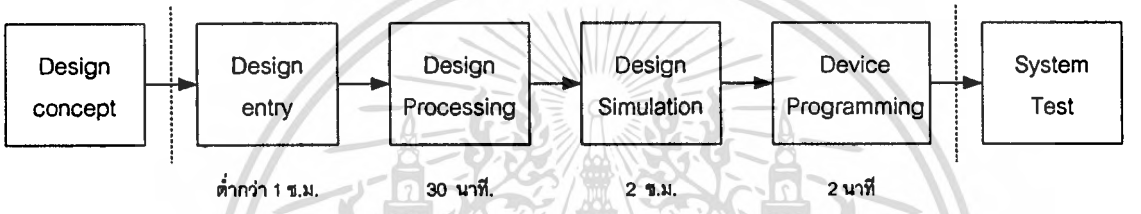
รูปที่ 3.1 แผนผังของแนวทางการออกแบบวงจรดิจิตอล

3.1.1 Standard Logic

เป็นการออกแบบวงจรลอจิกโดยใช้วงจรรวมแบบมาตรฐาน เช่น วงจรรวมแบบ TTL ในตระกูล 74xx หรือแบบ CMOS ในตระกูล 40xx มาประกอบกันเพื่อสร้างเป็นวงจรที่ต้องการ ซึ่งวิธีการนี้เป็นวิธีที่ใช้กันมาในอดีต ผลที่ได้คือวงจรโดยรวมจะมีขนาดใหญ่ ทำให้สิ้นเปลืองพลังงานและมีราคาแพง นอกจากนี้ความน่าเชื่อถือของวงจรยังมีค่าต่ำอีกด้วย

3.1.2 Programmable Logic Device (PLD)

วงจรรวมประเภท PLD คือวงจรรวมที่ผู้ใช้สามารถออกแบบและสร้างวงจรที่ต้องการลงในตัวอุปกรณ์ได้เอง โดยไม่จำเป็นต้องส่งวงจรรวมไปผลิตที่โรงงาน ซึ่งในปัจจุบันมีวงจรรวมประเภท PLD อยู่หลายชนิด และบริษัทผู้ผลิตหลายรายได้หันมาพัฒนาวงจรรวม PLD กันมากขึ้น ส่งผลให้มีประสิทธิภาพดีขึ้นแต่ราคาลดลงกว่าในอดีตมาก ทำให้ใช้ต้นทุนในการออกแบบต่ำ จึงมีความเสี่ยงในการลงทุนน้อยเพราะไม่จำเป็นต้องผลิตเป็นจำนวนมากเหมือนกับวงจรรวมแบบ ASIC ซึ่งขั้นตอนการออกแบบวงจรตลอดจนการจำลองผลการออกแบบจะทำผ่านโปรแกรมที่ประมวลผลบนคอมพิวเตอร์ ทำให้ขั้นตอนการออกแบบวงจรจนกระทั่งสำเร็จเป็นต้นแบบ (Prototype) ใช้เวลาน้อยมาก แสดงได้ดังรูปที่ 3.2 จึงมีความยืดหยุ่นในการออกแบบสูง



รูปที่ 3.2 ขั้นตอนการออกแบบวงจรดิจิตอลโดยใช้ PLD

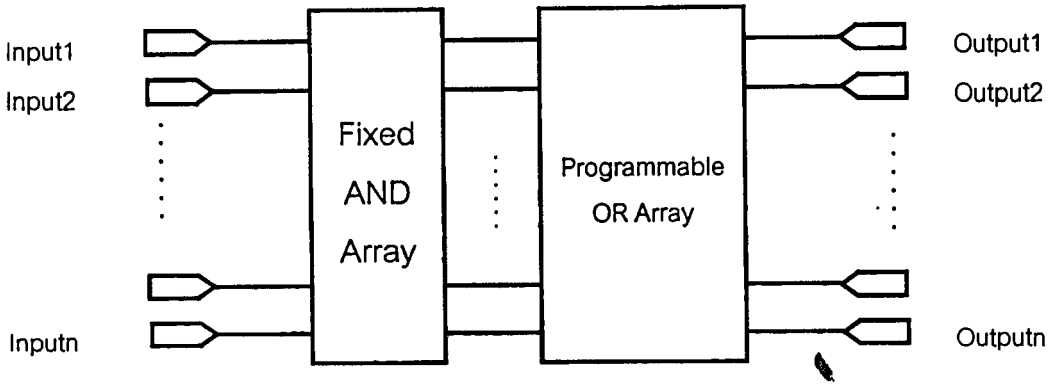
อุปกรณ์ PLD แบ่งออกได้เป็น 2 ประเภท

3.1.2.1 Simple PLD

คืออุปกรณ์ PLD ที่สร้างขึ้นในยุคแรก ๆ นับจากปี 1960 โดยมีจุดประสงค์เพื่อใช้แทนที่การออกแบบวงจรที่ใช้วงจรรวมของลอจิกเกทหลาย ๆ วงจร โดยสามารถออกแบบวงจรเหล่านั้นได้ โดยใช้วงจรรวมเพียงวงจรเดียว ทำให้ประหยัดเนื้อที่ของแผ่นวงจรและพลังงานที่ใช้ โดยอุปกรณ์ PLD โดยส่วนใหญ่จะประกอบด้วยแถวลำดับ(Array)ของแอนด์เกทและออร์เกทจำนวนมาก ซึ่งผู้ใช้สามารถกำหนดการเชื่อมต่อกันของแถวลำดับเหล่านี้ได้ โดยการตัดหรือต่อฟิวส์ (Fuse) ในแถวลำดับนั้น ๆ เพื่อสร้างเป็นวงจรที่ต้องการ ซึ่งถ้าพิจารณาจากสถาปัตยกรรมของการออกแบบวงจรรวมภายในอุปกรณ์ PLD แล้วสามารถแบ่งออกได้เป็น 4 ประเภท

3.1.2.1.1 Programmable Read-Only Memory (PROM)

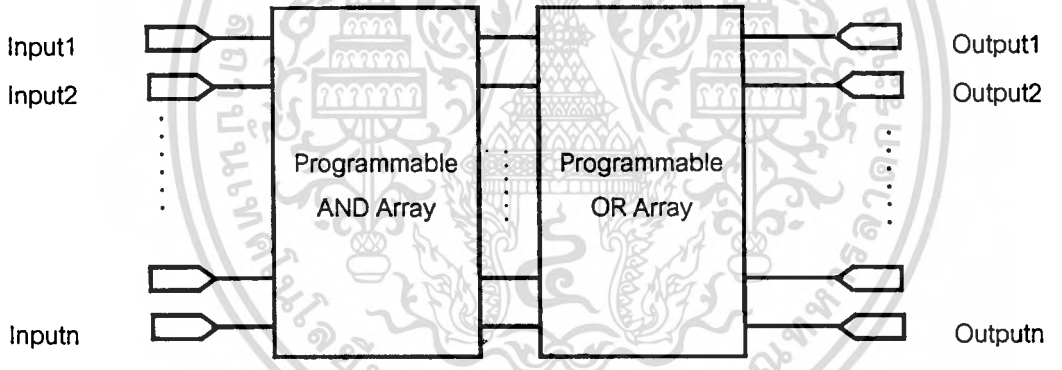
ซึ่ง PROM จะประกอบด้วยแถวลำดับของแอนด์เกทและออร์เกทจำนวนมาก โดยแถวลำดับของแอนด์เกทจะถูกต่อให้มีลักษณะเป็นวงจรถอดรหัส (Decoder) และแถวลำดับของออร์เกทที่ผู้ใช้สามารถกำหนดการเชื่อมต่อได้แสดงดังรูปที่ 3.3 โดยทั่วไปแล้วมักมีการใช้ PROM เพื่อทำหน้าที่เป็นหน่วยความจำ



รูปที่ 3.3 โครงสร้างของ PROM

3.1.2.1.2 Programable Logic Array (PLA)

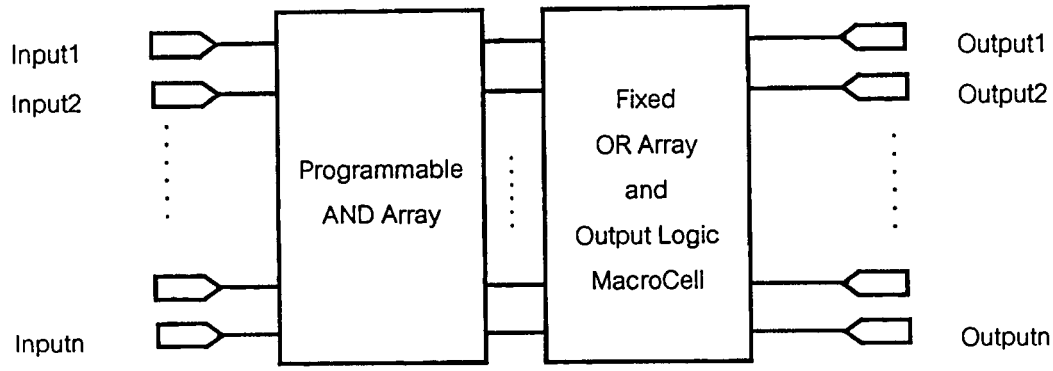
โดย PLA จะประกอบด้วยแถวลำดับของแอนด์เกทและออร์เกทจำนวนมากที่ผู้ใช้สามารถกำหนดการเชื่อมต่อกันของแถวลำดับได้ทั้งสองประเภท โดย PLA ถูกพัฒนาขึ้นเพื่อลดข้อจำกัดบางประการของ PROM โครงสร้างของ PLA แสดงดังรูปที่ 3.4



รูปที่ 3.4 โครงสร้างของ PLA

3.1.2.1.3 Programmable Array Logic (PAL)

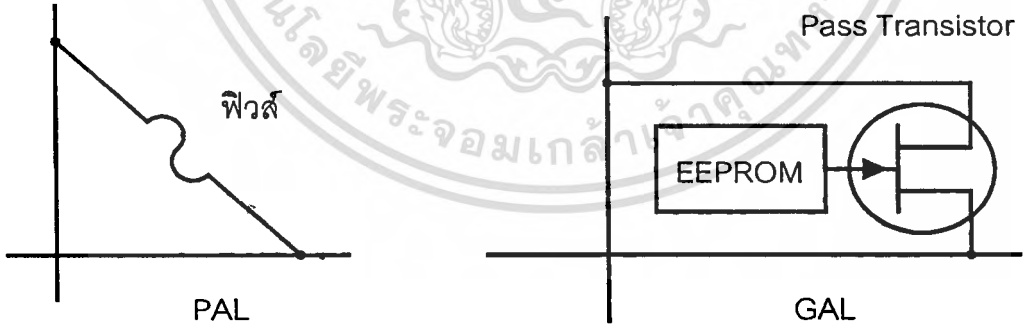
PAL ถูกพัฒนาขึ้นเพื่อลดข้อจำกัดของ PLA ในเรื่องของ Propagation Delay time ที่เกิดจากการเชื่อมต่อฟิวส์ในแถวลำดับแต่ละส่วน โดยโครงสร้างของ PAL จะประกอบด้วยแถวลำดับของแอนด์เกทที่ผู้ใช้สามารถกำหนดการเชื่อมต่อได้และแอร์เรย์ของออร์เกทที่ถูกกำหนดการเชื่อมต่อไว้แล้ว นอกจากนี้เพื่อให้ PAL สามารถใช้งานได้หลากหลายขึ้นได้มีการเพิ่มส่วนของเอาท์พุทลอจิก (Output Logic) ขึ้น ซึ่งผู้ใช้ต้องเลือกประเภทของเอาท์พุทลอจิกให้เหมาะสมกับวงจรที่ต้องการ โดยโครงสร้างของ PAL แสดงดังรูปที่ 3.5



รูปที่ 3.5 โครงสร้างของ PAL

3.1.2.1.4 Generic Array Logic (GAL)

อุปกรณ์ PLD ทั้ง 3 ประเภทที่กล่าวมาเป็นอุปกรณ์ PLD ที่สามารถโปรแกรมได้เพียงครั้งเดียว (One-time Programmable Device) เนื่องจากใช้ฟิวส์ในการตัดหรือต่อแถวลำดับต่าง ๆ ซึ่ง GAL ถูกพัฒนาขึ้นเพื่อลดข้อจำกัดดังกล่าวนี้ โดย GAL จะใช้สภาวะการนำกระแสหรือหยุดนำกระแสของทรานซิสเตอร์ในการควบคุมการเชื่อมต่อแถวลำดับซึ่งสภาวะดังกล่าวจะถูกควบคุมโดยค่าลอจิกที่เก็บใน EEPROM ดังรูปที่ 3.6 เป็นการเปรียบเทียบโครงสร้างของฟิวส์ที่ใช้ใน PAL และโครงสร้างที่ใช้ใน GAL มีผลให้ GAL มีคุณสมบัติในการโปรแกรมซ้ำได้ (Reprogrammable) นอกจากนี้ยังมีการปรับปรุงส่วนเอาท์พุทลอจิกให้สามารถใช้งานได้หลากหลายขึ้น โดยสามารถโปรแกรมให้เป็นวงจรคอมบิเนชั่น หรือวงจรซีเควนเชียลก็ได้

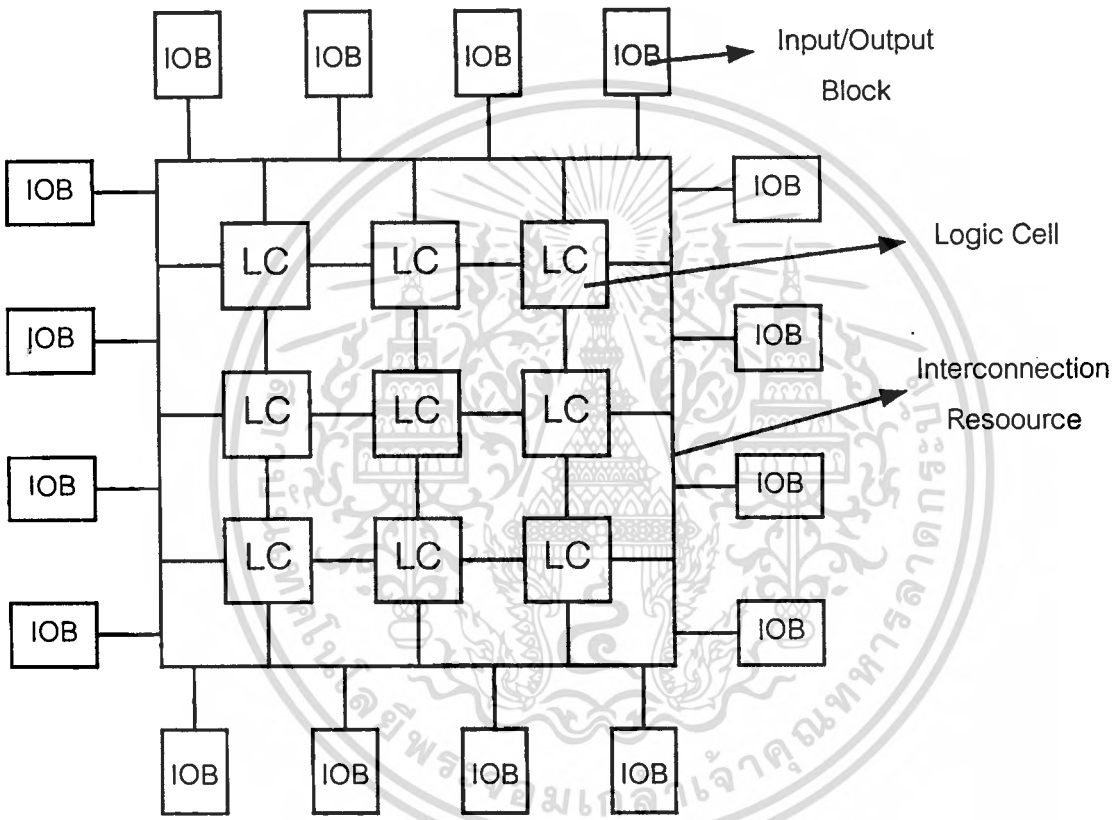


รูปที่ 3.6 การเปรียบเทียบโครงสร้างของฟิวส์ที่ใช้ใน PAL และที่ใช้ใน GAL

3.1.2.2 Field Programmable Gate Array (FPGA)

FPGA ถูกพัฒนาให้มีประสิทธิภาพการทำงานและปริมาณความหนาแน่นของเกทสูงกว่า Simple-PLD มาก โดยถูกพัฒนาให้สามารถทำงานที่ความถี่สูงได้มากกว่า 100 MHz และความหนาแน่นเกทสูงถึงหลายแสนเกทต่อชิป จึงนำไปออกแบบเป็นวงจรลอจิกที่ซับซ้อนได้ และ FPGA ใน

ปัจจุบัน ได้ถูกออกแบบให้มีคุณสมบัติในการโปรแกรมซ้ำได้ ผู้ใช้จึงเปลี่ยนแปลงหรือแก้ไขวงจรที่ออกแบบได้ง่าย ซึ่งสถาปัตยกรรมโดยทั่วไปของ FPGA แสดงดังรูป 3.7 ซึ่งประกอบด้วยลอจิกเซล (Logic Cell) เป็นจำนวนมากที่ถูกจัดวางในรูปเมทริกซ์ ซึ่งลอจิกเซลเหล่านี้จะถูกนำมาประกอบกันเพื่อสร้างเป็นวงจรลอจิกที่ใหญ่ขึ้น โดยการเชื่อมต่อกันของลอจิกเซลแต่ละเซลจะกระทำผ่าน Interconnection Resouce และการติดต่อกับภายนอกของลอจิกเซลจะกระทำผ่านส่วนอินพุท/เอาท์พุทบล็อค (Input/Output Block)



รูปที่ 3.7 โครงสร้างโดยทั่วไปของ FPGA

3.1.3 Application Specific Integrated Circuit (ASIC)

วงจรรวมประเภท ASIC มักถูกใช้งานในเชิงพาณิชย์ เนื่องจากต้นทุนการผลิตต่อหนึ่งตัวจะต่ำกว่าวงจรรวมประเภทอื่น ถ้ามีปริมาณการผลิตสูงนับหมื่นตัวขึ้นไป โดยการใช้งานวงจรรวม ASIC นั้น ผู้ใช้สามารถใช้โปรแกรมสำเร็จรูปออกแบบและจำลองผลการทำงานจนเป็นที่พอใจแล้ว จากนั้นต้องส่งไฟล์ต้นแบบไปให้บริษัทผู้ผลิตทำการเจือสารเพื่อสร้างเป็นวงจรรวม ซึ่งผู้ใช้ไม่สามารถโปรแกรมได้ด้วยตนเองเหมือนการใช้อุปกรณ์ PLD ดังนั้นช่วงเวลาการผลิตนับตั้งแต่ออกแบบวงจรจนกระทั่งได้ผลิต

ภัณฑ์ต้นแบบ จึงใช้เวลานานนับเดือนและมีค่าใช้จ่ายในการเจือสารสูง และถ้าพิจารณาจากสถาปัตยกรรมภายในของวงจรสามารถจำแนกวงจรรวม ASIC ได้เป็น 3 ประเภท

3.1.3.1 **เกตแอร์เรย์** (Gate Array)

โครงสร้างภายในจะประกอบด้วยแถวลำดับของวงจรเกตกับขั้วต่อสายไฟ (Pad) สำหรับต่อกับวงจรภายนอก ผู้ใช้มีหน้าที่ออกแบบการเชื่อมต่อระหว่างเกตและขั้วต่อสายไฟ เพื่อให้ได้วงจรที่ต้องการแล้วจึงส่งผลการออกแบบนั้นไปยังโรงงานผู้ผลิตวงจรรวมเกตแอร์เรย์ เพื่อทำการเจือสารต่อไป ซึ่งโดยทั่วไปแล้วจะใช้พื้นที่ของชิปประมาณ 25-30% สำหรับสร้างวงจรเกต ส่วนพื้นที่ที่เหลือจะใช้สมการเชื่อมโยงวงจรเกตเข้าด้วยกัน

3.1.3.2 **เซลล์มาตรฐาน** (Standard Cell)

ปัญหาการใช้พื้นที่ซิลิคอนอย่างมากมายสำหรับการเชื่อมโยงวงจรของวงจรรวมแบบเกตอาเรย์ทำให้เกิดข้อจำกัดในความซับซ้อนของวงจรที่ใช้ ประกอบกับผู้ใช้จำนวนมากต้องการรวบรวมวงจรรวมมาตราฐาน อาทิ วงจรตระกูล 7400 วงจรตระกูล 4000 จำนวนหลายตัวเข้าเป็นวงจรรวม ASIC เพียงตัวเดียว ทำให้เกิดวงจรรวม ASIC แบบเซลล์มาตรฐานขึ้น วงจรรวม เซลล์มาตรฐานนี้ ผู้ใช้เป็นผู้เลือกกลุ่มวงจรทำหน้าที่ต่าง ๆ เช่น เกต ฟลิบฟลอป ตัวนับ ตัวเลื่อน หน่วยความจำ หรือกระทั่งไมโครโปรเซสเซอร์จากแฟ้มข้อมูล (Library) ของผู้ผลิต ซึ่งอาจจะเป็นแฟ้มข้อมูลคอมพิวเตอร์ เหมาะสมสำหรับวงจรเชิงเลขที่มีความซับซ้อน การผลิตวงจรรวม เซลล์มาตรฐานจะมีต้นสูงกว่าวงจรรวมเกตอาเรย์และใช้เวลาในการออกแบบและเจือสารยาวกว่าสองถึงสามเท่าตัว วงจรรวมเซลล์มาตรฐานจึงเหมาะสมกับการใช้งานเชิงพาณิชย์ที่มีปริมาณการผลิตนับหมื่นตัวขึ้นไป บทบาทวงจรรวมเซลล์มาตรฐานจะมีเพิ่มมากขึ้นในอนาคต เมื่อต้นทุนการเจือสารลดลง

3.1.3.3 **ฟูลัสตัม** (Full Custom)

วงจรรวมฟูลัสตัมนี้ผู้ใช้เป็นผู้ออกแบบเองทั้งหมด ตั้งแต่ระดับวงจรเชิงเลขจนกระทั่งถึงระดับกายภาพ แล้วจึงส่งข้อมูลการออกแบบไปให้ผู้ผลิตเจือสารในรูปแฟ้มข้อมูลมาตรฐาน เช่น GDS II, CIF การใช้ฟูลัสตัม ASIC นี้เท่าที่แพร่หลายอยู่ในปัจจุบัน จะเป็นไปเพื่อการศึกษาและการวิจัยและเพื่อการผลิตจำนวนน้อย ส่วนใหญ่จะเป็นการเจือสารในลักษณะที่ใช้ค่าใช้จ่ายร่วมกัน กล่าวคือรวบรวมการออกแบบหลายวงจรลงบนแผ่นซิลิคอนเดียวกันเพื่อประหยัดค่าเจือสาร

3.2 ขั้นตอนในการออกแบบวงจรลอจิกโดยใช้ PLD

ขั้นตอนการออกแบบวงจรลอจิกโดยใช้ PLD มีรายละเอียดดังนี้

1. System Specification and Analysis เป็นขั้นตอนของการสร้างข้อกำหนด (Specification) และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการของการแก้ปัญหา

2. Modeling and Simulation เป็นการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษาที่เรียกว่า Hardware Description Language (HDL) ซึ่งเป็นภาษาที่สามารถอธิบายการทำงานของวงจรดิจิตอลได้โดยผู้ใช้เพียงแต่อธิบายพฤติกรรมของวงจรเท่านั้น โดยไม่จำเป็นต้องลงลึกไปในรายละเอียดของการต่อกันของเกตต่าง ๆ จากนั้นจึงนำไปคอมไพล์และจำลองการทำงานของระบบเพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3. Logic and Test Synthesis ในขั้นตอนนี้ระบบช่วยการออกแบบ (Design Tool) จะสังเคราะห์ (Synthesis) วงจรที่ได้จากรูปแบบที่เขียนขึ้น ให้สามารถนำไปโปรแกรมลงบนอุปกรณ์ PLD ที่เลือกใช้ จากนั้นจึงทำการวิเคราะห์เพื่อหาค่าเวลาหน่วงสูงสุดที่จะเกิดขึ้น

4. Programming and Testing เมื่อผลที่ได้ถูกต้องตามข้อกำหนดแล้ว จึงนำไปโปรแกรมลงบนชิป PLD ที่ต้องการ จากนั้นจึงทดสอบสัญญาณจริงที่ได้กับข้อกำหนดที่ต้องการ

3.3 ข้อได้เปรียบของการออกแบบวงจรลอจิกโดยใช้ PLD

1. ใช้เวลาน้อยในการออกแบบจนได้เป็นชิปต้นแบบ
2. ต้นทุนในการออกแบบต่ำกว่า ASIC มาก
3. มีความเสี่ยงในการลงทุนน้อยเนื่องจากไม่จำเป็นต้องผลิตเป็นจำนวนมากและสามารถผลิตเพิ่มได้ตามต้องการ
4. เปลี่ยนแปลงและแก้ไขวงจรที่ออกแบบในภายหลังได้ง่าย และใช้ต้นทุนต่ำ โดยเฉพาะอย่างยิ่งอุปกรณ์ประเภท FPGA จะมีคุณสมบัติ In-System Programming (ISP) หมายถึงผู้ใช้แก้ไขวงจรได้โดยไม่ต้องถอดชิปออกจากแผ่นวงจร เพียงแต่ป้อนข้อมูลใหม่ที่ต้องการเปลี่ยนแปลงลงไปที่ขาสัญญาณที่กำหนดไว้เท่านั้น

3.4 ทางเลือกในการใช้งานอุปกรณ์ PLD

1. ใช้ PLD ในงานที่ต้องการผลิตไม่มากนัก
2. ใช้ PLD ในช่วงเริ่มต้นผลิตสินค้า ซึ่งผู้ผลิตอาจจะยังไม่แน่ใจในปริมาณตลาด อีกทั้งวงจรที่ออกแบบอาจยังมีจุดบกพร่องที่ต้องแก้ไขและปรับปรุง

3. เมื่อต้องการเปลี่ยนรูปแบบจากผลิตจากการใช้ PLD เป็นการใช้ ASIC ในเชิงพาณิชย์แล้ว จะทำได้ง่าย ถ้าเป็นการออกแบบด้วยภาษาที่เป็นมาตรฐาน เช่น VHDL หรือ Verilog เป็นต้น
4. ในปัจจุบันด้วยเทคโนโลยีที่สูงขึ้น มีผลให้ PLD มีราคาถูกลงมาก ซึ่งสินค้าหลายอย่างถึงแม้ต้องผลิตเป็นจำนวนมาก ก็ยังสามารถผลิตโดยใช้ PLD ได้ในต้นทุนที่คุ้มค่า

3.5 เทคโนโลยีการโปรแกรม PLD

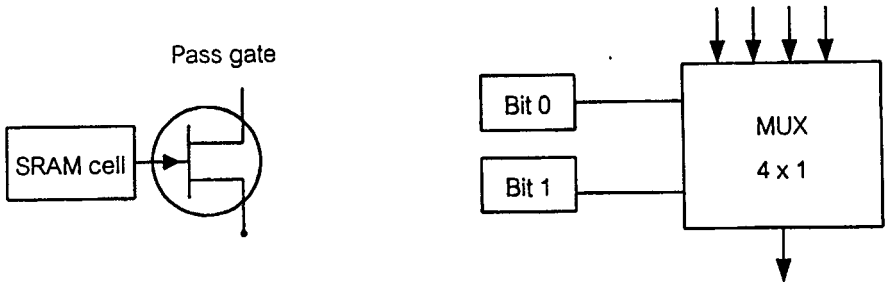
3.5.1 Programming Technology

การโปรแกรมอุปกรณ์ PLD โดยส่วนมากจะใช้รูปแบบของสวิทช์ที่ถูกโปรแกรมได้ทางไฟฟ้า (Electrically Programmable Switch) ซึ่งคุณสมบัติของสวิทช์นี้ เช่น ขนาด คุณสมบัติในการรักษาข้อมูล (Volatile) เทคโนโลยีในการสร้าง และค่าความต้านทานหรือค่าตัวเก็บประจุแฝงของ PLD ในแต่ละบริษัทก็จะแตกต่างกันไป เทคโนโลยีของการโปรแกรม PLD ที่นิยมกันในปัจจุบันจะมี 3 ประเภท คือ

