การออกแบบวงจรสังเคราะห์ความถี่แบบดิจิตอลโดยใช้คอดิกอัลกอลิทึม ที่สามารถโปรแกรมได้ ด้วย FPGA

PROGRAMMABLE DIGITAL FREQUENCY SYNTHESIZER BASED ON CORDIC ALGORITHM USING FPGA

โดย

นายธงชัย สุบิน

นายพฤทธิ์ เกษรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

การออกแบบวงจรสังเคราะห์ความถี่แบบดิจิตอลโดยใช้คอดิกอัลกอลิทึม ที่สามารถโปรแกรมได้ ด้วย FPGA

PROGRAMMABLE DIGITAL FREQUENCY SYNTHESIZER BASED ON CORDIC ALGORITHM USING FPGA

โดย

นายธงชัย สุบิน 44015010

นายพฤทธิ์ เกษรัตน์ 44015018

อาจารย์ที่ปรึกษา

รศ.ดร. กอบชัย เดชหาญ

อาจารย์ ศรวัฒน์ ชิวปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

ปริญญานิพนธ์ปีการศึกษา 2546

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การออกแบบวงจรสังเคราะห์ความถี่แบบดิจิตอล**
โดยใช้คอดิกอัลกอลิทึมที่สามารถโปรแกรมได้ด้วย FPGA
PROGRAMMABLE DIGITAL FREQUENCY SYNTHESIZER
BASED ON CORDIC ALGORITHM USING FPGA

ผู้จัดทำ

1. นายธงชัย สุบิน 44015010

2. นายพฤทธิ์ เกษรัตน์ 44015018

(.....) อาจารย์ที่ปรึกษา

รศ.ดร.กอบชัย เดชหาญ

(.....) อาจารย์ที่ปรึกษา

อาจารย์ ศรวัฒน์ ชิวปรีชา

การออกแบบวงจรสังเคราะห์ความถี่แบบดิจิตอล โดยใช้คอดิกอัลกอลิทึมที่สามารถโปรแกรมได้ด้วย FPGA

Programmable Digital Frequency Synthesizer Based on CORDIC Algorithm using FPGA

โดย นายธงชัย สุบิน 44015010

นายพฤทธิ์ เกษรัตน์ 44015018

อาจารย์ที่ปรึกษา รศ.ดร. กอบชัย เดชหาญ

อาจารย์ศรวัฒน์ ชิวปรีชา

บทคัดย่อ

โครงงานนี้นำเสนอการออกแบบและสร้างวงจรสังเคราะห์ความถี่แบบดิจิตอล (Digital Frequency Synthesizer) สัญญาณไซน์และโคไซน์โดยใช้คอดิกอัลกอลิทึม (Cordic Algorithm) ที่สามารถกำหนดความถี่ของสัญญาณได้ โดยจะรับค่าจากคอมพิวเตอร์ ผ่านทางพอร์ตอนุกรม (Serial Port) ส่งผ่านไปยังอุปกรณ์ FPGA (Field Programmable Gate Array) ซึ่งถูกออกแบบให้เป็นวงจรกำเนิดสัญญาณดิจิตอล (Digital Oscillator) ที่ใช้โครงสร้างของคอดิกแบบลำดับโดยในการอธิบายการทำงานพฤติกรรมของวงจรด้วยภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language)

ABSTRACT

Digital Frequency Synthesizer is designed and implemented, the sinusoidal signal is generated based on CORDIC algorithms. The frequency of signal can be assigned by giving the values from computer and via serial port to FPGA. FPGA is designed to generate the digital oscillator with the parallel CORDIC structure. VHDL describes the operations of the circuit.

สารบัญ

		หน้า
บทที่ 1	บทนำ	
	1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์	1
	1.2 ความมุ่งหมายและวัตถุประสงค์ของโครงงาน	1
บทที่ 2	ทฤษฎีและหลักการ	3
	2.1 ทฤษฎีตัวกำเนิดสัญญาณไซน์แบบดิจิตอล	3
	2.2 CORDIC Algorithm	6
	2.2.1 การประยุกต์ใช้กับฟังก์ชันไฮเปอร์บอลิก 13	
	2.2.2 โครงสร้างทางฮาร์ดแวร์ของ CORDIC algorithm	15
	2.3 การเขียนภาษา VHDL	20
	2.3.1 รูปแบบการบรรยายพฤติกรรม	20
	2.3.2 การออกแบบจากบนลงล่าง	22
	2.3.3 ภาษา VHDL และ ส่วนประกอบต่างๆของภาษา	24
	2.3.3.1 หน่วยการออกแบบเอนทิตี	24
	2.3.3.2 หน่วยการออกแบบสถาปัตยกรรม	25
	2.3.3.3 หน่วยการออกแบบแพ็คเก็ต	29
	2.3.3.4 หน่วยการออกแบบโครงแบบ	30
	2.3.4 ชุดคำสั่งลำดับ (Sequential Statements)	31
	2.3.4.1. Process Statement	31
	2.3.4.2. Wait Statement	32
	2.3.4.3. IF-THEN-ELSE statement	33
	2.3.4.4. CASE Statement	33
	2.4 การส่งข้อมูลแบบอะซิงโครนัส	34
	2.4.1 ขั้นตอนการส่งข้อมูล	35
	2.4.2 ตรวจสอบความผิดพลาด	35
	2.4.3 การส่งข้อมูลแบบซิงโครนัส	36
	2.4.4 การส่งข้อมูลซิงโครนัสอักขระ	36
	2.4.5 การส่งข้อมูลซิงโครนัสบิต	37
	2.4.6 การสื่อสารแบบอนุกรมและ RS-232	38
	2.4.7 การสื่อสารแบบอนุกรม	38
	2.4.8 การสื่อสารแบบอะซิงโครนัส	39

สารบัญ (ต่อ)

	หน้า
2.4.9 มาตรฐานพอร์ตอนุกรม RS-232	41
2.4.10 คอนเน็คเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ	42
2.4.11 UART	44
2.4.12 ชนิดของ UART	45
2.5 การแปลงดิจิตอลเป็นอนาล็อก	45
2.5.1 ตัวแปลงดิจิตอลเป็นอนาล็อกแบบทีทีแอล	45
บทที่ 3 การออกแบบและการสร้าง	48
3.1 การออกแบบวงจรเชิงเลขด้วยอุปกรณ์ FPGA	48
3.2 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์	49
3.2.1 การจำลองการทำงานของวงจร	50
3.2.2 การสังเคราะห์วงจร	50
3.2.3 การแบ่งวงจร	50
3.2.4 การวางอุปกรณ์	51
3.2.5 การเชื่อมต่อสัญญาณ	51
3.2.6 การโปรแกรมอุปกรณ์ FPGA	51
3.3 การออกแบบ	51
3.3.1 การออกแบบ โครงสร้างของ CORDIC Algorithm	52
3.4 การเขียนโปรแกรมวิชวลเบสิกเพื่อใช้งานพอร์ตอนุกรม	54
3.4.1 คอนโทรล MSComm	54
3.4.2 ComPort	54
3.4.3 Setting	55
3.4.4 PortOpen	56
3.5 Direct Digital Frequency Synthesizer	57
3.5.1 การออกแบบ และการสร้าง	59
3.6 การสร้างและการออกแบบ CORDIC	61
3.7 วงจรแปลงระดับแรงดัน	61
3.7.1 หลักการทำงาน	
3.8 วงจรแปลงระดับสัญญาณดิจิตอลเป็นอนาล็อก	62
3.9 การออกแบบวงจรและลายวงจร	63

สารบัญ (ต่อ)

	หน้า
บทที่ 4 การทดลองและผลการทดลอง	67
4.1 การจำลองการทำงานส่วนของ CORDIC ด้วย MATLAB	67
4.2 โปรแกรมการใช้งานการเชื่อมต่อผ่านพอร์ตอนุกรมโดยใช้วิชวลเบสิก	68
4.3 การออกแบบวงจรส่วนต่างๆ โดยใช้ภาษา VHDL	72
4.3.1 ส่วนของวงจร DIV 1000	72
4.3.2 ส่วนของวงจร LATCH40	73
4.3.3 ส่วนของวงจร ONEPULSE	74
4.3.4 ส่วนของวงจร SERIAL_COMMUNICATION	75
4.3.5 ส่วนของวงจร PHASE_ACCUMULATION	76
4.3.6 ส่วนของวงจร MUX_OUT	77
4.3.7 ส่วนของวงจร LFSR	78
4.3.8 ส่วนของวงจร DDS	79
4.3.8 ส่วนของวงจร CORDIC	80
4.4 ผลการทดสอบการทำงานของ DDS OSCILLATOR	81
4.4.1 การทดสอบการกำเนิดสัญญาณซายน์ ที่ความถี่ต่างๆ	81
4.4.2 การทดสอบการกำเนิดสัญญาณสามเหลี่ยม ที่ความถี่ต่างๆ	86
4.4.3 การทดสอบการกำเนิดสัญญาณสี่เหลี่ยม ที่ความถี่ต่างๆ	88
4.4.4 การทดสอบการกำเนิดสัญญาณฟันเลื่อย ที่ความถี่ต่างๆ	91
4.4.5 การทดสอบการกำเนิดสัญญาณสุ่ม ที่ความถี่ต่างๆ	95
4.7 ผลการทดสอบการทำงานของ CORDIC OSCILLATOR	96
บทที่ 5 บทสรุปและวิจารณ์	102
ภาคผนวก	
กิตติกรรมประกาศ	
บรรณานุกรม	

สารบัญรูป

หน้า

รูปที่ 2.1 โครงสร้างของตัวกำเนิดไซน์แบบตรง 4

รูปที่ 2.2 โครงสร้างของตัวกำเนิดไซน์แบบคู่ควบ 5

รูปที่ 2.3 การหมุนของเวคเตอร์ในพิกัดคาร์ทีเชียล 6

รูปที่ 2.4 ผลการกระทำของค่า σ_i ที่เกิดกับค่า θ 9

รูปที่ 2.5 แสดงตำแหน่งของเวคเตอร์ R ในแต่ละพิกัด 13

รูปที่ 2.6 โครงสร้างโดยการต่อชุด CORDIC element(CE) ลำดับกัน 16

รูปที่ 2.7 โครงสร้างแบบวนค่ากลับ 16

รูปที่ 2.8 โครงสร้างของชุดคำนวณ CORDIC หนึ่งชุด 17

รูปที่ 2.9 โครงสร้างของ CORDIC algorithm แบบวนค่ากลับ (loop) 18

รูปที่ 2.10โครงสร้างของ CORDIC algorithm แบบลำดับ (cascaded) 19

รูปที่ 2.11 แสดงขั้นตอนการออกแบบจากบนลงล่าง 23

รูปที่ 2.12 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี้ 24

รูปที่ 2.13 แสดงรูปแบบของ RS_flipflop 25

รูปที่ 2.14 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม 26

รูปที่ 2.15 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop
ตามฟังก์ชั่นบูลีน $Q = \overline{QB + R}$ และ $QB = \overline{Q + S}$ 27

รูปที่ 2.16 แสดงโครงสร้างภายในสถาปัตยกรรมของ RS_flipflop 27

รูปที่ 2.17 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะโครงสร้าง 28

รูปที่ 2.18 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะพฤติกรรม 28

รูปที่ 2.19 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะผสม 29

รูปที่ 2.20 แสดงโครงสร้างโดยทั่วไปของส่วนการประกาศแพ็กเก็ต 30

รูปที่ 2.21 แสดงโครงสร้างโดยทั่วไปของบอดี้แพ็กเก็ต 30

รูปที่ 2.22 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ 31

รูปที่ 2.23 ตัวอย่างการสื่อสารข้อมูลแบบอะซิงโครนัสจากพอร์ตอนุกรมของเครื่อง PC 36

รูปที่ 2.24 บล็อกหรือเฟรมอักขระของการส่งข้อมูลแบบซิงโครนัสอักขระ

(ก).รูปแบบของเฟรมอักขระ 37

(ข).เฟรมอักขระ 8 บิต 37

รูปที่ 2.25 เฟรมข้อมูลของการส่งข้อมูลแบบซิงโคนัส 38

รูปที่ 2.26 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม 39

รูปที่ 2.27 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส 40

สารบัญรูป (ต่อ)

หน้า

รูปที่ 2.28 การจัดขาของคอนเน็คเตอร์อนุกรมตามมารตรฐาน RS-232 ทั้งแบบ DB-9 และDB-25

(ก).คอนเน็คเตอร์อนุกรม 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์) 42

(ข).คอนเน็คเตอร์อนุกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์) 42

รูปที่ 2.29 การต่ออุปกรณ์ภายนอกกับพอร์ทอนุกรมของคอมพิวเตอร์ในลักษณะต่าง ๆ

(ก). การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ Null modem 43

(ข). การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ RS-232 โดยใช้สายเพียง 3 เส้น 43

รูปที่ 2.30 การแปลงระดับแรงดัน TTL เป็นระดับ ดีทูเอ 45

รูปที่ 2.31 ตัวแปลงดิจิตอลเป็นอนาล็อก 2R แบบ TTL 46

รูปที่ 2.32 ตัวแปลงดิจิตอลเป็นอนาล็อก 2R แบบ TTL ที่ใช้ควบคุมมอเตอร์เล็ก ๆ 47

รูปที่ 3.1 ลักษณะของ FPGA และการนำไปใช้งาน 48

รูปที่ 3.2 ขั้นตอนการออกแบบโดยใช้อุปกรณ์ FPGA 49

รูปที่ 3.3 ขั้นตอนการทำงานของ ตัวกำเนิดสัญญาไซน์ 51

รูปที่ 3.4 การออกแบบโครงสร้างของ CORDIC Algorithm 52

รูปที่ 3.5 โครงสร้างของ CORDIC ใช้คำนวณ SIN,COS 53

รูปที่ 3.6 แสดง Block Diagram โครงสร้างของวงจรสังเคราะห์ความถี่ที่ใช้หน่วยความจำเป็นค่า ขนาดของสัญญาณเอาไว้ 57

รูปที่ 3.7 วงจรสร้างสัญญาณซายน์แบบเชิงเลข 57

รูปที่ 3.8 แสดงการกำเนิดสัญญาณซายน์แบบดิจิตอล ที่มีค่า Phase Increment เท่ากับ 55 58

รูปที่ 3.9 Phase Increment เท่ากับ 55 59

รูป.3.10 โครงสร้างการกำเนิดสัญญาณแบบตรง 59

รูปที่3.11 แสดงการทำงานของวงจรกำเนิดสัญญาณ Sine และ Cosine โดยใช้ CORDIC Algorithm 61

รูปที่ 3.12 วงจรแปลงระดับแรงดัน 62

รูปที่ 3.13 วงจรแปลงระดับสัญญาณดิจิตอลเป็นอนาล็อก 62

รูปที่ 3.14 วงจรรวมของ MAX 232 กับ DAC 63

รูปที่ 3.15 ลายวงจรและการวางอุปกรณ์ 65

รูปที่ 3.16 ภาพถ่ายอุปกรณ์และวงจรใช้งานจริง 66

รูปที่ 4.1 ผลการจำลองการทำงานของ CORDIC ในการหาค่า COS θ และ SIN θ โดยใช้ MATLAB 67

รูปที่ 4.2 แสดง เมนูหลัก ของโปรแกรมการใช้งาน Digital Oscillator 68

รูปที่ 4.3 แสดงเมนู ของ Digital Cordic Oscillator 69

รูปที่ 4.4 แสดงเมนู ของ Direct Digital Synthesizer 70

รูปที่ 4.5 แสดงเมนู Project Credits 71

สารบัญรูป (ต่อ)

	หน้า
รูปที่ 4.6 สัญลักษณ์ของส่วนวงจร DIV 1000	72
รูปที่ 4.7 การจำลองการทำงานของส่วนวงจร DIV 1000	72
รูปที่ 4.8 สัญลักษณ์ของส่วนวงจร LATCH	73
รูปที่ 4.9 ผลการจำลองการทำงานของส่วนวงจร LATCH	73
รูปที่ 4.10 สัญลักษณ์ของส่วนวงจร ONEPULSE	74
รูปที่ 4.11 ผลการจำลองการทำงานของส่วนวงจร ONEPULSE	74
รูปที่ 4.12 สัญลักษณ์ของส่วนวงจร SERIAL_COMMUNICATION	75
รูปที่ 4.13 ผลการจำลองการทำงานของส่วนวงจร SERIAL_COMMUNICATION	75
รูปที่ 4.14 สัญลักษณ์ของส่วนวงจร PHASE_ACCUMULATION	76
รูปที่ 4.15 ผลการจำลองการทำงานของส่วนวงจร PHASE_ACCUMULATION	76
รูปที่4.16 สัญลักษณ์ของส่วนวงจร MUX_OUT	77
รูปที่4.17ผลการจำลองการทำงานของส่วนวงจร MUX_OUT	77
รูปที่4.18 สัญลักษณ์ของส่วนวงจร LFSR	78
รูปที่4.19 ผลการจำลองการทำงานของส่วนวงจร LFSR	78
รูปที่4.20 สัญลักษณ์ของส่วนวงจร DDS	79
รูปที่4.21ผลการจำลองการทำงานของส่วนวงจร DDS	79
รูปที่ 4.22 สัญลักษณ์ของส่วนวงจร CORDIC	80
รูปที่ 4.23 ผลการจำลองการทำงานของส่วนวงจรCORDIC	80
รูปที่ 4.24 แสดงสัญญาณซายน์ที่ความถี่ต่าง ๆ	85
รูปที่ 4.25 รูปแสดงสัญญาณสามเหลี่ยมที่ความถี่ต่าง ๆ	87
รูปที่ 4.26 แสดงสัญญาณสี่เหลี่ยมที่ความถี่ต่าง ๆ	90
รูปที่ 4.27 แสดงสัญญาณฟันเลื่อยที่ความถี่ต่าง ๆ	94
รูปที่ 4.28 แสดงสัญญาณสุ่ม	95
รูปที่ 4.29 แสดงสัญญาณซายน์ที่สร้างจาก CORDIC	101

สารบัญตาราง

หน้า

ตารางที่ 2.1 แสดงการคำนวณหาค่าของ sine และ cosine 11

ตารางที่ 2.2 แสดงการคำนวณหาค่า $\tan^{-1}(\frac{y_0}{x_0})$ 12

ตารางที่ 2.3 แสดงความสัมพันธ์ของฟังก์ชั่นไฮเปอร์บอลิกในพิกัดต่าง ๆ 14

ตารางที่ 2.4 แสดงค่า $\delta = (m,i)$, k_m ในแต่ละพิกัด 15

ตารางที่ 2.5 แสดงบิตพาริตี้ของข้อมูล 41

ตารางที่ 3.1 การหาค่ามุมจากโครงสร้าง CORDIC 53

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ปัจจุบันความก้าวหน้าทางด้านเทคโนโลยีไมโครอิเล็กทรอนิกส์ ได้รับการพัฒนาอย่างรวดเร็ว ดังที่ได้เห็นได้ว่ามีวงจรดิจิตอลอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นจากชิ้นส่วนอุปกรณ์อิเล็กทรอนิกส์จำนวนมาก ถูกนำมาประกอบกันอยู่บนแผงไฟฟ้า (Printed Circuit Board หรือ PCB) ที่มีขนาดใหญ่แต่ในปัจจุบันการออกแบบวงจรทางดิจิตอลจะเป็นการออกแบบในลักษณะที่ใช้ภาษาอธิบายพฤติกรรมของวงจร (Hardware Description Language) ที่เป็นภาษาคอมพิวเตอร์ระดับสูง (High Level Language) เช่นเดียวกับ C หรือ Pascal เรียกว่า Hardware description Language หรือ HDL ช่วยในการออกแบบ เพื่อให้การออกแบบนั้นมีประสิทธิภาพ สะดวก รวดเร็ว แล้วใช้ซอฟต์แวร์สังเคราะห์ลอจิก (Logic Synthesis Tool) แปลงให้เป็นข้อมูลที่สามารถนำโปรแกรมลงในชิพไอซีที่สามารถโปรแกรมได้ เช่น FPGA (Field Programmable Gate Array) เพื่อสร้างเป็นวงจรตามที่ต้องการ

ในปริญญานิพนธ์นี้ได้นำเอาแนวคิดของ CORDIC (Coordinate Rotation Digital Computer) Algorithm มาศึกษาการทำงาน และประยุกต์ใช้งาน กับวงจรกำเนิดสัญญาณดิจิตอล และสามารถกำเนิดสัญญาณ Sinusodial ที่มีเฟสต่างกัน 90 องศา (Sine และ Cosine) ได้ในเวลาเดียวกัน แล้วนำไปสร้างฮาร์ดแวร์ ซึ่งใช้การบรรยายพฤติกรรมการทำงานด้วยภาษา VHDL (Very High Speed Integrated Circuit Hardware Discription Language) ในการออกแบบ

ในงานด้านสื่อสารโทรคมนาคม ที่มีการใช้วงจรหรือระบบที่ต้องการที่มีเฟสต่างกัน 90 องศาไปใช้งาน เช่น วงจรมอดูเลเตอร์ แบบ Phase Shift Keying (PSK) หรือแบบ Quadrature Modulation ซึ่งสามารถนำวงจรกำเนิดสัญญาณดิจิตอลที่ใช้โครงสร้างของคอดิกมาใช้ร่วมกับวงจรเหล่านี้ได้ เพื่อลดความยุ่งยากในการออกแบบวงจรแบบเดิม จึงควรมีการศึกษาและพัฒนาอุปกรณ์กำเนิดสัญญาณดิจิตอล ที่มีเฟสและความถี่แบบเที่ยงตรงสูง เพื่อให้การทำงานของระบบมีความเที่ยงตรงสูงเช่นกัน

1.2 ความมุ่งหมายและวัตถุประสงค์ของโครงงาน

ในโครงงานกล่าวถึงการนำคอดิกอัลกอลิเทิมมาประยุกต์ให้งานร่วมกับวงจรกำเนิดสัญญาณดิจิตอล โดยในการอธิบายพฤติกรรมการทำงานของวงจรด้วยภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language) การทำโครงงานนี้มีจุดประสงค์ดังนี้

1. เพื่อศึกษาทฤษฎีและหลักการทำงานของ CORDIC Algorithm และวงจรกำเนิดสัญญาณดิจิตอล ที่มีโครงสร้างแบบคู่ควบ (Couple Form)
2. เพื่อศึกษารูปแบบการเขียนภาษาวีเอชดีแอลในการออกแบบระบบฮาร์ดแวร์ดิจิตอล ซึ่งเริ่มตั้งแต่การออกแบบ แก้ไขตรวจสอบ จำลองการทำงาน จนถึงขั้นผลิตวงจรหรือสังเคราะห์วงจร
3. เพื่อศึกษาลักษณะการทำงาน การควบคุมและฟังก์ชั่นการใช้งานต่าง ๆ ของเอฟพีจีเอ

4. เพื่อศึกษาโปรแกรมต่าง ๆ ที่ใช้ในการออกแบบฮาร์ดแวร์ดิจิตอล เช่น MAXPLUS+II
5. เพื่อศึกษาการเชื่อมต่อคอมพิวเตอร์กับเอฟพีจีเอผ่านทางพอร์ตอนุกรม
6. สามารถสร้างวงจรกำเนิดสัญญาณซายน์

บทที่2
ทฤษฎีและหลักการ

2.1 ทฤษฎีตัวกำเนิดสัญญาณไซน์แบบดิจิตอล

ตัวกำเนิดสัญญาณไซน์แบบดิจิตอลจะอยู่ในรูปแบบจำกัดของ two-pole resonator โดยที่ Complex-conjugate poles จะอยู่บน unit circle โดยเราพิจารณาที่ second-order ของระบบซึ่งมี transfer function

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.1}$$

เพื่อให้ pole ทั้งสองตัววางบนเส้นรอบวงในลักษณะ Complex Conjugate ดังนั้นจึงกำหนดค่า a_1 และ a_2 ดังสมการที่(2.2)

$$a_1 = -2r \cos \omega_0 \quad \text{และ} \quad a_2 = r^2 \tag{2.2}$$

เมื่อแทนค่า a_1 และ a_2 ลงในสมการที่ (2.2)จะทำให้ pole ทั้งสองตัวมีค่าดังสมการที่(2.3)

$$p = re^{\pm j\omega_0} \tag{2.3}$$

แต่เนื่องจากตัวกำเนิดไซน์จำเป็นต้องวาง pole ทั้งสองตัวอยู่บนเส้นรอบวงของวงกลมรัศมีหนึ่งหน่วย ดังนั้นจะต้องกำหนดค่า $r = 1$ เมื่อกำหนดให้ input ของระบบเป็น $\delta(n)$ จะได้ผลตอบสนองทาง Output ดังสมการที่ (2.4)

$$h(n) = \frac{b_0 r^n}{\sin \omega_0} u(n) \sin(n+1)\omega_0 \tag{2.4}$$

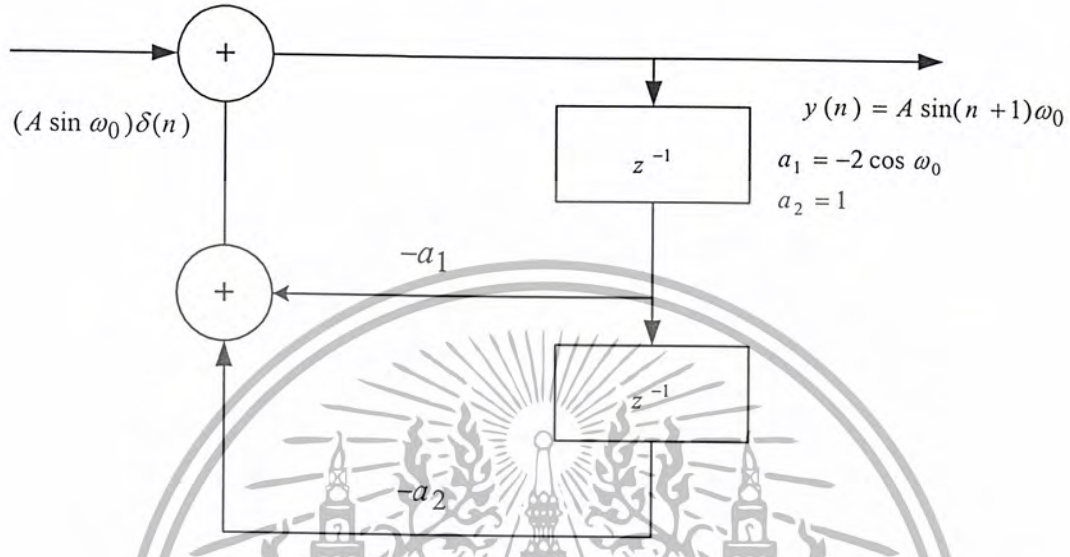
กำหนดให้$b_0 = a \sin \omega_0$และแทนค่า $r = 1$ ลงในสมการที่(2.4) จะได้ผลตอบสนอง Output ดังสมการที่ (2.5)

$$h(n) = a \sin(n+1)\omega_0 u(n) \tag{2.5}$$

ดังนั้น impulse response ของระบบ second-order ที่มี pole เป็นคู่ conjugate บน unit circle เป็น sinusoidal และเรียกระบบนี้ว่า Digital Sinusoidal Oscillator หรือ Digital Sinusoidal Generator ตัว Digital Sinusoidalเป็นส่วนประกอบพื้นฐานของ digital frequency synthesizer

Block Diagram ของ System Function ของสมการที่ (2.1) แสดงดังรูปที่ 2.1 และสามารถเขียนได้ในลักษณะของสมการ Difference equation ของระบบนี้เป็น

$$y(n) = -a_1 y(n-1) - a_2 y(n-2) + b_0 \delta(n) \tag{2.6}$$



รูปที่ 2.1 โครงสร้างของตัวกำเนิดไซน์แบบตรง

$$h(n) = a \sin(n+1)\omega_0 u(n) \tag{2.7}$$

เมื่อ parameter $a_1 = -2 \cos \omega_0$ และ $b = A \sin \omega_0$ และเงื่อนไขเริ่มต้น $y(-1) = y(-2) = 0$ เพราะฉะนั้นถ้าดูในสมการที่ (2.6) เราจะได้

$$y(0) = A \sin \omega_0$$

$$y(1) = 2 \cos \omega_0 y(0) = 2a \sin 2\omega_0 \cos \omega_0 = A \sin 2\omega_0$$

$$y(2) = 2 \cos \omega_0 in 2\omega_0 - A \sin \omega_0$$

$$= 2A \cos \omega_0 \sin 2\omega_0 - a \sin \omega_0$$

$$= A(4 \cos^2 \omega_0 \sin \omega_0 - \sin \omega_0)$$

$$= A(4 \cos^2 \omega_0 - 1) \sin \omega_0$$

$$= 3A \sin \omega_0 - 4 \sin^2 \omega_0 = A \sin 3\omega_0$$

ต่อไปเป็นการประยุกต์ใช้งานโดยหา impulse ที่ n=0 จุดประสงค์คือ เป็นการเริ่มต้น Oscillation สัญญาณ sinusoidal หลังจากนั้นจะ Oscillate ด้วยตัวเอง เพราะว่าระบบนี้เป็นระบบของ no damping

สิ่งที่เราสนใจคือการ Oscillation สัญญาณ sinusoidal ที่ได้จากระบบนี้ในสมการ(2.6)เพราะได้เงื่อนไขเริ่มต้นจากการที่ set input เป็น 0 โดยเงื่อนไขเริ่มต้น $y(-1) = 0, y(-2) = -A \sin \omega_0$ ดังนั้น zero input response ของระบบ second-order จะอธิบายโดยสมการ homogeneous difference equation

$$y(n) = -a_1 y(n-1) - a_2 y(n-2) \tag{2.8}$$

จากเงื่อนไขเริ่มต้นที่กำหนดมาแล้ว นำไปแทนในสมการ (2.6) แล้วทำการกระตุ้นด้วย impulse ในความเป็นจริง difference equation ในสมการ (2.7) สามารถหาค่าได้โดยตรงจากเอกลักษณ์ตรีโกณมิติ

$$\sin\alpha + \sin\beta = 2\sin\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right) \tag{2.9}$$

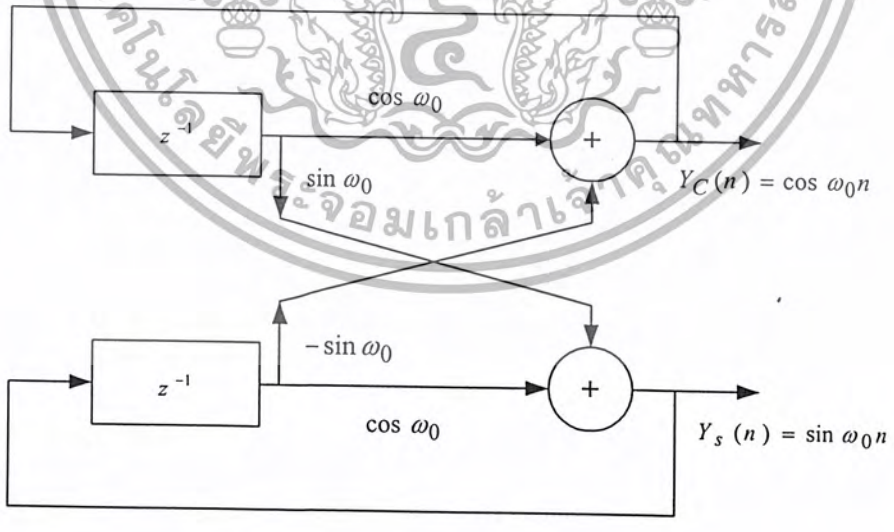
โดยกำหนดให้ $\alpha = (n+1)\omega_0$, $\beta = (n-1)\omega_0$ และ $y(n) = \sin(n+1)\omega_0$

ในการประยุกต์ใช้งานการ modulation จะใช้สัญญาณ sinusoidal สองสัญญาณที่มี phase ต่างกัน 90 องศา ซึ่งเราต้องสร้างสัญญาณ sinusoidal นี้คือ $A\sin\omega_0 n$ และ $A\cos\omega_0 n$ ซึ่งสร้างจากรูปแบบที่เรียกว่า Coupled-form oscillator ดังนั้นเราจะได้สมการจากสูตรตรีโกณมิติ

$$\cos(\alpha+\beta) = \cos\alpha\sin\beta - \sin\alpha\sin\beta$$
$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$

กำหนดให้ $\alpha = n\omega_0$, $\beta = \omega_0$

$$y_c(n) = \cos n\omega_0 u(n)$$
$$y_s(n) = \sin n\omega_0 u(n) \tag{2.10}$$



รูปที่ 2.2 โครงสร้างของตัวกำเนิดไซน์แบบคู่ควบ

ดังนั้นเราจะได้สอง Coupled difference equation

$$\begin{aligned} y_c(n) &= (\cos\omega_0)y_c(n-1) - (\sin\omega_0)y_s(n-1) \\ y_s(n) &= (\sin\omega_0)y_c(n-1) - (\cos\omega_0)y_s(n-1) \end{aligned} \tag{2.11}$$

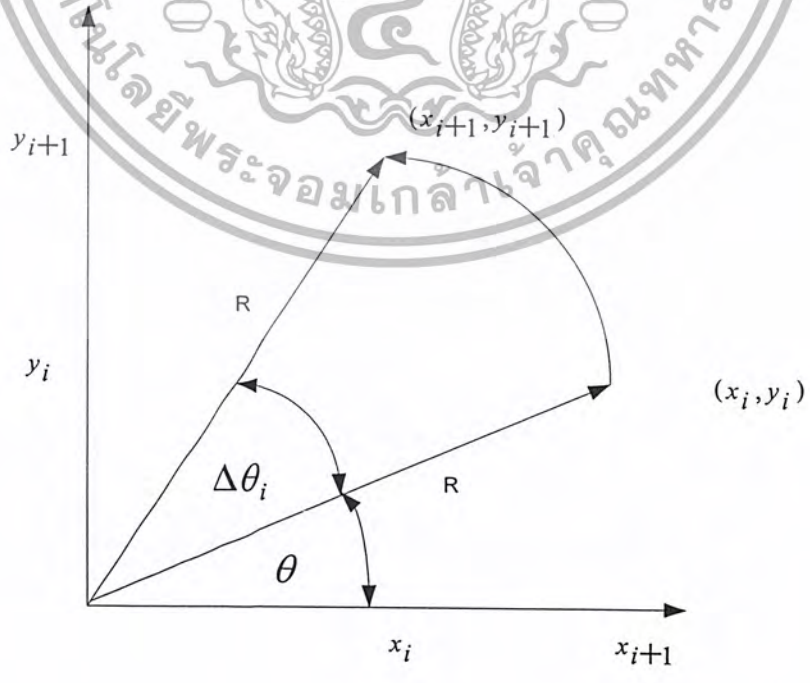
สามารถกระจายอยู่ในรูป Matrix

$$\begin{bmatrix} y_c \\ y_s \end{bmatrix} = \begin{bmatrix} \cos\omega_0 & -\sin\omega_0 \\ \sin\omega_0 & \cos\omega_0 \end{bmatrix} \begin{bmatrix} y_c(n-1) \\ y_s(n-1) \end{bmatrix} \tag{2.12}$$

โครงสร้างของ Coupled-form oscillator ที่สามารถใช้งานได้จริงแสดงดังรูปที่ 2.2 ระบบนี้จะมี 2 output ที่ไม่ต้องมี input ใด ๆ มากระตุ้น แต่เราต้องทำการกำหนดเงื่อนไขค่าเริ่มต้นที่ $y_c(-1) = A\cos\omega_0$ และ $y_s(-1) = -A\sin\omega_0$ แล้วระบบจะ oscillate ด้วยตัวเอง

2.2 CORDIC Algorithm

CORDIC (Coordinate Rotation Digital Computer) เป็นทฤษฎีของการหาค่าในฟังก์ชันในตรีโกณมิติเช่น sine, cosine, tangent, arcsine, arcsine, arctangent เป็นต้น ซึ่งจะทำการคำนวณค่าโดยวิธีการหมุนเวคเตอร์ในระบบของพิกัดคาร์ทีเซียล ซึ่งให้ค่าผลลัพธ์ของขนาดหรือมุมของของเวคเตอร์นั้นๆออกมา การคำนวณจำเป็นต้องมีการกำหนดค่าเริ่มต้นที่ค่าๆหนึ่งแล้วจึงค่อยๆทำการลดค่าหรือวนรอบในการเปลี่ยนแปลงค่าไปเรื่อยๆจนได้ค่าที่ต้องการ โดยจะพิจารณา Vector diagram ดังที่แสดงใน



รูปที่ 2.3

จากรูปที่ 2.3 กำหนดให้ R คือขนาดของเวคเตอร์

θ คือมุมของเวคเตอร์และ x, y คือตำแหน่งของเวคเตอร์ในพิกัดคาร์ทีเชียล ซึ่งจะได้ความสัมพันธ์คือ

$$\tan\theta = \frac{y_i}{x_i}$$

และ
$$\tan(\theta + \Delta\theta_i) = \frac{y_{i+1}}{x_{i+1}} \qquad (2.13)$$

จากเอกลักษณ์ของตรีโกณมิติ $\tan(x \pm y) = \dfrac{\tan x \pm \tan y}{1 \mp \tan x \tan y}$

เขียนสมการที่ (2.13) ใหม่ได้เป็น

$$\frac{y_{i+1}}{x_{i+1}} = \frac{\dfrac{y_i}{x_i} + \tan\Delta\theta_i}{1 - \left(\dfrac{y_i}{x_i}\right)\tan\Delta\theta_i}$$

นำ $\dfrac{x_i}{y_i}$ คูณเศษและส่วนด้านขวาของสมการตลอดแล้วทำการจัดรูปใหม่

$$\frac{y_{i+1}}{x_{i+1}} = \frac{y_i + x_i\tan\Delta\theta_i}{x_i - y_i\tan\Delta\theta_i} \qquad (2.14)$$

พิจารณาเฉพาะขนาดของเวคเตอร์

$$y_{i+1}^2 + x_{i+1}^2 = y_i^2 + x_i^2 \qquad (2.15)$$

จาก สมการที่ (2.14) และ (2.15) $y_i^2 + x_i^2 = x_{i+1}^2 + x_{i+1}^2\left[\dfrac{y_i + x_i\tan\Delta\theta_i}{x_i - y_i\tan\Delta\theta_i}\right]^2$

คูณตลอดด้วย $(x_i - y_i\tan\Delta\theta_i)^2$

$$\begin{aligned}(x_i - y_i\tan\Delta\theta_i)^2(y_i^2 + x_i^2) &= (x_i - y_i\tan\Delta\theta_i)^2 x_{i+1}^2 + (y_i + x_i\tan\Delta\theta)^2 x_{i+1}^2 \\ &= x_{i+1}^2\left(x_i^2 + x_i^2\tan^2\Delta\theta_i + y_i^2\tan^2\Delta\theta_i + y_i^2\right) \\ &= x_{i+1}^2(x_i^2 + y_i^2)(1 + \tan^2\Delta\theta_i)\end{aligned}$$

$$\therefore \quad x_{i+1} = \frac{x_i - y_i \tan \Delta\theta_i}{\sqrt{1 + \tan^2 \Delta\theta_i}} \tag{2.16}$$

จาก (2.14)
$$y_{i+1} = x_{i+1}\left[\frac{y_i + x_i \tan \Delta\theta_i}{x_i - y_i \tan \Delta\theta_i}\right]$$

แทน x_i+1 จาก (2.16) จะได้
$$y_i+1 = \frac{x_i - y_i \tan \Delta\theta_i}{\sqrt{1 + \tan^2 \Delta\theta_i}}\left[\frac{y_i + x_i \tan \Delta\theta_i}{x_i - y_i \tan \Delta\theta_i}\right]$$
$$= \frac{y_i - x_i \tan \Delta\theta_i}{\sqrt{1 + \tan^2 \Delta\theta_i}} \tag{2.17}$$

นำเทอมของส่วนมาพิจารณา
$$= \sqrt{1 + \tan^2 \Delta\theta_i}$$
$$= \left(\frac{\cos^2 \Delta\theta_i + \sin^2 \Delta\theta_i}{\cos^2 \Delta\theta_i}\right)^{1/2}$$
$$= \left(\frac{1}{\cos \Delta\theta_i}\right) \tag{2.18}$$

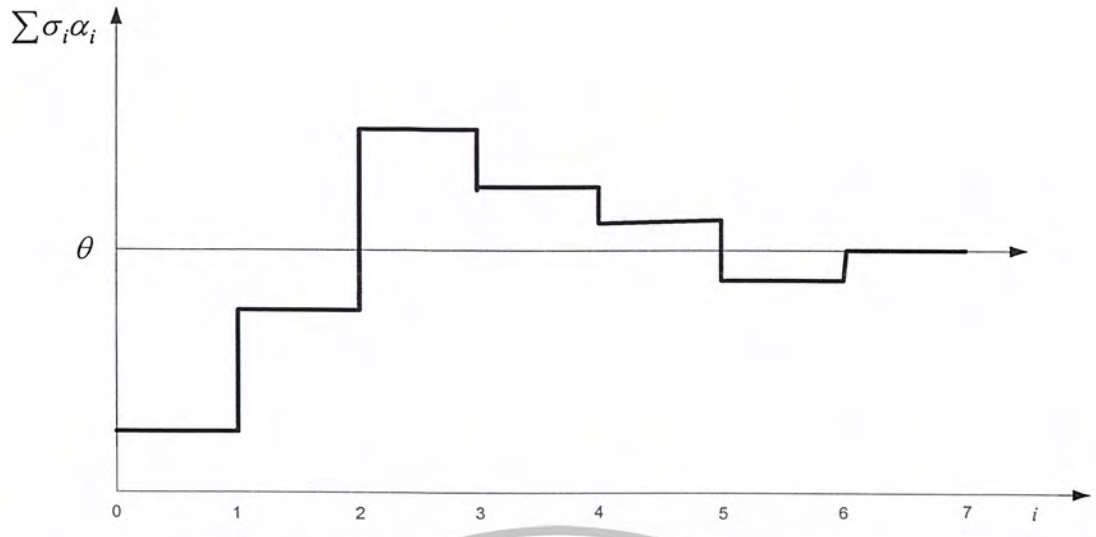
จาก (2.18) เขียนสมการ (2.16) และ (2.17) ใหม่เป็น

$$\left.\begin{aligned} x_{i+1} &= x_i \cos \Delta\theta_i - y_i \sin \Delta\theta_i \\ y_{i+1} &= y_i \cos \Delta\theta_i + x_i \sin \Delta\theta_i \end{aligned}\right\} \tag{2.19}$$

เห็นได้ว่าสามารถหาตำแหน่งของเวคเตอร์อีกตำแหน่งหนึ่งได้โดยผ่านความสัมพันธ์จากสมการนี้ ซึ่งวิธีการของ CORDIC จะถูกกำหนดค่าการหมุนโดย $\Delta\theta_i$ และถ้าหากให้ลำดับในการหมุนมีค่าเท่ากับ α_i โดยมีการหมุนทั้งหมด n ครั้งและแต่ละครั้งถูกกำหนดเครื่องหมายโดย σ_i ดังนั้นจะทำให้ได้ค่าของ $\Delta\theta_i$ ตามสมการข้างล่าง

$$\Delta\theta_i = \sum_{i=0}^{n-1} \sigma_i \alpha_i \quad ; \sigma_i \in \{-1, 1\} \tag{2.20}$$

โดยที่ σ_i จะมีการเปลี่ยนแปลงตามค่ามุมของแต่ละครั้งที่มีการหมุนเป็นตำแหน่ง α_i ซึ่งจะขึ้นอยู่กับผลของค่าผลรวมก่อนหน้านั้นเพื่อพิจารณาค่าของ $\Delta\theta_i$ ไปในทิศทางเดียวโดยเปรียบเทียบกับค่าของ $\Delta\theta_i$ ดังรูปที่ 2.4