1. SRAM Programming Technology
2. Floating Gate Programming Technology
3. Anti-fuse Programming Technology

3.5.1.1 SRAM programming Technology

รูปแบบนี้จะใช้เซลของ SRAM ในการกำหนดลักษณะ (Configuration) และการเชื่อมต่อ (Interconnect) ของวงจร โดยการกำหนดลักษณะจะกระทำโดยการควบคุมผ่าน Pass gate หรือมัลติเพล็กเซอร์ ดังแสดงในรูป 3.8 ซึ่งเมื่อ "1" ถูกเก็บไว้ใน SRAM แล้ว Pass Transistor จะอยู่ในสภาวะลัดวงจร ทำให้เกิดการเชื่อมต่อกันที่เทอร์มินอลทั้งสองข้าง และสำหรับมัลติเพล็กเซอร์แล้ว สถานะของ SRAM จะควบคุมการเชื่อมต่อกันของเอาท์พุทกับอินพุทตัวใดตัวหนึ่ง โดยผู้ใช้สามารถโปรแกรมการเชื่อมต่อกันของอุปกรณ์ซ้ำได้ตามต้องการโดยไม่จำกัดจำนวนครั้ง ทำให้เกิดความยืดหยุ่นในการใช้งานมากขึ้น แต่ข้อเสียของวิธีการโปรแกรมแบบ SRAM ก็คือข้อมูลที่ถูกโปรแกรมจะสูญหายไปถ้าหยุดจ่ายพลังงานให้กับวงจรรวม ดังนั้นจึงต้องต่อหน่วยความจำภายนอกเพื่อใช้ในการโปรแกรม โดยการโปรแกรมจะกระทำอย่างอัตโนมัติในขณะเริ่มจ่ายพลังงานให้กับวงจรรวม ดังนั้นบนตัววงจรรวมจึงต้องมีวงจรพิเศษที่ทำหน้าที่จัดการเกี่ยวกับการโปรแกรมตัวชิปด้วย



รูปที่ 3.8 โครงสร้างของเซลที่ใช้ในการโปรแกรม

3.5.1.2 Floating Gate Programming Technology

วิธีนี้ใช้เทคโนโลยีของการสร้าง EPROM หรือ EEPROM มาเป็นส่วนควบคุม Pass Transistor โดยมีข้อดีคือเป็นวงจรรวมที่สามารถโปรแกรมซ้ำได้เหมือนกับ SRAM FPGA แต่ไม่จำเป็นต้องต่อหน่วยความจำภายนอกไว้ แต่การโปรแกรมซ้ำของ Floating Gate ไม่สามารถทำได้รวดเร็วเหมือนกับ SRAM ซึ่งขั้นตอนการโปรแกรมจะเหมือนกับการใช้งาน EPROM หรือ EEPROM

3.5.1.3 Antifuse Programming Technology

Antifuse จะมีคุณสมบัติตรงข้ามกับ Fuse คือ ความต้านทานจะเปลี่ยนจากค่าสูงไปเป็นค่าต่ำเมื่อถูกโปรแกรมด้วยแรงดันค่าสูง และเนื่องจาก antifuse มีค่าความต้านทานค่อนข้างต่ำประมาณ 100-600 โอห์ม และมีขนาดเล็ก ทำให้ขนาดของวงจรรวมค่อนข้างเล็กกว่า 2 วิธีแรก การโปรแกรม Antifuse ต้องใช้วงจรพิเศษที่สามารถจ่ายแรงดันสูง (ประมาณ 10-20V) เพื่อใช้ในการโปรแกรม

ตารางที่ 3.1 เปรียบเทียบคุณสมบัติของวิธีการโปรแกรมแต่ละแบบ

Technology and Process	Volatile	Reprogramability	Area	Ron (ohm)	C (fF)	Extra step
SRAM 1.2 μm	yes	Yes in circuit	large	0.9-2x	10-20	0
Anti-fuse 1.2 μm	no	no	small	300-600	1-5	3
EPROM 1.2 μm	no	Yes, out of circuit	small	50-100	10-20	3
EEPROM 1.2 μm	no	Yes, in circuit	2x EPROM	2-4x	10-20	5

3.6 รายละเอียดของ FPGA ในตระกูล FLEX10K

เนื่องจากในวิทยานิพนธ์ฉบับนี้ได้เลือกใช้วงจรรวม FPGA ของบริษัท Altera ในตระกูล FLEX (Flexible Logic Element matrix) คือ FLEX10K เพื่อนำมาใช้กำเนิดสัญญาณ PWM ดังนั้นในหัวข้อนี้จะกล่าวถึงสถาปัตยกรรมของวงจรรวมในตระกูลนี้

3.6.1 คุณสมบัติโดยทั่วไป

- มีความจุของเกตสูงคือตั้งแต่ 10,000-250,000 เกท
- ประกอบด้วยส่วน Embeded Array Block (EAB) ซึ่งสามารถประยุกต์ใช้เป็น RAM, ROM หรือวงจรลอจิกอื่น ๆ ได้ โดยแต่ละบล็อคจะประกอบด้วยหน่วยความจำ 2048 บิต
- สามารถโปรแกรมให้ทำงานที่ระดับแรงดันขนาด 5 โวลท์ และ 3.3 โวลท์ ได้
- ใช้กำลังงานต่ำ โดยปกติแล้วในสภาวะ Stand by จะกินกระแสต่ำกว่า 0.5 mA
- ใช้เทคโนโลยี SRAM ในการโปรแกรม จึงไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ โดยถูกโปรแกรมได้ใน 2 ลักษณะคือ โปรแกรมจาก ROM ภายนอกที่ใช้เก็บโปรแกรม และจะโหลดโปรแกรมเข้าในตัว FLEX เมื่อเริ่มต้นจ่ายไฟให้กับตัววงจรรวมอย่างอัตโนมัติ หรือถูกโปรแกรมจากตัวโปรแกรมเฉพาะของทางบริษัท เช่น Byte Blaster download cable เป็นต้น
- ในการเชื่อมต่อภายใน (Interconnection) จะออกแบบโดยใช้ Fast track Intercon nection ทำให้การส่งผ่านสัญญาณในส่วนต่าง ๆ เป็นไปอย่างรวดเร็วและสามารถคาดเดาค่าหน่วงเวลา (delay time) รวมได้
- ในแต่ละ Logic Element (LE) จะประกอบด้วย carry chain ซึ่งจะมีประโยชน์มากในการสร้างวงจรบวกเลข วงจรนับหรือวงจรเปรียบเทียบความเร็วสูง และประกอบด้วย cascade chain ซึ่งจะถูกใช้ในการขยายจำนวนอินพุทของวงจรลอจิก ซึ่งการใช้งานส่วนนี้จะถูกกำหนดโดยอัตโนมัติจากโปรแกรมช่วยการออกแบบ (Software tool)
- สามารถโปรแกรมอัตราสลูว์ (Slew Rate) ของส่วนเอาท์พุท เพื่อลดสัญญาณรบกวนจากการสวิทชิ่งได้
- มีขาสัญญาณอินพุทเอาท์พุทมากถึง 400 ขา
- ใช้กับงานที่ต้องการความเร็วสูง เช่น ถ้านำไปสร้างเป็นวงจรนับขนาด 16 บิท จะนับสัญญาณนาฬิกา (Clock Signal) ได้สูงสุดถึง 200 MHz
- รองรับมาตรฐาน IEEE1149.1-1990 ในการทำ boundary scan test
- รองรับคุณสมบัติ In System Programming (ISP) คือสามารถโปรแกรมตัววงจรรวมได้โดยไม่ต้องถอดวงจรรวมออกจากแผ่นวงจรต้นแบบ

- มีโปรแกรมช่วยการออกแบบ(Developement Tool) ได้แก่ Max + Plus II ที่ทำงานบนคอมพิวเตอร์ส่วนบุคคล ซึ่งจะทำหน้าที่จัดวางและเชื่อมโยงอุปกรณ์ภายในต่าง ๆ ได้อย่างอัตโนมัติ (Automatic place and Routing) และยังสามารถคอมไพล์ภาษา Hardware Deseription Language เช่น VHDL หรือ AHDL ได้

3.6.2 รายละเอียดโดยทั่วไป

FLEX10K เป็นอุปกรณ์ FPGA จากบริษัท Altera โดยใช้เทคโนโลยีการโปรแกรมแบบ CMOS SRAM จึงสามารถโปรแกรมซ้ำได้ไม่จำกัดจำนวนครั้ง โดยชิปในตระกูลนี้จะมีความจุของเกตตั้งแต่ 10,000 – 250,000 เกต และมีจำนวนขาอินพุทเอาท์พุทตั้งแต่ 60-470 ขา จึงสามารถนำไปออกแบบเป็นวงจรลอจิกที่ซับซ้อนต่าง ๆ ได้มากมายดังเช่น ตารางที่ 3.2 แสดงพฤติกรรมของ FLEX10K เมื่อนำไปออกแบบเป็นวงจรต่าง ๆ

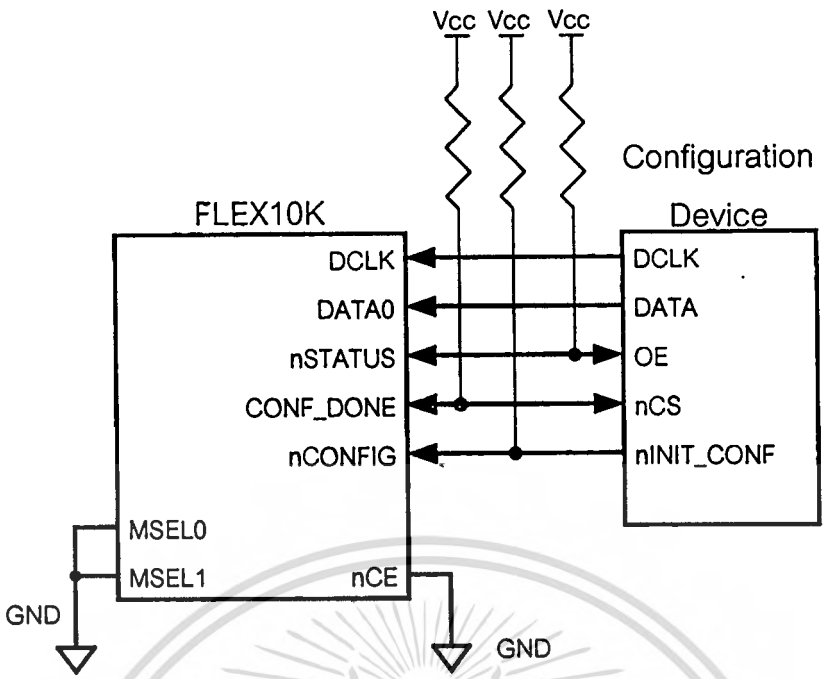
ตารางที่ 3.2 แสดงพฤติกรรมของ FLEX10K เมื่อนำไปออกแบบเป็นวงจรต่าง ๆ

Application	Resource used		Performance Speed Grade				Unit
	LE	EAB	-1	-2	-3	-4	
16 bit loadable counter	16	0	204	166	125	95	MHz
16 bit Accumu lator	16	0	204	166	125	95	MHz
16 to 1 Multiplexer	10	0	4.2	5.8	6.0	7.0	nS
256x8 RAM read cycle	0	1	172	145	108	84	MHz
256x8 RAM write cycle	0	1	106	89	68	63	MHz

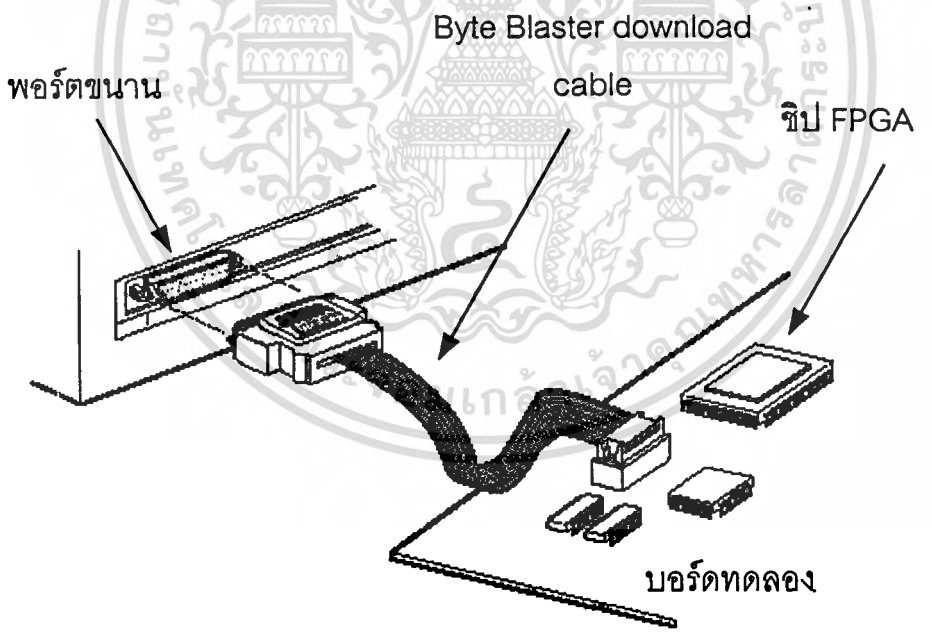
การโปรแกรม FLEX10K จะกระทำได้ใน 2 ลักษณะคือ

1. โปรแกรมโดยใช้หน่วยความจำภายนอก (Configuration Device) ในขณะเริ่มจ่ายพลังงานให้วงจร ซึ่งอุปกรณ์เหล่านี้จะส่งข้อมูลแบบอนุกรมให้กับ FLEX10K โดยอัตโนมัติ ดังรูปที่ 3.9

2. โปรแกรมจากคอมพิวเตอร์ผ่านทางอุปกรณ์เฉพาะของ Altera เช่น BitBlaster down load cable จะมีรูปแบบการโปรแกรมแบบอนุกรม หรือ ByteBlaseter download cable จะมีรูปแบบการโปรแกรมแบบขนาน เป็นต้น ดังรูปที่ 3.10



รูปที่ 3.9 การเชื่อมต่อ FLEX10K กับ Configuration device



รูปที่ 3.10 การโปรแกรม FLEX10K ผ่านทาง Byte Blaster download cable

นอกจากนี้อุปกรณ์ FLEX **ทุกตัวยังสนับสนุนการใช้งานในรูปแบบ** In-System Programming **คือ สามารถโปรแกรมได้ ขณะที่วงจรรวมอยู่บนบอร์ดทดลอง โดยไม่จำเป็นต้องถอดวงจรรวมออกมาเข้าเครื่องโปรแกรม จึงมีความยืดหยุ่นในการใช้งานสูง**

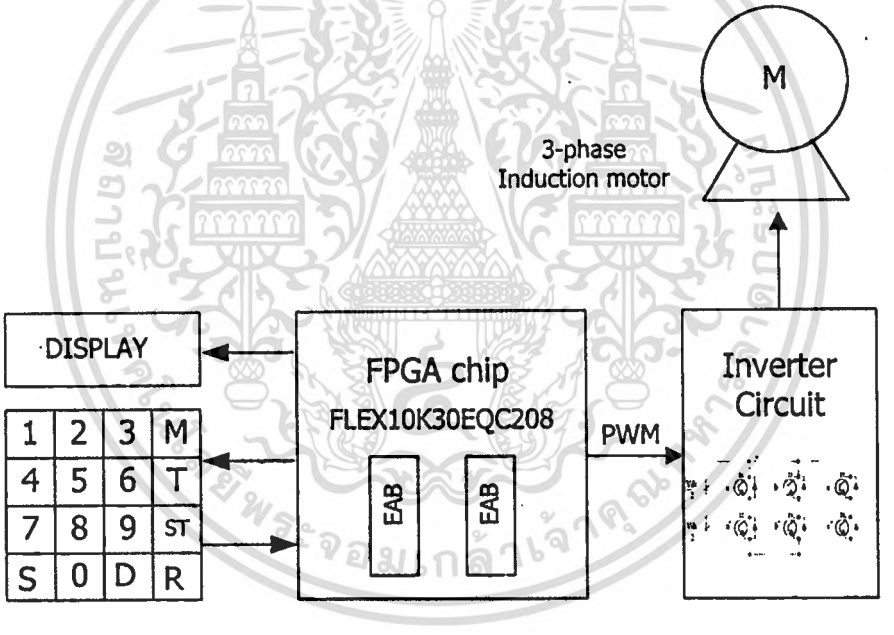
บทที่ 4
การออกแบบ

4.1 บทนำ

ในบทนี้จะกล่าวถึงการใช้งานวงจรรวม การทำงานของระบบโดยรวม การออกแบบองค์ประกอบย่อยแต่ละส่วนของวงจรรวม และวงจรอินเวอร์เตอร์ที่ใช้ในการทดสอบการทำงานของวงจรรวมที่ได้ออกแบบไว้

4.2 การใช้งานวงจรรวม

การใช้งานวงจรรวมที่ออกแบบไว้แสดงดังรูปที่ 4.1



รูปที่ 4.1 การใช้งานวงจรรวม FPGA ที่ออกแบบไว้

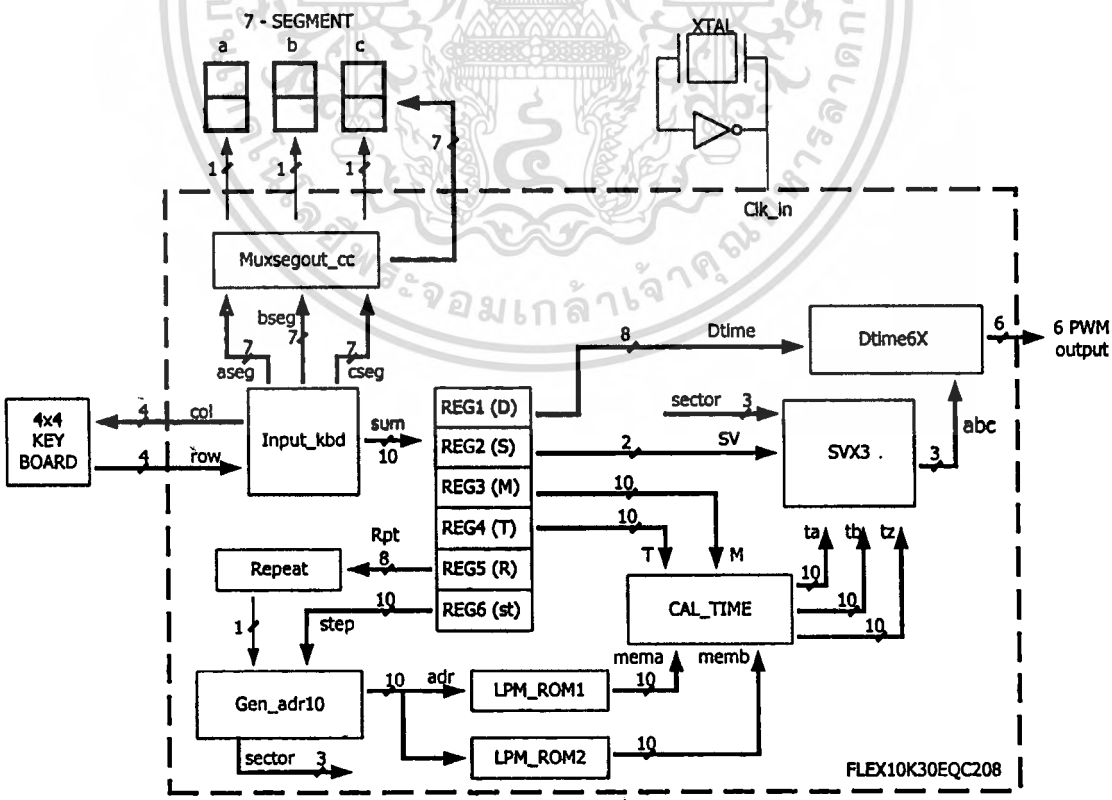
โดยตัวแปรอินพุทที่ใช้ในการกำหนดคุณลักษณะของสัญญาณ PWM ทั้ง 6 ตัวแปรคือ ค่าดัชนีการมอดูเลท(M), คาบเวลาการสุ่ม(T), ความถี่หลักมูล(St), รูปแบบการมอดูเลท(S), จำนวนการสร้างเวคเตอร์ซ้ำ(R) และค่าเดดไทม์(D) จะได้มาจากการป้อนค่าผ่านคีย์บอร์ดแบบเมตริกซ์ขนาด 4 x 4 โดยวงจรรวมจะทำการสแกนคีย์บอร์ด และรับค่าตัวแปรอินพุทต่างๆ เข้ามาเก็บไว้ในรีจิสเตอร์ภายใน และแสดงผลค่าของตัวแปรอินพุทที่รับเข้ามานั้น ออกทาง 7 เซกเมนต์ขนาด 3 หลัก จากนั้นวงจรรวมจะนำค่าตัวแปรอินพุทที่ได้ไปคำนวณเพื่อสร้างเป็นสัญญาณ PWM และส่ง

ไปควบคุมการทำงานของชุดอินเวอร์เตอร์ต่อไป ซึ่งระบบโดยรวมจะเป็นระบบควบคุมแบบลูปเปิด (Open loop system) โดยข้อกำหนดเฉพาะตัวของวงจรกำเนิดสัญญาณ PWM ที่สร้างขึ้นจะเป็นดังนี้

1. ปรับเปลี่ยนค่าดัชนีการมอดูเลทได้ 1000 ค่า ตั้งแต่ 0 -1.15
2. ปรับเปลี่ยนค่าคาบเวลาการสุ่มได้ 1000 ค่า โดยคาบเวลาการสุ่มนี้จะสัมพันธ์กับสัญญาณนาฬิกาที่ใช้เป็นฐานเวลาของระบบ
3. ปรับเปลี่ยนความถี่หลักมูลได้ 512 * 1000 ค่า
4. ปรับเปลี่ยนเดดไทม์ได้ 256 ค่า
5. ปรับเปลี่ยนสัญญาณนาฬิกาอ้างอิงของระบบได้ในช่วงกว้าง
6. ปรับเปลี่ยนรูปแบบการสร้างสัญญาณ PWM ได้ 3 รูปแบบ คือ Alternating zero vector sequence , Symmetric sequence และ Bus clamped

4.3 การทำงานของวงจรโดยรวม

รูปที่ 4.2 คือบล็อคไดอะแกรมแสดงการทำงานของวงจรโดยรวม



รูปที่ 4.2 บล็อคไดอะแกรมของวงจรโดยรวม

โดยส่วน Input_kbd จะทำหน้าที่สแกนคีย์บอร์ดแบบเมตริกซ์ขนาด 4 x 4 ผ่านทางขาสัญญาณ col และ row เพื่อรับค่าตัวแปรอินพุทจากผู้ใช้ โดยผู้ใช้ต้องใส่ตัวเลขจำนวนเต็มในฐานสิบจำนวน 3 หลักและตามด้วยชนิดของตัวแปรที่ต้องการ จากนั้นส่วน Input_kbd จะประมวลผลเพื่อเปลี่ยนค่าในฐานสิบของตัวเลขทั้ง 3 หลักให้เป็นค่าในฐานสองขนาด 10 บิต แล้วส่งไปเก็บยังรีจิสเตอร์ REG1-REG6 ตามชนิดของตัวแปรที่เลือกไว้ เพื่อนำค่าที่เก็บในตัวแปรต่างๆ นั้นไปประมวลผลต่อไป และในขณะเดียวกันก็จะแสดงผลค่าของตัวแปรที่กำหนดนั้นออกทาง 7-Segment ขนาด 3 หลักด้วย ซึ่งค่าในรีจีสเตอร์ทั้ง 6 ตัวนี้จะถูกส่งไปยังส่วนต่าง ๆ เพื่อใช้ในการประมวลผล โดย

- ค่าใน REG1(D) จะถูกส่งให้กับส่วนควบคุมการสร้างเดดไทม์ (Dtime6X) เพื่อสร้างการหน่วงเวลาของสัญญาณ PWM ตามค่าเดดไทม์ที่ผู้ใช้ต้องการ

- ค่าใน REG2(S) มีได้ 3 ค่าคือ 1-3 จะใช้ค่าในรีจีสเตอร์ตัวนี้ในการกำหนดรูปแบบของการกำเนิดสัญญาณ PWM ซึ่งสามารถเลือกได้ 3 รูปแบบคือ

  - ถ้า REG2 = 1 คือเลือกใช้วิธี Alternating zero vector sequence
  - ถ้า REG2 = 2 คือเลือกใช้วิธี Symmetric sequence
  - ถ้า REG2 = 3 คือเลือกใช้วิธี Bus clamped

- ค่าใน REG3(M) และ REG4(T) จะถูกส่งให้กับส่วน CAL_TIME เพื่อใช้คำนวณช่วงเวลา ta, tb และ tz ของแต่ละเวคเตอร์ร่วมกับค่าคงที่ที่เก็บอยู่ใน LPM_ROM1 และ LPM_ROM2

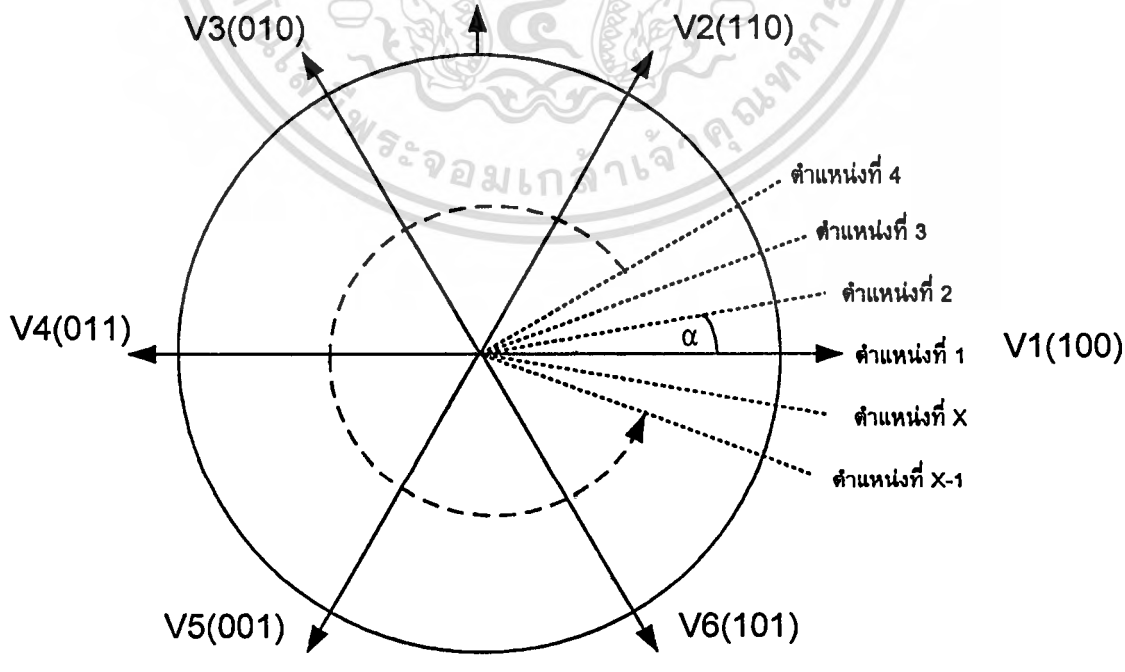
- ค่าใน REG5(R) จะถูกส่งให้กับส่วน Repeat เพื่อกำหนดจำนวนครั้งของการสร้างเวคเตอร์แรงดันซ้ำที่ตำแหน่งเดิม จะใช้ในกรณีที่ต้องการให้ความถี่มูลฐานของสัญญาณ PWM มีค่าต่ำมากๆ