รูปที่ 2.4 ผลการกระทำของค่า σ_i ที่เกิดกับค่า θ

จากรูปที่ 2.4 จะเห็นว่าค่าผลรวมของ α_i ($\sum \sigma_i \alpha_i$) จะมีค่าที่เป็นไปได้ทั้งช่วงที่มากกว่าและน้อยกว่าค่าของมุม θ และเมื่อจำนวนค่า i สูงขึ้นเรื่อยๆนั่นคือเกิดการหมุนของตำแหน่งเวคเตอร์จำนวนหลายครั้งขึ้นก็จะได้ผลรวมของ α_i จนเท่ากับค่าของมุม θ ซึ่งจะเห็นได้ว่า σ_i แต่ละครั้งเป็นตัวแปรสำคัญที่จะกำหนดให้ค่าผลรวมมีทิศทางเข้าหาค่าของ θ และจากวิธีการนี้เองจะได้สมการที่มาช่วยในการพิจารณาเครื่องหมาย(ค่าของ σ_i) คือ

$$z_{i+1} = z_i - \sum_{i=0}^{n-1} \sigma_i \alpha_i \tag{2.21}$$

โดยที่

$$\sigma_i \begin{cases} +1 & z_i \geq 0 \\ -1 & z_i < 0 \end{cases} \tag{2.22}$$

พิจารณาจากสมการที่ (2.19) จะเห็นว่าค่าของ $\Delta\theta_i$ ก็คือ α_i ที่ตำแหน่ง i ค่าต่างๆนั่นเองซึ่งสามารถเขียนสมการใหม่ได้เป็น

$$\left.\begin{aligned} x_{i+1} &= x_i \cos\alpha_i - \sigma_i y_i \sin\alpha_i \\ y_{i+1} &= y_i \cos\alpha_i + \sigma_i x_i \sin\alpha_i \end{aligned}\right\} \tag{2.23}$$

เพื่อให้ง่ายต่อการคำนวณจะใช้การประมาณค่าของ $\tan\alpha_i$ เทียบเคียงกับอนุกรมกำลังของ 2 ดังสมการที่ (2.24)

$$\tan\alpha_i = 2^{-i} \quad ; i = 0,1,2,\ldots,n-1 \tag{2.24}$$

จากสมการที่ (2.23) และ (2.24) จะได้ว่า

$$x_{i+1} = \cos\alpha_i \left[x_i - \sigma_i y_i \tan\alpha_i\right]$$
$$y_{i+1} = \cos\alpha_i \left[y_i + \sigma_i x_i \tan\alpha_i\right]$$

ดังนั้น

$$x_{i+1} = k_i \left[x_i - \sigma_i 2^{-i} y_i\right]$$
$$y_{i+1} = k_i \left[y_i + \sigma_i 2^{-i} x_i\right] \qquad (2.25)$$

และเมื่อ $k_i = \cos\alpha_i$ ทำการจัดอยู่ในรูปของเทอม 2^{-i} ได้ดังนี้

$$= \frac{1}{\sqrt{\frac{\cos^2\alpha_i + \sin^2\alpha_i}{\cos^2\alpha_i}}}$$
$$= \frac{1}{\sqrt{1+\tan^2\alpha_i}}$$
$$= \frac{1}{\sqrt{1+2^{-2i}}}$$

ซึ่งเมื่อกระทำการวนซ้ำหรือหมุนเวคเตอร์ไปแต่ละค่าเรื่อยๆนำค่า k_i มาแยกพิจารณาก็จะได้สัมประสิทธิ์ใหม่เป็น

$$k = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} \qquad (2.26)$$
$$\cong 0.607253$$

ดังนั้น จะพิจารณาเฉพาะเทอมของตัวแปรที่จะนำมาใช้ในการวนซ้ำจากสมการที่ (2.21) และ (2.25) จะเขียนความสัมพันธ์ใหม่ได้เป็น

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i$$
$$y_{i+1} = y_i + \sigma_i 2^{-i} x_i \qquad (2.27)$$
$$z_{i+1} = z_i - \sigma_i \tan^{-1}(2^{-i})$$

ซึ่งจากสมการที่ (2.27) นี้สามารถคำนวณหาค่าผลลัพธ์ที่ต้องการโดยการป้อน x_i, y_i และ z_i ที่ $i=0$ เป็นค่าเริ่มต้นให้ระบบโดยที่ค่าของ z_0 จะต้องอยู่ในช่วงของ $\pm\frac{\pi}{2}$ เท่านั้นซึ่งถ้าค่าของ z_0 นอกเหนือไปจากนี้อาจจะนำคุณสมบัติของตรีโกณมิติมาช่วยดังที่แสดงในสมการที่(2.28)

$$\cos(z \pm 2\pi) = \cos z \ , \sin(z \pm 2\pi) = \sin z$$
$$\cos(z - \pi) = -\cos z \ , \sin(z - \pi) = -\sin z \qquad (2.28)$$

เมื่อทำการกำหนดหรือป้อนค่าเริ่มต้นแล้วทำการคำนวณค่าออกมาก็จะได้ผลดังตัวอย่างในตารางที่ 2.1 ซึ่งกำหนดให้ $z_0 = 30^\circ$, $x_0 = k$ และ $y_0 = 0$

i	σ_i	$x_{i+1} \rightarrow \cos z_0$	$y_{i+1} \rightarrow \sin z_0$	$z_{i+1} \rightarrow 0$
		k =0.607253	0.000000	30.0000000
0	1	0.607253	0.607253	-15.00000
1	-1	0.910880	0.303627	11.565051
2	1	0.834973	0.531347	-2.471192
3	-1	0.901391	0.426975	4.653824
4	1	0.874705	0.483312	1.077490
5	1	0.859602	0.510647	-0.712420
6	-1	0.867581	0.497216	0.182754
7	1	0.863697	0.503994	-0.264860
8	-1	0.865666	0.500620	-0.041049

ตารางที่ 2.1 แสดงการคำนวณหาค่าของ sin และ cosine

โดยที่ค่าจริงของ $\cos 30^\circ = 0.866025$ และ $\sin 30^\circ = 0.500000$ นอกจากนี้ยังสามารถที่จะหาค่าอื่นๆได้จากการพิจารณาจากผลลัพธ์ของเวคเตอร์ที่ได้จากการหมุนโดยต้องมีตัวใดตัวหนึ่งที่จะถูกนำมาเปรียบเทียบในการพิจารณาเครื่องหมายตัวอย่างเช่น ถ้ากำหนดให้ผลลัพธ์ของ $y_n = 0$ จะทำให้ได้ผลลัพธ์จากการคำนวณเป็นดังนี้

$$x_n = \sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 - \tan^{-1}\left(\frac{y_0}{x_0}\right) \qquad (2.29)$$

เช่นเดียวกันกับวิธีการที่แสดงในตัวอย่างข้างต้นดังตารางที่ 2.1 คือทำการลดค่า z_n ลงสู่ค่าศูนย์ แต่ในตัวอย่างนี้จะทำการลดค่าของ y_n ลงแทนดังนั้นเทอมที่จะนำมาใช้ในการพิจารณาเครื่องหมายจะกลายเป็นเทอมของ y_i ที่ตำแหน่งต่างๆดังสมการ (2.30)

$$\sigma_i = \begin{cases} -1 & y_i \geq 0 \\ +1 & y_i < 0 \end{cases} \tag{2.30}$$

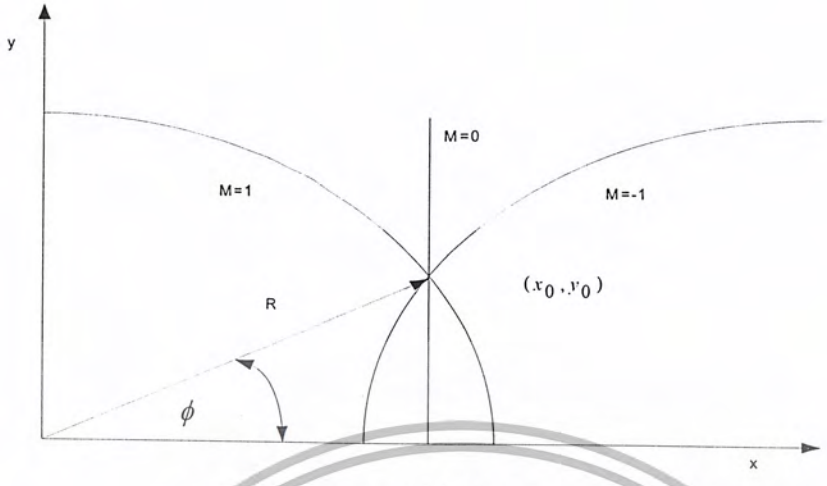
เมื่อทำการกำหนดค่าเริ่มต้นให้ตามสมการที่ (2.30) โดยให้ค่า $x_0 = 1, y_0 = 0.41421$ และก็จะได้ผลการคำนวณดังตารางที่ 2.2

i	σ_i	$x_{i+1} = \sqrt{x_0^2 + y_0^2}/k$	$y_{i+1} \rightarrow 0$	$z_{i+1} \rightarrow \tan^{-1}\left(y_0/x_0\right)$
		1.000000	0.414210	0.000000
0	-1	1.414210	-0.585790	45.000000
1	1	1.707105	0.121315	18.434949
2	-1	1.737434	-0.305461	32.471192
3	1	1.775617	-0.088282	25.346176
4	1	1.781135	0.022694	21.769842
5	-1	1.781844	-0.032966	23.559752
6	1	1.782359	-0.005125	22.664578
7	1	1.782399	0.008800	22.216964
8	-1	1.782433	0.000184	22.440775

ตารางที่ 2.2 แสดงการคำนวณหาค่า $\tan^{-1}(\frac{y_0}{x_0})$

ซึ่งค่าของ $\tan^{-1} 0.41421 = 22.49983$ จะเห็นได้ว่ายิ่งเพิ่มรอบในการคำนวณมากเท่าใด ค่าที่ได้ก็จะใกล้เคียงกับค่าจริงมากขึ้นและยังสามารถนำไปประยุกต์ใช้เพื่อหาค่าอื่นๆ ในฟังก์ชันตรีโกณมิติได้อีกมากมาย

2.2.1 การประยุกต์ใช้กับฟังก์ชันไฮเปอร์บอลิก (Hyperbolic Function)



รูปที่ 2.5 แสดงตำแหน่งของเวคเตอร์ R ในแต่ละพิกัด

พิจารณาจากรูปที่ 2.5 เป็นการแสดงตำแหน่งของเวคเตอร์ R และมุม (ϕ) ณ.จุดของเวคเตอร์ (x_0, y_0) โดยมีตัวแปร m เป็นตัวกำหนดระบบพิกัดซึ่งจะสามารถเขียนเป็นสมการได้ดังนี้

$$R = \sqrt{x_0^2 + m y_0^2} \tag{2.31}$$

$$\phi = \frac{1}{\sqrt{m}} \tan^{-1}\left(\sqrt{m}\,\frac{y_0}{x_0}\right) \tag{2.32}$$

หรือในทางตรงกันข้ามคือ

$$x_0 = R\cos\left(\sqrt{m}\,\phi\right)$$
$$y_0 = \frac{1}{\sqrt{m}} \tan^{-1}\left(\frac{\phi}{\sqrt{m}}\right) \quad \text{โดยที่} m \in \{1, 0, -1\} \tag{2.33}$$

ซึ่งจะสามารถแบ่งรูปแบบพิกัดออกเป็น 3 แบบคือฟังก์ชัน ไฮเปอร์บอลิก (Hyperbolic; m = -1), เชิงเส้น(Linear; m = 0) และวงกลม(Circular; m = 1) ดังที่แสดงในรูปที่ 2.6 ดังนั้นในสมการที่ (2.28) สามารถนำมาเขียนความสัมพันธ์ใหม่ได้เป็น

$$\begin{aligned} x_{i+1} &= x_i - m\sigma_i \delta_{m,i} y_i \\ y_{i+1} &= y_i + \sigma_i \delta_{m,i} x_i \\ z_{i+1} &= z_i - \sigma_i \alpha_{m,i} \end{aligned} \tag{2.34}$$

โดยที่พารามิเตอร์แต่ละตัวมีค่าดังต่อไปนี้

m เป็นตัวแปรที่กำหนดพิกัดของระบบที่มีค่าเป็น $\{1,0,-1\}$

σ_i ตัวกำหนดทิศทาง(เครื่องหมาย)ของการหมุนแต่ละตำแหน่ง $\{1,-1\}$

$\alpha_{m,i}$ มุมที่เปลี่ยนไปที่ค่า i แต่ละตำแหน่ง

$\delta_{m,i}$ ค่าของ 2^{-i} ที่จะถูกกำหนดค่า i ในแต่ละพิกัดต่างๆกัน $\{0 < \delta_{m,i} \leq 1\}$

i ตำแหน่งของการหมุน $\{0,1,2,\ldots., n-1\}$

ซึ่งในส่วนที่เป็นขนาด(สัมประสิทธิ์ตัวคูณ)และส่วนของมุมแต่ละลำดับในพิกัดต่างๆสามารถจะสรุปความสัมพันธ์ได้ดังนี้

m	1	0	-1
$\alpha_{m,i}$	$\tan^{-1}\delta_{1,i}$	$\delta_{0,i}$	$\tanh^{-1}\delta_{-1,i}$
$k_{m,i}^{-1}$	$\sqrt{1+\delta_{1,i}^2}$	1	$\sqrt{1-\delta_{-1,i}^2}$

ตารางที่ 2.3 แสดงความสัมพันธ์ของฟังก์ชั่นไฮเปอร์บอลิกในพิกัดต่าง ๆ

ดังนั้นในสมการที่ (2.28) และ (2.21) จะถูกนำมาเขียนได้ใหม่ในส่วนของสัมประสิทธิ์ตัวคูณกับผลรวมของมุมหลังการหมุนไปจำนวน n รอบแล้วได้เป็นดังนี้

$$k_m = \prod_{i=0}^{n-1} k_{m,i} = \prod_{i=o}^{n-1} \frac{1}{\sqrt{1+m\delta_{m,i}^2}} \tag{2.35}$$

$$\alpha_m = \sum_{i=0}^{n-1} \sigma_i \alpha_{m,i} = \begin{cases} \sum_{i=0}^{n-1} \sigma_i \tan^{-1}\delta_{1,i} & m = 1 \\ \sum_{i=0}^{n-1} \sigma_i \delta_{0,i} & m = 0 \\ \sum_{i=0}^{n-1} \sigma_i \tanh^{-1}\delta_{-1,i} & m = -1 \end{cases} \tag{2.36}$$

จากที่ได้ทราบมาแล้วในตัวอย่างที่แสดงในตารางที่ 2.1 และ 2.2 ซึ่งจะเป็นการหาค่า $\sin z_0, \cos z_0$ และ $\tan^{-1} z_0$ ในโหมดของไฮเปอร์บอลิกก็เช่นกันสามารถที่จะกำหนดค่าเริ่มต้นและคำนวณหาผลลัพธ์ได้ด้วยวิธีการเดียวกันนี้นอกจากนี้ยังสามารถนำไปหาค่าของฟังก์ชันพิเศษอื่นๆโดยผ่านความสัมพันธ์ตามสมการที่ (2.25)

$$e^{z} = \cosh z + \sinh z$$
$$e^{-z} = \cosh z - \sinh z$$
$$\ell n w = 2\tanh^{-1}\left(\frac{w-1}{w+1}\right)$$
$$\sqrt{w} = \sqrt{\left(w+\frac{1}{4}\right)^{2} - \left(w-\frac{1}{4}\right)^{2}}$$

$$\tan w = \frac{\sin w}{\cos w}$$
$$\tanh w = \frac{\sinh w}{\cosh w} \qquad (2.37)$$

และเพื่อให้ง่ายต่อการสร้างของ CORDIC ขั้นตอนการเพิ่มของ $\delta_{m,i}$ ในแต่ละรอบการคำนวณจึงพิจารณาในเทอมกำลังของสองคือ

$$\delta m, i = 2^{-s(m,i)} \quad i \in \{0,1,2,\ldots, n-1\} \qquad (2.38)$$

ในส่วนของ $\delta(m,i)$ ก็จะมีความแตกต่างกันไปตามระบบของพิกัดนั้นๆซึ่งในกรณีของไฮเปอร์บอลิกจะทำการคำนวณซ้ำที่ $i = 4,13,40\ldots..k,3k+1$ จึงจะให้ค่าที่ใกล้เคียงค่าจริงและ $\tanh^{-1}(2^{0})$ จะไม่มีในข้อกำหนด(หาค่าไม่ได้)ดังนั้นสำหรับที่ค่า $m=-1$ จึงเริ่มที่ $i=1$ ในตารางข้างล่างนี้จะแสดงตัวอย่างของค่า $\delta=(m,i)$ ในแต่ละพิกัดและค่า k_m ที่จะได้ดังนี้

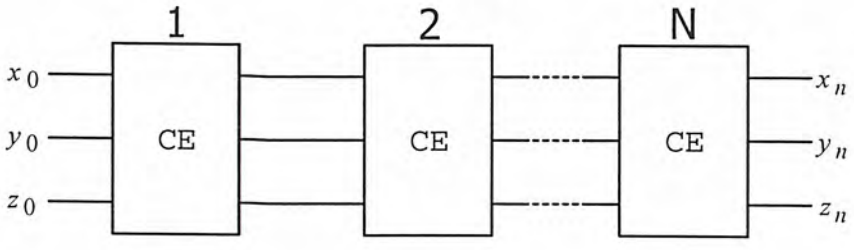
m	$s(m,i)$	k_m
1	0,1,2,3,4,5,...,i,...	0.607253
0	1,2,3,4,5,6,...,$i+1$,...	1.000000
-1	1,2,3,4,4,5,...,12,13,13,14,...	1.207497

ตารางที่ 2.4 แสดงค่า $\delta=(m,i)$, k_m ในแต่ละพิกัด

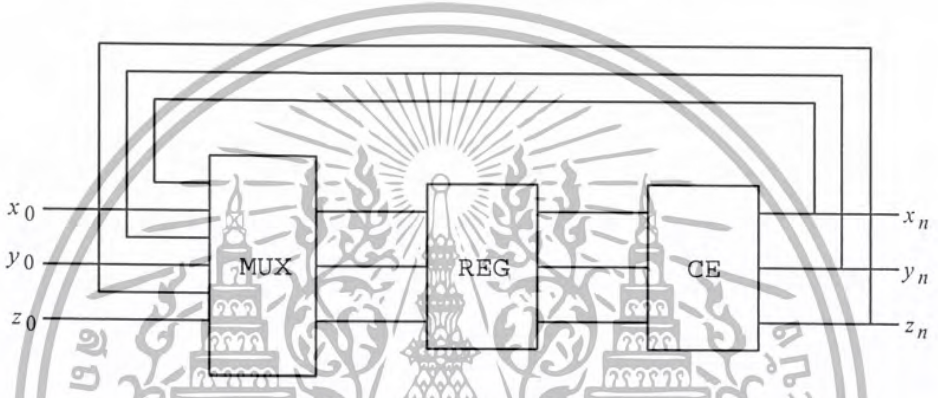
2.2.2โครงสร้างทางฮาร์ดแวร์ของ CORDIC algorithm

หลักการของ CORDIC algorithm จะสามารถนำไปสร้างเป็นชุดคำนวณของสมการ CORDIC แต่ละชุดต่อลำดับกันเป็นจำนวน n ลำดับซึ่งถ้าพิจารณาตามสมการที่ (2.22) จะเห็นว่าแต่ละชุดจะต้องมีการกำหนดค่าที่ต้องการหา,ค่าตัวแปรที่กำหนดพิกัดของระบบ(m)และระดับของตำแหน่งค่า i

ในรูปที่ 2.6 จะแสดงลำดับของชุดการคำนวณ CORDIC แต่ละชุดหรือฮาร์ดแวร์อาจจะมีโครงสร้างในลักษณะที่เป็นแบบวนค่าในการคำนวณดังเช่นในรูปที่ 2.7

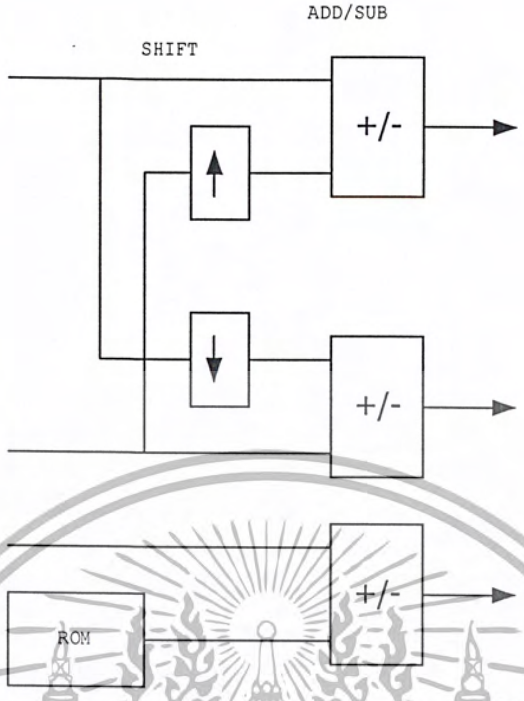


รูปที่ 2.6 โครงสร้างโดยการต่อชุด CORDIC element(CE) ลำดับกัน



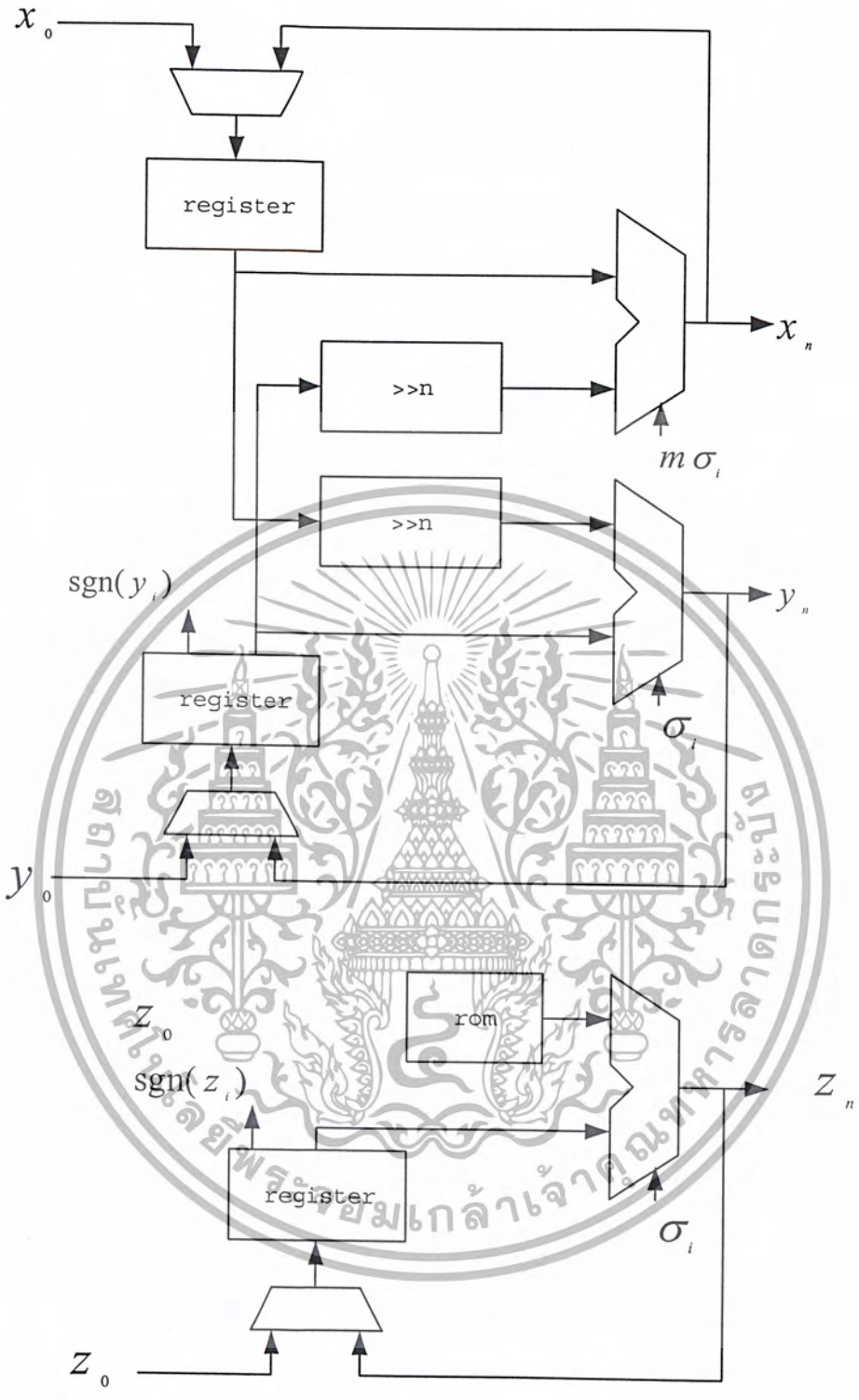
รูปที่ 2.7 โครงสร้างแบบวนค่ากลับ

ส่วนของการสร้างชุดคำนวณของสมการ CORDIC แต่ละชุดนั้นจะพิจารณาการคำนวณในรูปของเลขฐานสอง(binary)ซึ่งจากสมการที่ (2.27) จะเห็นว่าแต่ละสมการจะมีเทอมของการบวกและลบ (adder and subtracter) อยู่สองเทอมใหญ่ๆและส่วนของเทอมย่อยจะมีการคูณของตัวแปรเกิดขึ้นซึ่งตัวแปร σ_i และ m จะเป็นตัวแปรที่กำหนดเครื่องหมายและในระบบเลขฐานสองการเปลี่ยนเครื่องหมายจะใช้วิธีการทำ two's complement ดังนั้นชุดนี้จึงเพียงแค่เปรียบเทียบเครื่องหมายและส่งไปควบคุมการทำ two's complement ส่วนอีกตัวแปรคือ $\delta_{m,i}$ ซึ่งก็คือค่าของ 2^{-i} โดยที่ค่าของ i จะมีค่าแตกต่างกันไปตามระบบพิกัดที่ทำการคำนวณ แต่อย่างไรก็ตามเมื่อพิจารณาในระบบเลขฐานสองแล้วจะเห็นว่าค่าของ 2^{-i} ก็คือการเลื่อน(shift)ไปทางด้านขวามือเป็นจำนวน i ตำแหน่งนั่นเองซึ่งหมายความว่าค่าของ x_i และ y_i ในสมการที่ (2.28) ที่ถูกคูณอยู่กับตัวแปรเหล่านี้จะไม่ถูกนำเข้าวงจรคูณโดยตรงแต่จะทำการเลื่อนและทำ two's complement ค่านั้นๆแทนซึ่งนับเป็นข้อดีของ CORDIC algorithm ดังนั้นสามารถเขียนโครงสร้างในส่วนของการคำนวณ CORDIC แต่ละชุดได้ดังรูปที่ 2.8

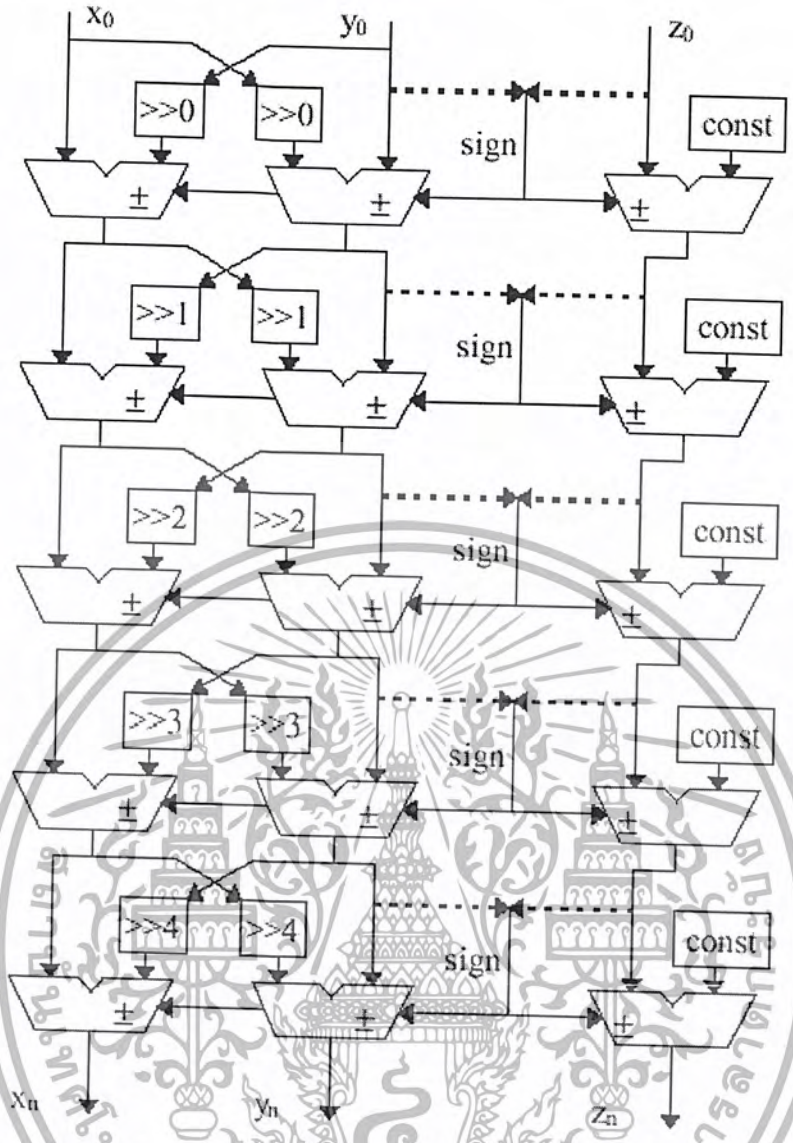


รูปที่ 2.8 โครงสร้างของชุดคำนวณ CORDIC หนึ่งชุด

โดยที่ ROM จะเป็นตัวเก็บค่าของ $\tan^{-1} 2^{-i}$ ที่ $i = 0,1,2,3,...,N$ ตามจำนวนชุดการคำนวณ



รูปที่ 2.9 โครงสร้างของ CORDIC algorithm แบบวนค่ากลับ (loop)



รูปที่ 2.10 โครงสร้างของ CORDIC algorithm แบบลำดับ (cascade)

ในรูปที่ 2.9 และ 2.10 เป็นตัวอย่างแสดงโครงสร้างของ CORDIC algorithm ที่จะถูกสร้างในการนำไปใช้งานจริงซึ่งในรูปจะแสดงในส่วนที่เป็นโครงสร้างของการคำนวณเท่านั้นและสามารถที่จะเพิ่มหรือลดจำนวนรอบการทำงานได้ตามความเหมาะสมทั้งนี้ทั้งนั้นการเปลี่ยนแปลงดังกล่าวก็จะมีผลต่อจำนวนอุปกรณ์ที่นำมาใช้สร้างเป็นฮาร์ดแวร์และความแม่นยำในการคำนวณเป็นต้น ดังนั้นจึงควรพิจารณาเลือกออกแบบใช้งานให้เหมาะสม

2.3 การเขียนภาษา VHDL

2.3.1 Terminology and Convention

การเขียนรูปแบบของระบบดิจิตอลด้วยภาษา VHDL นั้น จะมีศัพท์เทคนิคเฉพาะ ฉะนั้นในส่วนนี้จะเป็นการบรรยาย และอธิบายศัพท์บางคำที่จะต้องพบในรายงานชุดนี้

ลักษณะของรูปแบบ (modal styles) : ลักษณะการเขียนรูปแบบ (model) ด้วยภาษา VHDL สามารถแบ่งได้เป็น

- Behavioral Model : หรือที่เรียกอีกอย่างว่า algorithmic description เป็นรูปแบบที่บรรยายพฤติกรรมของระบบดิจิตอล ในส่วนที่บรรยายมีโครงสร้างคล้ายกับภาษาชั้นสูง (high level language) ทั่วไป เช่น PASCAL หรือ C เป็นต้น ในการจำลองการทำงาน (simulation) คำสั่งแต่ละคำสั่ง (statement) จะถูกประเมินผลเป็นไปตามลำดับ (sequential) จากบนลงล่าง ยกเว้นในกรณีของคำสั่ง LOOP หรือการเรียกใช้โปรแกรมย่อยรูปแบบลักษณะนี้จะไม่ให้รายละเอียดที่เกี่ยวกับการผลิต หรือโครงสร้างของ Hardware แต่ในทางตรงข้ามที่รายละเอียดเกี่ยวกับความสัมพันธ์ระหว่าง input กับ output ที่ดี

- Dataflow Model : เรียกอีกอย่างหนึ่งได้ว่า " Register transfer level " (RTL) เป็นรูปแบบที่ถูกเขียนขึ้น เพื่อจุดประสงค์ที่จะใช้เป็นเครื่องมือสำหรับสังเคราะห์วงจรอัตโนมัติ รูปแบบลักษณะนี้ส่วนใหญ่จะเป็น procedural constructs และ functional operators

- Structural Model : เป็นรูปแบบที่แสดงการเชื่อมต่อกันระหว่างอุปกรณ์ต่างๆ ที่ประกอบกันขึ้นเป็นวงจรหรือระบบดิจิตอล และสามารถเรียกอีกอย่างได้ว่า " netlist representation " เป็นการเขียนที่แสดงให้เห็นโครงสร้างของ hardware

- Mixed – Level Model : จากคุณสมบัติที่อ่อนตัวของภาษา VHDL จึงสามารถที่จะเขียนรูปแบบโดยใช้ลักษณะต่างๆ บรรยายวงจรหรือระบบดิจิตอลเดียวกันได้ ฉะนั้นรูปแบบเช่นนี้จึงมีการเขียนแบบผสม

Concurrency ในภาษา VHDL นั้น ชุดคำสั่งจะทำงานในเวลาเดียวกันและอิสระต่อกัน ลักษณะเช่นนี้เป็นคุณสมบัติที่เป็นความจริงทางฟิสิกส์ของวงจรอิเล็กทรอนิกส์ ชุดคำสั่งนี้เรียกว่า "concurrent statement " และจะทำงานก็ต่อเมื่อมีการเปลี่ยนแปลงค่าของสัญญาณ

Sequential : นอกจากความสามารถที่ชุดคำสั่งจะทำงานแบบ concurrent แล้วบางครั้งการเขียนรูปแบบในลักษณะที่บรรยายพฤติกรรมของวงจร มีความจำเป็นที่จะต้องให้ชุดคำสั่งทำงานเป็นลำดับขั้นเรียงกันจากบนลงล่าง อย่างเช่นการเขียนแบบ behavioral model เป็นต้น ชุดคำสั่งที่เป็น sequential นี้จะใช้ในโปรแกรมย่อย (subprogram)และ process statement

Driver : สัญญาณต่างๆ (signal) ใน VHDL นั้นจะถูกควบคุมด้วยตัวขับหรือ " driver " สัญญาณเหล่านี้จะรับค่าใหม่ (ระดับของสัญญาณ) ได้ด้วยตัวขับนี้เอง

Transaction :การเกิด transaction กับ signal นั้นจะเกิดขึ้นเมื่อมีการกำหนดค่าๆหนึ่งให้กับ signal signal นั้น ค่าใหม่ที่ signal ได้รับอาจจะมีผลหรือไม่มีผลทำให้เกิดการเปลี่ยนแปลงของระดับสัญญาณ (event) เช่นการเปลี่ยนจากค่า logic '0' เป็นค่า logic '0' เป็นต้น

Event : คือการเปลี่ยนระดับค่าของ SIGNAL จากระดับหนึ่งไประดับอื่น อย่างเช่น ในระบบดิจิตอลการเปลี่ยนจาก Logic '0' เป็นค่า Logic'1' หรือในทางตรงกันข้ามถือว่า SIGNAL นั้นเกิด "event" ฉะนั้นจะเห็นได้ว่า การที่จะเกิด event ได้นั้นจะต้องเกิด transaction ไม่จำเป็นต้องเกิด event ทุกครั้ง

Sensitivity List : คือรายชื่อของ signal ต่างๆ ที่มีผลทำให้เกิดการทำงานของ concurrent statement เมื่อเกิด event ขึ้นกับ signal ตัวใดตัวหนึ่งหรือหลายตัวพร้อมกันในรายชื่อนั้น

- Object : ในภาษา นั้นคำว่า ใช้เขียนเพื่อบ่งบอกถึงองค์ประกอบส่วนหนึ่งของรูปแบบ ซึ่งเปรียบได้เหมือนกับภาชนะที่มีไว้สำหรับบรรจุค่าต่างๆ สามารถแบ่งออกได้เป็นสามชั้น (class) ด้วยกันคือ

CONSTANT : ได้แก่ object ประเภทหนึ่งที่เมื่อกำหนดค่าเริ่มต้นให้แล้วจะคงค่านั้นไว้ตลอด ไม่สามารถดัดแปลงหรือแก้ไขได้ สามารถประกาศใช้ได้ในส่วนประกาศต่างๆ ของรูปแบบ (model)

SIGNAL : หมายถึง object ประเภทหนึ่งที่สามารถกำหนดค่าที่สัมพันธ์กับเวลาให้ได้นั้น หมายความว่าSIGNAL สามารถรับค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง SIGNAL จะรับค่าๆหนึ่งได้จากขับสัญญาณหรือ driver ซึ่งตัวขับนี้อาจจะเก็บค่าในอนาคตสำหรับ SIGNAL ไว้ด้วย SIGNAL สามารถประกาศใช้ได้ในส่วนที่เป็น sequential body เท่านั้น ดังนั้น SIGNAL จึงสามารถถูกนำไปใช้ตลอดโครงสร้างของรูปหรือที่เรียกว่า global object

VARIABLE : หรือตัวแปรได้แก่ object ที่สามารถกำหนดค่าใด ๆ ให้ได้และสามารถที่จะเปลี่ยนแปลงค่าได้ตลอดการจำลองการทำงาน แต่จะเก็บค่าเพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง เนื่องจาก VARIABLE สามารถประกาศใช้ได้ในส่วนที่เป็น sequential body เท่านั้นอันได้แก่ส่วนประกอบของ PROCESS, FUNCTION หรือ PROCEDURE ดังนั้น VARIABLE จึงสามารถนำไปใช้ได้เฉพาะในขอบเขตที่ถูกประกาศใช้เท่านั้น (local object)

- ประเภทของ object ที่กำหนดไว้แล้ว (predefined type) : ได้แก่ TYPE ที่กำหนดไว้ใน package ชื่อ STANDARD และกำหนดโดย IEEE ว่าจะต้องมีในระบบที่ใช้พัฒนา VHDL ฉะนั้นจึงไม่จำเป็นต้องประกาศใช้ในทุกรูปแบบที่เขียนขึ้น TYPE ประเภทนี้ได้แก่

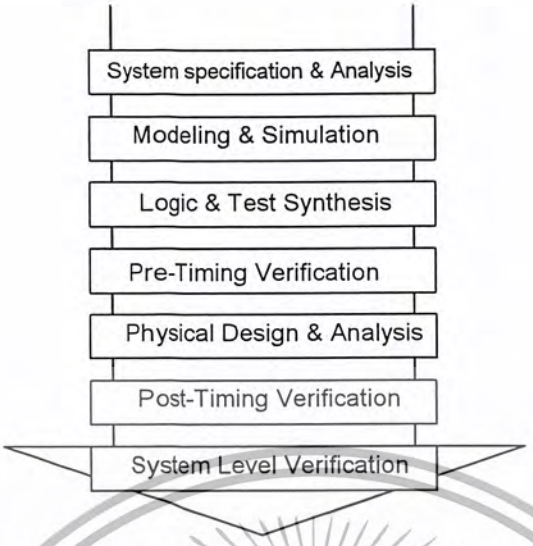
1) BOOLEAN คือกลุ่มของค่า FALSE และ TURE
2) BIT คือกลุ่มของค่า '0' และ '1'
3) INTEGER คือกลุ่มของค่า –214748347 ถึง 214748347
4) REAK คือกลุ่มของค่า –1.0E38 ถึง 1.5E38
5) CHARACTER คือกลุ่มของค่าพยัญชนะ 'A' – 'Z', 'a' – 'z' อักษรหรือเครื่องหมายพิเศษและตัวอักษรควบคุม
4) TIME ได้แก่หน่วยเวลาที่มีค่าพื้นฐานเป็นวินาที (second ย่อด้วย s หรือ S)
5) SEVERITY LEVEL คือกลุ่มของค่า NOTE, WARING, ERROR, FAILURE ส่วนต่างในการเขียน VHDL

การเขียนรูปแบบหรือ modeling ด้วยภาษา VHDL มีความจำเป็นที่จะต้องแนะนำให้รู้จักกับส่วนต่าง ๆ ของแบบ (design units) ที่ใช้ภาษาเสียก่อน และนี่ก็เป็นขั้นตอนแรกที่สำคัญที่สุดของการศึกษา ศึกษาเรียนรู้การใช้ภาษา VHDL เขียนรูปแบบบรรยายระบบดิจิตอลในมุมมองของการออกแบบลักษณะ Top-Down Design นอกจากนั้นการที่จะเข้าใจในเรื่องของโครงสร้าง และส่วนต่าง ๆ ของรูปแบบ VHDL ให้ถูกต้องเสียก่อน

การออกแบบวงจรเชิงเลข (Digital Circuit) นั้น ในปัจจุบันก้าวหน้าไปอย่างมากโดยการใช้ภาษาบรรยายการทำงานของวงจร (Hardware Description Language : HDL) ซึ่งเป็นภาษาที่ใช้สำหรับออกแบบฮาร์ดแวร์ โดยภาษาที่เป็นมาตรฐานสากลเช่น Verilog หรือ VHDL (VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit)) หรือภาษาที่ไม่เป็นมาตรฐานเช่น AHDL (Altera Hardware Description Language) หรือ PHDL (Philips Hardware Description Language) เป็นต้น มาบรรยายการทำงานของวงจรที่ได้ออกแบบไว้ ซึ่งในวิทยานิพนธ์นี้ได้ใช้ภาษา VHDL มาทำการออกแบบวงจร Digital Oscillator ทำให้ลดความยุ่งยากในการนำเอาอุปกรณ์มาเชื่อมต่อให้เป็นวงจร รวมทั้งลดเวลาที่ใช้ในการออกแบบและทดสอบการทำงาน ซึ่งมีความแตกต่างเป็นอย่างมากเมื่อเปรียบเทียบกับการออกแบบในอดีตที่ผ่านมา คือผู้ออกแบบจะต้องนำเอาอุปกรณ์แต่ละตัวที่ทำการออกแบบไว้ มาทำการต่อทดลองในแผงวงจรจริง และทำการทดสอบวงจรเพื่อหาข้อผิดพลาด ซึ่งต้องใช้เวลานานกับการแก้ปัญหาแต่ละอย่างที่เกิดขึ้น แต่ในการออกแบบด้วยภาษา VHDL ผู้ออกแบบเพียงแต่เขียนซอสโค๊ด (Source Code) บรรยายการทำงานของวงจร หลังจากนั้นก็ทำการคอมไพล์ (Compile) แล้วจำลองการทำงาน (Simulate) ดูว่าได้ฟังก์ชั่นการทำงานและไทม์มิ่ง (Timing)ตามที่ต้องการหรือไม่จากนั้นก็นำซอสโค๊ดที่ได้ไปทำการสังเคราะห์ด้วยโปรแกรมสังเคราะห์ (Synthesis Tool) สุดท้ายนำวงจรที่ได้จากการสังเคราะห์ไปทำการแม็ป (Map) ลงไปยัง FPGA (Field Programmable Gate Array) เพื่อเป็นชิป (Chip) ต้นแบบสำหรับการนำไปทดสอบการทำงาน

2.3.2 การออกแบบจากบนลงล่าง

ในการพัฒนาวงจรรวมเชิงเลขขนาดใหญ่ที่มีความซับซ้อน ผู้ออกแบบมักจะมองการออกแบบให้อยู่ในรูปของบล็อกไดอะแกรมก่อน จากนั้นจึงวิเคราะห์ให้ลึกถึงรายละเอียดต่อไป ซึ่งภาษา VHDL นั้นอนุญาตให้อธิบายการทำงานของแต่ละบล็อก และวิเคราะห์การทำงาน แก้ไขและปรับปรุงการทำงานจากผลที่วิเคราะห์ เพื่อให้ได้การทำงานตามที่ต้องกาโดยการออกแบบในลักษณะนี้เรียกว่า หลักการออกแบบจากบนลงล่าง (Top-Down Design) ซึ่งถ้าเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom-Up Design) จะเห็นได้ว่าการออกแบบจากล่างขึ้นบนจะใช้เวลาในการออกแบบมากกว่าเพราะเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic Capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ จำลองการทำงาน ตรวจสอบความถูกต้อง ซึ่งใช้เวลามาก และถ้าวงจรที่ต้องการออกแบบมีความซับซ้อนก็จะเป็นเรื่องที่ยากมากในการออกแบบลักษณะนี้ ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นวิธีการที่เหมาะสมสำหรับการออกแบบและพัฒนาวงจรที่มีความซับซ้อนมากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบ



รูปที่ 2.11 แสดงขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.11 แสดงให้เห็นถึงขั้นตอนการออกแบบจากบนลงล่างทั้งนี้ในทางปฏิบัติอาจจะมีข้อแตกต่างไปจากนี้บ้างเล็กน้อย โดยขั้นตอนของการออกแบบจากบนลงล่างมีรายละเอียด ดังนี้

1. ขั้นตอนการสร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา

2. ขั้นตอนการเขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL สำหรับ บรรยายพฤติกรรมการทำงานพร้อมทั้งจำลองการทำงานเพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3. ขั้นตอนการสังเคราะห์ซึ่งจะต้องทำการกำหนดเทคโนโลยีที่จะมารองรับวงจรที่ออกแบบและอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระบบช่วยออกแบบจะทำการสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยระดับเกท (Gate Level) และการเชื่อมต่อกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของเน็ทลิสต์ (Net list) ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้

4. หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือเน็ทลิสต์แล้ว ข้อมูลที่ได้นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงานในเรื่องของความถูกต้องของฟังก์ชั่นแล้ว ยังมีข้อมูลที่ เกี่ยวกับเวลาด้วย ซึ่งจากความจริงที่ว่า อุปกรณ์อิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของ การเคลื่อนผ่าน (Propagation Delay time) เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับนาโนวินาที แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชั่นต่างๆจำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้นจนอาจจะทำให้การทำงานของวงจรทั้งหมดผิดไปหรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาสูงๆได้

5. ขั้นตอนของการผลิตเป็นวงจรจริง (Technology and Device Mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของอุปกรณ์ FPGA หรือวงจรรวม ASIC

6. หลังจากที่ได้วงจรจริงมาแล้วยังต้องมีความจำเป็นที่จะต้องตรวจสอบการทำงานที่คำนึงถึง เวลาด้วยเพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่นๆให้ เป็นระบบ เพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วยอินพุทและเอาท์พุทแพด (Pad) ซึ่งเป็นจุดต่อสำหรับรับและส่งสัญญาณกับภายนอก

7. หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่นๆให้เป็นระบบแล้วนั้น จะต้องทดสอบ การทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่นๆอีกครั้ง ซึ่งเป็นการทดสอบการทำงานจริงขั้น สุดท้าย

2.3.3 ภาษา VHDL และ ส่วนประกอบต่างๆของภาษา

วิวัฒนาการของภาษา VHDL นั้นเริ่มต้นประมาณปี ค.ศ. 1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกา หรือ DOD (Department of Defence) ได้ทำการพัฒนาโครงการที่มีชื่อว่า VHSIC ซึ่งเป็นการพัฒนาโปรแกรม ซึ่งจัดเป็นภาษาระดับสูงเช่นเดียวกับภาษาC หรือ Pascal แต่สามารถบรรยายพฤติกรรมการทำงานของวงจรเชิงเลข หรือโครงสร้างของวงจรได้ ทั้งนี้เพื่อให้สามารถออกแบบและสร้างวงจรรวมได้รวดเร็วขึ้น

ในการเขียนรูปแบบบรรยายระบบเชิงเลขในลักษณะของการออกแบบจากบนลงล่างจะต้องทำความเข้าใจในเรื่องของโครงสร้างและส่วนประกอบต่างๆของรูปแบบภาษา VHDL เสียก่อนซึ่งส่วนประกอบที่สำคัญและเป็นพื้นฐานของการเขียนมี 4 หน่วย คือ

- หน่วยการออกแบบเอนทิตี (Entity Design unit)
- หน่วยการออกแบบสถาปัตยกรรม (Architecture Design unit)
- หน่วยการออกแบบแพ็กเก็ต (Package Design unit)

หน่วยการออกแบบโครงแบบ (Configuration Design unit)

2.3.3.1 หน่วยการออกแบบเอนทิตี

หน่วยการออกแบบนี้เป็นส่วนที่ใช้สำหรับติดต่อระหว่างภายนอกกับรูปแบบที่เขียนขึ้นโดยเป็นการกำหนดจุดเชื่อมต่อของรูปแบบ กำหนดทิศทางการไหลของสัญญาณ และประเภทของค่าที่สามารถกำหนดให้กับสัญญาณตามจุดต่างๆของข้อมูลที่ไหลผ่านจุดต่อเหล่านั้น รูปที่ 2.12 แสดงให้เห็นถึงโครงสร้างของหน่วยการออกแบบเอนทิตี

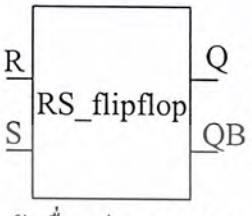
```
Entity component_name is
    Input and Output ports
    Physical and other parameters
End [component_name];
```

รูปที่ 2.12 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบเอนทิตี้

ส่วนนี้จะขึ้นต้นด้วยคำว่า Entity และ is ระหว่างคำทั้งสองคำเป็นส่วนสำหรับชื่อของรูปแบบที่ต้องการจะเขียน(component_name)หลังจากนั้นจะตามด้วยส่วนที่ใช้กำหนดช่องทางเข้าและออกของข้อมูล(input-output) รวมทั้งพารามิเตอร์อื่นๆและที่สำคัญคือหน่วยการออกแบบเอนทิตีจะต้องปิดท้ายด้วยคำว่าEndและเครื่องหมายอัฒภาคเสมอ (;)

```
Entity  RS_ff  is
        port(S,R : in bit;
             Q,QB : out bit);
End RS_ff;
```

(a) หน่วยการออกแบบเอนทิตี ในรูปของภาษา VHDL



(b) มุมมองของตัวเชื่อมประสาน (Interfacing)

รูปที่ 2.13 แสดงรูปแบบของ RS_flipflop

ในรูปที่ 2.12 เป็นหน่วยการออกแบบเอนทิตีที่บรรยายอุปกรณ์ชื่อ RS_flipflop ในส่วนหัวของเอนทิตีมีการกำหนดจุดต่อ 4 จุด ภายใต้ชุดคำสั่ง portโดยที่ 2 จุดแรกเป็นจุดให้ข้อมูลไหลผ่านเข้า ได้แก่ R, S ซึ่งกำหนดด้วยทิศทางการติดต่อกับโลกภายนอกเป็นการไหลเข้าของข้อมูล (in) ส่วนจุดเอาท์พุทเป็นจุดให้ข้อมูลไหลออก ได้แก่ Q,QB ซึ่งกำหนดด้วยทิศทางการติดต่อกับภายนอกเป็นการไหลออก (out) ส่วนประเภทของข้อมูลที่จะไหลเข้าและออกนั้นเป็นประเภท bit ที่สามารถมีค่าได้เพียงสองค่าเท่านั้น คือ “0” และ “1” เท่านั้น

2.3.3.2 หน่วยการออกแบบสถาปัตยกรรม

หน่วยการออกแบบสถาปัตยกรรมคือส่วนที่ใช้เขียนบรรยายพฤติกรรมของรูปแบบในมุมมองของการจำลองการทำงาน พฤติกรรมต่างๆที่บรรยายในส่วนนี้ขึ้นอยู่กับข้อมูลที่ผ่านเข้าและออกตรงช่องทาง ตลอดจนพารามิเตอร์ต่างๆที่กำหนดในหน่วยการออกแบบเอนทิตี รูปที่ 2.11 แสดงให้เห็นถึงโครงสร้างของหน่วยการออกแบบสถาปัตยกรรม

```
Architecture identifier of component_name is
        [declaration]
    Begin
        Specification of the functionality
        of the component in terms of its
        input lines and as influenced by
        physical and other parameters
End [identifier];
```

รูปที่ 2.14 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบสถาปัตยกรรม

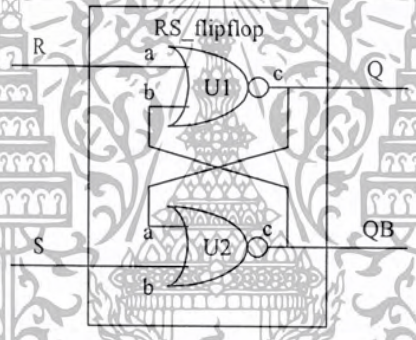
ส่วนของหน่วยการออกแบบสถาปัตยกรรมเริ่มต้นด้วยคำว่า Architecture และตามด้วยชื่อ (identifier) สิ่งที่ต้องกำหนดลงไปได้แก่ สิ่งที่แสดงให้เห็นว่า Architecture นั้นใช้บรรยายหน่วยการออกแบบเอนทิตีใดๆ (of <entity design unit> is) ส่วนที่อยู่ระหว่าง Architecture และ Begin เป็นพื้นที่ส่วนประกาศหน่วยของสถาปัตยกรรมกำหนด (Architecture declaration area) ที่เป็นส่วนเผื่อเลือก (Option) ในบริเวณนี้สามารถใช้เขียนประกาศกำหนดค่าต่างๆที่จะนำไปใช้ภายในสถาปัตยกรรมนั้นได้ อาทิเช่น ประเภท (Type) ต่างๆ (ตัวอย่างเช่น bit, bit_vector), สัญญาณ (signal), ค่าคงที่ (constant), โปรแกรมย่อย (ได้แก่ function และ procedure) และอุปกรณ์ (component) ส่วนที่ใช้บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้าและไหลออกของรูปแบบ (สัญญาณที่กำหนดในชุดคำสั่ง port) นั้นจะถูกบรรยายในบริเวณเนื้อที่ระหว่างคำว่า Begin กับ End ของหน่วยการออกแบบสถาปัตยกรรม และนอกจากนั้นชุดคำสั่งทุกคำสั่งที่อยู่ภายในบริเวณนี้จะเป็นชุดคำสั่งแบบแข่งขนาน (Concurrent statement) เท่านั้น คือทุกๆ statement จะทำงานพร้อมกัน ลำดับก่อนหลังจะ ไม่มีผลต่อการทำงานของรูปแบบ หน่วยการออกแบบสถาปัตยกรรมจะต้องปิดท้ายด้วยคำสั่ง End และชื่อของสถาปัตยกรรมนั้นๆโดยทั่วไปการเขียนรูปแบบระบบเชิงเลขด้วยภาษา VHDL สามารถเขียนได้ในลักษณะต่างๆ ดังนี้

- ลักษณะการไหลของข้อมูล (Dataflow style)
- ลักษณะพฤติกรรม (Behavioral style)
- ลักษณะโครงสร้าง (Structural style)
- ลักษณะผสม (Mixed Model style)

```
Architecture dataflow of RS_ff is
        Begin
                Q  <=  not(QB or R);
                QB <= not(Q or S);
End dataflow;
```

รูปที่ 2.15 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ตามฟังก์ชั่นบูลีน $Q = \overline{QB + R}$ และ $QB = \overline{Q + S}$

รูปที่ 2.15 ส่วนที่บรรยายความสัมพันธ์ระหว่างข้อมูลที่ไหลเข้า (R, S) กับข้อมูลที่ไหลออก (Q, QB) ประกอบด้วยชุดคำสั่งแบบแข่งขนาน 2 ชุด ซึ่งเขียนเป็นประเภทการไหลของข้อมูล หรือเรียกว่า ระดับการถ่ายโอนข้อมูลระหว่างรีจิสเตอร์ (RTL : Register Transfer Level)



รูปที่ 2.16 แสดงโครงสร้างภายในสถาปัตยกรรมของ RS_flipflop

รูปที่ 2.16 เป็นหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะโครงสร้างซึ่งเปรียบเสมือนการนำอุปกรณ์ที่มีอยู่ใน ไลบรารี่ (Library) มาต่อเป็นวงจรตามต้องการ โดยใช้นอร์เกท 2 อินพุท (nor2) จำนวนสองตัวมาสร้างตามฟังก์ชั่นบูลีนของรูปที่ 2.19

```
Architecture struc of RS_ff is
        component nor2
            port(a,b : in bit;
                 c :out bit);
        end component;

    Begin
        U1 : nor2 port map(R,QB,Q);
        U2 : nor2 port map(S,Q,QB);
End struc;
```

รูปที่ 2.17 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะโครงสร้าง

```
Architecture behave of RS_ff is
    Begin
        process(R,S)
            begin
                if R='0' and S='1' then
                    Q <='1';
                    QB <= '0';
                elsif R='1' and S='0' then
                    Q <= '1';
                    QB <='0';
                end if;
        end process;
End behave;
```

รูปที่ 2.18 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะพฤติกรรม

รูปที่ 2.18 เป็นการเขียนบรรยายการทำงานของรูปแบบในลักษณะพฤติกรรม ซึ่งจะเห็นได้ว่ามีลักษณะที่เหมือนกับการเขียนโปรแกรมทั่วไป โดยจะต้องมีการใช้งานส่วนที่เรียกว่า process และการทำงานของรูปแบบจะขึ้นอยู่กับการเปลี่ยนแปลงของสิ่งที่อยู่ภายใน process (อินพุท R, S) ซึ่งเรียกว่า

Sensitivity list การเขียนในลักษณะนี้ลำดับก่อนหลังของชุดคำสั่งจะมีผลต่อการทำงานของรูปแบบที่เขียนขึ้น

```
Architecture mixed of RS_ff is
        component nor2
                port(a,b : in bit;
                        c : out bit);
        end component;

Begin
        U1 : nor2 port map(R,QB,Q);
        QB <= not(Q or S);
End mixed;
```

รูปที่ 2.19 แสดงหน่วยการออกแบบสถาปัตยกรรมของ RS_flipflop ในลักษณะผสม

ไม่ว่าจะเขียนบรรยายส่วนของสถาปัตยกรรมของ RS_flipflop ในลักษณะของพฤติกรรม การไหลของข้อมูล โครงสร้าง หรือผสมที่นำเอาแต่ละลักษณะมาเขียนไว้ในส่วนของสถาปัตยกรรมก็ตามต่างก็มีพฤติกรรมเดียวกัน และจะให้ผลลัพธ์จากการจำลองการทำงานที่เหมือนกัน ซึ่งถือว่าเป็นข้อดีของภาษา VHDL

2.3.3.3 หน่วยการออกแบบแพ็กเก็ต

ข้อมูลต่างๆ ตลอดจนโปรแกรมย่อยที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบเชิงเลขสามารถเก็บไว้ในส่วนของแพ็กเก็ตได้และข้อมูลเหล่านี้สามารถเรียกไปใช้ได้โดยหน่วยการออกแบบเอนทิตีหน่วยการออกแบบสถาปัตยกรรมหรือจากหน่วยการออกแบบแพ็กเก็ตอื่นๆ โดยปกติแล้วแพ็กเก็ตจะแบ่งออกเป็น 2 ส่วน คือ การประกาศแพ็กเก็ต (Package Declaration) และส่วนของบอดี้แพ็กเก็ต (Package Body) เนื่องจากแพ็กเก็ตถูกสร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่กำลังเขียนอยู่ ฉะนั้นการที่จะนำแพ็กเก็ตไปใช้นั้นจะต้องมีการเชื่องโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถกระทำได้ด้วยชุดคำสั่ง USE

Package Declaration

ส่วนที่มีความสำคัญที่สุดของแพ็กเก็ต(ถ้ามองในแง่ของการนำไปใช้จากภายนอก)ได้แก่ส่วนการประกาศแพ็กเก็ต เพราะจะเป็นส่วนที่กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็เก็ตสำหรับนำไปใช้ภายนอกตัวของแพ็กเก็ตเองสิ่งใดๆที่ถูกประกาศไว้ในส่วนของบอดี้แพ็กเก็ตแต่ไม่ได้ถูกประกาศไว้ในส่วนการประกาศแพ็กเก็ตจะไม่สามารถถูกนำค่าและพฤติกรรมไปใช้ส่วนนอกได้ ซึ่งสามารถเปรียบเทียบ

ได้ดับสิ่งที่ประกาศไว้ในส่วนของการประกาศเอนทิตีคือจุดเชื่อมต่อหรือพอร์ทที่มีหน้าที่ติดต่อกับโลกภายนอกฉะนั้นโดยทั่วไปแล้วแพ็กเก็ตสามารถสร้างขึ้นได้โดยไม่จำเป็นต้องมีส่วนบอดี้และยังสามารถถูกนำไปใช้จากรูปแบบภายนอกได้ เช่น ใช้สำหรับประกาศชนิด (Type) หรือสัญญาณเช่นเดียวกันกับส่วนบอดี้แพ็กเก็ตที่ไม่จำเป็นต้องมีส่วนของการประกาศแพ็กเก็ต แต่แพ็กเก็ตนั้นจะไม่สามารถถูกนำไปใช้จากรูปแบบอื่นได้

```
Package package_name is
  Package_declaration_part
End package_name;
```

รูปที่ 2.20 แสดงโครงสร้างโดยทั่วไปของส่วนการประกาศแพ็กเก็ต

Package body

โครงสร้างที่ประกอบด้วยคำสั่งต่างๆในรูปของคำสั่งลำดับ (Sequence)ที่ใช้บรรยายฟังก์ชั่นการทำงานของโปรแกรมย่อย(Subprogram)ทั้งหลายที่ชื่อของโปรแกรมย่อยนั้นๆที่ถูกประกาศไปในส่วนของการประกาศแพ็กเก็ตแล้วจะถูกเก็บไว้ในส่วนบอดี้แพ็กเก็ตทั้งนี้รวมทั้งการกำหนดค่าคงที่ต่างๆอันได้แก่ค่าคงที่ที่ถูกประกาศชื่อก่อนในส่วนของการประกาศแพ็กเก็ตแต่ถูกกำหนดค่าในส่วนของบอดี้แพ็กเก็ต ฉะนั้นส่วนบอดี้แพ็กเก็ตจึงไม่จำเป็นต้องมีถ้าในส่วนของการประกาศแพ็กเก็ตไม่มีการประกาศชื่อที่เป็นโปรแกรมย่อยหรือค่าคงที่ การเขียนบอดี้แพ็กเก็ตนั้นเป็นไปตามกฎเกณฑ์ที่แสดงในรูปที่ 2.21

```
Package body package_name is
  declarative part
End package_name;
```

รูปที่ 2.21 แสดงโครงสร้างโดยทั่วไปของบอดี้แพ็กเก็ต

2.3.3.4หน่วยการออกแบบโครงแบบ

ดังที่ทราบกันแล้วว่ารูปแบบหนึ่งของระบบดิจิตอล ไม่ว่าจะเป็นอะไร จะมีหน่วยการออกแบบเอนทิตีได้ เพียงหนึ่งหน่วยเท่านั้น แต่ในขณะที่ หน่วยการออกแบบเอนทิตี หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรม ที่เป็นหน่วยรอง ได้หลายหน่วย ดังนั้นจะต้องมีหน่วยการออกแบบโครงแบบมาเพื่อกำหนดการใช้โครงแบบ (Configuration) ประกอบเอนทิตีกับหน่วยการออกแบบสถาปัตยกรรมหน่วยไหนเข้าด้วยกัน

```
Configuration identifier of entity_name is
        Configuration_declarative_part
End;
```

รูปที่ 2.22 แสดงโครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

2.3.4 ชุดคำสั่งลำดับ (Sequential Statements)

ภาษา VHDL สามารถใช้เขียนรูปแบบ (modeling) บรรยายระบบดิจิตอลในลักษณะของ behavioral description ที่โครงสร้างภายในประกอบด้วย sequential statement การศึกษาในรายละเอียดของโครงสร้างดังกล่าว สำหรับ software engineering ที่มีความคุ้นเคยกับการเขียน โปรแกรมด้วยภาษาชั้นสูง อาทิเช่น C หรือ PASCAL อยู่ก่อนแล้ว จะสามารถเข้าใจโครงสร้างแบบ sequential ได้ง่าย เพียงแต่ต้องทำความเข้าใจเกี่ยวกับลักษณะการทำงานของ hardware เพิ่มเติม ในภาษา VHDL มีคำสั่งดังต่อไปนี้

- WAIT statement
- VARIABLE assignment
- Signal assignment
- IF-THEN-ELSE statement
- CASE statement
- Loops
- NEXT statement
- EXIT statement
- RETURN statement
- NULL statement
- Procedure call
- ASSERTION statement

จะกล่าวเฉพาะ statement ที่สำคัญๆ เพื่อความเข้าใจในการทำงานแบบ sequential เท่านั้น ตามที่เคยกล่าวมาแล้วว่าภาษา VHDL เป็นภาษาที่มีคุณสมบัติเป็นแบบแข่งขนานนั้น คือ ชุดคำสั่งภายในตัวโครงสร้างจะเป็นชุดคำสั่งแบบแข่งขนาน เช่นเดียวกับภาษา ADA ชุดคำสั่งลำดับหรือ sequential statement ที่เรียกว่า อันได้แก่ process statement

2.3.4.1. Process Statement

หัวใจสำคัญของ Concurrent shell ที่ทำให้สามารถเขียน VHDL model เพื่อบรรยายพฤติกรรมของระบบดิจิตอลอิเล็กทรอนิกส์ในลักษณะ behavioral description ได้แก่คำสั่ง process ที่โครงสร้างภายในจะประกอบด้วยชุดคำสั่งแบบลำดับเท่านั้น ชุดคำสั่งเหล่านี้จะทำงานเป็นลำดับจากบนลงล่าง เมื่อ PROCESS ถูกกระตุ้นให้ทำงาน

คำ PROCESS ในบรรทัดแรกของโครงสร้าง แสดงถึงจุดเริ่มต้นของชุดคำสั่ง process ในบางกรณีใน architecture หนึ่งอาจจะมีโครงสร้างของชุดคำสั่ง process หลายชุดได้ คำ END PROCESS บอกถึงจุดสิ้นสุดของชุดคำสั่ง process เมื่อคำสั่ง END PROCESS ถูกปฏิบัติแล้วชุดคำสั่ง process จะหยุดการทำงานลงชั่วขณะ (แต่ยังคง active อยู่ตลอดเวลา) จนกว่าจะมีสัญญาณอย่างน้อยตัวหนึ่งใน sensitivity list เกิด event ขึ้นอีก ชุดคำสั่ง process เป็นชุดคำสั่งแบบแข่งขนาน นั้นหมายความว่า โดยปกติแล้วชุดคำสั่ง process จะทำงานตลอดเวลา การที่ชุดคำสั่งทั้งหลายอยู่ภายในบล็อกของชุดคำสั่ง process เป็นชุดคำสั่งแบบลำดับ นั้นถ้าชุดคำสั่ง process ใดที่ไม่มี sensitivity list เป็นตัวควบคุมการทำงาน จะทำให้เกิดการทำงานที่เปรียบเสมือนว่าเป็นวงรอบ (loop) ที่ไม่รู้จบขึ้น วิธีการป้องกันการเกิดเหตุการณ์เช่นนี้ คือการเติม wait statement ลงในส่วนของชุดคำสั่ง process

2.3.4.2. Wait Statement

ชุดคำสั่ง process สามารถมี sensitivity list ได้เพียงอันเดียว หมายความว่าชุดคำสั่ง process จะถูกกระตุ้นได้จากการที่สัญญาณหนึ่งในรายชื่อที่เกิด event ขึ้นเท่านั้น หลังจากที่ถูกกระตุ้นแล้วคำสั่งทั้งหลายที่อยู่ภายในจะทำงานแบบลำดับลงมาจนกระทั่งหมด และชุดคำสั่ง process จะหยุดการทำงานชั่วคราวจนกว่าจะมี event เกิดขึ้นอีกกับสัญญาณตัวใดตัวหนึ่งในรายชื่อนั้นอีก ถ้ากรณีที่ภายในรายชื่อประกอบด้วยสัญญาณหลายตัว และเกิด event ขึ้นในเวลาเดียวกัน จะมีสัญญาณเพียงตัวเดียวจากทั้งหมดเท่านั้น ที่กระตุ้นการทำงาน ของชุดคำสั่ง process ซึ่งไม่สามารถที่จะบอกได้ว่าเป็นตัวใด ดังนั้นการใช้ชุดคำสั่ง process ร่วมกับ sensitivity list จึงมีขีดจำกัดอยู่มากในภาษา VHDL มีวิธีการหลีกเลี่ยงปัญหาเช่นนี้โดยใช้ชุดคำสั่ง wait statement

คำสั่ง wait statement นั้นให้ความคล่องตัวกว่า เพราะ wait statement มีข้อดีคือ ประการแรกสามารถที่จะกำหนดลงตรงตำแหน่งใดๆ ภายในโครงสร้างของชุดคำสั่งลำดับได้ เพื่อระงับการทำงานภายในชุดคำสั่ง process ตรงตำแหน่งที่ wait statement อยู่ ส่วนการใช้ sensitivity list เป็นตัวควบคุมการทำงานชุดคำสั่ง process จะหยุดตรงตำแหน่งสุดท้ายของชุดคำสั่ง process เท่านั้นประการที่สอง สามารถที่จะกำหนด wait statement ได้หลายๆอันในแต่ละตำแหน่ง ตามความต้องการโดยไม่จำกัด ส่วนประการสุดท้าย wait statement มีหลายรูปแบบ นั่นหมายความว่าสามารถที่จะใช้ควบคุม PROCESS ได้หลายลักษณะ แต่มีสิ่งที่ต้องจำไว้ว่า wait statement ไม่สามารถที่จะใช้ร่วมกับ sensitivity list ภายใน PROCESS เดียวกันได้ ฉะนั้นผู้เขียนรูปแบบจึงต้องตัดสินใจก่อนว่าจะใช้อะไร ทั้งนี้ขึ้นอยู่กับระบบดิจิตอลที่จะเขียนบรรยายและประสบการณ์ในการเขียน

ในภาษา VHDL สามารถใช้ wait statement ได้ 4 แบบคือ

- WAIT ON signal_list --signal sensitivity
- WAIT UNTIL condition; --condition
- WAIT FOR time; --timeout
- WAIT; --forever

แต่ละอย่างมีความหมายและวิธีใช้ที่แตกต่างกันตามตัวอย่างต่อไปนี้

- WAIT ON clock, clear, preset, d;

คำสั่งจะหยุดการทำงานของชุดคำสั่งลำดับ ไว้จนกว่าจะเกิด event ขึ้นที่สัญญาณ clock หรือ clear หรือ present หรือ d

- WAIT UNTIL(clock = "1");

การทำงานของชุดคำสั่งลำดับจะหยุดที่ตำแหน่งนี้ และจะทำงานต่อไปเมื่อสัญญาณ clock เกิด event และ boolean expression เป็น TRUE ทุกครั้งเมื่อลำดับการทำงานถูกหยุดตรงจุดนี้ และก่อนที่จะทำต่อไปได้นั้นต้องตรวจสอบว่ามี event เกิดขึ้นบนสัญญาณ clock หรือเปล่า และสัญญาณ clock มีค่าเป็น "1" ฉะนั้น boolean expression จะต้องเป็นจริง

- WAIT FOR 10 NS;

คำสั่งนี้เป็นการรอหรือหยุดในขณะที่จำลองการทำงาน (simulation) จนกว่าเวลาการจำลองจะล่วงเลยไปแล้ว 10 ns. จะมีผลทำให้ PROCESS เริ่มทำงานต่อไปนี้

- WAIT;

เป็นคำสั่งให้ PROCESS หยุดทำงานตลอดไป โดยไม่มีการเริ่มต้นการทำงานใหม่ บางครั้งการเขียนรูปแบบบรรยายการทำงานของระบบดิจิตอล มีความจำเป็นที่ต้องใช้คำสั่งนี้ เพื่อหยุดการทำงานของอย่างถาวรนอกจากนั้นยังสามารถใช้ควบคุม (ON,UNTIL และ FOR) ร่วมกันใน wait statement ได้ส่วนใหญ่ของ VHDL analyzer จะตรวจสอบเพื่อให้เป็นที่แน่ใจว่าใน PROCESS มี sensitivity list หรือ wait statement อย่างใดอย่างหนึ่ง เพื่อป้องกันปัญหาอันอาจจะเกิดขึ้นได้

2.3.4.3. IF-THEN-ELSE statement

เป็นชุดคำสั่งพื้นฐานที่ใช้สอบถาม เพื่อการตัดสินใจกระทำอะไรบางสิ่งบางอย่าง

IF statement

Format : IF condition THEN
Sequential_statement
END IF;

2.3.4.4. CASE Statement

โครงสร้างอีกอันหนึ่งที่เป็นลำดับ และมีความคล้ายคลึงกับ IF-THEN-ELSE คือ CASE statement เพราะใช้เป็นคำสั่งเลือกหนทางปฏิบัติ ตามข้อแม้ (condition) ที่กำหนดให้

คำสั่ง CASE และ END CASE กำหนดจุดเริ่มต้นและจุดสิ้นสุด คำสั่งWHENใช้สำหรับกำหนดตัวเลือกที่จะนำมาเปรียบเทียบกับ expression ตัวเลือกใดเป็นไปตาม expression ที่กำหนด PROCESS จะเริ่มต้นทำงานที่ชุดคำสั่งลำดับที่ตามมาจนกระทั่งคำสั่งสุดท้ายของตัวเลือกนั้นๆ และจะออกจาก CASE statement โดยไม่ทำหนทางเลือกอื่นๆ ที่ยังคงเหลืออยู่

ชุดคำสั่งแบบแข่งขนาน (Concurrent Statement)

ภาษา VHDL เป็นภาษาที่มีการทำงานในลักษณะแข่งขนาน concurrency หรือสามารถที่จะมองชุดคำสั่งแบบแข่งขนานแต่ละอันเป็น PROCESS ที่เชื่อมต่อกันด้วย signal แต่ละ PROCESS ทำงานอิสระ

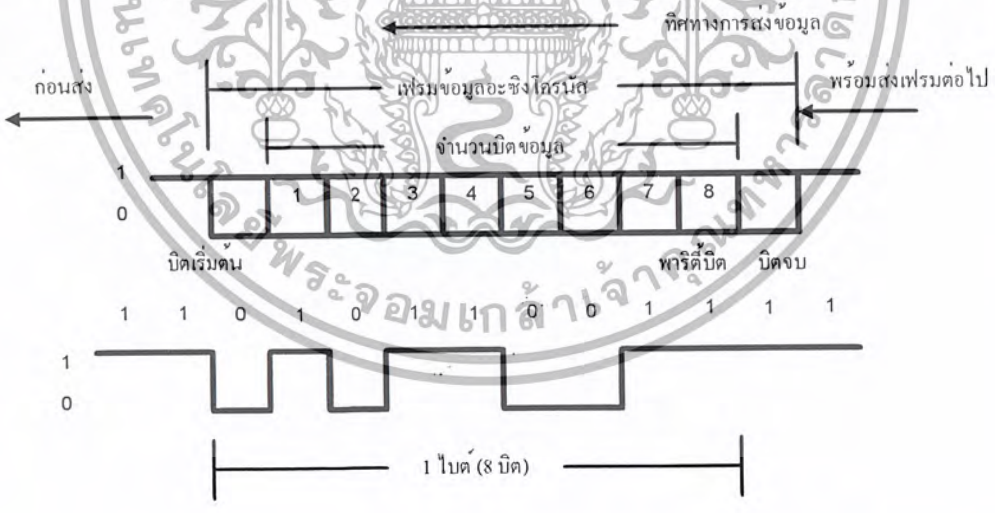
ไม่ขึ้นต่อกัน ที่เรียกว่า asynchronous ดังนั้น concurrent statement ส่วนใหญ่จึงสามารถเขียนแทนได้ด้วย PROCESS statement

ชุดคำสั่งแบบ sequential ไม่สามารถที่จะใช้ในรูปของชุดคำสั่งแบบแข่งขนานได้ มีบางคำที่สามารถใช้ใน VHDL ได้ทั้งสองรูปแบบ เช่น signal assignment เป็นต้น ชุดคำสั่งที่ใช้ในโครงสร้างแบบ concurrent ซึ่งมีทั้งหมดได้แก่

1) Signal assignment statement
2) Component instantiation statement
3) Assert statement
4) Generate statement
5) Process statement
6) Procedure statement
7) Block statement

2.4 การส่งข้อมูลแบบอะซิงโครนัส

ในการส่งข้อมูลแบบอะซิงโครนัส กลุ่มของบิตจำนวน 5 บิต (รหัสโบดอต) หรือ 8 บิต (รหัสแอสกี) จะแทนตัวอักขระที่ถูกส่งออกเป็นเฟรม (Frame) บางครั้งเราเรียกว่าเป็นการส่งข้อมูลแบบ "Start/Stop" การส่งข้อมูลจะส่งทีละอักขระโดยที่ช่วงเวลาระหว่างอักขระจะเป็นเท่าไรก็ได้ ดังนั้น เครื่องรับจะต้องตรวจสอบว่า บิตใดเป็นบิตเริ่มต้นของอักขระและบิตใดเป็นบิตสุดท้ายของอักขระ



รูปที่ 2.23 การส่งสัญญาณข้อมูลดิจิตอลเป็นเฟรมโดยวิธีอะซิงโครนัส

ในการส่งอักขระแต่ละตัวจะประกอบด้วยบิตเริ่มต้น (1 บิต) + ข้อมูล (8 หรือ 7 บิต) + 1 พาราตี้บิต (แต่อาจจะไม่ใช่ก็ได้) + บิตจบ (1 บิต) รวมเป็น 10 บิต คิดเป็น 1 เฟรม

2.4.1 ขั้นตอนการส่งข้อมูล

ขั้นตอนการส่งข้อมูลดิจิตอลโดยวิธีอะซิงโครนัสมีดังนี้

1. ก่อนจะเริ่มทำการส่งข้อมูล สัญญาณจะมีค่าเป็น "1" ตลอดเวลา

2. เมื่อเริ่มส่งข้อมูลสัญญาณของบิตแรกจะเปลี่ยนเป็น "0" นั่นคือบิตเริ่มต้น เครื่องรับจะเริ่มสัญญาณนาฬิกาของตัวเอง เมื่อเวลาผ่านไป ½ บิต ถ้าสัญญาณยังคงเป็น "0" ต่อไปอีก ½ บิต ต่อมาก็จะเป็นการเริ่มของการส่งสัญญาณข้อมูล แต่สัญญาณกลับไปเป็น "1" อีก ก็แสดงว่าเกิดความผิดพลาดอันเกิดจากสัญญาณรบกวนในสายส่ง และยังไม่มีสัญญาณข้อมูลใด ๆ ส่งมายังปลายทาง

3. หลังจากได้เริ่มบิตเริ่มต้นแล้ว ผู้ส่งจะเริ่มส่งรหัสบิตของอักขระ อาจจะเป็น 5 บิต หรือ 8 บิต หรือ 7 บิต แล้วตามด้วยพาราตี้บิต (อาจจะใช้หรือไม่ใช้ก็ได้) ตามรูปที่ 1 เป็นการส่งสัญญาณข้อมูลขนาด 8 บิต สำหรับ 1 อักขระ โดยเป็นสัญญาณข้อมูล 7 บิต บิตที่ 8 เป็นพาราตี้บิตคี่ (Odd) จากนั้นสัญญาณจะเป็น "1" ไปอีก 1 บิต ซึ่งถือว่าเป็นบิตจบ สัญญาณจะเป็น "1" ต่อไปเรื่อย ๆ จนกว่าจะเริ่มมีการส่งสัญญาณข้อมูลในเฟรมต่อไป

การส่งสัญญาณแบบอะซิงโครนัสมีใช้กันอย่างกว้างขวาง เพราะเทคนิคการส่งข้อมูลไม่ยาก รวมทั้งสายส่งสัญญาณก็มีราคาถูก ส่วนใหญ่ใช้ในการส่ง-รับข้อมูลระหว่างเครื่องคอมพิวเตอร์ส่วนบุคคล (PC) กับศูนย์บริการข้อมูลที่อยู่ไกลออกไป เช่น โฮสต์คอมพิวเตอร์ของตลาดหลักทรัพย์หรือระบบธนาคาร

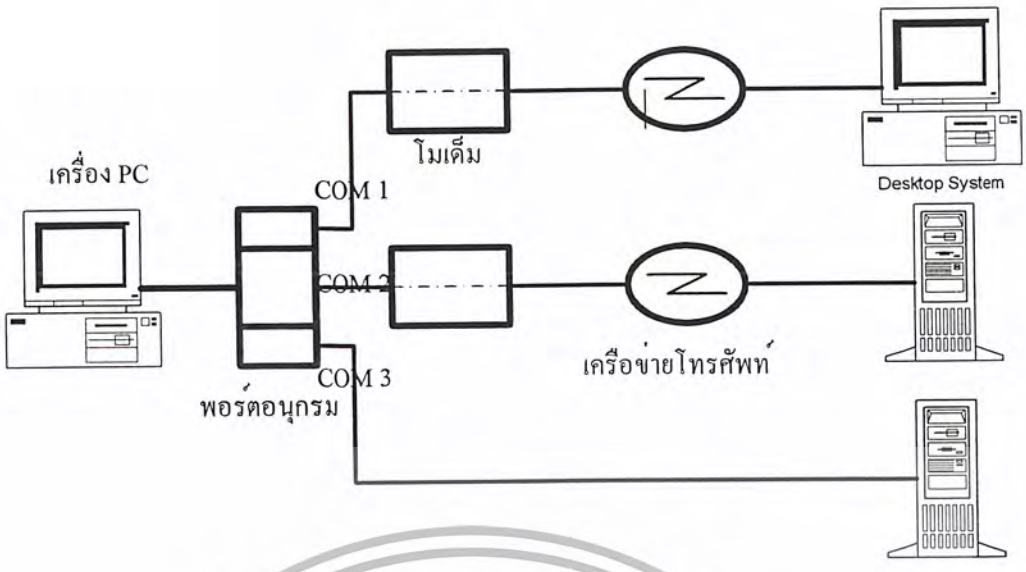
เนื่องจากการสื่อสารข้อมูลดิจิตอลแบบอะซิงโครนัสมีความเร็วในการส่งข้อมูลต่ำ จึงมักใช้กับเทอร์มินอลที่ไม่มีบัฟเฟอร์ นอกจากนั้นเวลาประมาณ 20 เปอร์เซ็นต์ของการส่งอักขระแต่ละตัวจะสูญเสียไปกับบิตเริ่มต้น บิตจบ และพาราตี้บิต

ในการส่งข้อมูลที่เป็นบล็อกโดยในแต่ละบล็อกจะมีอักขระมากกว่า 1 อักขระ มักจะส่งโดยวิธีซิงโครนัสซึ่งจะกล่าวต่อไป

2.4.2 การตรวจสอบความผิดพลาด

การตรวจสอบความผิดพลาดของข้อมูลในการสื่อสารข้อมูลดิจิตอลแบบอะซิงโครนัส เราสามารถตรวจสอบได้จากพาราตี้บิต ซึ่งเป็นพาริตี้คี่ (Odd) และพาริตี้คู่ (Even) เช่น ถ้าเราส่งข้อมูลเป็นพาริตี้คู่ นั่นคือเมื่อรวมบิตของอักขระทั้งหมดที่เป็น "1" กับพาริตี้บิต (ซึ่งอาจจะเป็น "0" หรือ "1") แล้วจะได้เป็นจำนวนคู่ ซึ่งเมื่อเครื่องรับได้รับเฟรมข้อมูลไปแล้ว ถ้ารวมบิตของ "1" ทั้งหมดได้เป็นจำนวนคี่แล้วแสดงว่าข้อมูลที่รับมามีความผิดพลาดเกิดขึ้น แต่นับบิต "1" ทั้งหมดได้จำนวนคู่เช่นกัน เราก็ไม่สามารถตรวจสอบความผิดพลาดได้

ตัวอย่างจากรูปที่ 2.23 บิตข้อมูลมี 7 บิตคือ 1011001 และกำหนดพาริตี้บิตเป็นพาริตี้คู่ ดังนั้นค่าของพาริตี้บิตจึงเป็น "1" เพื่อให้จำนวนบิต "1" ทั้งหมดใน 8 บิตรวมกันเป็นเลขคี่ วิธีการตรวจสอบความผิดพลาดในการส่งข้อมูลแบบอะซิงโครนัสดังที่ได้กล่าวมาเราเรียกว่า การตรวจสอบพาริตี้ (Parity Check)



รูปที่ 2.23 ตัวอย่างการสื่อสารข้อมูลดิจิตอลแบบอะซิงโครนัสจากพอร์ตอนุกรมของเครื่อง PC

2.4.3 การส่งข้อมูลแบบซิงโครนัส

สำหรับการสื่อสารข้อมูลคอมพิวเตอร์ที่ให้ประสิทธิภาพที่ดีกว่าการสื่อสารข้อมูลแบบอะซิงโครนัส คือการสื่อสารข้อมูลแบบซิงโครนัส ลักษณะของข้อมูลที่ถูกส่งผ่านสายส่งผ่านสายสื่อสารจะถูกส่งไปเป็นกลุ่มบิต (Block of Characters or Bits) โดยไม่จำเป็นต้องมีบิตเริ่มต้นและบิตจบเช่นเดียวกับการสื่อสารแบบอะซิงโครนัส

การพิจารณาเวลาเริ่มต้นและเวลาสิ้นสุดของบล็อกข้อมูลแต่ละบล็อกเราพิจารณาจากกลุ่มบิต ส่วนหัว (Header) และกลุ่มบิตส่วนท้าย (Trailer) ของบล็อกข้อมูล กลุ่มบิตพิเศษทั้ง 2 กลุ่มนี้เป็นกลุ่มบิตที่แทนข่าวสารการควบคุมการส่งข้อมูลมากกว่าที่จะบิตข้อมูล ข้อมูลที่รวมเข้ากับข่าวสารการควบคุมการส่งข้อมูลนี้เราเรียกว่า (Frame) รูปแบบของเฟรมนั้นขึ้นอยู่กับว่า การส่งข้อมูลนั้นจะเป็นแบบอะซิงโครนัสอักขระ (Character Synchronization) หรือแบบซิงโครนัสบิต (Bit Synchronization)

2.4.4 การส่งข้อมูลซิงโครนัสอักขระ

การส่งข้อมูลแบบซิงโครนัสอักขระ เฟรมที่ใช้ในการส่งข้อมูลจะมีรูปแบบเป็นเฟรมอักขระ (Character Oriented Frame) คืออักขระในบล็อกข้อมูลจะถูกเรียงลำดับกัน ส่วนบิตพิเศษที่แทนการควบคุมก็จะอยู่ในรูปของอักขระเช่นกัน เฟรมข้อมูลจะเริ่มต้นด้วยอักขระซิงโครนัส (Synchronization Character) หรือเราเรียกสั้น ๆ ว่า อักขระซิงก์ (SYN) ซึ่งอาจจะมีมากกว่า 1 อักขระ (ดูรูปที่ 2.24 ประกอบ) อักขระซิงก์จะมีรูปแบบของบิตที่แน่นอน เพื่อให้ทางผู้รับสามารถรู้ได้ทันทีว่าเมื่อมีอักขระซิงก์เข้ามาที่เครื่องรับแสดงว่าจะเป็นการเริ่มต้นของบล็อกข้อมูลแล้ว ทางผู้รับก็จะเตรียมพร้อมรับบล็อกข้อมูลที่จะตามมา ต่อมาอักขระซิงก์ก็จะเป็นอักขระควบคุมซึ่งจะเป็นข่าวสาร เช่น บอกจุดเริ่มต้นของบล็อกข้อมูล จำนวนอักขระในบล็อกข้อมูล ความยาวของบล็อกข้อมูล เป็นต้น จากนั้นจึงตามบล็อกข้อมูล

จำนวนสูงสุดของบิตแทนอักขระใน 1 เฟรม จะเป็นจำนวนทวีคูณกับจำนวนบิตข้อมูลใน 1 อักขระคือถ้าใน 1 อักขระมี 8 บิต (7 บิตข้อมูล + 1 พาริตี้บิต ดังตัวอย่างในรูปที่ 2.24 ข.) ดังนั้นจำนวนบิตของเฟรมจะเท่ากับ 2^7 หรือ 128 บิต