- ค่าใน REG6(St) จะถูกส่งให้กับส่วน Gen_adr10 เพื่อใช้ในการกำหนดตำแหน่งหน่วยความจำ(Address) ของ LPM_ROM1 และ LPM_ROM2 ซึ่งทำหน้าที่เก็บข้อมูลคงที่เพื่อใช้ในการคำนวณเวลา ta, tb และ tz โดยค่าตัวแปรที่เก็บใน REG6 ทำหน้าที่กำหนดระยะการกระโดดข้ามตำแหน่งหน่วยความจำของ LPM_ROM1 และ LPM_ROM2 ทำให้สามารถเปลี่ยนความถี่หลักมูลของสัญญาณ PWM ที่ต้องการได้ (สำหรับสมการของการคำนวณค่าความถี่หลักมูลจะกล่าวถึงในหัวข้อต่อไป) โดย LPM_ROM1 จะเก็บข้อมูลคงที่ขนาด 10 บิตของสมการ $\frac{3}{4}(\cos\alpha - \frac{1}{\sqrt{3}}\sin\alpha)$ และ LPM_ROM2 เก็บข้อมูลของสมการ $\frac{\sqrt{3}}{2}\sin\alpha$ ซึ่งเก็บเฉพาะค่า α ในช่วง $0 \le \alpha \le 60^{\circ}$ เท่านั้นหรือเก็บค่าเพียงเซกเตอร์เดียว โดย LPM_ROM แต่ละตัวจะถูกออกแบบให้มีขนาด 1024 แอดเดรส โดยแต่ละแอดเดรสเก็บข้อมูลขนาด 10 บิต ดังนั้นความละเอียดสูงสุดของมุม α จึงเท่ากับ 60 / 1024 = 0.0585 องศา จากนั้นข้อมูลทั้ง 4 ตัวคือ M(จาก REG3), T(จาก REG4),

mema(จาก LPM_ROM1) และ memb(จาก LPM_ROM2) จะถูกส่งไปยัง CAL_TIME เพื่อใช้ในการคำนวณค่า ta, tb และ tz ตามสมการที่ 2.20-2.22 จากนั้นค่าเวลาทั้งสามตัวนี้จะถูกส่งไปยังส่วน SVX3 เพื่อนำไปกำหนดเป็นช่วงเวลาของการสร้างลำดับการสวิตชิ่ง ตามรูปแบบของการมอดูเลทแบบสเปซเวคเตอร์ที่ต้องการ โดยเอาท์พุตของ SVX3 จำนวน 3 เอาท์พุท (abc) จะถูกส่งให้กับส่วน Dtime6X เพื่อสร้างการหน่วงเวลาของสัญญาณที่จะส่งไปควบคุมทรานซิสเตอร์ที่อยู่ในกิ่งเดียวกันไม่ให้นำกระแสพร้อมกัน โดยค่าเวลาในการหน่วงนี้จะได้มาจาก REG1 ซึ่งเอาท์พุตของส่วน Dtime6X จำนวน 6 เอาท์พุตจะเป็นเอาท์พุทของวงจรรวม ที่จะนำไปควบคุมการทำงานของอินเวอร์เตอร์

4.4 การคำนวณความถี่หลักมูล

แนวคิดของการออกแบบนั้นจะแบ่งวงกลมในระนาบเชิงซ้อนออกเป็น X ส่วนเท่า ๆ กันดังรูปที่ 4.3 ดังนั้นความละเอียดของมุมในการสร้างเวคเตอร์แรงดันอ้างอิงจะเท่ากับ 360/X องศา ซึ่งตำแหน่งมุมต่างๆ เหล่านี้จะถูกใช้เป็นตัวแปร (α) ของสมการ $\frac{3}{4}(\cos\alpha - \frac{1}{\sqrt{3}}\sin\alpha)$ และ $\frac{\sqrt{3}}{2}\sin\alpha$ แล้วเก็บเป็นข้อมูลคงที่ลงในหน่วยความจำ LPM_ROM1 และ LPM_ROM2 เพื่อใช้ในการคำนวณค่า ta, tb และ tz ของสมการ 2.20 - 2.22 และเนื่องจากการสร้างเวคเตอร์แรงดันในแต่ละเซคเตอร์จะใช้วิธีการและข้อมูลคงที่ที่เหมือนกัน ดังนั้นข้อมูลที่เก็บลงหน่วยความจำจึงเก็บเพียงเซคเตอร์เดียวเท่านั้น



รูปที่ 4.3 การแบ่งวงกลมในระนาบเชิงซ้อนออกเป็น X ส่วนเท่าๆ กัน

โดยจะพิจารณาว่า คาบเวลาของความถี่หลักมูล (T_{fund}) จะเท่ากับเวลาที่เวคเตอร์แรงดันใช้ในการเคลื่อนที่ในระนาบเชิงซ้อนเป็นวงกลมครบ 1 รอบ ซึ่งถ้ากำหนดให้คาบเวลาการสุ่ม (T_s) ของการสร้างเวคเตอร์แต่ละเวคเตอร์มีค่าคงที่ ดังนั้นคาบเวลาของความถี่หลักมูลจะเท่ากับ

$$T_{fund} = nT_s \tag{4.1}$$

โดย n คือจำนวนครั้งในการสร้างเวคเตอร์แรงดันเพื่อให้เคลื่อนที่เป็นวงกลมครบ 1 รอบ

เนื่องจากในทางปฏิบัตินั้นมักจะกำหนดให้คาบเวลาการสุ่มและค่าความละเอียดของมุมมีค่าคงที่ ดังนั้นในการเปลี่ยนแปลงคาบเวลาของความถี่หลักมูล จึงทำได้โดยการเปลี่ยนแปลงค่า n ตามสมการที่ 4.1 ซึ่งค่า n นี้จะสัมพันธ์กับตัวแปรอินพุท St ตามสมการ 4.2

$$n = \frac{X}{St} \tag{4.2}$$

ซึ่ง St เป็นค่าจำนวนเต็มของตัวแปรที่เก็บใน REG6 โดยที่ $0 \le St \le \frac{X}{6}$

ซึ่งการออกแบบวงจรรวมของวิทยานิพนธ์ฉบับนี้ ได้เลือกใช้วงจรรวม FPGA ของบริษัท Altera ในตระกูล FLEX คือ FLEX10K30EQC208 ซึ่งโครงสร้างภายในจะเสมือนกับประกอบด้วยลอจิกเกทประมาณ 30,000 ตัว จึงสามารถประยุกต์สร้างเป็นวงจรลอจิกขนาดใหญ่ได้ นอกจากนี้ยังประกอบด้วยส่วน Embeded Array Block(EAB) จำนวน 6 บล็อค โดยแต่ละบล็อคจะประกอบด้วยหน่วยความจำ 4096 บิต ซึ่งสามารถประยุกต์ใช้งานเป็นหน่วยความจำภายในของวงจรรวมได้ ซึ่งถ้าพิจารณาจากขนาดความจุของส่วน EAB ที่อยู่ภายในตัว FLEX10K30EQC208 พบว่าขนาดที่เหมาะสมของ LPM_ROM1 และ LPM_ROM2 แต่ละตัวคือ 1024 x 10 บิต ซึ่งเกิดจากการนำ EAB มาประยุกต์ใช้จำนวน 5 บล็อค

ดังนั้นวงจรรวมที่ออกแบบไว้นี้ จะแบ่งวงกลมในระนาบเชิงซ้อน (X) ออกเป็น 1024 x 6 = 6144 ส่วน หรือความละเอียดของมุมจะเท่ากับ 360/6144 = 0.0585 องศา ซึ่งสมการหาคาบเวลาของความถี่หลักมูลจะได้

$$T_{fund} = \frac{6144}{St} T_s \tag{4.3}$$

รูปที่ 4.4 เป็นตัวอย่างการสร้างเวคเตอร์แรงดันโดยกำหนดให้ตัวแปร St = 1 ส่วนในรูปที่ 4.5 เป็นการกำหนดให้ตัวแปร St = 2 เช่น ถ้ากำหนดความถี่การสุ่มเท่ากับ 10kHz จะได้คาบเวลาการสุ่มเท่ากับ 100 μs ดังนั้น

ถ้า St = 1 ดังนั้น $T_{fund} = \frac{6144}{1} 100\ \mu s = 0.6144$ วินาที

จะได้ $F_{fund} = \frac{1}{T_{fund}} = \frac{1}{0.6144} = 1.62$ เฮิรตซ์

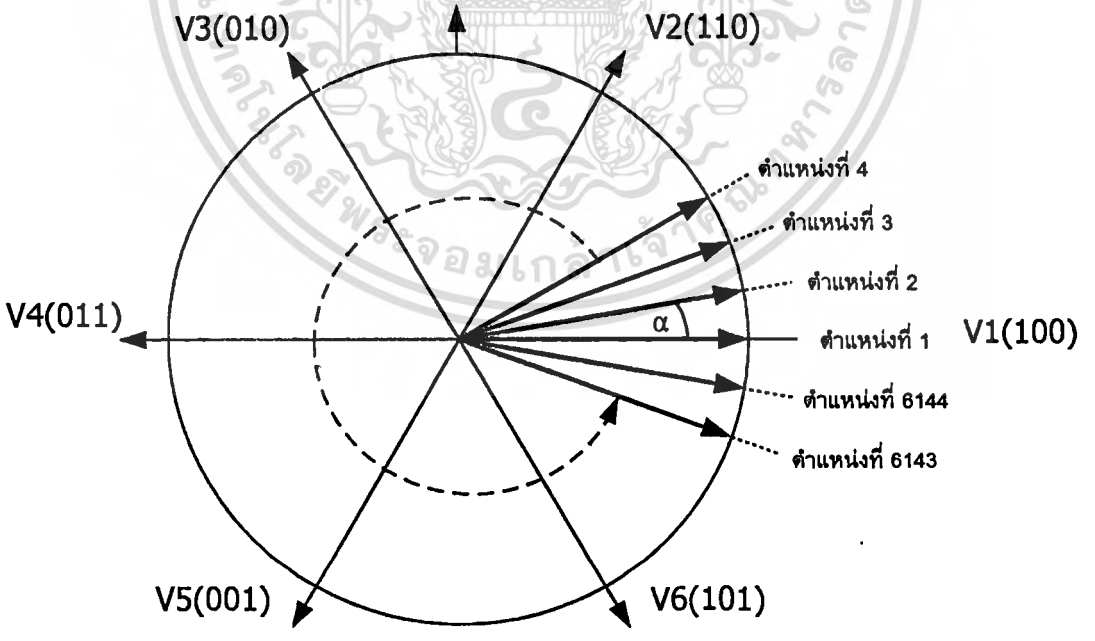
หรือถ้า St = 2 ดังนั้น $T_{fund} = \frac{6144}{2} 100\ \mu s = 0.3072$ วินาที

จะได้ $F_{fund} = \frac{1}{0.3072} = 3.25$ เฮิรตซ์

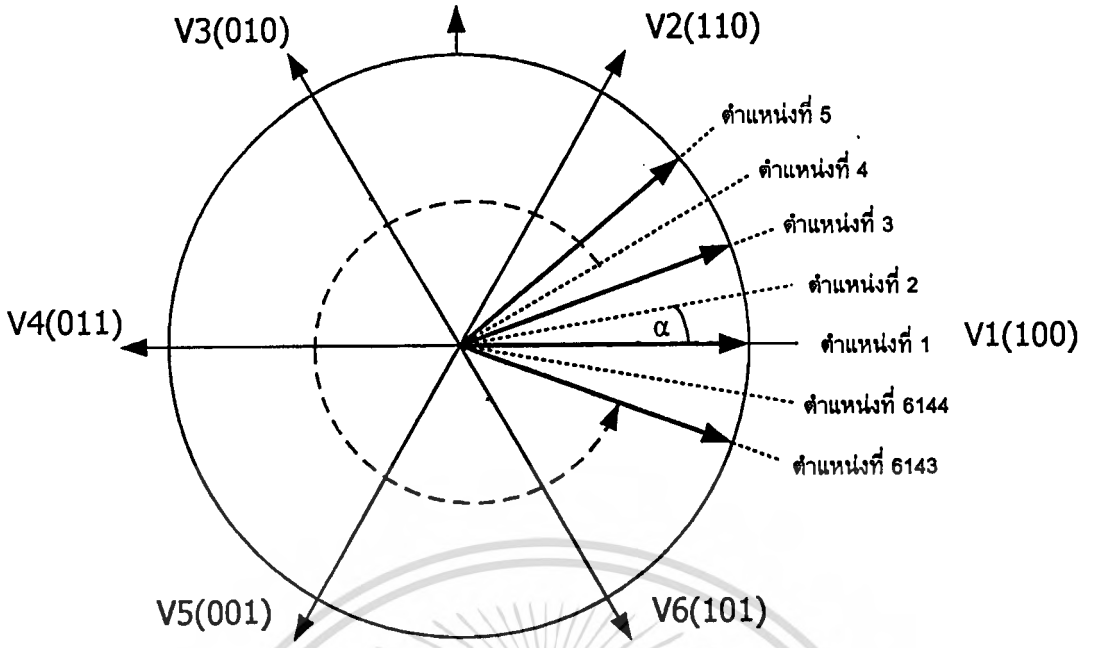
ดังนั้นค่าความถี่หลักมูลที่มากที่สุด จะได้จากกรณีที่ St = 999 ซึ่งคำนวณได้จาก

ถ้า St = 999 ดังนั้น $T_{fund} = \frac{6144}{999} 100\ \mu s = 0.615$ มิลลิวินาที

ดังนั้น $F_{fund} = \frac{1}{0.615\ ms} = 1626$ เฮิรตซ์



รูปที่ 4.4 การสร้างเวคเตอร์แรงดันเมื่อกำหนด St = 1



รูปที่ 4.5 การสร้างเวคเตอร์แรงดันเมื่อกำหนด St = 2

จากตัวอย่างในการคำนวณข้างต้น กำหนดความถี่การสุ่มเป็น 10 kHz แล้วพบว่าความถี่หลักมูลต่ำสุดจะได้ประมาณ 1.67 Hz ซึ่งหากพิจารณาในทางปฏิบัติแล้วยังไม่ใช่ความถี่ที่ต่ำนัก และจากสมการที่ 4.3 จะพบว่าคาบเวลาของความถี่หลักมูลจะแปรผันตามค่าของคาบเวลาการสุ่ม (T_s) แต่จะแปรผกผันกับค่าของตัวแปร St ดังนั้นหากต้องการเพิ่มคาบเวลาของความถี่หลักมูลจึงน่าจะกระทำได้ 2 ลักษณะคือ

- วิธีแรกคือการเพิ่มค่าของคาบเวลาการสุ่ม ก็จะมีผลให้คาบเวลาของความถี่หลักมูลมีค่าเพิ่มขึ้นด้วย แต่วิธีนี้ไม่น่าจะเหมาะสมนักเพราะในทางปฏิบัติมักจะกำหนดให้ความถี่ของการสุ่มมีค่าคงที่

- วิธีที่สองคือการเพิ่มจำนวนครั้งของการสร้างเวคเตอร์ต่อการเคลื่อนที่ครบ 1 รอบให้มากขึ้น ซึ่งก็คือการแบ่งช่วงมุม α ของการสร้างเวคเตอร์ให้มีความละเอียดมากขึ้น แต่วิธีนี้จำเป็นต้องเพิ่มจำนวนของข้อมูลคงที่ที่เก็บไว้ใน LPM_ROM1 และ LPM_ROM2 แต่เนื่องจากจำนวนหน่วยความจำภายในของ FLEX10K30EQC208 มีค่าจำกัด ดังนั้นจึงไม่สามารถเพิ่มจำนวนข้อมูลคงที่ที่เก็บอยู่ใน LPM_ROM1 และ LPM_ROM2 ได้ ซึ่งปัญหานี้อาจจะแก้ไขโดยการต่อหน่วยความจำภายนอกเช่น EPROM เป็นต้น เพิ่มเข้าไปเพื่อทำหน้าที่เก็บข้อมูลคงที่แทน LPM_ROM1 และ LPM_ROM2 ซึ่งจะทำให้สามารถแบ่งช่วงมุมของการสร้างเวคเตอร์ได้ละเอียดมากขึ้นเท่าที่ผู้ใช้ต้องการ แต่วิธีนี้จะทำให้วงจรรวมที่ต้องการออกแบบต้องเสียคุณสมบัติความเป็นตัวควบคุมแบบ

ชิปเดี่ยว(Single chip controller) ซึ่งไม่ตรงกับจุดมุ่งหมายในการออกแบบ ดังนั้นในการออกแบบจึงไม่เลือกใช้วิธีนี้

ดังนั้นเพื่อเพิ่มคาบเวลาของความถี่หลักมูล ในการออกแบบจึงใช้วิธีการสร้างเวคเตอร์แรงดันซ้ำที่ตำแหน่งเดิม เพื่อเพิ่มจำนวนครั้ง(ค่า n ตามสมการที่ 4.1) ของการสร้างเวคเตอร์ต่อการเคลื่อนที่ครบ 1 รอบให้มากขึ้น ดังนั้นสมการหาคาบเวลาของความถี่หลักมูลจะเปลี่ยนเป็น

$$T_{fund} = \frac{6144}{St} (R+1)\, T_s \qquad (4.3)$$

โดยที่ R เป็นค่าจำนวนเต็มของตัวแปรที่เก็บใน REG5 มีค่าได้ตั้งแต่ $0 \le R \le 999$ ใช้แทนจำนวนครั้งในการสร้างเวคเตอร์ซ้ำ เช่น ถ้ากำหนดคาบเวลาการสุ่มเท่ากับ 100 μs และตัวแปร St = 1 ดังนั้น

ถ้า R = 3 ดังนั้น $T_{fund} = \frac{6144}{1}\ 100\ \mu S\ (3+1) = 2.4576$ วินาที

จะได้ $F_{fund} = \frac{1}{T_{fund}} = \frac{1}{2.4576} = 0.4069$ เฮิรตซ์

ดังนั้นค่าความถี่หลักมูลที่มากที่สุด จะได้จากกรณีที่ $R = 999$ ซึ่งคำนวณได้จาก

ถ้า St = 999 ดังนั้น $T_{fund} = \frac{6144}{1}\ 100\ \mu S\ (999+1) = 614.4$ วินาที

ดังนั้น $F_{fund} = \frac{1}{T_{fund}} = \frac{1}{614.4} = 0.0016$ เฮิรตซ์

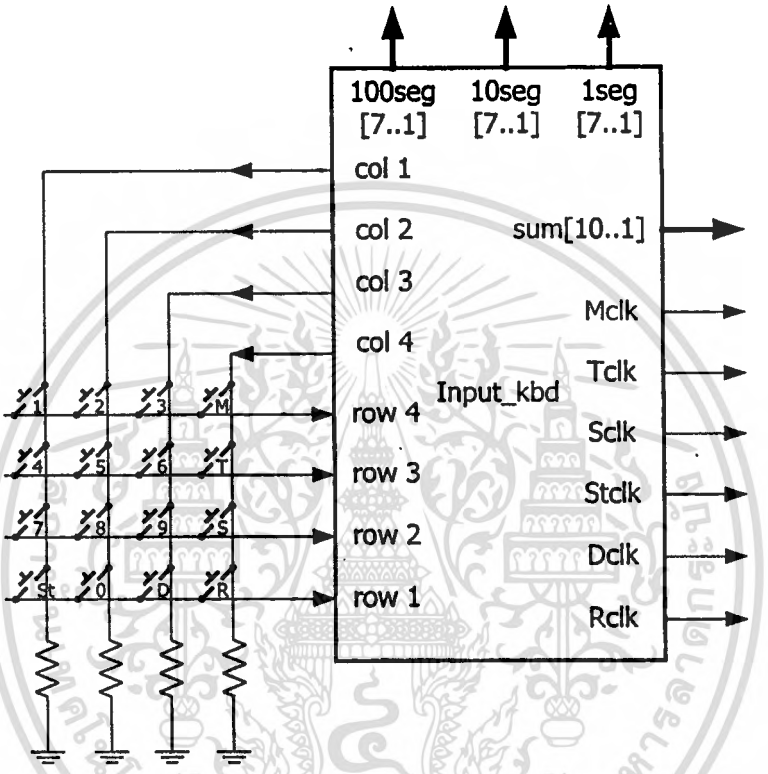
4.5 การออกแบบวงจรในแต่ละส่วน

จากบล็อคไดอะแกรมของวงจรโดยรวมในรูป 4.2 สามารถแยกการทำงานออกเป็นส่วน ๆ ได้ โดยแต่ละส่วนจะถูกออกแบบโดยใช้ภาษา AHDL ซึ่งรายละเอียดของการใช้งานภาษา AHDL จะอยู่ในภาคผนวก ข. และภาษา AHDL ที่แสดงการทำงานของแต่ละบล็อคไดอะแกรมแสดงดังภาคผนวก ค.

4.5.1. Input_kbd

รูปที่ 4.6 แสดงบล็อคไดอะแกรมของส่วน Input_kbd ส่วนนี้จะทำหน้าที่สแกนคีย์บอร์ดแบบเมตริกซ์ขนาด 4 x 4 โดยส่งสัญญาณสแกนออกทางขา col[4..1] และรับค่าการกดคีย์เข้าทาง

row[4..1] โดยเมื่อมีการกดคีย์ที่ใช้แทนตัวเลข 0-9 ค่าของคีย์นั้นจะถูกแสดงออกทาง 7-Segment และเมื่อมีการกดคีย์ที่ใช้แทนตัวแปรต่าง ๆ เช่น M, T, S, St, D หรือ R จะทำให้ส่วน Input_kbd นำตัวเลข 3 ตัวสุดท้ายที่ได้จากการกดคีย์ไปประมวลผลให้เป็นเลขฐานสองขนาด 10 บิต ส่งออกทาง sum[10..1] พร้อมกับส่งสัญญาณนาฬิกา Mclk, Tclk, Sclk, Stclk, Dclk หรือ Rclk ตัวใดตัวหนึ่งออกไปเพื่อนำค่า sum[10..1] ที่ได้ไปเก็บยังรีจิสเตอร์ตัวที่ตรงกับตัวแปรที่ต้องการ ซึ่งค่าในรีจิสเตอร์เหล่านี้จะถูกใช้ในการคำนวณค่าเวลา ta, tb และ tz ของเวคเตอร์แรงดันแต่ละเวคเตอร์



รูปที่ 4.6 บล็อคไดอะแกรมของส่วน Input_kbd

4.5.2. Register ที่ใช้เก็บตัวแปรต่าง ๆ ได้แก่

- REG D เป็นรีจิสเตอร์ขนาด 8 บิต ใช้ในการเก็บค่าเดดไทม์ของสัญญาณ
- REG S เป็นรีจิสเตอร์ขนาด 2 บิต โดยค่าที่เก็บนี้ จะใช้แทนรูปแบบของการกำเนิดสัญญาณ PWM ที่ต้องการ
- REG M เป็นรีจิสเตอร์ขนาด 10 บิต ใช้ในการเก็บค่าดัชนีการมอดูเลท
- REG T เป็นรีจิสเตอร์ขนาด 10 บิต ใช้ในการเก็บคาบเวลาการสุ่ม(T_s)
- REG R เป็นรีจิสเตอร์ขนาด 8 บิต ใช้ในการเก็บจำนวนครั้งของการสร้างเวคเตอร์ซ้ำ
- REG St เป็นรีจิสเตอร์ขนาด 10 บิต ใช้ในการเก็บ step เพื่อใช้ในการกำหนดความถี่มูลฐาน

4.5.3. Gen_adr10

ส่วนนี้ทำหน้าที่กำหนดตำแหน่งหน่วยความจำของ LPMROM1 และ LPMROM2 เพื่อส่งให้กับส่วน CAL_TIME เพื่อใช้ในการคำนวณเวลา ta, tb และ tz โดยขอบขาลงของ clk_adr10 ที่ส่งมาจากส่วน Repeat มีผลให้ส่วน Gen_adr10 นำค่า step[10..1] บวกกับค่า adr[10..1] ค่าปัจจุบัน ได้เป็นค่า adr[10..1] ค่าใหม่ ซึ่งค่านี้จะถูกใช้เป็นตำแหน่งหน่วยความจำตำแหน่งใหม่ของ LPMROM1 และ LPMROM2 ส่วนค่า sector[3..1] จะเป็นค่าที่ใช้บอกตำแหน่งเซคเตอร์ปัจจุบันของการสร้างเวคเตอร์แรงดัน เพื่อส่งให้กับส่วน SVX3 โดยค่า sector[3..1] จะมีได้ตั้งแต่ 1 ถึง 6 และจะเพิ่มขึ้น 1 ค่า ทุกครั้งที่ค่าใน adr[10..1] เกิด Overflow (หมายถึงมีค่าการนับมากกว่า 1023) และเมื่อค่า sector[3..1] เพิ่มขึ้นมากกว่า 6 ก็จะถูกรีเซ็ทให้กลายเป็น 1 ทันที

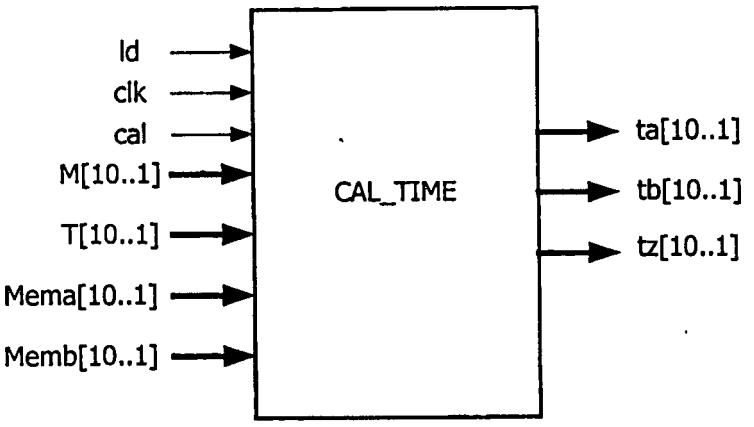
รูปที่ 4.7 บล็อคไดอะแกรมของส่วน Gen_adr10

4.5.4. LPM_ROM1 และ LPM_ROM2

เนื่องจากภายใน FLEX 10K จะประกอบด้วย EAB ซึ่งมีโครงสร้างเป็น RAM จึงสามารถประยุกต์ใช้งานเป็นหน่วยความจำ ROM ที่ใช้เก็บข้อมูลคงที่ได้ โดย LPM_ROM1 จะเก็บข้อมูลคงที่ของสมการ $\frac{3}{4}(\cos\alpha - \frac{1}{\sqrt{3}}\sin\alpha)$ และ LPM_ROM2 จะเก็บข้อมูลคงที่ของสมการ $\frac{\sqrt{3}}{2}\sin\alpha$ โดยสมการทั้งสองนี้ใช้ α เป็นตัวแปรอิสระ ซึ่งมีค่าได้ตั้งแต่ $0 \le \alpha \le 60$ องศา ซึ่งเมื่อพิจารณาจากจำนวน EAB ที่อยู่ภายใน FLEX10K30EQC208 แล้ว พบว่า ขนาดที่เหมาะสมของ LPM_ROM1 และ LPM_ROM2 คือ 1024 x 10 บิต ดังนั้นมุม α ของสมการทั้งสองจึงมีความละเอียดที่ 60/1024 = 0.0585 องศา ซึ่งข้อมูลเอาท์พุทของส่วนนี้จะถูกส่งไปยังส่วน CAL_TIME เพื่อใช้ในการคำนวณร่วมกับค่า M[10..1] และ T[10..1]

4.5.5. CAL_TIME

หน้าที่หลักของ CAL_TIME คือนำค่าจาก LPM_ROM1(Mema [10..1]), LPM_ROM2 (Memb[10..1]), M[10..1] และ T[10..1] มาคำนวณเพื่อให้ได้ค่า ta, tb และ tz โดยโครงสร้างภายในของ CAL_TIME จะประกอบด้วยวงจรคูณขนาด 10 บิต 1 วงจร และใช้วงจรสเตทแมชชีนควบคุมการทำงานของวงจรคูณนี้ บล็อคไดอะแกรมของส่วน CAL_TIME แสดงดังรูปที่ 4.8



รูปที่ 4.8 บล็อคไดอะแกรมของส่วน CAL_TIME

จะใช้สัญญาณ cal ซึ่งส่งมาจาก SVX3 เป็นสัญญาณเริ่มต้นการทำงานของวงจรสเตทแมชชีนภายใน (สัญญาณ cal จะมีสถานะเป็น "1" ในขณะที่มีการสร้างเวคเตอร์ศูนย์) การทำงานในแต่ละสถานะของสเตทแมชชีน ss แสดงดังรูปที่ 4.9 แบ่งได้เป็น

- สถานะ S0 เป็นสถานะเริ่มต้น สเตทแมชชีน ss จะรออยู่ที่สถานะนี้จนกว่าสัญญาณ cal เป็น "1" จึงจะกระโดดไปสถานะ S1

- สถานะ S1 นำค่า M คูณกับ T แล้วนำผลลัพธ์เฉพาะ 10 บิตบนของผลคูณไปเก็บไว้ในรีจิสเตอร์ MTff

- สถานะ S2 นำค่าจากรีจิสเตอร์ MTff คูณกับค่าจาก LPM_ROM1 (ในที่นี้ใช้ชื่อว่า Mema) แล้วนำผลลัพธ์เฉพาะ 10 บิตบนของผลคูณไปเก็บในรีจิสเตอร์ aff

- สถานะ S3 นำค่าจาก MTff คูณกับค่าจาก LPM_ROM2 (ใช้ชื่อว่า Memb) แล้วนำผลลัพธ์เฉพาะ 10 บิตบน เก็บในรีจิสเตอร์ bff

- สถานะ S4 สเตทแมชชีน ss จะวนที่สถานะนี้จนกว่าสัญญาณ cal จะเป็น "0" จึงจะกลับไปยังสถานะเริ่มต้น S0

ข้อมูลทั้งสองตัวจาก aff และ bff จะถูกนำไปคำนวณเพื่อหาค่า ta, tb และ tz โดย

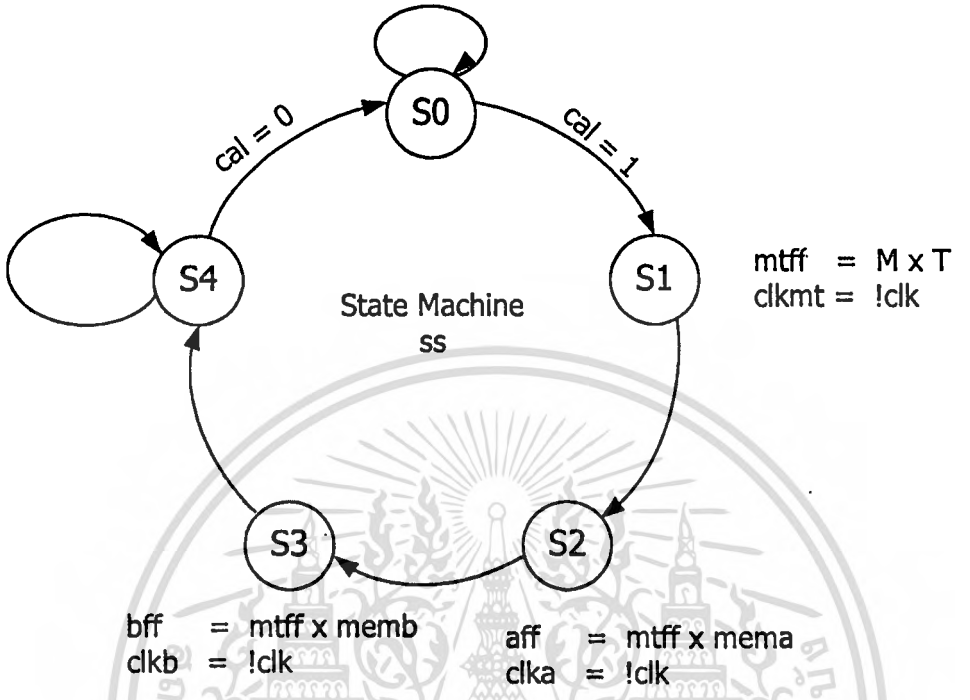
$$ta = aff$$

$$tb = bff$$

$$tz = T_s - (aff + bff)$$

สัญญาณ ld ซึ่งส่งมาจาก SVX3 จะเป็นสัญญาณที่ใช้ในการค้างค่า ta, tb และ tz ลงในรีจิสเตอร์ ta, tb และ tz ตามลำดับ และเนื่องจากในบางกรณีค่าเวลา ta หรือ tb ที่คำนวณได้จะมีค่า

เป็น 0 ซึ่งจะมีผลให้สัญญาณ PWM ที่กำเนิดได้มีความผิดพลาด ดังนั้นในการออกแบบจึงกำหนดให้ค่าเวลาต่ำสุดของของ ta หรือ tb มีค่าเป็น 2



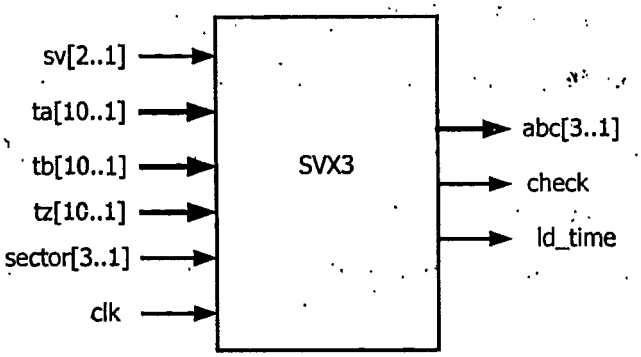
รูปที่ 4.9 สเตทแมชชีน ss

4.5.6. SVX3

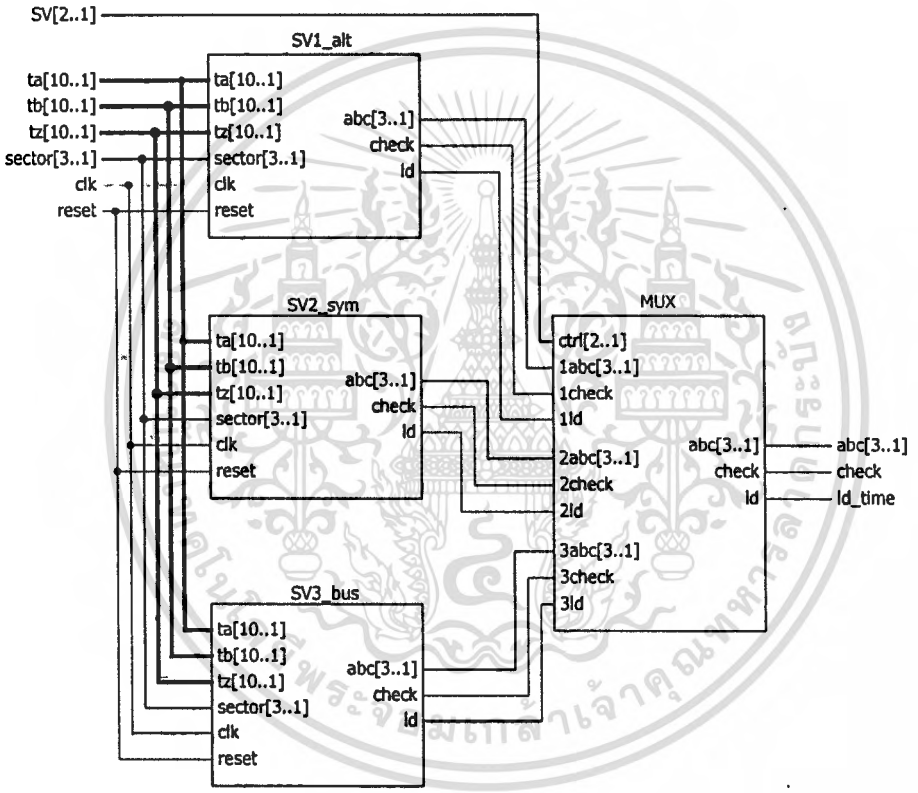
เนื่องจากในวิทยานิพนธ์ฉบับนี้ ได้กล่าวถึงรูปแบบการกำเนิดสัญญาณPWM อยู่ 3 วิธี คือ

1. Alternating zero vector scheme
2. Symmetric sequence scheme
3. The Bus clamped scheme

ซึ่งในแต่ละวิธีจะมีข้อดี-ข้อเสีย และคุณสมบัติที่แตกต่างกัน และคุณสมบัติต่างๆ จะเปลี่ยนแปลงไปเมื่อมีการเปลี่ยนจุดทำงาน และเพื่อให้วงจรรวมที่ออกแบบนี้สามารถใช้งานได้หลากหลายขึ้นโดยสามารถปรับเปลี่ยนรูปแบบของการกำเนิดสัญญาณ PWM ให้เหมาะสมกับสภาวะการใช้งาน จึงได้ออกแบบให้วงจรรวมนี้ สามารถปรับเปลี่ยนวิธีการกำเนิดสัญญาณได้ทั้ง 3 วิธี ซึ่งส่วน SVX3 จะเป็นส่วนที่ทำหน้าที่สร้างรูปแบบการสวิตชิ่งของสัญญาณ โดยจะนำค่า ta, tb และ tz ที่คำนวณได้จากส่วน CAL_TIME มากำหนดเป็นช่วงเวลาในการสร้างเวคเตอร์แต่ละเวคเตอร์และใช้ค่า sector[3..1] ซึ่งส่งมาจากส่วน Gen_adr10 ในการกำหนดรูปแบบการสวิตชิ่งให้ตรงกับรูปแบบในแต่ละเซคเตอร์ บล็อคไดอะแกรมของ SVX3 แสดงดังรูปที่ 4.10



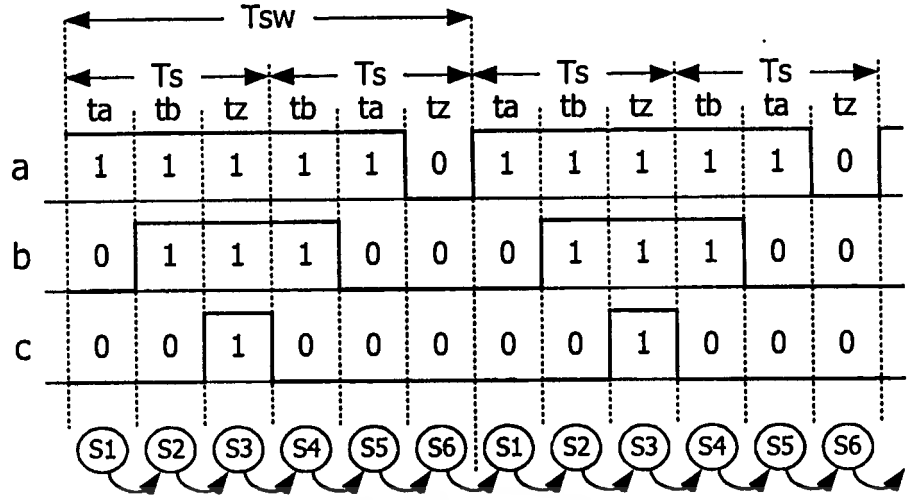
รูปที่ 4.10 บล็อคไดอะแกรมของ SVX3



รูปที่ 4.11 โครงสร้างภายในของ SVX3

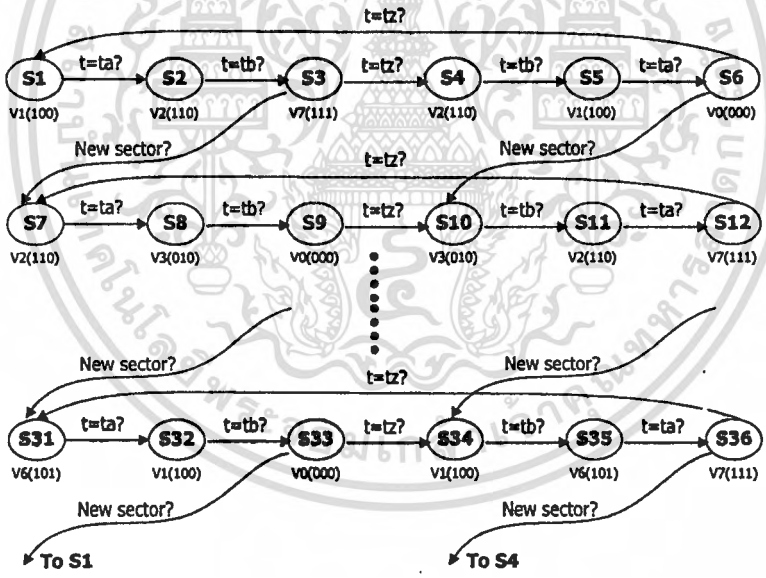
โครงสร้างภายในของ SVX3 แสดงดังบล็อคไดอะแกรมรูปที่ 4.11 ซึ่งประกอบด้วย

1. SV1_alt ส่วนนี้เป็นส่วนสร้างสัญญาณ PWM แบบ Alternating Zero Sequence ซึ่งจะออกแบบโดยใช้วงจรสเตทแมชชีนทำการสร้างสัญญาณจำนวน 36 สถานะ โดยจะแบ่งเป็นเซคเตอร์ละ 6 สถานะ เช่น ถ้าต้องการสร้างสัญญาณในเซคเตอร์ที่ 1 จะได้ความสัมพันธ์ระหว่างรูปสัญญาณและสถานะของสเตทแมชชีนดังรูปที่ 4.12



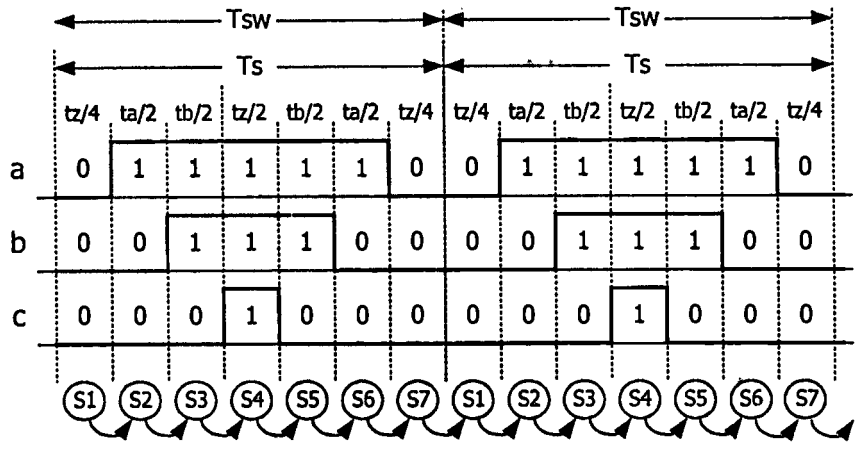
รูปที่ 4.12 ความสัมพันธ์ระหว่างสัญญาณ PWM และสถานะของสเตทแมชชีน ในเซคเตอร์ที่ 1 ของวิธี Alternating Zero Sequence

ถ้าพิจารณาจากรูปสัญญาณในทั้ง 6 เซคเตอร์แล้ว จะเขียนเป็นแผนภาพสถานะ(State Diagram) ของสเตทแมชชีน SV1 ได้เป็นรูปที่ 4.13

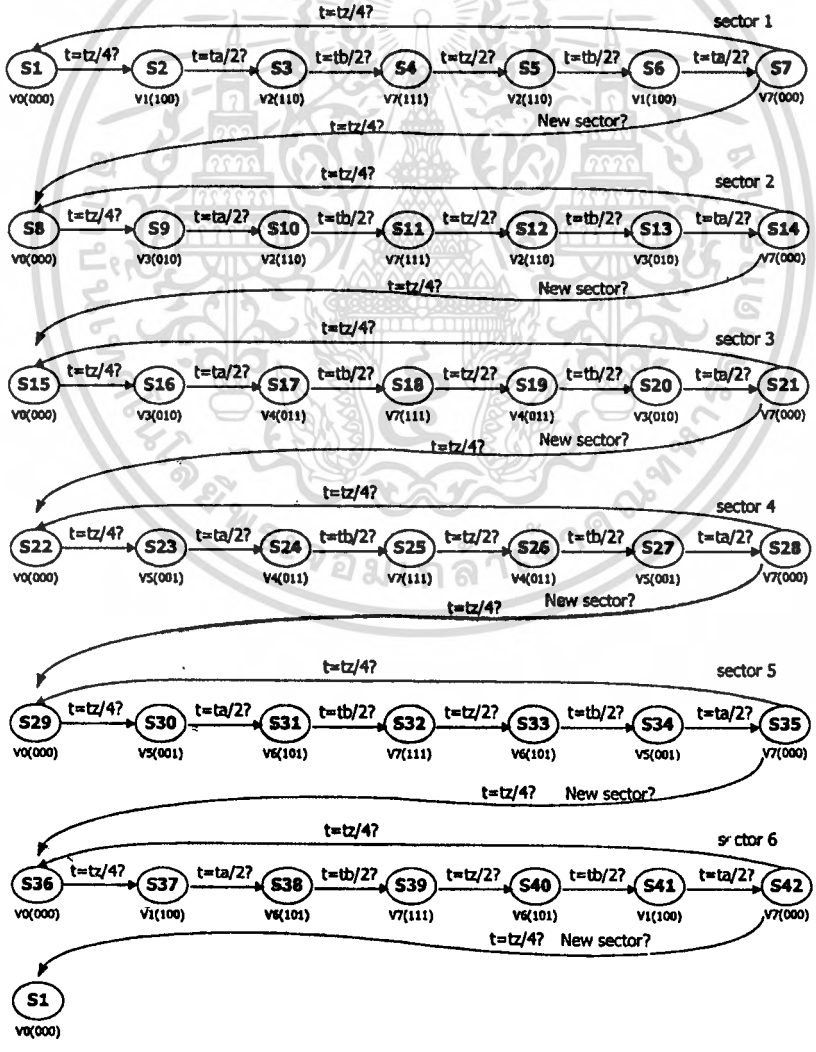


รูปที่ 4.13 แผนภาพสถานะ(State Diagram) ของสเตทแมชชีน SV1_alt

2. SV2_sym เป็นส่วนสร้างสัญญาณแบบ Symmetric Sequence ภายในประกอบด้วยวงจรสเตทแมชชีนจำนวน 42 สถานะ จากรูปที่ 4.14 เป็นการเปรียบเทียบสัญญาณในเซคเตอร์ที่ 1 กับสเตทแมชชีนที่ออกแบบไว้ และรูปที่ 4.15 เป็นแผนภาพสถานะของสเตทแมชชีนที่ใช้สร้างสัญญาณแบบ Symmetric Sequence

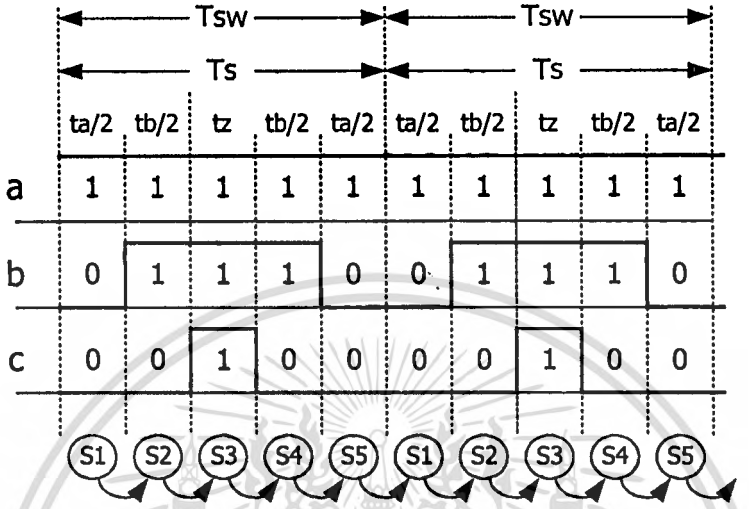


รูปที่ 4.14 ความสัมพันธ์ระหว่างสัญญาณ PWM และสถานะของสเตทแมชชีนในเซคเตอร์ที่ 1 ของวิธี Symmetric Sequence

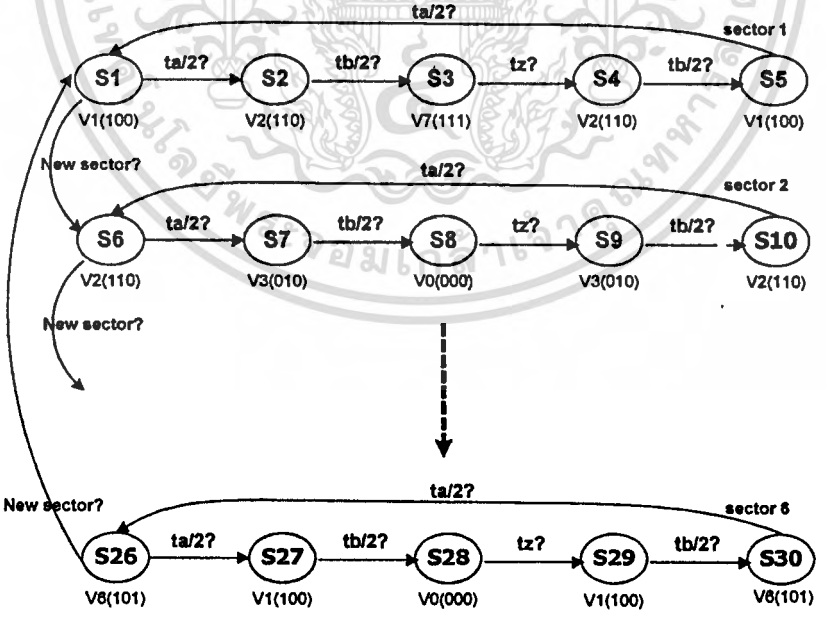


รูปที่ 4.15 แผนภาพสถานะของสเตทแมชชีนที่ใช้สร้างสัญญาณแบบ Symmetric Sequence

3. SV3_bus เป็นส่วนสร้างสัญญาณแบบ Bus Clamped ภายในประกอบด้วยวงจรสเตทแมชชีนจำนวน 30 สถานะ จากรูปที่ 4.16 เป็นการเปรียบเทียบการสร้างสัญญาณในเซคเตอร์ที่ 1 กับสเตทแมชชีนที่ออกแบบไว้ ซึ่งสเตทแมชชีนของทั้ง 6 เซคเตอร์เขียนได้ดังรูปที่ 4.17



รูปที่ 4.16 ความสัมพันธ์ระหว่างสัญญาณ PWM และสถานะของสเตทแมชชีนในเซคเตอร์ที่ 1 ของวิธี Bus clamped



รูปที่ 4.17 แผนภาพสถานะของสเตทแมชชีนที่ใช้สร้างสัญญาณแบบ Bus clamped

จากนั้นสัญญาณเอาท์พุทของแต่ละบล็อค ซึ่งได้แก่

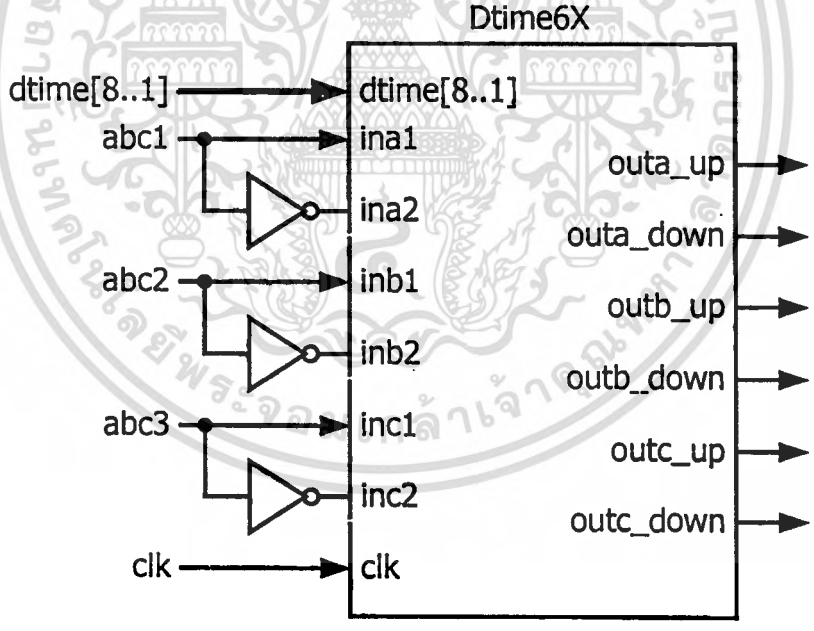
- abc[3..1] เป็นสัญญาณ PWM 3 สัญญาณ

- Check สัญญาณนี้จะเป็น "1" ในขณะที่มีการสร้างเวคเตอร์ศูนย์ ใช้เป็นสัญญาณแอคทีฟส่งให้กับสัญญาณ cal ของส่วน CAL_TIME เริ่มคำนวณค่า ta, tb และ tz รวมทั้งใช้เป็นสัญญาณนาฬิกาให้กับส่วน Gen_adr10 ทำการนับค่าตำแหน่งหน่วยความจำค่าใหม่

- ld_time จะเกิดขึ้นในช่วงสัญญาณนาฬิกาลูกสุดท้ายของการสร้างเวคเตอร์ศูนย์ สัญญาณนี้จะถูกส่งให้กับสัญญาณ ld ของส่วน CAL_TIME เพื่อใช้ในการค้างค่า ta, tb และ tz ไว้ที่รีจิสเตอร์เอาท์พุทของส่วน CAL_TIME

4.5.7 Dtime6X

ในส่วนนี้แสดงดังบล็อคไดอะแกรมในรูป 4.18 โดยจะนำสัญญาณ abc[3..1] จากส่วน SVX3 มาผ่าน NOT เกท ได้เป็นสัญญาณ 6 สัญญาณ จากนั้นวงจรจะทำการตรวจสอบขอบขาขึ้นของสัญญาณทั้ง 6 สัญญาณนี้ และจะหน่วงเวลาไปเท่ากับค่าเดดไทม์ (dtime[8..1]) ที่ตั้งไว้

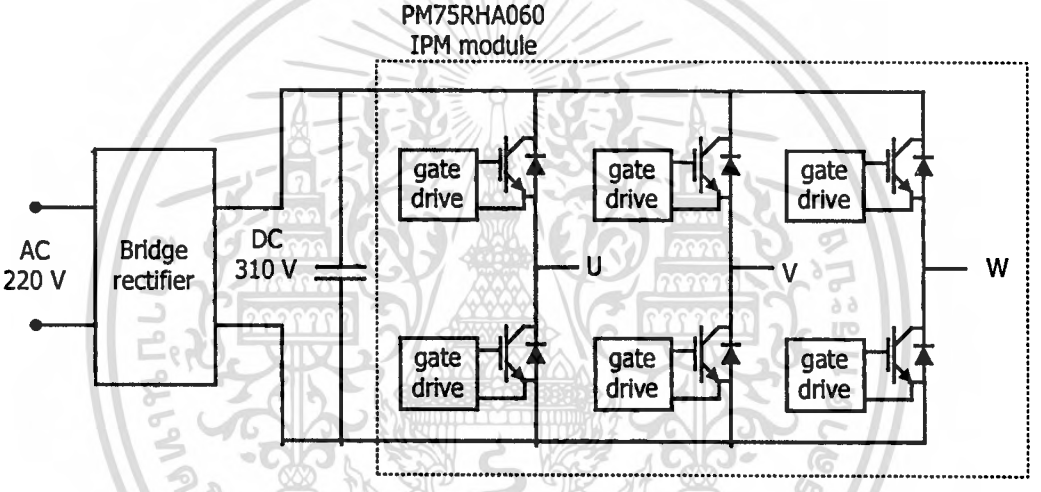


รูปที่ 4.18 บล็อคไดอะแกรมของส่วน DTIME6X

ซึ่งเอาท์พุททั้ง 6 ของส่วน Dtime6X จะเป็นเอาท์พุทของวงจรรวมที่ใช้ในการควบคุมชุดอินเวอร์เตอร์

4.6 การออกแบบวงจรอินเวอร์เตอร์

วงจรอินเวอร์เตอร์ที่ใช้ในการทดลองจะใช้โมดูลสำเร็จรูปของมิตซูบิชิเบอร์ PM75RHA060 ซึ่งวงจรภายในประกอบด้วยไอจีบีทีจำนวน 6 ตัว ต่อกันเป็นอินเวอร์เตอร์ 3 เฟส โมดูลนี้ทนแรงดันสูงสุด 600 โวลต์ และกระแส 75 แอมป์ ส่วนดีซีบัสจะต่อกับตัวเก็บประจุค่า 3300 ไมโครฟารัด 400 โวลต์ และ ตัวเก็บประจุขนาดเล็กค่า 0.1 ไมโครฟารัด 630 โวลต์ ทำหน้าที่ในการกรองสัญญาณรบกวน สำหรับส่วนดีซีบัสจะได้มาจากวงจรบริดจ์ ซึ่งแปลงไฟกระแสสลับ 220 โวลต์เป็นไฟกระแสตรง 310 โวลต์ สัญญาณจากวงจรรวม FPGA จำนวน 6 สัญญาณ จะถูกแยกกราวด์ออกจากวงจรอินเวอร์เตอร์ ด้วยออปโตคัปเปอร์ H11L1 ซึ่งมี ton และ toff ประมาณ 1 ไมโครวินาที รายละเอียดของอุปกรณ์แต่ละตัวอยู่ในภาคผนวก ง.