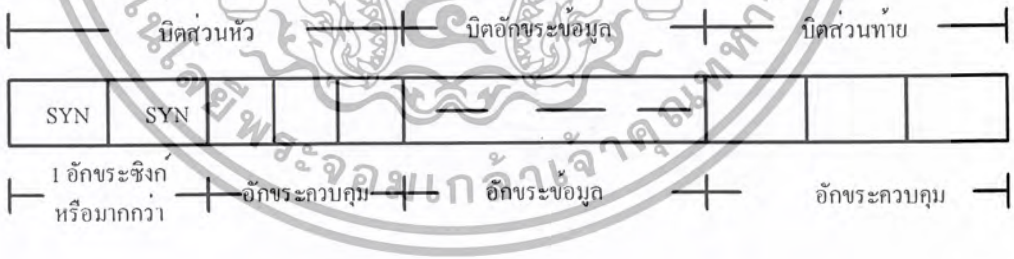
เราจะเห็นว่าบิตส่วนหัวซึ่งประกอบด้วยอักขระซิงก์ (SYN) และอักขระควบคุม จะมีไว้สำหรับบอกแก่เครื่องรับว่าบัดนี้ข้อมูลกำลังจะมาถึง เครื่องรับจะเริ่มรับ (อ่าน) ข้อมูลเมื่ออักขระข้อมูลเริ่มต้นผ่านไป (SYN) ในขณะเดียวกันเครื่องรับก็จะหารหัสที่ระบุจุดสิ้นสุดของข้อมูล (ETX หรือ ETB) หรือบิตส่วนท้ายนั่นเอง

ส่วนการใส่อักขระ SYN ที่ส่วนหัว 2 อักขระหรือมากกว่า ก็เพื่อป้องกันการผิดพลาดเมื่อผู้รับตรวจพบการเปลี่ยนแปลงในสัญญาณที่ผ่านเข้ามากับสายสื่อสารซึ่งมีลักษณะเหมือนกับอักขระ SYN ดังนั้นถ้าเครื่องรับตรวจพบอักขระ SYN ติดกัน 2 อักขระ แสดงว่าต่อไปจะเป็นการเริ่มต้นส่วนข้อมูลที่แท้จริงแล้ว

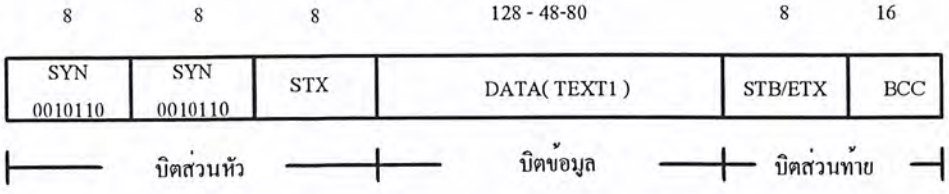
ในการส่งข่าวสารของแต่ละบล็อก จำนวนบิตในแต่ละบล็อกไม่จำเป็นต้องเป็นจำนวนทวีคูณของจำนวนบิตของอักขระ อาจจะมีแค่อักขระส่วนหัวและอักขระส่วนท้ายโดยไม่มีอักขระข้อมูลเลยก็ได้

2.4.5 การส่งข้อมูลแบบซิงโครนัสบิต

แทนที่ข้อมูลจะถูกส่งไปทีละอักขระเรียงลำดับกัน เช่น การส่งข้อมูลแบบซิงโครนัสอักขระ ในวิธีการส่งข้อมูลแบบซิงโครนัสบิต ข้อมูลจะถูกส่งออกไปทีละบิตเรียงลำดับกัน ทั้งบิตข้อมูลและบิตการควบคุมโดยไม่จำเป็นต้องแปลงมาอยู่ในรูปของอักขระ การเริ่มต้นบล็อคหรือ เฟรมข้อมูลจะเริ่มต้นด้วยกลุ่มบิตพิเศษขนาด 8 บิตซึ่งเรียกว่า แฟลก (Flag) และจะมีแฟลกรูปแบบเหมือนเดิมกับแฟลกเริ่มต้นเพิ่มในส่วนท้ายของบล็อก หรือเฟรมข้อมูลเป็นแฟลกจบ (ดูรูปที่ 2.24 ประกอบ)



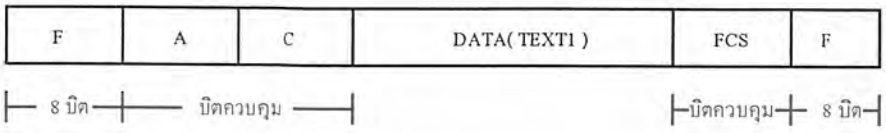
ก. รูปแบบของเฟรมอักขระ



ข. เฟรมอักขระ 8 บิต

รูปที่ 2.24 บล็อกหรือเฟรมอักขระของการส่งข้อมูลแบบซิงโครนัสอักขระ

เครื่องรับของผู้รับจะรอคอยสัญญาณดิจิตอล หรือกลุ่มบิตที่ผ่านเข้ามาที่มีรูปแบบเดียวกันกับแฟลกเพื่อเป็นสัญญาณให้เริ่มต้นเฟรมข้อมูล หลังจากแฟลกเริ่มต้นผ่านไปแล้วจะเป็นกลุ่มของบิตควบคุม (Control Field) ทำหน้าที่บอกความยาวของบล็อกข้อมูล ตำแหน่งปลายทางของผู้รับ วิธีการอ่านบิตข้อมูล และข่าวสารอื่น ๆ จากนั้นจะเป็นบิตข้อมูลจริง ๆ ที่ต้องการส่งไปให้ผู้รับ จบบิตข้อมูลจะเป็นบิตควบคุมตรวจสอบความผิดพลาดของข้อมูล (Frame Check Sequence) และจะจบ เฟรมด้วยแฟลกจบ



รูปที่ 2.25 เฟรมข้อมูลของการส่งข้อมูลแบบซิงโครนัส

2.4.6 การสื่อสารแบบอนุกรมและ RS-232

มีทางเลือกอยู่ 2 ทางในการที่จะเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพ่วงอื่น ๆ หรือคอมพิวเตอร์ด้วยกัน นั่นคือการรับส่งข้อมูลแบบขนานและการรับส่งข้อมูลแบบอนุกรม การรับส่งข้อมูลแบบขนานจะเป็นการรับหรือส่งข้อมูลคราวละ 4 หรือ 8 บิตในเวลาเดียวกัน ซึ่งจะทำให้การรับและส่งข้อมูลทำได้ที่ความเร็วสูง ซึ่งก็หมายความว่าจำนวนของสายที่ใช้ในการส่งจะต้องมีมากเท่ากับจำนวนบิตของข้อมูลที่จะส่งด้วย นอกจากนี้ยังจะต้องรวมถึงสายที่ใช้สำหรับการควบคุมและการตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจจะต้องใช้สายมากเป็น 2 เท่าของจำนวนบิตของข้อมูลที่จะส่งได้ ซึ่งก็เป็นปัญหาในเรื่องราคาของสายที่ใช้ในการเชื่อมต่อแบบขนานจะมีราคาแพง

ในขณะที่การรับส่งข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลาย ๆ บิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะคอยเช็คข้อมูลมาให้ครบทุกบิตเสียก่อนจึงทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลอนุกรมอาจมีความเร็วต่ำกว่าแบบขนาน ในด้านจำนวนสายสัญญาณการรับส่ง ข้อมูลแบบอนุกรมจะใช้จำนวนสายที่น้อยกว่ามาก อย่างน้อยที่สุดใช้เพียง 2 – 3 เส้นเท่านั้น แต่อัตราเร็วในการรับส่งข้อมูลอาจต่ำกว่าแบบขนาน อย่างไรก็ตามการรับส่งแบบอนุกรมสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

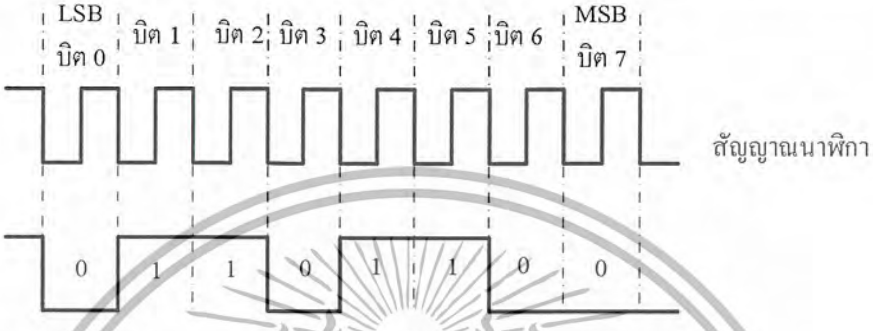
2.4.6 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นจะแบ่งได้เป็น 2 แบบ คือ การสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสก็คือคีบอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นจะเป็นสายของข้อมูล ดังนั้นการติดต่อกับแบบซิงโครนัสนี้

จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูล และกราวน์ รูปที่ 2.24 แสดงให้เห็นถึงไทมิ่งไดอะแกรมของการส่งข้อมูลแบบซิงโครนัส

2.4.7 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัสคือการรับและส่งข้อมูลไปในสาย โดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับ



รูปที่ 2.26 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

และภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (Baud rate) มีหน่วยเป็นบิตต่อวินาที (bit per second: bps)

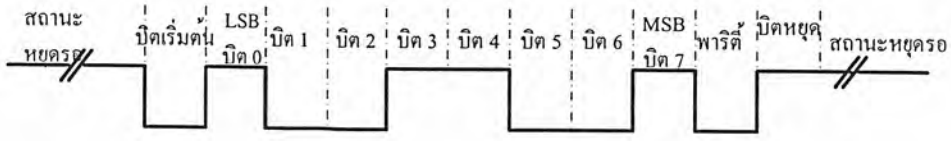
รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสจะประกอบด้วย 4 ส่วนด้วยกัน คือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรมจะมีขนาด 5,6,7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Vit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1, 1.5 หรือ 2 บิต

รูปที่ 2.27 แสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส ซึ่งเมื่อไม่มีข้อมูลที่จะส่ง ขา DATA จะ มีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (Waiting State) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา DATA มีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต ซึ่งจะเรียกบิตนี้ว่าบิตเริ่มต้นจากนั้นบิตข้อมูลจะถูกส่งออกไป โดยเริ่มจากบิตที่มีค่านัยต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวน 5,6,7 หรือ 8 บิตก็ได้ จากนั้นจะตามด้วยบิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่จะส่งบิตปิดท้าย ซึ่งจะให้ขาดดาต้ามีสถานะลอจิก 1 อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมา สำหรับการรับและส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver / Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ ในการรับและส่ง

ข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600, และ 19200 บิตต่อวินาที



รูปที่ 2.27 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าบอร์ดเรตได้สูงถึง 115200 บิตต่อวินาที เนื่องจากบอดเรตคือ จำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้บอดเรตในกรส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็สามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ ต่อวินาที

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (Odd) แบบคู่ (Even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก "1" ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบิตพาริตี้เข้าไปด้วย ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิต และมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสอง จะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิก "1" จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตี้เป็นคู่ค่าในบิตพาริตี้ จะต้องมีลอจิกเป็น "0" แต่ถ้าพาริตี้เป็นคี่ ค่าบิตพาริตี้จะต้องเป็น "1" เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตี้มีจำนวนบิตที่เป็นลอจิก "1" มีจำนวนรวมกันเป็นเลขคี่ ในตารางที่ 1 แสดงตัวอย่างของบิตพาริตี้ในการรับส่งข้อมูลอนุกรม

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART ซึ่งทางภาครับจะต้องทำการกำหนดคุณสมบัติการตรวจสอบพาริตี้คี่หรือพาริตี้คู่ จากนั้นภาครับของ UART จะทำการตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับได้ออกมาเป็นเลขคี่ ทางภาครับจะแสดงข้อผิดพลาดออกมาให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดที่เกิดขึ้นในการถ่ายทอดได้ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีการนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

ตารางที่ 2.5 บิตพาริตี้ของข้อมูล

ข้อมูล	บิทพาริตี้คู่	บิตพาริตี้คี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	0	1
11111111	1	0

2.4.9 มาตรฐานพอร์ตอนุกรม RS-232

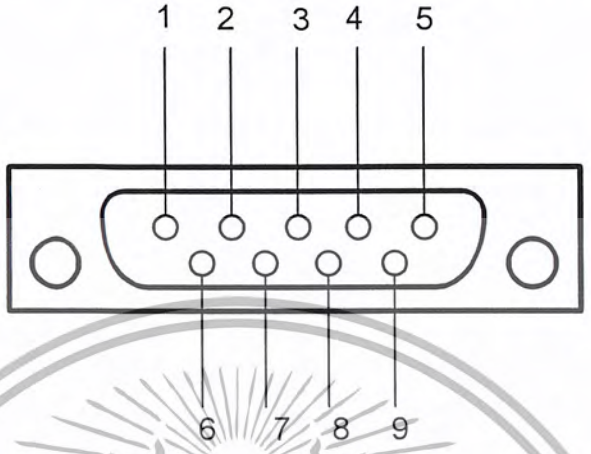
มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมและแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้สื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดที่อยู่ห่างไกลกัน โดยคณะกรรมการที่เรียกว่า สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industrial Association : EIA) ได้วางมาตรฐานที่มีชื่อเรียกว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็คเตอร์เป็นแบบ 1 DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ –3 ถึง –12V แสดงว่ามีข้อมูล (Mark) และ +3 ถึง +12V แสดงว่าเป็นช่องว่าง (Space)

มาตรฐาน RS – 232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่าอุปกรณ์ DTE จะ ต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัว เช่น ไมโครคอนโทรลเลอร์ หรือ ไมโครคอมพิวเตอร์ซึ่งมีความสามารถในการสร้างบิตข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่าน มาตรฐาน RS - 232

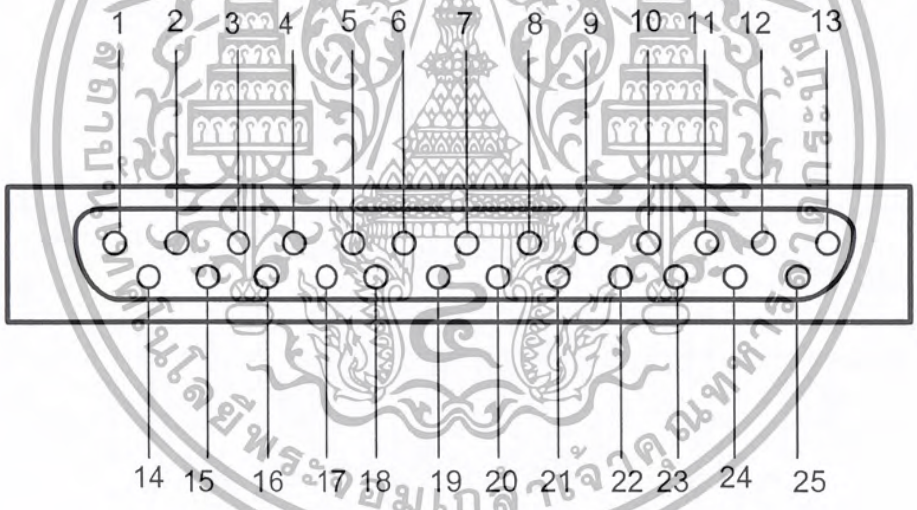
ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างกนึ่งที่เห็นได้ชัดคือ คอนน็คเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็คเตอร์ของ DCE จะเป็นตัวเมียซึ่งพอร์ตของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็คเตอร์ที่อยู่ที่โมเด็มจะเป็นแบบ DCE สำหรับการใช้งานบนคอมพิวเตอร์ พอร์ตอนุกรม RS – 232 มักถูกใช้เชื่อมต่อกับโมเด็ม หรือเมาส์ โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุดถึง 20 เมตร

2.4.10 คอนเน็คเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็คเตอร์แบบ DB-9 ตัวผู้ หรือคอนเน็คเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็คเตอร์แบบ DB-9 เนื่องจากขาอื่นที่เคยใช้งานในอดีต ปัจจุบันมีการใช้งานไม่มากนัก โดยแสดงรูปร่างและตำแหน่งดังรูปที่ 2.28 (ก) และ 2.28 (ข)



(ก).คอนเน็คเตอร์อนุกรม 9 ขาหรือแบบ DB-9 (มองจากด้านหลังคอมพิวเตอร์)



(ข).คอนเน็คเตอร์อนุกรม 25 ขาหรือแบบ DB-25 (มองจากด้านหลังคอมพิวเตอร์)

รูปที่ 2.28 การจัดขาของคอนเน็คเตอร์อนุกรมตามมารตรฐาน RS-232 ทั้งแบบ DB-9 และDB-25

สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกในรูปที่ 2.29 ลูกศรในรูปแสดงถึงทิศทางของข้อมูลในรูปที่ 2.29 (ก) เป็นการเชื่อมต่อแบบ Null modemหรือการเชื่อมต่อโดยตรงโดยไม่ผ่าน

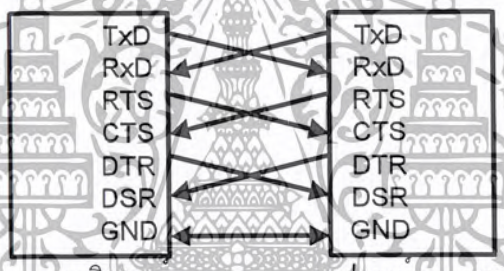
โมเด็มโดยมีการตรวจสอบหรือแฮนด์เช็คเต็มรูปแบบ ส่วนรูปที่ 2.29(ข) เป็นการเชื่อมต่อแบบ Null modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูลอีกเส้นหนึ่งรับข้อมูลและเส้นสุดท้ายเป็นกราวด์ สำหรับรายละเอียดมีดังนี้

Data Carrier Detect : DCD หรืออาจเรียกว่าCarrier Detect : CD ขานี้จะแอกตีฟ เมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูล เช่น โมเด็มสำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก

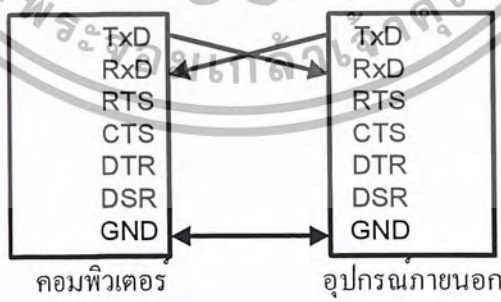
Receiver Data : RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์บัฟเฟอร์

Transmitted Data : TD หรือ TxD ขานี้ใช้เพื่อส่งข้อมูลจากคอมพิวเตอร์โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลออกไป

Data Terminal Ready : DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้ อุปกรณ์ปลายทางรับรู้ว่า ต้องการติดต่อด้วย DTR นี้จะต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทางและขา DTR ของอุปกรณ์ปลายทางจะต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ถ้าใช้การเชื่อมต่อเป็นแบบ Null modem ซึ่งใช้สายในการเชื่อมต่อเพียง 3 เส้น



(ก).การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ Null modem



(ข). การต่ออุปกรณ์ภายนอกเข้ากับคอมพิวเตอร์แบบ RS-232 โดยใช้สายเพียง 3 เส้น

รูปที่ 2.29 การต่ออุปกรณ์ภายนอกกับพอร์ทอนุกรมของคอมพิวเตอร์ในลักษณะต่าง ๆ

จะต้องต่อขา DTR และ DSR ของตัวมันเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

Signal Ground : GND ขากราวน์ของระบบ

Data Set Ready : DSR ขานี้จะใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DSR

Request To Send : RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null modem 3 สาย จะต้องเชื่อมต่อขา RTS และ CTS ของตัวมันเองเข้าด้วยกัน เพื่อจะให้การรับและส่งข้อมูลเกิดขึ้นได้ตลอดเวลา

Clear To Send : CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ ข้อมูลที่ขาTxD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ปกติในการสื่อสาร โดยทั่วไปสายนี้จะไม่ถูกใช้งานจะใช้ก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มกับโปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

2.4.11 UART

UART มาจากคำว่า Universal Asynchronous Receiver Transmitter ซึ่งหมายถึง อุปกรณ์ที่ทำหน้าที่รับส่งข้อมูลแบบอะซิงโครนัสนั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรม แบบอซิงโครนัสแล้วส่งออกไปและทำหน้าที่แปลงสัญญาณอนุกรมแบบอซิงโครนัสที่ป้อนเข้ามายังคอมพิวเตอร์แล้วยังแจ้งข้อมูลอื่นให้คอมพิวเตอร์ทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต), รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอด ข้อมูล (ผิดพลาดจากพาริตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารอยู่ในช่วง 1-65535 UART สามารถรับส่งข้อมูลได้ทั้งแบบ Half duplex และ full duplex โดยการส่งแบบ Half duplex เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบ Full duplex สามารถรับและส่งในคราวเดียวกัน

2.4.12 ชนิดของ UART

ในเครื่องคอมพิวเตอร์ทั่วไปมี UART สองเบอร์คือ 8250 ซึ่งเป็นแบบที่ใช้กันมานาน UART เบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเดียวกัน ทำให้การรับและส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้น แต่ UART เบอร์นี้ถือว่าเป็นต้นแบบของ UART โดยคอมพิวเตอร์ทุก ๆ รุ่นจะต้องสนับสนุนการทำงานตามรูปแบบของ UART เบอร์นี้

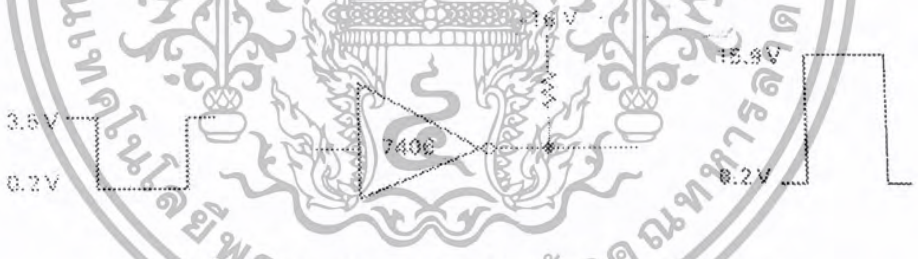
UART อีกเบอร์คือ 16450 มีความสามารถรับส่งข้อมูลได้ที่ความเร็ว 115,200 บิต ต่อวินาที และเริ่มรีจีสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิฟรีจิสเตอร์แบบ FIFO ขนาด 16 ไบท์ เข้าไปทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ 256 กิโลบิตต่อวินาที ได้โดยคอมพิวเตอร์ในปัจจุบันใช้ UART เบอร์นี้หรือใหม่กว่า เช่นเบอร์ TL164C750 ซึ่งมีรีจิสเตอร์แบบ FIFO ขนาด 16 ไบท์ทำงานได้ที่ระดับแรงดัน +5 V และ +3 V มีโหมดประหยัดพลังงานสามารถรับส่งข้อมูลได้ที่ความเร็ว 1 เมกกะบิตต่อวินาทีเมื่อใช้สัญญาณนาฬิกา 16 เมกกะเฮิร์ท

อย่างไรก็ตาม ความเร็วในการส่งข้อมูลที่มากมายของ UART เบอร์ใหม่ ๆ ก็ไม่ได้ช่วยให้การรับส่งข้อมูลของคอมพิวเตอร์ขึ้น เนื่องจากว่าคอมพิวเตอร์ยังใช้ความถี่ของสัญญาณนาฬิกาในการแปลงข้อมูลข้อมูลเพียง 1.8432 เมกะเฮิร์ทเท่านั้น

2.5 การแปลงดิจิตอลเป็นอนาล็อก(Analog to Digital Conversion)

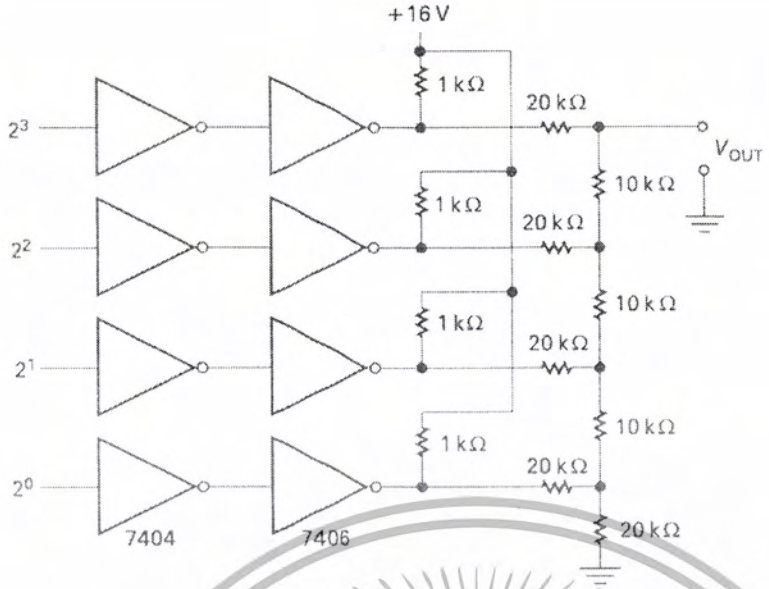
2.5.1ตัวแปลงดิจิตอลเป็นอนาล็อกแบบทีทีแอล(The TTL Digital to Analog Converter)

การทำให้โครงข่ายทำงานนั้น 1 จะต้องมีแรงดันจากแหล่งจ่าย และ 0 จะต้องเป็นกราวด์หรือ 0 โวลท์ แรงดันเอาท์พุทของทีทีแอลจะให้ 0 โวลท์ที่ดีหรืออย่างแย่ที่สุดก็เป็น 0.4 โวลท์ แต่ว่า 1 แรงดันเป็น 3.5 โวลท์ โดยการใช้เอาท์พุทแบบ คอลเลคเตอร์เปิด(Open-Collector) เช่น 7406 หรือ 7407 ตัวต้านทานที่ดึงไปที่ Vs เราสามารถแปลงระดับแรงดันทีทีแอลไปเป็นแรงดันที่ต้องการ โดยการใช้โครงข่ายตัวแปลงดิจิตอลเป็นอนาล็อก แรงดันเอาท์พุทของ 7406 ไม่ใช่กราวด์ หรือ Vs ที่แท้จริงแต่ก็ใกล้เคียงกันมาก แสดงในรูป 2.30



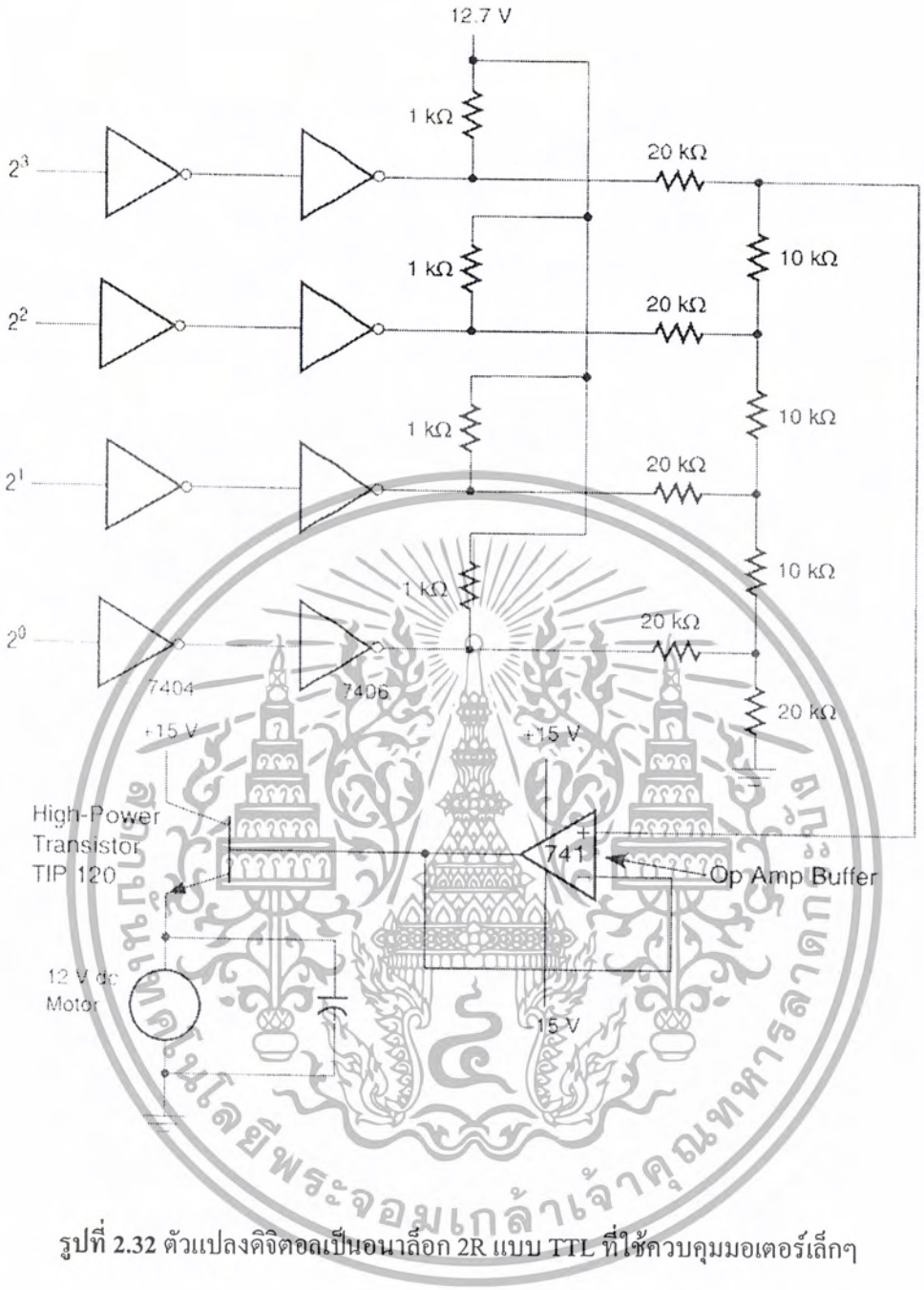
รูปที่ 2.30 การแปลงระดับแรงดัน TTL เป็นระดับดีทูเอ

จากรูป 2.30 แสดงชนิดของบัฟเฟอร์ที่ใช้สร้าง ตัวแปลงดิจิตอลเป็นอนาล็อกแบบทีทีแอลที่มีเอาท์พุทสูงสุด 15 โวลท์ เมื่อเอาท์พุทของ 7406 เป็นต่ำ(Low) อย่างดีที่สุดจะเป็น 0.1-0.2 โวลท์จากกราวด์ และจะทำให้เกิดความผิดพลาดขึ้น และเมื่อเอาท์พุท 7406 เป็นสูง(High) อิมพีแดนซ์แรงดันที่ 20 กิโลโอห์มจะเป็น 16 โวลท์ ในการกระทำแบบนี้ต้องบวกความต้านทาน 1 กิโลโอห์ม กับ 20 กิโลโอห์ม หมายความว่า ความต้านทาน 2R มีขนาดน้อย(5% ในกรณีนี้) ขณะที่เป็น 16 โวลท์ และไม่สูงจากกราวด์มากนักเมื่อเป็นต่ำ



รูปที่ 2.31 ตัวแปลงดิจิตอลเป็นอนาล็อก 2R แบบ TTL

ความผิดพลาดบางประการสามารถตัดออกไปได้ โดยการใช้ทฤษฎีอื่น แต่การใช้ตัวแปลงดิจิตอลเป็นอนาล็อกหลายๆตัวความผิดเพี้ยนนี้ยอมรับได้ ในรูปที่ 2.31 แสดงถึงตัวแปลงดิจิตอลเป็นอนาล็อกใช้ควบคุมความเร็วของดีซีมอเตอร์ เช่น อาจใช้กับแขนกล ตัวแปลงดิจิตอลเป็นอนาล็อกใช้ออปแอมป์เป็นบัฟเฟอร์



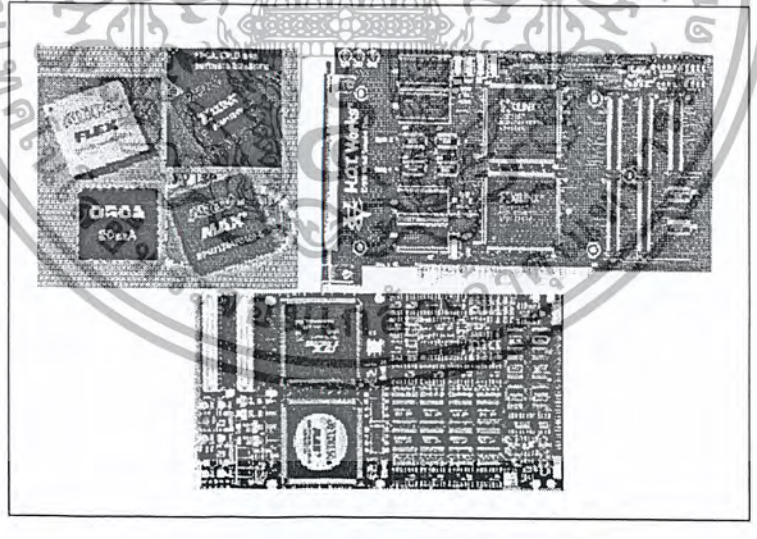
รูปที่ 2.32 ตัวแปลงดิจิตอลเป็นอนาล็อก 2R แบบ TTL ที่ใช้ควบคุมมอเตอร์เล็กๆ

บทที่ 3
การออกแบบและการสร้าง

3.1 การออกแบบวงจรเชิงเลขด้วยอุปกรณ์ FPGA

อุปกรณ์ FPGA (Field Programmable Gate Array) เป็นอุปกรณ์ที่ใช้ในการโปรแกรมวงจรที่ได้ออกแบบลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการทำงานตามที่ออกแบบไว้ ในการทำ FPGA ซึ่งเป็นวิธีการออกแบบ IC (Integrated Circuit) แบบ Semicustom อีกวิธีหนึ่ง เมื่อเทียบกับการทำ ASICs (Application Specific Integrated Circuits) แล้วนั้นก็มีทั้งข้อดีและข้อเสีย คือ การทำ FPGA จะมีข้อจำกัดในด้านขนาดของวงจรเพราะภายในอุปกรณ์ FPGA จะมีจำนวนเกท (gate) ให้ใช้จำนวนจำกัด และการทำ FPGA ก็เหมาะสำหรับการทำผลิตภัณฑ์ต้นแบบหรือเพื่อผลิตในปริมาณต่ำ ส่วนข้อดีของการทำ FPGA ก็คือระยะเวลาที่ใช้ในการทำตั้งแต่เขียนรหัส (code) อธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลด (download) นั้นน้อยกว่าการทำ ASIC มากและการตรวจสอบหรือแก้ไขการออกแบบก็ทำได้สะดวก

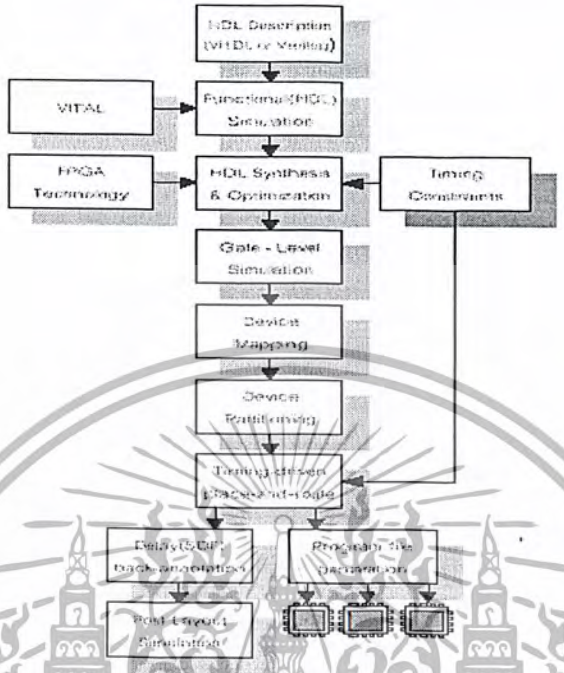
การทำ FPGA ในปัจจุบันมีประสิทธิภาพและความสะดวกมากขึ้น ทั้งนี้ก็เนื่องจากทางบริษัทผู้ผลิตอุปกรณ์ FPGA ได้เพิ่มความสามารถของอุปกรณ์ FPGA โดยเพิ่มจำนวนองค์ประกอบภายในหรือปรับปรุงโครงสร้างสถาปัตยกรรมภายในและยังได้เพิ่มประสิทธิภาพของซอฟต์แวร์ที่ใช้ทำ PPR (Partitioning, Placement and Routing) สำหรับอุปกรณ์นั้นๆด้วย ลักษณะของตัวFPGAและการนำไปใช้งานแสดงดังในรูปที่ 3.1



รูปที่ 3.1 ลักษณะของตัว FPGA และการนำไปใช้งาน

สำหรับตัวอุปกรณ์ FPGA นั้นก็มีโครงสร้างพื้นฐาน เทคโนโลยีที่ใช้สร้าง ตลอดจนเทคนิควิธีการโปรแกรมที่แตกต่างกันสำหรับผู้ผลิตแต่ละราย นอกจากนั้นอุปกรณ์ FPGA ของแต่ละผู้ผลิตก็มีโครงสร้างและความสามารถที่แตกต่างกันบางส่วน ในการใช้งานนั้นอุปกรณ์ FPGA สามารถนำไปประยุกต์ใช้งาน

ได้ เช่น การประมวลผลสัญญาณเชิงเลข (DSP : Digital Signal Processing) การออกแบบไมโครคอนโทรลเลอร์ เป็นต้น โดยมีขั้นตอนในการออกแบบ ดังแสดงในรูปที่ 3.2



รูปที่ 3.2 ขั้นตอนการออกแบบโดยใช้อุปกรณ์ FPGA

3.2 การออกแบบโดยใช้ภาษาอธิบายการทำงานของฮาร์ดแวร์

ในการออกแบบวงจรเชิงเลขนั้นทำได้โดยการวาดวงจร หรือใช้ภาษาอธิบายฮาร์ดแวร์ ในขั้นตอนนี้เป็นขั้นตอนที่ไม่แตกต่างกันระหว่างการออกแบบด้วย FPGA และ ASIC ในกรณีที่ใช้ภาษาอธิบายฮาร์ดแวร์ แต่ในกรณีที่ออกแบบโดยวิธีการวาดวงจรจะแตกต่างกันโดยที่การทำวิธีนี้จะต้องคำนึงถึงเทคโนโลยีที่จะใช้ซึ่งแต่ละเทคโนโลยีก็มีความแตกต่างกันไป จะเห็นได้ว่าการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ทำได้สะดวกกว่า เพราะการทำด้วยวิธีนี้ไม่ต้องคำนึงถึงเทคโนโลยีที่จะใช้ (Technology independence) และที่สำคัญการออกแบบด้วยวิธีนี้สามารถที่จะแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่าเพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยี

ในการเขียนโค้ด สิ่งที่ต้องคำนึงถึงคือเขียนอย่างไรจึงจะสามารถสังเคราะห์เป็นวงจรได้และให้คุณสมบัติของวงจรตามที่กำหนด เพราะลักษณะการเขียนโค้ดจะมีผลโดยตรงกับวงจรที่ได้ เนื่องจากในการสังเคราะห์วงจรนั้นซอร์ฟแวร์สังเคราะห์วงจร (Synthesis Tools) จะทำการสังเคราะห์ตามโค้ดที่เขียน ถ้าอธิบายการทำงานของวงจรเดียวกันแต่เขียนโค้ดในลักษณะที่ต่างกันเมื่อสังเคราะห์แล้วจะได้วงจรที่ต่างกัน และจากวงจรที่ต่างกัน เมื่อนำไปทำต้นแบบด้วย FPGA หรือการทำ ASIC แล้วจะได้ไอซีที่มี

คุณสมบัติต่างกันทั้งในด้านของขนาดหรือความเร็ว (area and time) ส่วนการเขียนโค้ดลักษณะใดเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดนั้นก็ขึ้นอยู่กับประสบการณ์ในการออกแบบ

3.2.1 การจำลองการทำงานของวงจร (Simulation)

ขั้นตอนนี้เป็นขั้นตอนที่สำคัญเพราะเป็นขั้นตอนที่ใช้ตรวจสอบฟังก์ชั่นการทำงานของวงจรว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหน เพื่อที่จะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สำหรับทำการจำลองการทำงานของวงจร เช่น V-System และ ModelSim ของบริษัท Model Technology

3.2.2 การสังเคราะห์วงจร

ในขั้นตอนนี้จะใช้ซอร์ฟแวร์สังเคราะห์วงจร (Synthesis tools) ทำการสังเคราะห์โค้ดเพื่อให้ได้เป็นวงจรขึ้นมา แต่ต้องตรวจสอบด้วยว่าซอร์ฟแวร์นั้นๆสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการใช้หรือไม่ โดย FPGA ที่นิยมใช้งานเช่นของบริษัท Xilinx ตระกูล XC4000 และบริษัท Altera ตระกูล FLEX 10 K ซอร์ฟแวร์สังเคราะห์วงจรที่นิยมใช้เช่นโปรแกรม Leonardo Spectrum ของบริษัท Exemplar Logic ซึ่งในขั้นตอนนี้ซอร์ฟแวร์สังเคราะห์วงจรจะแปลงโค้ดและทำการออปติไมซ์ (Optimization) เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ นอกจากนี้ยังสามารถกำหนดข้อบังคับสำหรับวงจรได้เช่น ข้อบังคับในเรื่องของเวลา (time constraints) หรือข้อบังคับในเรื่องของพื้นที่ ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอนออปติไมซ์เพื่อให้วงจรที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการออปติไมซ์คือการเทียบ (mapping) วงจรให้เข้ากับเทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างสถาปัตยกรรมภายในอุปกรณ์ FPGA ในกรณีของ Xilinx ตระกูล XC4000 และ Altera ตระกูล FLEX 10 K จะเทียบโดยใช้วิธี LUT (Look Up Table) เมื่อทำการสังเคราะห์วงจรเสร็จแล้วซอร์ฟแวร์สังเคราะห์วงจรก็จะมีการรายงานผลว่าวงจรที่ออกแบบไปนั้นเป็นอย่างไร เช่น มีความหน่วง (delay) เท่าไร ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เป็นต้น

3.2.3 การแบ่งวงจร (Partitioning)

ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์ให้เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกันมีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้เพื่อช่วยลดความหนาแน่นในตอนทำการเชื่อมต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอร์ฟแวร์ทำ โดยซอร์ฟแวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิปฟลอป (flipflop) ลงในทรัพยากรต่างๆ ที่มีอยู่ภายในอุปกรณ์ FPGA (CLBs, IOBs, BUFT และ edge decoder) หลังจากทำขั้นตอนนี้เสร็จแล้วสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนซอร์ฟแวร์ที่ใช้ในขั้นตอนนี้ขึ้นอยู่กับตัว FPGA ที่ใช้งาน เช่น FPGA ของบริษัท Xilinx จะใช้ Xilinx Foundation Series 2.1i ซึ่งซอร์ฟแวร์ตัวนี้จะรวมเอาซอร์ฟแวร์ย่อยอื่นๆอีก เพื่อให้การทำ PPR (Partitioning, Placement and Routing) เป็นไปอย่างต่อเนื่อง ส่วน FPGA ของบริษัท Altera จะใช้ Altera MAX+II

3.2.4 การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (partitioning) มาแล้วว่าควรจะอยู่ในตำแหน่งใดในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ที่ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กันเพื่อจะได้ค้นหาเส้นทาง (route) ได้ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมีความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือตัว Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด

3.2.5 การเชื่อมต่อสัญญาณ (Routing)

ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA เช่น ระหว่าง CLBs หรือระหว่าง CLBs กับ IOBs ขั้นตอนนี้จะทำต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอร์ฟแวร์ก็จะทำการเชื่อมต่อสัญญาณได้ไม่หมดหรือเกิดความหน่วงเกินค่าที่กำหนดในข้อบังคับ โดยสามารถทำขั้นตอนนี้ได้โดยใช้ซอร์ฟแวร์ เช่นกัน หรือจะทำการเชื่อมต่อสัญญาณด้วยตัวเอง (manual layout) ก็ได้ แต่ทางที่ดีควรใช้ซอร์ฟแวร์ทำดีกว่าโดยให้ทำการค้นหาเส้นทางหลายๆครั้งเพื่อหาครั้งที่ดีที่สุด นอกจากนั้นการกำหนดข้อบังคับทางเวลา (time constraints) จะช่วยให้ผลที่ได้จากการทำการเชื่อมต่อสัญญาณดีขึ้นได้