รูปที่ 4.19 วงจรส่วนอินเวอร์เตอร์ที่ใช้ในการทดลอง

บทที่ 5
ผลการทดลอง

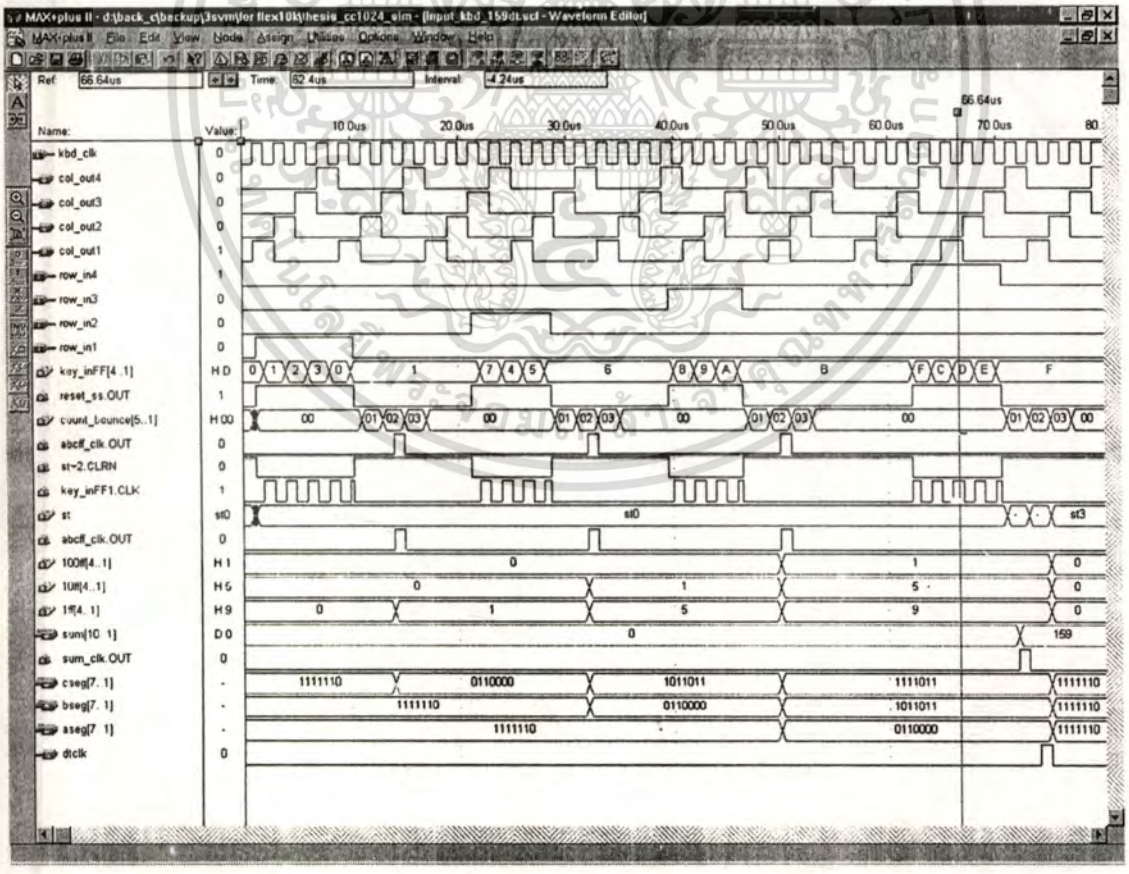
5.1 บทนำ

ในบทนี้ผลการทดลองจะถูกแบ่งออกเป็น 2 ส่วน โดยส่วนแรกจะเป็นผลจำลองการทำงานของบล็อกไดอะแกรมต่างๆ ที่ได้ออกแบบไว้ในบทที่ 4 โดยใช้โปรแกรม MAXPLUS II และในส่วนที่สองจะเป็นผลจริงของสัญญาณที่วงจรรวม FPGA กำเนิดออกมา รวมไปถึงผลของการนำสัญญาณ PWM ที่ได้ไปขับชุดอินเวอร์เตอร์เพื่อนำไปขับเคลื่อนมอเตอร์เหนี่ยวนำต่อไป

5.2 ผลจำลองการทำงานด้วยโปรแกรม MAXPLUS II

5.2.1 ผลจำลองการทำงานของส่วน Input_kbd

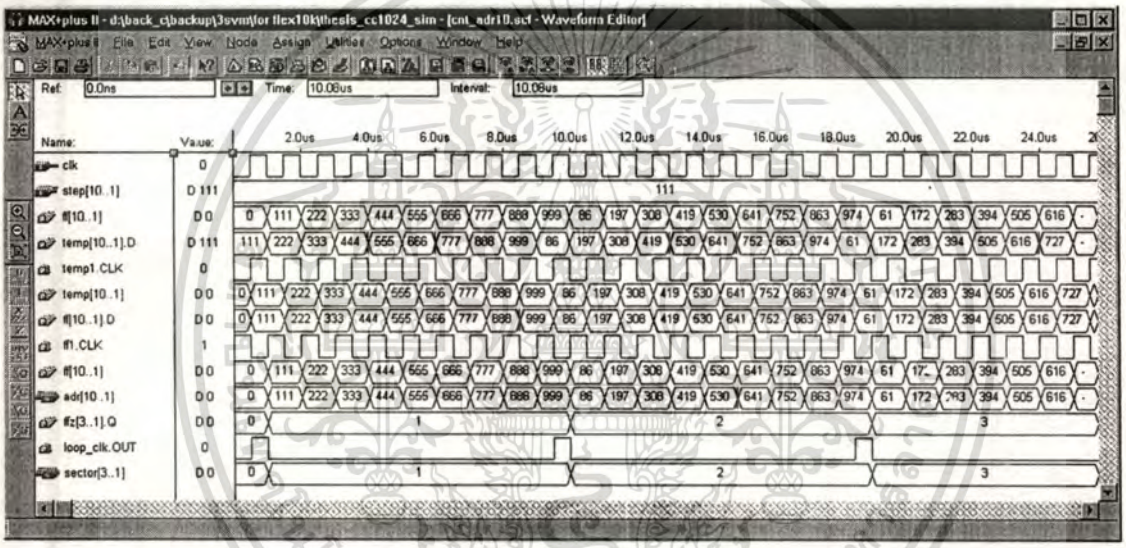
รูปที่ 5.1 เป็นผลจำลองการทำงาน เมื่อมีการกดคีย์ 1 5 9 dt ตามลำดับ โดยกำหนดให้มีการตรวจสอบสัญญาณเบาซ์(Bounce) ของการกดคีย์เพียง 3 สัญญาณนาฬิกาเท่านั้น



รูปที่ 5.1 ผลจำลองการทำงาน เมื่อมีการกดคีย์ 1 5 9 dt ตามลำดับ

5.2.2 ผลจำลองการทำงานของส่วน Gen_adr10

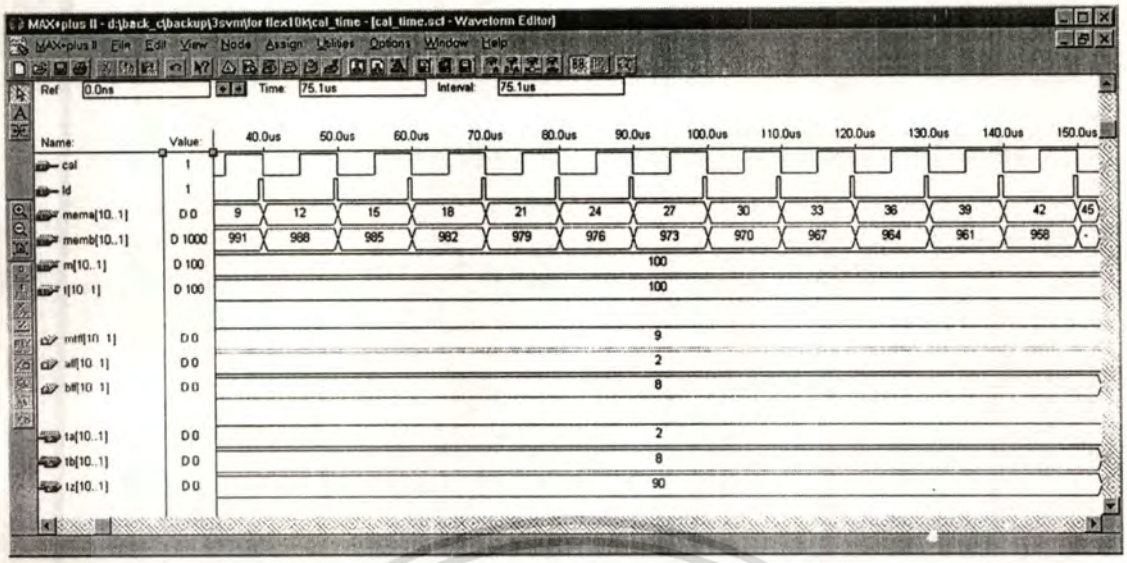
รูปที่ 5.2 เป็นผลจำลองการทำงานของส่วน Gen_adr10 โดยการทำงานของส่วนนี้จะนำค่า step[10..1] บวกกับค่าแอดเดรสเอาท์พุทค่าปัจจุบัน (adr [10..1]) เมื่อมีขอบขาลงของสัญญาณ clk มาแอคทีฟ โดยแอดเดรสค่านี้จะถูกส่งให้กับ LPM_ROM1 และ LPM_ROM2 เพื่อใช้กำหนดตำแหน่งแอดเดรสของข้อมูลคงที่ ที่เก็บใน LPM_ROM1 และ LPM_ROM2 ซึ่งเมื่อค่า adr [10..1] มากกว่า 1023 แล้วค่า sector [3..1] จะเพิ่มค่าขึ้น 1 ค่าโดยอัตโนมัติ โดยค่า sector [3..1] นี้จะถูกส่งให้กับส่วน SVX3 เพื่อกำหนดรูปแบบของการสวิทชิ่งในแต่ละเซคเตอร์ และค่า sector [3..1] นี้จะมีค่าได้ตั้งแต่ 1 ถึง 6 โดยผลการจำลองในรูปที่ 5.2 เป็นการกำหนดค่า step [10..1] = 111



รูปที่ 5.2 ผลจำลองการทำงานของส่วน Gen_adr10

5.2.3 ผลจำลองการทำงานของส่วน Cal_time

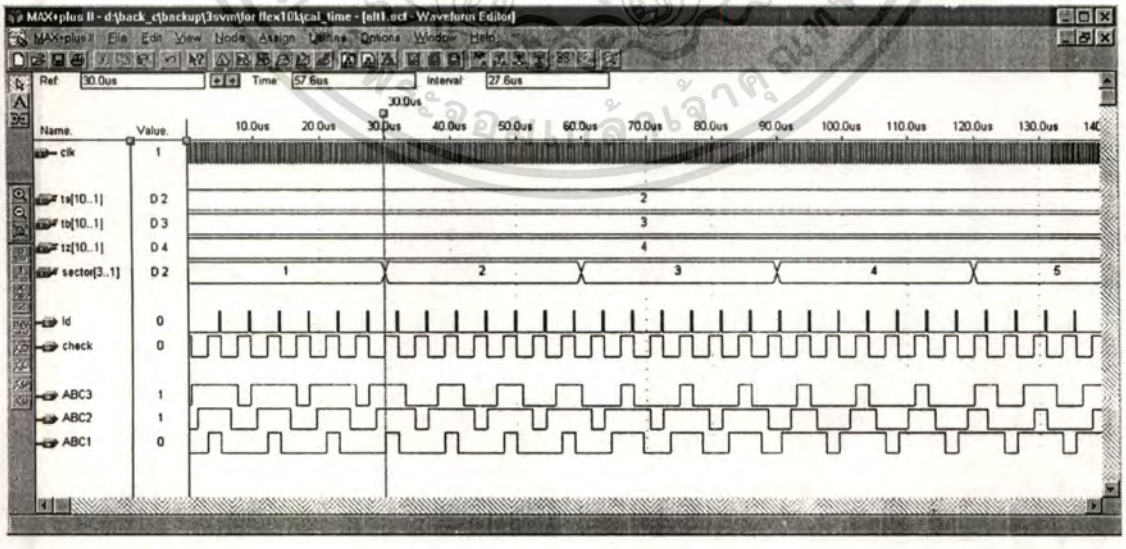
ส่วนนี้จะนำค่า M[10..1], T[10..1], ข้อมูลคงที่จาก LPM_ROM1 (mema[10..1]) และ LPM_ROM2 (memb[10..1]) มาคำนวณตามสมการที่ 2.20 - 2.22 เพื่อให้ได้ค่า ta, tb และ tz โดยในการจำลองนี้จะกำหนดให้ M[10..1] = 100 , T[10..1] = 100 ตามลำดับ ผลจำลองแสดงดังรูปที่ 5.3



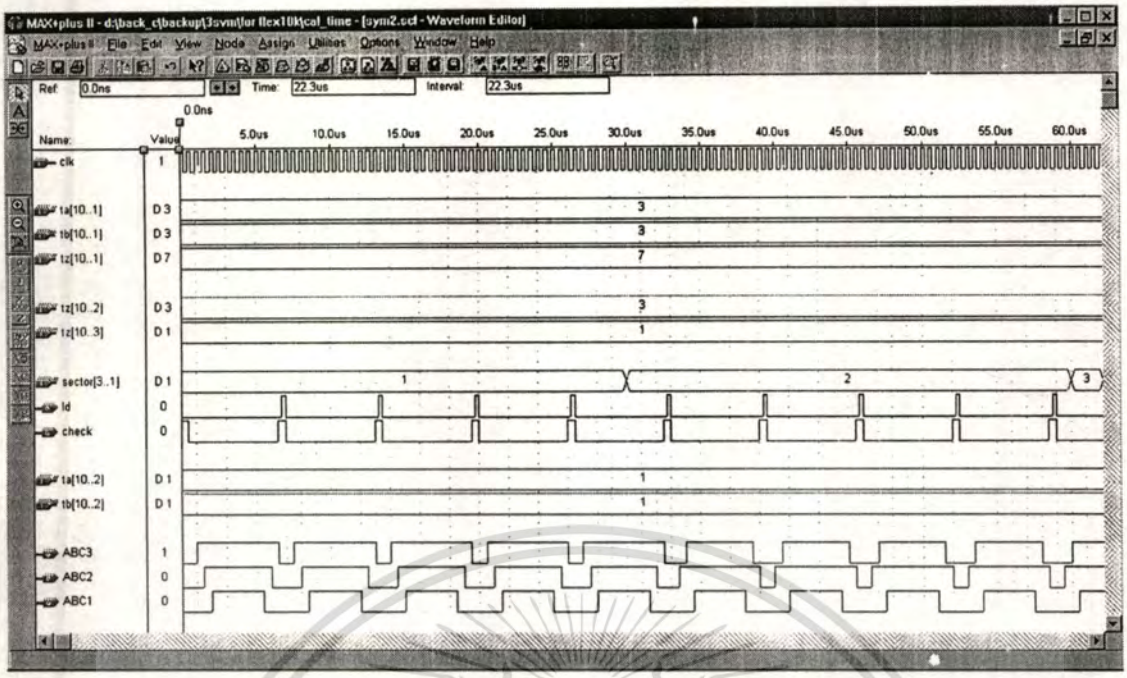
รูปที่ 5.3 ผลจำลองการทำงานของส่วน Cal_time

5.2.4 ผลจำลองการทำงานของส่วน SVX3

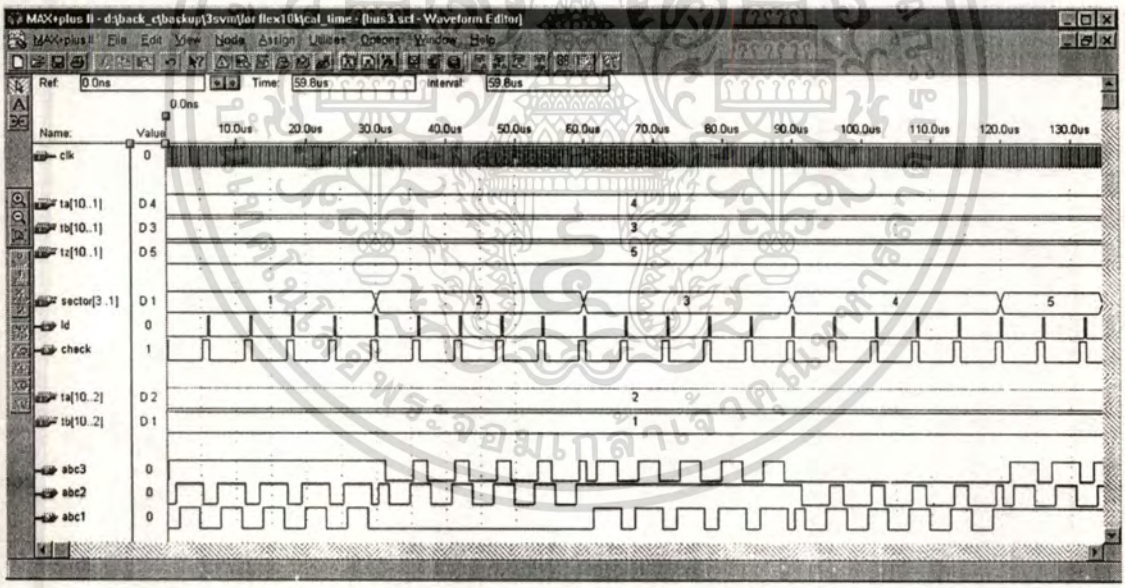
ส่วนนี้จะทำหน้าที่กำเนิดสัญญาณ PWM ตามรูปแบบของการมอดูเลทที่เลือกไว้ โดยการกำหนดค่าในรีจิสเตอร์ SV โดยในรูปที่ 5.4 จะเป็นผลจำลองการกำเนิดสัญญาณ PWM โดยวิธี Alternating zero vector sequence ในแต่ละเซคเตอร์ ส่วนในรูปที่ 5.5 จะเป็นผลจำลองการกำเนิดสัญญาณ PWM โดยวิธี Symmetric sequence ส่วนในรูปที่ 5.6 จะเป็นผลจำลองการกำเนิดสัญญาณ PWM โดยวิธี Bus clamped



รูปที่ 5.4 ผลจำลองการกำเนิดสัญญาณ PWM โดยวิธี Alternating zero vector sequence



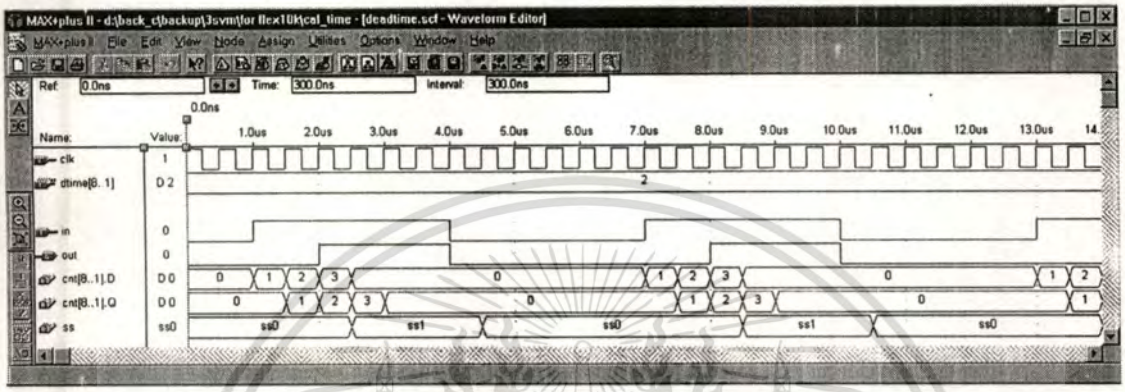
รูปที่ 5.5 ผลจำลองการกำเนิดสัญญาณ PWM โดยวิธี Symmetric sequence



รูปที่ 5.6 ผลจำลองการกำเนิดสัญญาณ PWM โดยวิธี Bus clamped

5.2.5 ผลจำลองการทำงานของส่วน Dtime6X

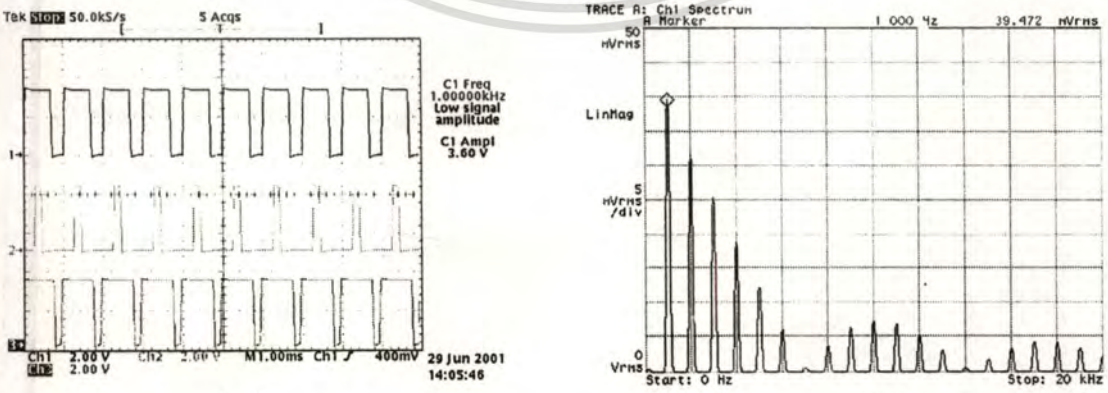
ส่วนนี้ทำหน้าที่สร้างสัญญาณหน่วง (Delay time) ให้กับสัญญาณ PWM ที่จะนำไปขับทรานซิสเตอร์กำลังที่อยู่ในกิ่งเดียวกัน เพื่อไม่ให้เกิดการนำกระแสพร้อมกัน โดยเวลาหน่วงที่สร้างนี้จะขึ้นอยู่กับฐานเวลาของสัญญาณนาฬิกาและค่าที่เก็บในรีจิสเตอร์ dtime[8..1] โดยรูปที่ 5.7 เป็นผลการจำลองเมื่อกำหนดค่า dtime[8..1] = 2



รูปที่ 5.7 ผลจำลองเมื่อกำหนดค่า dtime[8..1] = 2

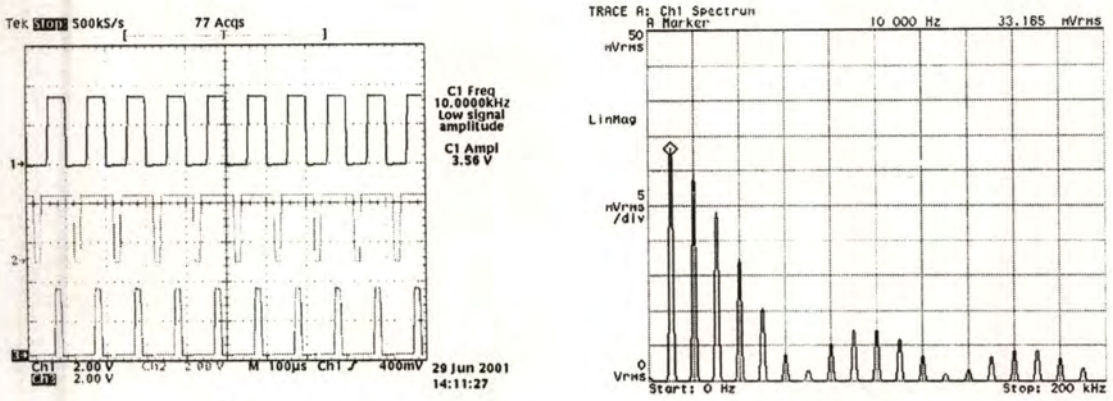
5.3 ผลของสัญญาณจริงที่กำเนิดจากวงจรรวม FPGA

ในส่วนนี้จะเป็นผลของรูปสัญญาณจริงที่กำเนิดจากวงจรรวม FLEX10K30EQC208 เมื่อมีการกำหนดค่าตัวแปรของสัญญาณต่างๆ กัน โดยรูปที่ 5.8 (ก) – 5.10 (ก) แสดงสัญญาณ PWM ในทั้ง 3 เฟส โดยกำหนดให้สร้างสัญญาณด้วยวิธี Alternating zero sequence และสวิตชิ่งที่ความถี่ 1 กิโลเฮิรตซ์, 10 กิโลเฮิรตซ์ และ 100 กิโลเฮิรตซ์ ตามลำดับ ส่วนรูปที่ 5.8 (ข) – 5.10 (ข) จะเป็นผลการวัดสเปคตรัมของสัญญาณที่ความถี่ต่างๆ



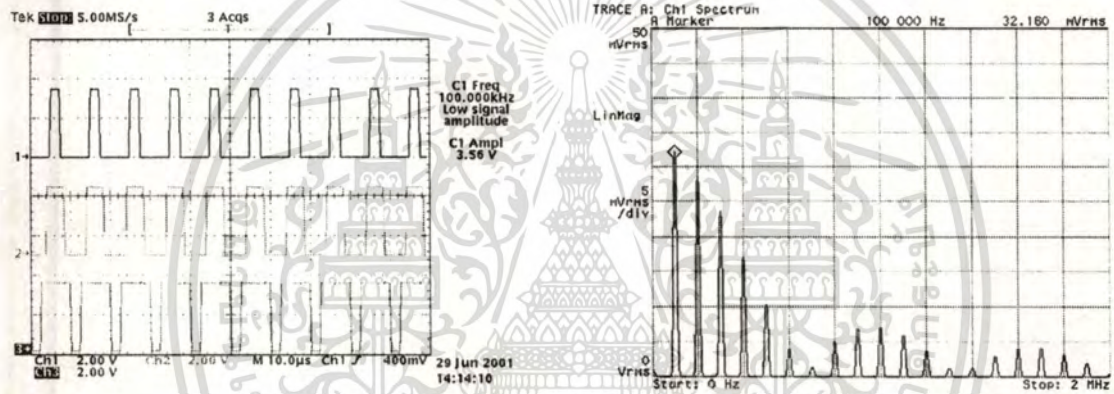
(ก) (ข)

รูปที่ 5.8 สัญญาณ PWM สวิตชิ่งที่ความถี่ 1 กิโลเฮิรตซ์ และสเปคตรัมของสัญญาณ



(ก) (ข)

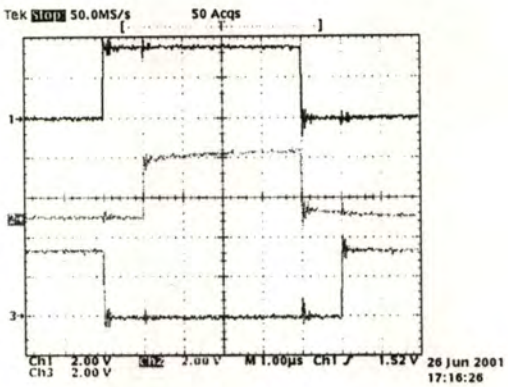
รูปที่ 5.9 สัญญาณ PWM สวิตชิ่งที่ความถี่ 10 กิโลเฮิรตซ์ และสเปคตรัมของสัญญาณ



(ก) (ข)

รูปที่ 5.10 สัญญาณ PWM สวิตชิ่งที่ความถี่ 100 กิโลเฮิรตซ์ และสเปคตรัมของสัญญาณ

สัญญาณในรูปที่ 5.11 แสดงผลของการสร้างเดดไทม์ เมื่อกำหนดให้ตัวแปร dt มีค่าเป็น 2 โดยจะทำให้เกิดช่วงเวลาเดดไทม์เท่ากับ 1 ไมโครวินาที โดยสัญญาณในช่องที่ 1 ของออสซิลโลสโคปเป็นสัญญาณ PWM อ้างอิง ส่วนสัญญาณในช่องที่ 2 เป็นสัญญาณที่ใช้ขับทรานซิสเตอร์ตัวบนของชุดอินเวอร์เตอร์ ส่วนสัญญาณในช่องที่ 3 เป็นสัญญาณที่ใช้ขับทรานซิสเตอร์ตัวล่าง



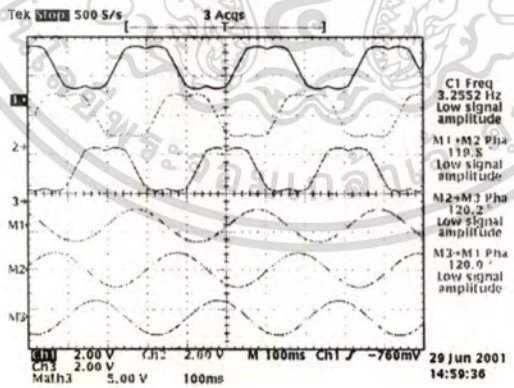
รูปที่ 5.11 ผลการสร้างเดดไทม์เมื่อกำหนดค่าตัวแปร dt = 2

สัญญาณในช่องที่ 1 เป็นสัญญาณอ้างอิง

สัญญาณในช่องที่ 2 เป็นสัญญาณที่ใช้ขับทรานซิสเตอร์กำลังตัวบน

สัญญาณในช่องที่ 3 เป็นสัญญาณที่ใช้ขับทรานซิสเตอร์กำลังตัวล่าง

สัญญาณในรูปที่ 5.12 – 5.14 จะเป็นผลการทดลองเมื่อนำสัญญาณที่ได้จากวงจรรวมไปผ่านฟังก์ชั่น Hi resolution ของออสซิลโลสโคป Tektronic รุ่น TDS477 ซึ่งจะเป็นฟังก์ชั่นที่ทำหน้าที่กรองความถี่สูงของสัญญาณ PWM ออกไป ดังนั้นสัญญาณที่ได้นี้จะใช้แทนมอดูเลติ้งฟังก์ชั่นของสัญญาณ PWM โดยกำหนดให้สวิตชิ่งที่ความถี่ 10 กิโลเฮิรตซ์ และสร้างความถี่หลักมูลเท่ากับ 3.25 เฮิรตซ์



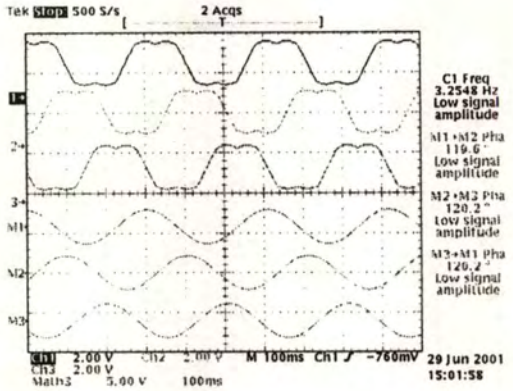
รูปที่ 5.12 มอดูเลติ้งฟังก์ชั่นของสัญญาณ PWM ที่สร้างจากวิธี Alternating zero sequence

สัญญาณในช่องที่ 1, 2 และ 3 เป็นการวัดสัญญาณในแต่ละเฟสเทียบกับกราวด์

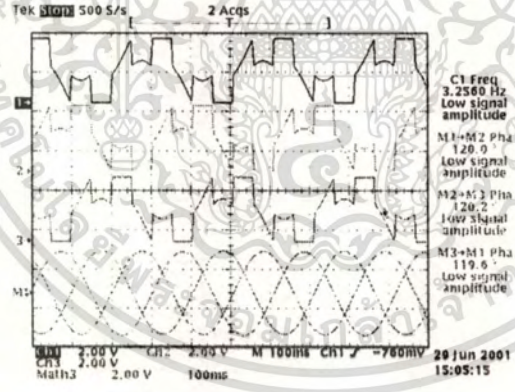
สัญญาณ M1 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 1 และ 2

สัญญาณ M2 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 2 และ 3

สัญญาณ M3 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 3 และ 1

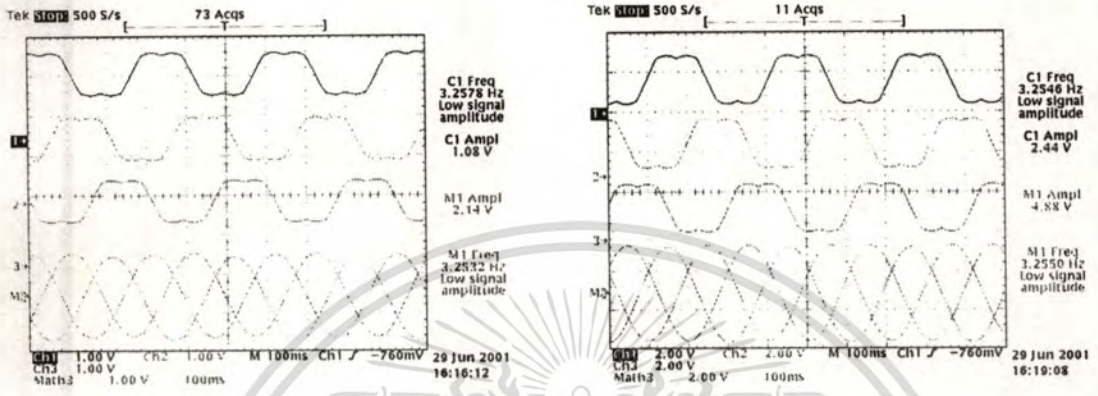


รูปที่ 5.13 มอดูเลติ้งฟังก์ชั่นของสัญญาณ PWM ที่สร้างจากวิธี Symmetric sequence
สัญญาณในช่องที่ 1, 2 และ 3 เป็นการวัดสัญญาณในแต่ละเฟสเทียบกับกราวด์
สัญญาณ M1 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 1 และ 2
สัญญาณ M2 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 2 และ 3
สัญญาณ M3 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 3 และ 1



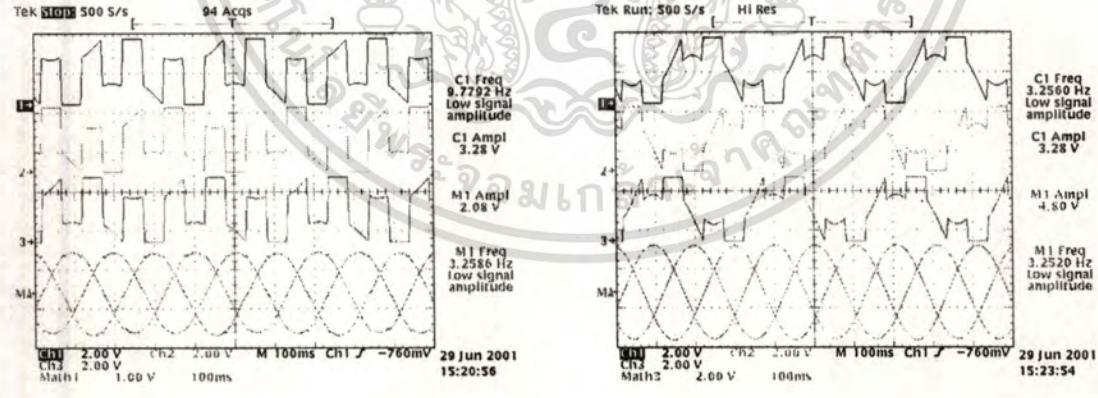
รูปที่ 5.14 มอดูเลติ้งฟังก์ชั่นของสัญญาณ PWM ที่สร้างจากวิธี Bus clamped
สัญญาณในช่องที่ 1, 2 และ 3 เป็นการวัดสัญญาณในแต่ละเฟสเทียบกับกราวด์
สัญญาณ M1 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 1 และ 2
สัญญาณ M2 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 2 และ 3
สัญญาณ M3 เป็นการวัดสัญญาณเทียบกันระหว่างสัญญาณในช่องที่ 3 และ 1

ส่วนรูปที่ 5.15 – 5.16 แสดงผลของการกำหนดค่าดัชนีการมอดูเลทของสัญญาณ เมื่อกำหนดให้ตัวแปร M มีค่าเป็น 400, 900 ตามลำดับ ซึ่งเทียบได้กับค่าดัชนีการมอดูเลทเท่ากับ 0.4 และ 0.9 ตามลำดับ โดยรูปที่ 5.15 เป็นผลจากวิธี Alternating zero vector sequence ส่วนรูปที่ 5.16 เป็นผลจากวิธี Bus clamped



(ก) (ข)

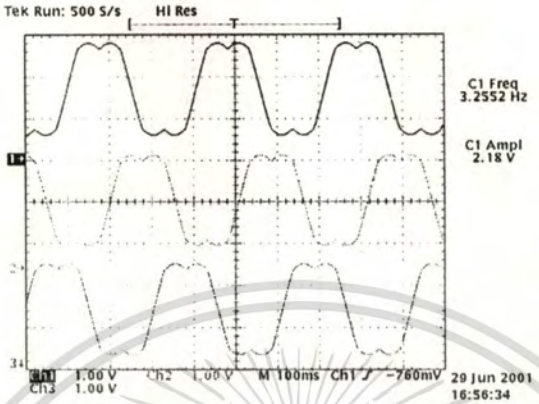
รูปที่ 5.15 มอดูเลติ้งฟังก์ชั่นของสัญญาณที่สร้างจากวิธี Alternating zero sequence เมื่อทำการกำหนดมอดูเลชั่นอินเด็กมีค่า 0.4 และ 0.9 ตามลำดับ



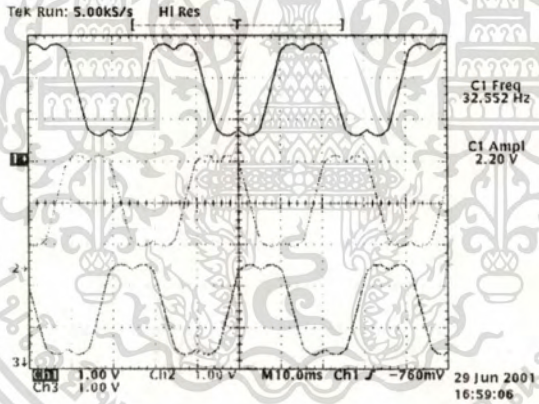
(ก) (ข)

รูปที่ 5.16 มอดูเลติ้งฟังก์ชั่นของสัญญาณที่สร้างจากวิธี Bus clamped เมื่อกำหนดมอดูเลชั่นอินเด็กมีค่า 0.4 และ 0.9 ตามลำดับ

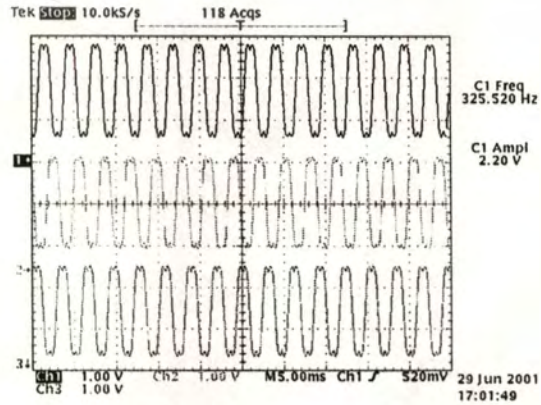
รูปที่ 5.17 – 5.19 เป็นผลของตัวแปร Step ที่มีต่อความถี่หลักมูล เมื่อกำหนดให้ตัวแปร Rpt = 0, T = 100 และ Step มีค่าเป็น 1, 10, 100 มีผลให้ความถี่หลักมูลเท่ากับ 3.25, 32.5 และ 325.5 เฮิรตซ์ ตามลำดับ และสวิตชิ่งที่ความถี่ 10 กิโลเฮิรตซ์



รูปที่ 5.17 ความถี่หลักมูลเมื่อกำหนดตัวแปรStep = 1 **มีผลให้ความถี่หลักมูลเท่ากับ** 3.25 **เฮิรตซ์**

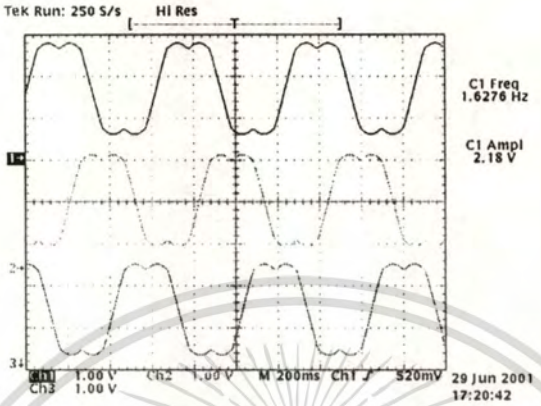


รูปที่ 5.18 ความถี่หลักมูลเมื่อกำหนดตัวแปรStep = 10 **ให้ความถี่หลักมูลเท่ากับ** 32.5 **เฮิรตซ์**

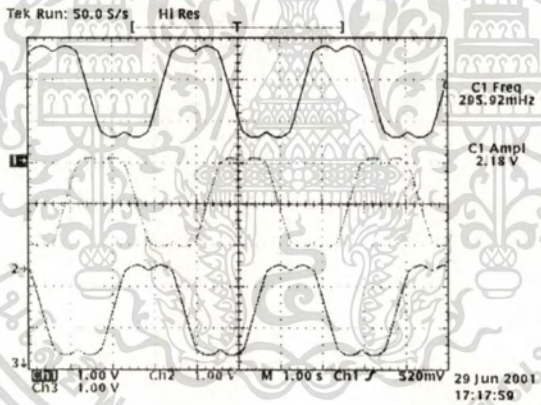


รูปที่ 5.19 ความถี่หลักมูลเมื่อกำหนดตัวแปรStep = 100 **ให้ความถี่หลักมูลเท่ากับ** 325.5 **เฮิรตซ์**

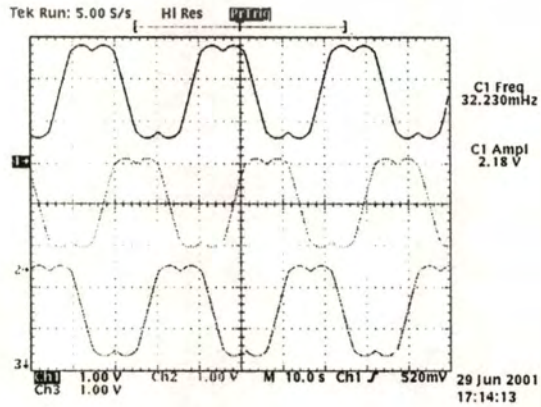
ส่วนรูปที่ 5.20 – 5.22 เป็นผลของตัวแปร Rpt ที่มีต่อความถี่หลักมูล เมื่อกำหนดให้ตัวแปร Rpt มีค่าเป็น 1, 10 และ 100 ตามลำดับ ตัวแปร Step = 1 และ T = 100 มีผลให้ความถี่หลักมูลเท่ากับ 1.62, 0.295 และ 0.032 เฮิรตซ์ ตามลำดับและสวิตชิ่งที่ความถี่ 10 กิโลเฮิรตซ์



รูปที่ 5.20 ความถี่หลักมูลเมื่อกำหนดตัวแปร Rpt = 1 **มีผลให้ความถี่หลักมูลเท่ากับ** 1.62 **เฮิรตซ์**



รูปที่ 5.21 ความถี่หลักมูลเมื่อกำหนดตัวแปร Rpt = 10 **ให้ความถี่หลักมูลเท่ากับ** 0.295 **เฮิรตซ์**



รูปที่ 5.22 ความถี่หลักมูลเมื่อกำหนดตัวแปร Rpt = 100 **ให้ความถี่หลักมูลเท่ากับ** 0.032 **เฮิรตซ์**

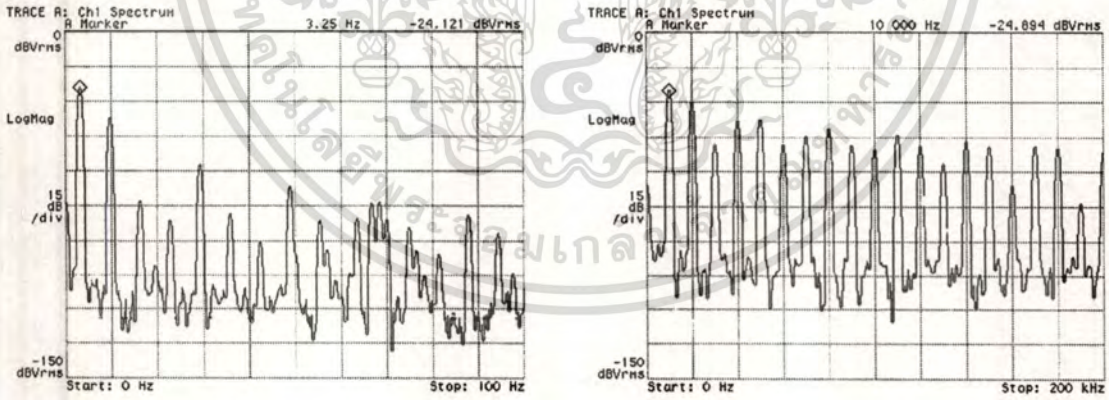
5.4 ผลการวัดสเปคตรัมของสัญญาณ

เมื่อนำสัญญาณ PWM ที่สร้างจากวงจรรวม ไปวิเคราะห์หาค่าฮาร์โมนิคที่ความถี่ต่างๆ โดยใช้ Vector signal analyzer ของ HP รุ่น 89441A และกำหนดเงื่อนไขในการสร้างสัญญาณดังนี้

1. T = 100 กำหนดคาบเวลาการสุ่มเท่ากับ 10 กิโลเฮิรตซ์ มีผลให้สัญญาณที่สร้างจากวิธี Alternating zero sequence สวิตชิ่งที่ความถี่ 10 กิโลเฮิร์ต ส่วนสัญญาณที่สร้างจากวิธี Symmetric sequence และ Bus clamped สวิตชิ่งที่ความถี่ 20 กิโลเฮิร์ต
2. M = 0.9 กำหนดมอดูเลชั่นอินเด็กเท่ากับ 0.9
3. Rpt = 0 ,St = 1 กำหนดความถี่หลักมูลเท่ากับ 3.25 เฮิรตซ์

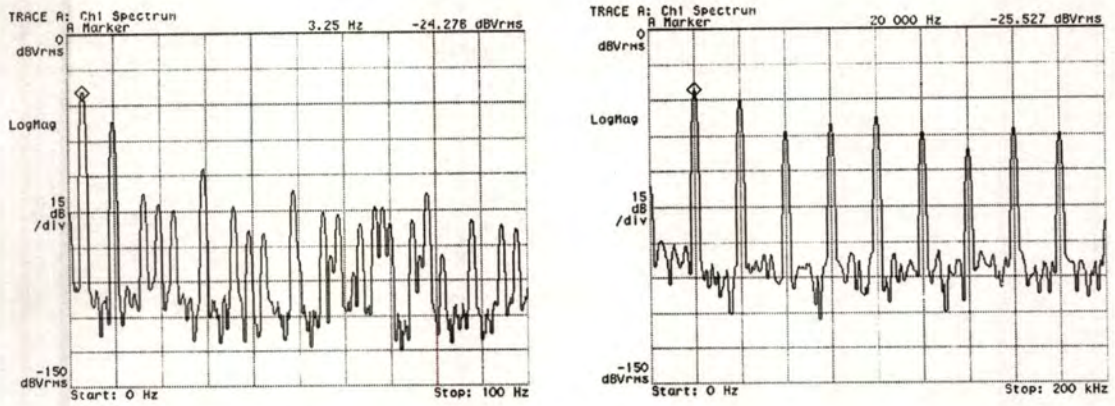
ได้ผลการทดลองดังรูปที่ 5.23 – 5.25 โดยการวัดจะแบ่งออกเป็น 2 ช่วง คือ

1. สัญญาณในรูป (ก) จะเป็นการวัดในช่วงความถี่ต่ำ เพื่อวัดองค์ประกอบความถี่ต่ำที่เกิดขึ้นของสัญญาณทั้ง 3 รูปแบบ โดยกำหนดช่วงการวัดตั้งแต่ 0 – 100 เฮิรตซ์
2. สัญญาณในรูป (ข) เป็นการวัดในช่วงความถี่สูง เพื่อวัดองค์ประกอบที่เกิดขึ้นในย่านของความถี่สวิตชิ่งของสัญญาณ โดยกำหนดช่วงการวัดตั้งแต่ 0 – 200 กิโลเฮิรตซ์



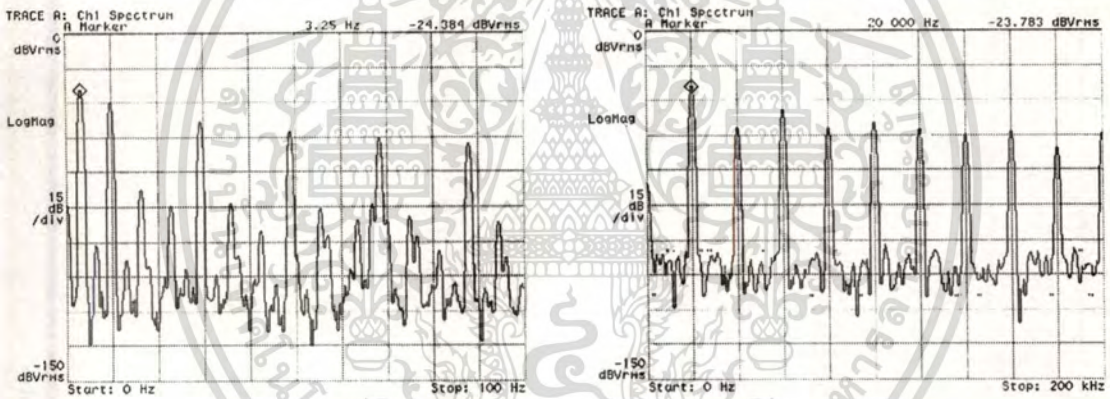
(ก) (ข)

รูปที่ 5.23 ฮาร์โมนิคในช่วงความถี่ต่ำและความถี่สูงของสัญญาณที่สร้างจากวิธี Alternating zero sequence



(ก) (ข)

รูปที่ 5.24 ฮาร์โมนิคในช่วงความถี่ต่ำและความถี่สูงของสัญญาณที่สร้างจากวิธี Symmetric sequence

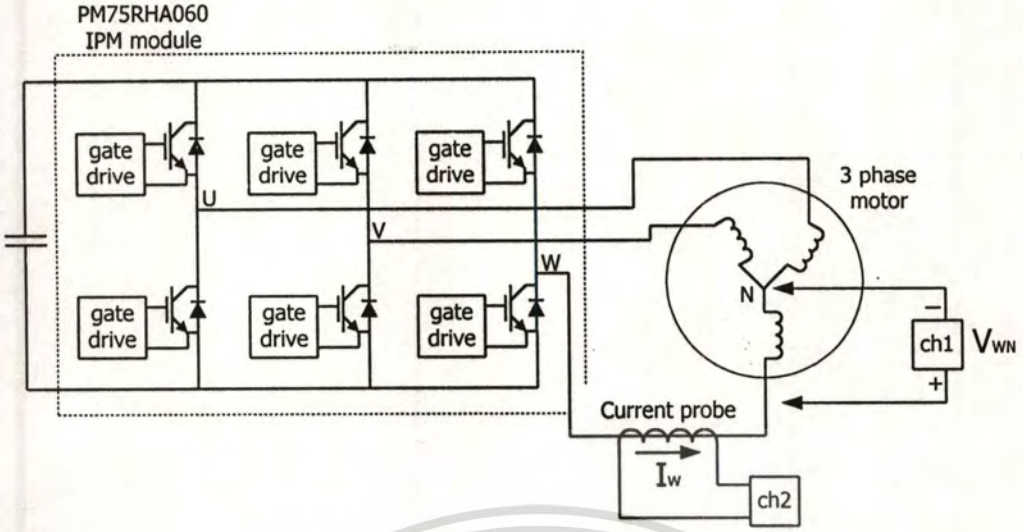


(ก) (ข)

รูปที่ 5.25 ฮาร์โมนิคในช่วงความถี่ต่ำและความถี่สูงของสัญญาณที่สร้างจากวิธี Bus clamped

5.5 ผลการนำสัญญาณ PWM ไปขับชุดอินเวอร์เตอร์

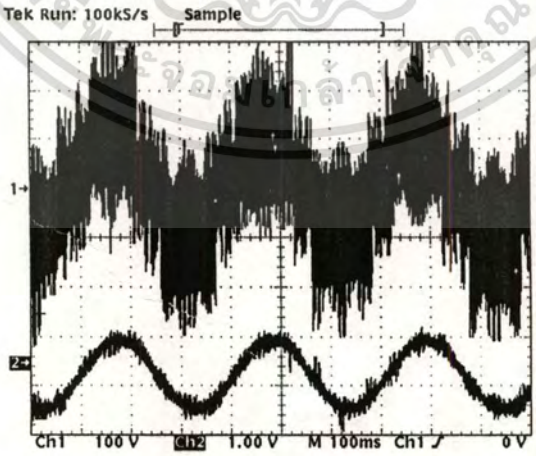
เมื่อนำสัญญาณที่กำเนิดจากวงจรรวม FPGA ไปขับเคลื่อนมอเตอร์เหนี่ยวนำแบบ 3 เฟส ที่ถูกต่อแบบ star ผ่านชุดอินเวอร์เตอร์ที่ออกแบบไว้ แล้ววัดสัญญาณดังรูปที่ 5.26 โดยสัญญาณในช่องที่ 1 วัดแรงดันเฟสของมอเตอร์ ส่วนสัญญาณในช่องที่ 2 วัดกระแส โดยใช้ Current probe ของ Tektronix รุ่น A622 โดยตั้งสเกลของการวัดกระแสไว้ที่ 100 mV/A



รูปที่ 5.26 การวัดสัญญาณแรงดันเฟสและกระแสของวงจรทดลอง

ผลการทดลองในรูปที่ 5.27 เป็นรูปแรงดันเฟสและกระแสของมอเตอร์ โดยกำหนดเงื่อนไขในการสร้างสัญญาณดังนี้

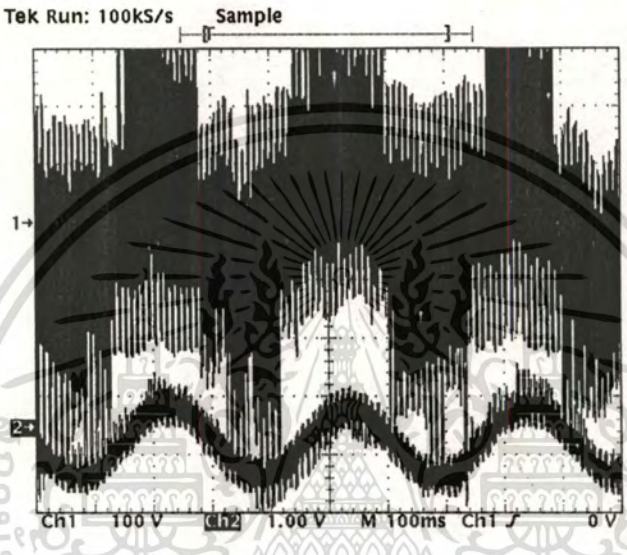
- ตัวแปร SV = 1 (เลือกใช้วิธี Alternating zero vector sequence)
- ตัวแปร T = 100 (ความถี่สวิตชิ่งเท่ากับ 10 กิโลเฮิรตซ์)
- ตัวแปร M = 900 (มอดูเลชั่นอินเด็กเท่ากับ 0.9)
- Dt = 10 (เดดไทม์เท่ากับ 5 ไมโครวินาที)
- ตัวแปร Rpt = 0 และ St = 1 จึงสร้างสัญญาณที่มีความถี่หลักมูลเท่ากับ 3.25 เฮิรตซ์



รูปที่ 5.27 แรงดันเฟสและกระแสของมอเตอร์ที่สร้างจากวิธี Alternating zero vector sequence โดยแรงดันเฟสในช่องที่ 1 ใช้สเกล 100 V/div, กระแสในช่องที่ 2 ใช้สเกล 100 mV/A

ผลการทดลองในรูปที่ 5.28 เป็นรูปแรงดันเฟสและกระแสของมอเตอร์ โดยกำหนดให้

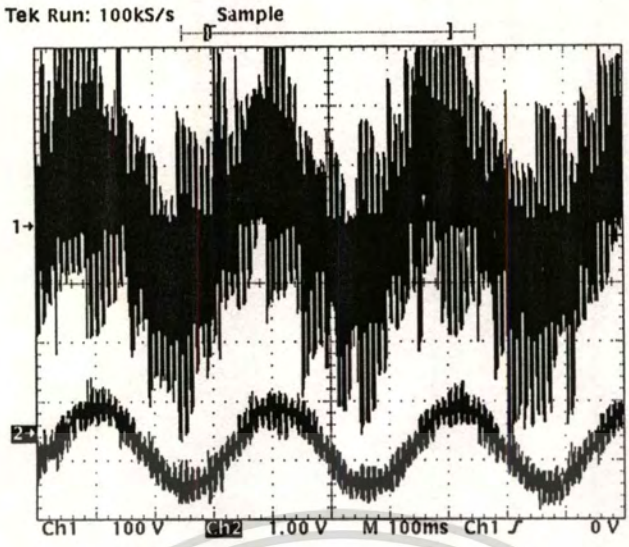
- ตัวแปร SV = 2 (เลือกใช้วิธี Symmetric sequence)
- ตัวแปร T = 100 (ความถี่สวิตชิ่งเท่ากับ 20 กิโลเฮิรตซ์)
- ตัวแปร M = 900 (มอดูเลชั่นอินเด็กเท่ากับ 0.9)
- Dt = 10 (เดดไทม์เท่ากับ 5 ไมโครวินาที)
- ตัวแปร Rpt = 0 และ St = 1 จึงสร้างสัญญาณที่มีความถี่หลักมูลเท่ากับ 3.25 เฮิรตซ์