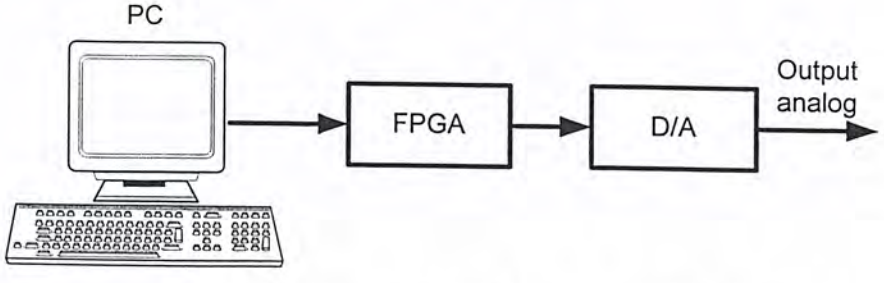
3.2.6 การโปรแกรมอุปกรณ์ FPGA (Configuration)

หลังจากที่วงจรผ่านขั้นตอนต่างๆจนกระทั่งผ่านการทำ PPR (Partitioning, Placement and Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGA ได้แล้ว ในการดาวน์โหลดนี้ก่อนอื่นต้องแปลงแบบวงจรรวมที่ได้ให้เป็นข้อมูลวงจร (Configuration data) ซึ่งอยู่ในรูปของบิทสตรีม (Bit-stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มีฟังก์ชั่นการทำงานตามวงจรที่ออกแบบไว้

จากที่อธิบายมาทั้งหมดจะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้น ทำได้สะดวกกว่าการทำ ASIC มากเพราะใช้เวลาน้อยกว่ามาก ส่วนสำคัญที่ใช้ในการทำ FPGA คือซอร์ฟแวร์ที่ใช้ตั้งแต่การเขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงในอุปกรณ์ FPGA ซึ่งซอร์ฟแวร์ที่ใช้ต้องเป็นซอร์ฟแวร์ที่ใช้ทำงานต่อเนื่องกัน

3.3 การออกแบบ

การออกแบบตัวกำเนิดสัญญาณไซน์แบบดิจิตอลด้วย FPGAs เป็นการออกแบบโดยการนำเอา personal Computer(PC) กับ FPGAs ต่อทำงานร่วมกัน ดังรูปที่ 3.3



รูปที่ 3.3 ขั้นตอนการทำงานของ ตัวกำเนิดสัญญาไซน์

โดยสามารถอธิบายการทำได้ดังนี้ใช้คอมพิวเตอร์ทำหน้าที่รับข้อมูลจากคีย์บอร์ด (keyboard)พร้อมทั้งแสดงข้อมูลออกมาทางหน้าจอคอมพิวเตอร์โดยผ่านโปรแกรมวิชวลเบสิก(Visual Basic) แล้วทำการส่งค่าไปยังอุปกรณ์ FPGA ที่ทำหน้าที่เป็นตัวประมวลผลทางดิจิตอลดังจะได้อธิบายต่อไป แล้วนำค่าที่ประมวลผลได้ส่งค่าไปยัง D/A จะได้เอาท์พุท ออกมาเป็นสัญญาณไซน์ (sinusoidal) ตามต้องการ

ส่วนของ FPGA จะสร้างฮาร์ดแวร์โดยอาศัยการอธิบายพฤติกรรมการทำงานของวงจรด้วยภาษาวีเอชดีแอล(VHDL)จะประกอบไปด้วย 2 ส่วนคือส่วนของโครงสร้างของ CORDIC และส่วนของโครงสร้างของวงจรกำเนิดสัญญาณดิจิตอลที่มีโครงสร้างแบบตรง(Direct Digital Frequency Synthesizer)

3.3.1 การออกแบบโครงสร้างของ CORDIC Algorithm

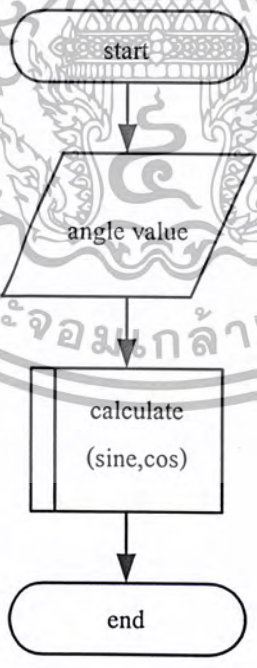
จากหลักการของ CORDIC Algorithm ซึ่งแสดงในรูปสมการพื้นฐานคือ

$$x_{i+1} = x_i \cos\theta_i - y_i \sin\theta_i$$

$$y_{i+1} = x_i \sin\theta_i + y_i \cos\theta_i$$

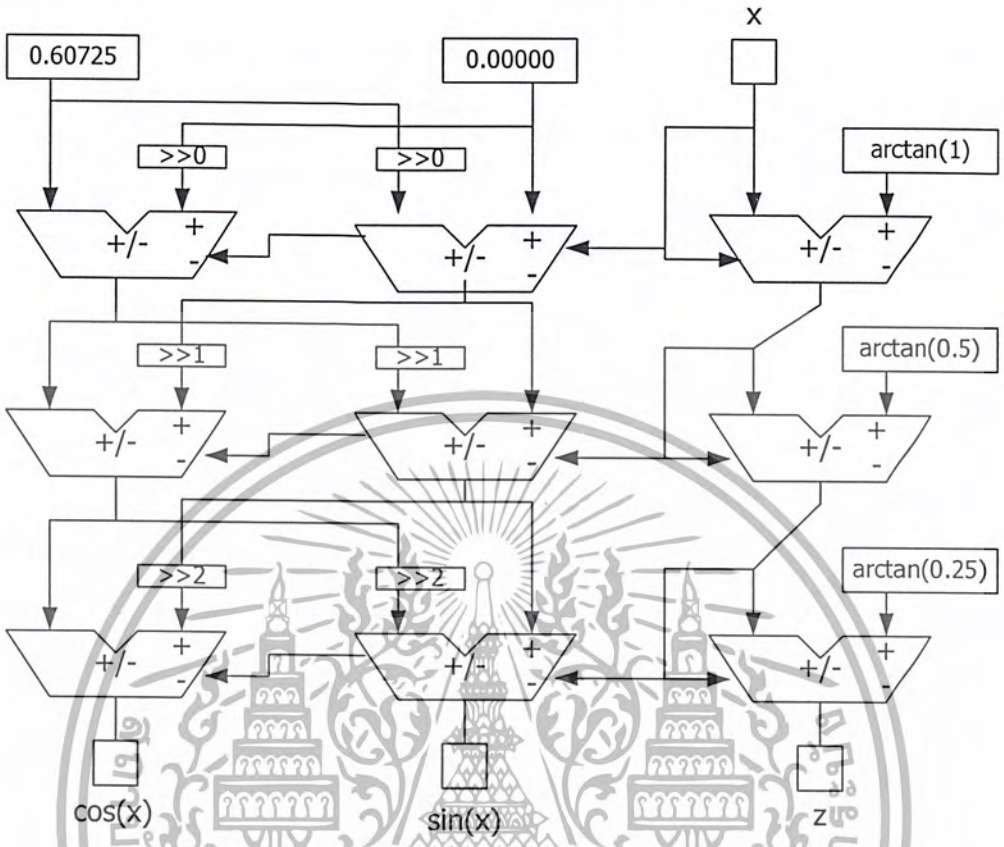
สามารถเขียนในรูปMatrix

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$



รูปที่ 3.4 การออกแบบโครงสร้างของ CORDIC Algorithm

จาก Flow Chat สามารถอธิบายการทำงานได้ดังนี้ ส่วนของ CORDIC จะรับค่ามุมมาจากคอมพิวเตอร์จะคำนวณหาค่า cos θ และ sin θ เพื่อกำเนิดสัญญาณดิจิตอล



รูปที่ 3.5 โครงสร้างของ CORDIC ใช้คำนวณ SIN,COS

ตัวอย่างของการหาค่า COS และ SIN จากโครงสร้างของ CORDIC โดยใช้โปรแกรม MATLAB

มุม (องศา)	COSθ	SINθ
30	0.8660	0.500
45	0.7071	0.7071
60	0.5000	0.8660
90	-7.1115e-008	1.000

ตารางที่ 3.1 การหาค่ามุมจากโครงสร้าง CORDIC

3.4 การเขียนโปรแกรมวิชวลเบสิกเพื่อใช้งานพอร์ตอนุกรม

3.4.1 คอนโทรล MSComm

สำหรับการใช้งานวิชวลเบสิกตั้งแต่เวอร์ชัน 2 เป็นต้นมา ในวิชวลเบสิกจะมีคัสตอมคอนโทรลสำหรับการสื่อสารอนุกรมผ่านทางพอร์ตอนุกรมของคอมพิวเตอร์มาให้ โดยในวิชวลเบสิก เวอร์ชัน 2 และเวอร์ชัน 3 จะใช้ชื่อว่า MSCOMM.VBX ส่วนเวอร์ชัน 4 ใช้ชื่อว่า MSCOMM16.OCX สำหรับการทำงานกับระบบปฏิบัติการ 16 บิต และ MSCOMM32.OCX สำหรับการทำงานกับระบบปฏิบัติการ 32 บิต สำหรับในวิชวลเบสิก เวอร์ชัน 5 จะมีเพียง MSCOMM32.OCX เท่านั้นเพราะถูกออกแบบมาให้ใช้งานกับระบบปฏิบัติการ 32 บิต

MSComm จัดเตรียมทางเลือกเอาไว้ 2 ทางเพื่อความสะดวกในการสื่อสารข้อมูลทางแรกคือการสื่อสารข้อมูลที่กระตุ้นด้วยเหตุการณ์ (event-driven communications) เป็นรูปแบบการใช้งานที่มีประสิทธิภาพมากสำหรับการตอบสนองแบบทันทีทันใด เช่น เมื่อตัวอักษรถูกส่งมาที่พอร์ตอนุกรมหรือเกิดการเปลี่ยนแปลงที่ขา Data Carrier Detect (DCD) หรือขา Request To Send RST) เหตุการณ์ Oncomm ของ MSComm จะสามารถตรวจจับสัญญาณได้ทันที ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อคุณสมบัติ CommEvent ต่อไป ส่วนทางเลือกที่สองเป็นการคอยตรวจสอบค่าเหตุการณ์และความผิดพลาดที่เกิดขึ้นด้วยการดูค่าที่เปลี่ยนแปลงภายในคุณสมบัติ CommEvent หลังจากให้โปรแกรมทำงานในฟังก์ชันต่าง ๆ ไปเรียบร้อยแล้วซึ่งวิธีใช้งานได้ดีในกรณีที่โปรแกรมมีขนาดเล็ก

คอนโทรล MSComm 1 ตัวสามารถควบคุมการทำงานของพอร์ตอนุกรมได้ 1 พอร์ต ถ้าในโปรแกรมที่ใช้งานต้องการติดต่อพอร์ตอนุกรมมากกว่า 1 พอร์ตจะต้องใช้คอลโทรล MSComm มากกว่า 1 ตัว เพื่อควบคุมพอร์ตอนุกรมในแต่ละพอร์ต โดยแอดเดรสของพอร์ตอนุกรมและ แอสแดรสของการเกิดอินเทอร์รัปต์สามารถเปลี่ยนแปลงได้จากการแก้ไขค่าที่ Control Panel

ถึงแม้ว่า คอนโทรล MSComm จะมีคุณสมบัติ (property) มากมาย แต่สามารถทำความ เข้าใจได้ไม่ยากดังนี้

3.4.2 CommPort

ใช้ในการกำหนดและอ่านค่าพอร์ตอนุกรมที่ติดต่ออยู่ (COM1, COM2 , COM3, Com4)

รูปแบบการใช้งาน

object, CommPort[= value]

โดย Value เป็นค่าของพอร์ตอนุกรม ชนิดของข้อมูลเป็น Integer ค่า Value สามารถกำหนดได้ในช่วง 1-16 (ค่าเริ่มต้นกำหนดไว้ที่ 1) เมื่อมีการกำหนดค่าแล้วทำการเปิดพอร์ตโดยใช้คุณสมบัติ PortOpen แต่ว่าพอร์ตนั้นไม่มีอยู่ในระบบ MSComm จะสร้างสัญญาณแสดงข้อผิดพลาด error 68 ขึ้นมา ซึ่งหมายถึงอุปกรณ์ตัวนี้ไม่มีอยู่ในระบบ ดังนั้นการเขียนโปรแกรมจึงต้องกำหนดตำแหน่งของพอร์ตอนุกรมก่อนที่จะใช้คำสั่ง OpenPort

3.4.3 Setting

ในการกำหนดและอ่านค่าอัตราบอด, พาริตี้, จำนวนบิตข้อมูล, จำนวนบิตปิดท้าย

รูปแบบการทำงาน

object.Setting [= value]

ค่า Value มีชนิดข้อมูลเป็นแบบ String มีรูปแบบเป็น "BBBB,P,D,S" โดย BBBB เป็นค่าอัตราบอด, P เป็นค่าพาริตี้, D เป็นจำนวนของบิตข้อมูล และ S เป็นจำนวนข้อมูลของบิตปิดท้าย ปกติแล้วค่านี้ถูกกำหนดไว้เป็น "9600,N,8,1"

ค่าบอดเรตมาตรฐานที่ใช้กับ MSComm มีดังนี้

110 บิตต่อวินาที

300 บิตต่อวินาที

600 บิตต่อวินาที

1200 บิตต่อวินาที

2400 บิตต่อวินาที

9600 บิตต่อวินาที (ค่าปกติ)

14400 บิตต่อวินาที

19200 บิตต่อวินาที

28800 บิตต่อวินาที

38400 บิตต่อวินาที (สงวน)

56000 บิตต่อวินาที (สงวน)

12800 บิตต่อวินาที (สงวน)

256000 บิตต่อวินาที (สงวน)

สำหรับค่ามาตรฐานในการกำหนดค่าพาริตี้มีดังนี้

สัญลักษณ์	รายละเอียด
E	พาริตี้คู่ (Event)
M	ลอจิก "1" (Mark)
N	ไม่ใช้ (ค่าปกติ)
O	พาริตี้ (Odd)
S	ลอจิก "0" (space)

ค่าที่ใช้ในการกำหนดจำนวนบิตมี 5 ค่าคือ 4,5,6,7 และ 8 (เป็นค่าปกติ)

ค่าที่ระบุระบุจำนวนบิตปิดท้ายมี 3 ค่าคือ 1 (เป็นค่าปกติ) ,1,5 และ 2

ตัวอย่างการใช้งานคำสั่ง Setting โดยจะเป็นการกำหนดค่าบอดเรตเท่ากับ 9600 ไม่มีพาริตี้ จำนวนข้อมูล 8 บิต และบิตปิดท้าย 1 บิต สามารถเขียนโปรแกรมได้ดังนี้

MSComm.Setting = "9600, N, 8, 1"

หมายเหตุ สาเหตุที่ค่ากำหนดจะต้องอยู่ภายในเครื่องหมายคำพูด "" เนื่องจาก ค่าที่กำหนดนี้อยู่ในรูป ตัวแปร String

3.4.4 PortOpen

ใช้ในการกำหนดและอ่านค่าสถานะของพอร์ตอนุกรม เพื่อเปิดและปิดพอร์ตอนุกรม

รูปแบบการใช้งาน

object.PortOpen [= value]

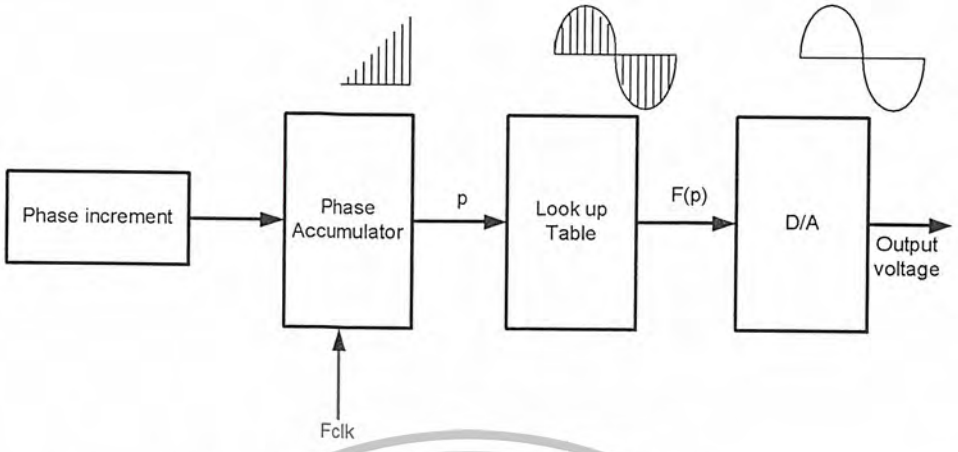
ค่า Value มีชนิดข้อมูลเป็นแบบบูลีน คือ True และ False โดย True หมายถึงการเปิดพอร์ตอนุกรม และ Falseหมายถึงการปิดพอร์ตอนุกรม สำหรับการปิดพอร์ตนั้นจะมีการเคลียร์บัฟเฟอร์รับข้อมูลและบัฟเฟอร์ส่งข้อมูลด้วย คอนโทรล MSComm จะปิดพอร์ตอนุกรมโดยอัตโนมัติเมื่อออกจากโปรแกรม ก่อนที่จะใช้คุณสมบัติ PortOpen ต้องตรวจสอบให้แน่ใจก่อนว่าคุณสมบัติ CommPort นั้นได้ทำการกำหนดตำแหน่งของพอร์ตอนุกรมไว้ถูกต้องหรือไม่ มิเช่นนั้น MSComm จะแสดงข้อผิดพลาด error 68 แจ้งแก่ผู้ใช้งาน หรือถ้าพอร์ตอนุกรมนั้นถูกปิดเอาไว้แล้ว โปรแกรมก็จะแจ้งข้อผิดพลาดออกมาเช่นเดียวกัน

ถ้าคุณสมบัติ DTREnable หรือ RTSEnable ถูกกำหนดให้เป็น True ก่อนที่จะทำการเปิดพอร์ต ค่าคุณสมบัติ DTREnable หรือ RTSEnable จะถูกเซตเป็น False หลังจากเปิดพอร์ต แต่ถ้าเซตเป็น False หลังจากเปิดโปรแกรมแล้ว ค่าที่กำหนดไว้จะเป็นค่าเดิม

ตัวอย่างการใช้คำสั่งเปิดพอร์ต เพื่อติดต่อสื่อสารกับพอร์ตอนุกรม COM1 และมีบอดเรต 9600 บิตต่อวินาที ไม่มีพาริตี้ จำนวนข้อมูล 8 บิต และบิตปิดท้าย 1 บิต มีดังนี้

```
MSComm1.Setting = "9600, N, 8, 1"
MSComm1.Commport = 1
MSComm1.PortOpen = True
```

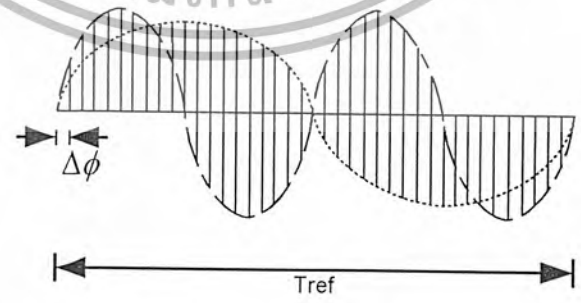
3.5 Direct Digital Frequency Synthesizer (DDS)



รูปที่ 3.6 แสดง Block Diagram โครงสร้างของวงจรสังเคราะห์ความถี่ที่ใช้หน่วยความจำเป็นค่าขนาดของสัญญาณเอาไว้

วงจรสังเคราะห์ความถี่ดิจิตอลแบบโดยตรง มีโครงสร้างหลัก ๆ ตามรูปที่ 3.6 จะอาศัยหน่วยความจำเพื่อเก็บค่าขนาดของสัญญาณที่เฟสต่าง ๆ เอาไว้ ซึ่งรูปสัญญาณที่เก็บเอาไว้ อาจจะเป็นรูปสัญญาณซายน์ (Sine) สัญญาณรูปฟันเลื่อย (sawtooth) หรือรูปสัญญาณอื่น ๆที่ผู้ใช้งานได้ทำการออกแบบไว้ เมื่อต้องการรูปสัญญาณออกมาที่เอาท์พุต ก็จะทำการเรียกข้อมูลที่เฟสต่าง ๆ ที่เก็บเอาไว้ออกมา เพื่อป้องกันให้กับวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอก ดังรูปที่ 3.6

หากต้องการจะเปลี่ยนความถี่ของสัญญาณก็ทำได้โดย การเปลี่ยนอัตราการเรียกข้อมูลออกจากหน่วยความจำ (เป็นความเร็วของสัญญาณนาฬิกา) หรือเปลี่ยนลำดับการเรียกข้อมูลที่เฟสต่าง ๆ เพื่อให้จำนวนของ ข้อมูลในหนึ่งคาบของสัญญาณเปลี่ยนไป ดังเช่นในรูปที่ 3.7 จะเห็นว่ามีรูปของสัญญาณสองความถี่ รูปสัญญาณที่มีความถี่สูงกว่า จะมีจำนวนข้อมูลที่ถูกเรียกออกมาน้อยกว่ารูปสัญญาณที่มีความถี่ต่ำกว่า



รูปที่ 3.7 วงจรสร้างสัญญาณซายน์แบบเชิงเลข

มีข้อสังเกตคือ ระยะ ระหว่างข้อมูลที่ถูกเรียกออกมาของสัญญาณทั้งสองความถี่มีค่าเท่ากัน ดังนั้นความละเอียดในการเปลี่ยนความถี่จะขึ้นอยู่กับขนาดของหน่วยความจำที่ใช้เก็บข้อมูล

หลักการของวงจรสังเคราะห์ความถี่ดิจิตอลแบบโดยตรง (Direct Digital Frequency Synthesizer) นั้นจะมีสัญญาณนาฬิกา (Fclk) นำมาป้อนให้กับส่วนของ Phase Accumulator ซึ่งวงจรนี้จะทำการบวกวนซ้ำด้วยค่าที่รับเข้ามาจากส่วน เพิ่มเฟส ผลของการบวกจะได้ค่าออกมาเป็น เฟสสะสม P ค่าสะสมเฟส P นี้จะมีลักษณะเพิ่มขึ้นด้วยจำนวนเท่ากันทุกครั้งที่ทำการบวก ตามจังหวะสัญญาณนาฬิกาอ้างอิง (Fclk) ค่าเฟสสะสม P จะนำไปชี้ในตารางเปิดดู (Look up table) ซึ่งได้จัดเก็บค่าขนาดของรูปสัญญาณไว้แล้ว ซึ่งจะได้สัญญาณ F(P) ออกมา ซึ่งเป็นค่าแบบดิจิตอล มาทำการเปลี่ยนเป็นค่า อนาลอกด้วยวงจร Digital – to – Analog Convertor (DAC) ก็จะได้ Output voltage เป็นสัญญาณแบบที่ต้องการ

ความถี่ที่สามารถกำเนิดได้จากโครงสร้างนี้ คือ

$$Fout\ (min) = Fclk / 2^n$$

$$Fout = (\text{Frequency Input} \times Fclk) / 2^n$$

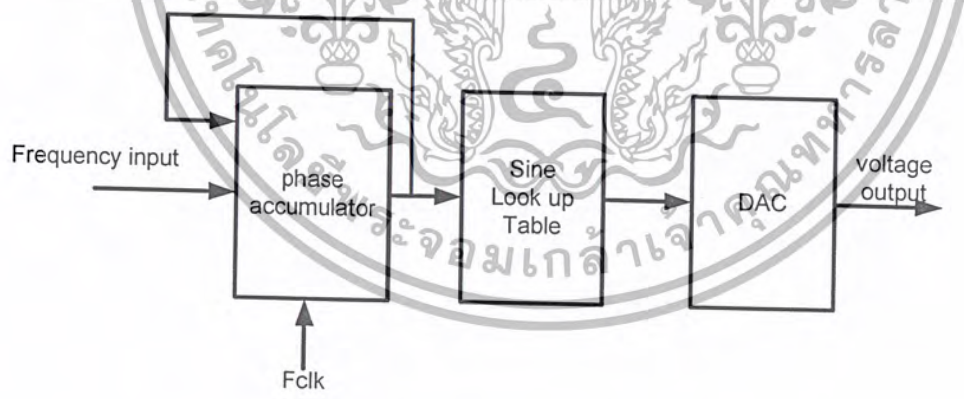
เมื่อ

Fout = Frequency Output

Fclk = Frequency Clock

N = จำนวน bit ของ Phase Accumulator

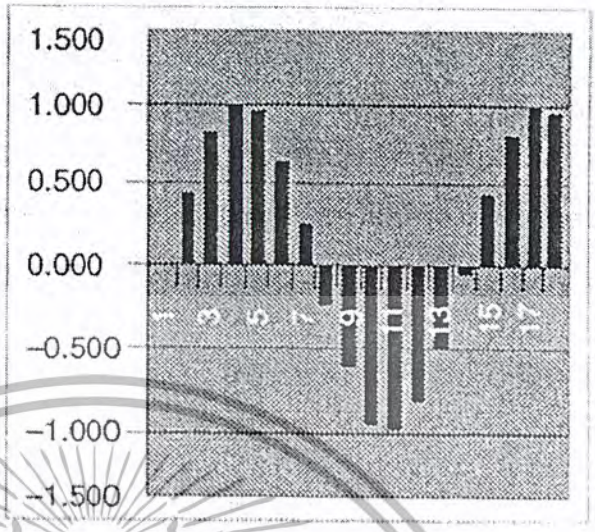
ถ้าค่า N มีค่ามาก Step size ของความถี่จะมีความละเอียดมากขึ้นด้วย



รูปที่ 3.8 แสดงการกำเนิดสัญญาณซายน์แบบดิจิตอล ที่มีค่า Phase Increment เท่ากับ 55

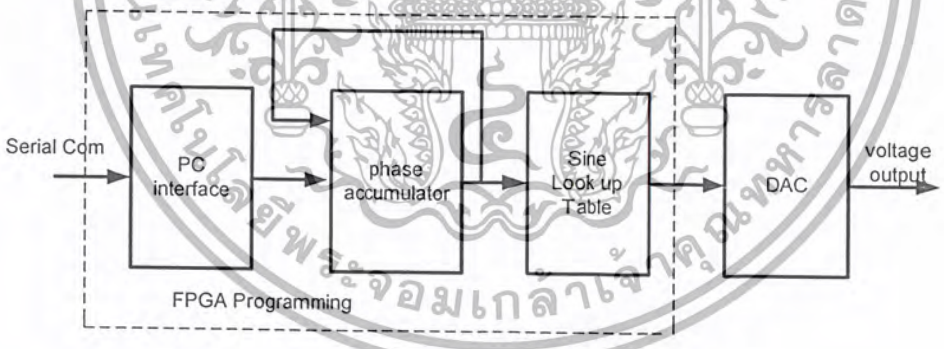
Phase increment register > 55

Time	Phase	Sinus
0	0	0.000
1	55	0.462
2	110	0.819
3	165	0.991
4	220	0.940
5	275	0.676
6	330	0.259
7	385	–0.216
8	440	–0.643
9	495	–0.924
10	550	–0.996
11	605	–0.843
12	660	–0.500
13	715	–0.044
14	770	–0.423
15	825	0.793
16	880	0.985
17	935	0.954



รูปที่ 3.9 Phase Increment เท่ากับ 55

3.5.1 การออกแบบ และการสร้าง



รูป.3.10 โครงสร้างการกำเนิดสัญญาณแบบตรง

ในการออกแบบและการสร้างโครงสร้างของ การกำเนิดสัญญาณแบบตรง (Direct Digital Frequency Synthesizers) ที่ออกแบบโดยใช้ VHDL เมื่อให้ FPGA ทำงานนั้นจะมีอยู่ 3 ส่วนหลัก ๆ คือ

. ส่วนที่ใช้ติดต่อกับคอมพิวเตอร์ (Computer Interface) ซึ่งส่วนนี้จะใช้ในการรับข้อมูล แบบ Asynchronous ซึ่งมีข้อมูล 8 bits และ Start bit 1 bit และ Stop bit 1 bit ผ่านทางพอร์ตอนุกรมของคอมพิวเตอร์ ในส่วนทาง Hardware นั้นจะต้องมีวงจรปรับระดับแรงดันก่อนที่จะส่งเข้า FPGA ด้วย

ในการออกแบบทางด้าน FPGA ของส่วนนี้ ก็จะประกอบไปด้วยส่วนหลัก ๆ คือ

1.1 ส่วนหารความถี่ เนื่องจากสัญญาณนาฬิกาที่ป้อนให้วงจรทำงานนั้นใช้ความถี่ 9.6 MHz แต่อัตราการส่งของข้อมูลที่ออกจากคอมพิวเตอร์ ผ่านทางพอร์ตอนุกรมนั้น เรากำหนดอัตราการรับส่งที่ 9600 บิต ต่อวินาที ดังนั้นต้องทำการหารสัญญาณนาฬิกา เพื่อให้ได้ 9600 Hz เพื่อให้การรับส่งข้อมูลเป็นไปอย่างถูกต้อง

1.2 ส่วนของการรับข้อมูล จะเป็นส่วนที่ใช้รับข้อมูลจากคอมพิวเตอร์ และจะมีการตรวจสอบ Stop bit เพื่อใช้เป็นสัญญาณ enable ให้กับส่วนของวงจร Latch ด้วย

1.3 ส่วนของวงจร Latch จะรับค่าเข้ามาเก็บไว้จนข้อมูลที่ส่งจากคอมพิวเตอร์รับได้หมด แล้วจึงจะทำการส่งข้อมูลไปให้ส่วนอื่น ๆ ที่ต้องใช้งานต่อไป

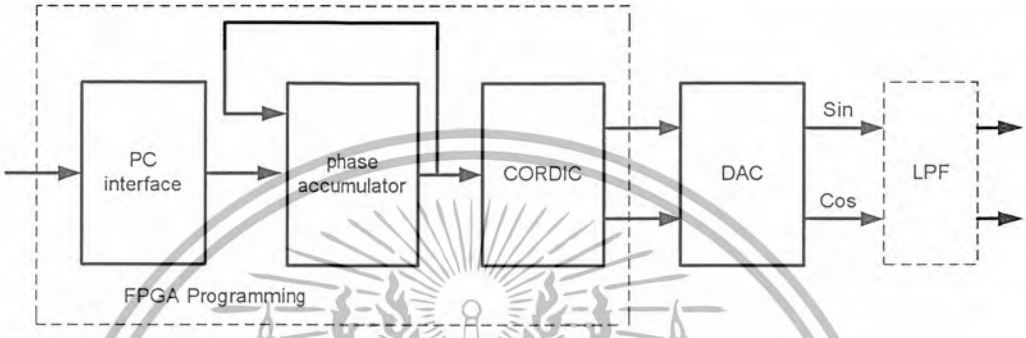
2. ส่วนของวงจร Phase Accumulator ในส่วนของ Phase Accumulator นี้ ได้ทำการออกแบบให้ทำการประมวลผลสัญญาณขนาด 24 bits และจะมีการส่งออกเป็น Output เพื่อไปชี้เป็น Address ของ Look up table ของสัญญาณที่ได้ออกแบบไว้

3. ส่วนของตารางเปิดดู (Look up table) เป็นส่วนที่ใช้ในการเก็บรูปแบบของสัญญาณที่เราจะทำการกำเนิดออกไป ซึ่งในส่วนของโครงสร้าง DDS ที่ออกแบบไว้นี้ จะกำเนิดรูปแบบสัญญาณได้ 5 รูปแบบสัญญาณ โดยรูปแบบสัญญาณไซน์ และสัญญาณสามเหลี่ยม จะมีการเก็บค่าไว้ในตารางเปิดดู โดยใช้เพียง 64 ค่า ส่วนของสัญญาณรูปสี่เหลี่ยม จะเก็บค่าไว้เพียง 2 ค่า คือ “0” และ “1” ในการกำเนิดสัญญาณแบบนี้ ทำได้โดยการตรวจสอบ bit สูงสุดที่รับเข้ามาจาก Phase Accumulator

สัญญาณรูปฟันเลื่อย ได้จากค่า Output ของ Phase Accumulator และสัญญาณสุ่มจะได้จากโครงสร้างของ Linear Feed Back Shift Register (LFSR) ซึ่งค่าสัญญาณทุกแบบจะใช้ 8 bit เพื่อส่งให้กับ Digital-to-Analog Converter ขนาด 8 bit

3.6 การสร้างและการออกแบบ CORDIC

ในการสร้างของการกำเนิดสัญญาณแบบ CORDIC นี้จะให้สัญญาณ Sine และ Cosine พร้อมกัน ซึ่งโครงสร้างเล็ก ๆ จะคล้ายกับโครงสร้างของการออกแบบวงจรกำเนิดสัญญาณแบบตรง ต่างกันตรงที่เราจะใช้ส่วนของ CORDIC นี้ไปทำงานแทนการใช้ Look up table ซึ่งค่าอินพุตของ CORDIC ก็ได้มาจากส่วนของวงจร Phase Accumulator ซึ่งมีขนาด 20 bit และการประมวลผลจะใช้โครงสร้างของ CORDIC แบบลำดับ ซึ่งใช้การประมวลผล 10 ลำดับ ก่อนที่จะได้ค่าสัญญาณ Sine และ Cosine ขนาด 8 bit ส่งให้ส่วนวงจร Digital-to-Analog Converter



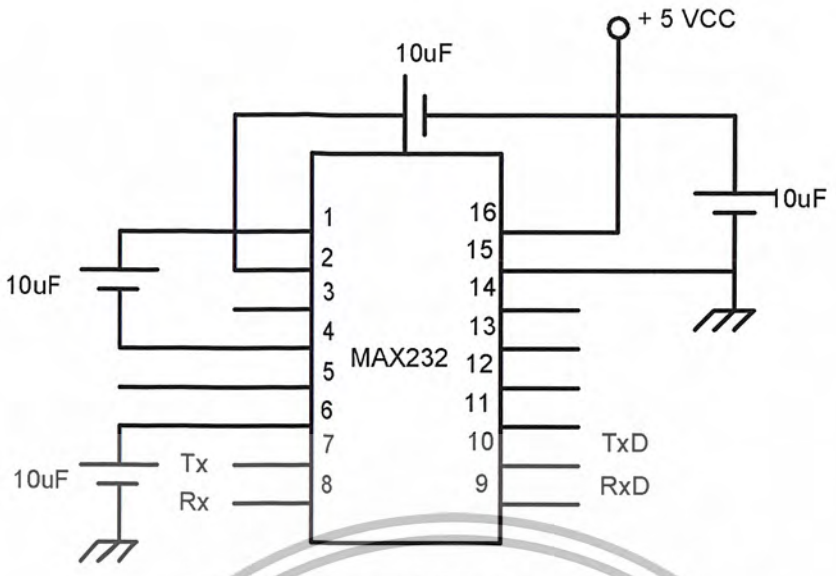
รูปที่3.11 แสดงการทำงานของวงจรกำเนิดสัญญาณ Sine และ Cosine โดยใช้ CORDIC Algorithm

3.7 วงจรแปลงระดับแรงดัน

ในการเชื่อมต่อวงจรที่ทำงานระดับแรงดันแบบทีทีแอล เข้ากับพอร์ต RS-232ของเครื่องคอมพิวเตอร์ที่มีระดับแรงดัน -15 โวลต์ ถึง +15 โวลต์ นั้นจะต้องมีวงจรพิเศษเพื่อทำการเปลี่ยนแปลงระดับแรงดันให้เหมาะสมซึ่งในที่นี้เราได้เลือกใช้ MAX 232 ซึ่งใช้อุปกรณ์ประกอบจากภายนอกน้อย คือ ใช้ C เพียง 5 ตัวเท่านั้น

3.7.1 หลักการทำงาน

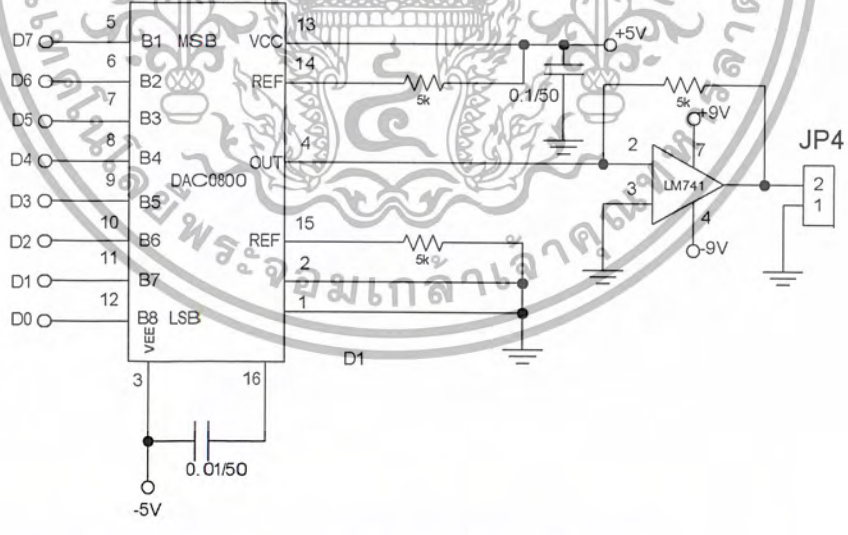
จากวงจรจะใช้ C ที่ต่อระหว่างขา 1 กับ 3, ระหว่างขา 4 กับ 5, ขา 2 กับ 6 เป็นตัวกำหนดระดับแรงดันที่ใช้ในการเชื่อมต่อโดยขา RII และ R2I จะเป็นขาที่รับระดับแรงดัน -15 โวลต์ ถึง +15 โวลต์ และแปลงออกเป็นแรงดัน 0 โวลต์ และ +5 โวลต์ ตามลำดับ ออกที่ขา R10 และ R20 ส่วนขา T1I และ T2I จะรับแรงดันที่เป็น 0 โวลต์ และ +5 โวลต์ ตามลำดับ แปลงเป็นระดับแรงดัน -10 โวลต์ +10 โวลจ์ ออกที่ขา T10 และ T20



รูปที่ 3.12 วงจรแปลงระดับแรงดัน เบอร์ MAX232

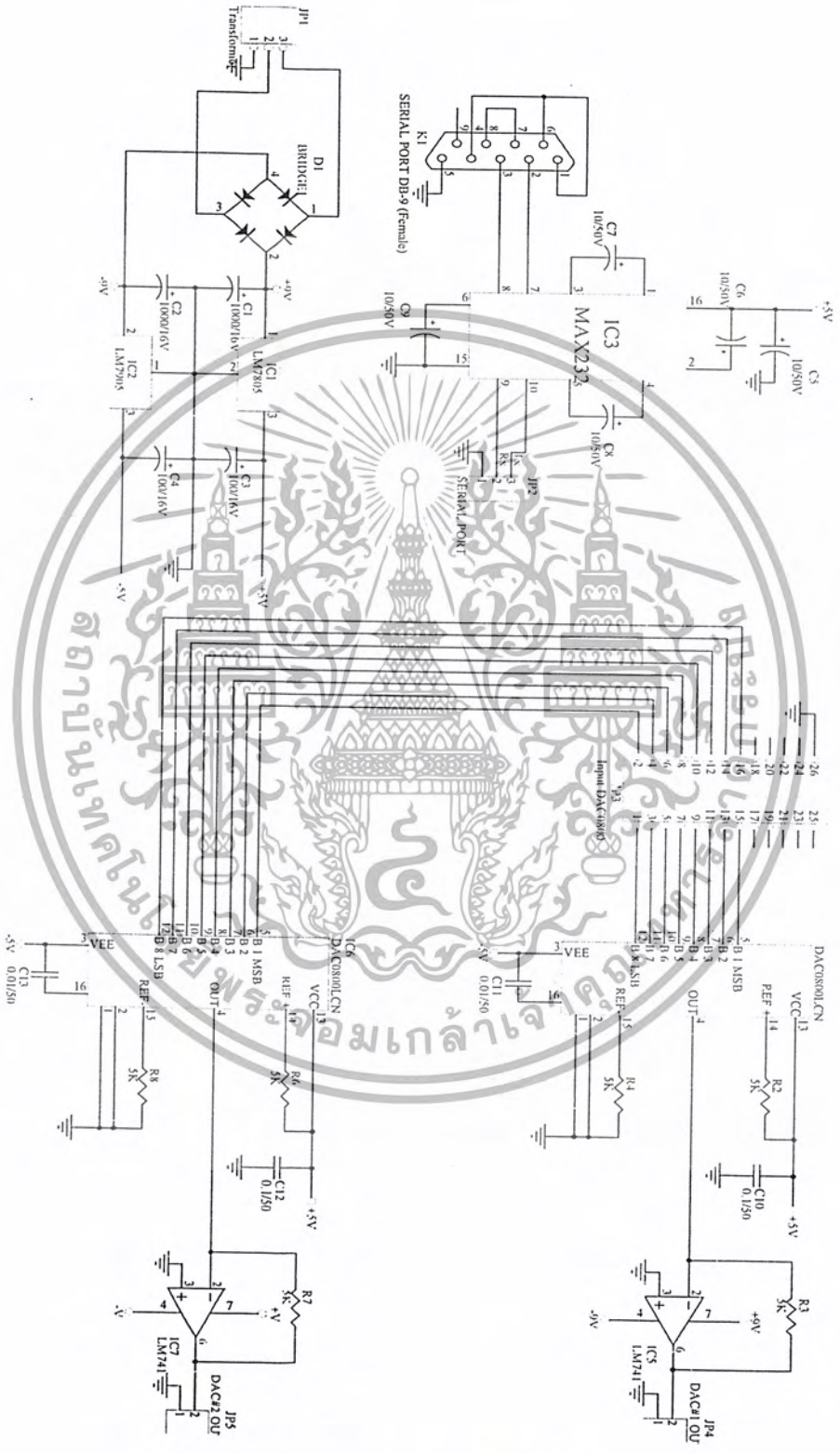
3.8 วงจรแปลงระดับสัญญาณดิจิตอลเป็นอนาล็อก

เป็นส่วนที่นำสัญญาณ Output จาก FPGA ที่เป็นสัญญาณแบบดิจิตอล ขนาด 8 bit มาแปลงให้สัญญาณ Analog โดยใช้ IC DAC 0800 วงจรใช้งานแสดงดังรูป