รูปที่ 5.28 แรงดันเฟสและกระแสของมอเตอร์ที่ได้จากวิธี Symmetric sequence
แรงดันเฟสในช่องที่ 1 ใช้สเกล 100 V/div, กระแสในช่องที่ 2 ใช้สเกล 100 mV/A

ผลการทดลองในรูปที่ 5.29 เป็นรูปแรงดันเฟสและกระแสของมอเตอร์ โดยกำหนดให้

- ตัวแปร SV = 3 (เลือกใช้วิธี Bus clamped)
- ตัวแปร T = 100 (ความถี่สวิตชิ่งเท่ากับ 20 กิโลเฮิรตซ์)
- ตัวแปร M = 900 (มอดูเลชั่นอินเด็กเท่ากับ 0.9)
- Dt = 10 (เดดไทม์เท่ากับ 5 ไมโครวินาที)
- ตัวแปร Rpt = 0 และ St = 1 จึงสร้างสัญญาณที่มีความถี่หลักมูลเท่ากับ 3.25 เฮิรตซ์



รูปที่ 5.29 แรงดันเฟสและกระแสของมอเตอร์ที่ได้จากวิธี Bus clamped
แรงดันเฟสในช่องที่ 1 ใช้สเกล 100 V/div, กระแสในช่องที่ 2 ใช้สเกล 100 mV/A

บทที่ 6
วิเคราะห์และสรุปผลการทดลอง

6.1 วิเคราะห์ผลการทดลอง

จากการวิเคราะห์ผลของ THD และความสูญเสียจากการสวิตชิ่งใน [1], [3], [4] และ [5] แสดงให้เห็นว่าสัญญาณ PWM ที่ให้ผลของ THD มีค่าสูงจะให้ผลของความสูญเสียจากการสวิตชิ่งมีค่าต่ำ ตารางที่ 6.1 เป็นการเปรียบเทียบลักษณะของสเปซเวคเตอร์มอดูเลชั่นทั้ง 3 รูปแบบที่ได้นำเสนอในวิทยานิพนธ์ฉบับนี้

ตารางที่ 6.1 เปรียบเทียบลักษณะของสเปซเวคเตอร์มอดูเลชั่นทั้ง 3 รูปแบบ

	รูปแบบการมอดูเลท		
	Alt	Sym	Bus
จำนวนครั้งของการสวิตชิ่งต่อคาบเวลาการสุ่ม(Ts)	3	6	4
ความถี่สวิตชิ่ง	1/2Ts	1/Ts	1/Ts
จำนวนสถานะของ State machine ที่ใช้ในการออกแบบ	36	42	30

สัญญาณที่สร้างจากวิธี Alternating zero vector ซึ่งใน [1] เรียกว่า DI จะใช้เวคเตอร์ศูนย์ คือ U_0 และ U_7 สลับกันในแต่ละคาบเวลาการสุ่ม จึงมีผลให้คาบเวลาการสวิตชิ่งเป็นสองเท่าของคาบเวลาการสุ่ม ซึ่งจากการทดลองได้กำหนดคาบเวลาการสุ่มของสัญญาณทั้ง 3 วิธีไว้เท่ากันที่ 50 ไมโครวินาที ดังนั้นความถี่สวิตชิ่งของวิธี Alternating zero vector จะเป็น 10 กิโลเฮิรตซ์ แต่ความถี่สวิตชิ่งของอีกสองวิธีจะเป็น 20 กิโลเฮิรตซ์ และจากผลการวัดสเปคตรัมของสัญญาณที่สร้างจากวิธี Alternating zero vector พบว่าจะเกิดฮาร์โมนิคขึ้นที่ค่าครึ่งหนึ่งของความถี่การสุ่ม (ดังรูปผลการทดลองที่ 5.23) ซึ่งจากผลข้อนี้ ใน [5] ได้ทำการวิเคราะห์เปรียบเทียบค่า THD ของทั้งสามวิธีได้ผลว่า THD ของวิธี Alternating zero vector จะมีค่ามากกว่าที่ได้จากอีก 2 วิธี โดยเฉพาะอย่างยิ่งในช่วงที่มอดูเลชั่นอินเด็กมีค่าสูงๆ แต่อย่างไรก็ตามวิธี Alternating zero vector จะมีจำนวนครั้งของการสวิตชิ่งต่อ 1 คาบเวลาการสุ่มต่ำที่สุด ดังนั้นความสูญเสียจากการสวิตชิ่งจึงต่ำกว่าอีก 2 วิธี [5] และเนื่อง

จากความสมมาตรของสัญญาณที่สร้างจากวิธี Symmetric sequence วิธีนี้จึงมีค่า THD ต่ำที่สุด [4] แต่เนื่องจากจำนวนครั้งของการสวิตชิ่งของวิธี Symmetric sequence มีค่าสูงกว่าอีก 2 วิธี ดังนั้น ความสูญเสียจากการสวิตชิ่งจึงสูงที่สุด นอกจากนี้จำนวนสถานะของสเตทแมชชีนของวงจรลอจิกของวิธี Symmetric sequence มีค่าสูงสุดมีผลให้วงจรลอจิกที่สร้างโดยวิธีนี้มีขนาดใหญ่ที่สุด

จากการทดลองนำสัญญาณ PWM ที่กำเนิดได้จากวงจรรวม FPGA ไปขับเคลื่อนมอเตอร์เหนี่ยวนำ 3 เฟส 1 แรงม้า 4 โพล 220 โวลต์ 3.5 แอมป์ ที่ถูกต่อแบบสตาร์ ผ่านชุดอินเวอร์เตอร์ที่มีอัตราทนแรงดันไฟตรง 600 โวลต์ ทนกระแสสูงสุด 75 แอมป์ พบว่า

1. สามารถเปลี่ยนแปลงความถี่สวิตชิ่งของอินเวอร์เตอร์ได้ โดยในการออกแบบใช้คริสตอลโมดูลความถี่ 2 เมกกะเฮิร์ตเป็นฐานเวลาของการสวิตชิ่ง ดังนั้นจึงกำหนดความถี่สวิตชิ่งต่ำสุดได้ที่ 1 กิโลเฮิร์ต และสูงสุดได้ที่ 200 กิโลเฮิร์ต นอกจากนี้การเปลี่ยนความถี่ของคริสตอลโมดูลเป็นค่าอื่นๆ ก็จะทำให้สามารถเปลี่ยนแปลงความถี่สวิตชิ่งต่ำสุดและสูงสุดได้
2. สามารถปรับเปลี่ยนค่าความถี่หลักมูลของสัญญาณได้ และถ้ามีการเลือกใช้ค่าของตัวแปร Rpt, Step และ Ts อย่างเหมาะสม จะทำให้กำหนดความถี่หลักมูลเป็นค่าใดๆ ก็ได้
3. สามารถปรับเปลี่ยนค่าเวลาเดดไทม์ได้ โดยค่าในตัวแปร Dt ที่กำหนดลงไปนั้นจะสัมพันธ์กับความถี่ของคริสตอลโมดูลที่ใช้เป็นฐานเวลาของระบบ
4. สามารถปรับเปลี่ยนดัชนีการมอดูเลทได้ ซึ่งจากการทดลองพบว่าเมื่อกำหนดมอดูเลชั่นอินเด็กค่าต่ำๆ จะทำให้แรงบิดของมอเตอร์มีค่าน้อย และถ้าเพิ่มมอดูเลชั่นอินเด็กให้มากขึ้น แรงบิดของมอเตอร์ก็จะเพิ่มขึ้นตามไปด้วย

6.2 สรุปผลการทดลอง

ในวิทยานิพนธ์นี้ ได้ออกแบบวงจรรวมเพื่อใช้ในการกำเนิดสัญญาณ PWM เพื่อนำไปขับเคลื่อนมอเตอร์เหนี่ยวนำ 3 เฟสผ่านชุดอินเวอร์เตอร์ โดยสามารถปรับเปลี่ยนพารามิเตอร์ต่างๆ ของสัญญาณ PWM ได้แม้ในขณะกำลังใช้งานอยู่ จึงมีความยืดหยุ่นในการใช้งานสูง โดยได้เลือกใช้วงจรรวมประเภท FPGA ในการสร้างวงจรลอจิก ซึ่งจากขั้นตอนในการออกแบบวงจร การจำลองผลการทำงาน ความยืดหยุ่นในการใช้งาน ตลอดจนความถี่สวิตชิ่งที่สูงกว่าการกำเนิดสัญญาณ PWM โดยใช้ไมโครโปรเซสเซอร์ จึงน่าจะสรุปได้ว่าการนำวงจรรวม FPGA มาใช้ในการกำเนิดสัญญาณ PWM น่าจะมีข้อดีกว่าการกำเนิดโดยใช้ไมโครโปรเซสเซอร์ในหลายๆ ด้าน ข้อเสียเพียงประการเดียวของการออกแบบวงจรลอจิกโดยใช้วงจรรวม FPGA ก็คือราคาที่ค่อนข้างแพงของอุปกรณ์ FPGA เมื่อเทียบกับราคาของไมโครโปรเซสเซอร์ในปัจจุบัน แต่เมื่อพิจารณาถึงความเร็วในการออกแบบจนกระทั่งได้เป็นวงจรต้นแบบ(Prototype) รวมไปถึงการเปลี่ยนรูปแบบการใช้งานจาก FPGA เป็นวงจรรวมแบบ ASIC ที่

มีต้นทุนในการผลิตต่ำถ้ามีการผลิตเป็นจำนวนมาก พบว่าการออกแบบวงจรรวมโดยใช้ FPGA ก็เป็นอีกทางเลือกหนึ่งที่น่าสนใจในทางอุตสาหกรรม

6.3 แนวทางในการพัฒนาต่อไป

วงจรรวมที่ทำการออกแบบนี้มีจุดประสงค์เพื่อให้ง่ายต่อการใช้งาน จึงออกแบบให้วงจรรวมนี้เชื่อมต่อกับคีย์บอร์ดและตัวแสดงผลแบบ 7 เซ็กเมนต์ เพื่อกำหนดค่าพารามิเตอร์ต่างๆ ของสัญญาณ การพัฒนาวงจรรวมนี้ต่อไปในอนาคตมีได้หลายแนวทาง อาจทำได้โดยการออกแบบวงจรควบคุมต่างๆ เพิ่มเข้าไป เช่น วงจรควบคุมกระแส หรือวงจรชดเชยผลจากเดดไทม์ เป็นต้น ก็จะทำให้วงจรรวมใช้งานได้หลากหลายมากขึ้น

เอกสารอ้างอิง

[1] Victor R. Stefanovic, and Slobodan N. Vukosavic, " Space vector PWM Voltage control with optimized switching strategy," *IEEE IAS-1992 Ann. Meeting*, pp 1025-1033.

[2] H.W. van der Broeck, H.C. Skudelny, and G.V.Stanke, "Analysis and Realization of a Pulse Width Modulation Based on Voltage Space Vectors", *IEEE Trans. On Ind. App.*, vol 24, no. 1 , pp. 142-150, Feb. 1988.

[3] D.G.Holmes, "The significance of Zero Space Vector for Carrier Based PWM Schemes," *Proc.IEEE-IAS 1995 Ann.* Meeting, pp. 2451-2458.

[4] J.Holtz, "Pulse Width Modulation – A survey," *IEEE Trans. on Ind. Elec.,* vol. 39, no. 5, pp. 410-420, Oct. 1992.

[5] V.Himamshu Prasad, Dushan Boroyevich and Stephen Dubovsky, "Comparison of high frequency PWM algorithms for a Voltage Source Inverter," *Proc VPEC 1996 Ann. Meet,* pp.115-122.

[6] J.T. Boys and P.G.Handley,"Harmonic Analysis of Space Vector Modulated PWM Waveforms," *IEE proc.*,vol. 137, Pt. B, pp. 197-204, July 1990.

ภาคผนวก ก.

โปรแกรมภาษา AHDL ที่ใช้ในการออกแบบ

โปรแกรมของส่วน Input_kbd.tdf

```
INCLUDE "lpm_mult.inc";
CONSTANT bounce_number = 20;
SUBDESIGN input_kbd
(
row_in[4..1], kbd_clk       :INPUT;
col_out[4..1]               :OUTPUT;
svclk, mclk, tclk, stepclk,
dtclk, rptclk               :OUTPUT;
sum[10..1]                  :OUTPUT;
100seg[7..1], 10seg[7..1],
1seg[7..1],abcff_clk        :OUTPUT;
)

VARIABLE
amult : lpm_mult WITH (LPM_WIDTHA=4,
LPM_WIDTHB=7, LPM_WIDTHS=4,
LPM_WIDTHP=11);
bmult : lpm_mult WITH (LPM_WIDTHA=4,
LPM_WIDTHB=4, LPM_WIDTHS=4,
LPM_WIDTHP=8);
col_out[4..1], cnt[2..1], count_bounce[5..1],
key_inFF[4..1]              :DFF;
100ff[4..1], 10ff[4..1], 1ff[4..1],
sumff[10..1]                :DFF;
100a[10..1], 10b[8..1], 1c[4..1],
abc[10..1]                  :NODE;
key_inff_clk, abcff_clk, sum_clk,
key[4..1], bounce_chk       :NODE;
keyout[4..1], dec[4..1], reset_ss,
clear_abcff, chk            :NODE;
ss : MACHINE WITH STATES (ss0, ss1);
st : MACHINE WITH STATES (st0, st1, st2,
st3);
BEGIN

-- SCAN AND DECODE KEYBOARD =>
KeyOut1-Keyout15 --
col_out[].clk = kbd_clk;
col_out[] = (col_out[3..1],col_out[3..1]==0);
cnt[].clk = kbd_clk;
cnt[] = cnt[]+1;

IF row_in[1] THEN key[]=(0,0,cnt[])+0;
ELSIF row_in[2] THEN key[]=(0,0,cnt[])+4;
ELSIF row_in[3] THEN key[]=(0,0,cnt[])+8;
ELSIF row_in[4] THEN key[]=(0,0,cnt[])+12;
ELSE key[]=0;
END IF;

reset_ss = (row_in[4] # row_in[3] # row_in[2] #
row_in[1]);
key_inff_clk = !kbd_clk & reset_ss;
key_inFF[].clk = key_inff_clk;
key_inFF[].d = key[];
keyout[] = key_inFF[];

-- Check Interval Time --
ss.clk = kbd_clk ;
ss.reset = reset_ss;
count_bounce[].clk = kbd_clk;
count_bounce[].clrn = !reset_ss;
CASE ss IS
WHEN ss0 =>
count_bounce[] = count_bounce[] + 1;
IF count_bounce[] >= bounce_number THEN
        bounce_chk = !kbd_clk;
        chk = b"1";
        ss = ss1;
END IF;
WHEN ss1 =>
```

```
bounce_chk = b"0";
        chk = b"0";
END CASE;

-- DECODE KeyOut to DECIMAL_NUMBER --
  TABLE
keyout[] => dec[];
0       => 10   ;
1       => 1    ;
2       => 2    ;
3       => 3    ;
4       => 15   ;
5       => 4    ;
6       => 5    ;
7       => 6    ;
8       => 14   ;
9       => 7    ;
10      => 8    ;
11      => 9    ;
12      => 13   ;
13      => 11   ;
14      => 0    ;
15      => 12   ;
END TABLE;

-- Store in reg4b x 3 for calculation --
abcff_clk= (bounce_chk & (dec[]<10));
100ff[].clk= abcff_clk;
10ff[].clk= abcff_clk;
1ff[].clk  = abcff_clk;
100ff[].clrn= !clear_abcff;
10ff[].clrn= !clear_abcff;
1ff[].clrn = !clear_abcff;
100ff[].d = 10ff[].q;
10ff[].d  = 1ff[].q;
1ff[].d = dec[];
st.clk = kbd_clk;
st.reset  = reset_ss;
CASE st IS
WHEN st0 =>
        IF chk == 0 THEN st = st0;
        ELSE IF dec[] >= 10 THEN st = st1;
                ELSE st = st0;
                END IF;
        END IF;
WHEN st1 =>
        sum_clk = !kbd_clk;
        st = st2;
WHEN st2 =>
        CASE dec[] IS
        WHEN 15 => mclk = !kbd_clk;
        WHEN 14 => tclk = !kbd_clk;
        WHEN 13 => stepclk = !kbd_clk;
        WHEN 12 => dtclk = !kbd_clk;
        WHEN 11 => svclk = !kbd_clk;
        WHEN 10 => rptclk = !kbd_clk;
        END CASE;
        st = st3;
WHEN st3 =>
        clear_abcff = b"1";
        st = st3;
END CASE;

-- Calculation : a x 100 + b x 10 + c : using 2
mults--

sumff[].clk = sum_clk;
amult.dataa[] = 100ff[];
amult.datab[] = 100;
100a[] = amult.result[9..0];
```

```
bmult.dataa[] = 10ff[];
bmult.datab[] = 10;
10b[] = bmult.result[];
1c[] = 1ff[];
abc[] = 100a[10..1] + (0,0,10b[]) +
(0,0,0,0,0,0,1c[]);
sumff[].d = abc[];
sum[] = sumff[];

-- DECODE to 7-Segment Display --
TABLE
100ff[]   =>  100seg[];
1         =>  b"0110000";
2         =>  b"1101101";
3         =>  b"1111001";
4         =>  b"0110011";
5         =>  b"1011011";
6         =>  b"1011111";
7         =>  b"1110000";
8         =>  b"1111111";
9         =>  b"1111011";
0         =>  b"1111110";
END TABLE;

TABLE
10ff[]    => 10seg[];
1         =>  b"0110000";
2         =>  b"1101101";
3         =>  b"1111001";
4         =>  b"0110011";
5         =>  b"1011011";
6         =>  b"1011111";
7         =>  b"1110000";
8         =>  b"1111111";
9         =>  b"1111011";
0         =>  b"1111110";
END TABLE;

TABLE
1ff[]     =>  1seg[];
1         =>  b"0110000";
2         =>  b"1101101";
3         =>  b"1111001";
4         =>  b"0110011";
5         =>  b"1011011";
6         =>  b"1011111";
7         =>  b"1110000";
8         =>  b"1111111";
9         =>  b"1111011";
0         =>  b"1111110";
END TABLE;

END;
```

โปรแกรม Gen_adr10.tdf

```
-- This sub design is for generateing address
(adr[10..1]) to LPMROM1 and LPMROM2 and
sector number(sec[3..1]) for gen_pattern
subdesign. --
-- So output adr[] is from 0 to 1023 ,sector is
from 1 to 6. --

SUBDESIGN gen_adr10
(
step[10..1], clk          : INPUT;
adr[10..1], sector[3..1]  : OUTPUT;
)
VARIABLE
adr_count[10..1]          : DFF;
sector_number[3..1]       : DFF;
```

```
temp[11..1]                : DFF;
sector_clk                 : node;
ss : machine with states (s0, s1);

BEGIN
adr_count[].clk = !clk;
temp[].clk = clk;
ss.clk = clk;
sector_number[].clk = sector_clk;
temp[] = (0,adr_count[10..1]) + (0,step
[10..1]);
adr_count[10..1] = temp[10..1];
adr[] = adr_count[];

case ss  is
when  s0 =>
if temp[] >= 1024 then
sector_clk = !clk;

        ss = s1;
else ss = s0;
end if;
when s1 =>
if temp[11]== b"0" then ss = s0;
        else ss = s1;
end If;
end case;
IF sector_number[] < 6 THEN
sector_number[] = sector_number[] + 1;
ELSE    sector_number[] = 1;
END IF;
sector[] = sector_number[];
END;
```

โปรแกรม CAL_TIME.TDF

```
-- CLK provided to this module must be grater
than 200ns  when implement on FELX10K20
in UP1 Education Board   --
INCLUDE "lpm_mult.inc";
CONSTANT WIDTHA = 10;
CONSTANT WIDTHB = 10;
SUBDESIGN cal_time
(clk, cal, ld                      : INPUT;
M[WIDTHA..1], T[WIDTHB..1]         :INPUT;
Mema[WIDTHA..1], Memb[WIDTHB..1] :
INPUT;
ta[10..1], tb[10..1], tz[10..1]  : OUTPUT;)
VARIABLE
mult : lpm_mult WITH
(LPM_WIDTHA=WIDTHA
,LPM_WIDTHB=WIDTHB ,
LPM_WIDTHS=WIDTHA
,LPM_WIDTHP=WIDTHA + WIDTHB);
aff[10..1], bff[10..1], MTff[10..1]    : DFF ;
ta[10..1], tb[10..1], tz[10..1]        : DFF ;
ss : MACHINE WITH STATES (s0, s1, s2, s3,
s4);
a[10..1], b[10..1], mt[10..1]          : NODE ;
clka, clkb, clkmt                      : NODE ;

BEGIN
aff[].clk = clka;
bff[].clk = clkb;
MTff[].clk = clkmt;
ss.clk = clk;
ta[].clk = !ld;
tb[].clk = !ld;
tz[].clk = !ld;
MTff[].d = mt[];
aff[].d = a[];
```

```
bff[].d = b[];
CASE ss IS
WHEN s0 =>
IF cal THEN  ss = s1;ELSE ss = s0;
END IF;
WHEN s1 =>
mult.dataa[] = M[];
mult.datab[] = T[];
mt[] = mult.result[19..10];
clkmt = !clk;
ss = s2;
WHEN s2 =>
mult.dataa[] = MTff[];
mult.datab[] = Mema[];
IF mult.result[19..10] <= 2 THEN a[] = 2;
ELSE a[] = mult.result[19..10];
END IF;
clka = !clk;
ss = s3;
WHEN s3 =>
mult.dataa[] = MTff[];
mult.datab[] = Memb[];
IF mult.result[19..10] <= 2 THEN b[] = 2;
ELSE b[] = mult.result[19..10];END IF;
clkb = !clk;
ss = s4;
WHEN s4 =>
IF !cal THEN ss=s0; END IF;
END CASE;

ta[].d = aff[];
tb[].d = bff[];
tz[].d = (T[])-(aff[]+bff[]);
END;
```

โปรแกรมส่วน alt1.tdf

```
-- Alternating Zero Scheme --

SUBDESIGN alt1
(
ta[10..1], tb[10..1], tz[10..1]: INPUT ;
clk, sector[3..1], reset      : INPUT;
abc[3..1], check              :OUTPUT;
)
VARIABLE
cnt[10..1] : DFF;
sout : machine of bits (abc[3..1])
with states
( s0, s1=b"100",s2=b"110",s3=b"111",
s4=b"110",s5=b"100", s6=b"000",s7=b"110",
s8=b"010", s9=b"000",s10=b"010",
s11=b"110", s12=b"111",s13=b"010",
s14=b"011", s15=b"111", s16=b"011",
s17=b"010", s18=b"000",s19=b"011",
s20=b"001", s21=b"000", s22=b"001",
s23=b"011", s24=b"111",s25=b"001",
s26=b"101", s27=b"111", s28=b"101",
s29=b"001", s30=b"000",s31=b"101",
s32=b"100", s33=b"000", s34=b"100",
s35=b"101", s36=b"111");

BEGIN
sout.clk = clk; cnt[].clk = clk;
sout.reset = reset;
CASE sout IS
WHEN s0 =>
sout=s1; cnt[]=1; check=b"1";
WHEN s1 =>
IF cnt[] == ta[] THEN
sout=s2; cnt[]=1;
ELSE  cnt[] = cnt[]+1; END IF;
```

```
WHEN s2 =>
IF cnt[] == tb[] THEN sout=s3; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s3 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 2 THEN sout = s7;
        ELSE sout = s4; END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s4 =>
IF cnt[] == tb[] THEN sout=s5; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s5 =>
IF cnt[] == ta[] THEN sout=s6; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s6 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 2 THEN sout = s10;
        ELSE sout = s1;  END IF;  cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s7 =>
IF cnt[] == ta[] THEN sout=s8; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s8 =>
IF cnt[] == tb[] THEN sout=s9; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s9 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 3 THEN sout = s13;
        ELSE sout = s10;
        END IF;
        cnt[]=1;
ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s10 =>
IF cnt[] == tb[] THEN
sout=s11; cnt[]=1;
ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s11 =>
IF cnt[] == ta[] THEN  sout=s12; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s12 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 3 THEN sout = s16;
        ELSE sout = s7; END IF;cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s13 =>
IF cnt[] == ta[] THEN sout=s14; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s14 =>
IF cnt[] == tb[] THEN sout=s15; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s15 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 4 THEN sout = s19;
        ELSE sout = s16;END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s16 =>
IF cnt[] == tb[] THEN sout=s17; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s17 =>
IF cnt[] == ta[] THEN sout=s18; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s18 =>
check = b"1";
```

```
IF cnt[] == tz[] THEN
        IF sector[] == 4 THEN sout = s22;
        ELSE sout = s13;END IF;  cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s19 =>
IF cnt[] == ta[] THEN sout=s20; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s20 =>
IF cnt[] == tb[] THEN sout=s21; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s21 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 5 THEN sout = s25;
        ELSE sout = s22;  END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s22 =>
IF cnt[] == tb[] THEN sout=s23; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s23 =>
IF cnt[] == ta[] THEN sout=s24; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s24 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 5 THEN sout = s28;
        ELSE sout = s19; END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s25 =>
IF cnt[] == ta[] THEN sout=s26; cnt[]=1;
ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s26 =>
IF cnt[] == tb[] THEN sout=s27; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s27 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 6 THEN sout = s31;
        ELSE sout = s28;END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s28 =>
IF cnt[] == tb[] THEN sout=s29; cnt[]=1;
        ELSE    cnt[] = cnt[]+1;END IF;
WHEN s29 =>
IF cnt[] == ta[] THEN sout=s30; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s30 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 6 THEN sout = s34;
        ELSE sout = s25; END IF;cnt[]=1;
        ELSE    cnt[] = cnt[]+1;END IF;
WHEN s31 =>
IF cnt[] == ta[] THEN sout=s32; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s32 =>
IF cnt[] == tb[] THEN  sout=s33; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s33 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 1 THEN sout = s1;
        ELSE sout = s34; END IF;cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s34 =>
IF cnt[] == tb[] THEN sout=s35; cnt[]=1;
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s35 =>
IF cnt[] == ta[] THEN sout=s36; cnt[]=1;
```

```
ELSE    cnt[] = cnt[]+1; END IF;
WHEN s36 =>
check = b"1";
IF cnt[] == tz[] THEN
        IF sector[] == 1 THEN sout = s4;
        ELSE sout = s31; END IF;
        cnt[]=1;
ELSE    cnt[] = cnt[]+1;
END IF;


END CASE;


END;
```