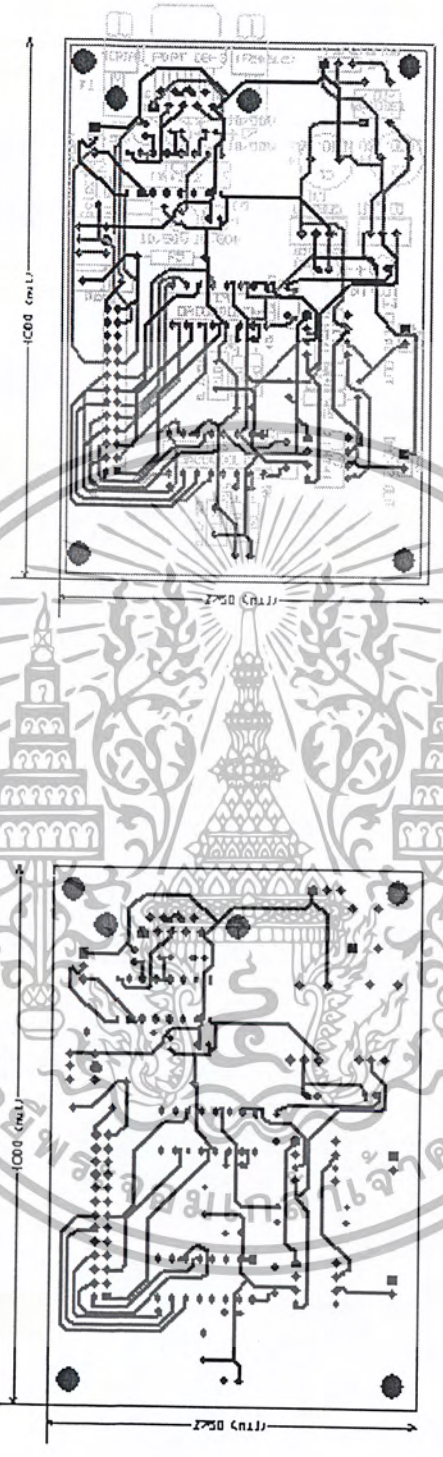


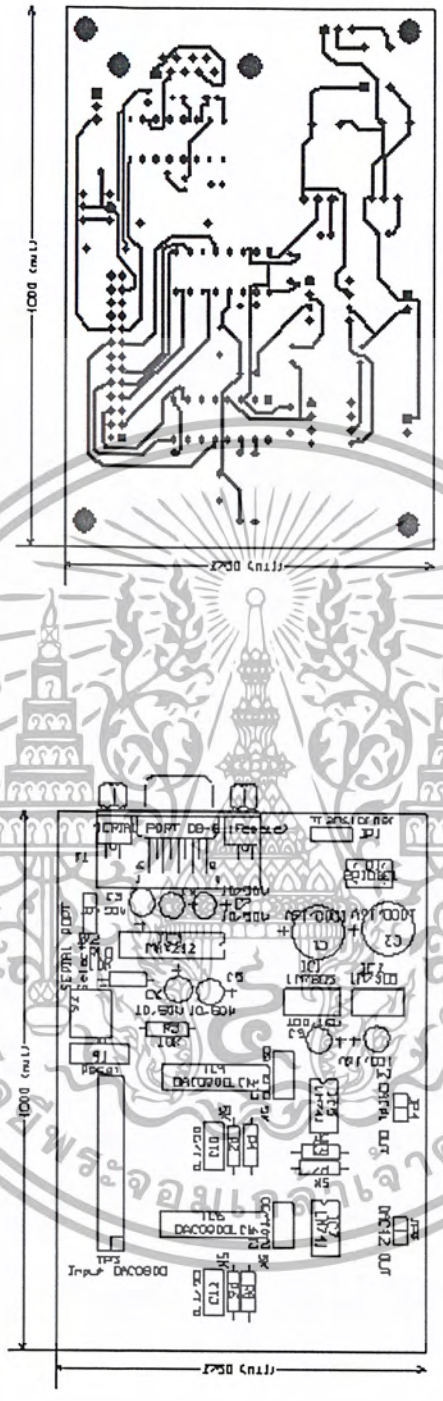
รูปที่ 3.13 วงจรแปลงระดับสัญญาณดิจิตอลเป็นอนาล็อก

3.9 การออกแบบลายวงจรและลายวงจร

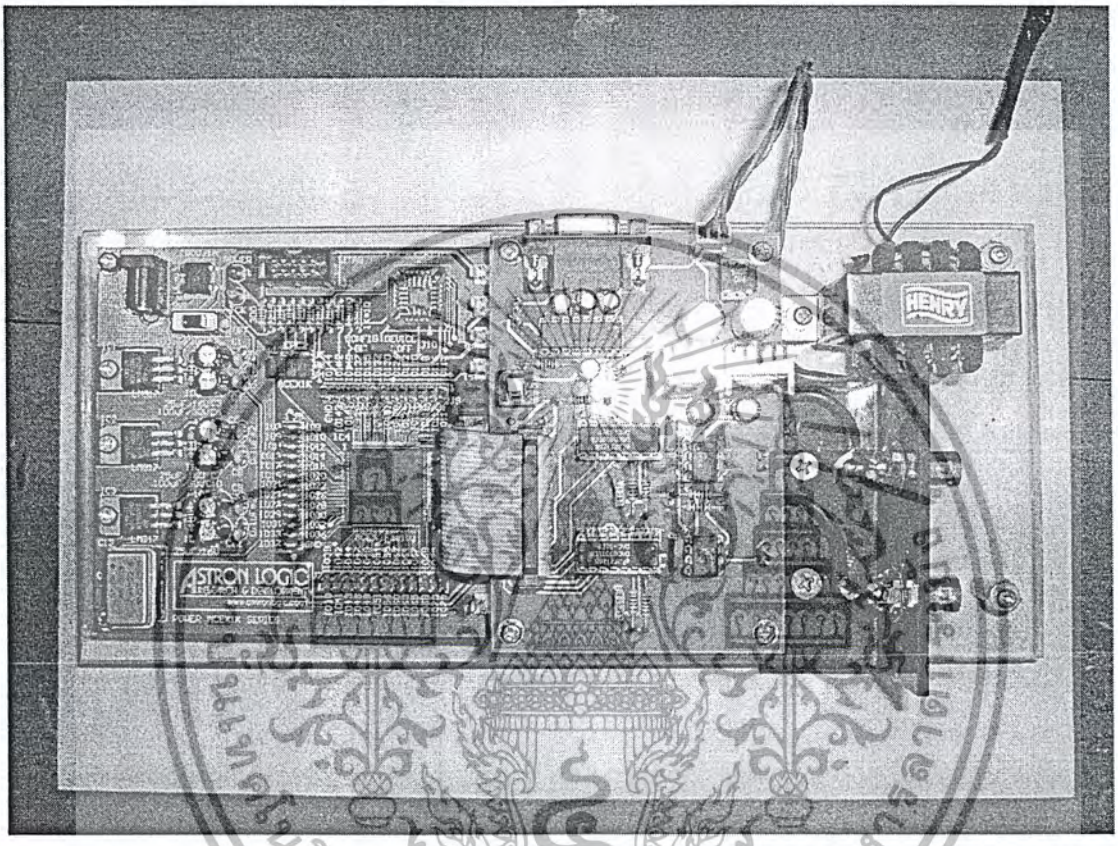


รูปที่ 3.14 วงจรรวมของ MAX 232 กับ DAC





รูปที่ 3.15 ลายวงจรและการวางอุปกรณ์



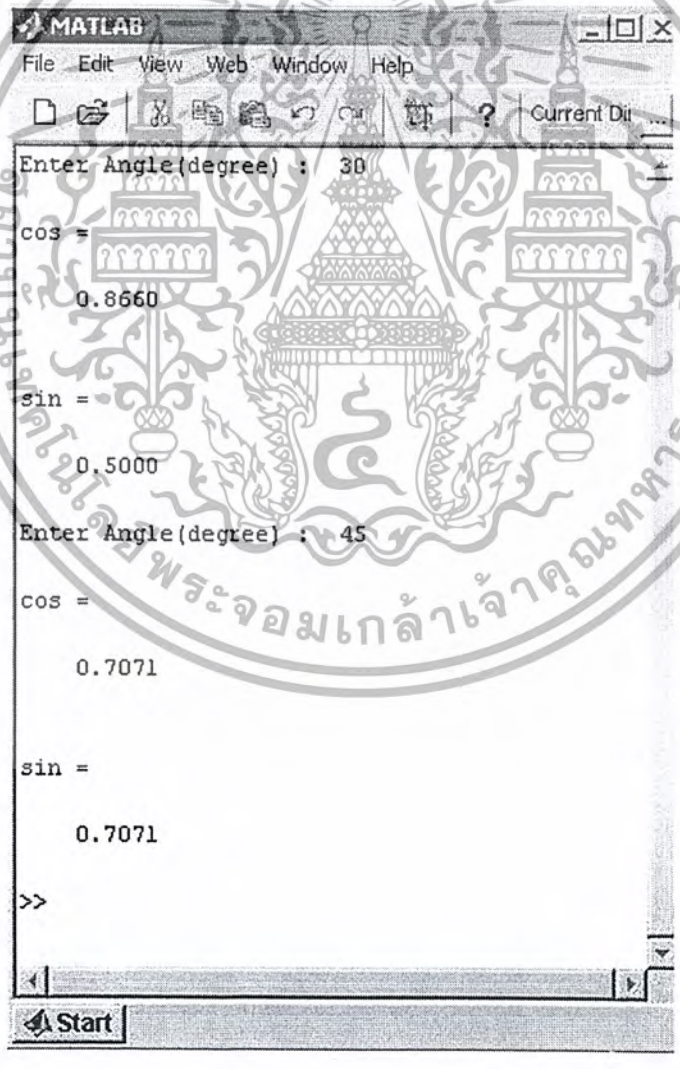
รูปที่ 3.16 ภาพถ่ายอุปกรณ์และวงจรใช้งานจริง

บทที่4
การทดลองและผลการทดลอง

การออกแบบวงจรกำเนิดสัญญาณไซน์แบบดิจิตอล ซึ่งประกอบด้วย 2 ส่วน คือ ส่วนของโครงสร้าง CORDIC และส่วนของโครงสร้างวงจรกำเนิดสัญญาณแบบดิจิตอล ที่มีโครงสร้างแบบคู่ควบ ในส่วนของ CORDIC จะทำการจำลองการทำงานโดยใช้โปรแกรม MATLAB และวงจรกำเนิดสัญญาณดิจิตอลที่มีโครงสร้างแบบคู่ควบนั้น จะใช้การอธิบายพฤติกรรมการทำงานของวงจร ด้วยภาษา VHDL และจำลองการทำงาน

4.1 การจำลองการทำงานส่วนของ CORDIC ด้วย MATLAB

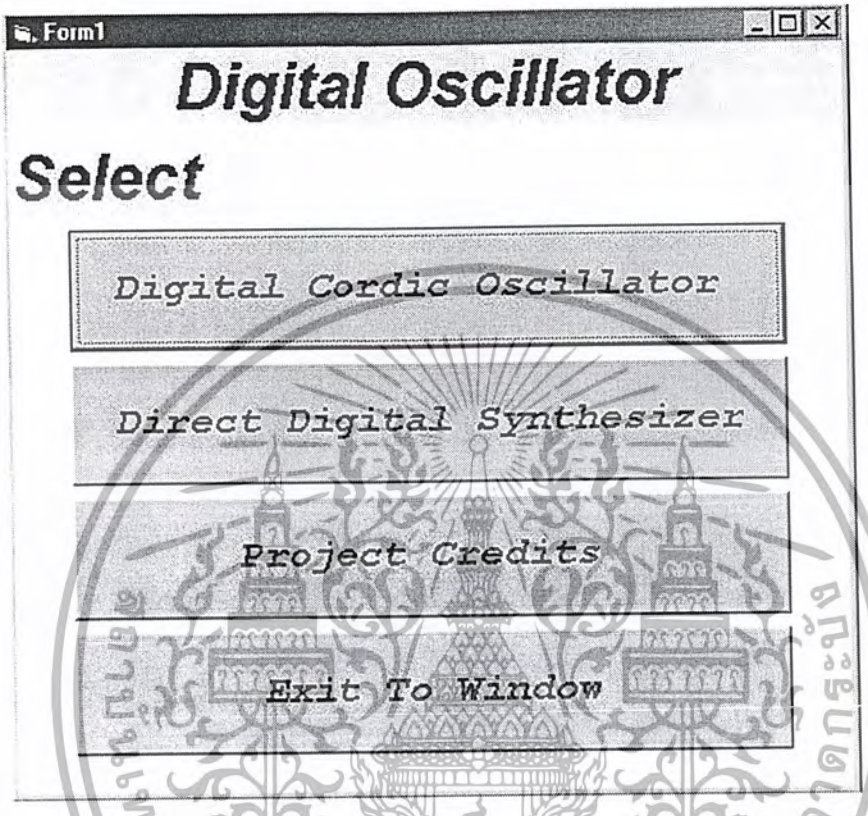
จะอาศัยโปรแกรม MATLAB ในการจำลองการทำงาน เพื่อหาค่ามุม COS θ และ SIN θ เพื่อเป็นค่าเริ่มต้นให้กับส่วนของวงจรกำเนิดสัญญาณดิจิตอล ที่มีโครงสร้างแบบคู่ควบ



รูปที่ 4.1 ผลการจำลองการทำงานของ CORDIC ในการหาค่า COS θ และ SIN θ โดยใช้ MATLAB

4.2 โปรแกรมการใช้งานการเชื่อมต่อผ่านพอร์ตอนุกรม โดยใช้วิชวลเบสิก

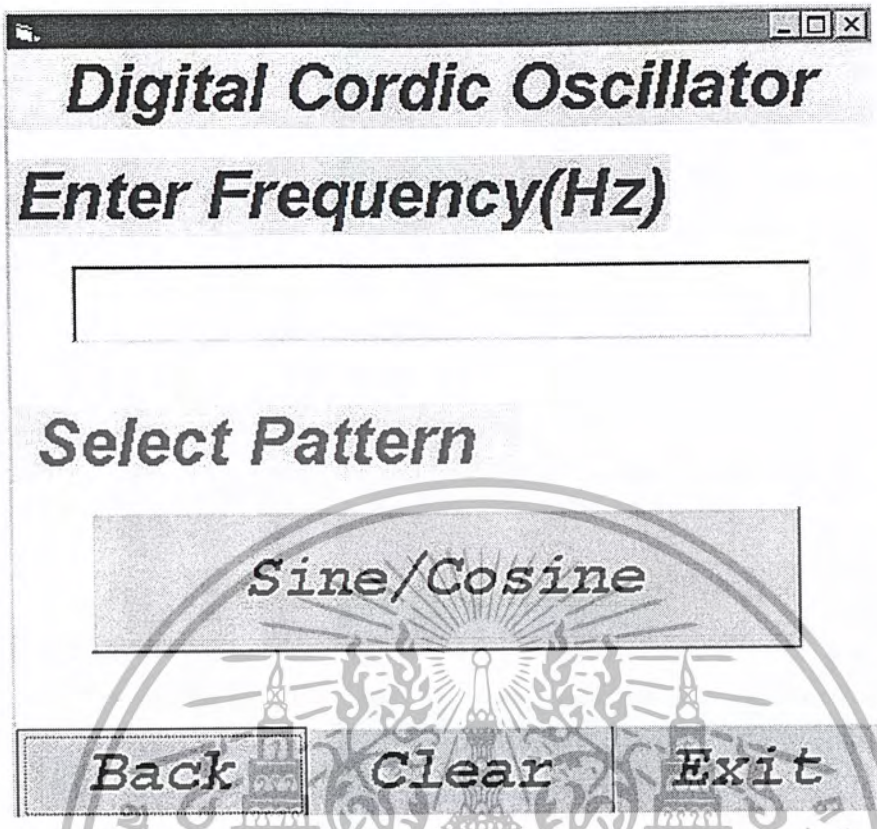
การทำงานของโปรแกรมที่เขียนด้วยวิชวลเบสิกมีความสะดวกในการเชื่อมต่อผ่านพอร์ตอนุกรม เราจึงนำข้อดีดังกล่าว มาประยุกต์ใช้ในการทำ Oscillator



รูปที่ 4.2 แสดง เมนูหลัก ของโปรแกรมการใช้งาน Digital Oscillator

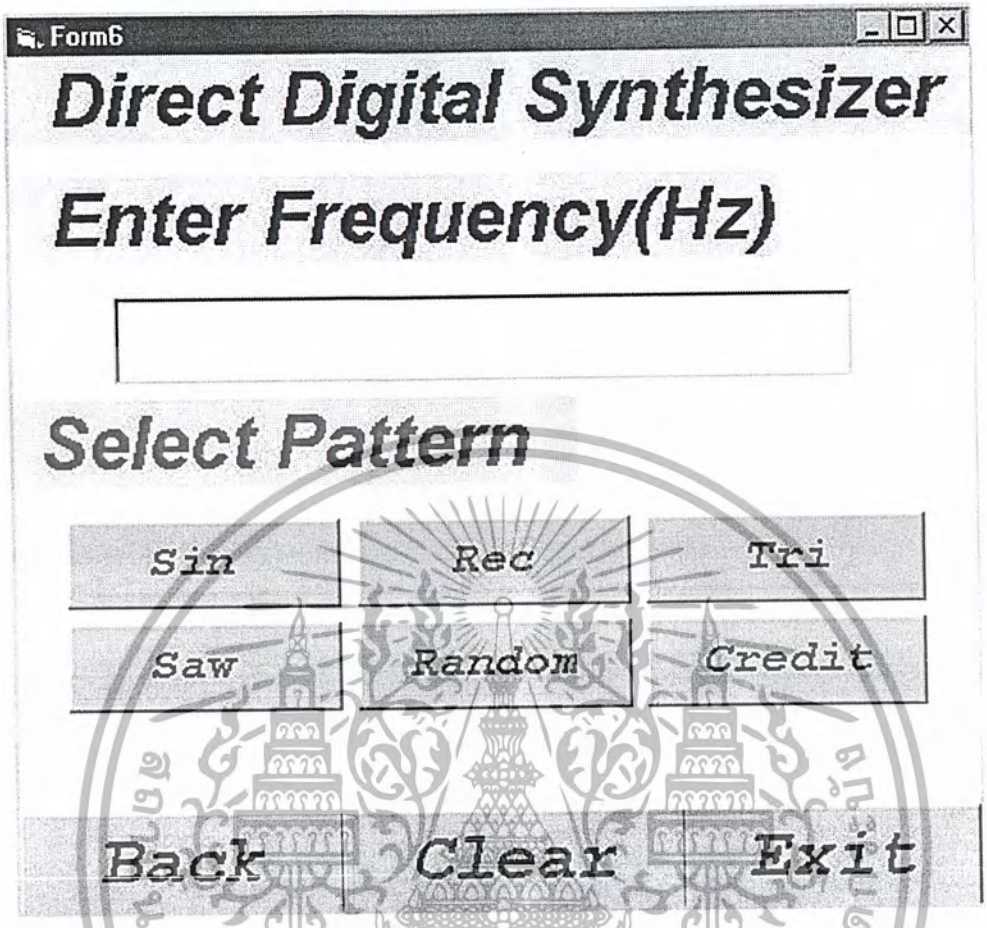
จากรูปที่ 4.2 ประกอบด้วย Text Block ที่ใช้สำหรับเชื่อมไปยัง Menu ต่าง ๆ ดังนี้

1. Digital Cordic Oscillator ใช้สำหรับในการเรียกใช้ Cordic Oscillator
2. Digital DDS Oscillator ใช้สำหรับในการเรียกใช้ DDS Oscillator
3. Project Credits ใช้สำหรับเรียกดูชื่อผู้จัดทำ และอาจารย์ที่ปรึกษา
4. Exit To Window ใช้สำหรับออกไปยัง Window



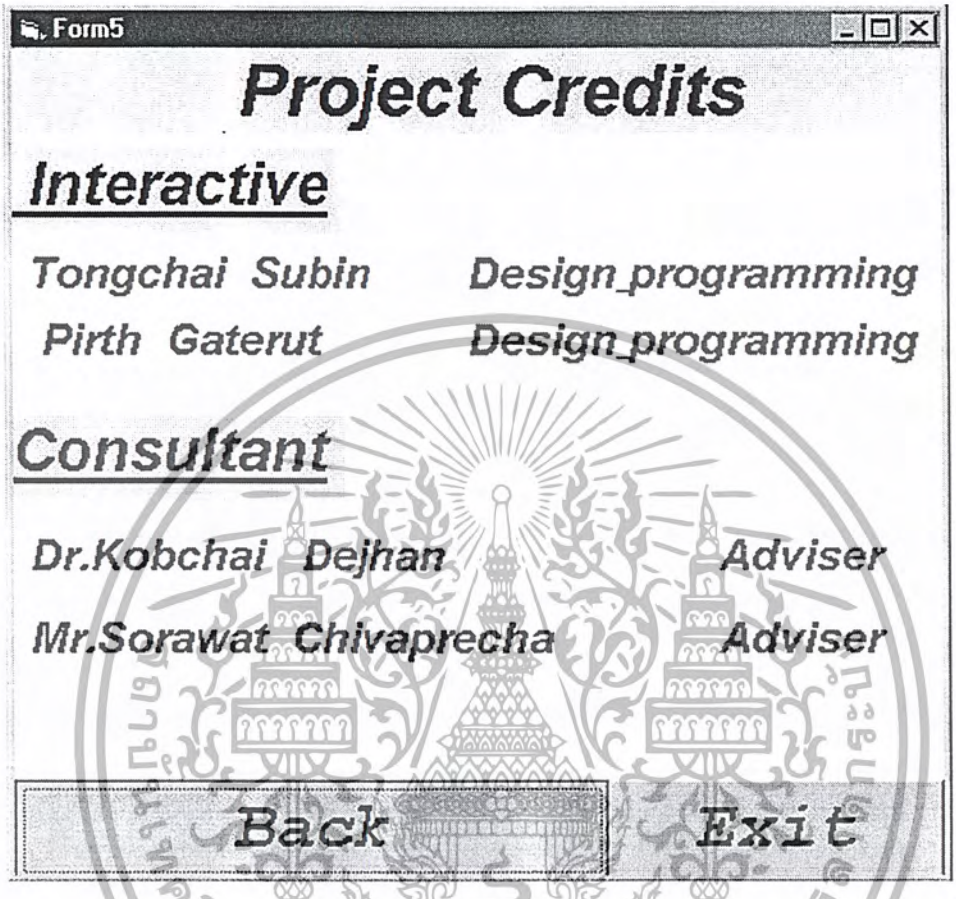
รูปที่ 4.3 แสดงเมนู ของ Digital Cordic Oscillator

จากรูปที่ 4.3 ทำการป้อนความถี่เพื่อที่จะสร้างสัญญาณตามที่ต้องการ โดยสามารถแสดงได้ทั้ง Sine และ Cosine



รูปที่ 4.4 แสดงเมนู ของ Direct Digital Synthesizer

จากรูปที่ 4.4 เราสามารถจะเลือกที่จะทำการกำเนิดสัญญาณในรูปแบบต่างๆ ได้ดังนี้ คือ สัญญาณซายน์ สัญญาณสี่เหลี่ยม สัญญาณสามเลี่ยม สัญญาณฟันเลื่อย สัญญาณสุ่ม ที่ความถี่ต่างๆ ได้ตามต้องการ



รูปที่ 4.5 แสดงเมนู Project Credits

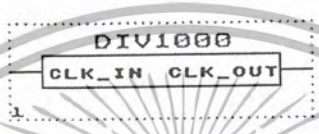
จากรูปที่ 4.5 แสดงรายละเอียดของผู้จัดทำและอาจารย์ที่ปรึกษาในส่วนของการทำโปรเจ็ค

4.3 การออกแบบวงจรส่วนต่างๆ โดยใช้ภาษา VHDL

4.3.1 ส่วนของวงจร DIV 1000

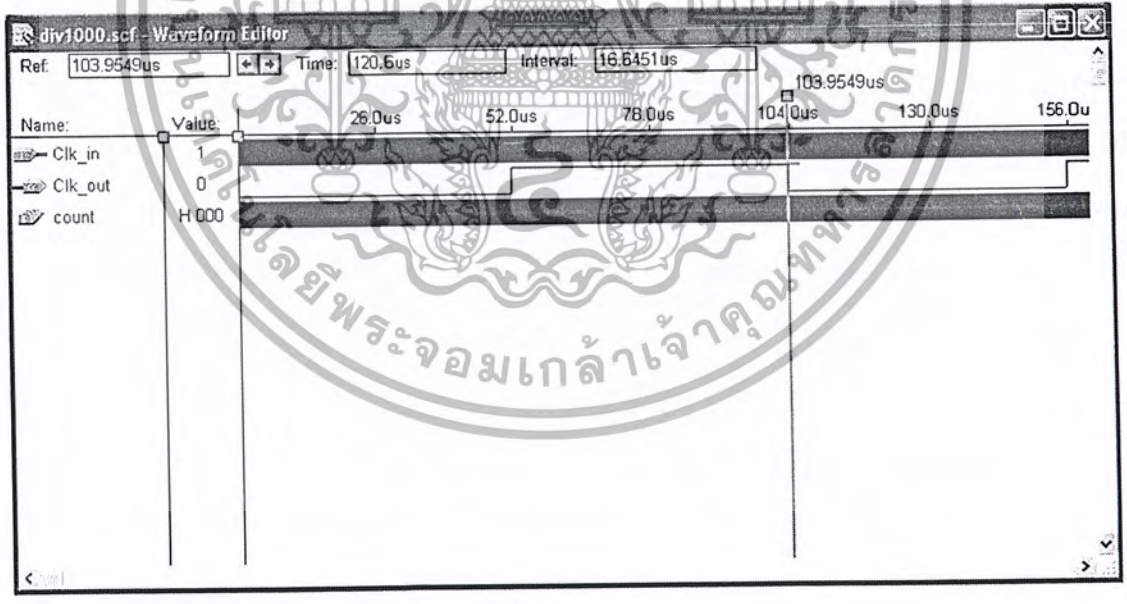
ทำหน้าที่เป็นโมดูลของวงจรหารความถี่ เนื่องจากภายในบอร์ด ACEX1K-50 ใช้แหล่งกำเนิดแบบ Module Oscillator ความถี่ 9.6MHz แต่ความถี่ที่ได้จะใช้เป็น Baud Rate ที่เราต้องการในการรับ มีค่าเท่ากับ 9600MHz เพราะฉะนั้นเราจะต้องทำการหารความถี่ที่ได้จาก Module Oscillator ลงให้เหลือเพียง 9600 MHz ซึ่งค่าของตัวหารจะเท่ากับ 9.6MHz / 9600 Hz เท่ากับ 1000

มีลักษณะดังรูป



รูปที่ 4.6 สัญลักษณ์ของส่วนวงจร DIV 1000

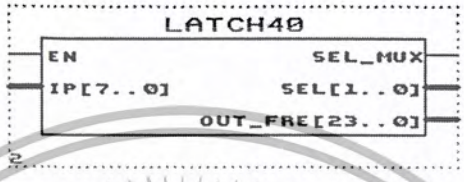
จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้



รูปที่ 4.7 ผลการจำลองการทำงานของส่วนวงจร DIV 1000

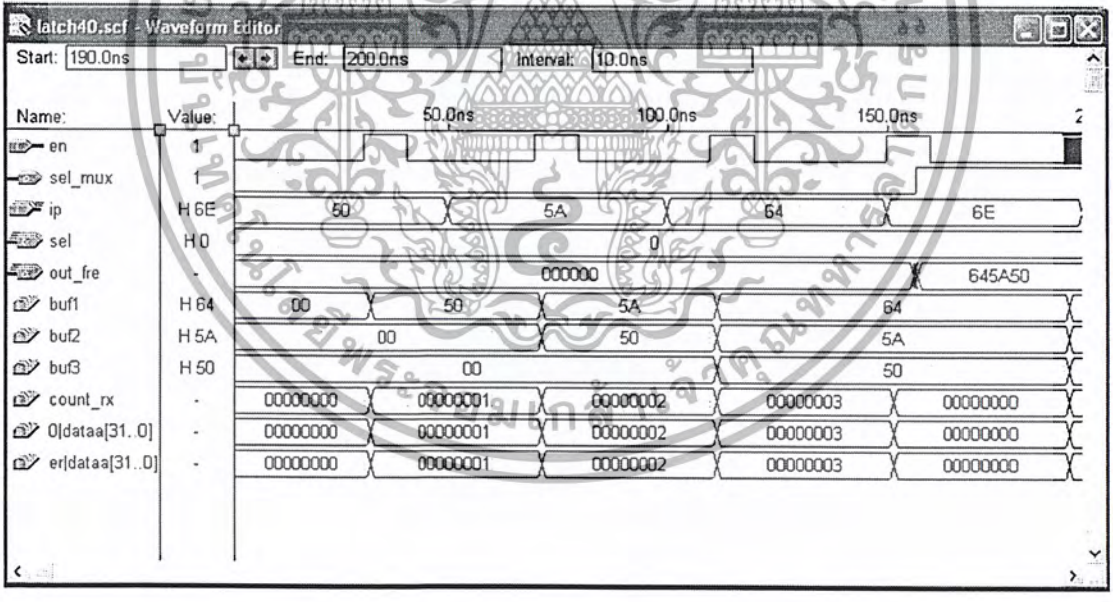
4.3.2 ส่วนของวงจร LATCH

เป็นโมดูลของวงจร D Flip Flop ทำหน้าที่ Latch ค่าข้อมูลที่รับมาได้ และส่งค่าที่ได้รับมาให้กับส่วนต่างๆ ที่จะนำไปใช้งาน มีลักษณะดังรูป



รูปที่ 4.8 สัญลักษณ์ของส่วนวงจร LATCH

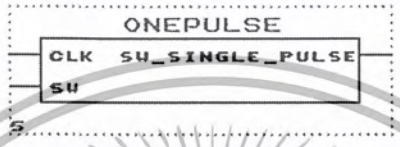
จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้



รูปที่ 4.9 ผลการจำลองการทำงานของส่วนวงจร LATCH

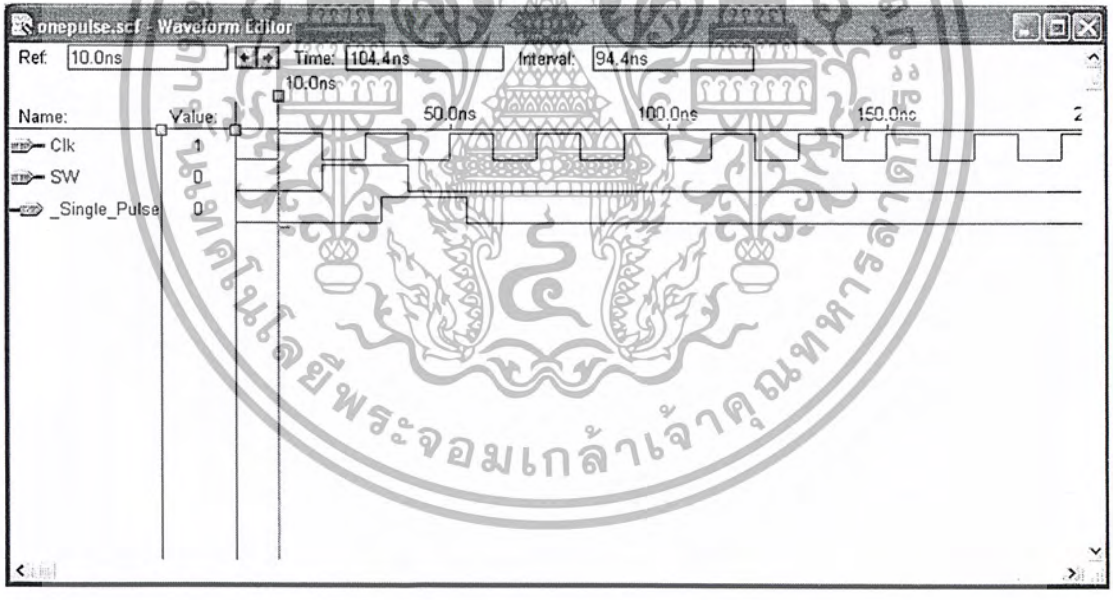
4.3.3 ส่วนของวงจร ONEPULSE

เป็นโมดูลรับค่าจากส่วนของ RX_EN ของส่วน SERIAL_COMMUNICATION โมดูลนี้จะทำการ Debounce สัญญาณที่รับและให้ Pulse ออกมา 1 ลูกเท่านั้น ขนาดของ Pulse จะเท่ากับความถี่ที่ต่ออยู่กับอินพุต CLK ซึ่งมีลักษณะดังรูป



รูปที่ 4.10 สัญลักษณ์ของส่วนวงจร ONEPULSE

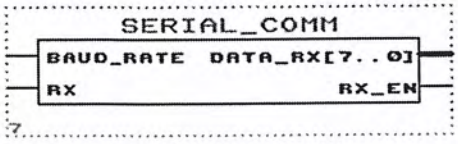
จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้



รูปที่ 4.11 ผลการจำลองการทำงานของส่วนวงจร ONEPULSE

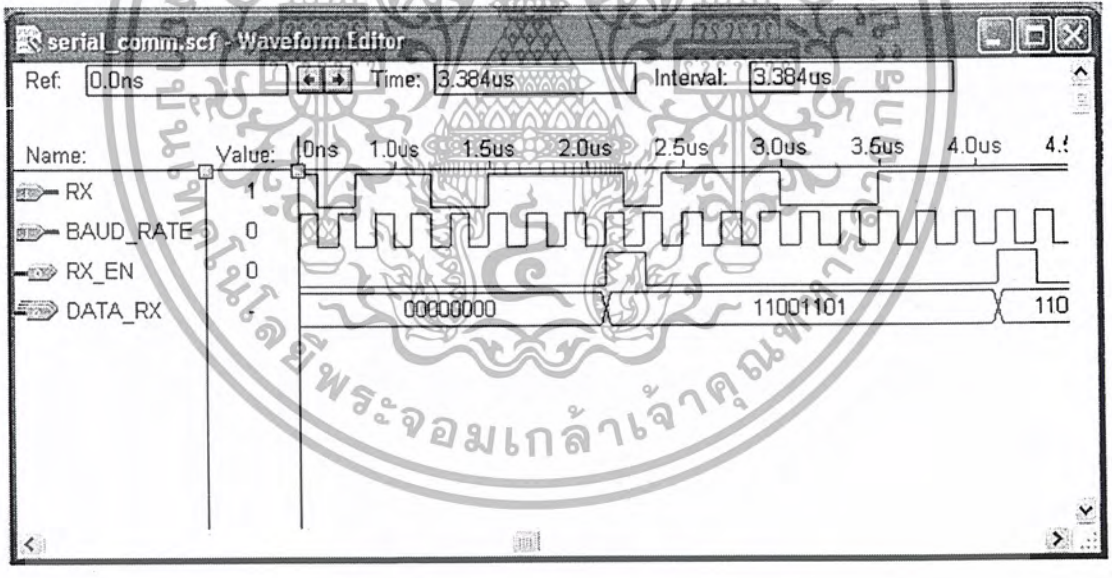
4.3.4 ส่วนของวงจร SERIAL_COMMUNICATION

เป็นโมดูลของวงจรรับ-ส่งข้อมูลแบบอนุกรม ที่รับค่ามาจากพอร์ตอนุกรมของคอมพิวเตอร์ และผ่านวงจรปรับระดับแรงดันก่อนที่จะส่งเข้ามาที่เอฟพีจีเอ มีลักษณะดังรูป



รูปที่ 4.12 สัญลักษณ์ของส่วนวงจร SERIAL_COMMUNICATION

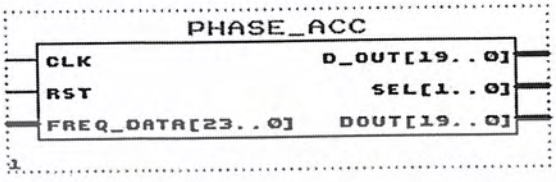
จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้



รูปที่ 4.13 ผลการจำลองการทำงานของส่วนวงจร SERIAL_COMMUNICATION

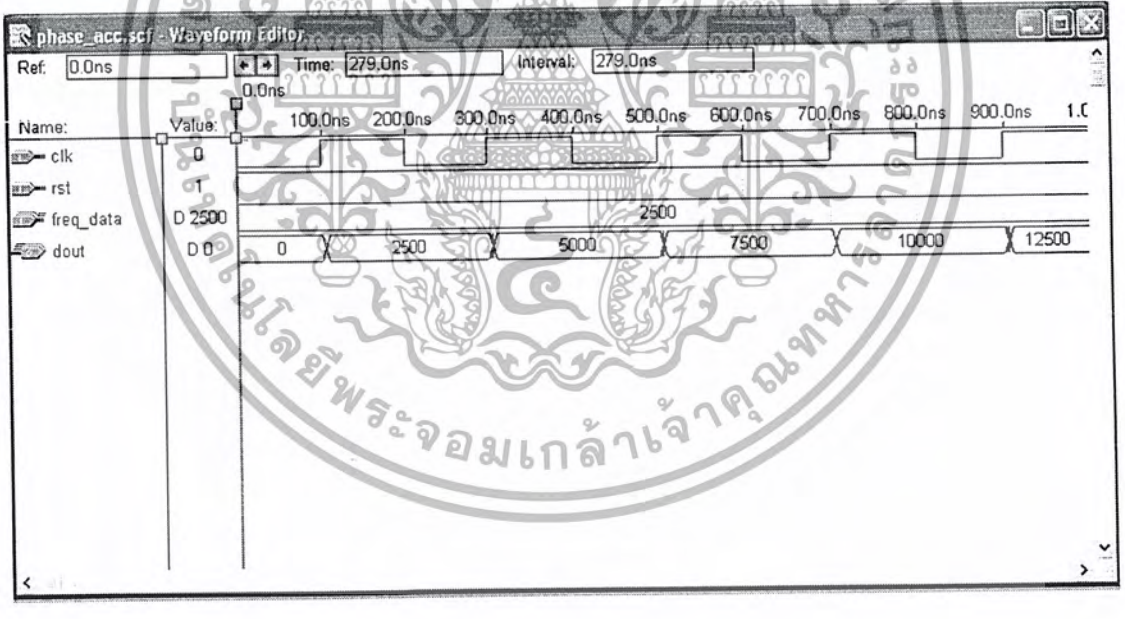
4.3.5 ส่วนของวงจร PHASE_ACCUMULATION

เป็นโมดูลที่ทำหน้าที่รับค่าของ Frequency Control Word จากส่วนของ LATCH และจะทำการบวกสะสมด้วยค่าของ Frequency Control Word ที่รับเข้ามา และส่งค่าเพื่อทำการชี้ค่าใน ROM



รูปที่ 4.14 สัญลักษณ์ของส่วนวงจร PHASE_ACCUMULATION

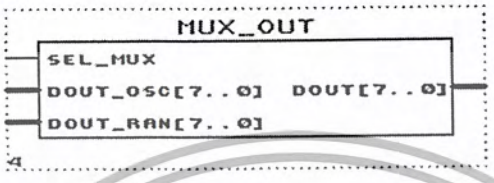
จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้



รูปที่ 4.15 ผลการจำลองการทำงานของส่วนวงจร PHASE_ACCUMULATION

4.3.6 ส่วนของวงจร MUX_OUT

เป็นโมดูลที่ทำหน้าที่ทำการเลือกว่าจะให้ OUTPUT ของ DDS หรือของ LFSR ออกไปยัง OUTPUT ตามต้องการ



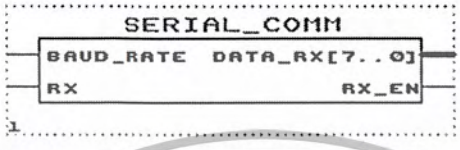
รูปที่ 4.16 สัญลักษณ์ของส่วนวงจร MUX_OUT

จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้

รูปที่ 4.17 ผลการจำลองการทำงานของส่วนวงจร MUX_OUT

4.3.7 ส่วนของวงจร LFSR

โมดูลของ LFSR ทำหน้าที่ผลิตสัญญาณ RANDOM โดยใช้โครงสร้างของ Linear Feedback Shift Register มีลักษณะดังรูป



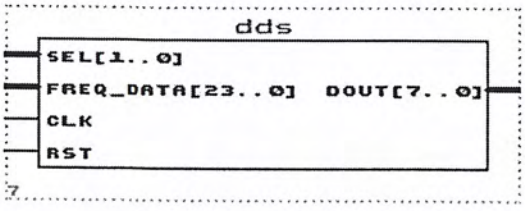
รูปที่ 4.18 สัญลักษณ์ของส่วนวงจร LFSR

จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้

รูปที่ 4.19 ผลการจำลองการทำงานของส่วนวงจร LFSR

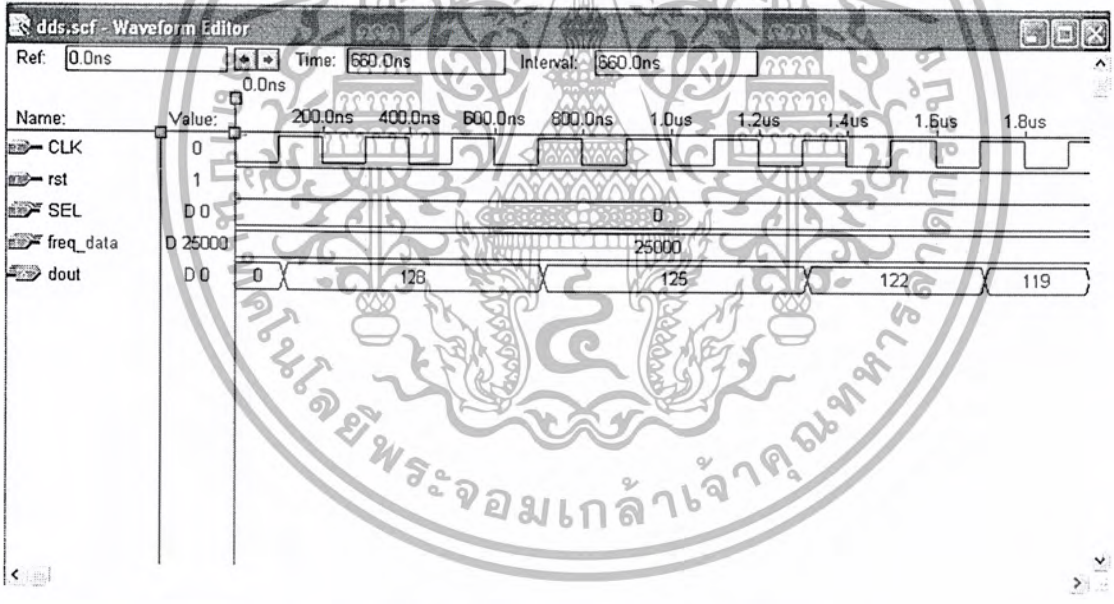
4.3.8 ส่วนของวงจร DDS

เป็นโมดูลที่ทำหน้าที่ผลิตสัญญาณซายน์, สามเหลี่ยม, สี่เหลี่ยม, ฟันเลื่อย



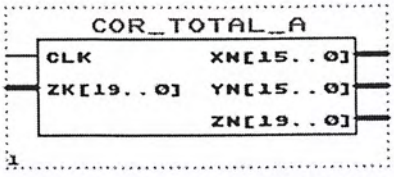
รูปที่ 4.20 สัญลักษณ์ของส่วนวงจร DDS

จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้



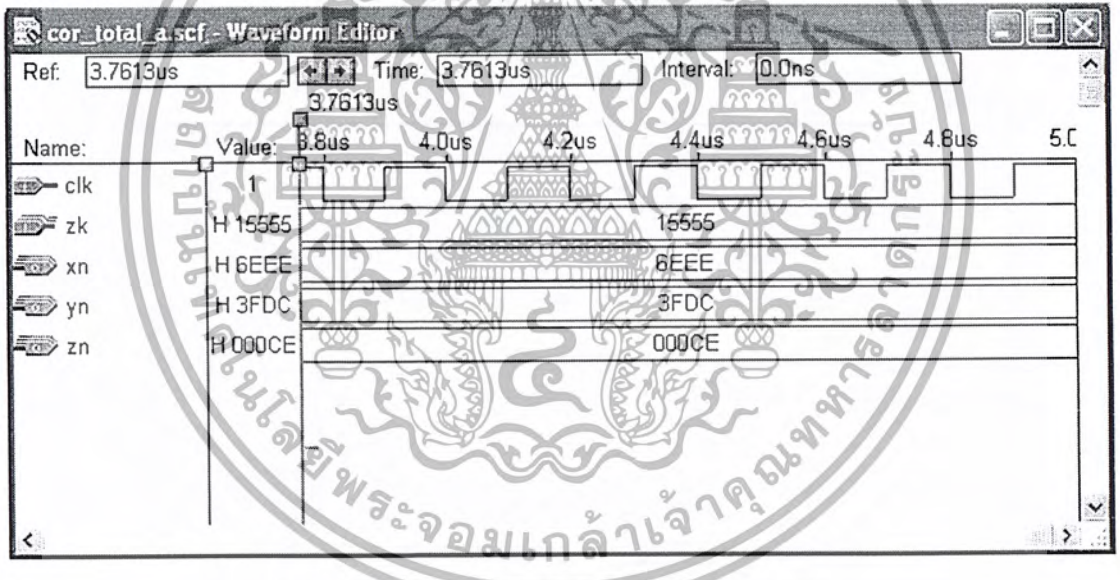
รูปที่ 4.21 ผลการจำลองการทำงานของส่วนวงจร DDS

4.3.9 ส่วนของวงจร CORDIC



รูปที่ 4.22 สัญลักษณ์ของส่วนวงจร CORDIC

จากโปรแกรมที่เขียนขึ้นสามารถจำลองการทำงาน (simulation) ได้ดังนี้

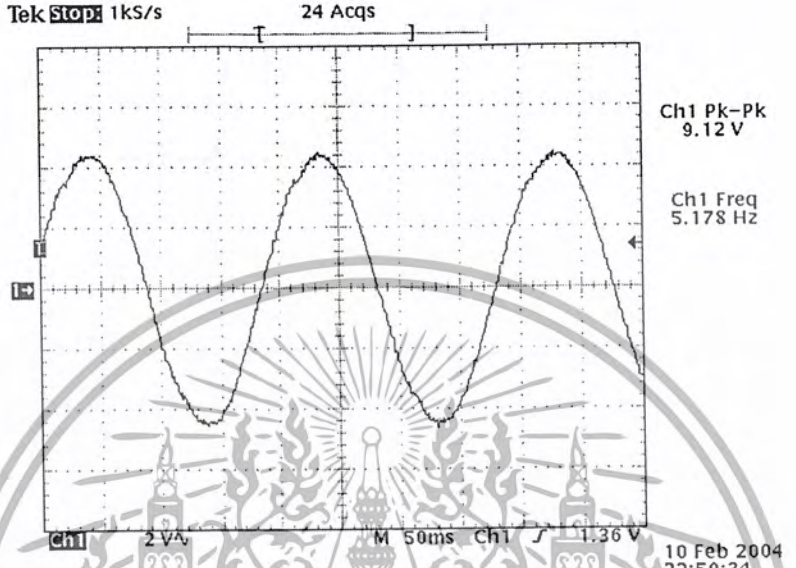


รูปที่ 4.23 ผลการจำลองการทำงานของส่วนวงจรCORDIC

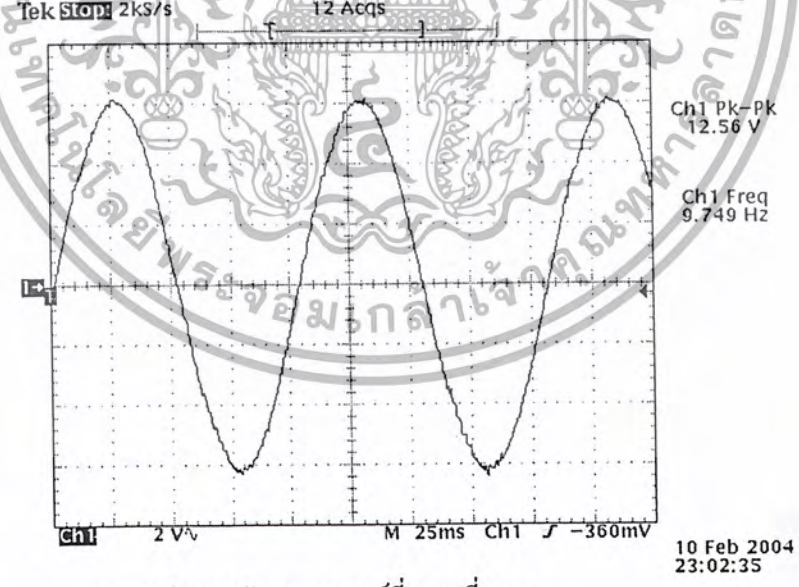
4.4 ผลการทดสอบการทำงานของ DDS OSCILLATOR

การทดสอบการป้อนค่าความถี่ผ่านตัวเครื่องคอมพิวเตอร์โดยใช้ ภาษา วิชวลเบสิก(Visual Basic) ในการเขียนโปรแกรมใช้งานที่ความถี่ต่างๆสามารถทำการทดสอบได้ดังนี้

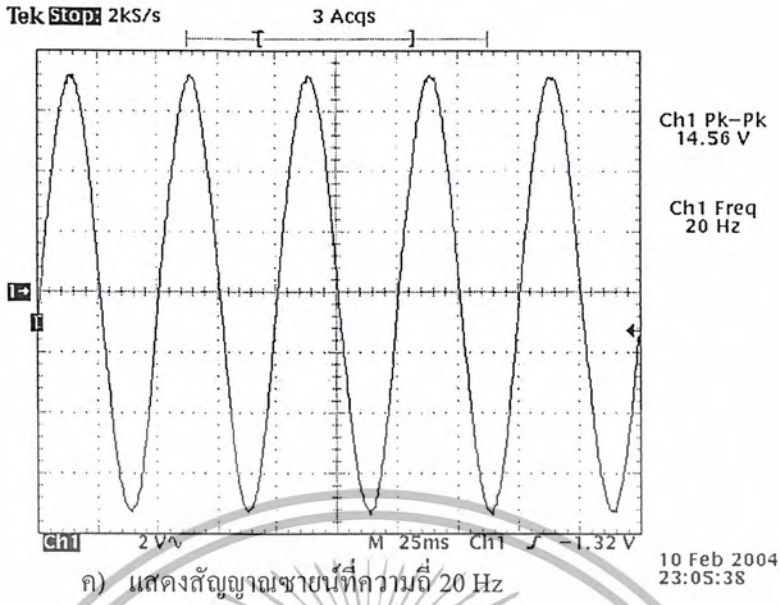
4.4.1 การทดสอบการกำเนิดสัญญาณซายน์ ที่ความถี่ต่างๆดังนี้



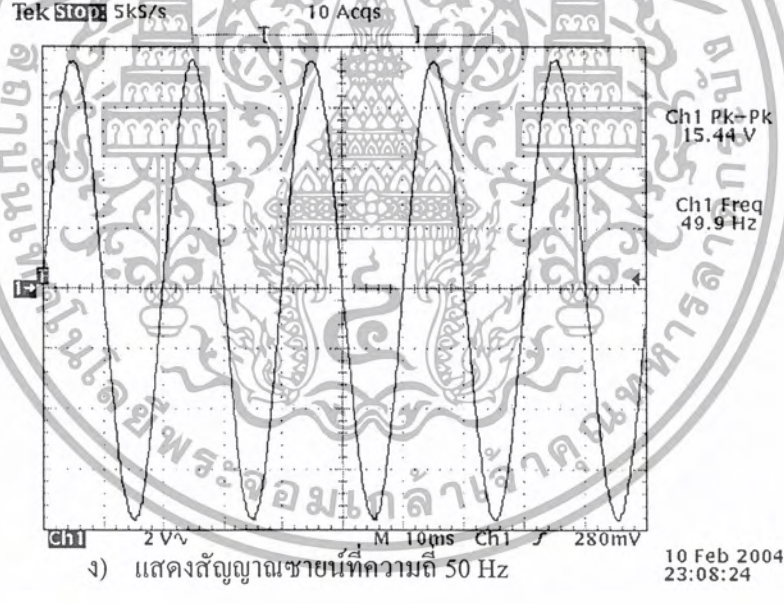
ก) รูปแสดงสัญญาณซายน์ที่ความถี่ 5 Hz



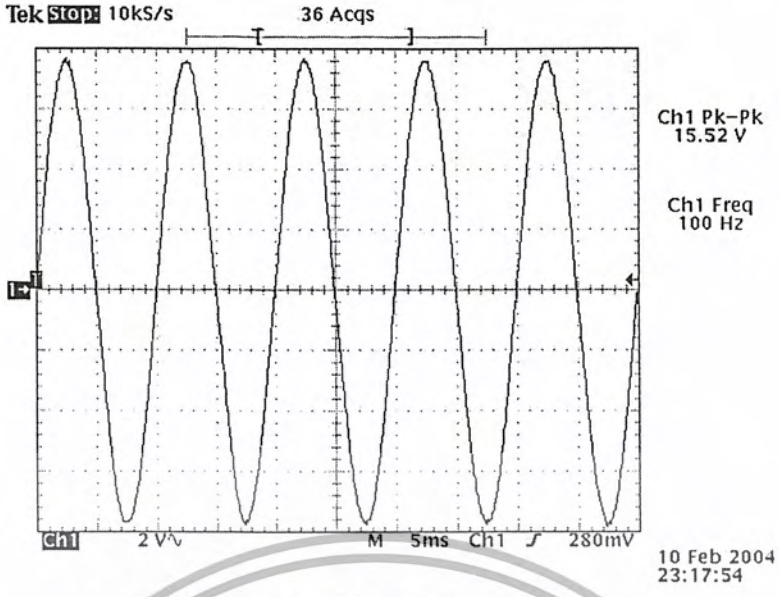
ข) รูปแสดงสัญญาณซายน์ที่ความถี่ 10 Hz



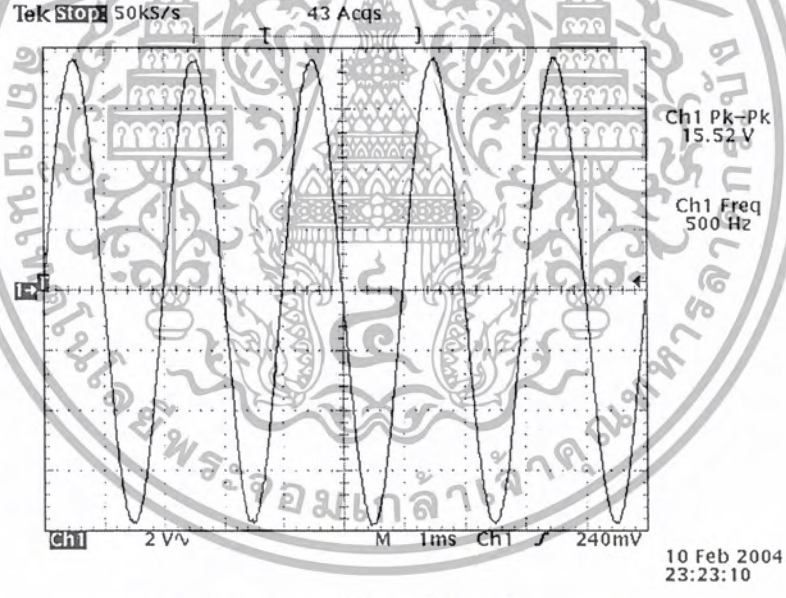
ค) แสดงสัญญาณซายน์ที่ความถี่ 20 Hz



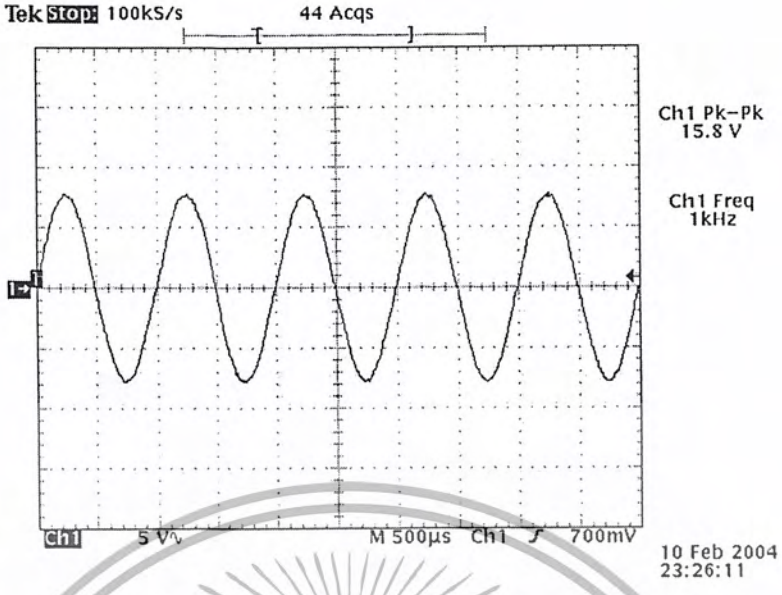
ง) แสดงสัญญาณซายน์ที่ความถี่ 50 Hz



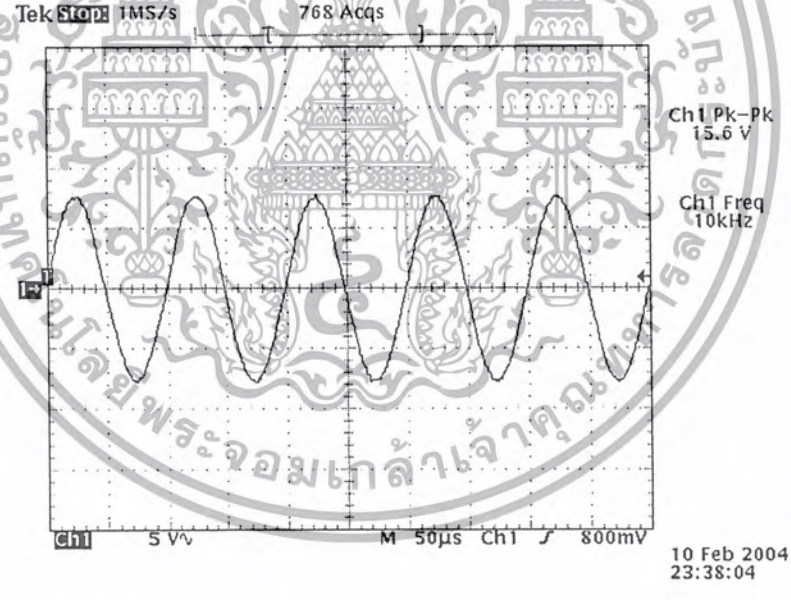
จ) แสดงสัญญาณซายน์ที่ความถี่ 100 Hz



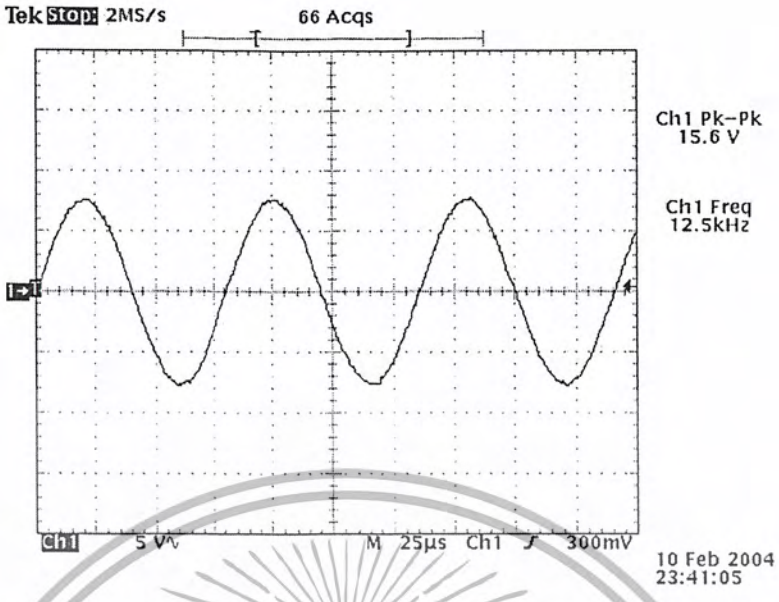
ฉ) แสดงสัญญาณซายน์ที่ความถี่ 500 Hz



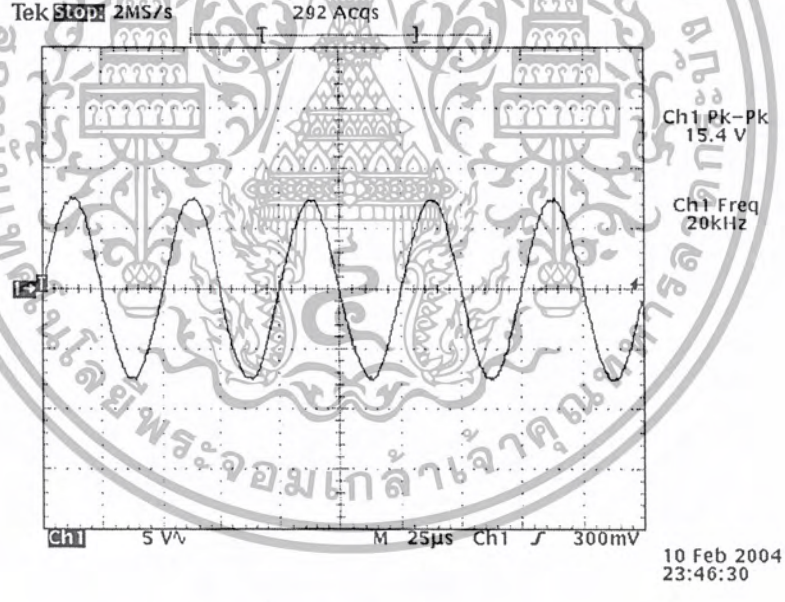
ช) แสดงสัญญาณซายน์ที่ความถี่ 1 kHz



ช) แสดงสัญญาณซายน์ที่ความถี่ 10 kHz



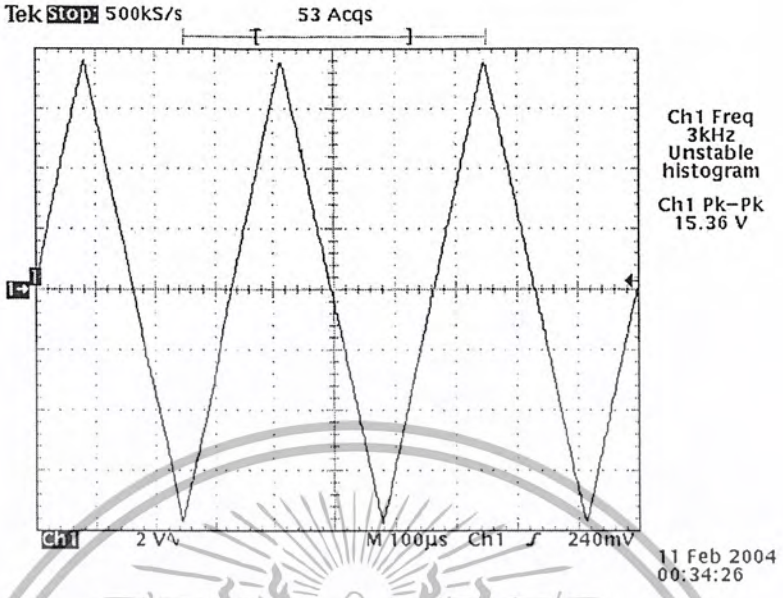
ช) แสดงสัญญาณซายน์ที่ความถี่ 12.5 kHz



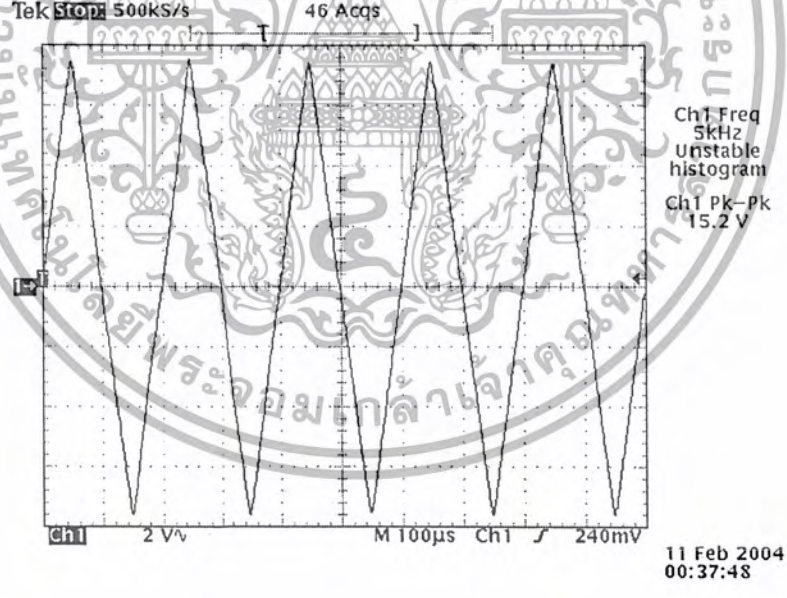
ญ) แสดงสัญญาณซายน์ที่ความถี่ 20 kHz

รูปที่ 4.24 แสดงสัญญาณซายน์ที่ความถี่ต่าง ๆ

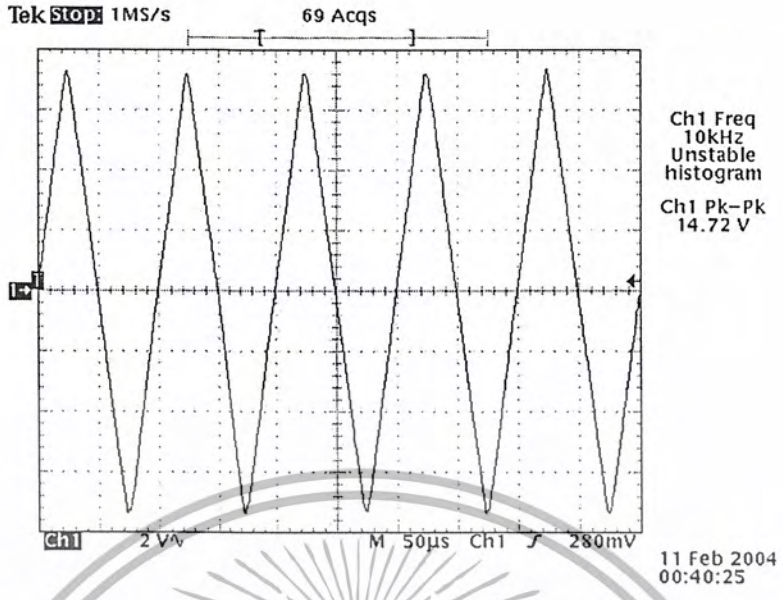
4.4.2 การทดสอบการกำเนิดสัญญาณสามเหลี่ยม ที่ความถี่ต่างๆดังนี้



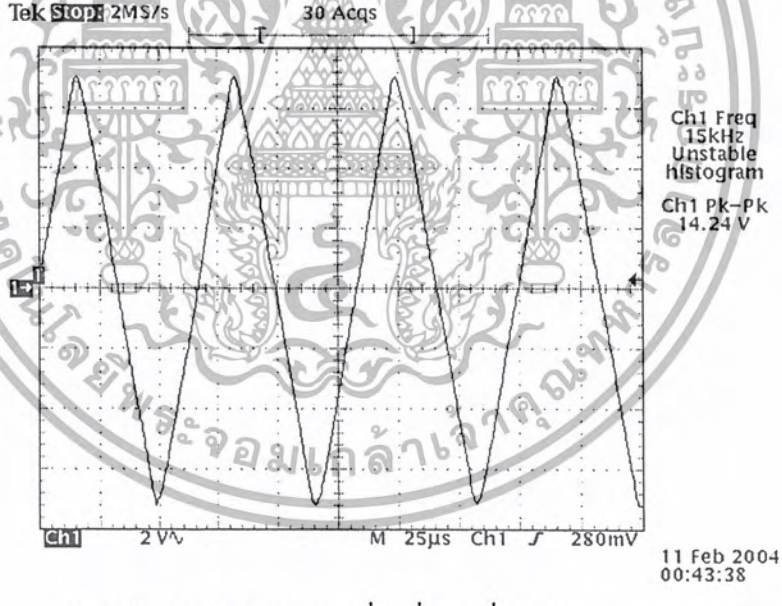
ก) รูปแสดงสัญญาณสามเหลี่ยมที่ความถี่ 3 kHz



ข) รูปแสดงสัญญาณสามเหลี่ยมที่ความถี่ 5 kHz



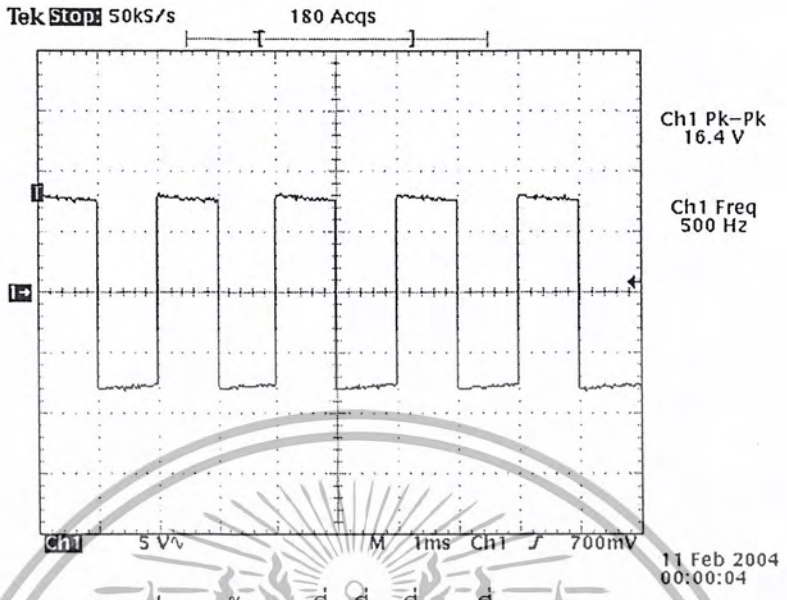
ค) รูปแสดงสัญญาณสามเหลี่ยมที่ความถี่ 10kHz



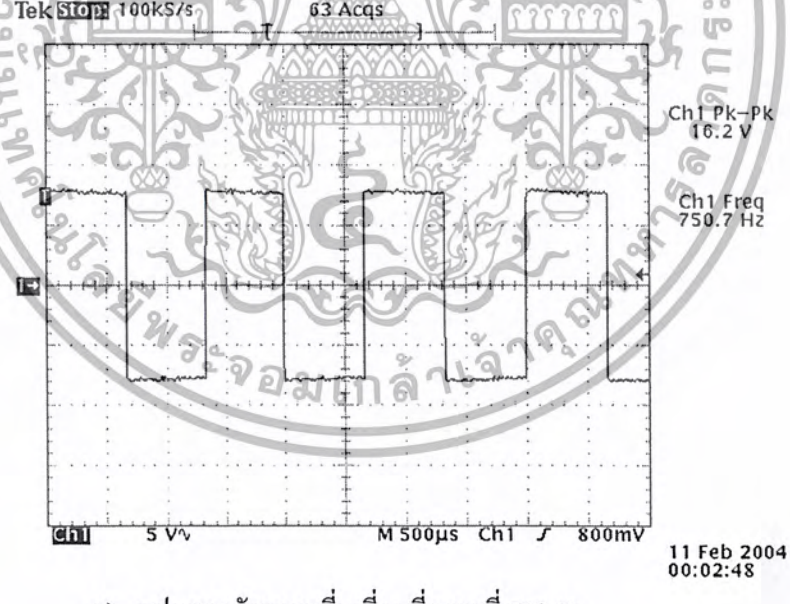
ง) รูปแสดงสัญญาณสามเหลี่ยมที่ความถี่ 15 kHz

รูปที่ 4.25 รูปแสดงสัญญาณสามเหลี่ยมที่ความถี่ต่าง ๆ

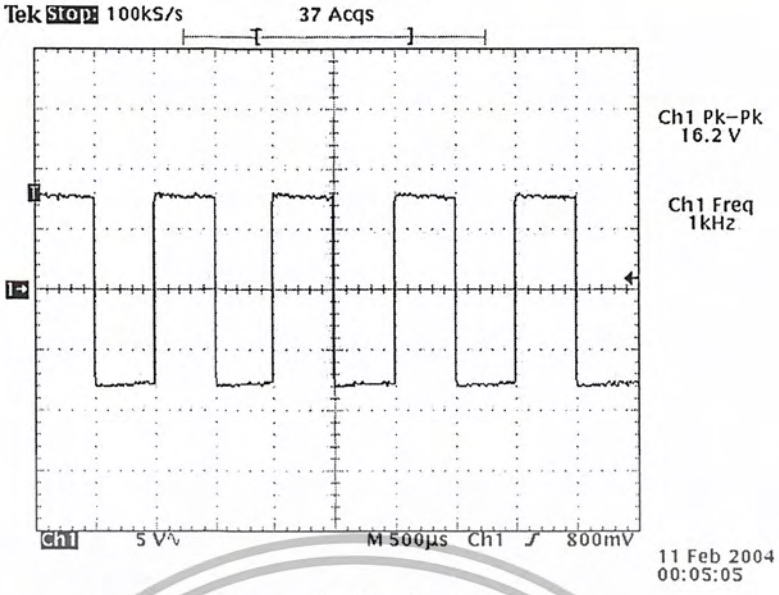
4.4.3 การทดสอบการกำเนิดสัญญาณสี่เหลี่ยม ที่ความถี่ต่างๆดังนี้



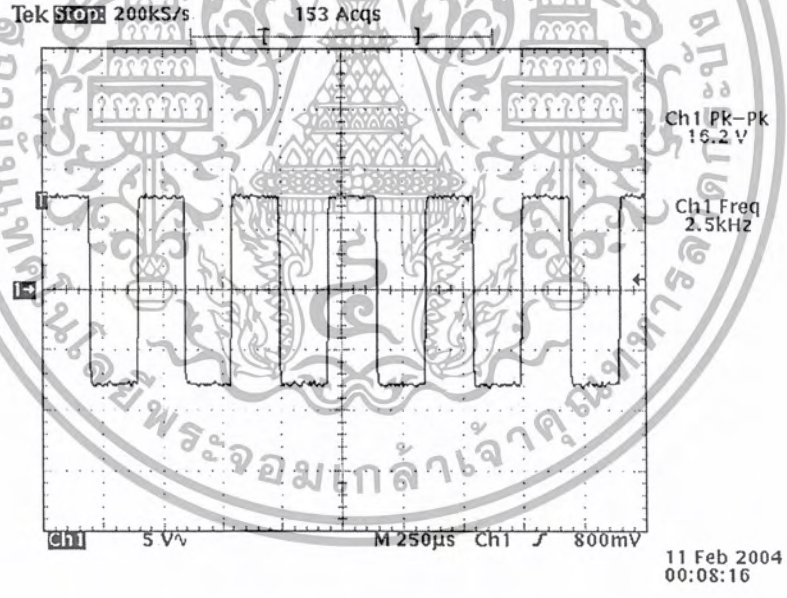
ก) รูปแสดงสัญญาณสี่เหลี่ยมที่ความถี่ 500 Hz



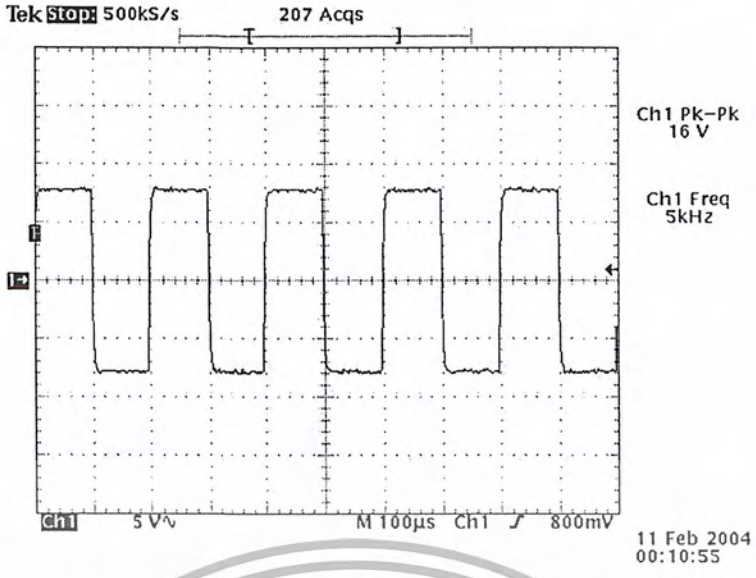
ข) รูปแสดงสัญญาณสี่เหลี่ยมที่ความถี่ 750 Hz



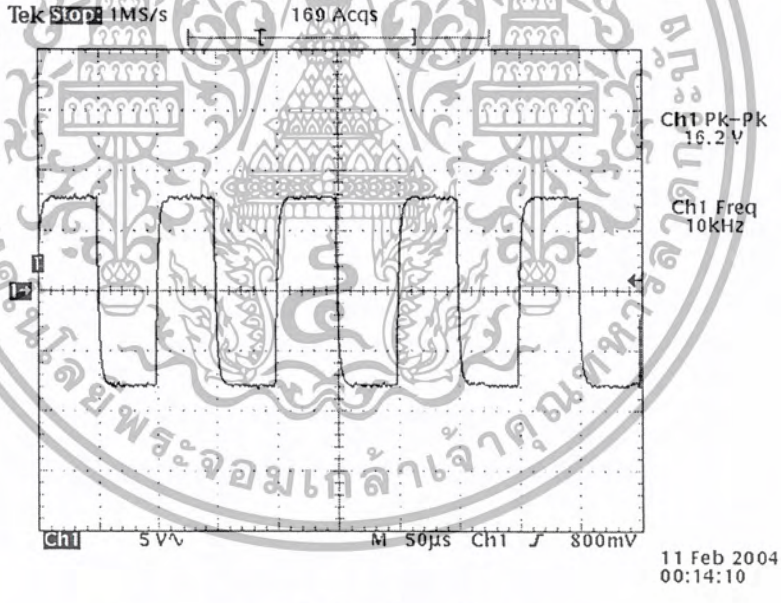
ค) รูปแสดงสัญญาณสี่เหลี่ยมที่ความถี่ 1 kHz



ง) รูปแสดงสัญญาณสี่เหลี่ยมที่ความถี่ 2.5 kHz



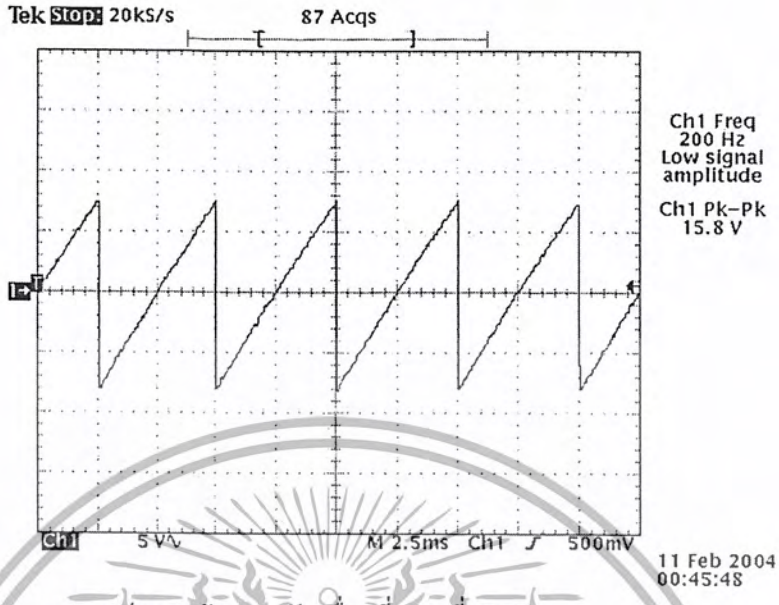
ง) รูปแสดงสัญญาณสี่เหลี่ยมที่ความถี่ 5 kHz



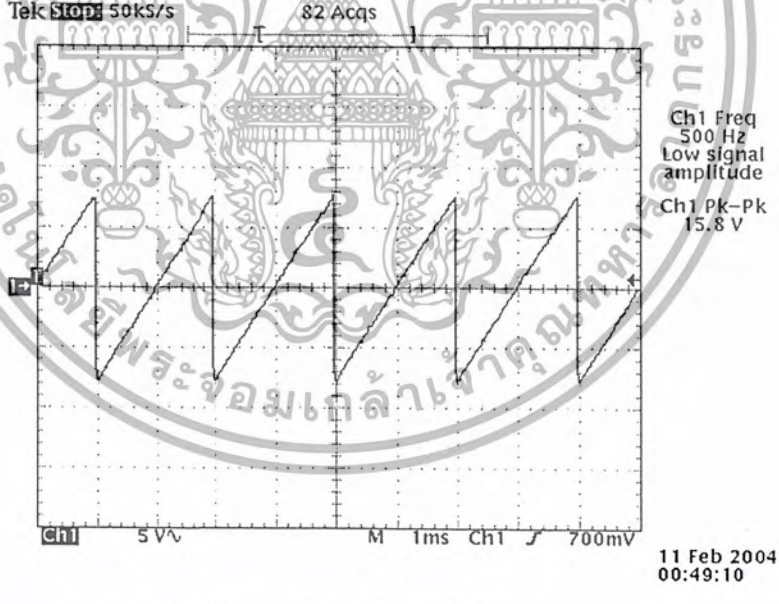
จ) รูปแสดงสัญญาณสี่เหลี่ยมที่ความถี่ 10 kHz

รูปที่ 4.26 แสดงสัญญาณสี่เหลี่ยมที่ความถี่ต่าง ๆ

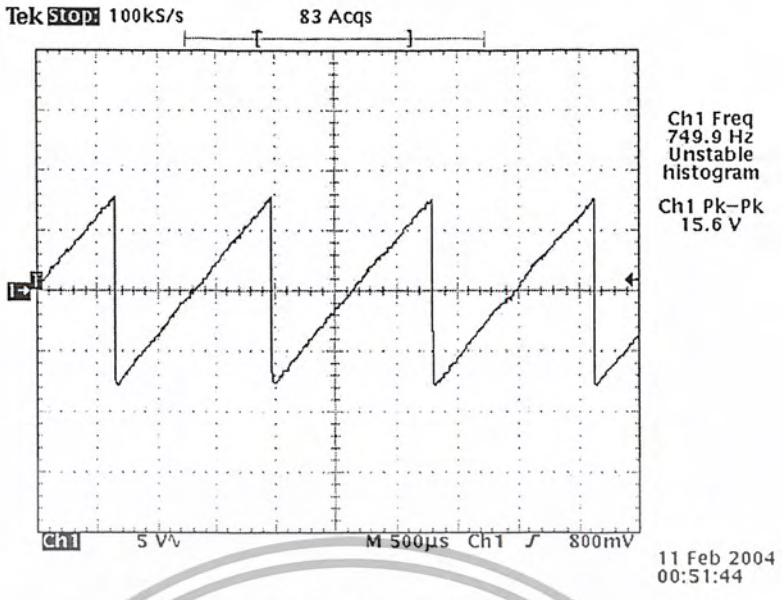
4.4.4 การทดสอบการกำเนิดสัญญาณฟันเลื่อย ที่ความถี่ต่างๆดังนี้



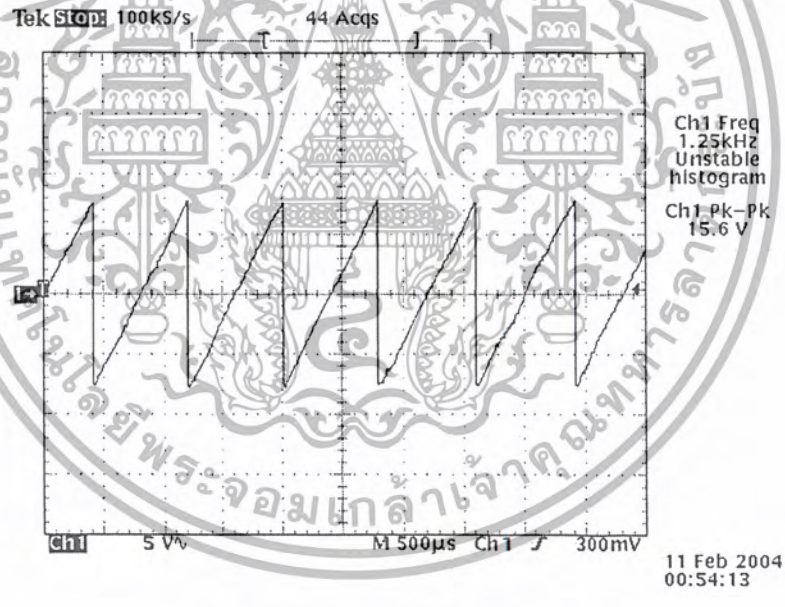
ก) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 200Hz



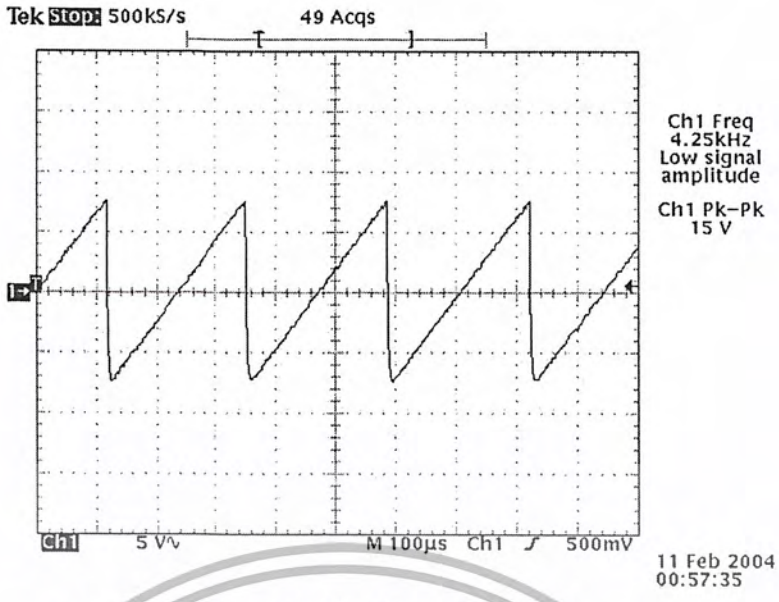
ข) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 500Hz



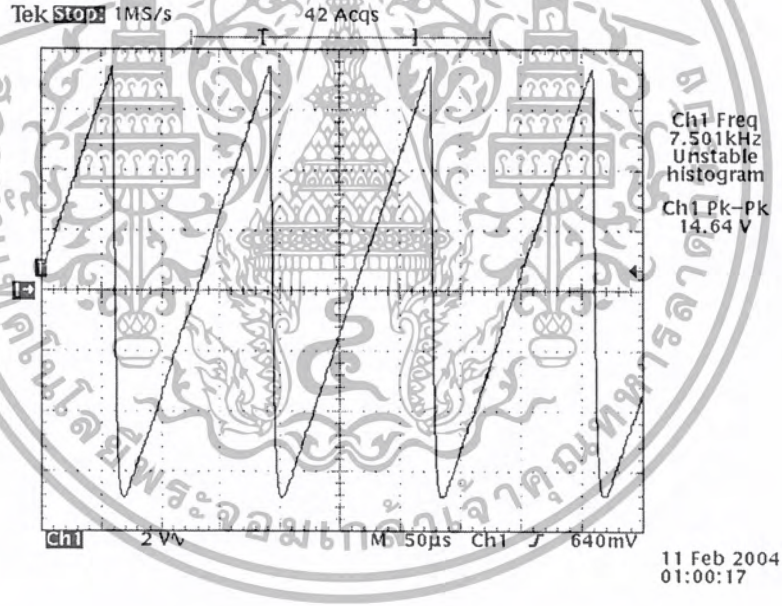
ค) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 750Hz



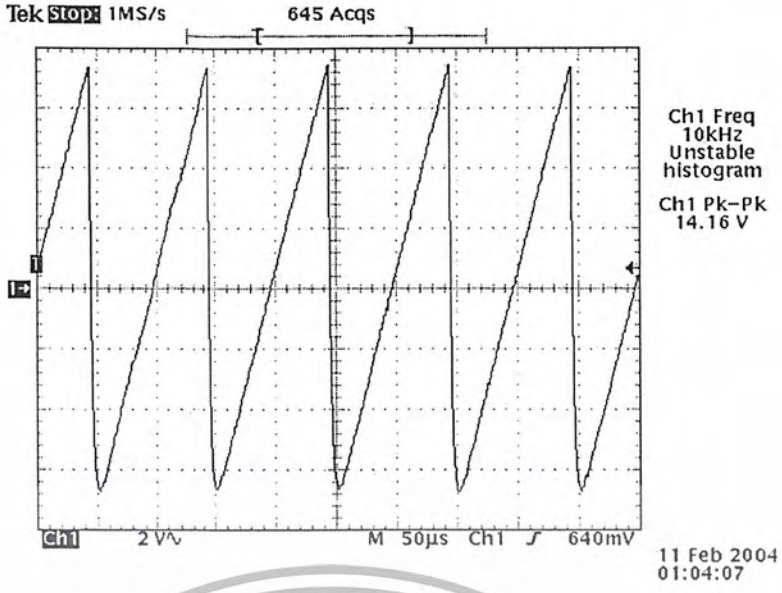
ง) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 1.25 kHz



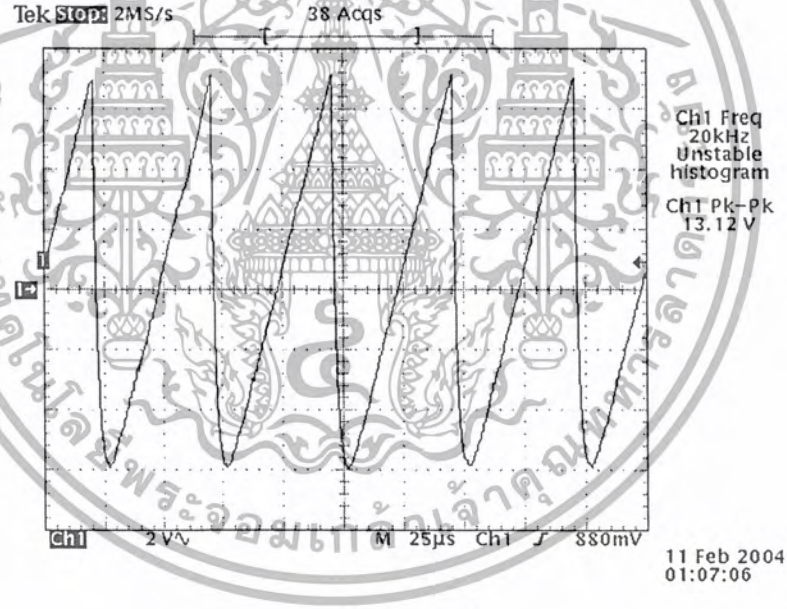
จ) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 4.25 kHz



ฉ) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 7.5kHz



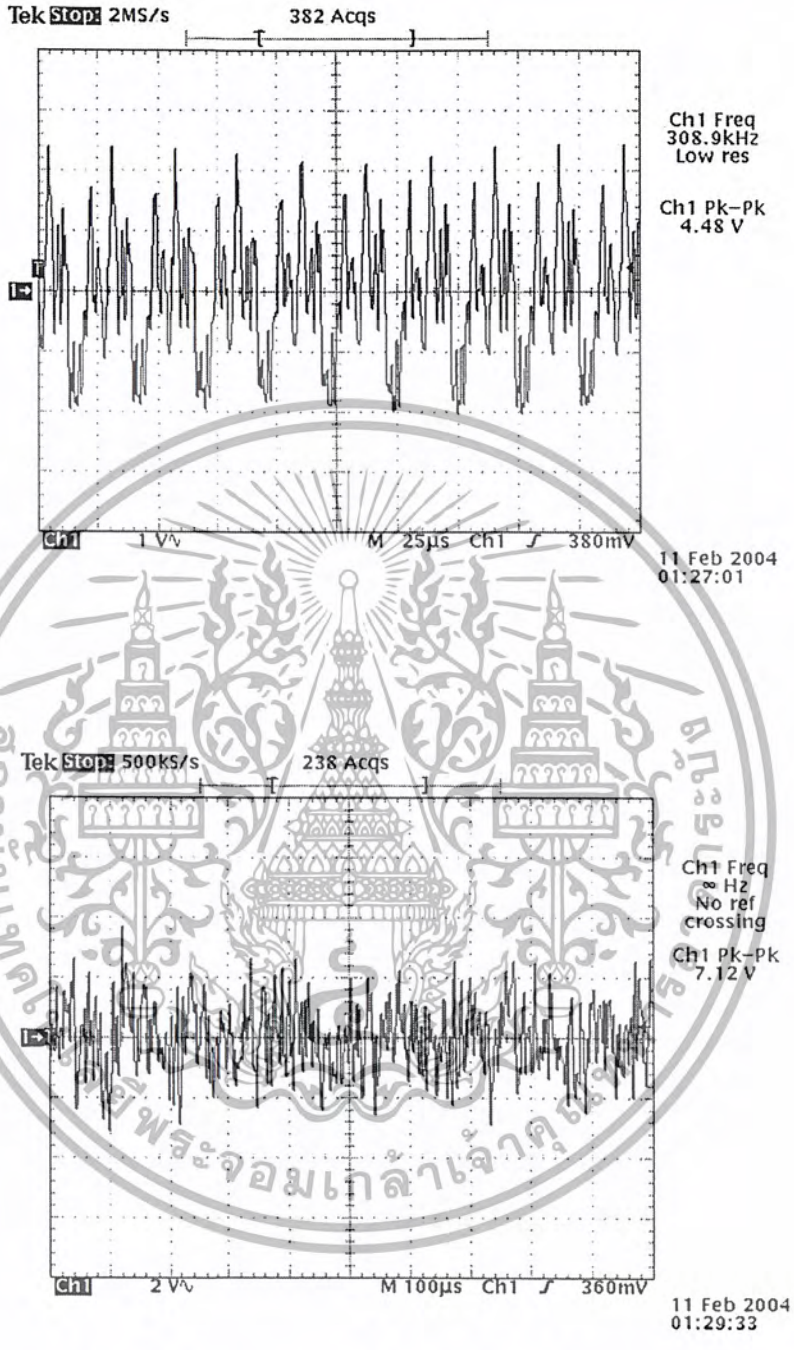
ช) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 10 kHz



ซ) รูปแสดงสัญญาณฟันเลื่อยที่ความถี่ 20 kHz

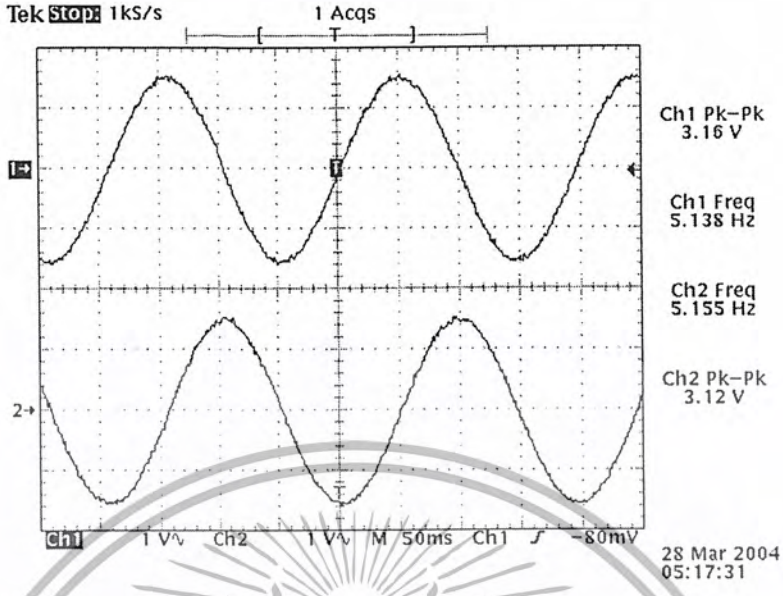
รูปที่ 4.27 แสดงสัญญาณฟันเลื่อยที่ความถี่ต่าง ๆ

4.4.5 การทดสอบการกำเนิดสัญญาณสุ่ม

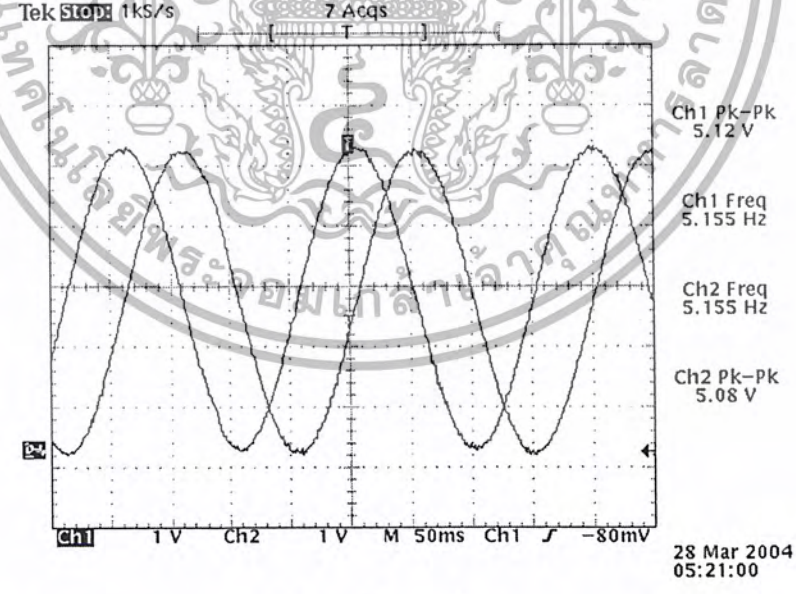


รูปที่ 4.28 แสดงสัญญาณสุ่มที่ order 8 และ 32

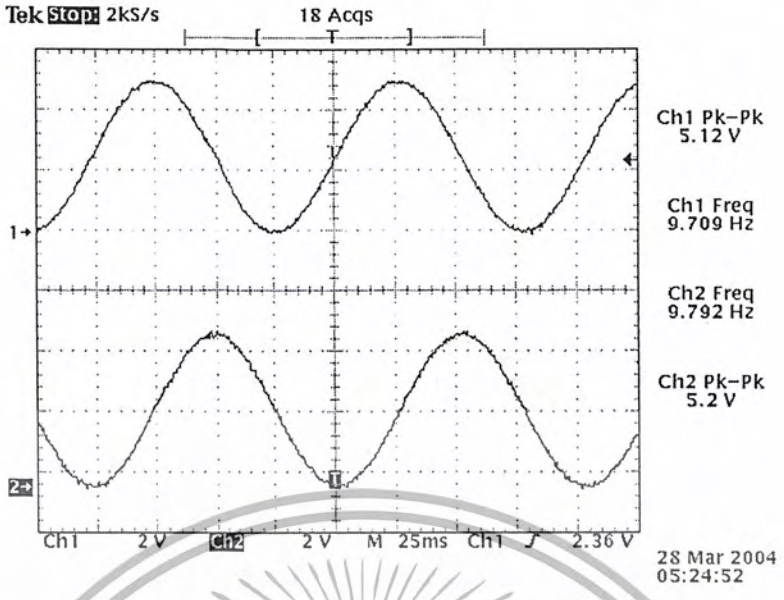
4.5 ผลการทดสอบการทำงานของ CORDIC OSCILLATOR



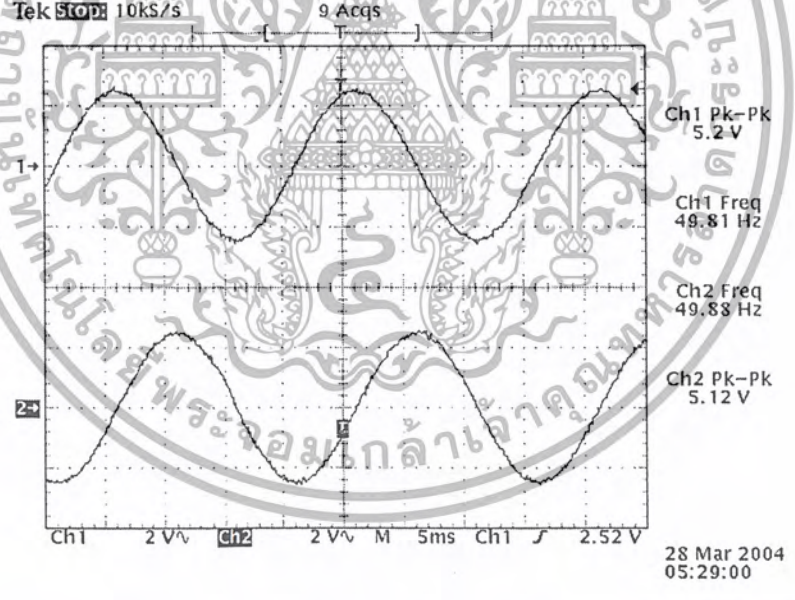
ก) รูปแสดงสัญญาณซายน์ที่ความถี่ 5 Hz



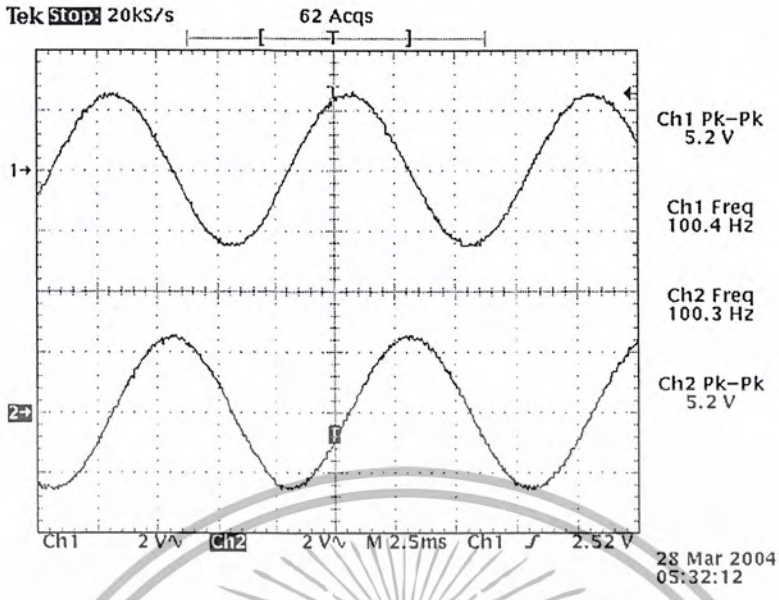
ข) รูปแสดงสัญญาณซายน์ที่ความถี่ 5 Hz เพื่อแสดงเฟสของสัญญาณ



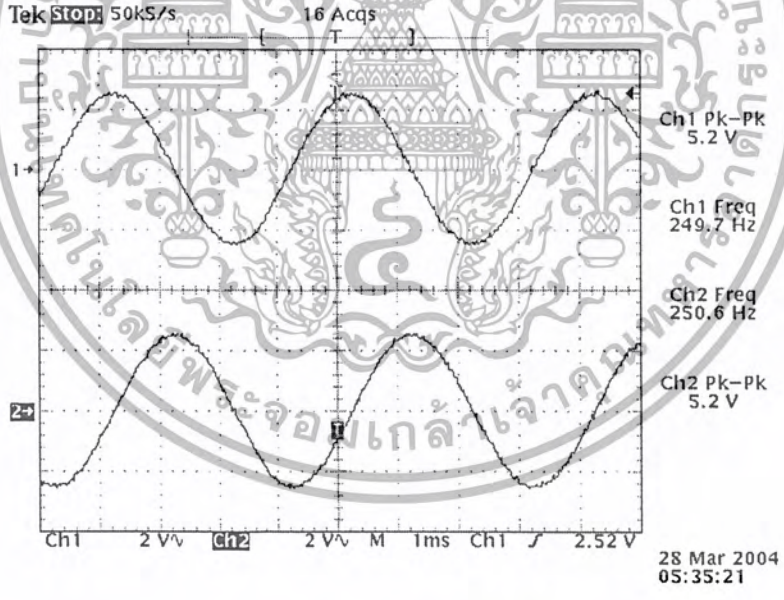
ค) รูปแสดงสัญญาณซายน์ที่ความถี่ 10 Hz



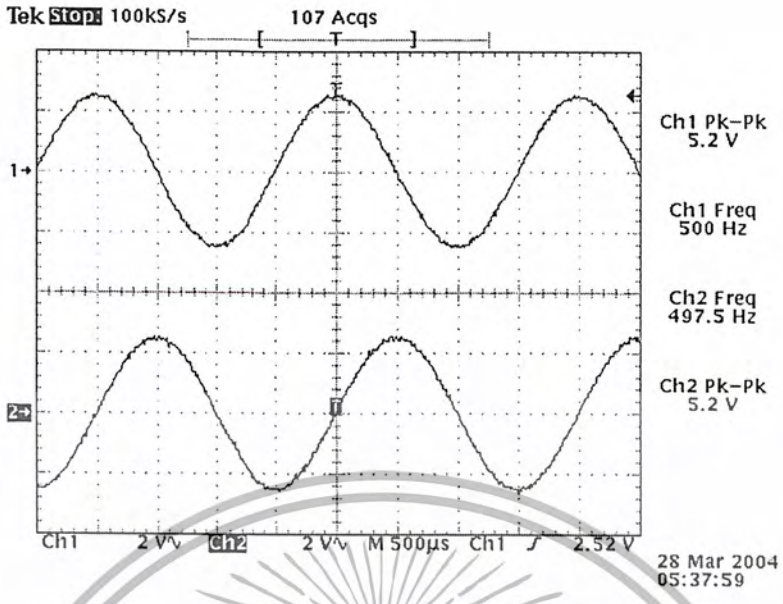
ง) รูปแสดงสัญญาณซายน์ที่ความถี่ 50 Hz



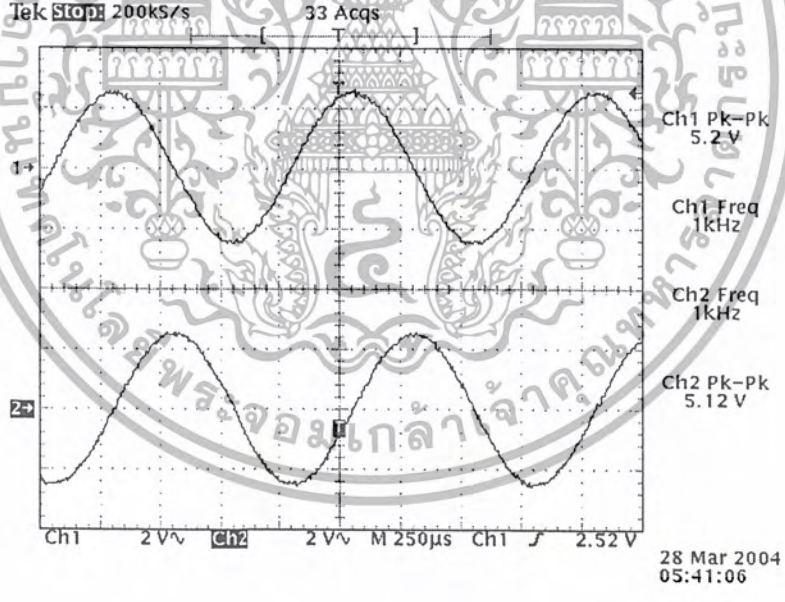
จ) รูปแสดงสัญญาณซายน์ที่ความถี่ 100 Hz



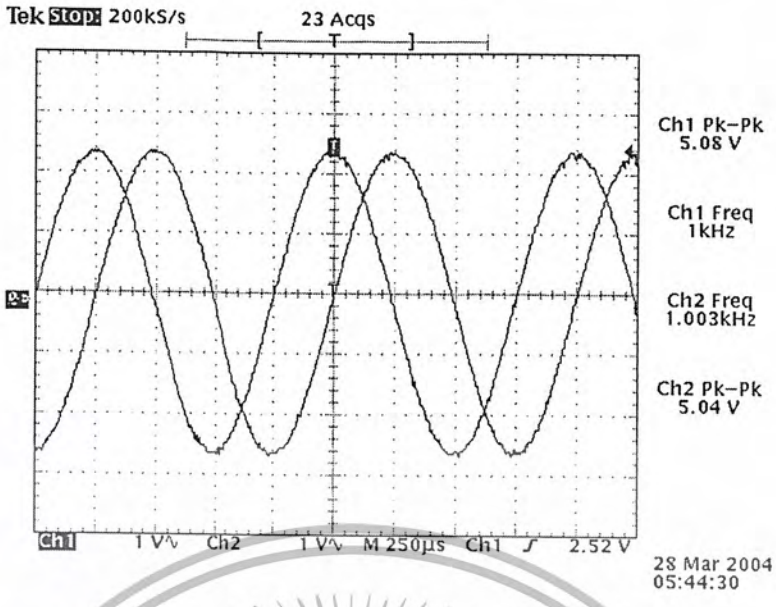
ฉ) รูปแสดงสัญญาณซายน์ที่ความถี่ 250 Hz



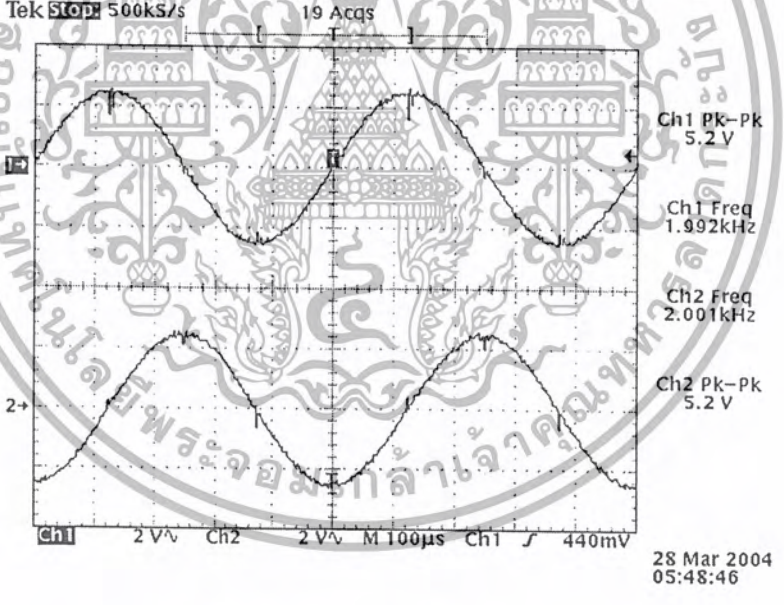
ช) รูปแสดงสัญญาณซายน์ที่ความถี่ 500 Hz



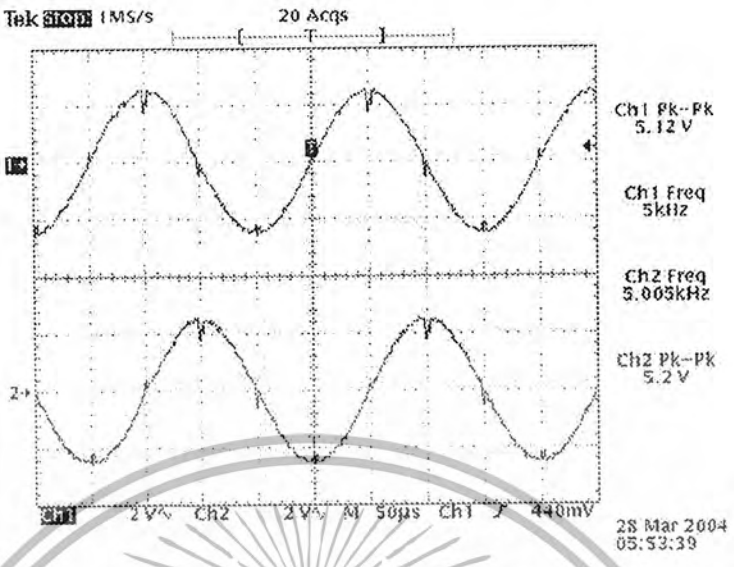
ซ) รูปแสดงสัญญาณซายน์ที่ความถี่ 1 kHz



ฌ) รูปแสดงสัญญาณซายน์ที่ความถี่ 1 kHz เพื่อแสดงเฟสของสัญญาณ



ญ) รูปแสดงสัญญาณซายน์ที่ความถี่ 2 kHz



ฎ) รูปแสดงสัญญาณซายน์ที่ความถี่ 5 kHz

รูปที่ 4.29 แสดงสัญญาณซายน์ที่สร้างจาก CORDIC

จากการทดลองในการกำเนิดสัญญาณโดยวิธี Direct Digital Frequency Synthesizer :DDS จะได้สัญญาณรูปไซน์, สี่เหลี่ยม, สามเหลี่ยม, สามเหลี่ยม, ฟันเลื่อย และ สัญญาณสุ่ม ความถี่ที่กำเนิดได้จะมีความเที่ยงตรงสูงและมีค่าความถี่ที่กำเนิดได้ประมาณ 20 kHz ที่สัญญาณไม่เกิดความผิดเพี้ยนซึ่งหากต้องการความถี่ที่สูงกว่านี้สามารถทำได้โดยการเพิ่มความถี่ของสัญญาณนาฬิกาที่ป้อนให้กับระบบ และเพิ่มวงจรDigital to Analog จาก 8 บิทเป็น 16 บิท ส่วนการทดลองการกำเนิดสัญญาณโดยวิธีของ CORDIC Algorithm จะได้สัญญาณรูปไซน์และโคไซน์พร้อมกัน และที่ความถี่ 5 kHz สัญญาณที่ได้จะมีความผิดเพี้ยนสามารถแก้ไขได้โดยใช้วงจรกรองความถี่ต่ำผ่าน หากต้องการความถี่และความละเอียดสูงขึ้นก็สามารถแก้ไขได้เช่นเดียวกันกับวิธีของ DDS คือการเพิ่มสัญญาณนาฬิกา และใช้ Digital to Analog ขนาด 16 บิท

บทที่5
บทสรุปและวิจารณ์

ทฤษฎีของ CORDIC Algorithm จะอาศัยการเปลี่ยนแปลงตำแหน่งของเวคเตอร์ ผลจากการเปลี่ยนตำแหน่งหรือการหมุนของเวคเตอร์จะทำให้เกิดมุมขึ้น และเมื่อทำการจัดความสัมพันธ์ของมุมกับตำแหน่งของเวคเตอร์จะทำให้ได้สมการพื้นฐานของ CORDIC Algorithm และสามารถจัดรูปสมการของ CORDIC เพื่อประยุกต์หาค่าที่ต้องการได้ ซึ่งความละเอียดของค่าที่หาได้จะขึ้นอยู่กับจำนวนของการวนรอบการทำงาน

ในการออกแบบวงจรกำเนิดความถี่โดยใช้โครงสร้างของ CORDIC นั้น จะทำการคำนวณค่า sine และ cosine ของมุมที่รับเข้ามาได้เพียง 1 จตุภาค(quadrant)เท่านั้น ดังนั้นในการใช้งานจริงเพื่อให้ได้สัญญาณครบทุกมุม (360 องศา) จะต้องมีการจัดความสัมพันธ์ของอินพุตและเอาท์พุต

ส่วนการออกแบบวงจรกำเนิดสัญญาณแบบตรง (Direct Digital Frequency Synthesizer: DDS) นั้นจะอาศัยการเก็บค่าขนาดของสัญญาณไว้ในตารางเปิดดู (Lookup Table) แล้วใช้การชี้ตำแหน่งในการกำเนิดสัญญาณออกมา ซึ่งในการออกแบบค่าที่เก็บในตารางเปิดดูนั้นจะเก็บค่าขนาดของสัญญาณเพียง 1 ใน 4 ของรูปสัญญาณ 1 คาบเท่านั้น และจะอาศัยการจัดความสัมพันธ์ของอินพุตเพื่อให้ได้เอาท์พุตเป็นสัญญาณครบ 1 คาบเช่นเดียวกันกับการกำเนิดสัญญาณโดยใช้ CORDIC

ในการสร้างสัญญาณแบบดิจิตอลโดยใช้ CORDIC จะได้ค่าสัญญาณไซน์และโคไซน์ในเวลาเดียวกันส่วนการออกแบบโดยวิธีของ DDS จะให้รูปของสัญญาณที่หลากหลายกว่า (ขึ้นอยู่กับค่าขนาดของสัญญาณที่เก็บไว้ในตารางเปิดดู) ซึ่งทั้งหมดนี้เมื่อออกแบบและสร้างวงจรด้วยภาษาวีเอชดีแอล (VHDL) แล้วจำลองการทำงานเพื่อดูผลที่ได้ซึ่งมีความใกล้เคียงกับทางทฤษฎี ทั้งอาจเกิดความผิดพลาดอันเนื่องมาจากการสร้างจริงนั้นจะแทนค่าจำนวนจริงให้อยู่ในรูปไบนารี่บิต เมื่อใช้จำนวนบิตในการแทนน้อยจะทำให้ค่าที่ได้มีความคลาดเคลื่อนซึ่งอาจแก้ปัญหาโดยการใช้บิตในการแทนค่าจำนวนจริงให้มากขึ้น แต่ก็จะส่งผลโดยตรงต่อการสร้างฮาร์ดแวร์ คืออาจจะใช้ลอจิกเซลล์ในการสร้างวงจรมากขึ้นเป็นต้น

สำหรับการประยุกต์ใช้งานวงจรกำเนิดสัญญาณแบดิจิตอลโดยใช้โครงสร้างของ CORDIC นั้นเหมาะกับวงจรหรือระบบที่ต้องการสัญญาณที่มีเฟสต่างกัน 90 องศา (ไซน์และโคไซน์) พร้อมกัน ตัวอย่างเช่น วงจร Quadrature Modulation และ Phase Shift Keying (PSK) เป็นต้น ส่วนวงจรกำเนิดสัญญาณดิจิตอลแบบ DDS เหมาะกับระบบที่ใช้งานโดยทั่ว ๆ ไป ที่ต้องการสัญญาณพื้นฐาน เช่น สัญญาณไซน์ , สัญญาณรูปสามเหลี่ยม , สัญญาณรูปสี่เหลี่ยม หรือ สัญญาณรูปฟันเลื่อย เป็นต้น หรืออาจจะออกแบบให้ได้สัญญาณเฉพาะ

จากการศึกษาโครงงาน ทำให้มีความรู้เกี่ยวกับการนำเอาแนวความคิด ทฤษฎีต่าง ๆ มาใช้งานร่วมกัน รวมทั้งวงจรที่ออกแบบสามารถนำมาสร้างเป็นฮาร์ดแวร์ด้วยภาษาวีเอชดีแอล ที่ความยืดหยุ่นในการออกแบบสูง และวงจรที่ออกแบบสามารถใช้ร่วมกับคอมพิวเตอร์โดยใช้ภาษาวิชวลเบสิค ในการช่วยในการเชื่อมต่อ (Interface) ซึ่งสิ่งเหล่านี้เหมาะแก่การศึกษาค้นคว้าและพัฒนาเป็นอย่างยิ่ง

โปรแกรมภาษาวีเอชดีแอล

โปรแกรม Div 1000

```
library ieee;
use ieee.std_logic_1164.all;
entity Div1000 is
port(Clk_in:in std_logic;
      Clk_out:out std_logic
     );
end Div1000;
architecture rtl of Div1000 is
begin
     process(Clk_in)
     variable Clk_temp:std_logic:='0';
     variable count:integer range 0 to 499;
     begin
       if Clk_in'Event and Clk_in='1'then
          if count<499 then
           count:=count+1;
           Clk_temp:=Clk_temp;
          Else
           count:=0;
           Clk_temp:=not(Clk_temp);
          end if;
          Clk_out<=Clk_temp;
       end if;
     end process;
end rtl;
```

โปรแกรมของ Lacth

```
library ieee;
use ieee.std_logic_1164.all;
entity latch40 is
port(en:in std_logic;
     ip:in std_logic_vector(7 downto 0);
     sel_mux:out std_logic;
     sel:out std_logic_vector(1 downto 0);
     out_fre:out std_logic_vector(23 downto 0)
     );
end latch40;

architecture rtl of latch40 is
     signal buf1,buf2,buf3: std_logic_vector(7 downto 0);

begin
     process(en,ip)
     VARIABLE count_rx : integer;

     begin
       if en'event and en='1' then
          if count_rx < 3 then
               count_rx:=count_rx+1;
               buf1<=ip;
               buf2<=buf1;
               buf3<=buf2;
          else
               count_rx:=0;
               out_fre<=buf1&buf2&buf3;
               sel_mux<=ip(2);
               sel(1 downto 0)<=ip(1 downto 0);
          end if;
       end if;
     end process;
end rtl;
```

โปรแกรม Serial_Communication

```
library ieee;
use ieee.std_logic_1164.all;
entity SERIAL_COMM is
port(
     BAUD_RATE:in std_logic;
     RX:in std_logic;
     DATA_RX:out std_logic_vector(7 downto 0);
     RX_EN:out std_logic
          );
end SERIAL_COMM;
architecture rtl of SERIAL_COMM is
type State_type_RX is(idel,ReceiveData,Stop);
signal State_RX:State_type_RX:=Idel;
begin
process(BAUD_RATE,RX)
variable RX_Data_Count:integer range 0 to 7;
variable Buffer_RX:std_logic_vector(7 downto 0);
begin
     if BAUD_RATE'Event and BAUD_RATE='1' then
       case State_RX is
          when Idel=>
           RX_EN<='0';
           if RX='0' then
              RX_Data_Count:=0;
              State_RX<=ReceiveData;
           Else
              RX_Data_Count:=0;
              State_RX<=Idel;
           end if;
          when ReceiveData=>
            RX_EN<='0';
            if RX_Data_Count=7 then
               Buffer_RX(RX_Data_Count):=RX;
               State_RX<=Stop;
            Else
               Buffer_RX(RX_Data_Count):=RX;
               RX_Data_Count:=RX_Data_Count+1;
               State_RX<=ReceiveData;
            end if;
          when Stop=>
            RX_Data_Count:=0;
            data_rx<=buffer_rx;
            RX_En<='1';
            State_RX<=Idel;
          when others =>
            RX_Data_Count:=0;
            RX_En<='0';
            State_RX<=Idel;
```

```
            end case;
          end if;
        end process ;
end rtl;
```

โปรแกรม Onepulse

```
LIBRARY IEEE;
use ieee.std_logic_1164.all;
entity Onepulse is
port(Clk:in std_logic;
     SW:in std_logic;
     SW_Single_Pulse:out std_logic
     );
end Onepulse;
architecture rtl of Onepulse is
signal sw_debounce_delay:std_logic;
begin
     process(Clk)
     begin
       if(Clk'Event)and(Clk='1')then
            if(sw='1')and(sw_debounce_delay='0')then
               sw_single_pulse<='1';
            else
              sw_single_pulse<='0';
            end if;
            sw_debounce_delay<=sw;
          end if;
       --end if;
     end process;
end rtl;
```

โปรแกรม Phase accumulator

```
library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
------------------------------------------------------------
-
entity phase_acc24 is
port ( clk : in std_logic;
        rst : in std_logic;
       sel : in std_logic_vector(1 downto 0);
        freq_data: in std_logic_vector(23 downto 0);
        dout: out std_logic_vector(7 downto 0)
    );
end phase_acc24;
------------------------------------------------------------
--
architecture rtl of phase_acc24 is
------------------------------------------------------------
--
--signal address : unsigned( 5 downto 0);
  signal dout1   : std_logic_vector (7 downto 0);
--signal result  : std_logic_vector (7 downto 0);
--signal result1 : std_logic_vector (7 downto 0);
  signal accum   : std_logic_vector (23 downto 0);
--signal sign    : std_logic;

begin
------------------------------------------------------------
acc:process(clk,rst,freq_data)
variable count1:std_logic_vector(23 downto 0);
  begin
 count1:= freq_data ;
 if rst = '0' then
    accum<="000000000000000000000000";
     elsif clk'event and clk='1'then
       accum <= count1 + accum;
   end if;

   end process;
----------------------------------------
out_squre:process( accum)
begin
if  accum(23) ='0' then
     dout1 <=  "11111111";
else
     dout1 <= (others => '0');
end if;
```

```
end process;
-----------------------------------------

dout<= dout1 when sel ="01" else
        accum(23 downto 16);

end;
```

โปรแกรม LFSR

```
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------
----
entity lfsr_32 is
port(   Clock     : in std_logic;
          lfsr_out : out std_logic_vector(7 downto 0)
     );
end;
-------------------------------------------------------------
-----
architecture rtl of lfsr_32 is
signal reg : std_logic_vector( 31 downto 0);
signal feedback:std_logic;
begin
     process(Clock)

     begin
     if( Clock'event and Clock = '1' ) then
          if reg="00000000000000000000000000000000" then
               reg<="00000000000000000000000000000001";
          else
          reg(31 downto 8)<=reg(30 downto 7);
          reg(7)<=reg(6) xor feedback;
          reg(6)<=reg(5) xor feedback;
          reg(5)<=reg(4);
          reg(4)<=reg(3);
          reg(3)<=reg(2);
          reg(2)<=reg(1) xor feedback;
          reg(1)<=reg(0);
          reg(0)<=feedback;
          end if;
     end if;
feedback<=reg(31);
 end process;
          lfsr_out<=reg(31 downto 24);

end;
-------------------------------------------------------------
```

โปรแกรม Mux_out

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
-------------------------------------------------------------
-------------
ENTITY mux_out IS
  PORT(
       sel_mux       : IN STD_LOGIC;
      dout_osc,dout_ran: IN STD_LOGIC_VECTOR(7 downto 0);
      dout           : OUT STD_LOGIC_VECTOR(7 downto 0));
END mux_out ;
-------------------------------------------------------------
--------------
ARCHITECTURE a OF mux_out  IS
     SIGNAL sel : STD_LOGIC;
BEGIN

PROCESS(sel_mux,dout_osc,dout_ran)
     BEGIN
     sel<=sel_mux;
     IF sel = '0' THEN
         dout <= dout_osc;
     ELSE
         dout <= dout_ran;
     END IF;

     END PROCESS;
END a;
-------------------------------------------------------------
-------------
```

โปรแกรม DDS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
------------------------------------------------------------
-
entity dds is
port ( clk : in std_logic;
        rst : in std_logic;
        sel : in std_logic_vector(1 downto 0);
        freq_data: in std_logic_vector(23 downto 0);
        dout: out std_logic_vector(7 downto 0)
    );
end dds;
------------------------------------------------------------
--
architecture rtl of dds is
------------------------------------------------------------
signal address : unsigned( 5 downto 0);
signal dout1,dout2,dout3  : std_logic_vector (7 downto 0);
signal result  : std_logic_vector (7 downto 0);
signal result1 : std_logic_vector (7 downto 0);
signal accum   : std_logic_vector (23 downto 0);
signal sign    : std_logic;

begin
------------------------------------------------------------
acc:process(clk,rst,freq_data)
variable count1:std_logic_vector(23 downto 0);
  begin
 count1:= freq_data ;
 if rst = '0' then
    accum<="000000000000000000000000";
     elsif clk'event and clk='1'then
       accum<=count1 + accum;
   end if;
   end process;

------------------------------------------------------------
sign <= accum(23);
------------------------------------------------------------
```

```
process(accum)
begin
if accum(22) = '0' then
address <= unsigned ( accum (21 downto 16));
else
address <= unsigned (not accum (21 downto 16));
end if;
end process;
-------------------------------------------------------
lookup_sin:process(address,sign)
subtype slv8 is std_logic_vector(7 downto 0);
type rom64 is array (0 to 63 ) of slv8;
variable sinus_rom : rom64 :=(
"00000000","00000011","00000110","00001001","00001100",
"00001111","00010010","00010101","00011000","00011011",
"00011110","00100001","00100100","00100111","00101010",
"00101101","00110000","00110011","00110110","00111001",
"00111010","00111110","01000001","01000011","01000110",
"01001001","01001011","01001110","01010000","01010010",
"01010101","01010111","01011001","01011011","01011110",
"01100000","01100010","01100100","01100110","01100111",
"01101001","01101011","01101100","01101110","01110000",
"01110001","01110010","01110100","01110101","01110110",
"01110111","01111000","01111001","01111010","01111011",
"01111011","01111100","01111101","01111101","01111110",
"01111110","01111110","01111110","01111110");
begin
if sign = '1' then
     result <= sinus_rom(to_integer(address));
else
     result <= std_logic_vector (-
signed(sinus_rom(to_integer(address))));
end if;
end process;
-------------------------------------------------------
----------
outreg:process(clk,rst)
begin
if rst ='0' then
     dout1 <= (others => '0');
elsif rising_edge(clk) then
     dout1 <= result;
     dout1(7) <= not result(7);
     dout1(6 downto 0 ) <= result ( 6 downto 0);
end if;
end process;
-------------------------------------------------------
----------
```

```
out_squre:process( accum)
begin
if  accum(23) ='0' then
     dout2 <=  "11111111";
else
dout2 <= (others => '0');
end if;
end process;
-------------------------------------------------------


lookup_sawtooth:process(address,sign)
subtype slv8 is std_logic_vector(7 downto 0);
type rom64 is array (0 to 63 ) of slv8;
variable sawtooth_rom : rom64 :=(
"00000000","00000010", "00000100","00000110", "00001000",
"00001010","00001100", "00001110","00010000", "00010010",
"00010100","00010110", "00011000", "00011010","00011100",
"00011110","00100000", "00100010", "00100100","00100110",
"00101000","00101010", "00101100", "00101110","00110000",
"00110010","00110100", "00110110", "00111000","00111010",
"00111100","00111110", "01000000", "01000010","01000100",
"01000110","01001000", "01001010", "01001100","01001110",
"01010000","01010010", "01010100", "01010110","01011000",
"01011010","01011100", "01011110", "01100000","01100010",
"01100100","01100110", "01101000", "01101010","01101100",
"01101110","01110000", "01110010", "01110100","01110110",
"01111000","01111010", "01111100", "01111110");
begin
if sign = '1' then
     result1 <= sawtooth_rom(to_integer(address));
else
     result1 <= std_logic_vector (-
signed(sawtooth_rom(to_integer(address))));
end if;
end process;
-------------------------------------------------------
out_sawtooth:process(clk,rst)
begin
if rst ='0' then
     dout3 <= (others => '0');
elsif rising_edge(clk) then
     dout3<= result1;
     dout3(7) <= not result1(7);
     dout3(6 downto 0 ) <= result1 ( 6 downto 0);
end if;
end process;
-------------------------------------------------------
```

```
dout<= dout1 when sel ="00" else
       dout2 when sel ="01" else
       dout3 when sel ="10" else
       accum(23 downto 16);
end rtl;
```

โปรแกรม Matlab เพื่อทดสอบการทำงานของ CORDIC

```
clc;
angle=input(' Enter angle degree:');
z=1:24;
x=1:24;
y=1:24;
a=1:24;
k=1.64679;
z(1)=angle*(pi/180);
x(1)=1/k;
y(1)=0;
z_d=angle;
for i=0:23
   if z_d>=0
      z(i+2)=z(i+1)-(atan(2^(-i)));
      x(i+2)=x(i+1)-((2^(-i))*y(i+1));
      y(i+2)=y(i+1)+((2^(-i))*x(i+1));
      z_d=z(i+2)*(180/pi);
   else
      z(i+2)=z(i+1)+(atan(2^(-i)));
      x(i+2)=x(i+1)+((2^(-i))*y(i+1));
      y(i+2)=y(i+1)-((2^(-i))*x(i+1));
      z_d=z(i+2)*(180/pi);
   end
   format long g
   a(i+1)=atan(2^(-i))*(180/pi)
end
z1=z(i+2)*(180/pi)
x1=x(i+2)
y1=y(i+2)
```

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงได้ด้วยดีด้วยความกรุณาของ รศ.ดร.กอบชัย เดชหาญ และอาจารย์ศรวัฒน์ ชิวปรีชา อาจารย์ที่ปรึกษา ผู้คอยเอาใจใส่ดูแลและเมตตาแก่ศิษย์ด้วยดีเสมอมา คณะผู้จัดทำรู้สึกซาบซึ้งในความกรุณาเป็นอย่างยิ่ง และขอกราบพระคุณเป็นอย่างสูง ขอขอบคุณสำหรับกำลังใจ คำแนะนำเกี่ยวกับปัญหาต่างๆ และความช่วยเหลือทุกอย่างระหว่างการทำปริญญานิพนธ์นี้ของเพื่อนๆทุกคน และขอขอบพระคุณอาจารย์ทุก ๆ ท่านที่ประสิทธิ์ประสาทวิชาความรู้ต่าง ๆ ให้แก่ศิษย์ทั้งโดยทางตรงและทางอ้อม

สุดท้ายนี้ขอกราบขอบพระคุณบิดา มารดา ที่ให้ความสำคัญกับการศึกษาของลูกและให้การสนับสนุนเอาใจใส่ดูแลด้วยดีเสมอมา รวมทั้งกำลังใจอันยิ่งใหญ่อย่างหาที่เปรียบมิได้

ธงชัย สุบิน

พฤทธิ์ เกษรัตน์

บรรณานุกรม

1. D. L. Perry, "VHDL", McGraw-Hill Company ,Inc ,volume 3 ,1998.

2.คู่มือการอบรมเชิงปฏิบัติการ, "VHDL Application Workshop และ FPGA Design Workshop",ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

3. D.L.Perry,"VHDL", New York : McGraw – Hill, 1995

4. บริษัท แอสทรอน ลอจิก รีเสิร์ชแอนด์ดิวิลอปเมนต์ จำกัด, "เปิดโลก FPGA กับ "WIZARD PLDAO1,"2544\

5. มนัส สังวรศิลป์,วรรัตน์ ภัทรอมรกุล, " คู่มือการใช้งาน MATLAB ฉบับสมบูรณ์", ศูนย์การพิมพ์พลชัอินโฟเพลส ,กรุงเทพฯ 2543

6. J. G.Proakis and D. G. Manolakis, "Introduction to Digital Signal Processing" , Macmillan publisling,1988,p372-376