โปรแกรม Sym2.tdf

```
-- Symmetric sequence scheme clk period
must be grater than 80ns!!! --

SUBDESIGN sym2
(
ta[10..1], tb[10..1], tz[10..1]      : INPUT ;
clk, sector[3..1], reset             :INPUT;
abc[3..1], check                     : OUTPUT;
time_remain[10..1]                   : OUTPUT;
tz_new[10..1]                        : OUTPUT;
)
VARIABLE
cnt[10..1] : DFF;  -- count time ta tb tz --
t[10..1]                 : node;
time_remain[10..1]       : node;
tz_new[10..1]            : node;
sout : machine of bits (abc[3..1]) with states
(s0, s1=b"000", s2=b"100",  s3=b"110",
s4=b"111", s5=b"110", s6=b"100", s7=b"000",
8=b"000",s9=b"010",s10=b"110",s11=b"111",
s12=b"110",s13=b"010",s14=b"000",
s15=b"000",s16=b"010",s7=b"011",
s18=b"111",s19=b"011",s20=b"010",
s21=b"000", s22=b"000",s23=b"001",
s24=b"011",s25=b"111",s26=b"011",
s27=b"001",s28=b"000",s29=b"000",
s30=b"001",s31=b"101",s32=b"111",
s33=b"101",s34=b"001",s35=b"000",
s36=b"000",s37=b"100",s38=b"101",
s39=b"111",s40=b"101",s41=b"100",
s42=b"000" );

BEGIN
sout.clk = clk; cnt[].clk = clk;
sout.reset = reset;

t[] = ta[] + tb[] + tz[];
time_remain[] = t[] - ((ta[10..2],0) + (tb
[10..2],0) + (0,tz[10..2]) + (0,tz[10..3],0));
tz_new[] = (0,tz[10..2]) + time_remain[10..1];

CASE sout IS
WHEN s0 =>
sout = s1;
check=b"1";cnt[]=1;

-- This state machine is for sector 1
WHEN s1 =>
IF cnt[] == (0,0,tz[10..3]) THEN
sout=s2; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s2 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s3; cnt[]=1;
```

```
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s3 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s4; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s4 =>
IF cnt[] == tz_new[] THEN
sout=s5; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s5 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s6; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s6 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s7; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s7 =>
check=b"1";
IF cnt[] == (0,0,tz[10..3]) THEN
IF sector[] == 2 THEN sout = s8;
ELSE sout = s1; END IF;
cnt[] = 1;  ELSE cnt[] = cnt[]+1;
END IF;

-- This state machine is for sector 2
WHEN s8 =>
IF cnt[] == (0,0,tz[10..3]) THEN
sout=s9; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s9 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s10; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s10 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s11; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s11 =>
IF cnt[] == tz_new[] THEN
sout=s12; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s12 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s13; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s13 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s14; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s14 =>
check=b"1";
IF cnt[] == (0,0,tz[10..3]) THEN
IF sector[] == 3 THEN sout = s15;
ELSE sout = s8; END IF;cnt[] = 1;
ELSE cnt[] = cnt[]+1;END IF;

-- This state machine is for sector 3
WHEN s15 =>
IF cnt[] == (0,0,tz[10..3]) THEN
sout=s16; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s16 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s17; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s17 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s18; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
```

```
WHEN s18 =>
IF cnt[] == tz_new[] THEN
sout=s19; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s19 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s20; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s20 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s21; cnt[]=1;ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s21 =>
check=b"1";
IF cnt[] == (0,0,tz[10..3]) THEN
IF sector[] == 4 THEN sout = s22;
ELSE sout = s15; END IF;cnt[] = 1;
ELSE cnt[] = cnt[]+1;END IF;

-- This state machine is for sector 4
WHEN s22 =>
IF cnt[] == (0,0,tz[10..3]) THEN
sout=s23; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s23 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s24; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s24 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s25; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s25 =>
IF cnt[] == tz_new[] THEN
sout=s26; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s26 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s27; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s27 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s28; cnt[]=1;ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s28 =>
check=b"1";
IF cnt[] == (0,0,tz[10..3]) THEN
IF sector[] == 5 THEN sout = s29;
ELSE sout = s22; END IF;
cnt[] = 1;  ELSE cnt[] = cnt[]+1;
END IF;
-- This state machine is for sector 5
WHEN s29 =>
IF cnt[] == (0,0,tz[10..3]) THEN
sout=s30; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s30 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s31; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s31 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s32; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s32 =>
IF cnt[] == tz_new[] THEN
sout=s33; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s33 =>
IF cnt[] == (0,tb[10..2]) THEN
```

```
sout=s34; cnt[]=1; ELSE cnt[] = cnt[]+1;
END IF;
WHEN s34 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s35; cnt[]=1;ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s35 =>
check=b"1";
IF cnt[] == (0,0,tz[10..3]) THEN
IF sector[] == 6 THEN sout = s36;
ELSE sout = s29; END IF;cnt[] = 1;
ELSE cnt[] = cnt[]+1;END IF;

-- This state machine is for sector 6
WHEN s36 =>
IF cnt[] == (0,0,tz[10..3]) THEN
sout=s37; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s37 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s38; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s38 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s39; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s39 =>
IF cnt[] == tz_new[] THEN
sout=s40; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s40 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s41; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s41 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s42; cnt[]=1;ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s42 =>
check=b"1";
IF cnt[] == (0,0,tz[10..3]) THEN
IF sector[] == 1 THEN sout = s1;
ELSE sout = s36; END IF;cnt[] = 1;
ELSE cnt[] = cnt[]+1;END IF;
END CASE;
END;
```

โปรแกรม Bus3

```
-- Bus clamping Scheme  EVEN 60 degree --

SUBDESIGN bus3
(
ta[10..1], tb[10..1], tz[10..1]      : INPUT ;
clk, sector[3..1], reset             : INPUT;
abc[3..1], check                     :OUTPUT;
time_remain[2..1]                    : OUTPUT;
tz_new[10..1]                        : OUTPUT;

)
VARIABLE
cnt[10..1]              :DFF;
time_remain[2..1]       : node;
tz_new[10..1]           : node;

sout : machine of bits (abc[3..1])
with states
( s0, s1=b"100",s2=b"110",
s3=b"111", s4=b"110",  s5=b"100",
 s6=b"110",  s7=b"010",s8=b"000",
s9=b"010",s10=b"110",s11=b"010",
s12=b"011",s13=b"111", s14=b"011",
```

```
s15=b"010",s16=b"011", s17=b"001",
s18=b"000",s19=b"001", s20=b"011",
s21=b"001", s22=b"101", s23=b"111",
s24=b"101",s25=b"001",s26=b"101",
s27=b"100", s28=b"000", s29=b"100",
s30=b"101" );

BEGIN
sout.clk = clk; cnt[].clk = clk;
sout.reset = reset;

time_remain[] = (0,ta[1]) + (0,tb[1]);
tz_new[] = (tz[10..1]) +
(0,0,0,0,0,0,0,0,time_remain[2..1]);

CASE sout IS
WHEN s0 =>
sout = s1; check=b"1";cnt[]=1;
WHEN s1 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s2; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s2 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s3; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s3 =>
IF cnt[] == tz_new[] THEN sout=s4; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;
WHEN s4 =>
IF cnt[] == (0,tb[10.:2]) THEN
sout=s5; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s5 =>
check = b"1";IF cnt[] == (0,ta[10..2]) THEN
IF sector[] == 2 THEN  sout = s6;
ELSE sout = s1; END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;

WHEN s6 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s7; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s7 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s8; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s8 =>
IF cnt[] == tz_new[] THEN
sout=s9; cnt[]=1; ELSE    cnt[] = cnt[]+1;
END IF;
WHEN s9 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s10; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s10 =>
check = b"1";
IF cnt[] == (0,ta[10..2]) THEN
IF sector[] == 3 THEN sout = s11;
ELSE sout = s6; END IF; cnt[]=1;
ELSE    cnt[] = cnt[]+1;END IF;

WHEN s11 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s12; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s12 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s13; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
```

```
WHEN s13 =>
IF cnt[] == tz_new[] THEN
sout=s14; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s14 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s15; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s15 =>
check = b"1";
IF cnt[] == (0,ta[10..2]) THEN
IF sector[] == 4 THEN
sout = s16;ELSE sout = s11; END IF;
cnt[]=1; ELSE     cnt[] = cnt[]+1;
END IF;
WHEN s16 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s17; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s17 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s18; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s18 =>
IF cnt[] == tz_new[] THEN
sout=s19; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s19 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s20; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s20 =>
check = b"1";
IF cnt[] == (0,ta[10..2]) THEN
IF sector[] == 5 THEN sout = s21;
ELSE sout = s16; END IF; cnt[]=1;
ELSE     cnt[] = cnt[]+1;END IF;
WHEN s21 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s22; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s22 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s23; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s23 =>
IF cnt[] == tz_new[] THEN
sout=s24; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s24 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s25; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s25 =>
check = b"1";
IF cnt[] == (0,ta[10..2]) THEN
IF sector[] == 6 THEN sout = s26;
ELSE sout = s21; END IF; cnt[]=1;
ELSE     cnt[] = cnt[]+1;END IF;
WHEN s26 =>
IF cnt[] == (0,ta[10..2]) THEN
sout=s27; cnt[]=1;
ELSE     cnt[] = cnt[]+1;END IF;
WHEN s27 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s28; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s28 =>
IF cnt[] == tz_new[] THEN
sout=s29; cnt[]=1; ELSE   cnt[] = cnt[]+1;
```

```
END IF;
WHEN s29 =>
IF cnt[] == (0,tb[10..2]) THEN
sout=s30; cnt[]=1; ELSE   cnt[] = cnt[]+1;
END IF;
WHEN s30 =>
check = b"1";
IF cnt[] == (0,ta[10..2]) THEN
IF sector[] == 1 THEN sout = s1;
ELSE sout = s26; END IF; cnt[]=1;
ELSE     cnt[] = cnt[]+1;END IF;
END CASE;
END;
```

โปรแกรม Deadtime.tdf

```
SUBDESIGN deadtime
(
clk, dtime[8..1], in : INPUT;
out                  : OUTPUT;
)

VARIABLE
ss : MACHINE WITH STATES (ss0, ss1);
cnt[8..1] :          DFF;

BEGIN
ss.clk = clk;
cnt[].clk = clk;

IF dtime[] == 0 THEN out = in ;
ELSE
CASE ss IS
WHEN ss0 =>
IF in == 1 THEN  cnt[] = cnt[] + 1;
END IF;
IF cnt[] == dtime[] THEN
ss = ss1 ; out = b"1";
ELSE
ss = ss0; END IF;

WHEN ss1 =>
IF in == 1 THEN  out = b"1";
ELSE ss = ss0; out = b"0";END IF;
END CASE;
END IF;
END;
```

ภาคผนวก ข.

ผลงานวิจัยที่ได้รับการตีพิมพ์

PROCEEDINGS

1999 IEEE International Symposium on Intelligent Signal Processing and Communication Systems

Signal Processing and Communications Beyond 2000

December 8-10, 1999

IGBT Inverter with Vector Modulation Implemented on FPGA chip

Theerayod Wiangtong*, Worranart Sangchai**, Pichit Lumyong**
*Electronic Department, Mahanakorn University of Technology
**Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang
Phone&Fax (+662) 9883655 Ext 239, Email: theerayo@mut.ac.th

Abstract

This paper presents an application of ALTERA FPGA Device, in FLEX10K family, producing Pulse Width Modulation (PWM) signals with vector modulation technique for IGBT inverter. Using a single FPGA chip for the practical implementation of the modulator, rather than system consisted of microprocessor and external memory, achieves many advantages including less power and space consuming, short design time, more speed and reliability. This designed circuit can generate PWM signal in many different frequencies, and also, the input values used to adjust output signal are able to be obtained through either 4x4 keypad or microprocessor port.

1. Introduction

The Pulse Width Modulation (PWM) Technique called " Vector Modulation", which is based on the space vector theory, is the most important development step in the last few years [2]. Although, plenty of PWM methods have been created in the past, vector modulation technique seem to be the best alternative for the three phase switching power converter [1], [4]. It provides an optimization of converter operation, reducing the commutations of the power semiconductor.

There are previous examples concerned about implementation of this technique. Firstly, discrete components; principally including ROM and counter, are used together. The switching times are already calculated and then stored in ROM, resulting in the simple circuit. However, the frequency of motor speed is difficult to control and the system is not compact. Secondly, the fast microprocessor is employed to calculate the switching times. Nevertheless, to obtain higher switching frequency, faster processor is necessary resulting in high cost, and also, a long time to develop software in a new processor structure. Moreover, microprocessor controlled by software is not suitable for the switching power circuit, which has plenty of noise, resulting in the high risk to collapse.

This paper presents a different and better solution for the practical implementation of the modulator, achieving many advantages. This work is designed by using ALTERA development tools, namely MAX+PLUS II, and completed in a single FPGA chip with no external memory because of the large memory block, called EAB (Embedded Array Block) which contains 2048 bits each, buried in FLEX10K device family. This design method provides benefits described as the following.

1. The whole systems are implemented in only a single chip consequently the circuit is very compact.
2. System on FPGA chip provides more reliability because it does not need any software to control.
3. Faster design and verification and design change without penalty.

2. Principle of Space Vector PWM

The power circuit of the 3-phase inverter, shown in Fig. 1, consists of 6 transistors T1,..,T6 with their anti-parallel diodes. Transistors in the same leg are controlled inversely by using pulse signal. In this paper, vector modulation technique is exploited to generate that gating signals called PWM.

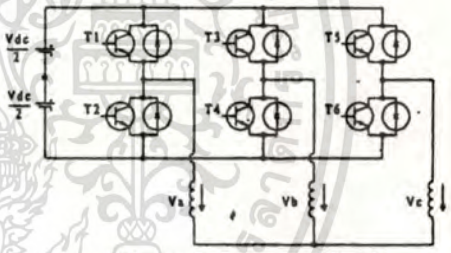


Figure 1. Inverter Circuit

The operational principle of the space vector PWM is clearly explained by representing the voltage vectors (V0,..,V7) as shown in Fig. 2.

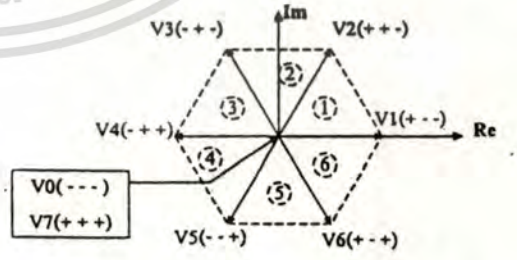


Figure 2. Voltage vectors generated by the inverter

For example, to produce vector V1, the on-off state of transistors in each leg is symbolized by the nomenclature (+ - -). There are six active vectors (V1,..,V6) and two zero vectors (V0, V7). Generally, the zero vector Vz which has no amplitude seems to be redundancy. However, for space vector

modulation, we need to add this zero vector with other two vectors in order to maintain the switching time.

Vector modulation technique is based on the application to the load of zero vector and the two adjacent vectors to produce the new vector V^*. As seen on Fig. 3, it is considered that the vector V^* is constant and stationary during the sampling interval T. The vector V^* located in sector 1 is composed of zero vector Vz and active vector Vx, Vy which are the fraction of vector Va = V1(+ - -), Vb = V2(+ + -) . Though, this diagram shows only vector located in sector 1, it can be also illustration of the rests. For example, when vector V^* is located in sector 2 then Va = V2(+ + -) and Vb = V3(- + -).

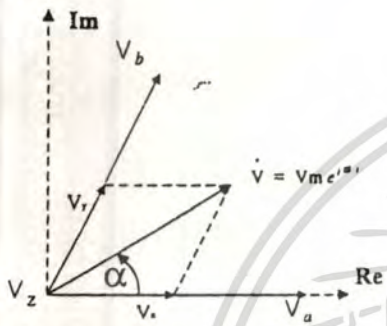


Figure 3. Vector diagram

The vector V^* in sector 1 can be expressed as the following equations.

$$V^* = (V_a \frac{ta}{T} + V_b \frac{tb}{T} + V_z \frac{to}{T}) \quad \text{.....(1)}$$

where ta, tb and to are the respective on-times to generate voltage vector Va, Vb and Vz. These times have the following restriction

$$ta + tb + to = T = \frac{Ts}{2} \quad \text{.....(2)}$$

Due to Vz = 0 and V^* = Vx + Vy, the new relation is

$$V^* = Vx + Vy = (V_a \frac{ta}{T} + V_b \frac{tb}{T}) \quad \text{.....(3)}$$

Hence

$$ta = \frac{|Vx|}{|V_a|} T$$
$$tb = \frac{|Vy|}{|V_b|} T \quad \text{.....(4)}$$

In Fig. 3, applying trigonometric relations, the amplitude of vector Vx and Vy are obtained.

$$Vx = V_m \cos(\alpha) - \frac{1}{\sqrt{3}} \sin(\alpha)$$
$$Vy = \frac{2}{\sqrt{3}} V_m \sin(\alpha) \quad \text{.....(5)}$$

Considering that |Va| = |Vb| = $2/3V_D$ and substitute Vx, Vy in equation (4), the following equations are accomplished.

$$ta = \frac{3}{2} M (\cos(\alpha) - \frac{1}{\sqrt{3}} \sin(\alpha)) T$$
$$tb = \sqrt{3} M \sin(\alpha) T$$
$$to = T - ta - tb \quad \text{.....(6)}$$

Where M = Modulation Index = Vm/V_D
T = ta + tb + to = Ts/2 (Ts = switching time)
α = vector angle in each sector $0 < \alpha < 60°$

3.The proper switching sequence

One of the most important of inverter circuit is the number of commutations and switching losses in the power semiconductor that can be minimized by using the redundancy property of the zero vector V0 (- - -) and V7 (+ + +). For example, in sector 1, the appropriate switching sequence should be seen in Fig. 4. It can be observed that vector $V1^*$ is built from vector Va = V1 (+ - -) in duration time of ta, vector Vb = V2 (+ + -) in duration time of tb. In the rest of the time T (to), it must be the zero vector Vz which can be either V0 (- - -) and V7 (+ + +). In order to switch only one phase, zero vector V7 is selected and then it continues creating the next vector $V2^*$ by using vector Vb firstly instead of Va.

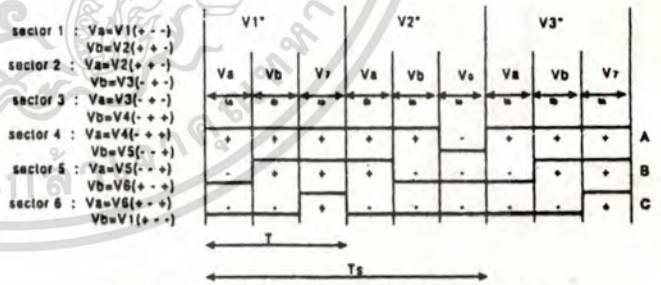


Figure 4. Switching sequence optimization

At connection between two neighbor sector, for example between sector 1 and sector 2, vector in sector 1 before changing to the next sector has switching pattern like either vector $V1^*$ or vector $V2^*$. To achieve one phase switching, we have to beware of the beginning vector in sector 2. The beginning vector in sector 2 must be Va = V2(+ + -) if the last vector in sector 1 has the same switching pattern as $V1^*$, vise versa, the beginning vector in sector 2 must be Vb = V3(- + -) if the last vector in sector 1 has the same switching pattern as $V2^*$.

The switching sequence of producing vector rotates in a cycle (6 sectors) can be described as the state machine shown in Fig. 5.

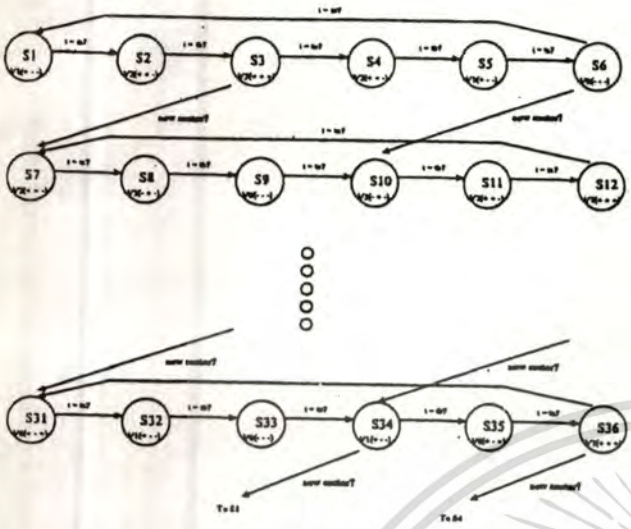


Figure 5. State machine of switching sequence

4. Dead Time

In ideal conditions, the gating signals to the power switches of same leg of PWM inverter should be complementary. Nevertheless, the turn-off time of a power switch is usually longer than its turn-on time, and, therefore, an appropriate delay time must be inserted between these two gating signals as shown in Fig. 6.

The length of this delay time is usually about 1.5 ~ 2 times the maximum turn-off time [5]. An adjustable delay-time is included in this design work which greatly facilitates its practical application.

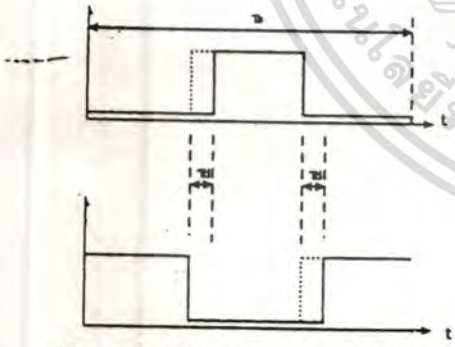


Figure 6. PWM signals with delay time

5. Circuit Design

In order to generate the voltage vector, one of the most important factors are switching times obtained by the equations (6). In the design, all of the function blocks designed by using Altera High Description Language (AHDL) combine together to create the system as shown in the Fig.7. From the block diagram, vector with constant amplitude determined by M rotates in the complex plane with constant speed determined by step and T.

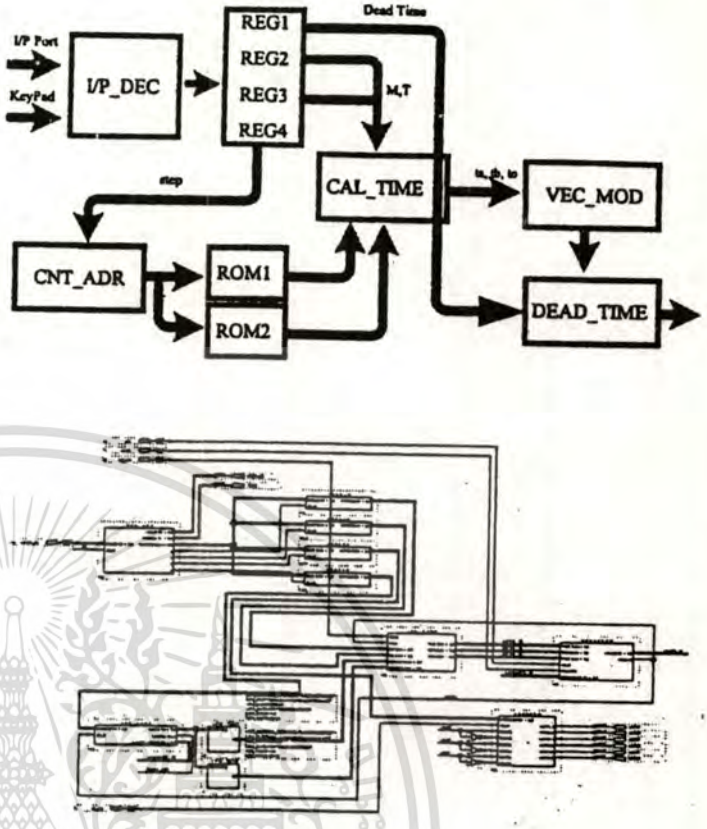


Figure 7. System block diagram and design circuit

The input parameters, including M, T, Dead time, Frequency, are collected from either 4x4 keypad or microprocessor interfacing port through **I/P_DEC** and then transfer to registers (**REG1-REG4**). The data contain in those registers are consequently used for producing the PWM signal. **CNT_ADR** generates data addressing corresponding to frequency (step) for **ROM1** and **ROM2**, and also, determine position of the current vector in which sector in order that the signals are generated correctly in **VEC_MOD**. **ROM1** and **ROM2** contain values of $\frac{3}{2}(\cos\alpha - \frac{1}{\sqrt{3}}\sin\alpha)$ and $\sqrt{3}\sin\alpha$ in one sector ($0 < \alpha < 60$) respectively. Those values are scaled to fit in memory which has a size of 10-bit width and 512 depth. Hence, the step value of counter can be in the range of 1-500 providing different 500 frequencies. In FLEX10K devices, the large memory block, called EAB (Embedded Array Block), is suitably utilized to contain those values, therefore, the external memory is nonessential. **CAL_TIME** calculates value of ta, tb, to and then forwards to **VEC_MOD** to generate signal, corresponding to switching pattern in each sector, to control IGBT circuit. There are 36 states (6 states a sector), previously shown in Fig. 5, in a cycle of rotation depend on position of the current vector.

Before sending the PWM to control transistors in three-phase IGBT inverter, **DEAD_TIME** function block is used to delayed with the value of dead-time which is able to be

predetermined by user in order to prevent the short-through phenomena as mention earlier.

6. Experiment and Results

This design work was implemented in FLEX10K20RC240 chip which is composed of 1152 logic cells (around 20,000 gate counts) as well as 6 EAB blocks. In the experiment, the PWM signals with fundamental frequency of 32 Hz and 50 Hz are produced by using modulation index of 0.8 and switching frequency of 10 KHz. Fig. 8 provided by simulator indicates the signal at pin abc1-abc3.

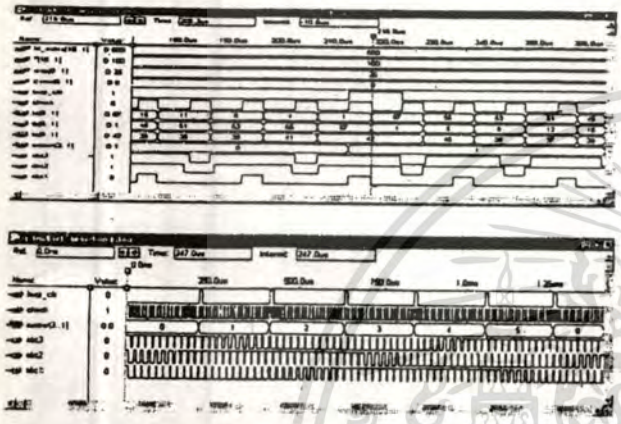
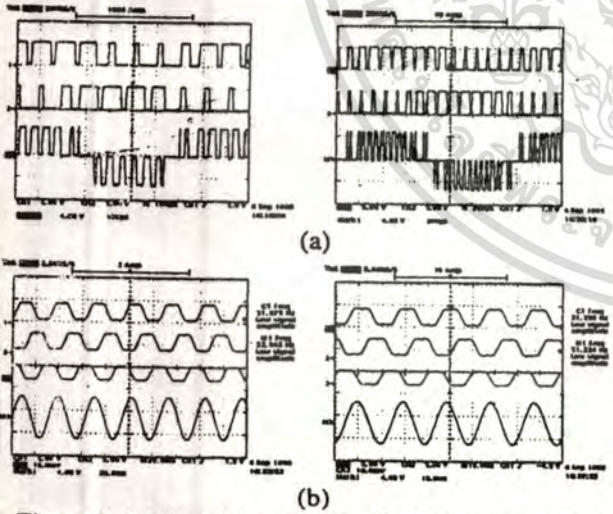


Figure 8. PWM signals from simulation

Notice that, the switching frequency in the simulation results is faster than that in practical in order to reduce the simulation time.



(a)

(b)

Figure 9. (a) Switching pattern (b) Line-to-GND and line-to-line voltage

The PWM signals from FPGA chip after implementation are shown in Fig. 9. As the results, Fundamental frequency, switching frequency, magnitude, and dead-time are programmable through 4x4 keypad and it can be practically used for driving inductive motor.

7. Summary

Space vectors are an outstanding analytical alternative to describe the behaviour of three-phase inverter. Optimization of state sequence achieves switching loss reduction. In this work, the logic gates are utilized around 75% together with 5 EAB blocks for memory. From the experiment results, it has been verified that this inverter generates practically sinusoidal load voltages with varieties of magnitude, dead time, and frequency. Furthermore, this way obtains several advantages for ac drive in terms of low cost, short design time, high performance, compact and will extensively use for motor drive in the future.

8. References

[1] H.W.van de Broeck,H.C.Skudelny and G.V.Stanke "Analysis and Realization of a Pulse Width Modulation Based on Voltage Space Vectors", IEEE Trans, Ind. Appl.vol 24,pp 142-150,Jan/Feb 1987.

[2] J.Rodriquez ,E. Wiechmann ,J. Holtz "IGBT Inverter with Vector Modulation", IEEE International Symposium on Industrial Electronics, p. 131-6, 1994

[3] Shoji Fukuda, Yoshitaka Iwaji Hirokazu Hasegawa "PWM Technique for Inverter with Sinusoidal output current" IEEE Trans. Vol 5, No.1 ,Jan 1990.

[4] J. Holtz, "Pulse Width Modulation - A Survey", Proceedings of the PESC'92, Espana 1992

[5] Ying-Yu Tzou, Hau-Jean Hsu "FPGA Realization of Space-Vector control IC for Three-Phase PWM Inverters" IEEE Trans. on Power Electronics, Vol.12, No.6, November 1997.

[6] Seung-Gi Jeong, Min-Ho Park "The Analysis and Compensation of Dead-Time Effects in PWM Inverter" IEEE Trans. on Industrial Electronic, Vol.38, No.2, April 1991

[7] Altera Corporation: '1998 Data Book', January 1998.

ประวัติผู้เขียน

นายวรนารถ แสงฉาย เกิดเมื่อวันที่ 26 เมษายน พ.ศ. 2513 ที่จังหวัดน่าน สำเร็จการศึกษาวิศวกรรมศาสตร์บัณฑิต (ไฟฟ้า) จากสถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ปีการศึกษา 2534 เข้าทำงานในตำแหน่งอาจารย์ประจำคณะวิศวกรรมศาสตร์ วิทยาลัยมหานคร ในปี พ.ศ. 2535 และปัจจุบันดำรงตำแหน่งอาจารย์ประจำภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีมหานคร