ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการสร้างวงจรเชิงเลขโดยใช้อุปกรณ์ FPGAs

Digital Circuit Design Using VHDL Language and Implementation Circuit Using FPGAs

นักศึกษา

1. นายดอนสัน ปงผาบ รหัสประจำตัว 38031304
2. นายปริญญา ทองกองทุน รหัสประจำตัว 38031315
3. นายสาธิต ไวยพันธ์ รหัสประจำตัว 38031328
4. นายโสภัทร ฉายสว่างวงศ์ รหัสประจำตัว 38031332

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

1. ผศ.ประเชิญ ไทรแจ่มจันทร์
2. อาจารย์กิติพงค์ มะโน
3. อาจารย์ปิยะ จิตธรรมมาภิรมย์

คณะกรรมการสอบปริญญานิพนธ์	ลายมือชื่อ
1. อาจารย์กิติพงค์ มะโน	.....
2. อาจารย์อำพล ทองระอา	.....
3. อาจารย์ไพบูลย์ พวงวงศ์ตระกูล	.....
4. อาจารย์ปิยะ ศุภวราสุวัฒน์	.....

วันเดือนปีที่สอบ วันที่ 11 เดือน ธันวาคม พ.ศ. 2539 เวลา 12.00 น. ถึง 13.00 น.

สถานที่สอบ ห้อง ค.310 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

ลงนาม .....

(ผศ.ดร. ธีระพล เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 15 เดือน ธ.ค. พ.ศ. 40

ปริญญานิพนธ์

การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการสร้างวงจรเชิงเลข โดยใช้อุปกรณ์ FPGAs

Digital Circuit Design Using VHDL Language and Implementation Circuit Using FPGAs

นายดอนสัน ปงผาบ
นายปริญญา ทองกองทุน
นายสาธิต ไวยพันธ์
นายโสภัทร ฉายสว่างวงศ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2539

ปริญญานิพนธ์

เรื่อง การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการสร้างวงจรเชิงเลข
โดยใช้อุปกรณ์ FPGAs

Digital Circuit Design Using VHDL Language and Implementation
Circuit Using FPGAs

ผู้จัดทำ

1. นายดอนสัน ปงผาบ
2. นายปริญญา ทองกองทุน
3. นายสาธิต ไวยพันธ์
4. นายโสภัทร ฉายสว่างวงศ์

อาจารย์ที่ปรึกษา

ลงนาม.....

(ผศ.ประเชิญ ไทรแจ่มจันทร์)

ลงนาม.....

(อาจารย์กิติพงค์ มะโน)

ลงนาม.....

(อาจารย์ปิยะ จิตธรรมมาภิรมย์)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม.....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการสร้างวงจรเชิงเลข
โดยใช้อุปกรณ์ FPGAs
Digital Circuit Design Using VHDL Language and Implementation
Circuit Using FPGAs

วัตถุประสงค์

1. เพื่อศึกษาภาษา VHDL และอุปกรณ์ FPGAs
2. เพื่อศึกษาโปรแกรมสำหรับออกแบบสร้างและเลียนแบบการทำงานของวงจรที่ใช้ในการโปรแกรมอุปกรณ์ FPGAs
3. เพื่อออกแบบวงจรเชิงเลขเฉพาะงานโดยใช้ภาษา VHDL
4. เพื่อสร้างวงจรเชิงเลขเฉพาะงานโดยใช้ภาษา VHDL
5. เพื่อนำวงจรเชิงเลขเฉพาะงานมาทดสอบและนำไปใช้งานได้

ประโยชน์ที่คาดว่าจะได้รับ

1. ได้ความรู้เกี่ยวกับภาษา VHDL และอุปกรณ์ FPGAs
2. ได้ความรู้เกี่ยวกับโปรแกรมสร้างวงจรและเลียนแบบการทำงานของวงจรเพื่อใช้ในการโปรแกรมลงอุปกรณ์ FPGAs
3. ได้ความรู้ในการออกแบบวงจรเชิงเลขเฉพาะงาน
4. สร้างวงจรเชิงเลขเฉพาะงานโดยใช้อุปกรณ์ FPGAs ได้
5. สามารถนำวงจรเชิงเลขเฉพาะงานมาทดสอบและใช้งานได้

การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการสร้างวงจรเชิงเลขโดยใช้อุปกรณ์ FPGAs

นายดอนสัน ปงผาบ
นายปริญญา ทองกองทุน
นายสาธิต ไวยพันธ์
นายโสภัทร ฉายสว่างวงศ์

อาจารย์ที่ปรึกษา
ผศ.ประเชิญ ไทรแจ่มจันทร์
อาจารย์กิติพงศ์ มะโน
อาจารย์ปิยะ จิตธรรมมาภิรมย์

ปีการศึกษา 2539

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอหลักการออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL และการสร้างวงจรเชิงเลขโดยใช้อุปกรณ์ FPGAs ซึ่งได้ออกแบบวงจรตั้งเวลาขนาด 2 ช่อง โดยใช้ภาษา VHDL และนำมาแปลงไฟล์เพื่อให้ได้ไฟล์ของวงจรโดยใช้โปรแกรมของบริษัท วิวลอจิก และนำไฟล์ของวงจรที่ได้ไปแปลงเป็นไฟล์ XNF และ bit ตามลำดับโดยใช้โปรแกรมของบริษัท XILINX จากนั้นนำไฟล์ที่มีส่วนขยายเป็น bit ส่งให้อุปกรณ์ FPGAs เพื่อให้อุปกรณ์ FPGAs ทำหน้าที่เป็นวงจรตั้งเวลาขนาด 2 ช่อง และได้ทดลองนำอุปกรณ์ FPGAs ไปทดสอบเป็นวงจรตั้งเวลาขนาด 2 ช่อง ปรากฏว่าได้ผลถูกต้องสอดคล้องตามหลักการที่นำเสนอทุกประการ

DIGITAL CIRCUIT USING VHDL LANGUAGE AND IMPLEMENTATION CIRCUIT USING FPGAs

MR.DONSON PONGPAB
MR.PRINYA THONGKONGTHUN
MR.SATHIT WAIYAPHUN
MR.SOPHAT CHAISAWANGWONG

ADVISORS

ASSIST.PROF.PRACHURN CHAIGAMGAND
MR.KITIPONG MANO
MR.PIYA JITTHAMMAPIROM

1996

ABSTRACT

This thesis presents the designing of the digital circuit using VHDL language and the implementation of the circuit using FPGAs. The two channels timer circuit have been designed by using the VHDL language. Then,the VHDL language will be converted to be a circuit file. And then, a circuit file will be converted to be a XNF file and a bit file respectively by the software of the XILINX Co.,Ltd.. After that, a bit file will be downloaded into the FPGAs device. Finally, the FPGAs device will be tested. Consequently, the FPGAs device can be work completely.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์นี้สามารถสำเร็จลุล่วงไปได้ด้วยดี จากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน นอกจากนี้ยังได้รับความกรุณาจากอาจารย์ที่ปรึกษาประจำโครงงาน และอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรมทุกท่านในการให้คำปรึกษาแนะนำ และ ความช่วยเหลือต่างๆ ตลอดจนให้โอกาสในการทำโครงงานอย่างเต็มที่ ทั้งด้านเวลาสถานที่ เครื่องมือและอุปกรณ์ต่างๆ และขอขอบคุณบุพการีผู้ให้กำเนิดที่ให้โอกาสในการศึกษา เพื่อนตลอดจนผู้ที่เกี่ยวข้องทุกคนที่ให้คำแนะนำต่างๆ และเป็นกำลังใจในการทำงาน จนทำให้โครงงานนี้สำเร็จลุล่วงไปได้ด้วยดี

สารบัญ

เรื่อง	หน้า
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	VIII
สารบัญภาพ	IX
บทที่ 1 บทนำ	1
บทที่ 2 ทฤษฎีและหลักการ	3
2.1 กล่าวนำ	3
2.2 แนะนำภาษา VHDL	5
2.2.1 ข้อกำหนดของภาษา VHDL	5
2.2.2 องค์ประกอบพื้นฐานในภาษา VHDL	8
2.3 การบรรยายเชิงพฤติกรรม	13
2.3.1 โพรเซส	13
2.3.2 การตรวจสอบการกระทำ	17
2.3.3 การหยุดรอ	18
2.3.4 การสร้างองค์ประกอบย่อย	19
2.3.5 แพคเกจ	20
2.4 การบรรยายเชิงข้อมูล	21
2.4.1 การกำหนดทางเลือกข้อมูล	21
2.4.2 การกำหนดการ์ด	22
2.4.3 การกำหนดโครงข่ายของการ์ด	24
2.4.4 การเลือกสรรข้อมูล	25
2.5 การบรรยายเชิงโครงสร้าง	27
2.5.1 ไลบรารีพาร์ท	27
2.5.2 การเชื่อมต่ออุปกรณ์พื้นฐาน	29
2.5.3 การเชื่อมต่ออุปกรณ์ชนิดเดียวกัน	33

เรื่อง หน้า

2.6 โครงสร้างภายในอุปกรณ์ FPGAs 37
2.7 ส่วนที่เป็นองค์ประกอบของลอจิก 39
2.8 ส่วนอินพุทและเอาท์พุทของอุปกรณ์ FPGAs 39
2.9 รายละเอียดการใช้งานของอุปกรณ์ FPGAs 41
2.9.1 การใช้งานในลักษณะสเลฟซีเรียล 42
2.9.2 การใช้งานในลักษณะมาสเตอร์ซีเรียล 44
2.10 ข้อควรระวังในการใช้อุปกรณ์ FPGAs 45
2.11 ขั้นตอนการออกแบบและจำลองการทำงานโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก 46
2.12 ขั้นตอนการสังเคราะห์วงจรโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก 54
2.12.1 การใช้โปรแกรมวิวซีนทีซิท 54
2.12.2 การใช้โปรแกรมวิวดรอว์ 56
2.13 ขั้นตอนในการโปรแกรมลงบนอุปกรณ์ FPGAs โดยใช้ซอฟแวร์ XDM 58
บทที่ 3 การออกแบบและการสร้าง 63
3.1 ลักษณะการออกแบบ 63
3.1.1 การออกแบบจากล่างขึ้นบน 63
3.1.2 การออกแบบจากบนลงล่าง 63
3.2 วิธีการออกแบบ 63
3.3 คุณสมบัติของวงจรตั้งเวลาขนาด 3 ช่อง 64
3.4 ผังการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง 65
3.5 หน้าที่การทำงานของขาสัญญาณแต่ละขา 66
3.6 ขั้นตอนการออกแบบวงจรตั้งเวลาขนาด 3 ช่อง 67
3.6.1 การออกแบบวงจรตั้งเวลาขนาด 3 ช่องโดยใช้ภาษา VHDL 67
3.6.2 การจำลองการทำงาน 79
3.6.3 การสังเคราะห์เป็นวงจรระดับเกท 79
3.6.4 การโปรแกรมวงจรลงบนไอซี FPGAs 81
3.7 การสร้างวงจรทดสอบ 81
3.7.1 วงจรไอซี FPGAs 81

เรื่อง	หน้า
3.7.2 วงจรภาคแสดงผล	82
3.7.3 วงจรควบคุมอุปกรณ์ไฟฟ้า	82
3.4.4 วงจรสวิตช์ควบคุมการทำงาน	83
บทที่ 4 การทดลองและผลการทดลอง	84
4.1 การดาวน์โหลดลงบนบอร์ดตัวอย่างของ FPGAs	84
4.2 การทดลองวงจรตั้งเวลาขนาด 2 ช่อง	86
4.3 ผลการทดลอง	104
บทที่ 5 สรุปอภิปรายและข้อเสนอแนะ	106
5.1 สรุป	106
5.2 ปัญหาและแนวทางการแก้ไข	106
5.3 แนวทางในการพัฒนา	108
ภาคผนวก ก ผังงานและโปรแกรมภาษา VHDL ของวงจรตั้งเวลาขนาด 2 ช่อง	109
ภาคผนวก ข เอกสารของอุปกรณ์ FPGAs ตระกูล XC4000A	175
บรรณานุกรม	199

สารบัญตาราง

ตาราง **หน้า**

ตารางที่ 2.1 คุณสมบัติของ FPGAs ตระกูลต่าง ๆ 38
ตารางที่ 2.2 ตารางประมาณการนับเกทของเกทพื้นฐาน 39
ตารางที่ 2.3 ลักษณะต่าง ๆของการคอนฟิกกูเรชัน 41
ตารางที่ 4.1 การตั้งดิพสวิตช์ต่างๆ ในบอร์ดตัวอย่างของ FPGAs 86
ตารางท 4.2 ผลการเปรียบเทียบเวลาของวงจรตั้งเวลาขนาด 2 ช่องกับนาฬิกาดิจิตอล 100

สารบัญภาพ

รูปภาพ **หน้า**

รูปที่ 2.1 ขั้นตอนการออกแบบระบบเชิงเลข 3
รูปที่ 2.2 การออกแบบระบบเส้นทางของข้อมูล 4
รูปที่ 2.3 ตัวอย่างการออกแบบแบบลำดับขั้น 6
รูปที่ 2.4 การกำหนดการเชื่อมต่อและสถาปัตยกรรม 8
รูปที่ 2.5 ผังการทำงานและการบรรยายการเชื่อมต่อของ clcok_component 9
รูปที่ 2.6 การบรรยายเชิงพฤติกรรมของ clock_component 10
รูปที่ 2.7 ตัวอย่างฟังก์ชัน 10
รูปที่ 2.8 ตัวอย่างโพรซีเจอร์ 11
รูปที่ 2.9 ตัวกระทำในภาษา VHDL 12
รูปที่ 2.10 รูปแบบของการบรรยายแบบโพรเซส 13
รูปที่ 2.11 ตัวอย่างการประกาศตัวกระทำภายในโพรเซส 14
รูปที่ 2.12 เงื่อนไขการกระทำในโพรเซส 15
รูปที่ 2.13 การกระทำในโพรเซส 15
รูปที่ 2.14 (ก) ตัวอย่างโมเดล D Flip-Flop 16
(ข) การบรรยายการเชื่อมต่อของ D Flip-Flop 16
รูปที่ 2.15 การบรรยายเชิงพฤติกรรมของ D Flip-Flop
โดยการใช้ตัวกระทำภายนอกโพรเซส 17
รูปที่ 2.16 การใช้ ASSERT 18
รูปที่ 2.17 การบรรยายโครงสร้างของวงจรเลือกสัญญาณ 2 ออก 1 19
รูปที่ 2.18 โครงสร้างของแพคเกจ 20
รูปที่ 2.19 (ก) ตัวอย่างโมเดลของวงจร 8-to-1 มัลติเพล็ก 21
(ข) การบรรยายเชิงข้อมูลของวงจร 8-to-1 มัลติเพล็ก 22
รูปที่ 2.20 ตัวอย่างการใช้ GUARDED ในการบรรยายการทำงานของ D Filp-Flop 22
รูปที่ 2.21 (ก) โมเดลของ D Flip-Flop 23
(ข) ตัวอย่างการใช้ GUARDED 23
รูปที่ 2.22 ตัวอย่างการใช้ NESTING GUARDED 24
รูปที่ 2.23 การกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน 25

รูปภาพ **หน้า**

รูปที่ 2.24 ฟังก์ชันเลือกสรรข้อมูลแบบ anding 26
รูปที่ 2.25 การใช้ฟังก์ชัน anding กับการเลือกสรรข้อมูล 26
รูปที่ 2.26 (ก) โมเดลของอินเวอร์เตอร์เกท 27
(ข) การบรรยายการเชื่อมต่อ 27
(ค) การบรรยายการทำงาน 28
รูปที่ 2.27 (ก) โมเดลของแนนด์เกท 2 อินพุท 28
(ข) การบรรยายการเชื่อมต่อ 28
(ค) การบรรยายการทำงาน 28
รูปที่ 2.28 (ก) โมเดลของแนนด์เกท 3 อินพุท 29
(ข) การบรรยายการเชื่อมต่อ 29
(ค) การบรรยายการทำงาน 29
รูปที่ 2.29 วงจรเปรียบเทียบทีละบิต 30
รูปที่ 2.30 สัญลักษณ์และสมการบูลีนของวงจรเปรียบเทียบทีละบิต 30
รูปที่ 2.31 การบรรยายการเชื่อมต่อของโปรแกรมเปรียบเทียบทีละบิต 31
รูปที่ 2.32 การบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต 32
รูปที่ 2.33 องค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิต 33
รูปที่ 2.34 วงจรเปรียบเทียบขนาด 4 บิต 34
รูปที่ 2.35 (ก) สัญลักษณ์ของวงจร nibble_comparator 34
(ข) การบรรยายการเชื่อมต่อของวงจร nibble_comparator 35
รูปที่ 2.36 การบรรยายการทำงานของวงจร 35
รูปที่ 2.37 ลักษณะของวงจรแบบสัญญลักษณ์ 36
รูปที่ 2.38 แผนผัง CLB ของตระกูล 4000 40
รูปที่ 2.39 แผนผัง IOBs ของตระกูล 4000 40
รูปที่ 2.40 ลำดับไดอะแกรมในการคอนฟิกเมื่อเริ่มป้อนแหล่งจ่ายไฟเข้าไอซี
และการโปรแกรมใหม่ 42
รูปที่ 2.41 แสดงการต่อใช้งานในลักษณะสเลฟซีเรียล 43
รูปที่ 2.42 แผนภูมิเวลาการป้อนข้อมูลโปรแกรมคอนฟิกในลักษณะสเลฟซีเรียล 43
รูปที่ 2.43 การต่อใช้งานในลักษณะมาสเตอร์ซีเรียล 44

รูปภาพ **หน้า**

รูปที่ 2.44 การต่อใช้งานในลักษณะมาสเตอร์พาราแรล 45
รูปที่ 2.45 หน้าจอเริ่มต้นของโปรแกรมวิวลอจิก 46
รูปที่ 2.46 ภาพเริ่มต้นของการใช้โปรแกรม Speed Wave 46
รูปที่ 2.47 หน้าจอ Open 47
รูปที่ 2.48 หน้าต่างของไฟล์ Fulladd.vhd 47
รูปที่ 2.49 หน้าต่างของ Analyze VHDL File 48
รูปที่ 2.50 หน้าต่างของ VHDL Library Manager 48
รูปที่ 2.51 หน้าต่างของ SCHEMLES 49
รูปที่ 2.52 สัญญาณอินพุทและเอาท์พุทโปรแกรม Fulladd.vhd 49
รูปที่ 2.53 หน้าต่างของเมนู Setup Waveform Streams 50
รูปที่ 2.54 หน้าต่างของ Creat New Stream 50
รูปที่ 2.55 การ Add Signal 51
รูปที่ 2.56 หน้าต่างจำลองการทำงานของ Fulladd.vhd 51
รูปที่ 2.57 หน้าต่างของ Setup Stimulus 52
รูปที่ 2.58 การกำหนดค่าสัญญาณอินพุท A,B และ Cin 52
รูปที่ 2.59 หน้าต่าง Run 53
รูปที่ 2.60 ผลการทำงานของโปรแกรม Fulladd.vhd 53
รูปที่ 2.61 หน้าต่างของโปรแกรมวิวซินทีซิท 54
รูปที่ 2.62 หน้าต่างของ Add VHDL Source Files 54
รูปที่ 2.63 หน้าต่างของ Specity Technology 55
รูปที่ 2.64 ข้อความเมื่อเสร็จสิ้นการคอมไพล์ไฟล์ Fulladd.vhd 55
รูปที่ 2.65 ข้อความเมื่อสิ้นเสร็จการสังเคราะห์ไฟล์ Fulladd.vhd 56
รูปที่ 2.66 หน้าต่างของโปรแกรมวิวดรอว์ 56
รูปที่ 2.67 รูปวงจรระดับเกทของโปรแกรม Fulladd.vhd 57
รูปที่ 2.68 หน้าต่างของ Properties 57
รูปที่ 2.69 หน้าจอเริ่มต้นของโปรแกรม XACT Design Manager 58
รูปที่ 2.70 หน้าต่าง Design Manager 59
รูปที่ 2.71 หน้าต่าง New Project 59

รูปภาพ **หน้า**

รูปที่ 2.72 หน้าต่าง Input Design 59
รูปที่ 2.73 หน้าต่าง Translate 60
รูปที่ 2.74 หน้าต่าง Part Selector 60
รูปที่ 2.75 ผลการ Transtate 61
รูปที่ 2.76 หน้าต่าง XC 4000 Design Implementation Option 61
รูปที่ 2.77 หน้าต่างกระบวนการสร้างไฟล์บิตสตรีม 61
รูปที่ 2.78 หน้าต่าง Communication Setup 62
รูปที่ 3.1 ผังการทำงานการออกแบบวงจรโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก และอุปกรณ์ FPGAs ของบริษัทไซลิงค์ 64
รูปที่ 3.2 ผังการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง 65
รูปที่ 3.3 การแยกส่วนวงจรตั้งเวลาขนาด 3 ช่องออกเป็นส่วนย่อย 68
รูปที่ 3.4 การอธิบายการทำงานแต่ละส่วนย่อยด้วยภาษา VHDL 69
รูปที่ 3.5 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักชั่วโมง 70
รูปที่ 3.6 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักนาที 71
รูปที่ 3.7 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักวินาที 73
รูปที่ 3.8 การบรรยายเชิงโครงสร้างของวงจรนาฬิกา 75
รูปที่ 3.9 การใช้โปรแกรมวิวเทรสแสดงสัญญาณต่างๆ ของวงจรนับหลักชั่วโมง 79
รูปที่ 3.10 การใช้โปรแกรมวิวเทรสแสดงสัญญาณต่างๆ ของวงจรนับหลักนาที 80
รูปที่ 3.11 วงจรระดับเกทของวงจรนับหลักชั่วโมง 80
รูปที่ 3.12 วงจรไอซี FPGAs 81
รูปที่ 3.13 วงจรภาคแสดงผล 82
รูปที่ 3.14 วงจรควบคุมอุปกรณ์ไฟฟ้า 82
รูปที่ 3.15 วงจรสวิตช์ควบคุมการตั้งเวลา 83
รูปที่ 4.1 ภาพถ่ายบอร์ดตัวอย่างของ FPGAs 84
รูปที่ 4.2 การเชื่อมต่อกันระหว่างคอมพิวเตอร์กับบอร์ดตัวอย่างของ FPGAs 85
รูปที่ 4.3 เวลาเริ่มต้นแสดงการทำงานหลังจากทำการดาวน์โหลดเสร็จสิ้น 87
รูปที่ 4.4 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อผ่านไป 1 นาที 87
รูปที่ 4.5 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อเวลาเปลี่ยนจาก 59 นาทีเป็น 1 ชั่วโมง 87

รูปภาพ **หน้า**

รูปที่ 4.6 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อเวลาเปลี่ยนจาก 23:59 เป็น 00:00 88
รูปที่ 4.7 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อเวลาเริ่มต้นที่ 00:00 88
รูปที่ 4.8 การตั้งเวลาในหลักชั่วโมงที่เวลา 04:00 88
รูปที่ 4.9 การตั้งเวลาของวงจรนาฬิกาที่เวลา 04:15 89
รูปที่ 4.10 เวลาเดินจาก 04:15 ไปจนถึง 00:00 89
รูปที่ 4.11 ตำแหน่งของดิพสวิตช์ตัวที่ 7และ8 ในการเลือกวงจรตั้งเวลาช่องที่ 1 90
รูปที่ 4.12 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเริ่มต้นของวงจรตั้งเวลาช่องที่ 1 90
รูปที่ 4.13 การตั้งเวลาในหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 1 90
รูปที่ 4.14 การตั้งเวลาในหลักนาทีของวงจรตั้งเวลาช่องที่ 1 91
รูปที่ 4.15 ตำแหน่งของดิพสวิตช์ตัวที่ 7 และ 8 เมื่อตั้งเวลาช่องที่ 1เสร็จ 91
รูปที่ 4.16 หลอดไฟแสดงผลและรีเลย์ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 1 ทำงาน 91
รูปที่ 4.17 รีเลย์ 1 และหลอดไฟแสดงผลของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 1 หยุดทำงานเมื่อกดสวิตช์ OFF CH1 92
รูปที่ 4.18 การตั้งเวลาในหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 1 92
รูปที่ 4.19 การตั้งเวลาของวงจรตั้งเวลาช่องที่ 1 93
รูปที่ 4.20 ตำแหน่งของดิพสวิตช์ตัวที่ 7 และ 8 เมื่อตั้งเวลาช่องที่ 1 เสร็จ 93
รูปที่ 4.21 ตำแหน่งของดิพสวิตช์ตัวที่ 7 และ 8 ในการเลือกตั้งเวลาช่องที่ 2 93
รูปที่ 4.22 การตั้งเวลาในหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 2 94
รูปที่ 4.23 การตั้งเวลาของวงจรตั้งเวลาช่องที่ 2 94
รูปที่ 4.24 ตำแหน่งของดิพสวิตช์ตัวที่ 8 เมื่อตั้งเวลาช่องที่ 2 เสร็จ 94
รูปที่ 4.25 หลอดไฟแสดงผลและรีเลย์ 1 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 1 ทำงาน 95
รูปที่ 4.26 หลอดไฟแสดงผลและรีเลย์ 2 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 2 ทำงาน 95
รูปที่ 4.27 หลอดไฟแสดงผลและรีเลย์ 1 หยุดทำงานเมื่อกดสวิตช์ OFF CH1 96
รูปที่ 4.28 หลอดไฟแสดงผลและรีเลย์ 2 หยุดทำงานเมื่อกดสวิตช์ OFF CH2 96
รูปที่ 4.29 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อตั้งเวลา 2 ช่องพร้อมกัน 97
รูปที่ 4.30 การรีเซ็ทเวลาของวงจรนาฬิกาหลักนาทีให้เป็น 00 97
รูปที่ 4.31 การรีเซ็ทเวลาของวงจรนาฬิกาหลักชั่วโมงให้เป็น 00 98
รูปที่ 4.32 การรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 1 หลักนาทีให้เป็น 00 98

รูปภาพ **หน้า**

รูปที่ 4.33 การรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 1 หลักชั่วโมงให้เป็น 00 99
รูปที่ 4.34 ภาพถ่ายการเชื่อมต่อสายดาวน์โหลดเคเบิลกับบอร์ดตัวอย่างของ FPGAs 101
รูปที่ 4.35 ภาพถ่ายแสดงเวลาเริ่มต้นการทำงานของวงจรตั้งเวลาขนาด 2 ช่อง 101
รูปที่ 4.36 ภาพถ่ายแสดงเวลา 59 นาที 102
รูปที่ 4.37 ภาพถ่ายแสดงเวลา 23 ชั่วโมง 59 นาที 102
รูปที่ 4.38 ภาพถ่ายแสดงการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 103
รูปที่ 4.39 ภาพถ่ายแสดงการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2 103

บทที่ 1
บทนำ

ในอดีตที่ผ่านมาการออกแบบวงจรเชิงเลขต้องใช้อุปกรณ์ต่อกันเป็นจำนวนมาก ทำให้วงจรมีขนาดใหญ่และสิ้นเปลืองเนื้อที่มาก การออกแบบและแก้ไขข้อผิดพลาดของวงจรทำได้ยากและเสียเวลามาก แต่จากวิวัฒนาการที่เปลี่ยนแปลงไปอย่างรวดเร็ว ทำให้การออกแบบวงจรเชิงเลขทำได้อย่างสะดวกและรวดเร็วขึ้นกว่าเดิมมาก ทั้งนี้เพราะได้มีการนำเอาระบบคอมพิวเตอร์ช่วยในการออกแบบทางวิศวกรรม (CAE ย่อมาจาก Computer Aided Engineering) ตลอดจนโปรแกรมต่างๆ ที่พัฒนามาเป็นเครื่องมือช่วยในการออกแบบ ซึ่งจะช่วยลดความยุ่งยากและระยะเวลาในการทำต้นแบบของวงจรเชิงเลขให้น้อยลง

การออกแบบวงจรเชิงเลขนั้นแบ่งเป็น 2 กลุ่ม คือฟูลลี่คัสตอม (Fully Custom) และแบบเซมิคัสตอม (Semi Custom) โดยการออกแบบแบบฟูลลี่คัสตอมนั้น เมื่อทำการออกแบบวงจรเชิงเลขโดยใช้กฎการออกแบบวงจรรวมของโรงงานผู้ผลิตไอซี และทำการสังเคราะห์จนได้วงจรระดับเกท (Gate Level Schematic) แล้วต้องนำไปทำเป็นวงจรในระดับทรานซิสเตอร์ (Transister) ในขั้นตอนสุดท้ายจะต้องนำไปให้โรงงานผลิตออกมาเป็นไอซีสำเร็จรูปให้ แต่แบบเซมิคัสตอมนั้นเราสามารถดำเนินการเองได้จนสิ้นสุดกระบวนการออกแบบ ซึ่งทางกลุ่มก็ได้เลือกใช้แบบเซมิคัสตอมในการออกแบบ โดยทำการออกแบบวงจรเชิงเลขด้วยภาษา VHDL (Very High Speed Integrated Circuit Hardware Description Language) และสังเคราะห์จนได้วงจรในระดับเกทแล้ว จึงนำวงจรโปรแกรมลงบนอุปกรณ์ประเภทที่สามารถโปรแกรมได้ (Programmable Logic Devices) เช่นอุปกรณ์ FPGAs (Field Programmable Gate Arrays) ซึ่งทางคณะผู้จัดทำได้เลือกใช้ในโครงงาน

ภาษา VHDL เป็นภาษาที่ใช้สำหรับบรรยายการทำงานทางฮาร์ดแวร์ของวงจรเชิงเลข (Digital Circuit)โดยตัวภาษานั้นสามารถบรรยายการทำงานของวงจรเชิงเลขได้หลายๆ ลักษณะ เราสามารถบรรยายถึงความสัมพันธ์ระหว่างสัญญาณอินพุทและเอาท์พุท หรือที่เรียกว่าการบรรยายแบบ Behavioral Description หรือบรรยายถึงลักษณะความสัมพันธ์ของสัญญาณต่างๆที่วิ่งอยู่ภายในวงจรเชิงเลขนั้น ซึ่งเรียกว่าการบรรยายแบบ Dataflow Description หรือแบบสุดท้ายเป็นการบรรยายในระดับต่ำ นั่นคือบรรยายว่าในวงจรมีอุปกรณ์ใดต่ออยู่บ้าง ซึ่งเรียกว่าการบรรยายแบบ Structure Description การออกแบบวงจรเชิงเลขโดยใช้ภาษา VHDL นั้น ผู้ออกแบบไม่จำเป็นต้องทราบถึงรายละเอียดของวงจรว่าประกอบด้วยอะไรบ้าง เพียงแต่รู้ถึงการทำงานของวงจร ซึ่งการออกแบบวงจรนั้นเราต้องทำการแบ่งวงจรออกเป็นส่วนย่อยๆ แล้วเขียนโปรแกรม

บรรยายการทำงานของแต่ละส่วนขึ้นมาด้วยภาษา VHDL จากนั้นทำการแปลภาษา (Compile) และจำลองการทำงาน (Simulate) เพื่อตรวจสอบดูว่าผลการทำงานตรงตามจุดประสงค์ที่ต้องการหรือไม่ ถ้าต้องการแก้ไขก็เพียงแต่เข้าไปแก้ไขที่โปรแกรมภาษา VHDL เท่านั้น แล้วทำการแปลภาษาและจำลองการทำงานใหม่ จนกว่าจะได้ผลการทำงานตามที่ต้องการจึงนำไปสังเคราะห์ (Synthesis) เพื่อให้ได้เป็นวงจรแสดงการเชื่อมต่อซึ่งประกอบด้วยเกท (Gate) ฟลิป-ฟลอป (Flip-Flop) และอุปกรณ์พื้นฐานต่างๆหลังจากนั้นจึงนำไปโปรแกรมลงบนอุปกรณ์ FPGAs

อุปกรณ์ FPGAs จะรวมกลุ่มของเกทกับอุปกรณ์โปรแกรมได้ของ PLDs (Programmable Logic Devices) เข้าด้วยกัน โดยภายในตัว FPGAs จะมีส่วนสำคัญ คือ เส้นทางการเชื่อมต่อระหว่างบล็อค ซึ่งเส้นทางเหล่านี้เราสามารถโปรแกรมได้ จำนวนเกทที่เราสามารถนำมาใช้ได้มีอยู่ระหว่าง 600 ถึง 25000 เกท ซึ่งสามารถนำมาใช้ประโยชน์ได้อย่างกว้างขวาง เพราะสามารถโปรแกรมให้ทำงานเป็นฟังก์ชันต่างๆ และยังสามารถโปรแกรมใหม่ได้

จะเห็นได้ว่าภาษา VHDL และอุปกรณ์ FPGAs เป็นเทคโนโลยีที่น่าศึกษา จึงได้ทำการออกแบบโครงงานขึ้นมาโดยมีเป้าหมายคือ สร้างวงจรตั้งเวลาขนาด 3 ช่องโดยใช้ภาษา VHDL บรรยายการทำงานของวงจรและใช้โปรแกรมสังเคราะห์ภาษา VHDL เป็นวงจรตามต้องการลงบนอุปกรณ์ FPGAs เนื้อหาในปริญญานิพนธ์ฉบับนี้ประกอบด้วยรายละเอียดต่างๆ ที่สำคัญดังนี้

บทที่ 2 ทฤษฎีและหลักการ กล่าวถึงภาษา VHDL ซึ่งได้แก่ ข้อกำหนดของภาษา VHDL องค์ประกอบพื้นฐานในภาษา VHDL, การบรรยายเชิงพฤติกรรม, การบรรยายเชิงข้อมูล, การบรรยายเชิงโครงสร้าง ในส่วนต่อมาจะกล่าวถึงอุปกรณ์ FPGAs ซึ่งได้แก่ โครงสร้างภายในส่วนที่เป็นองค์ประกอบของลอจิก, ส่วนอินพุทและเอาท์พุท, การใช้งานในลักษณะสเลฟซีเรียล การใช้งานในลักษณะมาสเตอร์ซีเรียล นอกจากนี้ยังกล่าวถึงขั้นตอนการใช้ซอฟแวร์ของบริษัทวิวลอจิก (Viewlogic) ในการเขียนโปรแกรมภาษา VHDL, การแปลภาษา, การจำลองการทำงาน การสังเคราะห์วงจร และการใช้ซอฟแวร์ของบริษัทไซลิงค์ (Xilinx) ในการแปลวงจรตั้งเวลาขนาด 3 ช่องที่สังเคราะห์ได้เป็นไฟล์บิตสตรีม (Bit Stream) เพื่อดาวน์โหลดลงบนอุปกรณ์ FPGAs

บทที่ 3 การออกแบบและการสร้างวงจรตั้งเวลาขนาด 3 ช่อง กล่าวถึงลักษณะการออกแบบ, วิธีการออกแบบ, คุณสมบัติและผังการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง หน้าที่การทำงานของขาสัญญาณแต่ละขา และขั้นตอนในการออกแบบวงจรตั้งเวลาขนาด 3 ช่อง

บทที่ 4 การทดลองและผลการทดลอง กล่างถึงการดาวน์โหลดโปรแกรม, การทดลองวงจรตั้งเวลาขนาด 2 ช่อง และผลการทดลองวงจรตั้งเวลาขนาด 2 ช่อง

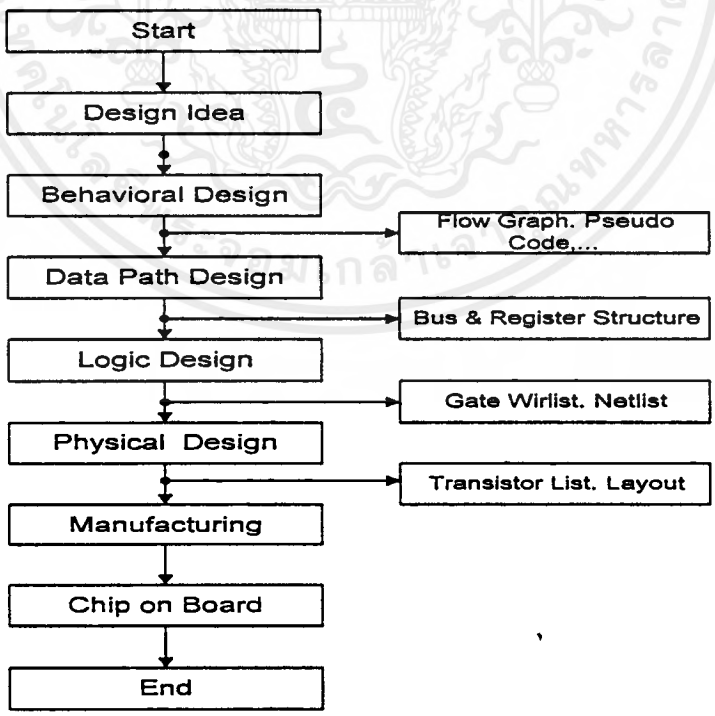
บทที่ 5 สรุปอภิปรายและข้อเสนอแนะ กล่าวถึง ปัญหา, แนวทางการแก้ไข และแนวทางในการพัฒนาเพื่อสามารถนำไปประยุกต์ใช้ได้อย่างกว้างขวาง

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล่าวนำ

ในขณะที่ขนาดและความซับซ้อนของระบบเชิงเลขเพิ่มมากขึ้นคอมพิวเตอร์เพื่อช่วยในการออกแบบ (CAD ย่อมาจาก Computer Aided Design) ถูกนำเข้ามาใช้ในกระบวนการออกแบบมากขึ้น อุปกรณ์และวิธีการออกแบบใหม่ ๆ ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้กับนักออกแบบ และภาษาสำหรับบรรยายอุปกรณ์ฮาร์ดแวร์ (HDL ย่อมาจาก Hardware Description Language) ก็เป็นเครื่องมืออย่างหนึ่งที่ได้รับการพัฒนามาอย่างต่อเนื่อง เพื่อช่วยในการปรับปรุงกระบวนการออกแบบระบบเชิงเลข

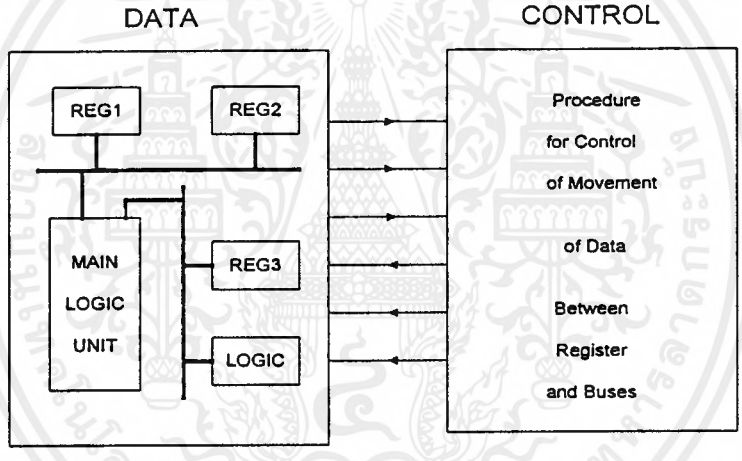
ในการออกแบบระบบเชิงเลข เริ่มตั้งแต่การกำหนดแนวความคิดเบื้องต้นจนกระทั่งได้ออกมาเป็นฮุปกรณ์ฮาร์ดแวร์ที่ใช้งานได้จะต้องผ่านขั้นตอนต่างๆมากมาย และในแต่ละขั้นตอนผู้ออกแบบจะต้องตรวจสอบผลลัพธ์สุดท้ายในแต่ละขั้น ทำการเพิ่มเติมตามความจำเป็นและก็เข้าสู่กระบวนการออกแบบในขั้นต่อไป



รูปที่ 2.1 ขั้นตอนการออกแบบระบบเชิงเลข

รูปที่ 2.1 แสดงขั้นตอนปกติที่ใช้ในการออกแบบระบบเชิงเลขทั่วไป ขั้นแรกผู้ออกแบบจะกำหนดแนวความคิดในการออกแบบ และทำการพัฒนาให้สามารถนำมาใช้ได้อย่างสมบูรณ์ ดังนั้นภายในขั้นตอนนี้จึงจำเป็นที่ผู้ออกแบบจะต้องสร้างรูปแบบระบบในเชิงพฤติกรรมขึ้นมาตรวจสอบซึ่งอาจจะเป็นผังงาน ผังแสดงแบบหรือรหัสคำสั่งเทียม (Pseudo Code) ก็ได้

ขั้นต่อไปเป็นการออกแบบระบบเส้นทางของข้อมูล ผู้ออกแบบจะกำหนดส่วนประกอบของรีจิสเตอร์ (Register) และวงจรตรรก (Logic) ที่จำเป็นเพื่อนำมาประกอบกันเป็นระบบที่สมบูรณ์ แต่ละองค์ประกอบสามารถเชื่อมต่อกันด้วยบัสหนึ่งหรือสองทิศทาง (Unidirectional or Bidirectional Bus) กระบวนการในการควบคุมการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์และวงจรตรรกจะขึ้นอยู่กับพฤติกรรมของระบบที่กำหนดไว้ดังรูปที่ 2.2 [10]



รูปที่ 2.2 การออกแบบระบบเส้นทางของข้อมูล

การออกแบบวงจรตรรกจะเป็นขั้นตอนต่อไป การออกแบบในขั้นนี้เกี่ยวข้องกับการใช้เกทพื้นฐานและฟลิป-ฟลอปเป็นส่วนประกอบของอุปกรณ์ย่อยต่างๆ ได้แก่ รีจีสเตอร์เก็บข้อมูลบัสวงจรตรรกและส่วนควบคุมฮาร์ดแวร์ ในขั้นสุดท้ายจะออกมาเป็นเครือข่ายของการเชื่อมโยงระหว่างเกทและฟลิป-ฟลอปนั่นเอง ต่อมาเป็นขั้นตอนการเปลี่ยนเครือข่ายการเชื่อมต่อในขั้นตอนที่แล้วให้เป็นทรานซิสเตอร์ลีสและเลย์เอาท์ (Transistor List and Layout) ในขั้นตอนนี้จะเกี่ยวข้องกับการจัดวางเกทและฟลิป-ฟลอปแทนด้วยทรานซิสเตอร์หรือไลบรารีเซลขั้นตอนสุดท้ายเป็นการส่งระบบที่ออกแบบไว้ไปทำการเจือสารที่โรงงานเพื่อผลิตออกมาเป็นวงจรรวมในที่สุด การออกแบบตามวิธีที่กล่าวมานี้เรียกว่าการออกแบบแบบฟูลลี่คัสตอม

2.2 แนะนำภาษา VHDL

ในปี 1981 สถาบันเพื่อการวิเคราะห์และป้องกัน (The Institute for Defense Analysis) ในสหรัฐอเมริกาได้จัดตั้งคณะทำงานขึ้นคณะหนึ่งเพื่อทำการพัฒนาภาษาที่ใช้ในการบรรยายหรืออธิบายรูปแบบการทำงานและความสัมพันธ์ของอุปกรณ์ฮาร์ดแวร์แบบใหม่ขึ้น ผลการทำงานของคณะทำงานชุดนี้ได้ก่อให้เกิดภาษาการบรรยายฮาร์ดแวร์ขึ้นเรียกว่า VHDL (VHSIC Hardware Description Language) โดย VHSIC เป็นชื่อย่อของแผนกหนึ่งของสถาบันที่ทำงานเกี่ยวข้องกับวงจรรวมที่มีความเร็วสูงมาก (Very High Speed Integrated Circuit) ต่อมาในปี 1985 IEEE ได้ทำการผลักดันให้ VHDL กลายเป็นภาษาที่เป็นมาตรฐานและมีการยอมรับกันอย่างกว้างขวางในวงการอุตสาหกรรมคอมพิวเตอร์ ด้วยความสามารถของ VHDL ในด้านการกำหนดพฤติกรรมการทำงานของวงจรให้นักออกแบบสามารถกำหนดรูปแบบพฤติกรรมการทำงานได้ทั้งของวงจรเชิงเลขทั่วไปและในระบบที่แตกต่างออกไป เช่น พฤติกรรมการทำงานของระบบเรดาห์หรือพฤติกรรมการทำงานของระบบเครือข่ายใยประสาทในสมองมนุษย์ ข้อดีหลักที่สำคัญของ VHDL ก็คือภาษานี้สามารถจะใช้ได้ตลอดในทุกระดับขั้นของการออกแบบ นั่นคือในกระบวนการออกแบบตั้งแต่ระดับสูง (System Level) จนถึงในระดับที่ต่ำกว่า (Lower Hardware Level) สามารถใช้ภาษาเดียวกันได้โดยตลอดทำให้เพิ่มประสิทธิภาพในการติดต่อระหว่างกลุ่มที่ทำงานรวมกันได้เป็นอย่างดี

2.2.1 ข้อกำหนดของภาษา VHDL

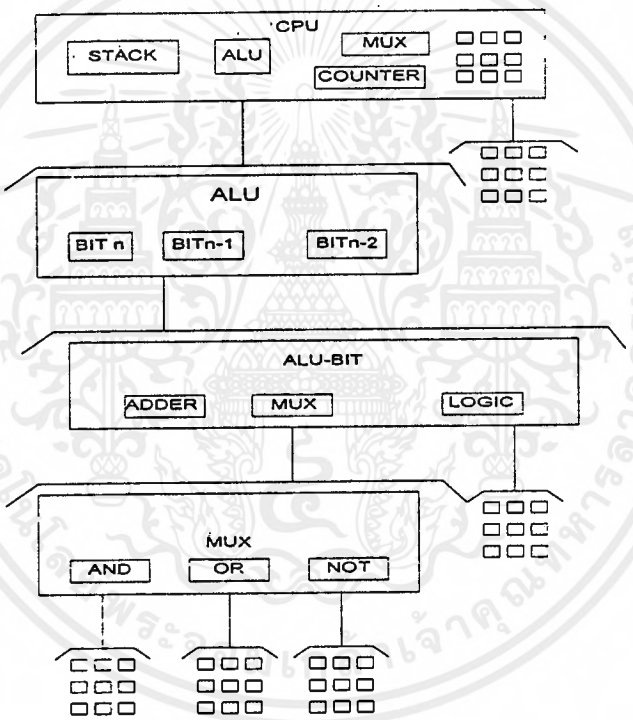
ในเอกสารของ DoD (TheUnitad State Department of Defense) ซึ่งออกมาในเดือนมกราคม ปี 1983 ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ไว้ดังนี้

ลักษณะทั่วไป

กำหนดไว้ว่า VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถในการอธิบายและออกแบบในระดับสูง ความสามารถในการเลียนแบบ (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้นภาษา VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับของระบบจนถึงระดับเกท เนื่องจากในการทำงานของระบบเชิงเลขจริงทุกองค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่จะทำงานไปพร้อมกัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญอย่างหนึ่งในภาษา VHDL ด้วยเช่นกัน สำหรับภาษาที่ใช้ในการบรรยายฮาร์ดแวร์แล้วความพร้อมเพรียงจะหมายถึงทุก ๆ คำสั่ง องค์ประกอบ เกท หรือวงจรตรรกจะนำมาปฏิบัติทั้งหมด ดังนั้นในตอนท้ายแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไปพร้อม ๆกัน

การสนับสนุนการออกแบบแบบลำดับขั้น

การออกแบบแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่ง สำหรับการออกแบบที่มีหลายระดับ ในการออกแบบจะประกอบด้วย ส่วนการบรรยายการเชื่อมต่อและส่วนการบรรยายหน้าที่การทำงาน หน้าที่การทำงานของระบบสามารถกำหนดได้ด้วยตัวเองหรือถูกกำหนดโดยโครงสร้างประกอบด้วยองค์ประกอบย่อยๆ ลงไปได้เช่นกัน แต่ที่ระดับล่างสุดองค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนดการทำงานโดยลักษณะแบบโครงสร้างได้ ดังรูป 2.3 [10]



รูปที่ 2.3 ตัวอย่างการออกแบบแบบลำดับขั้น

ไลบรารี

ภาษา VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของอุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูกต้องควรจะถูกเก็บไว้ในไลบรารี หลังจากที่ได้ผ่านการแปลเรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไปใช้ได้ด้วย

ลำดับคำสั่ง

แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกัน จะเป็นคุณสมบัติที่สำคัญของภาษา VHDL ก็ตาม ตัวภาษาเองยังได้มีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งเอาไว้ เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบก็ยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายในของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if-then-else และ loop การบรรยายแบบลำดับคำสั่ง ทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของภาษา VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันอยู่

การกำหนดคุณสมบัติ

นอกจากการกำหนดอินพุทเอาท์พุทแล้ว เงื่อนไขอื่นก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์เช่นกัน ทั้งนี้ก็รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้น ภาษาสำหรับการออกแบบที่ดีควรที่จะช่วยให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาดทางกายภาพ เวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่น ๆ ความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL

ชนิดของข้อมูล

ภาษา VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิดบิท และบูลีนเท่านั้น แต่ยังสามารถกำหนดชนิดของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม และชนิดลำดับการนับ (Enumerate type) หรือแม้แต่ชนิดของข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเอง

โปรแกรมย่อย

ความสามารถในการใช้ฟังก์ชันและโพรซีเจอร์ (Procedure) เป็นข้อกำหนดอีกอย่างหนึ่งในภาษา VHDL เราสามารถที่จะใช้โปรแกรมย่อยในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก และการกำหนดตัวกระทำต่าง ๆ ทั้งเก่าและใหม่ได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

การควบคุมเวลา

ภาษา VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ การตรวจสอบการออกแบบเกทหรือการหน่วงเวลาสามารถกระทำได้โดยการ กำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอยเหตุการณ์ นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณนาฬิกาได้

การกำหนดแบบโครงสร้าง

การกำหนดโครงสร้างขององค์ประกอบสามารถกระทำได้ในทุกระดับของการออกแบบ การกำหนดโครงสร้างขององค์ประกอบรวมที่เกิดจากองค์ประกอบย่อยๆ ที่แตกต่างกันหรือเหมือนกันก็เป็นข้อกำหนดมาตรฐานอย่างหนึ่ง

2.2.2 องค์ประกอบพื้นฐานในภาษา VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบในภาษา VHDL ประกอบด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 2.4 การบรรยายการเชื่อมต่อจะขึ้นต้นด้วยคำว่า ENTITY ตามด้วยชื่อขององค์ประกอบและคำว่า IS ภายในบรรยายถึงพอร์ตการติดต่ออินพุทเอาท์พุทพอร์ตขององค์ประกอบ ส่วนลักษณะภายนอกอื่นๆ เช่น เวลา อุณหภูมิ ก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้บรรยายหน้าที่การทำงานขององค์ประกอบ หน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุทเอาท์พุท และพารามิเตอร์อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อ ดังรูปที่ 2.4 เป็นการบรรยายหน้าที่ขององค์ประกอบเริ่มต้นหลังคำว่า BEGIN เป็นต้นไป

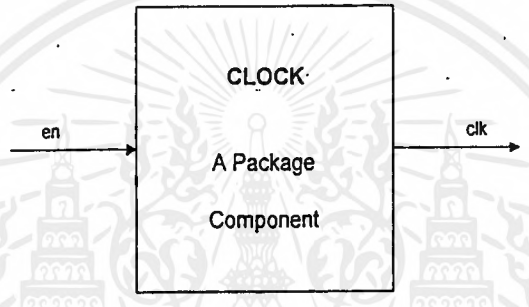
```
ENTITY component_name IS
        input and output port.
        physical and other parameters.
END component_name;
```

```
ARCHITECTURE identifier OF component_name IS
        declarations.
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 2.4 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

การกำหนดการเชื่อมต่อ

การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ ในระดับนี้จะต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบภายนอก ดังตัวอย่างในรูปที่ 2.5 แสดงผังการทำงานและการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับให้สัญญาณนาฬิกา บรรทัดแรกเป็นการกำหนดชื่อขององค์ประกอบซึ่งกำหนดเป็นชื่อ clock_component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็บ IN และ OUT กำหนดโหมดของสัญญาณเป็นอินพุทหรือเอาท์พุท และ BIT แสดงชนิดของข้อมูล




```
ENTITY clock_component IS
  PORT (en : IN BIT; clk : OUT BIT);
END clock_component;
```

รูปที่ 2.5 ผังการทำงานและการบรรยายการเชื่อมต่อของ clock_component

การกำหนดรูปแบบการบรรยาย

หน้าที่การทำงานขององค์ประกอบจะถูกบรรยายในส่วนนี้ การบรรยายสามารถกำหนดค่าของสัญญาณเอาท์พุทในจำนวนของอินพุตหรือในรูปขององค์ประกอบแบบอื่นๆ หรือทั้งสองอย่างรวมกันได้ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 2.6 ซึ่งเป็นการบรรยายในเชิงพฤติกรรม en เป็นอินพุทและ clk เป็นเอาท์พุท PROCESS เป็นค่าเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรมภายในโพรเซส กำหนดให้ periodic เป็นตัวแปรที่มีค่าเริ่มต้นเป็น '0' ถ้าสัญญาณ en มีค่าเป็น '1' ค่าของ periodic จะถูกคอมพลีเมนต์ (Complement) และส่งค่าให้กับ clk ซึ่งเป็นสัญญาณเอาท์พุท คำสั่ง WAIT กำหนดให้สัญญาณมีคาบเวลาเป็น 1 ไมโครวินาที

```
ARCHITECTURE  behavioral OF clock_component IS
      BEGIN
        PROCESS
           VARIABLE periodic : BIT :='0';
           BEGIN
                IF en = '1' THEN
                   periodic := NOT periodic ;
                END IF ;
                   clk <= periodic ;
                WAIT FOR 1 ns ;
        END PROCESS ;
END behavior ;
```

รูปที่ 2.6 การบรรยายเชิงพฤติกรรมของ clock_component

โปรแกรมย่อย

การใช้ฟังก์ชันและโพรซีเจอร์ในภาษา VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมชั้นสูงทั่วไป ค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรง เช่น ถ้าเราใช้ฟังก์ชันแทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรตรรกจริงๆ ในขณะที่ถ้าเราใช้โปรแกรมในการเปลี่ยนชนิดของข้อมูลหรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่ 2.7 แสดงการใช้ฟังก์ชัน โดยกำหนดให้ x เป็นตัวแปรชนิดบิต แทนการกระทำในสมการบูลีน รูปที่ 2.8 แสดงโพรซีเจอร์ที่เป็นตัวกลับสัญญาณ

```
FUNCTION hav (a,b,c : BIT) RETURN  BIT  IS
   VARIABLE  x : BIT
     BEGIN
          x := ((NOT a) AND (NOT b) AND c) ;
          RETURN x;
END   hav ;
```

รูปที่ 2.7 ตัวอย่างฟังก์ชัน

```
LIBRARY mgc_portable;
USE mgc_portable.qsim_logic.ALL;
ENTITY invert_example IS
    PORT (in_a : IN qsim_state;    out_b : OUT qsim_state;)
END invert_example
ARCHITECTURE  alg OF invert_example IS
      PROCEDURE qsim_inverter (SIGNAL bit_in : IN qsim_state;
                    SIGNAL bit_out : OUT qsim_state) IS
        BEGIN
                IF bit_in = '1' THEN
                    bit_out <= '0';
                ELSEIF  bit_in = '0' THEN
                    bit_out <= '1' ;
                ELSE
                    bit_out <= 'X' ;
                END IF ;
END qsim_state;
          BEGIN
              qsim_inverter (in_a,out_b) ;
END alg;
```

รูปที่ 2.8 ตัวอย่างโพรซีเจอร์

โอเปอร์เรเตอร์

การบรรยายเชิงพฤติกรรมในภาษา VHDL ก็มีตัวกระทำทางลอจิกและคณิตศาสตร์เช่นเดียวกับภาษาซอฟแวร์ทั่วไปดังรูปที่ 2.9 [10]

PREDEFINED OPERATORS	
LOGICAL OPERATORS :	NOT AND OR NAND NOR XOR
OPERAND TYPE :	BIT BOOLEAN
RESULT TYPE :	BIT BOOLEAN
RELATIONAL OPERATORS :	= /= < <= > >=
OPERAND TYPE :	any type
RESULT TYPE :	Boolean
ARITHMETIC OPERATORS :	+ - * / MOD REM ABS
OPERAND TYPE :	INTEGER REAL Physical
RESULT TYPE :	INTEGER REAL Physical
CONCANTENATION OPERATORS :	&
OPERAND TYPE :	array of any type
RESULT TYPE :	array of any type

รูปที่ 2.9 ตัวกระทำในภาษา VHDL

เวลาและความพร้อมเพรียง

ในวงจรอิเล็กทรอนิกส์ อุปกรณ์ทุกๆตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลาเข้ามาเกี่ยวข้องเสมอในทุกเหตุการณ์ที่เกิดขึ้น ภาษา VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อสามารถบรรยายรูปแบบ และการป้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของสถาปัตยกรรมบรรยาย (Architecture) จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โพรเซสซึ่งมีการทำงานภายในเป็นแบบลำดับคำสั่งก็ตาม หากมีหลายโพรเซสอยู่ภายในโครงสร้างเดียวกัน ทุกโพรเซสก็จะทำงานไปพร้อมกันด้วย

สัญญาณและตัวแปร

สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการส่งผ่านค่าของสัญญาณ เช่น w <= a AFTER 12 nS หมายถึงกำหนดค่าของสัญญาณ a ให้กับ w หลังจากเวลาผ่านไป 12 nS ในทางตรงกันข้ามตัวแปร

มีลักษณะเป็นเสมือนตัวกลางซอฟแวร์ที่ใช้ในการส่งผ่านข้อมูล และไม่มีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วย การกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ := ตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นใน ฟังก์ชัน โพรซีเจอร์ และโพรเซส

2.3 การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรม เป็นการบรรยายลักษณะการเปลี่ยนแปลงของข้อมูลในรูปแบบของอัลกอริธึม สำหรับการคำนวณผลลัพธ์ที่เกิดขึ้นสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูลที่เข้ามาโดยไม่คำนึงถึงว่าลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายใน ในหัวข้อนี้แสดงให้เห็นถึงประโยชน์ในการใช้โปรแกรมย่อยที่จำลองรูปแบบอินพุทและเอาท์พุทในระดับพฤติกรรมแทนการใช้ส่วนฮาร์ดแวร์ รวมถึงข้อกำหนดต่างๆ ที่ควรรู้

2.3.1 โพรเซส (Process)

โพรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดค่าให้กับสัญญาณ โพรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอและจะปฏิบัติงานตามคำสั่งพร้อมกันกับโพรเซสอื่นที่อยู่ภายในสถาปัตยกรรมบรรยายเดียวกัน โพรเซสจะปฏิบัติงานตามคำสั่งทันทีที่มีเหตุการณ์เกิดขึ้นสัญญาณที่อยู่ทางด้านขวามือของสัญญลักษณ์การกำหนดค่าให้กับสัญญาณ (<=) การบรรยายโพรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS ; รูปที่ 2.10 แสดงส่วนประกอบของการบรรยายแบบโพรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้ และส่วนของการปฏิบัติคำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ

```
PROCESS
    declarative part
        .....
BEGIN
    statement part
        .....
END PROCESS;
```

รูปที่ 2.10 รูปแบบของการบรรยายแบบโพรเซส

การกำหนดตัวกระทำภายในโพรเซส

ตัวกระทำภายในโพรเซสมี 3 ชนิดคือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวกระทำทั้งสามชนิดนี้หากมีการประกาศไว้ในโพรเซสใด ก็จะใช้ได้เฉพาะในโพรเซสนั้นเท่านั้น การติดต่อกับภายนอกหรือระหว่างโพรเซส สามารถทำได้โดยใช้สัญญาณหรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE

```
PROCESS
      FILE flush : TEXT IS IN "filename.dat";
     VARIABLE var : BIT;
      CONSTANT n : INTEGER := 0;
        BEGIN
          .....
END PROCESS;
```

รูปที่ 2.11 ตัวอย่างการประกาศตัวกระทำภายในโพรเซส

รูปที่ 2.11 แสดงตัวอย่างการประกาศตัวกระทำภายในโพรเซส ซึ่งจะอยู่ระหว่างคำสั่ง PROCESS และ BEGIN ค่าเริ่มต้นที่ถูกกำหนดให้กับตัวกระทำภายในโพรเซสจะถูกนำมาใช้ในตอนเริ่มต้นของการปฏิบัติเพียงครั้งเดียว ค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกโปรแกรมย่อย

การกำหนดการกระทำภายในโพรเซส

การกระทำภายในโพรเซสจะเป็นการปฏิบัติแบบลำดับ (Sequencial) เสมอ ภายในโพรเซสสามารถใช้รูปแบบของการใช้เงื่อนไขหรือการทำซ้ำได้เช่น IF-THEN-ELSE, CASE-WHEN FOR-LOOP และ WHILE-LOOP ดังตัวอย่างในรูปที่ 2.12 และรูปที่ 2.13

```
ARCHITECTURE demo OF partial_process IS
        .....
    BEGIN
        PROCESS
            .....
          BEGIN
            .....
            x <= '1';
            IF x = '1' THEN
                perfrom action_1 ;
            ELSIF
                perfrom action_2 ;
            END IF;
            .....
        END PROCESS;
END demo ;
```

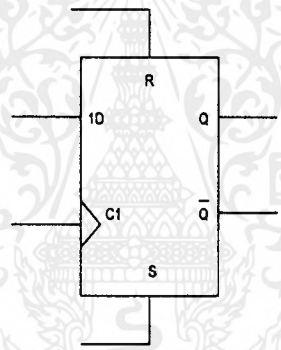
รูปที่ 2.12 เงื่อนไขการกระทำในโพรเซส

```
ARCHITECTURE demo OF partial_process IS
        .....
    BEGIN
        PROCESS
          BEGIN
            .....
            x <= a AFTER 10 nS;
            y <= b AFTER 6 nS;
            .....
        END PROCESS;
END demo;
```

รูปที่ 2.13 การกระทำในโพรเซส

การกระตุ้นและยับยั้งการกระทำของโพรเซส

การกระทำภายในโพรเซสจะเตรียมพร้อม และมีการปฏิบัติงานตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโพรเซสได้โดยการกำหนดรายการของสัญญาณที่ต้องการให้โพรเซสปฏิบัติงาน เมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่เรากำหนดไว้เท่านั้นเหตุการณ์ใด ๆ ที่เกิดขึ้นกับสัญญาณที่ไม่ได้กำหนดไว้ในรายการจะไม่ส่งผลให้มีการกระทำภายในโพรเซส รายการของสัญญาณนี้เรียกว่า Sensitivity List และถูกกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS รูปที่ 2.14 แสดง (ก) ตัวอย่างโมเดลและ (ข)ตัวอย่างการบรรยายการเชื่อมต่อของ D Flip-Flop รูปที่ 2.15 แสดงการบรรยายเชิงพฤติกรรมของ D Flip-Flop โดยการใช้ตัวกระทำภายนอกโพรเซส และมีรายการของสัญญาณ (rst, set, clk) เป็นตัวกระตุ้นการปฏิบัติงานภายในโพรเซส



รูปที่ 2.14 (ก) ตัวอย่างโมเดล D Flip-Flop

```
ENTITY d_sr_flipflop IS
    GENERIC (sq_delay, rq_delay, cq_delay : TIME := 6 nS);
    PORT (d, set, rst, clk : IN BIT; q, qb : OUT BIT);
END d_sr_flipflop;
```

รูปที่ 2.14 (ข) การบรรยายการเชื่อมต่อของ D Flip-Flop

```
ARCHITECTURE behavioral OF d_sr_flipflop IS
        SIGNAL state : BIT := '0';
      BEGIN
   dff: PROCESS (rst, set, clk)
          BEGIN
             IF set = '1' THEN
                state <= '1' AFTER sq_delay;
             ELSIF rst = '1' THEN
                state <= '0' AFTER rq_delay;
             ELSIF clk = '1' AND clk'event THEN
                state <= d AFTER cq_delay;
             END IF;
        END PROCESS;
          q <= state;
         qb <= NOT state;
END behavioral;
```

รูปที่ 2.15 การบรรยายเชิงพฤติกรรมของ D Flip-Flop โดยการใช้ตัวกระทำภายนอกโพรเซส

2.3.2 การตรวจสอบการกระทำ

ภาษา VHDL ได้จัดเตรียมรูปแบบคำสั่งเพื่อใช้ในการตรวจดูการกระทำต่างๆ ที่เกิดขึ้นภายในอุปกรณ์ไว้ดังนี้

```
ASSERT assertion_condition REPORT "reporting_message" SEVERITY severity_level;
```

คำสั่งนี้จะถูกปฏิบัติเมื่อเงื่อนไข assertion_condition มีค่าเป็น False และจะแสดงข้อความ "reporting_Message" ส่วนพารามิเตอร์ severity_level จะเป็นตัวกำหนดระดับความรุนแรงที่เกิดขึ้น ซึ่งแบ่งเป็น 4 ระดับคือ Note,Warning,Error และ False ระดับความรุนแรง Error หรือ Failure จะทำให้การเลียนแบบการทำงานหยุดทันทีหลังจากที่ได้แสดงข้อความ reporting_message

```
ARCHITECTURE behavioral OF d_sr_flipflop IS
        .....
    BEGIN
        ASSERT
            (NOT (set = '1' AND rst = '1'))
        REPORT
            "report and rst both 1"
        SEVERITY NOTE;
        .....
END behavioral;
```

รูปที่ 2.16 การใช้ ASSERT

ส่วนระดับ Note หรือ Warning จะแสดงข้อความ reporting_message แต่ยังคงเลียนแบบการทำงานต่อไป สมมติว่าเราต้องการตรวจสอบการทำงานของ D Flip-Flop ในขณะที่กำลังทำงานได้เกิดสัญญาณในกรณีที่ไม่พึงปรารถนา เช่น สัญญาณ set , rst มีค่าเป็น '1' พร้อมกันก็สามารถตรวจสอบได้โดยแทรกคำสั่ง ASSERT เพื่อตรวจสอบสัญญาณในโพรเซสได้ดังรูปที่ 2.16

2.3.3 การหยุดรอ

การหยุดรอเป็นรูปแบบคำสั่งที่ใช้เพื่อหน่วงเวลาของสัญญาณมีอยู่ 4 รูปแบบดังนี้

```
WAIT FOR waiting_time;
WAIT ON  waiting_sensitivity_list;
WAIT UNTIL waiting_condition;
WAIT;
```

คำสั่งหยุดรอสามารถใช้ได้ภายในโพรเซสและโพรซีเจอร์ที่ไม่ถูกกำหนด Sensitivity List ไว้เท่านั้น WAIT FOR จะหยุดรอเป็นเวลาเท่ากับ wait_time WAIT ON จะหยุดรอจนกว่าจะมีเหตุการณ์เกิดขึ้นกับสัญญาณ waiting_sensitivity_list WAIT UNTIL จะหยุดรอจนกว่าเงื่อนไข waiting_condition เปลี่ยนจาก False เป็น True และ WAIT จะหยุดรอตลอดไป

2.3.4 การสร้างองค์ประกอบย่อย

การสร้างองค์ประกอบย่อยเพื่อใช้ไลบรารี เป็นการบรรยายเชิงโครงสร้างซึ่งจะต้องทำการกำหนดจุดเชื่อมต่อระหว่างอุปกรณ์ต่างๆ ด้วยการประกาศไว้ในส่วนของ Architecture ประกอบด้วย 3 ส่วนตามลำดับ คือส่วนประกาศองค์ประกอบ (Component Declaration) ส่วนที่แสดงรายละเอียดขององค์ประกอบ (Component Specification) ส่วนที่กำหนดการเชื่อมต่อของพอร์ตกับสัญญาณขององค์ประกอบ (Component Instantiation) รูปที่ 2.17แสดงการบรรยายการเชื่อมต่อวงจรเลือกสัญญาณ 2 ออก 1 (Multiplex)

COMPONENT DECLARATION

Format:

```
COMPONENT identifier_name
        PORT (local_port_list);
END COMPONENT;
```

COMPONENT SPECIFICATION

Format:

```
FOR component_instruction_label : component_name USE binding_indication ;
Binding_indication :
ENTITY library.entity_name [ (architecture_name) ]
```

COMPONENT INSTANTATION

Format :

```
lable : component_declaration_name PORT MAP (association_list) ;
```

```
ENTITY mux IS
   PORT (d0,dI,sel : IN bit ; q : OUT bit) ;
END mux;
ARCHITECTURE struct OF mux IS
    COMPONENT and2
        PORT (a , b : IN bit ; z : OUT bit) ;
    END COMPONENT ;
    COMPONENT or2
        PORT (a , b : IN bit ; z : OUT bit) ;
```

รูปที่ 2.17 การบรรยายโครงสร้างของวงจรเลือกสัญญาณ 2 ออก 1

```
END COMPONENT ;
COMPONENT inv
    PORT (i : IN bit ; z : OUT bit) ;
END COMPONENT ;
SIGNAL aa, ab, nsel : bit ;
FOR u1     : inv  USE ENTITY invert (behav) ;
FOR u2,u3  : and2 USE ENTITY and_gt (dflw) ;
FOR u4     : or2  USE ENTITY or_gt (arch1) ;
  BEGIN
        u1 : inv  PORT MAP (sel, nsel);
        u2 : and2 PORT MAP (nsel, d1, ab);
        u3 : and2 PORT MAP (d0, nsel, aa);
        u1 : or2  PORT MAP (aa, ab, q);
    END struct ;
```

รูปที่ 2.17 การบรรยายโครงสร้างของวงจรเลือกสัญญาณ 2 ออก 1 (ต่อ)

2.3.5 แพคเกจ (Package)

แพคเกจ คือ กลุ่มของชนิดข้อมูล โปรแกรมย่อยหรืออุปกรณ์ต่างๆ ที่ออกแบบได้ออกแบบไว้แล้วนำมารวบรวมไว้เป็นกลุ่มๆ อยู่ภายใน เพื่อให้ผู้ออกแบบสามารถเรียกใช้ได้สะดวกดังรูปที่ 2.18

```
PACKAGE identifier IS
    package_declaritive_part
END identifier ;
```

```
PACKAGE BODY identifier IS
    package_declaritive_part
END identifier ;
```

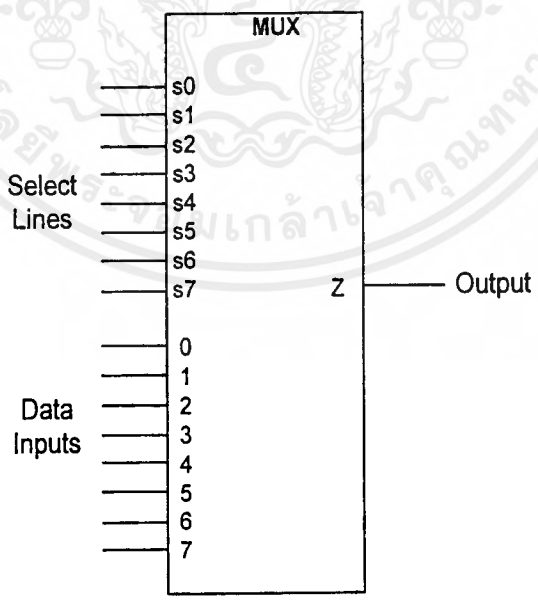
รูปที่ 2.18 โครงสร้างของแพคเกจ

2.4 การบรรยายเชิงข้อมูล

การบรรยายเชิงข้อมูล เป็นการบรรยายถึงการเลื่อนไหลของข้อมูลผ่านรีจิสเตอร์และบัสของระบบ เป็นระดับขั้นของการบรรยายที่อยู่ตรงกลางระหว่างการบรรยายเชิงพฤติกรรมและการบรรยายเชิงโครงสร้าง เครื่องมือที่ใช้ในการควบคุมการเลื่อนไหลของข้อมูลได้แก่ Conditional Selected และ Guarded ในหัวข้อนี้จะแสดงให้เห็นถึงลักษณะและรูปแบบของการบรรยายเชิงข้อมูล ข้อกำหนดและเงื่อนไขต่างพร้อมทั้งตัวอย่างประกอบ

2.4.1 การกำหนดทางเลือกข้อมูล

ในระบบเชิงเลขโครงสร้างของอุปกรณ์ฮาร์ดแวร์ส่วนใหญ่จะถูกใช้สำหรับการคัดเลือกและการนำข้อมูลเข้าสู่บัสและรีจิสเตอร์ รูปที่ 2.19 แสดงตัวอย่างโมเดลและการบรรยายเชิงข้อมูลของวงจร 8-to-1มัลติเพล็ก Sel_lines เป็นสัญญาณที่กำหนดขึ้นมาเพื่อรวมสัญญาณเลือกทั้ง 8 อินพุท (S7-S0) เข้าด้วยกัน และนำสัญญาณไปใช้ในการกำหนดเลือกอินพุทตัวใดตัวหนึ่ง ให้กับเอาท์พุท z ถ้าไม่มีอินพุทใดมีค่าเป็น '1' เลย ค่า '0' จะถูกส่งให้กับเอาท์พุท z หลังจากเวลาผ่านไป 3 nS ส่วนอินพุทอื่น (i7,i6,i5,i4,i3,i2,i1,i0) จะถูกส่งให้ z ขึ้นอยู่กับสัญญาณเลือกใด (s7,s6,s5,s4,s3,s2,s1,s0) มีค่า เป็น '1'



รูปที่ 2.19 (ก) ตัวอย่างโมเดลของวงจร 8-to-1 มัลติเพล็ก

```
ENTITY mux_8_to_1 IS
    PORT (i7, i6, i5, i4, i3, i2, i1, i0   : IN BIT;
          s7, s6, s5, s4, s3, s2, s1, s0 : IN BIT;     z : OUT BIT);
END mux_8_to_1;
ARCHITECTURE dataflow OF mux_8_to_1 IS
    SIGNAL sel_line : BIT_VECTOR (7 DOWNTO 0) ;
  BEGIN
        sel_line <= s7 & s6 & s5 & s4 & s3 & s2 & s1 & s0 ;
            WITH sel_line SELECT
                z <= '0' AFTER 3 NS WHEN "00000000",
                     i7  AFTER 3 NS WHEN "10000000",
                     i6  AFTER 3 NS WHEN "01000000",
                     i5  AFTER 3 NS WHEN "00100000",
                     i4  AFTER 3 NS WHEN "00010000",
                     i3  AFTER 3 NS WHEN "00001000",
                     i2  AFTER 3 NS WHEN "00000100",
                     i1  AFTER 3 NS WHEN "00000010",
                     i0  AFTER 3 NS WHEN OTHERS;
 END dataflow;
```

รูปที่ 2.19 (ข) การบรรยายเชิงข้อมูลของวงจร 8-to-1 มัลติเพล็ก

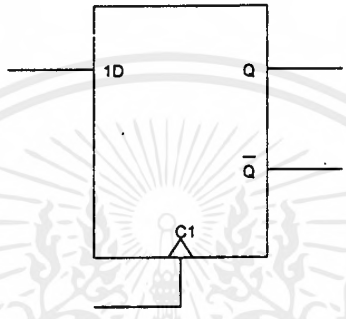
2.4.2 การกำหนดการ์ด

ในการบรรยายเชิงข้อมูล เราสามารถตั้งเงื่อนไขสำหรับการกำหนดค่าสัญาณใดก็ได้ โดยใช้คำสั่ง GUARDED ซึ่งมีรูปแบบในการเขียนดังนี้

```
target <= GUARDED waveforms or conditional waveforms or selected waveform;
```

รูปที่ 2.20 ตัวอย่างการใช้ GUARDED ในการบรรยายการทำงานของ D Flip-Flop

การกำหนด GUARDED ทำได้โดยใช้รูปแบบของคำสั่ง BLOCK ตามด้วยเงื่อนไขภายในวงเล็บ ซึ่งผลลัพธ์ที่ได้จากวงเล็บนี้คือ GUARDED นั่นเอง ในรูปที่ 2.21 เป็นการบรรยายการทำงานของ D flip-flop ที่มีการทำงานเมื่อสัญญาณนาฬิกามีการเปลี่ยนแปลงจาก '0' เป็น '1'หรือสัญญาณขาขึ้น จะเห็นว่า GUARDED ถูกใช้เป็นเงื่อนไขในการกำหนดค่าให้กับ q และ qb นั่นคือถ้าเงื่อนไขภายในวงเล็บมีค่าเป็น FALSE (GUARDED เป็น FALSE) ค่า d และ NOT d ก็จะไม่ถูกส่งค่าให้กับ q และ qb



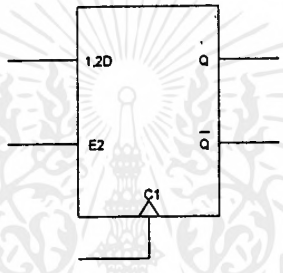
รูปที่ 2.21 (ก) โมเดลของ D flip-flop

```
ENTITY d_flipflop IS
  GENERIC (delay1 : TIME := 4 NS;  delay2 : TIME := 5 NS)
       PORT (d , c : IN BIT ;  q , qb : OUT BIT);
END d_flipflop;
ARCHITECTURE guarding OF d_flipflop IS
   BEGIN
       ff : BLOCK (c = '1' AND NOT c'STABLE)
       BEGIN
            q <= GUARDED d AFTER delay1 ;
           qb <= GUARDED NOT d AFTER delay2 ;
           END BLOCK ff ;
END guarding ;
```

รูปที่ 2.21 (ข) ตัวอย่างการใช้ GUARDED

2.4.3 การกำหนดโครงข่ายของการ์ด

เราสามารถกำหนดการ์ดเป็นโครงข่ายซ้อนกันได้โดยการกำหนดบล็อคซ้อนกัน ซึ่งจะทำให้เราสามารถออกแบบอุปกรณ์ที่มีการทำงานที่ซับซ้อนมากกว่า D flip-flop ในตัวอย่างข้างต้น ดังรูปที่ 2.22 นอกจากสัญญาณอินพุท d จะถูกส่งผ่านไปยัง q โดยมีเงื่อนไขว่า ((c = '1') AND (NOT c'STABLE)) ต้องเป็น TRUE แล้วยังมีเงื่อนไขเพิ่มเติมอีกว่าสัญญาณ enable (e) ต้องเป็น TURE ด้วย GUARD ใน gate : BLOCK แทนเงื่อนไขใดๆ ที่อยู่ภายนอก ในกรณีนี้หมายถึงเงื่อนไขภายในวงเล็บ ((c = '1') AND (NOT c'STABLE)) ในขณะที่ GUARDED จะแทนเงื่อนไขทั้งหมดในกรณีนี้คือ (e = '1') AND ((c = '1') AND (NOT c'STABLE))




```
ENTITY d_flipflop IS
    GENERIC (delay1 : TIME := 4 NS ;  delay2 : TIME := 5 NS)
        PORT (d, c, e : IN BIT; q, qb : OUT BIT);
END d_flipflop;
ARCHITECTURE guarding OF d_flipflop IS
   BEGIN
   edge : BLOCK (( c = '1' ) AND ( NOT c'STABLE ))
        BEGIN
          gate : BLOCK (e = '1' AND GUARD)
              BEGIN
                  q <= GUARDED d AFTER delay1 ;
                  qb <= GUARDED NOT d AFTER delay2 ;
                END BLOCK gate ;
          END BLOCK edge ;
END guarding ;
```

รูปที่ 2.22 ตัวอย่างการใช้ NESTING GUARDED

2.4.4 การเลือกสรรข้อมูล

ในทางฮาร์ดแวร์มีหลายกรณีที่มีความจำเป็นต้องเชื่อมต่อหลายเอาท์พุทไปยังอินพุทหนึ่ง ซึ่งในกรณีเช่นนี้อาจจะทำให้เกิดความสับสนของสัญญาณที่ปะปนกัน ทำให้ไม่สามารถรู้ค่าที่แน่นอนได้ ในทางภาษา VHDL หมายถึง การกำหนดค่าจากหลายสัญญาณให้กับสัญญาณเดียวซึ่งจะให้ผลลัพธ์ที่สับสนได้เหมือนกัน ดังตัวอย่างในรูปที่ 2.23 ซึ่งจะไม่สามารถบอกได้เลยว่าค่าของ circuit_node มีค่าเป็นอะไรถ้าสมมติว่าค่าอินพุท a เป็น '1' และค่าอินพุทอื่น เป็น '0'

```
ENTITY y_circuit IS
    PORT (a , b , c , d : IN BIT ;     z : OUT BIT) ;
END y_circuit;
ARCHITECTURE smoke_generator OF y_circuit IS
    SIGNAL circuit_node : BIT ;
     BEGIN
          circuit_node <= a ;
          circuit_node <= b ;
          circuit_node <= c ;
          circuit_node <= d ;
          z <= circuit_node ;
END smoke_generator;
```

รูปที่ 2.23 การกำหนดค่าสัญญาณที่ทำให้เกิดความสับสน

กรณีเช่นนี้เราสามารถแก้ไขได้โดยการกำหนดรูปแบบฟังก์ชันขึ้นมาเพื่อใช้เลือกสรรข้อมูล รูปที่ 2.24 แสดงตัวอย่างฟังก์ชันเลือกสรรข้อมูลแบบ AND ซึ่งกำหนดให้สัญญาณที่เข้ามาที่โหนดเดียวกันจะถูกนำมารวมกันในลักษณะของการ AND กันเสียก่อน เมื่อนำฟังก์ชันในรูปที่ 2.24 มาใช้ร่วมกับการบรรยายในรูปที่ 2.23 ก็สามารถแก้ปัญหาเรื่องความสับสนของข้อมูลได้ ดังแสดงในรูปที่ 2.25

```
FUNCTION anding (drivers : BIT_VECTOR) RETURN BIT IS
    VARIABLE accumulate : BIT := '1' ;
    BEGIN
        FOR I IN drivers' RANGE LOOP
            accumulate := accumulate AND drivers(i) ;
        END LOOP ;
    RETURN accumulate ;
END anding ;
```

รูปที่ 2.24 ฟังก์ชันเลือกสรรข้อมูลแบบ anding

```
ARCHITECTURE wired_and OF y_circuit IS
    FUNCTION anding (drivers : BIT_VECTOR) RETURN BIT IS
        VARIABLE accumulate : BIT := '1' ;
        BEGIN
            FOR i IN drivers' RANGE LOOP
                accumulate := accumulate AND drivers(i);
            END LOOP ;
        RETURN accumulate ;
    END anding ;
    SIGNAL circuit_node : anding BIT ;
        BEGIN
            circuit_node <= a ;
            circuit_node <= b ;
            circuit_node <= c ;
            circuit_node <= d ;
            z <= circuit_node ;
        END wired_and ;
```

รูปที่ 2.25 การใช้ฟังก์ชัน anding กับการเลือกสรรข้อมูล

2.5 การบรรยายเชิงโครงสร้าง

การบรรยายการทำงานของระบบในเชิงโครงสร้างจะต้องแสดงรายการของอุปกรณ์ทั้งหมดที่ใช้ในระบบและต้องกำหนดการเชื่อมต่อระหว่างอุปกรณ์ต่างๆ ด้วย เพราะว่าการบรรยายในระดับนี้เป็นการบรรยายที่ใกล้เคียงลักษณะของฮาร์ดแวร์จริงที่สุด ภาษา VHDL ได้จัดเตรียมเครื่องมือและลักษณะโครงสร้างของการบรรยายในระดับนี้ไว้ที่สำคัญ 4 ลักษณะคือ

1) ความสามารถในการเลือกหรือกำหนดอุปกรณ์ที่ต้องการได้จากไลบรารี

2) การสร้างไลบรารีเพื่อเก็บอุปกรณ์ที่ผู้ใช้ออกแบบไว้เองได้

3) กลไกในการเชื่อมต่อระหว่างอุปกรณ์

4) โครงสร้างการกำหนดอุปกรณ์ชนิดเดียวกันซ้ำๆ กัน

2.5.1 ไลบรารีพาร์ท

ในหัวข้อนี้เป็นตัวอย่างการใช้ภาษา VHDL อธิบายการทำงานของวงจรเกทพื้นฐาน คือ อินเวอร์เตอร์เกท แนนด์เกท 2 อินพุท และแนนด์เกท 3 อินพุท โดยทำการเขียนอธิบายการทำงานของวงจรเกทขึ้นเองหรือเรียกใช้งานจากไลบรารีในรูปส่วนของเกทก็ได้

อินเวอร์เตอร์เกทโมเดล

รูปที่ 2.26 แสดงโมเดลของอินเวอร์เตอร์เกท โดยในการบรรยายการเชื่อมต่อนั้น i1 เป็นสัญญาณอินพุท o1 เป็นขาสัญญาณเอาท์พุทและการบรรยายการทำงานของอินเวอร์เตอร์เกทนั้น o1 จะเป็นส่วนกลับของสัญญาณ i1 หลังจากเวลาผ่านไปได้ 4 nS

รูปที่ 2.26 (ก) โมเดลของอินเวอร์เตอร์เกท

```
Entity inv IS
    PORT (i1: IN BIT ;  o1: OUT BIT);
End inv;
```

รูปที่ 2.26 (ข) การบรรยายการเชื่อมต่อ

```
ARCHITECTURE single_delay OF inv IS
   BEGIN
        o1 <= NOT i1 ARTER 4 nS;
End single_delay;
```

รูปที่ 2.26 (ค) การบรรยายการทำงาน

แนนด์เกทโมเดล

รูปที่ 2.27 แสดงสัญลักษณ์และการบรรยายของแนนด์เกทชนิด 2 อินพุท

รูปที่ 2.27 (ก) โมเดลของแนนด์เกท 2 อินพุท

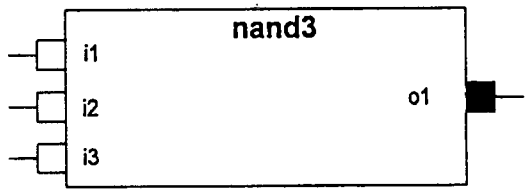
```
ENTITY nand2 IS
     PORT (i1 , i2 : IN BIT ;  o1 : OUT BIT) ;
END nand2 ;
```

รูปที่ 2.27 (ข) การบรรยายการเชื่อมต่อ

```
ARCHITECTURE single_delay  OF nand2  IS
   BEGIN
        o1  <=  i1  NAND i2  AFTER 5 nS ;
END single_delay ;
```

รูปที่ 2.27 (ค) การบรรยายการทำงาน

รูปที่ 2.28 แสดงสัญลักษณ์และการบรรยายของแนนด์เกทชนิด 3 อินพุท



รูปที่ 2.28 (ก) โมเดลของแนนด์เกท 3 อินพุท

```
ENTITY nand3 IS
    PORT ( i1, i2, i3 : IN BIT; o1 : OUT BIT);
END nand3;
```

รูปที่2.28 (ข) การบรรยายการเชื่อมต่อ

```
ARCHITECTURE single_delay OF nand3 IS
  BEGIN
    o1 <= NOT ( i1 NAND i2 AND I3 ) AFTER 6 nS ;
END single_delay ;
```

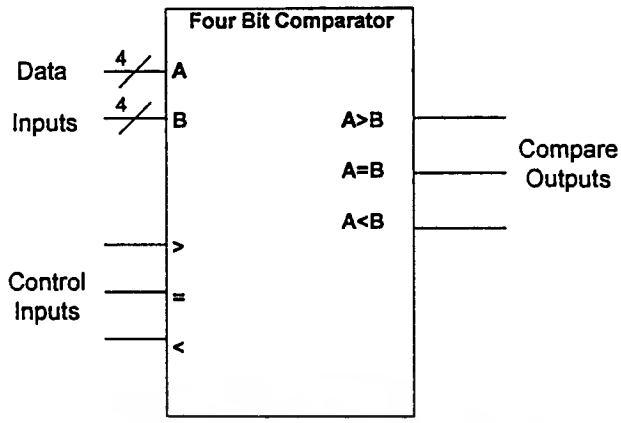
รูปที่ 2.28 (ค) การบรรยายการทำงาน

2.5.2 การเชื่อมต่ออุปกรณ์พื้นฐาน

เมื่อออกแบบเกทพื้นฐานเรียบร้อยแล้ว ขั้นตอนต่อไปเป็นการนำเกทพื้นฐานเหล่านี้มาเชื่อมต่อกันเป็นวงจร ในหัวข้อนี้จะแสดงการออกแบบและการบรรยายเชิงโครงสร้างของวงจรเปรียบเทียบโดยใช้อินเวอร์เตอร์เกทและแนนด์เกท

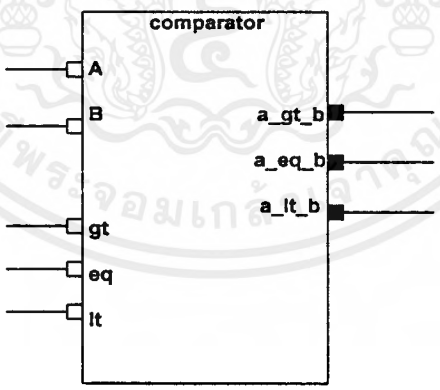
ตัวอย่างการออกแบบวงจรเปรียบเทียบ

วงจรเปรียบเทียบทีละบิต (Bit_comparator) ประกอบด้วยสัญญาณข้อมูล 2 อินพุท สัญญาณควบคุม 3 อินพุท และสัญญาณเปรียบเทียบ 3 เอาท์พุท ดังรูปที่ 2.29



รูปที่ 2.29 วงจรเปรียบเทียบทีละบิต

เอาท์พุท A>B มีค่าเป็น '1' เมื่ออินพุท A มีค่ามากกว่า B (AB = 10) หรือ Aเท่ากับ B และอินพุท > มีค่าเป็น '1' เอาท์พุท A = B มีค่าเป็น '1' เมื่ออินพุท A เท่ากับ B และอินพุท = มีค่าเป็น '1' ส่วนเอาท์พุท A < B จะตรงข้ามกับเอาท์พุท A > B นั่นคือมีค่าเป็น '1' เมื่ออินพุท A มีค่าน้อยกว่า B (AB = 01) หรือถ้า A เท่ากับ B และอินพุท < มีค่าเป็น '1' ซึ่งสามารถเขียนเป็นสมการบูลีนในรูปของแนนด์ เกทได้ ดังรูปที่ 2.30



a_gt_b = ((a.gt)'.(b'.gt)'.(a.b')')'

a_eq_b = ((a.b.eq)'.(a'.b'.eq)')'

a_lt_b = ((a'.lt)'.(b.lt)'.(a'.b)')'

รูปที่ 2.30 สัญลักษณ์และสมการบูลีนของวงจรเปรียบเทียบทีละบิต

ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบทีละบิต

รูปที่ 2.31 แสดงการบรรยายการเชื่อมต่อของวงจรเปรียบเทียบทีละบิตตามสัญลักษณ์ใน รูปที่ 2.31 เครื่องหมาย "--" ใช้แทนข้อความอธิบาย (Comment)

```
ENTITY bit_comparator IS
    PORT ( a , b ,              -- data inputs
           gt ,                 -- previous greater than
           eq ,                 -- previous equal
           lt  : IN BIT;        -- previous less than
         a_gt_b ,               -- greater
         a_eq_b ,               -- equal
         a_lt_b  : OUT BIT);    -- less than
END bit_comparator;
```

รูปที่ 2.31 การบรรยายการเชื่อมต่อของโปรแกรมเปรียบเทียบทีละบิท

รูปที่ 2.32 แสดงการบรรยายการทำงานภายในวงจรเปรียบเทียบ โดยกำหนดชื่อการบรรยายเป็น gate_level ภายในมีการกำหนดรายการเกทพื้นฐานที่ต้องใช้เอาไว้ 3 ชนิด และกำหนดชื่อเป็น n1,n2,n3 โดยกำหนดให้ n1 อ้างอิงถึง อินเวอร์เตอร์เกท n2 อ้างอิงถึง แนนด์เกท 2 อินพุท และ n3 อ้างอิงถึงแนนด์เกท 3 อินพุท คำสั่ง FOR ALL : n1 เป็นการกำหนดให้อุปกรณ์ทุกตัวที่ขึ้นต้นชื่อด้วย n1 ให้อ้างอิงกับอินเวอร์เตอร์เกท ในกรณีของ n2 และ n3 ก็เช่นเดียวกัน WORK เป็นการกำหนดตำแหน่งที่อยู่หรือไลบรารีที่เก็บอุปกรณ์ที่อ้างอิงถึง ซึ่งอาจจะเป็นชื่อไดเร็คทอรีใดๆ (กรณีที่ใช้คำ WORK จะหมายถึงไดเร็คทอรีปัจจุบัน) รูปที่ 2.33 แสดงองค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิตตามสมการบูลีนโดยใช้เกทพื้นฐานที่ได้ออกแบบไว้แล้ว

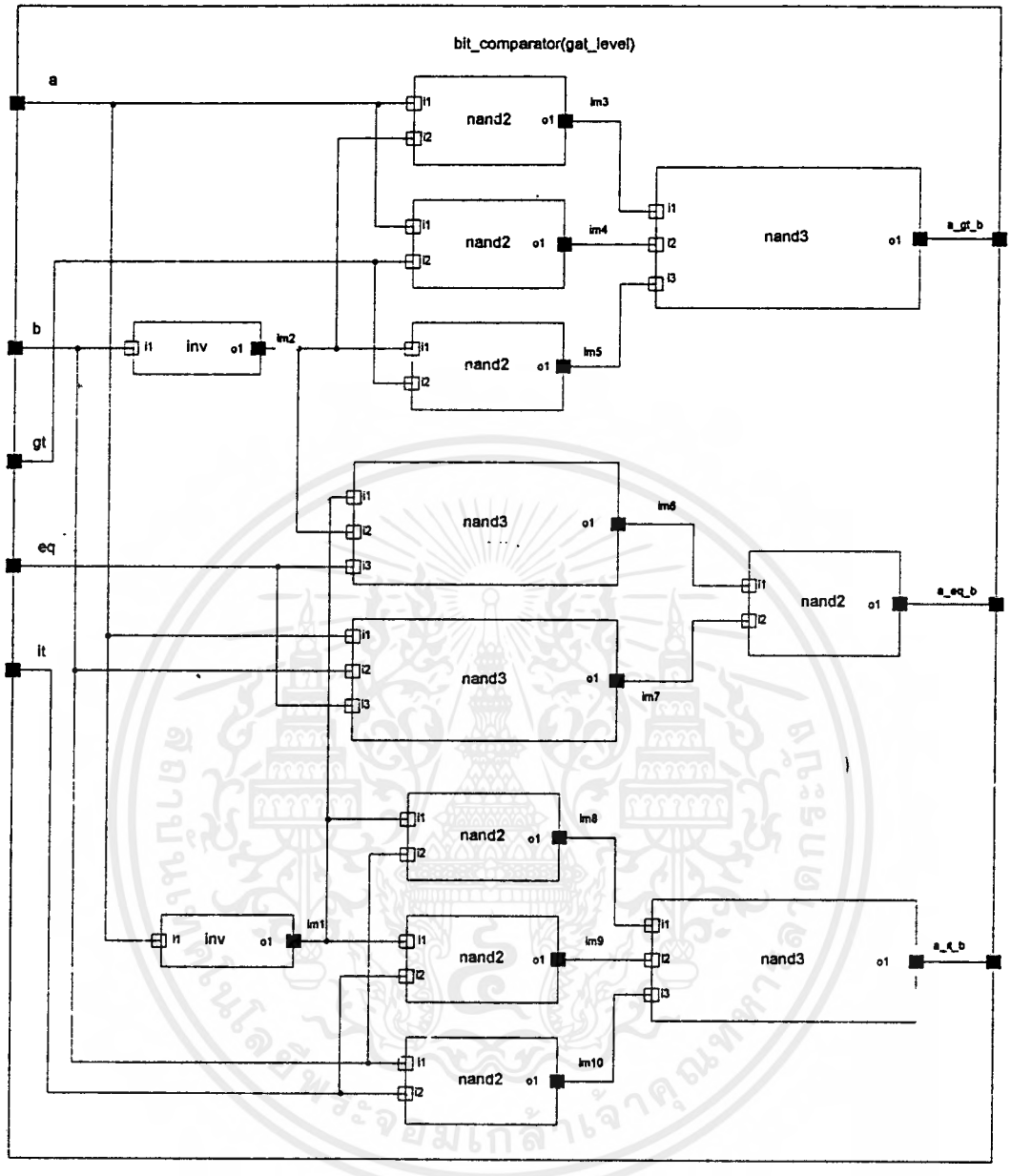
```
ARCHITECTURE gate_level OF bit_comparator IS
  COMPONENT n1 PORT (I1 : IN BIT; o1 : OUT BIT); END COMPONENT;
  COMPONENT n1 PORT (I1, I2 : IN BIT; o1 : OUT BIT); END COMPONENT;
  COMPONENT n1 PORT (I1, I2, I3 : IN BIT; o1 : OUT BIT); END COMPONENT;

  FOR ALL : n1 USE ENTITY WORK.inv (single_delay);
  FOR ALL : n2 USE ENTITY WORK.nand2 (single_delay);
  FOR ALL : n3 USE ENTITY WORK.nand3 (single_delay);
  SIGNAL im1, im2, im3, im4, im5, im6, im7, im8, im9, im10 : BIT;
    BEGIN
    -- a_gt_b output
        g0 : n1 PORT MAP (a, im1);
        g1 : n1 PORT MAP (b, im2);
        g2 : n2 PORT MAP (a, im2, im3);
        g3 : n2 PORT MAP (a, gt, im4);
        g4 : n2 PORT MAP (im2, gt, im5);
        g5 : n3 PORT MAP (im3, im4, im5, a_gt_b);

    -- a_eq_b output
        g6 : n3 PORT MAP (im1, im2, eq, im6);
        g7 : n3 PORT MAP (a, b, eq, im7);
        g8 : n2 PORT MAP (im6, im7, a_eq_b);

    -- a_lt_b output
        g9 : n2 PORT MAP (im1, b, im8);
        g10 : n2 PORT MAP (im1, lt, im9);
        g11 : n2 PORT MAP (b, lt, im10);
        g12 : n3 PORT MAP (im8, im8, im10, a_lt_b);
  END gate_level;
```

รูปที่ 2.32 การบรรยายการทำงานภายในวงจรเปรียบเทียบทีละบิต



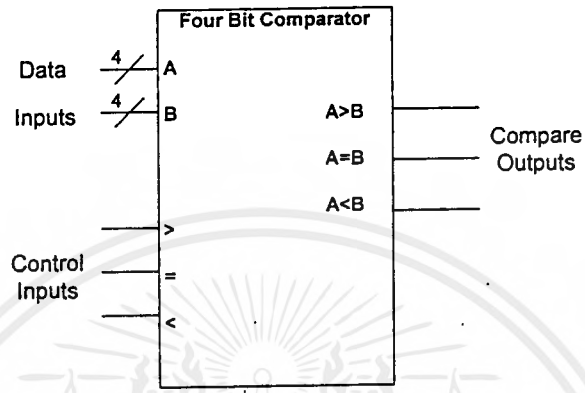
รูปที่ 2.33 องค์ประกอบและการเชื่อมต่อภายในวงจรเปรียบเทียบทีละบิต

2.5.3 การเชื่อมต่ออุปกรณ์ชนิดเดียวกัน

ในการเชื่อมต่ออุปกรณ์ต่างๆ เข้าด้วยกันจะต้องอ้างอิงถึงอุปกรณ์ทีละตัว แต่หากเป็นการเชื่อมต่ออุปกรณ์ชนิดเดียวกันจำนวนหลายตัวเข้าเดียวกันแล้วภาษา VHDL ได้จัดเตรียมเครื่องมือซึ่งเปรียบเสมือนเส้นทางลัดในการบรรยาย โดยไม่จำเป็นต้องอ้างถึงอุปกรณ์ทุกตัว ในหัวข้อต่อไปจะได้ยกตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิตที่เรียกว่า nibble_comparator ซึ่งประกอบด้วยวงจรเปรียบเทียบทีละบิตที่ได้ออกแบบไว้ในข้อก่อนหน้านี้จำนวน 4 ตัวต่อเข้าด้วยกัน

ตัวอย่างการออกแบบวงจรเปรียบเทียบขนาด 4 บิต

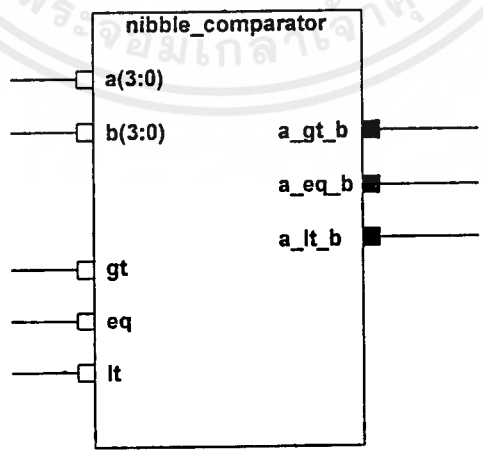
วงจรเปรียบเทียบขนาด 4 บิตมีสัญญาณข้อมูลขนาด 4 บิตจำนวน 2 อินพุท สัญญาณควบคุม 3 อินพุท และสัญญาณผลเปรียบเทียบ 3 เอาท์พุท ดังรูปที่ 2.34



รูปที่ 2.34 วงจรเปรียบเทียบขนาด 4 บิต

ตัวอย่างการบรรยาย VHDL ของวงจรเปรียบเทียบขนาด 4 บิต

รูปที่ 2.35 แสดงสัญลักษณ์ของวงจร nibble_comparator และการบรรยายการเชื่อมต่อของวงจร อินพุท a และ b เป็นอินพุทสำหรับข้อมูลขนาด 4 บิต เราสามารถกำหนดชนิดของอินพุทเป็น BIT_VECTOR ซึ่งเป็นอาเรย์ของ BIT ได้ด้วย



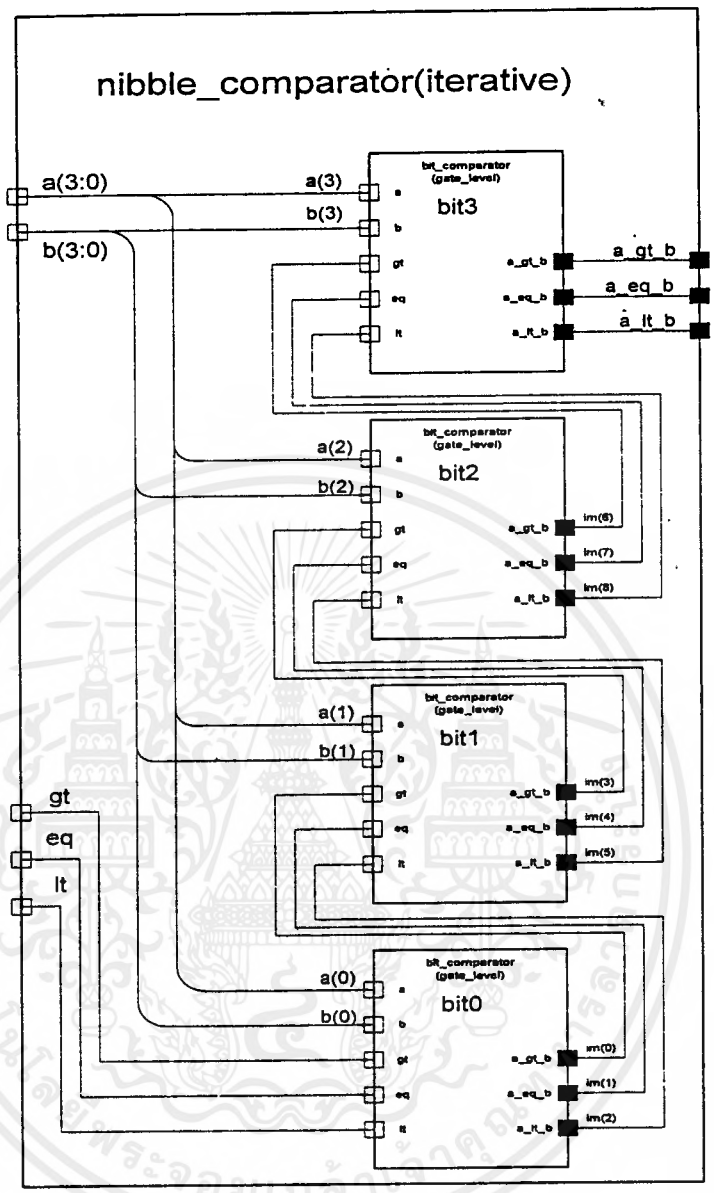
รูปที่ 2.35 (ก) สัญลักษณ์ของวงจร nibble_comparator

```
ENTITY nibble_comparator IS
    PORT (a, b, IN BIT_VECTOR (3 DOWNTO 0); -- data inputs
                gt ,                    -- previous greater than
                eq ,                    -- previous equal
                lt : IN BIT;            -- previous less than
          a_gt_b,                       -- a greater than b
          a_eq_b,                       -- a equal b
          a_lt_b : OUT BIT);            -- a less than b
END nibble_comparator;
```

รูปที่ 2.35 (ข) การบรรยายการเชื่อมต่อของวงจร nibble_comparator

```
ARCHITECTURE iterative OF nibble_comparator IS
  COMPONENT comp1 `
    PORT (a, b, gt, eq, lt : IN BIT ; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  FOR ALL   : comp1 USE ENTITY WORK.bit_comparator (gate_level);
  SIGNAL im : BIT_VECTOR (0 TO 8);
BEGIN
  c0    : comp1 PORT MAP (a(0), b(0), gt, eq, lt, im(0), im(1), im(2));
  c1to2 : FOR i IN 1 TO 2 GENERATE
  c     : comp1 PORT MAP (a(i), b(i), im(i*3-3), im(i*3-2), im(i*3-1),
                          im(i*3+0), im(i*3+1), im(i*3+2));
  END GENERATE;
  c3    : comp1 PORT MAP (a(3), b(3), im(6), im(7), im(8),
                          a_gt_b, a_eq_b, a_lt_b);
END iterative;
```

รูปที่ 2.36 การบรรยายการทำงานของวงจร



รูปที่ 2.37 ลักษณะของวงจรแบบสัญลักษณ์

รูปที่ 2.36 แสดงการบรรยายการทำงานของวงจรโดยใช้ comp1ซึ่งอ้างอิงถึงวงจรเปรียบเทียบทีละบิต ค่า n เป็นค่าคงที่ที่ใช้กำหนดจำนวนของวงจร comp1 ซึ่งในกรณีนี้เป็นวงจรเปรียบเทียบขนาด 4 บิต ดังนั้นจึงกำหนดค่า n เป็น 4 การบรรยายได้ใช้ FOR loop และ GENERATE ในการกำหนดการเชื่อมต่อของอุปกรณ์ชนิดเดียวกัน สังเกตว่าการบรรยายในลักษณะจะช่วยประหยัดเวลาและสะดวกมากขึ้น ทั้งนี้เราสามารถกำหนดจำนวนอุปกรณ์ที่มาเชื่อมต่อกันจำนวนเท่าใดก็ได้โดยไม่ต้องเปลี่ยนรูปแบบใดๆ เลยนอกจากค่าของ n เท่านั้น ในรูปที่2.37 แสดงลักษณะของวงจรแบบสัญลักษณ์

2.6 โครงสร้างภายในของอุปกรณ์ FPGAs

FPGAs จัดเป็นวงจรรวมเฉพาะกิจชนิดหนึ่งที่สามารถโปรแกรมเป็นวงจรเชิงเลขใดๆ ก็ได้ เช่นเดียวกับ EPLD ต่างกันที่ EPLD โปรแกรมลงบน EPROM ภายใน และสามารถโปรแกรมใหม่ได้หลังจากนำไปลบด้วยแสง UV แต่ FPGAs โปรแกรมลงบนสแตติกแรมภายในด้วยข้อมูลที่อยู่ภายนอก และสามารถโปรแกรมใหม่ได้โดยการรีเซตด้วยสัญญาณไฟฟ้านอกจากนั้น FPGAs ยังประหยัดไฟและมีความจุวงจรสูง (จำนวนเกตมาก) ได้อีกด้วย

วงจรรวมชนิดที่ใช้ในโครงงานนี้ผลิตโดยบริษัทไซลิงค์ (Xilinx) ซึ่งเป็นบริษัทที่ทำการค้นคว้าร่วมกับบริษัทเอ็มเอ็มไอ (MMI) สร้างเป็นกลุ่มของเกทจำนวน 600-25,000 เกท ดังแสดงในตารางที่ 2.1 การที่ต้องบอกขนาดของวงจรรวมเป็นจำนวนเกตเพราะจะได้รู้ว่าขนาดของวงจรที่ได้ออกแบบไว้สามารถโปรแกรมลงบนวงจรรวม FPGAs ได้หรือไม่

FPGAs	Appr.Gate Count	Max I/Os	Flip-Flops	RAM bits	Available CLBs
XC2064	1,000	58	122	0	64
XC2018	1,500	74	174	0	100
XC3020/3120	1,800	64	256	0	64
XC3030/3130	2,700	80	360	0	100
XC3042/3142	3,700	96	480	0	144
XC3064/3164	5,500	120	688	0	244
XC3090/3190	7,500	144	928	0	320
XC3195	9,000	176	1,320	0	484
XC4002A	2,000	64	256	2,048	64
XC4003/4003A	3,000	80	360	3,200	100
XC4003H	3,000	160	200	3,200	100
XC4004A	4,000	960	480	4,608	144
XC4005/4005A	5,000	122	616	6,072	196
XC4005H	5,000	192	392	6,272	196

ตารางที่ 2.1 คุณสมบัติของ FPGAs ตระกูลต่างๆ

FPGAs	Appr.Gate Count	Max I/Os	Flip-Flops	RAM bits	Available CLBs
XC4006	6,000	128	768	8,192	256
XC4008	8,000	144	936	10,368	324
XC4010	10,000	160	1,120	12,800	400
XC4013	13,000	192	1,536	18,432	576
XC4025	25,000	256	2,560	32,768	1,024

ตารางที่ 2.1 คุณสมบัติของ FPGAs ตระกูลต่างๆ (ต่อ)

NAIINR2 หมายถึงเกต NAND2 หรือเกต NOR2

Gate	Equipvalent gate count	Gate	Equipvalent gate count
INV	1	RS Latch	3
NAIINR2	1	D Latch	4
NAIINR3	2	D Latch with CLR	5
NAIINR4	2	D Latch with PRE	5
NAIINR6	5	D Latch with PRE/CLR	6
NAIINR8	6	D F/F	6
NAIINR9	7	D F/F with CLR	7
NAIINR12	8	D F/F with PRE	7
NAIINR16	11	D F/F with PRE/CL	8
BUFF	2	JK F/F with CLR	9
ANIIOR2	2	JK F/F with PRE	12
ANIIOR3	2	JK F/F with PRE/CL	13
ANIIOR4	3	T F/F with CLR	8
XOR2	3	T F/F with PRE	8
XNOR2	3	T F/F with PRE/CL	9

ตารางที่ 2.2 ตารางประมาณการนับเกตของเกตพื้นฐาน

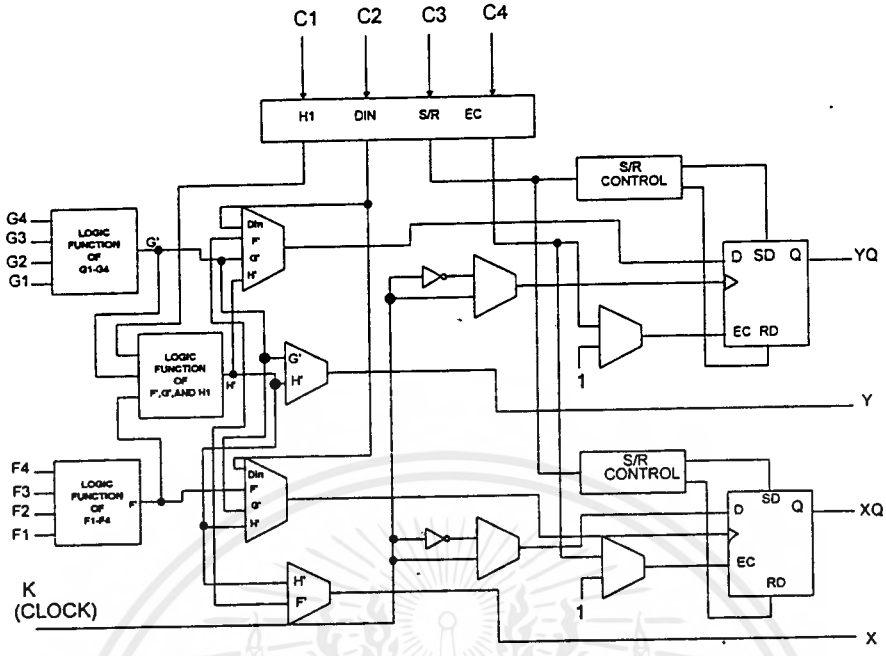
FPGAs มีโครงสร้างภายในใกล้เคียงกับสถาปัตยกรรมของเกตอาเรย์ (GAL, Gate Array Logic) มาก สามารถโปรแกรมและลบคอนฟิกกูเรชัน (Configuration) ภายในสแตติกแรม (Static RAM) ได้โดยใช้กระแสไฟฟ้า ซึ่งทำการโปรแกรมได้โดยดึงข้อมูลฐานสิบหกมาจากภายนอก เช่น Parallel EPROM หรือ Serial PROM ต่างกับ EPLD, PAL ที่มี EPROM อยู่ในตัว ภายใน FPGAs จะจัดเรียงเป็นลอจิกเซลล์ล้อมรอบภายนอกด้วยอินพุทเอาท์พุทเซล FPGAs ตัวแรกที่ผลิตโดยบริษัทไซลิงค์คือเบอร์ XC2064 (2000 Family) ประกอบด้วยเซลเรียงกันเป็นเมตริกซ์ (Matrix) เป็นจำนวน 64 เซล หลังจากนั้นผลิต FPGAs ตระกูล 3000 และ 4000 ซึ่งมีโครงสร้างซับซ้อนขึ้นสามารถเพิ่มจำนวนเกตได้มากขึ้นและดีขึ้น แต่ละเซลเรียกว่า CLB (Configurable Logic Block)

2.7 ส่วนที่เป็นองค์ประกอบของลอจิก (Configurable Logic Block)

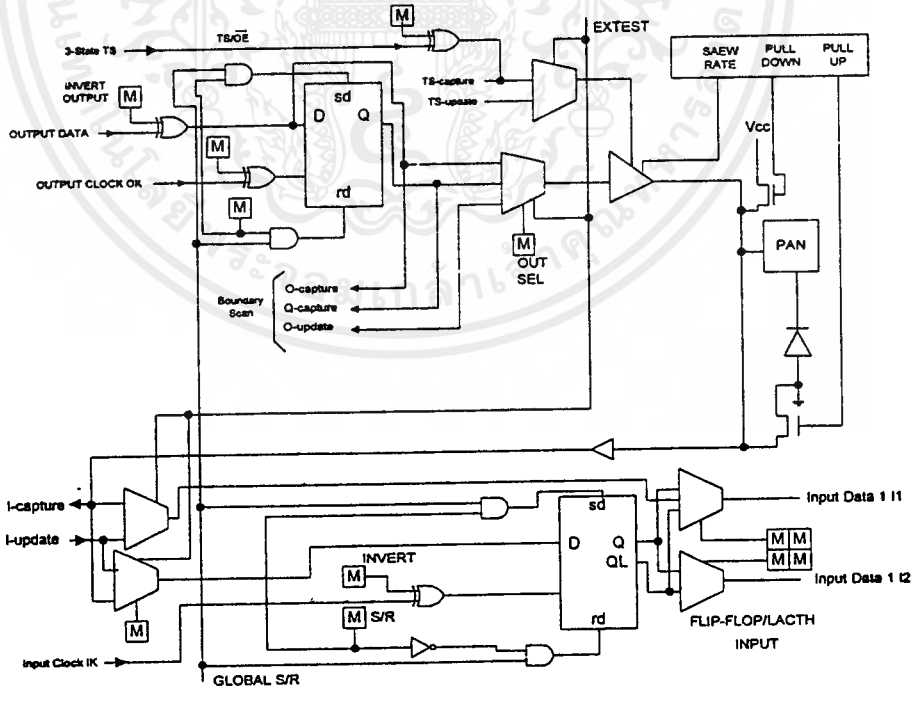
CLB จะจัดเรียงกันเป็นแบบเมตริกซ์แบบอาเรย์ขนาด M*N การออกแบบนั้นสามารถทำได้โดยการจัดการวาง CLB และต่อเชื่อมขาของ CLB ให้ต่อถึงกัน เราสามารถจัด CLB ให้เชื่อมต่อถึงกันได้โดยการทำด้วยมือหรือให้โปรแกรมที่สนับสนุน FPGAs ทำให้โดยอัตโนมัติ โดยวิธีของมันเองสำหรับไฟล์ที่ได้จากโปรแกรมเหล่านี้ เราเรียกว่าไฟล์ที่กำหนดการวางอุปกรณ์ (Configuration File) ซึ่งจะบรรจุโครงร่างภายในของ CLB ตามความเหมาะสม อีกด้านหนึ่งไฟล์กำหนดการวางอุปกรณ์นั้นจะเป็นไฟล์กระแสข้อมูล (Bit Stream) ซึ่งสามารถใช้โปรแกรมหน่วยความจำภายในของ FPGAs ได้ สำหรับรูปที่ 2.38 แสดง CLB ของ FPGAs ตระกูล 4000

2.8 ส่วนอินพุทและเอาท์พุทของอุปกรณ์ FPGAs

รอบนอกของ FPGAs จะประกอบด้วย IOBs ประมาณ 64 ถึง 144 ตัว ซึ่งขึ้นอยู่กับตระกูลของ FPGAsซึ่งIOBs จะเป็นตัวเชื่อมต่อระหว่างภายในกับภายนอกของวงจรลอจิกของ FPGAs ลักษณะของ IOBs จะมีลักษณะ 2 ทิศทาง สามารถโปรแกรมให้เป็นอินพุทหรือเอาท์พุทก็ได้ สำหรับรูปที่ 2.39 แสดง IOBs ของ FPGAs ตระกูล 4000



รูปที่ 2.38 แผนผัง CLB ของตระกูล 4000



รูปที่ 2.39 แผนผัง IOBs ของตระกูล 4000

2.9 รายละเอียดการใช้งานของอุปกรณ์ FPGAs

FPGAs สามารถทำงานได้หลายลักษณะ โดยกำหนดได้ที่ขาสัญญาณ M0 M1 M2 ดังแสดงในตารางที่ 2.3 ในลักษณะมาสเตอร์พาราเรล (Master Parallel Mode) รับโปรแกรมคอนฟิกทีละ 1ไบท์ (Byte) จากหน่วยความจำภายนอกที่เป็นแบบขนาน โดยสามารถรับโปรแกรมคอนฟิก (Config) จากแอดเดรส (Address) ต่ำหรือสูงก่อนก็ได้ การต่อลักษณะเพอริเฟอรัล (Peripheral) จะ รับโปรแกรมคอนฟิกทีละ 1ไบท์จากไมโครโปรเซสเซอร์ โดยสามารถโต้ตอบกันได้ว่าพร้อมหรือไม่ที่จะรับข้อมูลต่อไป การต่อลักษณะสเลฟซีเรียล (Slave Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิท (Bit) จากไมโครโปรเซสเซอร์ตามสัญญาณอินพุท CCLK ส่วนการต่อลักษณะมาสเตอร์ซีเรียล (Master Serial) จะรับโปรแกรมคอนฟิกทีละ 1 บิทจากหน่วยความจำภายนอกที่เป็นแบบอนุกรม

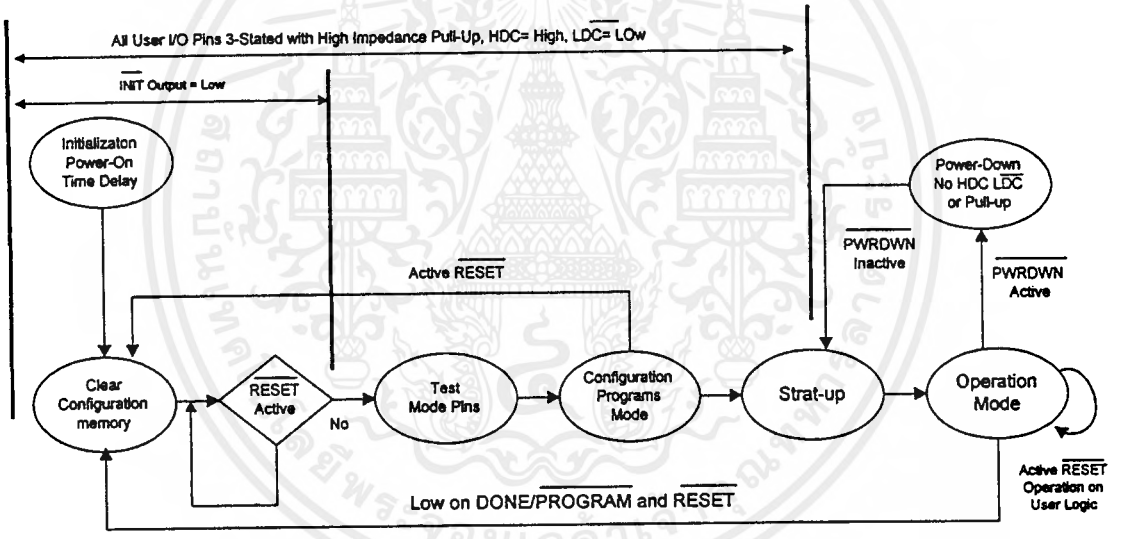
Mode	M2	M1	M0	CCLK	Data
Master Serial	0	0	0	output	Bit-Serial
Slave Serial	1	1	1	input	Bit-Serial
Master Parallel up	1	0	0	output	Byte-Wide,00000 up
Master Parallel down	1	1	0	output	Byte-Wide,3FFFF down
Peripheral Synchr.	0	1	1	input	Byte-Wide
Peripheral Asynchr.	1	0	1	output	Byte-Wide
Reserved	0	1	0	---	---
Reserved	0	0	1	---	---

ตารางที่ 2.3 โหมดต่างๆ ของการคอนฟิกกูเรชัน

จากความต้องการสร้างให้ใช้กระแสไฟฟ้าต่ำจากลักษณะการต่อใช้งานทั้ง 5 แบบจึงมีเพียง 2 แบบเท่านั้นที่เหมาะสม คือ แบบมาสเตอร์ซีเรียลและแบบสเลฟซีเรียล ส่วนแบบมาสเตอร์พาราแรลต้องใช้ EPROM 27CXXX ซึ่งกินกระแสมากกว่า PROM XC17XXX เหมาะในการทดสอบต้นแบบก่อน เมื่อวงจรต้นแบบทำงานได้ถูกต้องแล้วจึงทำการอัดโปรแกรมลง PROM

อีกทีหนึ่งเพราะว่าในแบบพาราแรลนั้น EPROM สามารถโปรแกรมได้ใหม่ต่างกับ PROM ที่โปรแกรมเพียงได้ครั้งเดียว

การใช้งาน FPGAs ในการต่อลักษณะสเลฟซีเรียลและมาสเตอร์ซีเรียล เมื่อเริ่มจ่ายไฟเข้าตัว FPGAs จะทำการลบข้อมูลหน่วยความจำที่ใช้ในคอนฟิก (Configuration Memory) ตรวจสอบลักษณะการคอนฟิกว่าเป็นลักษณะใดในตารางที่ 2.3 ว่าเป็นแบบอนุกรมหรือขนาน หลังจากนั้นจะเริ่มทำการโปรแกรมคอนฟิกสัญญาณ DONE/PROGRAM เป็น "0" ซึ่งอยู่ในระหว่างโปรแกรมและเมื่อข้อมูลในคอนฟิกที่รับมาจากภายนอกเต็มหน่วยความจำที่ใช้ในการคอนฟิก และความยาวของข้อมูลตรงกับที่ส่วนหัวของข้อมูลคอนฟิก สัญญาณ DONE/PROGRAMจะเป็น "1" ซึ่งหมายถึงโปรแกรมทำการคอนฟิกเสร็จสิ้น ดูรูป 2.40 ประกอบ



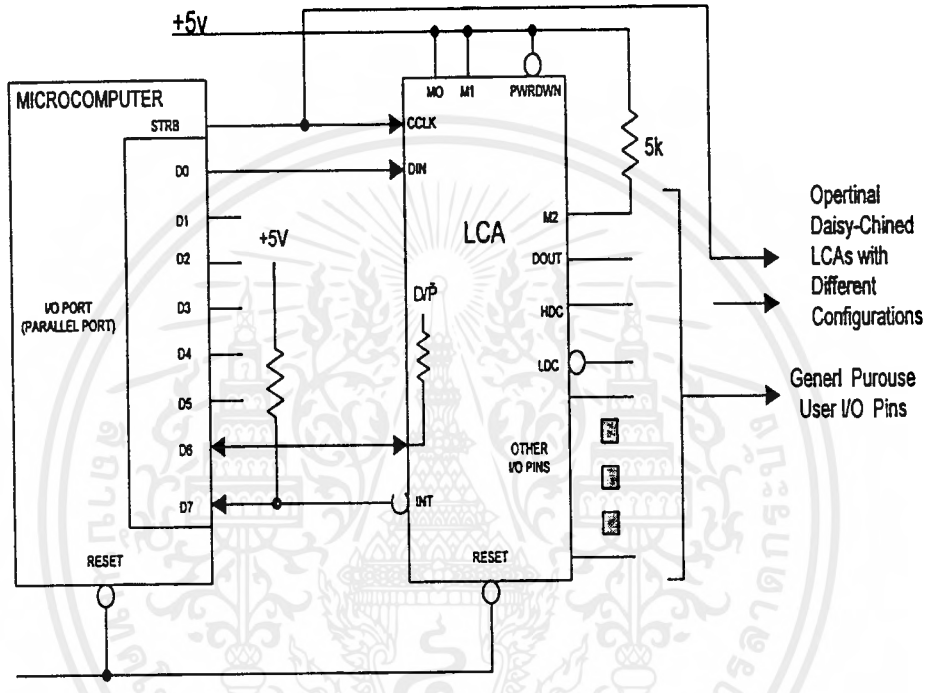
รูปที่ 2.40 ลำดับไดอะแกรมในการคอนฟิก

เมื่อเริ่มป้อนแหล่งจ่ายไฟเข้าไอซีและการโปรแกรมใหม่

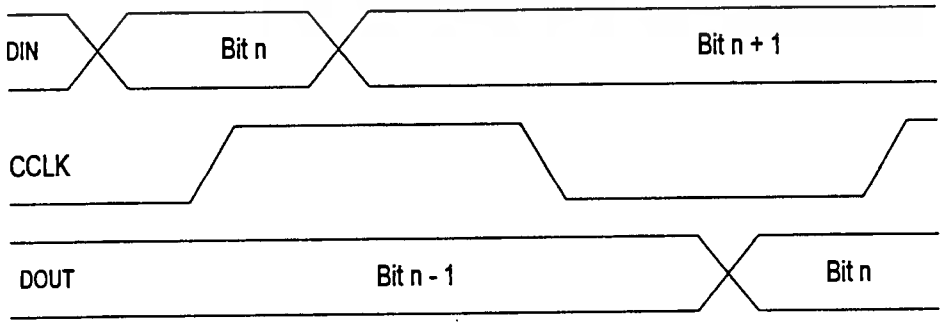
2.9.1 การใช้งานในลักษณะสเลฟซีเรียล

การต่อใช้งานในลักษณะนี้ เหมาะสมกับวงจรที่ออกแบบมาเพื่อทำงานร่วมกับไมโครคอมพิวเตอร์อยู่แล้ว ทั้งนี้เพราะ FPGAs ได้ใช้ความสามารถของไมโครคอมพิวเตอร์ในการเก็บและส่งข้อมูลคอนฟิกให้ เพียงแต่ต้องเขียนโปรแกรมเพื่อส่งโปรแกรมคอนฟิกให้เพิ่มลักษณะการต่อในลักษณะนี้เป็นดังรูปที่ 2.41 ซึ่งไมโครคอมพิวเตอร์จะสร้างสัญญาณเพื่อทำการคอนฟิกให้กับ

อุปกรณ์ FPGAs การป้อนโปรแกรมคอนฟิกให้ FPGAs ทำได้โดยต่อสัญญาณ Strobe เข้ากับขา CCLK และพอร์ต D0 เข้ากับขา DIN สร้างสัญญาณคล็อกป้อนที่ขา CCLK และป้อนโปรแกรมคอนฟิกแบบอนุกรมเข้าที่ขา DIN ดังแผนภูมิในรูปที่ 2.42



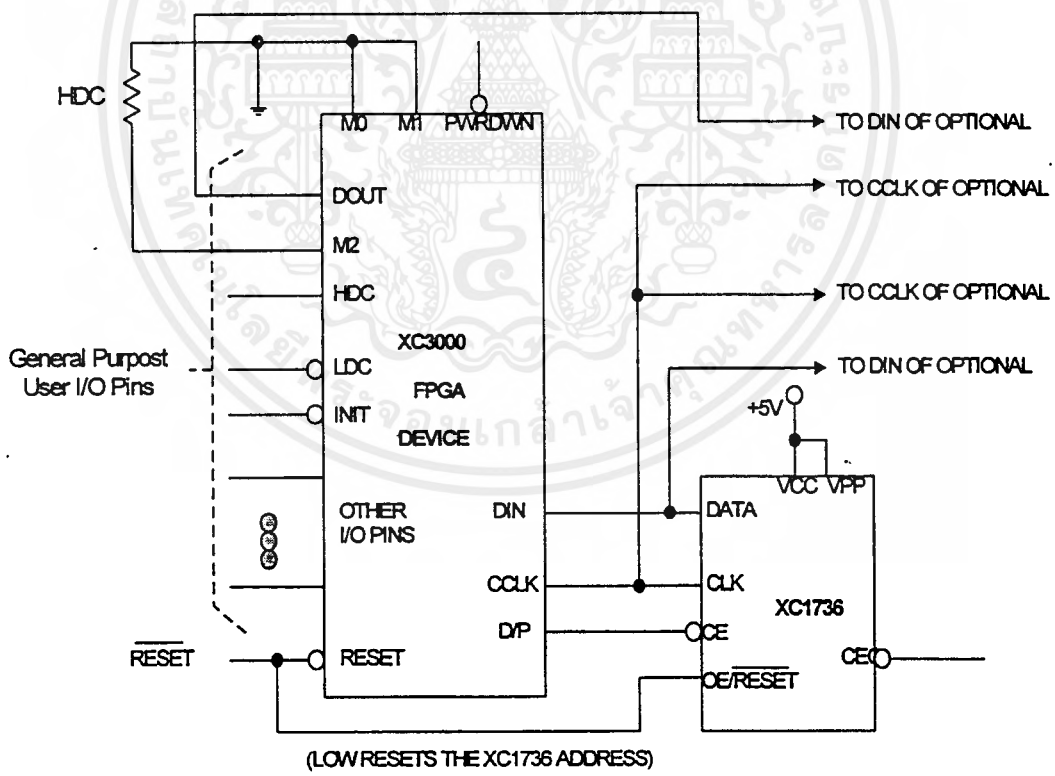
รูปที่ 2.41 แสดงการต่อใช้งานในลักษณะสเลฟซีเรียล



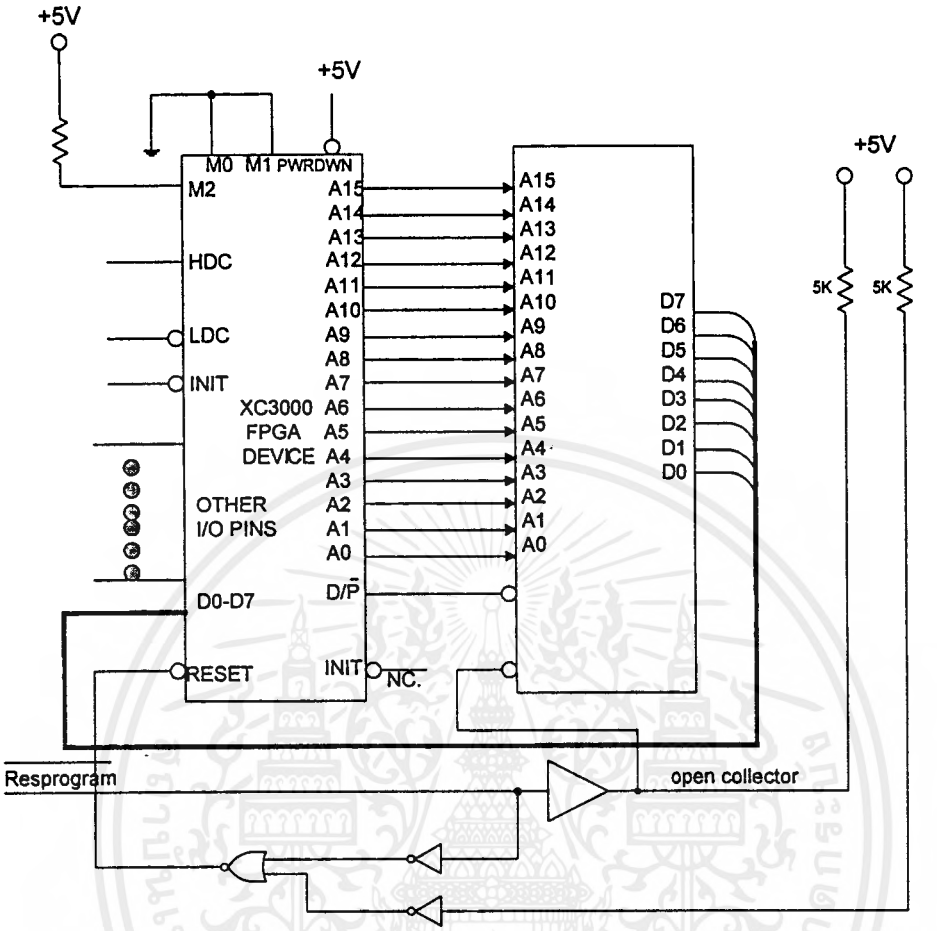
รูปที่ 2.42 แผนภูมิเวลาการป้อนข้อมูลโปรแกรมคอนฟิกในลักษณะสเลฟซีเรียล

2.9.2 การใช้งานในลักษณะมาสเตอร์ซีเรียล

การต่อใช้งานในลักษณะนี้ส่วนที่เก็บโปรแกรมคอนฟิกจะต่างจากการต่อลักษณะแรกคือใช้ PROM เบอร์ XC17XXX เป็นตัวเก็บโปรแกรม ทำให้ไม่ต้องเสียเวลาเขียนโปรแกรมเพื่อทำการคอนฟิก ซึ่งวิธีการอัดโปรแกรมคอนฟิกลง PROM ทำตามขั้นตอนดังนี้คือ เมคบิท (MakeBits) สร้างไฟล์ .BIT จากวงจรที่ออกแบบ และใช้โปรแกรม MakePROM สร้าง Hex ไฟล์แล้วทำการอัดโปรแกรมลง PROM ด้วยอุปกรณ์อัด PROM ที่มาพร้อมกับตัวโปรแกรมของไซลิงค์ PROM XC17XXX จะส่งสัญญาณเพื่อทำการคอนฟิกให้กับอุปกรณ์ FPGAs ดังแสดงในรูปที่ 2.43 D0-D7 เป็นขารับข้อมูลที่ใช้ในการคอนฟิกแบบขนาน A0-A15 เป็นแอดเดรสที่ FPGAs สร้างให้กับ EPROM เพื่ออ่านข้อมูลจากหน่วยความจำมาเก็บไว้ในสแตติกแรม (StaticRAM) แอดเดรสทั้ง 16 เส้นไม่จำเป็นต้องต่อให้ครบก็ได้ ขึ้นอยู่กับขนาดหน่วยความจำ EPROM ที่ใช้ และสามารถกำหนดให้นับขึ้นหรือลงได้



รูปที่ 2.43 การต่อใช้งานในลักษณะมาสเตอร์ซีเรียล



รูปที่ 2.44 แสดงการต่อใช้งานในลักษณะมาสเตอร์พาราแรล

2.10 ข้อควรระวังในการใช้อุปกรณ์ FPGAs

สิ่งที่สำคัญ คืออุปกรณ์ FPGAs ไวต่อความร้อนมาก การบัดกรีโดยหัวแร้งกำลังสูงหรือบัดกรีโดยจี้หัวแร้งที่ขาไอซีเป็นเวลานานจะทำให้ไอซีเสียหายได้ ระยะเวลาในการบัดกรีหนึ่งจุดไม่ควร เกิน 5-10 วินาที ควรใช้ซ็อกเกต (Socket) ไอซีในการประกอบวงจรลงแผ่นวงจรพิมพ์

การป้องกันไอซีจากแรงดันไฟฟ้าไม่ควรต่อสลับขั้วบวกกับขั้วลบจะทำให้ไอซีเสียได้ นอกจากนั้นแรงดันของแหล่งไฟต้องอยู่ในช่วงที่โรงงานกำหนดมา สำหรับ FPGAs ค่าแรงดันที่ใช้งานอยู่ในช่วง Vcc = 4.75 - 5.25 V และแรงดันที่ทนได้อยู่ในช่วง -0.5 - 7 V ดังนั้นก่อนป้อนแรงดันควรตรวจเช็คให้แน่ใจก่อน

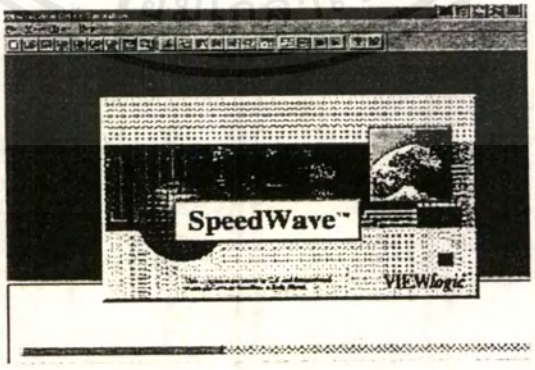
2.11 ขั้นตอนการออกแบบและจำลองการทำงานโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก (Viewlogic)

ซอฟแวร์ของบริษัทวิวลอจิกเวอร์ชัน 7.20 เป็นเวอร์ชันที่ทำงานบน Windows 95 เมื่อเริ่มต้นใช้โปรแกรมที่หน้าจอจะปรากฏรูปภาพดังรูปที่ 2.45

รูปที่ 2.45 หน้าจอเริ่มต้นของโปรแกรมวิวลอจิก

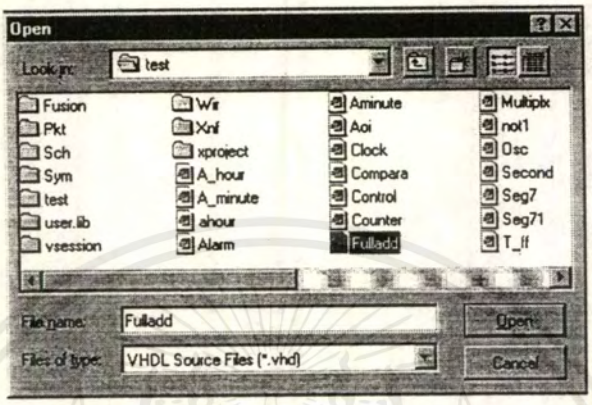
การใช้โปรแกรมสปีดเวฟ (Speed wave)

โปรแกรมสปีดเวฟ ใช้สำหรับเขียนโปรแกรมซอร์สโค้ด ด้วยภาษา VHDL โดยจะเขียนโปรแกรมภาษา VHDL เพื่อจำลองการทำงานของวงจร Fulladder เป็นตัวอย่าง ใช้ชื่อไฟล์ Fulladd.vhd คลิกเมาส์ที่ไอคอนสปีดเวฟบริเวณมุมขวาของจอภาพ หน้าจอจะปรากฏดังรูป ที่ 2.46



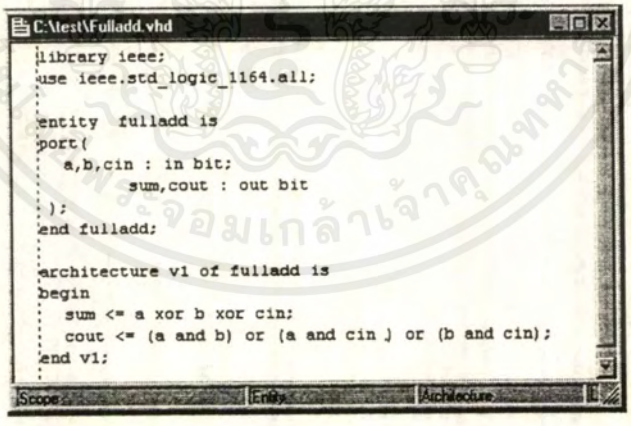
รูปที่ 2.46 ภาพเริ่มต้นของการใช้โปรแกรม Speed Wave

คลิกเมาส์ที่เมนู File และ Open เพื่อเรียกใช้โปรแกรม Fulladd.vhd ที่ได้เขียนไว้แล้วหรือ New เพื่อเขียนโปรแกรมใหม่ ดังรูปที่ 2.47



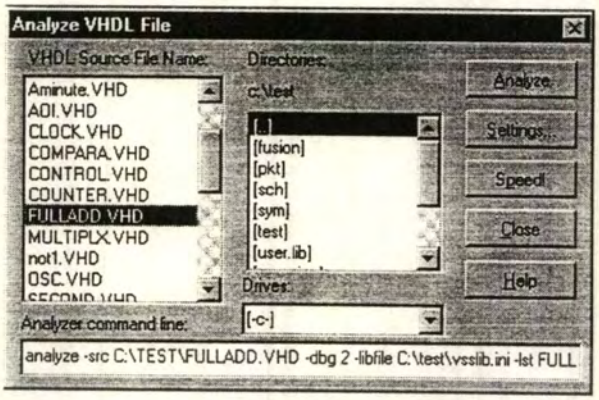
รูปที่ 2.47 หน้าจอ Open

เปิดไฟล์ Fulladd.vhd ซึ่งจะแสดงหน้าต่างของไฟล์ Fulladd.vhd ดังรูปที่ 2.48



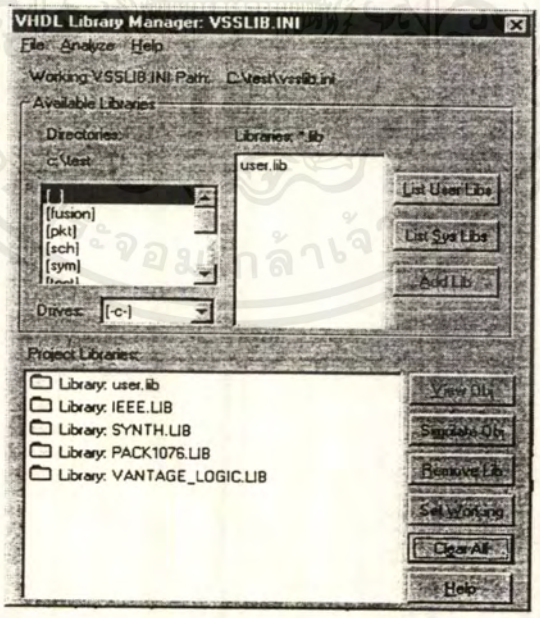
รูปที่ 2.48 หน้าต่างของไฟล์ Fulladd.vhd

เมื่อทำการเขียนโปรแกรมเรียบร้อยแล้วให้ทำการบันทึก และคลิกเมาส์ที่เมนู Tools พร้อมทั้งเลือกที่ Analyze VHDL เพื่อทำการคอมไพล์โปรแกรม Fulladd.vhd ดังรูปที่ 2.49



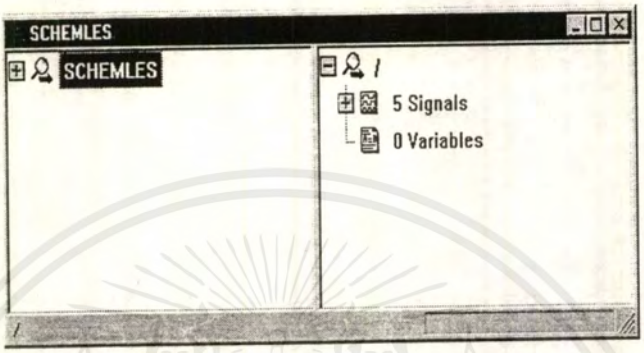
รูปที่ 2.49 หน้าต่างของ Analyze VHDL File

คลิกเมาส์ที่ Analyze เพื่อทำการคอมไพล์โปรแกรม Fulladd.vhd หากเกิด Error ให้กลับไปแก้ไขที่โปรแกรม Fulladd.vhd และทำการคอมไพล์โปรแกรมใหม่จนกว่าจะผ่าน เมื่อทำการคอมไพล์ไฟล Fulladd.vhd ผ่านแล้วให้คลิกเมาส์ที่เมนู Tools และเลือกคำสั่ง VHDL Manager ซึ่งจะแสดงหน้าต่างของ VHDL Library Manager ดังรูปที่ 2.50



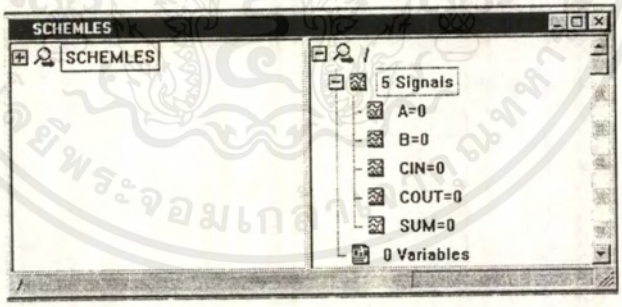
รูปที่ 2.50 หน้าต่างของ VHDL Library Manager

ดับเบิ้ลคลิกที่ไดเร็คทอรี Library : use.lib ซึ่งจะปรากฏไดเร็คทอรีย่อยของโปรแกรม *.vhd โปรแกรมต่างๆ เลือกไดเร็คทอรี Fulladd:Entity และคลิกเมาส์ที่ Simulate Obj ซึ่งจะแสดงหน้าต่างของสัญญาณต่างๆ ทั้งอินพุทและเอาท์พุทของโปรแกรม Fulladd โดยจะซ่อนอยู่หลังหน้าต่างโปรแกรม Fulladd.vhd ที่เรียกขึ้นมาก่อน ดังรูปที่ 2.51



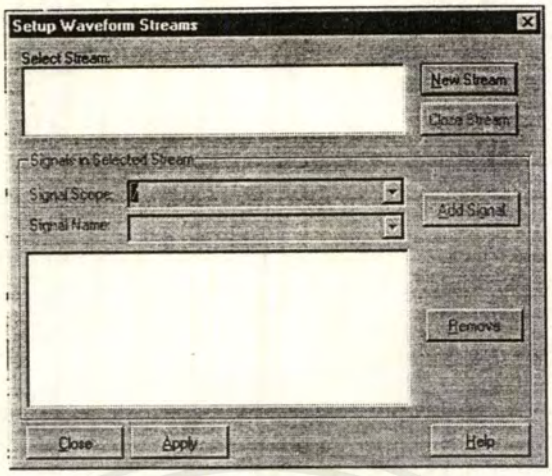
รูปที่ 2.51 หน้าต่างของ SCHEMLES

คลิกเมาท์ที่ไดเร็คทอรีทางหน้าต่างด้านขวา ซึ่งแสดงจะแสดงสัญญาณอินพุทและเอาท์พุททั้ง 5 สัญญาณของโปรแกรม Fulladd.vhd ให้เห็น ดังรูปที่ 2.52



รูปที่ 2.52 สัญญาณอินพุทและเอาท์พุทของโปรแกรม Fulladd.vhd

คลิกเมาส์ที่เมนู Setup และเรียกใช้คำสั่ง Wavefrom Streams จะขึ้นหน้าต่าง Setup Wavefrom Streams.ให้เห็น ดังรูปที่ 2.53

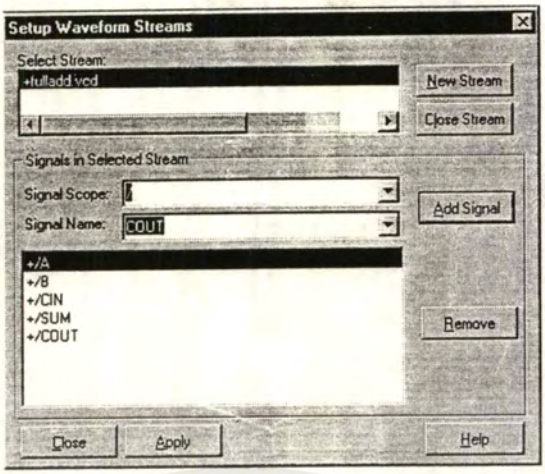


รูปที่ 2.53 หน้าต่างของเมนู Setup Wavefrom Streams

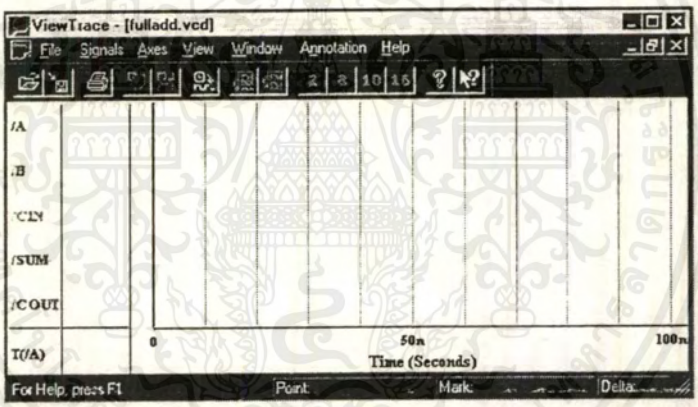
คลิกเมาส์ที่ New Streams ซึ่งจะปรากฏหน้าต่าง Create New Wavefrom Stream ดังรูปที่ 2.54

รูปที่ 2.54 หน้าต่างของ Create New Straem

ใส่ชื่อไฟล์ที่จะบันทึกลงในช่อง File Name และทำการเซฟ หลังจากนั้นใส่ชื่อสัญญาณทั้ง 5 สัญญาณในช่อง Signal Name ดังรูปที่2.55 แล้วคลิกเมาส์ที่ Add Signal จนครบทั้ง 5 สัญญาณแล้ว ให้คลิกเมาส์ที่ Apply เพื่อเรียกใช้โปรแกรมวิวเทรค (View Trace) ในการดูผลจำลองการทำงานของโปรแกรม Fulladd.vhd ดังรูปที่ 2.56

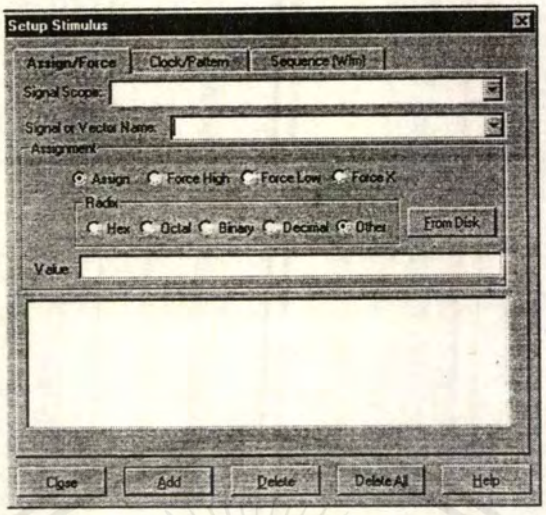


รูปที่ 2.55 การ Add Signal



รูปที่ 2.56 หน้าต่างจำลองการทำงานของ Fulladd.vhd

กลับไปที่หน้าต่างโปรแกรม SCHEMLES อีกครั้ง คลิกเมาส์ที่เมนู Setup และเลือกใช้คำสั่ง Stimulus ซึ่งจะปรากฏหน้าต่าง Setup Stimulus ขึ้นมาเพื่อใช้กำหนดค่าสัญญาณในการจำลองการทำงาน ดังรูปที่ 2.57

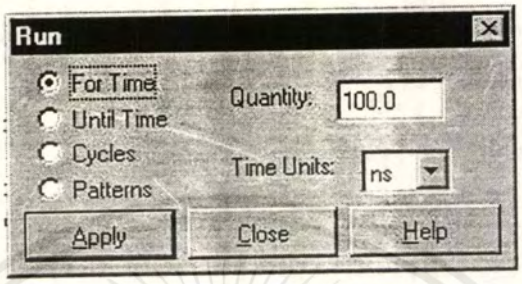


รูปที่ 2.57 หน้าต่าง Setup Stimulus

คลิกเมาส์ที่ Clock/Pattern เพื่อกำหนดค่าให้สัญญาณอินพุท ที่มีลักษณะเป็นสัญญาณนาฬิกา (Clock) โดยใส่สัญญาณอินพุทที่ Signal or Vector Name เช่น อินพุท A (/A)และคลิกที่ Clock ในกรณีที่เป็นสัญญาณนาฬิกา และที่ Binary ให้ใส่ค่า 0 1 ที่ Pattern และสัญญาณอินพุต B มีค่า 0 0 1 1 และ Cin มีค่าเป็น 0 0 0 0 1 1 1 1 ดังรูปที่ 2.58

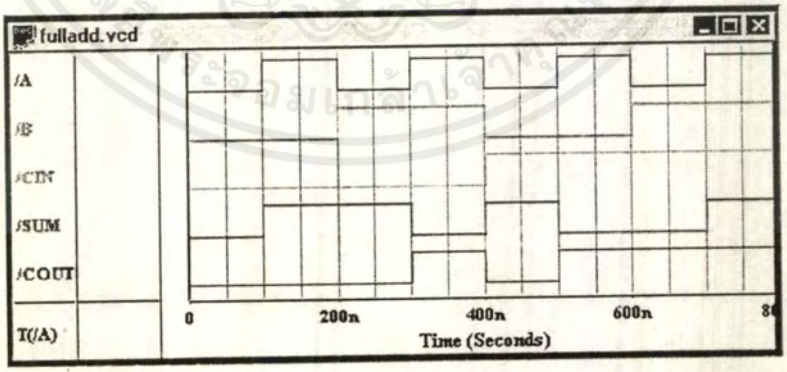
รูปที่ 2.58 การกำหนดค่าสัญญาณอินพุต A,B และ Cin

ในกรณีที่ต้องการกำหนดสัญญาณให้เป็น High หรือ Low ก็ให้คลิกเมาส์ที่ Assign/Force และเลือกที่ Force High หรือ Force Low จากรูปที่ 2.58 เมื่อทำการกำหนดค่าให้สัญญาณอินพุทเสร็จแล้ว ให้คลิกที่เมนู Simulate และเลือกคำสั่ง Run จะปรากฏหน้าต่าง Run ดังรูปที่ 2.59



รูปที่ 2.59 หน้าต่าง Run

การกำหนดช่วงเวลาการทำงาน โดยคลิกเมาส์ที่ Cycle เพื่อให้จำลองการทำงานเป็นวงรอบ หรือถ้าหากต้องการจำลองการทำงานเป็นช่วงเวลาให้คลิกที่ For Time และใส่ค่าเวลาที่ Quantity เมื่อใส่ค่าเสร็จแล้วให้คลิกที่ Apply และไปดูผลการจำลองการทำงานดังรูปที่ 2.56 โดยที่โปรแกรมวิวเทรดให้คลิกเมาส์ที่เมนู View และ เลือกคำสั่ง Time Range และคลิกที่ OK ก็จะแสดงผลการจำลองการทำงานของโปรแกรม Full Addder ทั้งหมดให้เห็นดังรูปที่ 2.60



รูปที่ 2.60 ผลการทำงานของโปรแกรม Fulladd.vhd

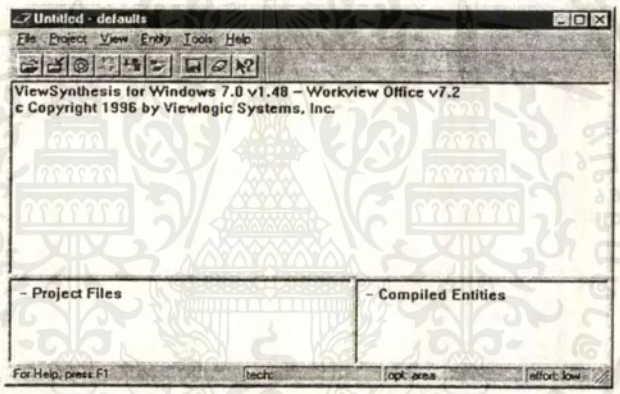
2.12 ขั้นตอนการสังเคราะห์วงจรโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก

การนำโปรแกรมภาษา VHDL มาสังเคราะห์ออกมาเป็นวงจรระดับเกทสามารถทำได้โดยการใช้โปรแกรมวิวซินทีซิท (View Synthesis) และดูรูปวงจรที่สังเคราะห์ขึ้นโดยใช้โปรแกรมวิวดรอว์ (View Draw)

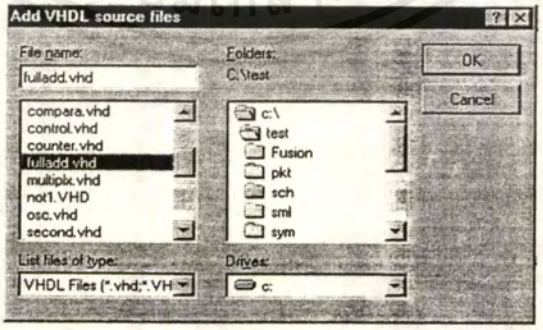
2.12.1 การใช้โปรแกรมวิวซินทีซิท (View Synthesis)

คลิกเมาส์ที่ไอคอนวิวซินทีซิทที่มุมบนด้านขวาของจอหรือเรียกจากเมนูของ Workview Office ซึ่งจะปรากฏหน้าต่าง Untitled defaults ของโปแกรมวิวซินทีซิท ดังรูปที่ 2.61

คลิกเมาส์ที่เมนู Project แล้วเลือกที่ File และ Add ซึ่งจะปรากฏหน้าต่าง Add VHDL Source Files ดังรูปที่ 2.62

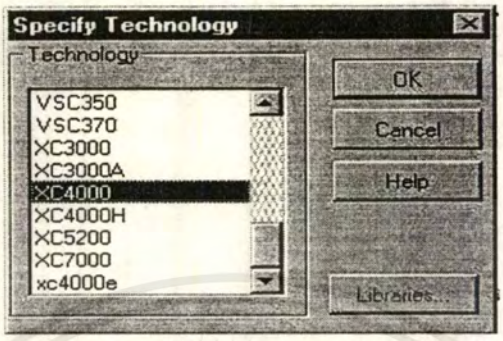


รูปที่ 2.61 หน้าต่างของโปรแกรมวิวซินทีซิท



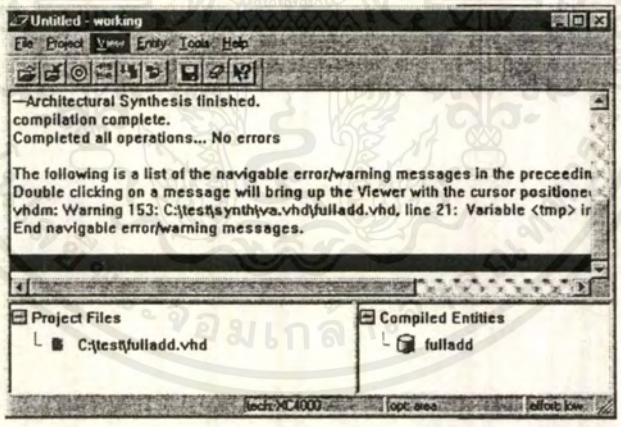
รูปที่ 2.62 หน้าต่างของ Add VHDL Source Files

เลือก Source files ชื่อ fulladd.vhd เสร็จแล้วคลิกเมาส์ที่เมนู Project และเลือก Technology ในที่นี้เลือก Technology XC4000 ดังรูปที่ 2.63



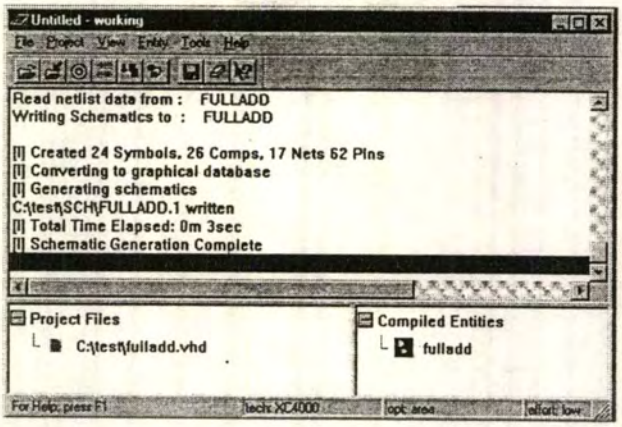
รูปที่ 2.63 หน้าต่างของ Specity Technology

คลิกเมาส์ที่เมนู Project แล้วเลือกที่ Compile ซึ่งโปรแกรมวิวซินทีซิทจะทำการคอมไพล์ไฟล์ fulladd.vhd ซึ่งจะปรากฏข้อความเมื่อทำการคอมไพล์เสร็จสิ้นดังรูปที่ 2.64



รูปที่ 2.64 ข้อความเมื่อเสร็จสิ้นการคอมไพล์ไฟล์ fulladd.vhd

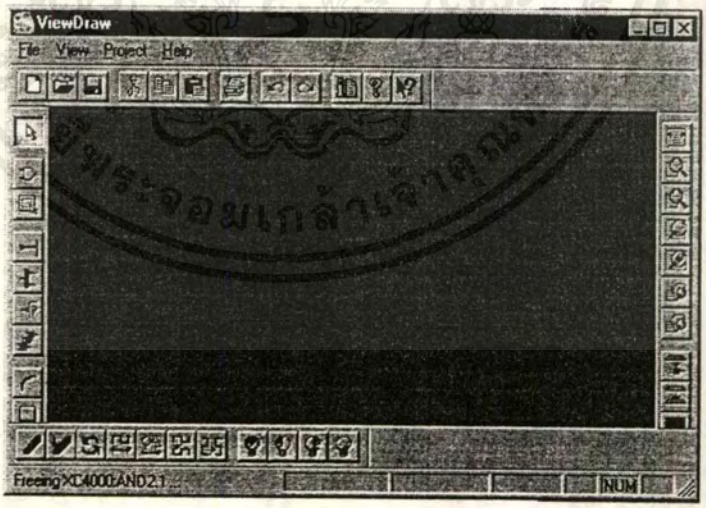
คลิกเมาส์ที่เมนู Project แล้วเลือกที่ Synthesis เพื่อทำการสังเคราะห์โปรแกรม fulladd.vhd เป็นวงจรระดับเกท เมื่อสังเคราะห์เสร็จที่ไดเร็คทอรี Compile Entities จะปรากฏรูปเกทที่ fulladd ดังรูปที่ 2.65



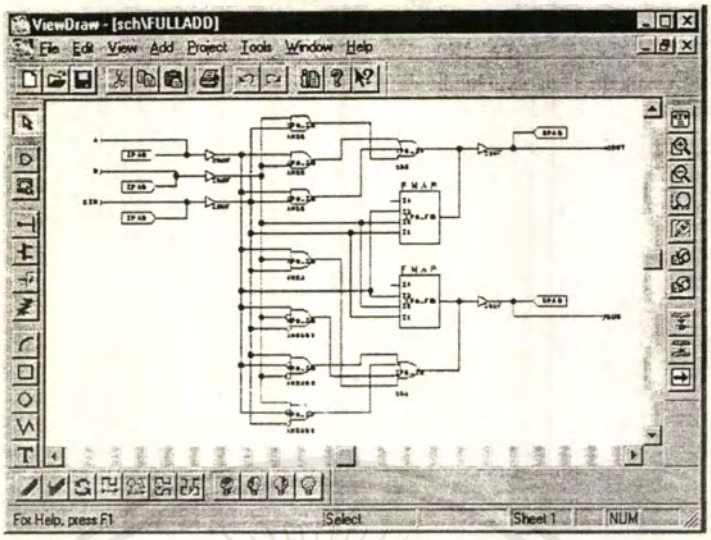
รูปที่ 2.65 ข้อความเมื่อเสร็จสิ้นการสังเคราะห์ไฟล์ fulladd.vhd

2.12.2 การใช้โปรแกรมวิวดรอว์ (View Draw)

คลิกเมาส์ที่ไอคอนวิวดรอว์ที่มุมบนด้านขวาของจอภาพ หรือเรียกจากเมนูของ Workview Office ซึ่งจะปรากฏหน้าต่างของโปรแกรมวิวดรอว์ ดังรูปที่ 2.66 เมื่อเข้าสู่โปรแกรมวิวดรอว์แล้วให้คลิกเมาส์ที่เมนู File และ Open ไฟล์ Fulladd.1 ซึ่งจะปรากฏรูปวงจรระดับเกทของโปรแกรม Fulladd.vhd ดังรูปที่ 2.67

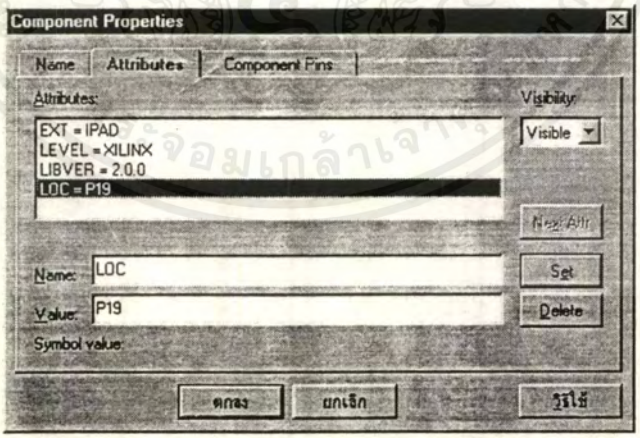


รูปที่ 2.66 หน้าต่างของโปรแกรมวิวดรอว์



รูปที่ 2.67 รูปวงจรระดับเกทของโปรแกรม Fulladd.vhd

เมื่อได้วงจรระดับเกทของโปรแกรม Fulladd.vhd แล้วซึ่งมีขาต่อใช้งาน 5 ขา คือ A, B, Cin Cout และ Sum โดยต้องทำการกำหนดขาสัญญาณทั้ง 5 เพื่อจะนำไปโปรแกรมลงบนอุปกรณ์ FPGAs ซึ่งจะมีวิธีการดังนี้คือ เลื่อนเมาส์ไปที่อินพุทและเอาท์พุทแพทและคลิกเมาส์ด้านซ้าย พร้อมทั้งเลือกคำสั่ง Properties โดยจะแสดงหน้าต่างดังรูปที่ 2.68



รูปที่ 2.68 หน้าต่างของ Properties

ที่ช่อง Name ให้ใส่ชื่อคำว่า LOC และช่อง Value ใส่ค่า P19 ซึ่งเป็นขาของอุปกรณ์ FPGAs ที่จะให้เป็นขาสัญญาณอินพุท A เสร็จแล้วคลิกเมาส์ที่ช่องตกลง โดยที่ขาของสัญญาณอินพุท A จะปรากฏข้อความ LOC = 19 ที่ IPAD

ขาสัญญาณอินพท B กำหนดเป็นขา LOC = P20

ขาสัญญาณอินพุท Cin กำหนดเป็นขา LOC = P23

ขาสัญญาณเอาท์พุท Cout กำหนดเป็นขา LOC = P61

ขาสัญญาณเอาท์พุท Sum กำหนดเป็นขา LOC = P62

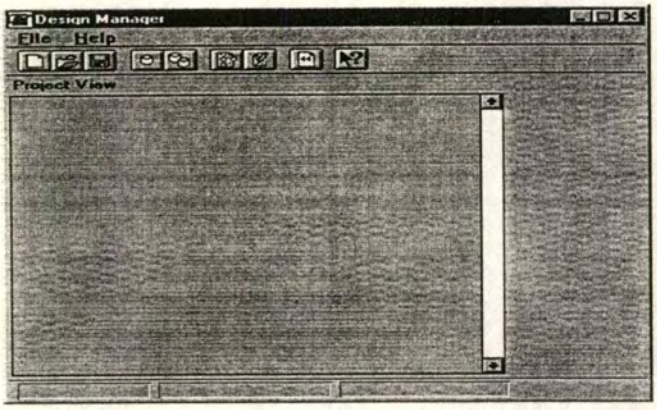
เมื่อกำหนดขาให้กับอินพุทเอาท์พุทแพทเสร็จแล้วให้ทำการบันทึกแฟ้มข้อมูล และไปสู่กระบวนการโปรแกรมลงอุปกรณ์ FPGAs โดยใช้ซอฟแวร์ของบริษัทไซลิงค์

2.13 ขั้นตอนการโปรแกรมลงบนอุปกรณ์ FPGAs โดยใช้ซอฟแวร์ XDM ของบริษัทไซลิงค์ (Xilinx)

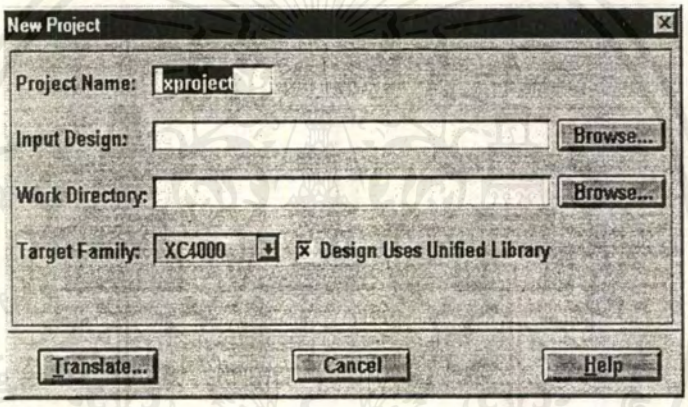
เมื่อเริ่มเข้าสู่โปรแกรม XACT Design Manager (XDM) ที่หน้าจอจะแสดงรูปภาพ ดังรูปที่ 2.69 หลังจากนั้นจะเข้าสู่หน้าต่าง Design Manager ดังรูปที่ 2.70 ให้คลิกเมาส์ที่เมนู File และเลือกที่ New Project โดยจะแสดงหน้าต่าง ดังรูปที่ 2.71

คลิกเมาส์ที่ Browse บริเวณช่อง Input Design โดยให้ไปที่ไดเร็คทอรีที่มีไฟล์ Fulladd.1 ซึ่งเป็นวงจรระดับเกทของโปรแกรม Fulladd.vhd ที่ได้สร้างไว้ในหัวข้อที่ 2.12.1 ดังรูปที่ 2.72

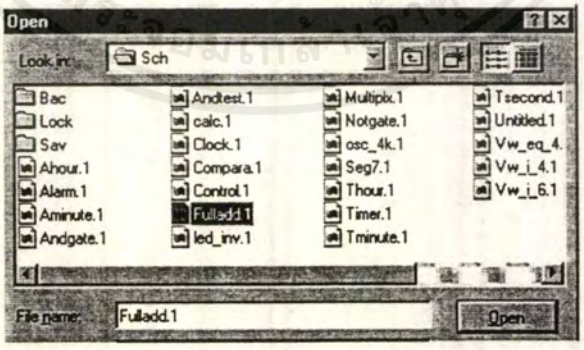
รูปที่ 2.69 หน้าจอเริ่มต้นของโปรแกรม XACT Design Manager



รูปที่ 2.70 หน้าต่าง Design Manager

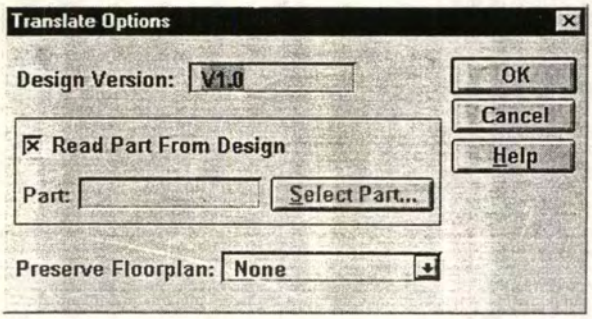


รูปที่ 2.71 หน้าต่าง New Project



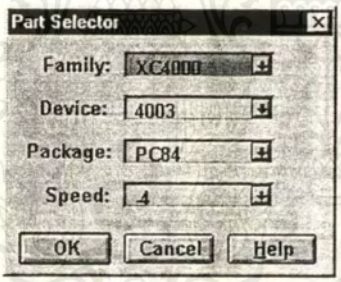
รูปที่ 2.72 หน้าต่าง Input Design

คลิกเมาส์ที่ Translate จะแสดงหน้าต่าง Translate ดังรูปที่ 2.73



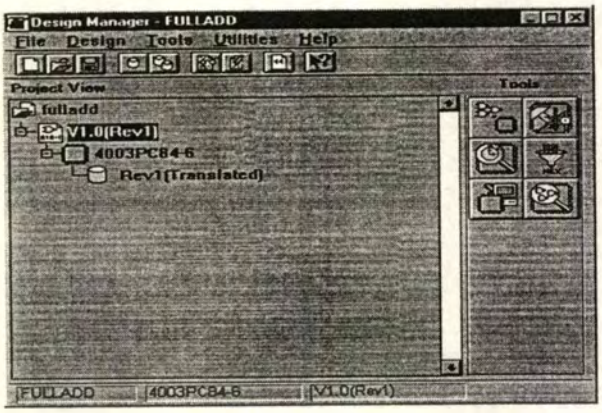
รูปที่ 2.73 หน้าต่าง Translate

คลิกเมาส์ที่ช่อง Real Part Design และที่ Select part ซึ่งจะแสดงหน้าต่างของ Part Selector และเปลี่ยน Speed จาก-4 เป็น-6 ซึ่งเป็นเบอร์ของไอซี FPGAs เบอร์ XC4005APC84C-6 ที่ใช้ในบอร์ดตัวอย่างของ FPGAs ดังรูปที่ 2.74

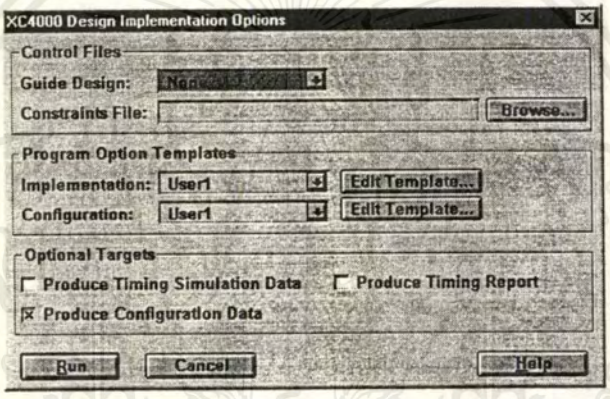


รูปที่ 2.74 หน้าต่าง Part Selector

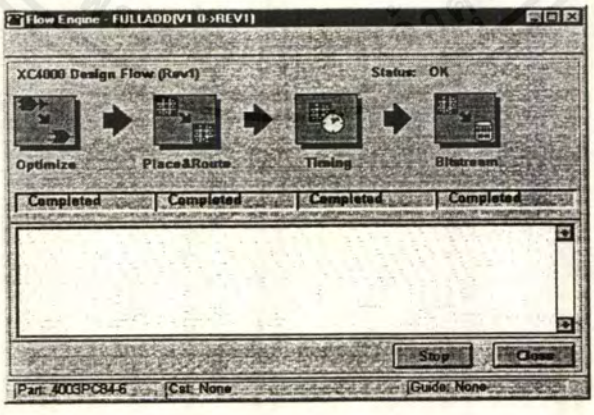
เมื่อทำการกำหนดค่าต่างๆ แล้วโปรแกรมจะทำการ Translateให้ เมื่อเสร็จการ Translate แล้วจะแสดงหน้าต่างดังรูปที่ 2.75 หลังจากนั้นให้คลิกที่เมนู Design เสร็จแล้วเลือกคำสั่ง Implement จะแสดงหน้าต่าง XC 4000 Design Implementation Option ดังรูปที่ 2.76 ที่ช่อง Option Targets จะให้เลือกการรายงานผลการ Run โดยให้เลือกรายงานผลทุกอย่าง เสร็จแล้วคลิกเมาส์ที่ Run ซึ่งจะเป็นกระบวนการ Design จากวงจรระดับเกทของไฟล์ Fulladd.1 ให้เป็นไฟล์บิตสตรีม (Bit Stream) เพื่อใช้ดาวน์โหลดลงบนอุปกรณ์ FPGAs ดังรูปที่ 2.77



รูปที่ 2.75 ผลการ Translate

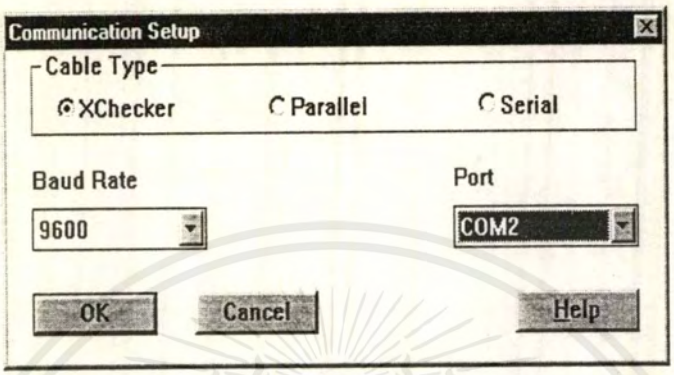


รูปที่ 2.76 หน้าต่าง XC 4000 Design Implementation Options



รูปที่ 2.77 หน้าต่างกระบวนการสร้างไฟล์บิทสตรีม

เมื่อเสร็จกระบวนการแล้วจะได้ไฟล์ที่มีนามสกุล .Bit เพื่อดาวน์โหลดลงบนบอร์ดตัวอย่างของ FPGAs โดยในการดาวน์โหลดไฟล์บิทสตรีมลงบนบอร์ดตัวอย่างของ FPGAs นั้น ให้คลิกเมาส์ที่เมนู Tools และเลือกที่ Hardware Debugger ซึ่งจะแสดงหน้าต่าง Communication Setup ดังรูปที่ 2.78



รูปที่ 2.78 หน้าต่าง Communication Setup

เมื่อทำการกำหนดค่า Baud Rate และเลือก Port ที่จะส่งข้อมูลออกไปแล้ว ให้คลิกที่ OK เมื่อเสร็จแล้วให้ไปที่หน้าต่าง Hardware Debugger คลิกที่เมนู Download และเลือกที่ Download Design ซึ่งโปรแกรม Debugger จะทำการ Download Design ไฟล์ Fulladd.Bit ลงบอร์ดตัวอย่างของ FPGAs ผ่านทางสายดาวน์โหลดเคเบิล เมื่อดาวน์โหลดเสร็จเรียบร้อยแล้วจึงนำ ลงบอร์ดตัวอย่างของ FPGAs ไปทดสอบการทำงานของวงจร Fulladder โดยมีขาสัญญาณอินพุทเอาท์พุทตามที่ได้กำหนดไว้ในหัวข้อ 2.12.2

บทที่ 3

การออกแบบและการสร้าง

การออกแบบและการสร้างวงจรตั้งเวลาขนาด 3 ช่อง มีรายละเอียดดังนี้

3.1 ลักษณะการออกแบบ

3.1.1 การออกแบบจากล่างขึ้นบน (Bottom Up Design)

ในอดีตถึงปัจจุบันการออกแบบในระบบดิจิตอลจะเป็นลักษณะที่เรียกว่า "การออกแบบจากล่างขึ้นบน" (Bottom Up Design) คือผู้ออกแบบจะเริ่มต้นกำหนดหัวข้องาน แล้วใช้หลักการทางทฤษฎีแบ่งออกเป็นฟังก์ชันการทำงานต่างๆ แล้วเริ่มต้นออกแบบ เมื่อได้วงจรตามที่ต้องการแล้วจะต้องหาอุปกรณ์มาตราฐานต่างๆ เช่น IC 74LSXX เป็นต้น เพื่อนำมารองรับฟังก์ชันการทำงานต่างๆที่ได้จากการออกแบบ ถ้าไม่สามารถหาอุปกรณ์มารองรับได้จะต้องออกแบบหรือดัดแปลงวงจรใหม่ ในขั้นตอนสุดท้ายคือการจำลองการทำงานโดยทดลองจากวงจรต้นแบบ

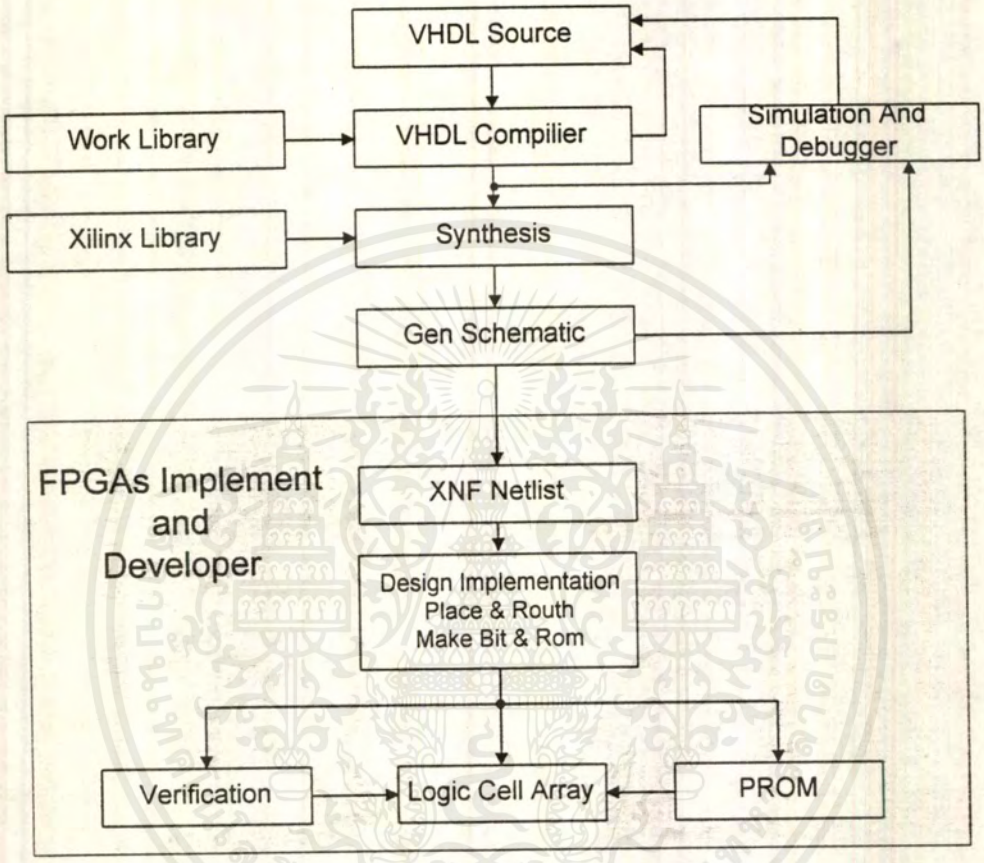
3.1.2 การออกแบบจากบนลงล่าง (Top Down Design)

จากการออกแบบในหัวข้อที่ 3.1.1 นั้นในกรณีที่การทำงานของวงจรไม่ตรงตามข้อกำหนดต้องทำการออกแบบและประกอบวงจรใหม่อีกครั้ง ซึ่งจะเห็นว่าขั้นตอนและกระบวนการดังกล่าวยุ่งยากและใช้เวลามาก ดังนั้นในการออกแบบวงจรสมัยใหม่สามารถออกแบบระบบดิจิตอลได้จากแนวคิดโดยสังเขป โดยวิธีการเขียนรูปแบบแล้วทดลองการทำงานของรูปแบบนั้นจนเป็นที่น่าพอใจแล้วค่อยๆ เพิ่มเติมรายละเอียดของระบบไปทีละขั้น ซึ่งในแต่ละขั้นตอนสามารถที่จะจำลองการทำงานภายใต้สภาวะแวดล้อมเดิมได้ ทำให้ไม่มีโอกาสที่รูปแบบการทำงานจะผิดไปจากวัตถุประสงค์เดิม เมื่อเกิดข้อผิดพลาดก็สามารถแก้ไขได้ทันที หลังจากนั้นจึงนำไปลงอุปกรณ์และทดสอบการทำงานของวงจรซ้ำอีกครั้ง การออกแบบในลักษณะนี้เรียกว่า "การออกแบบจากบนลงล่าง " (Top Down Design)

3.2 วิธีการออกแบบ

จากหัวข้อที่ 3.1 จะเห็นว่าการออกแบบจากบนลงล่างนั้น มีความสะดวกและรวดเร็วกว่าการออกแบบจากล่างขึ้นบนมาก เพราะสามารถจำลองการทำงานของวงจรที่ออกแบบจนแน่ใจว่าทำงานได้ตรงตามวัตถุประสงค์ที่ต้องการแล้ว จึงนำไปสร้างเป็นวงจรจริงในภายหลัง โครงงานนี้จึงได้เลือกวิธีการออกแบบจากบนลงล่างในการสร้างวงจรตั้งเวลา โดยทำการออกแบบวงจรตั้งเวลาขนาด 3 ช่องด้วยภาษา VHDL ซึ่งเป็นซอฟแวร์ของบริษัทวิวลอจิก และสร้างวงจรตั้งเวลาขนาด

3 ช่องตามที่ได้ออกแบบไว้ด้วยอุปกรณ์ FPGAs ของบริษัทไซลิงค์ ขั้นตอนการออกแบบโดยใช้ซอฟแวร์และอุปกรณ์ดังกล่าว แสดงได้ดังรูปที่ 3.1



รูปที่ 3.1 ผังการทำงานของการออกแบบวงจรโดยใช้ซอฟแวร์ของบริษัทวิวลอจิก และอุปกรณ์ FPGAs ของบริษัทไซลิงค์

3.3 คุณสมบัติของวงจรตั้งเวลาขนาด 3 ช่อง

วงจรตั้งเวลาขนาด 3 ช่องที่ทำการออกแบบ ได้กำหนดคุณสมบัติไว้ดังนี้คือ

- แสดงผลการทำงานโดยใช้ส่วนแสดงผลแบบ 7 ส่วนจำนวน 4 หลัก
- ใช้สัญญาณนาฬิกา 15 Hz. เป็นฐานเวลาในการทำงาน
- สามารถตั้งเวลาได้ 3 ช่อง
- วงจรตั้งเวลาแต่ละช่องสามารถตั้งเวลาได้เป็นอิสระจากกัน
- ใช้อุปกรณ์ FPGAs เบอร์ XC4005APC84C-6 ควบคุมการทำงาน

- วงจรตั้งเวลาสามารถตั้งเวลาได้ตั้งแต่ 1 นาที จนถึง 23 ชั่วโมง 59 นาที

- สามารถสลับการแสดงผลของส่วนแสดงผลแบบ 7 ส่วนระหว่างวงจรนาฬิกาและวงจรตั้งเวลา

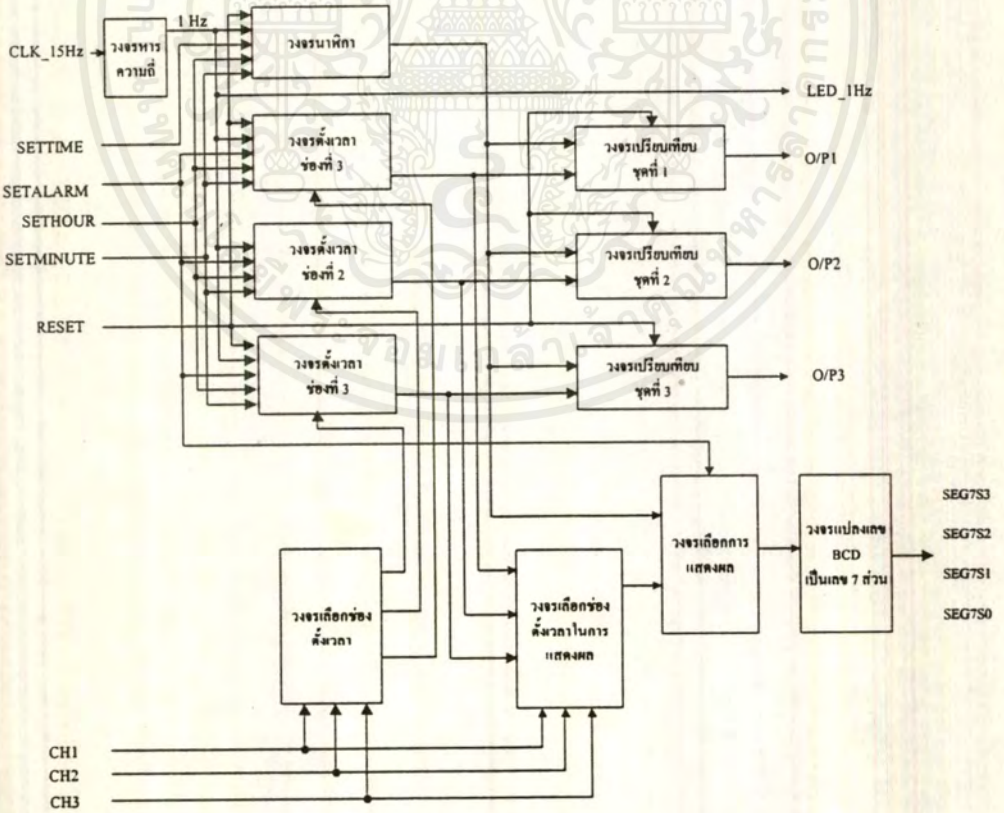
- วงจรนาฬิกาสามารถเดินได้ถึง 23 ชั่วโมง 59 นาที 59 วินาที และจะกลับมาเริ่มต้นที่ 0 นาฬิกาใหม่

- เมื่อนาฬิกาเดินถึงเวลาของช่องที่ตั้งเวลาไว้ จะทำให้หลอดไฟแสดงผลที่ช่องนั้นติดสว่างและรีเลย์ทำงาน

- สามารถรีเซ็ทเวลาให้เริ่มต้นที่ 0 นาฬิกาใหม่

3.4 ผังการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง

ผังการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง แสดงได้ดังรูปที่ 3.2 ซึ่งอธิบายการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง ในแต่ละส่วนได้ดังนี้คือ



รูปที่ 3.2 ผังการทำงานของวงจรตั้งเวลาขนาด 3 ช่อง

- วงจรหารความถี่ ทำหน้าที่หารสัญญาณนาฬิกาขนาด 15 Hz จากวงจรกำเนิดสัญญาณนาฬิกา ซึ่งเป็นวงจรภายในของไอซี FPGAs ให้เหลือ 1 Hz ป้อนให้กับวงจรนาฬิกาและวงจรตั้งเวลาแต่ละช่อง

- วงจรนาฬิกา ทำหน้าที่แสดงเวลาผ่านทางส่วนแสดงผลแบบ 7 ส่วน โดยแสดงเวลาในหลักชั่วโมง หลักนาที และมีหลอดไฟกะพริบเพื่อแสดงเวลาในหลักวินาที

- วงจรตั้งเวลา ทำหน้าที่ตั้งเวลาเพื่อควบคุมอุปกรณ์ไฟฟ้า โดยสามารถตั้งเวลาได้ตั้งแต่ 1 นาที จนถึง 23 ชั่วโมง 59 นาที

- วงจรเปรียบเทียบ ทำหน้าที่เปรียบเทียบเวลาวงจรตั้งเวลาแต่ละช่องกับเวลาของวงจรนาฬิกา เมื่อเวลาของวงจรทั้งสองเท่ากันเอาท์พุทของวงจรเปรียบเทียบจะมีสถานะเป็น “1” ทำให้วงจรควบคุมอุปกรณ์ไฟฟ้าทำงาน

- วงจรเลือกช่องตั้งเวลา ทำหน้าที่เลือกช่องในการตั้งเวลาโดยทำการตั้งเวลาได้ทีละช่อง

- วงจรเลือกการแสดงผล ทำหน้าที่สลับการแสดงผลของส่วนแสดงผลแบบ 7 ส่วนระหว่างวงจรนาฬิกาและวงจรตั้งเวลา

- วงจรแปลงเลข BCD (Binary Code Decimal) เป็นเลข 7 ส่วน ทำหน้าที่แปลงสัญญาณเอาท์พุทจากวงจรเลือกการแสดงผลที่เป็นเลข BCD ให้เป็นเลข 7 ส่วน เพื่อส่งไปแสดงผลที่ภาคแสดงผลแบบ 7 ส่วน

3.5 หน้าที่การทำงานของขาสัญญาณแต่ละขา

- ขา CLK_15 Hz เป็นสัญญาณฐานเวลาให้แก่วงจรตั้งเวลาขนาด 3 ช่อง โดยรับสัญญาณมาจากวงจรสร้างสัญญาณนาฬิกาขนาด 15Hz ซึ่งเป็นฟังก์ชันภายในของอุปกรณ์ FPGAs ที่สามารถเรียกใช้ได้ เข้ามาเพื่อหารความถี่ให้เหลือ 1Hz ป้อนให้วงจรนาฬิกาและวงจรตั้งเวลาแต่ละช่อง

- ขา RESET ทำหน้าที่รีเซ็ทเวลาของวงจรนาฬิกาและวงจรตั้งเวลา ให้เป็น 0 นาฬิกา

- ขา SET ALARM ใช้ตั้งเวลาให้กับวงจรตั้งเวลา และสลับการแสดงผลของส่วนแสดงผลแบบ 7 ส่วนให้แสดงเวลาจากวงจรตั้งเวลา

- ขา SET TIME ใช้ตั้งเวลาให้กับวงจรนาฬิกา

- ขา SET HOUR ใช้ตั้งเวลาให้กับวงจรนาฬิกา และวงจรตั้งเวลาในหลักชั่วโมง

- ขา SET MINUTE ใช้ตั้งเวลาให้กับวงจรนาฬิกา และวงจรตั้งเวลาในหลักนาที

- ขา SEG7S0 เป็นขาเอาท์พุทจำนวน 7 ขาที่ต่อเข้ากับส่วนแสดงผลแบบ 7 ส่วนในหลักนาที หลักหน่วย

- ขา SEG7S1 เป็นขาเอาท์พุทจำนวน 7 ขาที่ต่อเข้ากับส่วนแสดงผลแบบ 7 ส่วนในหลักนาที หลักสิบ

- ขา SEG7S2 เป็นขาเอาท์พุทจำนวน 7 ขาที่ต่อเข้ากับส่วนแสดงผลแบบ 7 ส่วนในหลักชั่วโมง หลักหน่วย

- ขา SEG7S3 เป็นขาเอาท์พุทจำนวน 7 ขาที่ต่อเข้ากับส่วนแสดงผลแบบ 7 ส่วนในหลักชั่วโมง หลักสิบ

- ขา LED_1Hz เป็นขาที่ต่อกับส่วนแสดงผลในหลักวินาที

- ขา CH1 ใช้ตั้งเวลาของวงจรตั้งเวลาช่องที่ 1

- ขา CH2 ใช้ตั้งเวลาของวงจรตั้งเวลาช่องที่ 2

- ขา CH3 ใช้ตั้งเวลาของวงจรตั้งเวลาช่องที่ 3

- ขา O/P1 ใช้ส่งสัญญาณไปควบคุมการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1

- ขา O/P2 ใช้ส่งสัญญาณไปควบคุมการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2

- ขา O/P3 ใช้ส่งสัญญาณไปควบคุมการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 3

- ขา OFF_CH1 ใช้หยุดการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1

- ขา OFF_CH2 ใช้หยุดการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2

- ขา OFF_CH3 ใช้หยุดการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 3

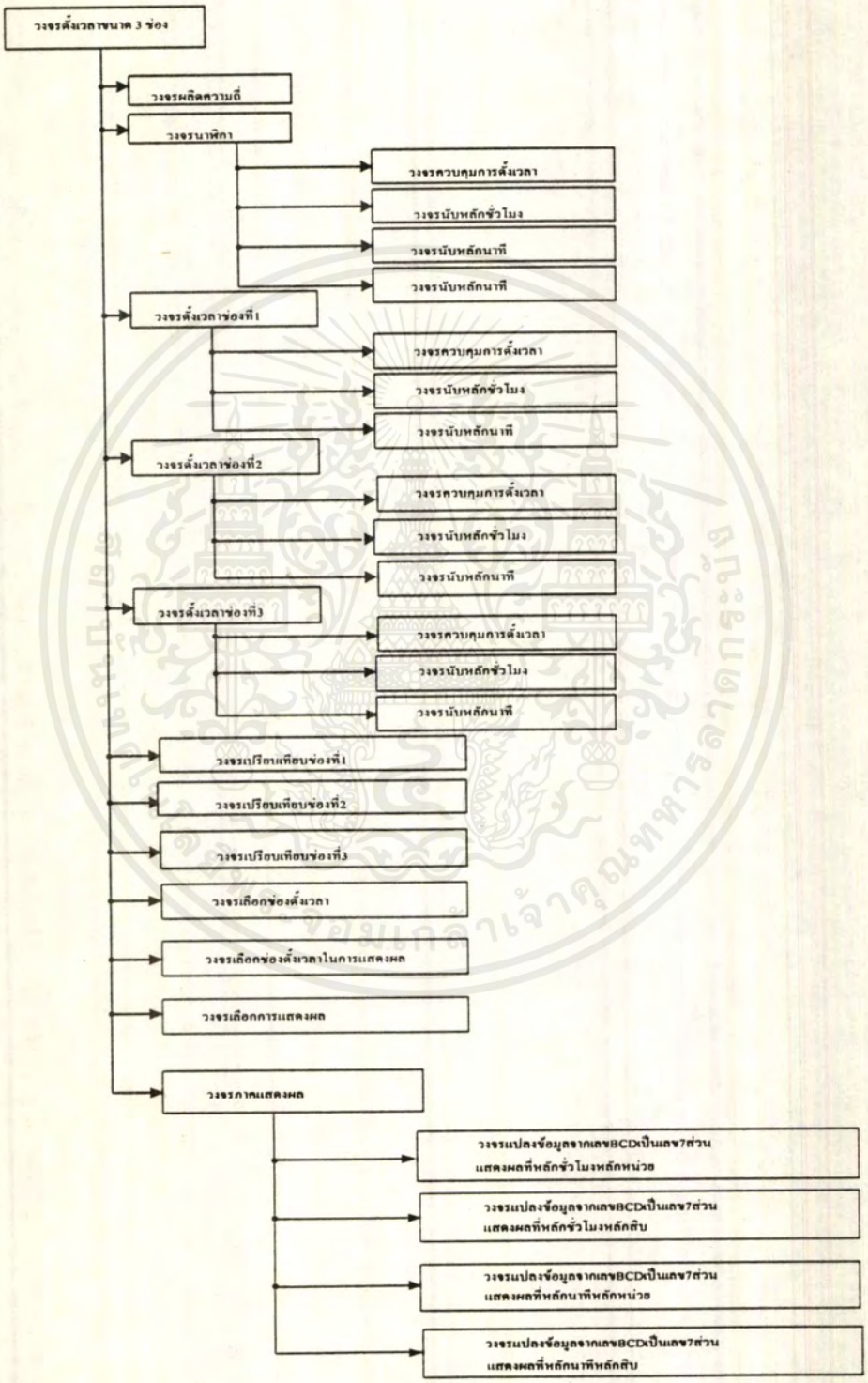
3.6 ขั้นตอนการออกแบบวงจรตั้งเวลาขนาด 3 ช่อง

3.6.1 การออกแบบวงจรตั้งเวลาขนาด 3 ช่องด้วยภาษา VHDL

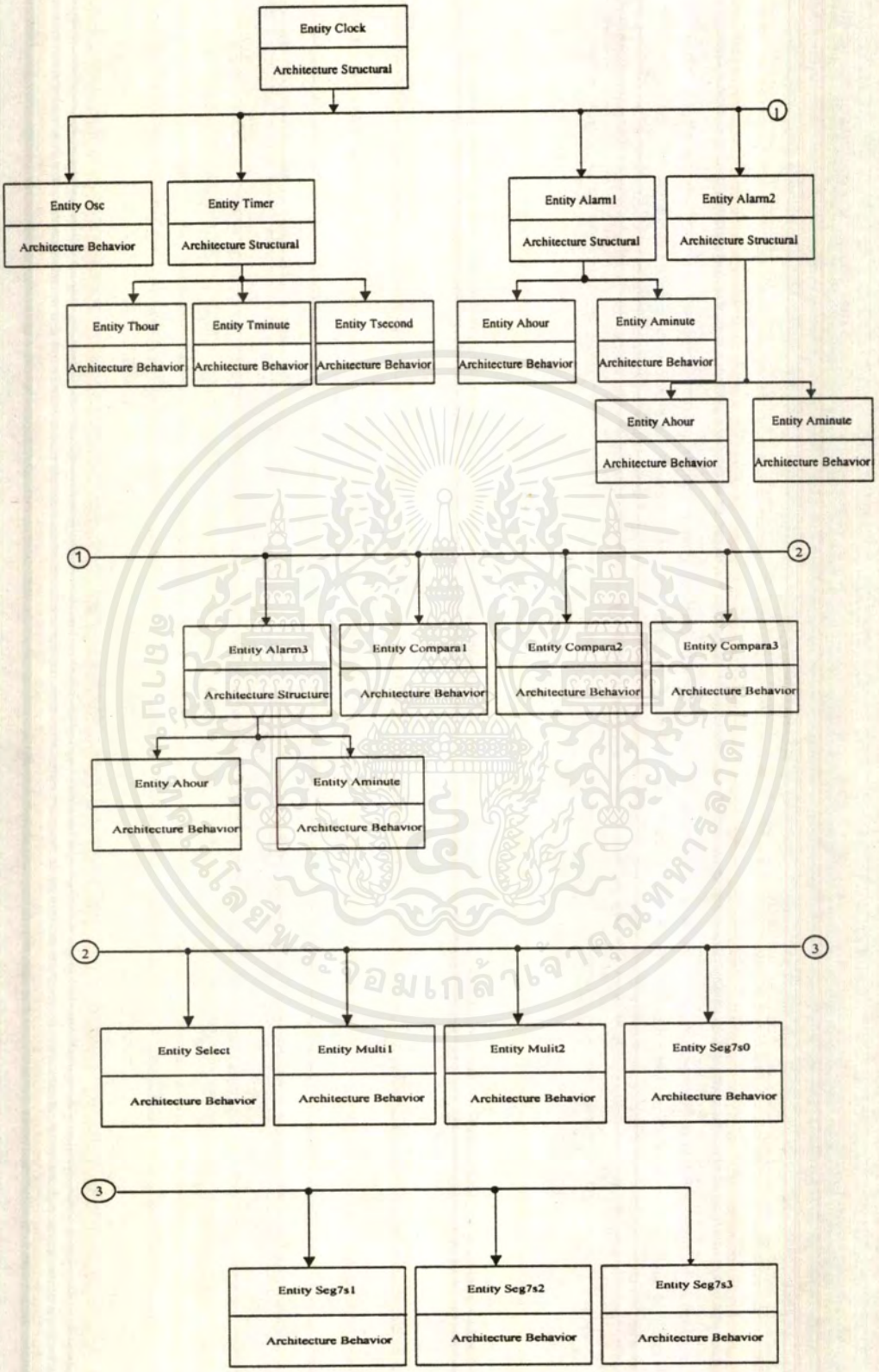
หลังจากที่ได้กำหนดคุณสมบัติของวงจรตั้งเวลาขนาด 3 ช่องเสร็จแล้ว โดยทราบว่าวงจรตั้งเวลาขนาด 3 ช่องที่มีขาสัญญาณอินพุทและเอาท์พุทกี่ขา มีการทำงานเป็นอย่างไร การออกแบบวงจรที่มีขนาดใหญ่ควรออกแบบเป็นผังการทำงาน ดังแสดงในรูปที่ 3.2 ซึ่งประกอบด้วยวงจรนาฬิกา วงจรตั้งเวลา วงจรเปรียบเทียบ เป็นต้น หลังจากนั้นทำการแยกส่วนในแต่ละวงจรออกเป็นส่วนย่อยลงมาเรื่อยๆ เป็นลำดับขั้น ดังรูปที่ 3.3

การออกแบบในลักษณะการแยกส่วนออกเป็นส่วนย่อยนี้ เมื่อเราใช้ภาษา VHDL ในการออกแบบนั้น หน้าที่การทำงานของแต่ละส่วนสามารถอธิบายได้ในลักษณะส่วนของโปรแกรมภาษา ซึ่งแสดงการทำงานของวงจรในส่วนย่อยนั้นการแตกวงจรใหญ่ๆ เป็นส่วนย่อยนี้ทำให้ง่ายต่อการออกแบบและง่ายต่อการทำความเข้าใจ เช่น วงจรนาฬิกา ประกอบไปด้วยส่วนของวงจรนับหลักชั่วโมง วงจรนับหลักนาที วงจรนับหลักวินาที และวงจรควบคุมการตั้งเวลา จากรูปที่ 3.4 เมื่อแยกส่วนของวงจรตั้งเวลาขนาด 3 ช่องออกเป็นส่วนย่อยต่างๆ แล้วใช้ภาษา VHDL อธิบายการ

ทำงานของวงจรในส่วนย่อย เมื่อเสร็จในแต่ละส่วนก็ประกอบส่วนย่อยแต่ละส่วนเข้าด้วยกันเป็นวงจรรวมขนาดใหญ่ที่สมบูรณ์



รูปที่ 3.3 การแยกส่วนวงจรตั้งเวลาขนาด 3 ช่องออกเป็นส่วนย่อย



รูปที่ 3.4 การอธิบายการทำงานแต่ละส่วนย่อยด้วยภาษา VHDL

ขอยกตัวอย่างให้เห็นถึงโปรแกรมเพียงบางส่วน เช่นวงจรนาฬิกา โดยโปรแกรมที่สมบูรณ์สามารถดูได้จากภาคผนวก ก. ที่วงจรนาฬิกาซึ่งภายในประกอบด้วยวงจรนับหลักชั่วโมง วงจรนับหลักนาที และหลักวินาที รวมทั้งวงจรควบคุมการตั้งเวลา โดยสามารถใช้ภาษาVHDL อธิบายการทำงานของวงจรนับหลักชั่วโมงดังแสดงในรูปที่ 3.5 ซึ่งเป็นการอธิบายการทำงานของวงจรนับหลักชั่วโมงที่กำหนดให้ทำงานเป็นวงจรนับ 24 ส่วนรูปที่ 3.6 และรูปที่ 3.7 เป็นการอธิบายการทำงานของวงจรนับหลักนาที และหลักวินาทีซึ่งทำงานเป็นวงจรนับ 60

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
LIBRARY synth;
USE synth.vhdlsynth.all;
ENTITY  Thour IS
    PORT (      clk             :IN  std_logic;
                reset           :IN  std_logic;
                bcdhour0        :OUT std_logic_vector (3 downto 0);
                bcdhour1         :OUT std_logic_vector (3 downto 0));
 END Thour;
ARCHITECTURE behavioral OF Thour IS
    BEGIN
        W: PROCESS (clk)
                variable bus0 :  std_logic_vector( 3 downto 0):= "0000";
                variable bus1 :  std_logic_vector( 3 downto 0):= "0000";
        BEGIN
                if (clk'event and clk = '1') then
                        if (reset = '0')then
                         bus0 := (bus0 + "0001");
                           if ( bus0 = "1010")then
                                bus0 := "0000";
```

รูปที่ 3.5 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักชั่วโมง

```
                bus1 := (bus1 + "0001");
                end if;
                if (bus1 = "0010")and(bus0 = "0100")then
                        bus0 := "0000";
                        bus1 := "0000";
                end if;
                else
                        bus0 := "0000";
                        bus1 := "0000";
                end if;
                bcdhour0 <= bus0;
                bcdhour1 <= bus1;
        end if;
    END PROCESS W ;
END behavioral ;
```

รูปที่ 3.5 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักชั่วโมง (ต่อ)

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
LIBRARY synth;
USE synth.vhdlsynth.all;

ENTITY Tminute IS
  PORT (    clk         :IN  std_logic;
            reset       :IN  std_logic;
            bcdmin0     :OUT std_logic_vector (3 downto 0);
            bcdmin1     :OUT std_logic_vector (3 downto 0);
```

รูปที่ 3.6 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักนาที

```
        carryout     :OUT std_logic);
END Tminute;

ARCHITECTURE behavioral OF Tminute IS
    BEGIN
      W: PROCESS (clk)
            variable bus0 :  std_logic_vector(3 downto 0):= "0000";
            variable bus1 :  std_logic_vector(3 downto 0):= "0000";
          BEGIN
            if (clk'event and clk = '1') then
                if (reset = '0')then
                  bus0 := (bus0 + "0001");
                  if (bus0 = "1010")then
                     bus0 := "0000";
                     bus1 := (bus1 + "0001");
                       carryout <= '0';
                  end if;
                     if (bus1 = "0110") then
                       bus0 := "0000";
                       bus1 := "0000";
                       carryout <= '1';
                     else
                       carryout <= '0';
                     end if;
                else
                     bus0 := "0000";
                     bus1 := "0000";
                end if;
```

รูปที่ 3.6 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักนาที (ต่อ)

```
        bcdmin0 <= bus0;
                bcdmin1 <= bus1;
            end if;
    END PROCESS W ;
END behavioral ;
```

รูปที่ 3.6 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักนาที (ต่อ)

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
LIBRARY synth;
USE synth.vhdlsynth.all;
ENTITY Tsecond IS
  PORT (    clk          :IN  std_logic;
            reset        :IN  std_logic;
            bcdsec0      :OUT std_logic_vector (3 downto 0);
            bcdsec1      :OUT std_logic_vector (3 downto 0);
            carryout     :OUT std_logic);
  END Tsecond;
ARCHITECTURE behavioral OF Tsecond IS
    BEGIN
      W: PROCESS (clk)
            variable bus0 :  std_logic_vector(3 downto 0):= "0000";
            variable bus1 :  std_logic_vector(3 downto 0):= "0000";
          BEGIN
            if (clk'event and clk = '1') then
                if (reset = '0')then
                  bus0 := (bus0 + "0001");
```

รูปที่ 3.7 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักวินาที

```
                if (bus0 = "1010")then
                    bus0 := "0000";
                    bus1 := (bus1 + "0001");
                    carryout <='0';
                end if;
                    if (bus1 = "0110") then
                        bus0 := "0000";
                        bus1 := "0000";
                        carryout <= '1';
                    else
                        carryout <= '0';
                    end if;
                else
                    bus0 := "0000";
                    bus1 := "0000";
                end if;
            bcdsec0 <= bus0;
            bcdsec1 <= bus1;
        end if;
    END PROCESS W ;
END behavioral ;
```

รูปที่ 3.7 การใช้ภาษา VHDL อธิบายการทำงานของวงจรนับหลักวินาที (ต่อ)

หลังจากที่เขียนโปรแกรมอธิบายการทำงานของวงจรนับหลักชั่วโมง หลักนาที และหลักวินาทีแล้ว ขั้นต่อมานำโปรแกรมทั้ง 3 ส่วนมาประกอบกันเป็นโปรแกรมของวงจรนาฬิกา โดยรูปที่ 3.8 แสดงโปรแกรมการทำงานของวงจรนาฬิกาซึ่งใช้การบรรยายเชิงโครงสร้าง โดยบรรยายว่าภายในประกอบด้วยวงจรนับหลักชั่วโมง หลักนาที หลักวินาทีและวงจรควบคุมการตั้งเวลา สุดท้ายจะได้โปรแกรมของวงจรนาฬิกาซึ่งทำหน้าที่เป็นส่วนแสดงเวลาที่เดินอยู่ในปัจจุบัน วงจรส่วนอื่นก็ใช้การบรรยายในลักษณะเดียวกัน ขั้นสุดท้ายก็นำเอาโปรแกรมแต่ละส่วนมารวมกันจน

ได้โปรแกรมของวงจรตั้งเวลาขนาด 3 ช่องที่สมบูรณ์ การแบ่งการออกแบบเป็นส่วนย่อยนี้จะพบว่าสามารถนำเอาโปรแกรมส่วนย่อยไปใช้งานในส่วนอื่นๆ ได้ ทำให้ลดความซ้ำซ้อน และง่ายต่อการแก้ไข

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY synth;
USE synth.vhdlsynth.all;
ENTITY Timer is
        PORT ( clk_1hz          :IN  std_logic;
               clk_15hz         :IN std_logic;
               reset            :IN  std_logic;
               settime          :IN std_logic;
               sethour          :IN std_logic;
               setminute        :IN std_logic;
               tsec_out0        :OUT std_logic_vector(3 downto 0);
               tsec_out1        :OUT std_logic_vector(3 downto 0);
               tmin_out0        :OUT std_logic_vector(3 downto 0);
               tmin_out1        :OUT std_logic_vector(3 downto 0);
               thr_out0         :OUT std_logic_vector(3 downto 0);
               thr_out1         :OUT std_logic_vector(3 downto 0));
        END Timer;

ARCHITECTURE behavioral OF Timer IS
    COMPONENT Tsecond
        PORT ( clk              :IN std_logic;
               reset            :IN std_logic;
               bcdsec0          :OUT std_logic_vector (3 downto 0);
```

รูปที่ 3.8 การบรรยายเชิงโครงสร้างของวงจรนาฬิกา

```
        bcdsec1          :OUT std_logic_vector (3 downto 0);
        carryout         :OUT std_logic);
END COMPONENT;
COMPONENT Tminute
    PORT ( clk           :IN std_logic;
        reset            :IN std_logic;
        bcdmin0          :OUT std_logic_vector (3 downto 0);
        bcdmin1          :OUT std_logic_vector (3 downto 0);
        carryout         :OUT std_logic);
    END COMPONENT;
COMPONENT Thour
    PORT ( clk           :IN std_logic;
        reset            :IN std_logic;
        bcdhour0         :OUT std_logic_vector (3 downto 0);
        bcdhour1         :OUT std_logic_vector (3 downto 0));
    END component;

    SIGNAL bus_reset     :std_logic;
    SIGNAL gate_reset    :std_logic;
    SIGNAL to_clock      :std_logic;
    SIGNAL to_clk_min    :std_logic;
    SIGNAL to_clk_hr     :std_logic;
    SIGNAL clk_minute    :std_logic;
    SIGNAL clk_hour      :std_logic;
    SIGNAL c_to_minute   :std_logic;
    SIGNAL c_to_hour     :std_logic;
    BEGIN
        bus_reset        <= reset OR gate_reset;
```

รูปที่ 3.8 การบรรยายเชิงโครงสร้างของวงจรนาฬิกา (ต่อ)

```
        to_clk_min      <= c_to_minute OR clk_minute;
        to_clk_hr       <= c_to_hour OR clk_hour;
PROCESS (settime, sethour, setminute, clk_1hz, clk_15hz)
BEGIN
        IF (settime = '1') THEN
                IF (sethour = '0') THEN
                        IF (setminute = '0') THEN
                                to_clock        <= clk_1hz;
                                gate_reset      <= '0';
                                clk_minute      <= '0';
                                clk_hour        <= '0';
                        ELSIF (setminute = '1') THEN
                                to_clock        <= '0';
                                gate_reset      <= '0';
                                clk_minute      <= clk_15hz;
                                clk_hour        <= '0';
                        end if;
                ELSIF (sethour = '1') THEN
                        IF (setminute = '0') THEN
                                to_clock        <= '0';
                                gate_reset      <= '0';
                                clk_minute      <= '0';
                                clk_hour        <= clk_15hz;
                        ELSIF (setminute = '1') THEN
                                to_clock        <= clk_1hz;
                                gate_reset      <= '0';
                                clk_minute      <= '0';
                                clk_hour        <= '0';
```

รูปที่ 3.8 การบรรยายเชิงโครงสร้างของวงจรนาฬิกา (ต่อ)

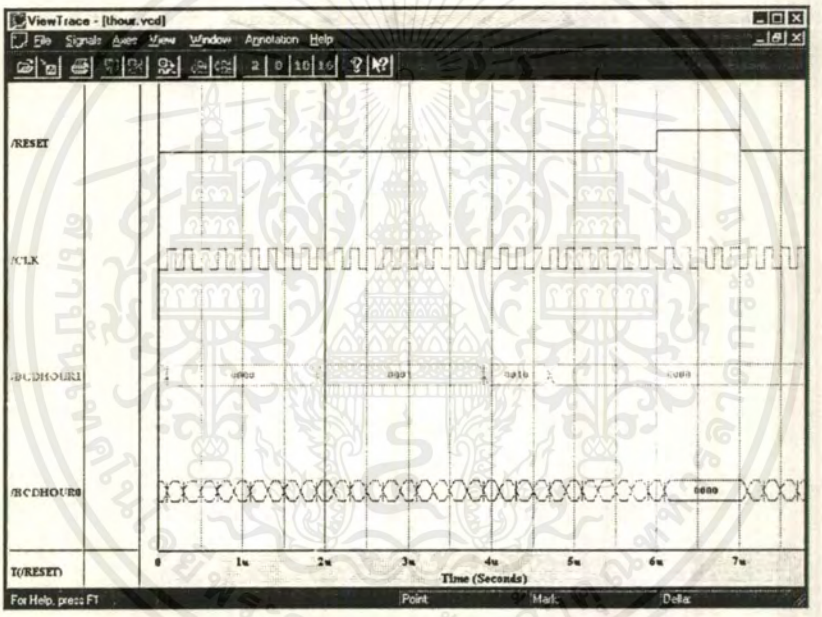
```
                    END IF;
                END IF;
            ELSIF (settime = '0') THEN
                to_clock        <= clk_1hz;
                gate_reset      <= '0';
                clk_minute      <= '0';
                clk_hour        <= '0';
            END IF;
    END PROCESS;

    block0: Tsecond
            PORT map (clk       => to_clock,
                      reset     => bus_reset,
                      bcdsec0   => tsec_out0,
                      bcdsec1   => tsec_out1,
                      carryout  => c_to_minute);
    block1: Tminute
            PORT map (clk       => to_clk_min,
                      reset     => bus_reset,
                      bcdmin0   => tmin_out0,
                      bcdmin1   => tmin_out1,
                      carryout  => c_to_hour);
    block2: Thour
            PORT map (clk       => to_clk_hr,
                      reset     => bus_reset,
                      bcdhour0  => thr_out0,
                      bcdhour1  => thr_out1);
    END behavioral;
```

รูปที่ 3.8 การบรรยายเชิงโครงสร้างของวงจรนาฬิกา (ต่อ)

3.6.2 การจำลองการทำงาน

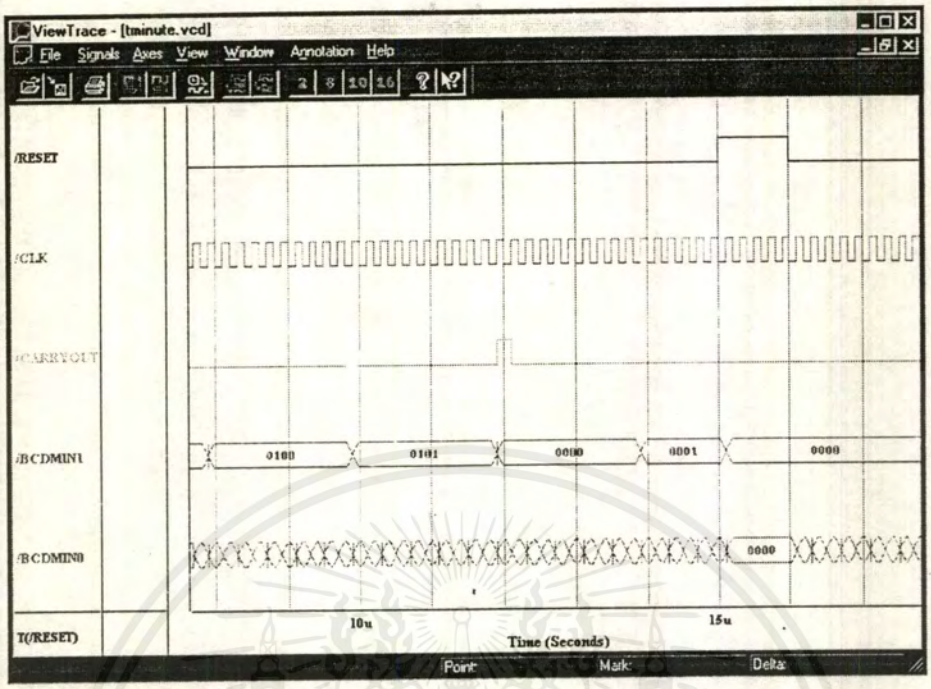
หลังจากที่เขียนโปรแกรมของวงจรตั้งเวลาขนาด 3 ช่องจนเสร็จสมบูรณ์ และทำการแปลภาษาจนไม่พบข้อผิดพลาด ขั้นตอนต่อมาจะเป็นการทดสอบการทำงานของโปรแกรมที่เขียนขึ้นมาเพื่อดูว่าการทำงานเป็นไปตามจุดประสงค์ที่ได้ออกแบบไว้ ในการจำลองการทำงานของโปรแกรมจะต้องสร้างสัญญาณทดสอบวงจรในทุกสภาวะ เพื่อให้แน่ใจว่าผลลัพธ์ที่ได้นั้นตรงตามที่ได้ออกแบบไว้ โดยการใช้โปรแกรมวิวเทรสเป็นตัวแสดงผลของสัญญาณต่างๆ ดังรูปที่ 3.9 และรูปที่ 3.10 เป็นผลของการใช้โปรแกรมวิวเทรสแสดงสัญญาณ ต่างๆ ของวงจรนับหลักชั่วโมงและหลักนาที ซึ่งการจำลองการทำงานและการใช้โปรแกรมวิวเทรสได้กล่าวไว้แล้วในหัวข้อ 2.11



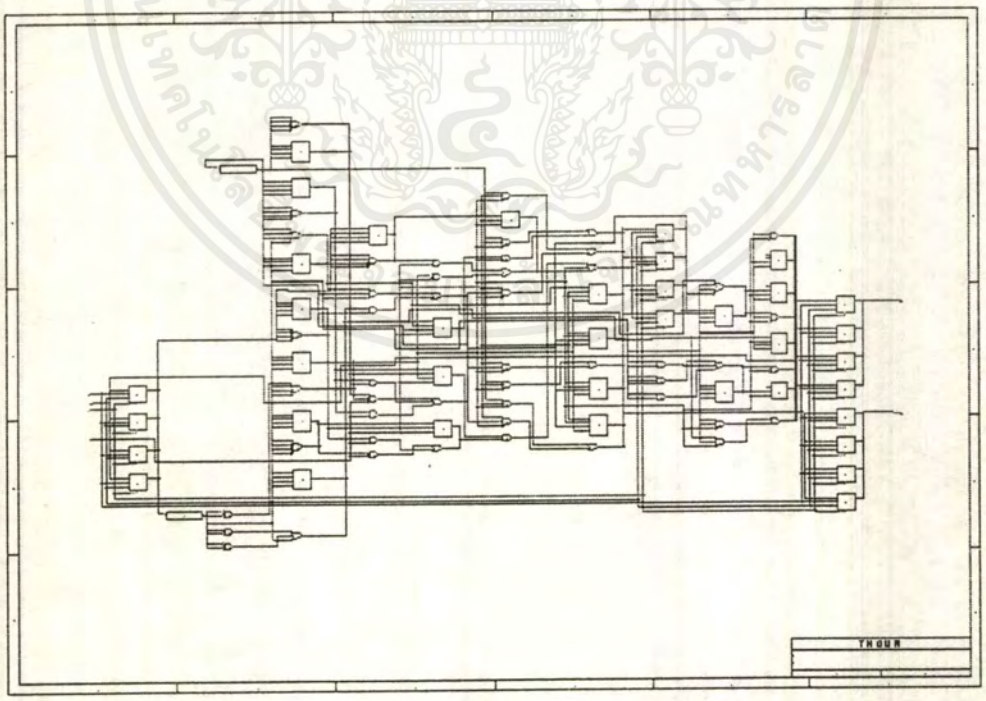
รูปที่ 3.9 การใช้โปรแกรมวิวเทรสแสดงสัญญาณต่างๆ ของวงจรนับหลักชั่วโมง

3.6.3 การสังเคราะห์เป็นวงจรระดับเกท

หลังจากจำลองผลการทำงานจนได้ผลการทำงานตรงตามจุดประสงค์ที่ออกแบบไว้แล้ว ขั้นตอนต่อมาคือ แปลงโปรแกรมของวงจรตั้งเวลาขนาด 3 ช่องให้เป็นวงจรซึ่งประกอบไปด้วยอุปกรณ์ดิจิตอลพื้นฐานจำพวกเกท และฟลิบฟลอบต่างๆ ในการสังเคราะห์เป็นวงจรระดับเกทนี้ต้องกำหนดด้วยว่าจะใช้อุปกรณ์พื้นฐานของบริษัทใด ซึ่งโครงงานนี้เลือกทำการพัฒนาวงจรที่ได้ลงบนอุปกรณ์ FPGAs เบอร์ XC4005APC84C-6 ของบริษัทไซลิงค์ ดังรูปที่ 3.11 ซึ่งเป็นวงจรระดับเกทของวงจรนับหลักชั่วโมง ส่วนขั้นตอนการสังเคราะห์วงจรระดับเกทได้กล่าวไว้ในหัวข้อที่ 2.12



รูปที่ 3.10 การใช้โปรแกรมวิวเทรสแสดงสัญญาณต่างๆ ของวงจรนับหลักนาที



รูปที่ 3.11 วงจรระดับเกทของวงจรนับหลักชั่วโมง

3.6.4 การโปรแกรมวงจรลงบนไอซี FPGAs

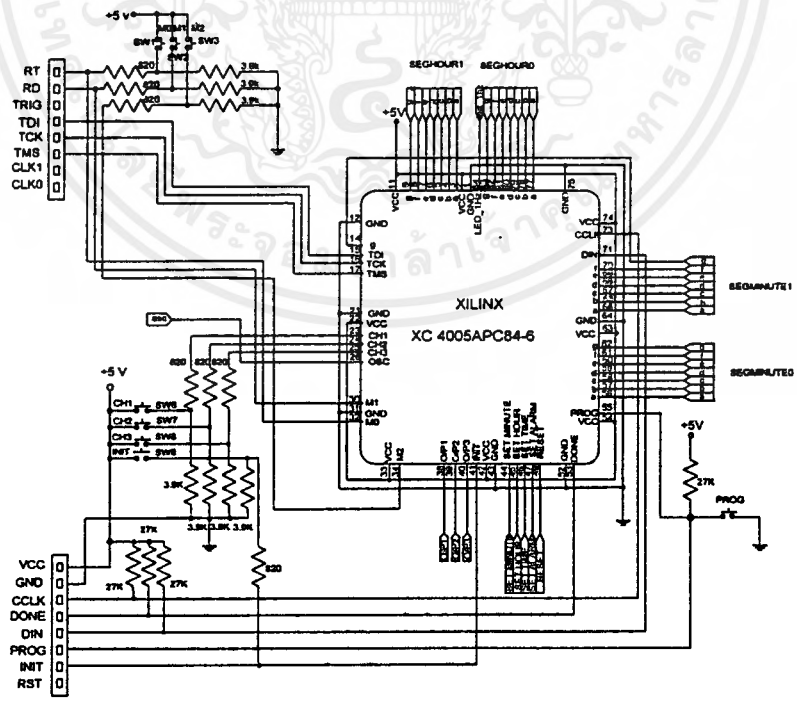
หลังจากสังเคราะห์วงจรระดับเกทของวงจรตั้งเวลาขนาด 3 ช่องแล้ว ขั้นตอนต่อมาเป็นการนำเอาวงจรที่ได้มาทำการดาวน์โหลดลงไอซี FPGAs โดยใช้ซอฟแวร์ XACT Development Tools ของบริษัทไซลิงค์ในการดาวน์โหลดวงจรตั้งเวลาขนาด 3 ช่องที่ได้ลงบนเซลล์ต่างๆ ภายในไอซี FPGAs ผลสุดท้ายจะได้ออกมาเป็นวงจรตั้งเวลาขนาด 3 ช่องบนไอซี FPGAs ซึ่งสามารถนำไปสร้างเป็นวงจรต้นแบบได้ทันที โดยขั้นตอนในการดาวน์โหลดได้กล่าวไว้แล้วในหัวข้อ 2.13

3.7 การสร้างวงจรทดสอบ

วงจรที่ใช้ในการทดสอบวงจรตั้งเวลาขนาด 3 ช่องนั้นเป็นแบบบอร์ดตัวอย่างของ FPGAs ซึ่งจะทำการโปรแกรมผ่านทางดาวน์โหลดเคเบิล (Download Cable) โดยวงจรที่ใช้ทดสอบวงจรตั้งเวลาขนาด 3 ช่องแบ่งออกเป็นวงจรส่วนต่างๆ ดังนี้

3.7.1 วงจรไอซี FPGAs

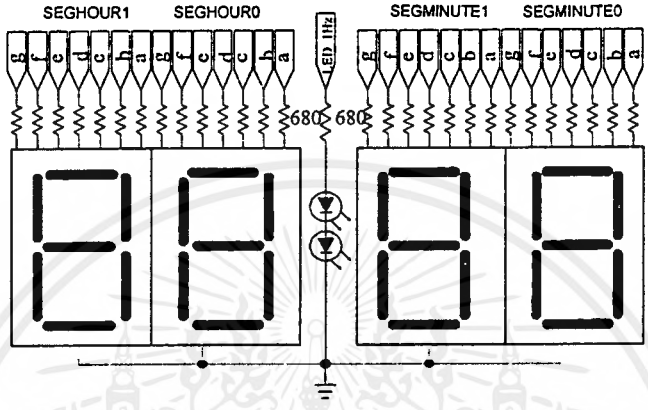
วงจรไอซี FPGAs เบอร์ XC4005APC84C-6 ทำหน้าที่ควบคุมการทำงานของวงจรตั้งเวลาขนาด 3 ช่องโดยรับโปรแกรมผ่านทางดาวน์โหลดเคเบิล



รูปที่ 3.12 วงจรไอซี FPGAs

3.7.2 วงจรภาคแสดงผล

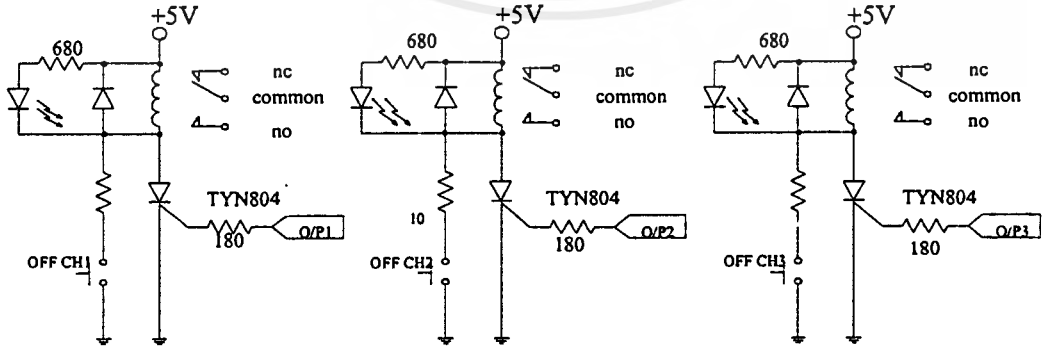
วงจรภาคแสดงผลทำหน้าที่แสดงเวลาของวงจรตั้งเวลาขนาด 3 ช่อง ผ่านทางส่วนแสดงผลแบบ 7 ส่วน ในหลักชั่วโมงและหลักนาที และแสดงผลหลักวินาทีผ่านทางหลอดไฟแสดงผลในลักษณะกระพริบ



รูปที่ 3.13 วงจรภาคแสดงผล

3.4.3 วงจรควบคุมอุปกรณ์ไฟฟ้า

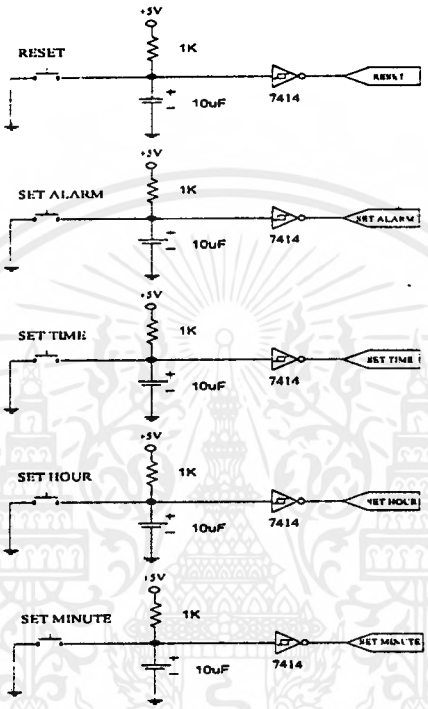
วงจรควบคุมอุปกรณ์ไฟฟ้า ทำหน้าที่ตัดต่อการใช้อุปกรณ์ไฟฟ้า เมื่อถึงเวลาที่ทำการตั้งเวลาไว้จะทำให้เอาท์พุทของวงจรเปรียบเทียบภายในวงจรตั้งเวลามีสภาวะเป็น "1" และกระตุ้นให้ SCR (Silicon Control Reactifier) นำกระแสทำให้ขดลวดของรีเลย์ครบวงจร หลอดไฟแสดงผลการทำงานจะติดสว่าง การหยุดการนำกระแสของ SCR ทำได้โดยกดสวิตช์ Off_CH ของแต่ละช่อง



รูปที่ 3.14 วงจรควบคุมอุปกรณ์ไฟฟ้า

3.4.4 วงจรสวิทต์ควบคุมการทำงาน

วงจรสวิตช์ควบคุมการทำงาน ทำหน้าที่เป็นขาอินพุทที่ใช้ในการตั้งเวลา เมื่อกดสวิตช์เอาท์พุทจากวงจรสวิตช์ตัวที่ถูกกดจะมีสถานะเป็น "1"



รูปที่ 3.15 วงจรสวิตช์ควบคุมการตั้งเวลา

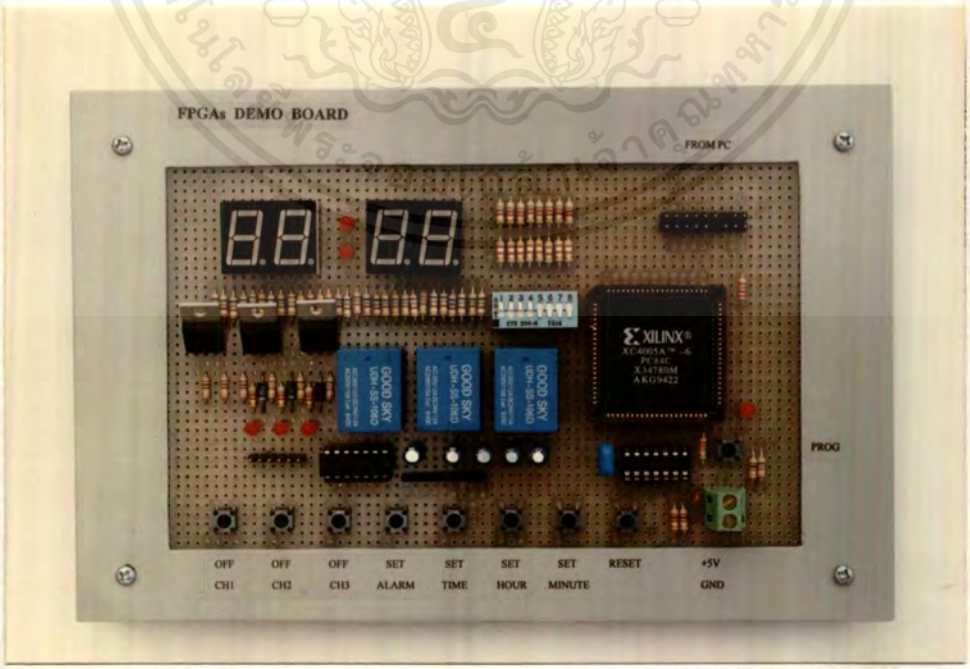
บทที่4

การทดลองและผลการทดลอง

จากการออกแบบวงจรตั้งเวลาขนาด 3 ช่องตามขั้นตอนที่กล่าวมา คือ เขียนโปรแกรมภาษา VHDLของวงจรตั้งเวลาขนาด 3 ช่อง แล้วนำไปจำลองการทำงานจนได้ผลการทำงานตามวัตถุประสงค์ที่ออกแบบ จึงนำโปรแกรมไปสังเคราะห์เป็นวงจรระดับเกท และแปลงไฟล์ที่ได้จากการสังเคราะห์เป็นไฟล์บิตสตรีม เพื่อทำการดาวน์โหลดลงบนไอซี FPGAs ปรากฏว่าวงจรตั้งเวลาขนาด 3 ช่องที่ออกแบบมีขนาดใหญ่ ไม่สามารถโปรแกรมลงบนไอซี FPGAs เบอร์ XC4005APC84C-6 ได้ ซึ่งคณะผู้จัดทำไม่มีไอซี FPGAs เบอร์ที่มีจำนวนเกทมากกว่านี้ จึงต้องลดจำนวนช่องของวงจรตั้งเวลาให้เหลือ 2 ช่อง เพื่อให้สามารถโปรแกรมลงไอซี FPGAs เบอร์ XC4005APC84C-6ได้ เนื้อหาในบทนี้จึงเป็นการทดลองวงจรตั้งเวลาขนาด 2 ช่องบนบอร์ดตัวอย่างของ FPGAs โดยมีขั้นตอนและผลการทดลองดังนี้

4.1 การดาวน์โหลดโปรแกรมลงบนบอร์ดตัวอย่างของ FPGAs

บอร์ดตัวอย่างของ FPGAs แสดงดังรูปที่ 4.1 ใช้ไอซี FPGAs เบอร์ XC4005APC84C-6 ควบคุมการทำงาน โดยสามารถใช้งานเป็นของวงจรตั้งเวลาได้ตั้งแต่ 1 ช่องถึง 3 ช่อง แต่ในการทดลองจะใช้วงจรตั้งเวลาเพียง 2 ช่อง เนื่องจากเหตุผลที่ได้กล่าวมาแล้ว

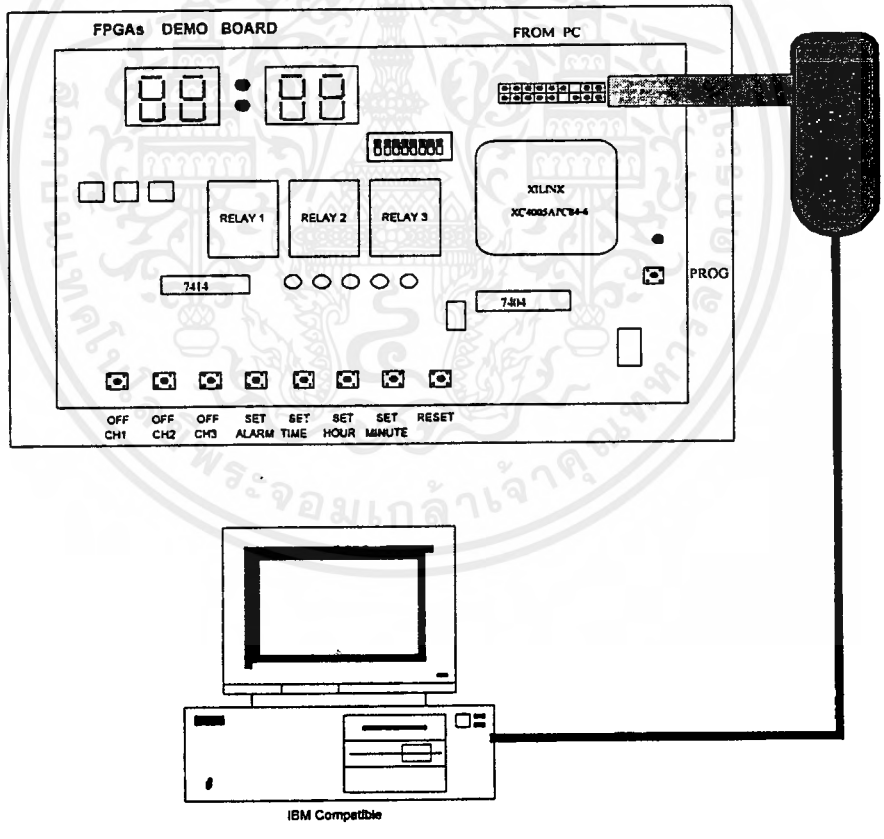


รูปที่ 4.1 ภาพถ่ายบอร์ดตัวอย่างของ FPGAs

เมื่อดำเนินการออกแบบตามขั้นตอนในบทที่ 3 จนได้ไฟล์ Aclock.bit ของวงจรตั้งเวลาขนาด 2 ช่องแล้ว จึงทำการทดสอบการทำงานของโปรแกรมและบอร์ดตัวอย่างของ FPGAs โดยการดาวน์โหลดไฟล์ Aclock.bit ลงบนบอร์ดตัวอย่างของ FPGAs เมื่อต่อสายดาวน์โหลดเคเบิลจากพอร์ต COM2 เข้ากับบอร์ดตัวอย่างของ FPGAs ดังแสดงในรูปที่ 4.2 แล้ว สามารถทำการดาวน์โหลดไฟล์บิตสตรีมได้ด้วยขั้นตอนต่อไปนี้

ขั้นตอนที่1 เข้าสู่โปรแกรม XDM แล้วคลิกเมาส์ที่เมนู Tools และเลือกที่คำสั่ง Hardware Debugger ซึ่งจะแสดงหน้าต่าง Communication Setup ดังรูปที่ 2.78 ในบทที่ 2

ขั้นตอนที่ 2 กำหนดค่าอัตราความเร็วในการส่งข้อมูล (Baud Rate) และเลือกพอร์ทที่จะส่งข้อมูลออกไป ในโครงงานนี้เลือกอัตราความเร็วในการส่งข้อมูลที่ 9600 Bit/Sec และเลือกส่งข้อมูลออกที่พอร์ต COM2



รูปที่ 4.2 การเชื่อมต่อกันระหว่างคอมพิวเตอร์กับบอร์ดตัวอย่างของ FPGAs

ขั้นตอนที่ 3 ตั้งดิพสวิตช์บอร์ดตัวอย่างของ FPGAs ให้ใช้งานลักษณะสเลฟซีเรียล ซึ่งสามารถตั้งดิพสวิตช์ได้ดังนี้ คือ

Dip SW	LABEL	SETTING
1	M0	ON
2	M1	ON
3	M2	ON
4	INIT	ON

ตารางที่ 4.1 การตั้งดิพสวิตช์ต่างๆ ในบอร์ดตัวอย่างของ FPGAs

ขั้นตอนที่ 4 เมื่อกำหนดอัตราความเร็วในการส่งข้อมูล และเลือกพอร์ตที่จะส่งข้อมูลออกไปแล้วให้ตอบตกลงจะกลับสู่หน้าต่าง Hardware Debugger ให้เลือกที่เมนู Download และเลือกที่ Download Design ซึ่งโปรแกรม Debugger จะทำการดาวน์โหลดไฟล์ Aclock.bit ลงสู่บอร์ดตัวอย่างของ FPGAs เมื่อทำการดาวน์โหลดเสร็จจึงนำบอร์ดตัวอย่างของ FPGAs ไปทดลองได้

4.2 การทดลองวงจรตั้งเวลาขนาด 2 ช่อง

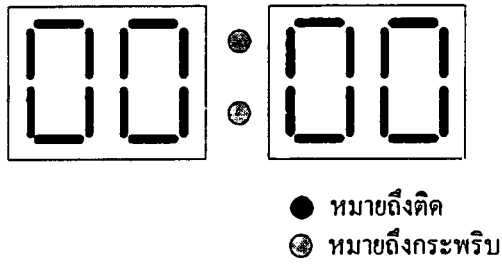
การทดลองวงจรตั้งเวลาขนาด 2 ช่อง มีขั้นตอนการทดลองดังนี้

ขั้นตอนที่1 จ่ายไฟเลี้ยง +5V ให้กับบอร์ดตัวอย่างของ FPGAs จะทำให้หลอดไฟแสดงผลสีแดงที่อยู่ใกล้กับสวิทช์ PROG ติดสว่าง

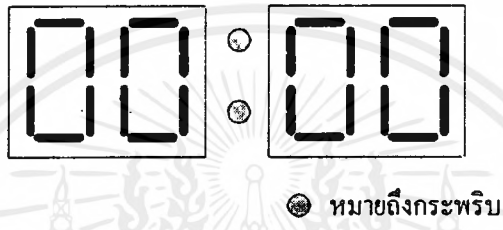
ขั้นตอนที่2 ทำการดาวน์โหลดไฟล์ Aclock.bit ลงบนบอร์ดตัวอย่างของ FPGAs ตามวิธีการในหัวข้อที่ 4.1

ขั้นตอนที่ 3 เมื่อทำการดาวน์โหลดเสร็จเรียบร้อยแล้ว ส่วนแสดงผลแบบ 7 ส่วนทั้ง 4 หลัก จะแสดงเวลาเริ่มต้นการทำงานเป็น 00:00 ดังรูปที่ 4.3

ขั้นตอนที่ 4 หลังจากส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเป็น 00:00 แล้ว นาฬิกาจะทำงานทันทีที่เสร็จสิ้นการดาวน์โหลด เมื่อเวลาผ่านไปครบ 1 นาทีส่วนแสดงผลแบบ 7 ส่วนจะแสดงเวลาเป็น 00:01 ดังรูปที่ 4.4

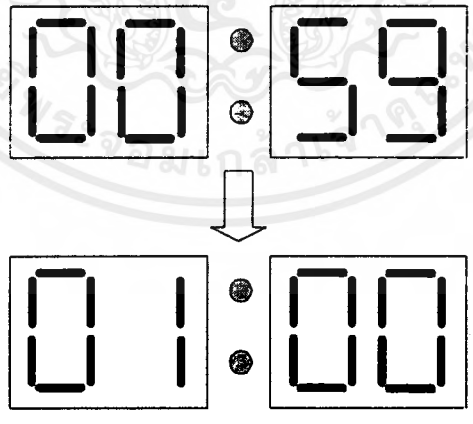


รูปที่ 4.3 เวลาเริ่มต้นแสดงการทำงานหลังจากทำการดาวน์โหลดเสร็จสิ้น



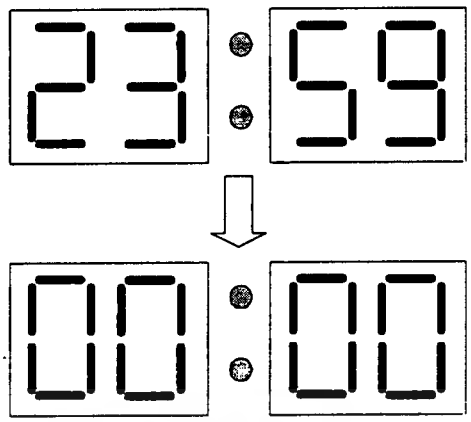
รูปที่ 4.4 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อผ่านไป 1 นาที

ขั้นตอนที่ 5 เมื่อนาฬิกาเดินถึงเวลา 59 นาทีส่วนแสดงผลแบบ 7 ส่วนจะแสดงเวลาเป็น 00:59 และจะแสดงเวลาเปลี่ยนจาก 00:59 เป็น 01:00 เมื่อเวลาเดินครบ 1 ชั่วโมง ดังรูปที่ 4.5



รูปที่ 4.5 ส่วนแสดงผลแบบ 7 ส่วนแสดงผลเมื่อเวลาเปลี่ยนจาก 59 นาทีเป็น 1 ชั่วโมง

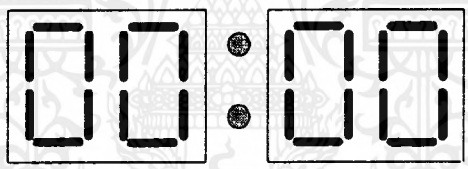
ขั้นตอนที่ 6 เมื่อเวลาผ่านไป 23 ชั่วโมง 59 นาทีส่วนแสดงแบบ 7 ส่วนจะแสดงเวลาเป็น 23:59 และจะวนกลับมาเป็น 00:00 เมื่อเวลาเดินครบ 24 ชั่วโมง ดังรูปที่ 4.6



รูปที่ 4.6 ส่วนแสดงผลแบบ 7 ส่วนแสดงผลเมื่อเวลาเปลี่ยนจาก 23:59 เป็น 00:00

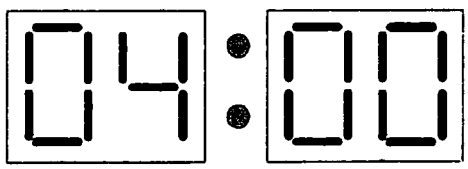
ขั้นตอนที่ 7 การทดลองตั้งเวลาวงจรนาฬิกาเป็น 4 ชั่วโมง 15 นาทีสามารถทำได้ดังนี้

1) เมื่อทำการดาวน์โหลดไฟล์บิตสตรีมลงบนบอร์ดตัวอย่างของ FPGAs เสร็จแล้ว ส่วนแสดงผลแบบ 7 ส่วนจะแสดงเวลาเริ่มต้นเป็น 00:00 ดังรูปที่ 4.7



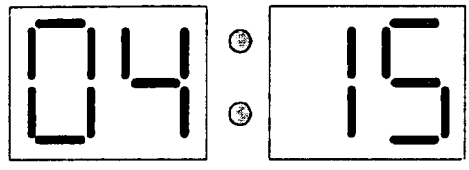
รูปที่ 4.7 ส่วนแสดงผลแบบ 7 ส่วนแสดงผลเวลาเริ่มต้นที่ 00:00

2) กดสวิตซ์ SET TIME และสวิตซ์ SET HOUR พร้อมกันเพื่อตั้งเวลาในหลักชั่วโมง เวลาในหลักชั่วโมงจะเพิ่มขึ้นครั้งละ 1 ชั่วโมงไปเรื่อยๆ จนได้เวลาที่ต้องการจึงปล่อยสวิตซ์ SET TIME และ SET HOUR ที่เวลา 04:00 ดังรูปที่ 4.8



รูปที่ 4.8 การตั้งเวลาในหลักชั่วโมงที่เวลา 04:00

3) กดสวิตช์ SET TIME และ SET MINUTE พร้อมกันเพื่อตั้งเวลาในหลักนาที เวลาในหลักนาทีจะเพิ่มขึ้นครั้งละ 1 นาทีไปเรื่อยๆ จนถึงเวลาที่ต้องการจึงปล่อยสวิตช์ SET TIME และ SET MINUTE ที่เวลา 04:15 ดังรูปที่ 4.9



รูปที่ 4.9 การตั้งเวลาของวงจรนาฬิกาที่เวลา 04:15

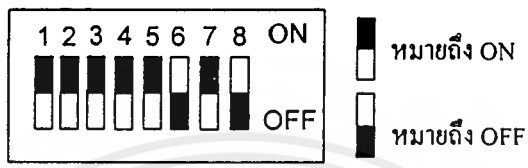
4) หลังจากตั้งเวลา 4 ชั่วโมง 15 นาทีเรียบร้อยแล้ว เวลาจะเดินไปเรื่อยๆ จนถึง 23 ชั่วโมง 59 นาที 59 วินาที ก็จะวนกลับมาเริ่มที่เวลา 00:00 ดังรูปที่ 4.10

รูปที่ 4.10 เวลาเดินจาก 04:15 ไปจนถึง 00:00

ขั้นตอนที่ 8 การทดลองตั้งเวลาของวงจรตั้งเวลาช่องที่ 1 เป็น 3 ชั่วโมง 20 นาที สามารถทำได้ดังนี้

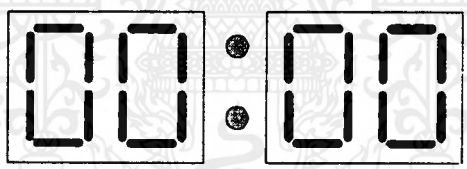
1) เมื่อดาวน์โหลดไฟล์ Aclock.bit ลงบนบอร์ดตัวอย่างของ FPGAs เรียบร้อยแล้ว เวลาของวงจรตั้งเวลาช่องที่ 1 จะมีเวลาเริ่มต้นที่ 00:00

2) เลือกช่องตั้งเวลาเป็นช่องที่ 1 โดยเลื่อนดิพสวิตซ์ตัวที่ 7 ให้อยู่ในตำแหน่ง ON และเลื่อนดิพสวิตซ์ตัวที่ 8 ให้อยู่ในตำแหน่ง OFF ดังรูปที่ 4.11



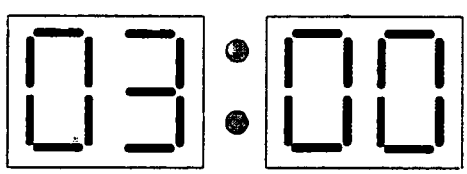
รูปที่ 4.11 ตำแหน่งของดิพสวิตซ์ตัวที่ 7 และ 8 ในการเลือกวงจรตั้งเวลาช่องที่ 1

3) กดสวิตซ์ SET ALARM จะทำให้ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาของวงจรตั้งเวลาช่องที่ 1 เป็น 00:00 ดังรูปที่ 4.12



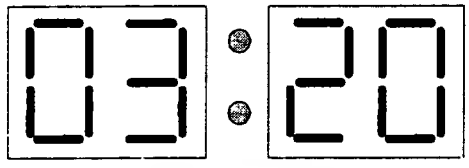
รูปที่ 4.12 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเริ่มต้นของวงจรตั้งเวลาช่องที่ 1

4) กดสวิตซ์ SET ALARM พร้อมกับสวิตซ์ SET HOUR เพื่อตั้งเวลาในหลักชั่วโมง เวลาบนส่วนแสดงผลแบบ 7 ส่วนจะเพิ่มขึ้นครั้งละ 1 ชั่วโมงไปเรื่อยๆ จนถึงเวลาที่ต้องการ ซึ่งเป็นเวลา 03:00 ดังรูปที่ 4.13 จึงปล่อยสวิตซ์ทั้งสองตัว



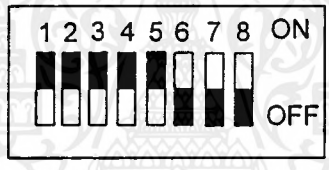
รูปที่ 4.13 การตั้งเวลาหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 1

5) กดสวิตซ์ SET ALARM พร้อมกับสวิตซ์ SET MINUTE เพื่อทำการตั้งเวลาในหลักนาที เวลาบนส่วนแสดงผลแบบ 7 ส่วนจะเพิ่มขึ้นครั้งละ 1 นาทีไปเรื่อยๆ จนถึงเวลาที่ต้องการ ซึ่งเป็นเวลา 03:20 ดังรูปที่ 4.14 จึงปล่อยสวิตซ์ทั้งสองตัว



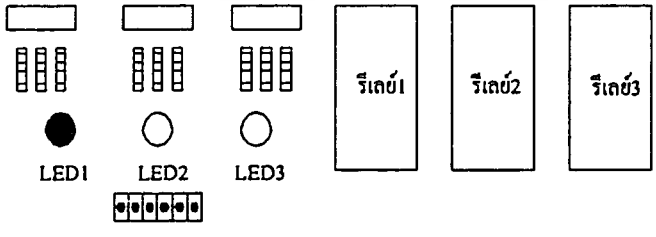
รูปที่ 4.14 การตั้งเวลาหลักนาทีของวงจรตั้งเวลาช่องที่ 1

6) เมื่อตั้งเวลาเสร็จเรียบร้อยแล้วให้เลื่อนดิพสวิตซ์ตัวที่ 7 ไปที่ตำแหน่ง OFF ดังรูปที่ 4.15



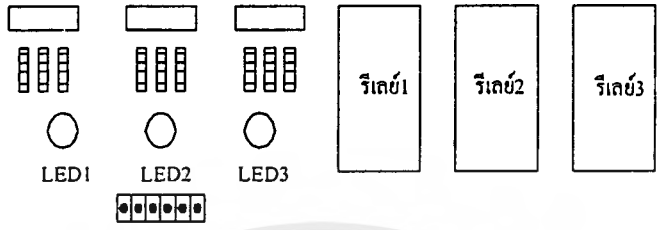
รูปที่ 4.15 ตำแหน่งของดิพสวิตซ์ตัวที่ 7 และ 8 เมื่อตั้งเวลาช่องที่ 1 เสร็จ

7) เมื่อตั้งเวลาของวงจรตั้งเวลาช่องที่ 1 เรียบร้อยแล้ว นาฬิกาเดินถึงเวลา 03:20 ซึ่งเท่ากับเวลาของวงจรตั้งเวลาช่องที่ 1 วงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 จะทำงานหลอดไฟแสดงผลของช่องที่ 1 ติดสว่าง และได้ยินเสียงรีเลย์ 1 ทำงาน ดังรูปที่ 4.16



รูปที่ 4.16 หลอดไฟแสดงผลและรีเลย์ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 1 ทำงาน

8) เมื่อต้องการหยุดการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 สามารถทำได้โดยการกดสวิตซ์ OFF CH1 รีเลย์ 1 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1จะหยุดทำงานและหลอดไฟแสดงผลจะดับ ดังรูปที่ 4.17



รูปที่ 4.17 รีเลย์ 1 และหลอดไฟแสดงผลของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 1 หยุดทำงานเมื่อกดสวิตซ์ OFF CH1

ขั้นตอนที่ 9 การทดลองวงจรตั้งเวลาขนาด 2 ช่อง

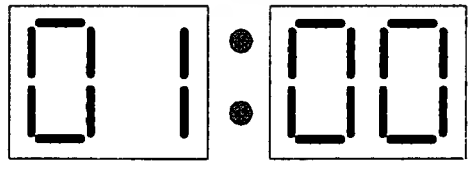
1) กำหนดการตั้งเวลาของวงจรตั้งเวลาแต่ละช่องเป็นดังนี้

ช่องที่ 1 ตั้งเวลาที่ 1 ชั่วโมง 15 นาที

ช่องที่ 2 ตั้งเวลาที่ 2 ชั่วโมง 30 นาที

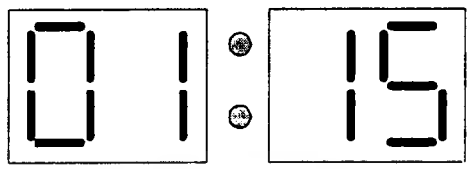
2) เลือกช่องตั้งเวลาช่องที่ 1 ตามขั้นตอนที่ 8.2

3) กดสวิตซ์ SET ALARM จะทำให้ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาของวงจรตั้งเวลาช่องที่ 1พร้อมกับกดสวิตซ์ SET HOUR ค้างไว้เพื่อทำการตั้งเวลาในหลักชั่วโมง เวลาบนส่วนแสดงผลแบบ 7 ส่วนจะเพิ่มขึ้นครั้งละ 1 ชั่วโมงไปเรื่อยๆ จนถึงเวลาที่ต้องการ ซึ่งเป็นเวลา 01:00 ดังรูปที่ 4.18 จึงปล่อยสวิตซ์ทั้งสองตัว



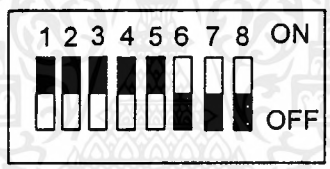
รูปที่ 4.18 การตั้งเวลาหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 1

4) กดสวิตซ์ SET ALARM พร้อมกับสวิตซ์ SET MINUTE ค้างไว้เพื่อทำการตั้งเวลาในหลักนาที เวลาบนส่วนแสดงผลแบบ 7 ส่วนจะเพิ่มขึ้นครั้งละ 1 นาทีไปเรื่อยๆ จนถึงเวลาที่ต้องการ ซึ่งเป็นเวลา 01:15 ดังรูปที่ 4.19 จึงปล่อยสวิตซ์ทั้งสองตัว



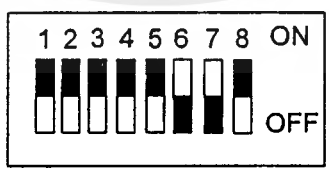
รูปที่ 4.19 การตั้งเวลาของวงจรตั้งเวลาช่องที่ 1

5) เมื่อตั้งเวลาเสร็จเรียบร้อยแล้วให้เลื่อนดิพสวิตซ์ตัวที่ 7 ไปที่ตำแหน่ง OFF ดังรูปที่ 4.20



รูปที่ 4.20 ตำแหน่งของดิพสวิตซ์ตัวที่ 7 และ 8 เมื่อตั้งเวลาช่องที่ 1 เสร็จ

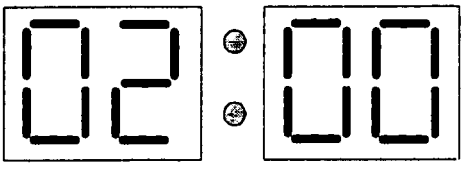
6) ทำการตั้งเวลาในช่องที่ 2 โดยเลื่อนดิพสวิตซ์ตัวที่ 8 ให้อยู่ในตำแหน่ง ON และเลื่อนดิพสวิตซ์ ตัวที่ 7 ให้อยู่ในตำแหน่ง OFF ดังรูปที่ 4.21



รูปที่ 4.21 ตำแหน่งของดิพสวิตซ์ตัวที่ 7 และ 8 ในการเลือกวงจรตั้งเวลาช่องที่ 2

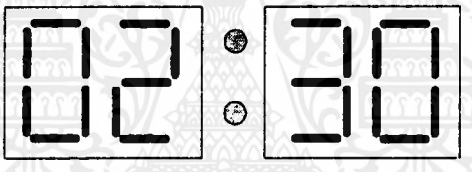
7) กดสวิตซ์ SET ALARM จะทำให้ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาของวงจรตั้งเวลาช่องที่ 2 พร้อมกับกดสวิตซ์ SET HOUR ค้างไว้เพื่อทำการตั้งเวลาในหลักชั่วโมง

เวลาบนส่วนแสดงผลแบบ 7 ส่วนจะเพิ่มขึ้นครั้งละ 1 ชั่วโมงไปเรื่อยๆ จนถึงเวลาที่ต้องการ ซึ่งเป็นเวลา 02:00 ดัง รูปที่ 4.22 จึงปล่อยสวิตซ์ทั้งสองตัว



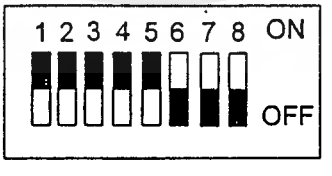
รูปที่ 4.22 การตั้งเวลาหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 2

8) กดสวิตซ์ SET ALARM พร้อมกับสวิตซ์ SET MINUTE ค้างไว้เพื่อทำการตั้งเวลาในหลักนาที เวลาบนส่วนแสดงผลแบบ 7 ส่วนจะเพิ่มขึ้นครั้งละ 1 นาทีไปเรื่อยๆ จนถึงเวลาที่ต้องการ ซึ่งเป็นเวลา 02:30 ดังรูปที่ 4.23 จึงปล่อยสวิตซ์ทั้งสองตัว



รูปที่ 4.23 การตั้งเวลาของวงจรตั้งเวลาช่องที่ 2

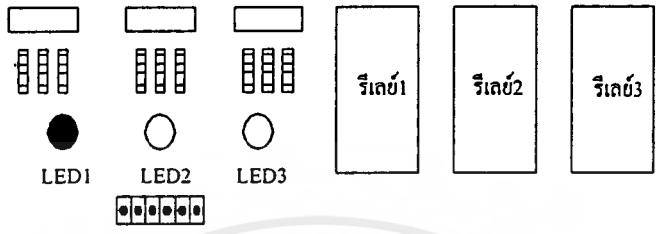
9) เมื่อตั้งเวลาช่องที่ 2 เสร็จเรียบร้อยแล้วให้เลื่อนดิพสวิตช์ตัวที่ 8 ไปที่ตำแหน่ง OFF ดังรูปที่ 4.24



รูปที่ 4.24 ตำแหน่งของดิพสวิตซ์ตัวที่ 8 เมื่อตั้งเวลาช่องที่ 2 เสร็จ

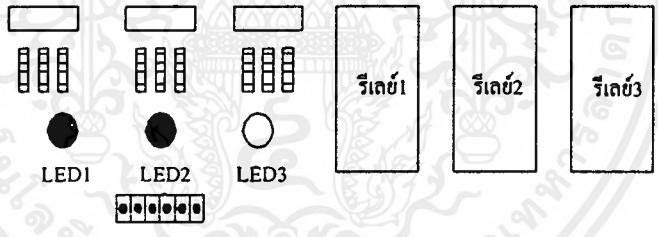
10) ทันทีที่ปล่อยมือจากสวิตซ์ SET ALARM ส่วนแสดงผลแบบ 7 ส่วนจะแสดงเวลาของวงจรนาฬิกาแทน

11) เมื่อเวลาของวงจรนาฬิกาเดินถึงเวลา 01:15 ซึ่งเท่ากับเวลาของวงจรตั้งเวลาช่องที่ 1จะทำให้วงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 ทำงาน หลอดไฟแสดงผลช่องที่ 1 ติดสว่าง และได้ยินเสียงรีเลย์ 1 ทำงาน ดังรูปที่ 4.25



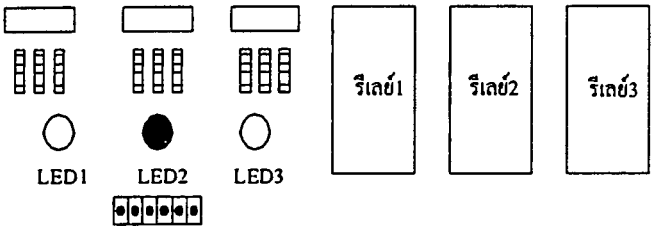
รูปที่ 4.25 หลอดไฟแสดงผลและรีเลย์ 1 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 1 ทำงาน

12) เมื่อเวลาของวงจรนาฬิกาเดินถึงเวลา 02:30 ซึ่งเท่ากับเวลาของวงจรตั้งเวลาช่องที่ 2 จะทำให้วงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2 ทำงาน หลอดไฟแสดงผลช่องที่ 2 ติดสว่าง ดังรูปที่ 4.26



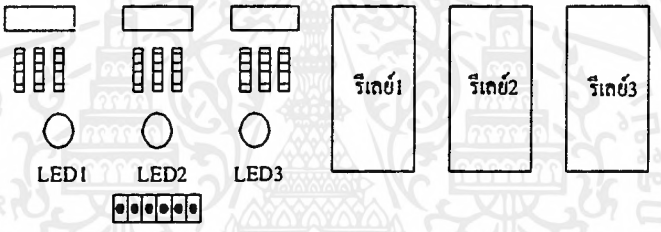
รูปที่ 4.26 หลอดไฟแสดงผลและรีเลย์ 2 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่อง 2 ทำงาน

13) เมื่อต้องการหยุดการทำงานของวงจรตั้งเวลาช่องที่ 1 สามารถทำได้โดยการกดสวิตซ์ OFF CH1 รีเลย์ 1 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 หยุดการทำงานและหลอดไฟแสดงผลจะดับ ดังรูปที่ 4.27



รูปที่ 4.27 หลอดไฟแสดงผลและรีเลย์ 1 หยุดทำงานเมื่อกดสวิตซ์ OFF CH1

14) เมื่อต้องการหยุดการทำงานของวงจรตั้งเวลาช่องที่ 1 สามารถทำได้โดยการกดสวิตซ์ OFF CH2 รีเลย์ 2 ของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2 หยุดการทำงานและหลอดไฟแสดงผลจะดับ ดังรูปที่ 4.28



รูปที่ 4.28 หลอดไฟแสดงผลและรีเลย์ 2 หยุดทำงานเมื่อกดสวิตซ์ OFF CH2

ขั้นตอนที่ 10 ทดลองตั้งเวลาให้กับวงจรตั้งเวลา 2 ช่องใหม่ตามขั้นตอนที่ 9 โดยกำหนดเวลาใหม่เป็น

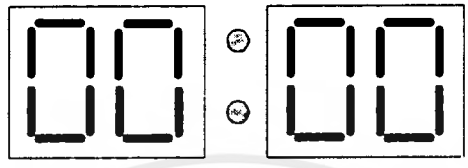
ช่องที่ 1 ตั้งเวลาที่ 15 ชั่วโมง 15 นาที

ช่องที่ 2 ตั้งเวลาที่ 10 ชั่วโมง 30 นาที

ผลปรากฏว่า เมื่อเวลาของวงจรนาฬิกาเท่ากับ เวลาของวงจรตั้งเวลาในช่องใด ก็จะทำให้วงจรควบคุมอุปกรณ์ไฟฟ้าในช่องนั้นทำงาน และถ้าเราต้องการให้วงจรควบคุมอุปกรณ์ไฟฟ้าหยุดการทำงานก็ทำการกดสวิตซ์ OFF CH ของวงจรควบคุมอุปกรณ์ไฟฟ้าของแต่ละช่อง

ขั้นตอนที่ 11 ทดลองตั้งเวลาทั้ง 2 ช่องให้ทำงานพร้อมกันที่เวลา 7 ชั่วโมง 7 นาที ผลปรากฏว่าเ มื่อเวลาของวงจรนาฬิกาเดินถึงเวลา 7 ชั่วโมง 7 นาที วงจรควบคุมอุปกรณ์ไฟฟ้าทั้ง 2 ช่องจะทำงานพร้อมกัน และถ้าเราต้องให้วงจรควบคุมอุปกรณ์ไฟฟ้าหยุดการทำงาน ก็ทำการกดสวิตซ์ OFF CH ของวงจรควบคุมอุปกรณ์ไฟฟ้าของแต่ละช่อง

ขั้นตอนที่12 ทดลองตั้งเวลาพร้อมกัน 2 ช่องโดยให้ดิพสวิตซ์ตัวที่ 7 และ 8 อยู่ในตำแหน่ง ON ทั้ง 2 ตัว ผลปรากฏว่าไม่สามารถตั้งเวลาได้ เมื่อกดสวิตซ์ SET ALARM ส่วนแสดงผลแบบ 7 ส่วน จะแสดงเวลาเป็น 00:00 ดังรูปที่ 4.29 เมื่อกดสวิตซ์ SET ALARM หรือ SET TIME จะไม่เกิดการเปลี่ยนแปลงใดๆ กับวงจรตั้งเวลา

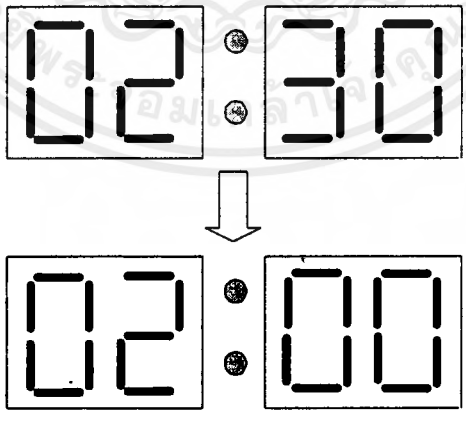


รูปที่ 4.29 ส่วนแสดงผลแบบ 7 ส่วนแสดงเวลาเมื่อตั้งเวลา 2 ช่องพร้อมกัน

ขั้นตอนที่ 13 ทดลองตั้งเวลาโดยให้ดิพสวิตซ์ตัวที่ 7 และ 8 อยู่ในตำแหน่ง OFF ทั้ง 2 ตัว ผลปรากฏว่าได้ผลเช่นเดียวกับข้อที่ 12

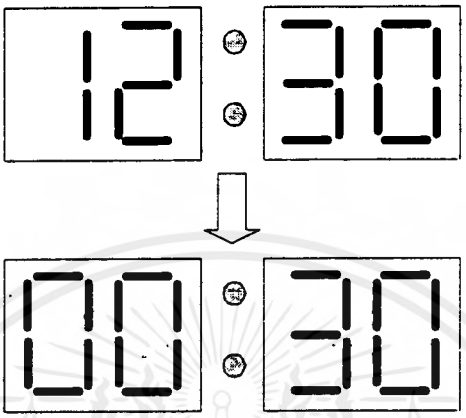
ขั้นตอนที่ 14 การทดลองรีเซ็ทเวลาของวงจรนาฬิกา สามารถทำได้ดังนี้

1) ทดลองรีเซ็ทเวลาของวงจรนาฬิกาในหลักนาทีโดยการกดสวิตซ์ RESET SET TIME และสวิตซ์ SET MINUTE พร้อมกัน เวลาในหลักนาทีของวงจรนาฬิกาจะเป็น 00 ดังรูปที่ 4.30



รูปที่ 4.30 การรีเซ็ทเวลาของวงจรนาฬิกาหลักนาทีให้เป็น 00

2) ทดลองรีเซ็ทเวลาของวงจรนาฬิกาในหลักชั่วโมง โดยกดสวิตช์ RESET SET TIME และสวิตช์ SET HOUR พร้อมกัน เวลาในหลักชั่วโมงของวงจรนาฬิกาจะเป็น 00 ดังรูปที่ 4.31

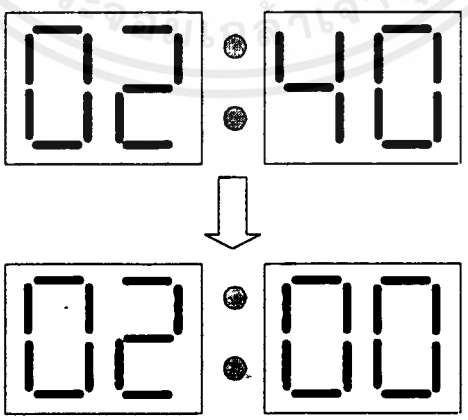


รูปที่ 4.31 การรีเซ็ทเวลาของวงจรนาฬิกาหลักชั่วโมงให้เป็น 00

ขั้นตอนที่ 15 การทดลองรีเซ็ทเวลาของวงจรตั้งเวลา สามารถทำได้ดังนี้

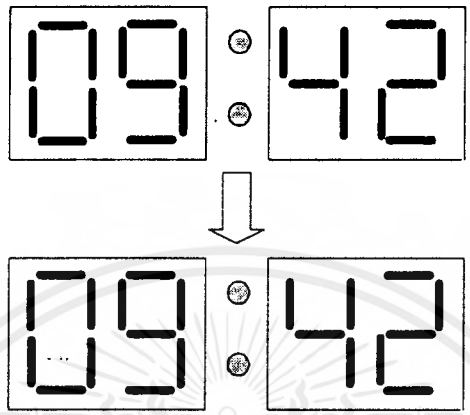
1) เลื่อนดิพสวิตช์ตัวที่ 7 ให้อยู่ในตำแหน่ง ON และตัวที่ 8 อยู่ในตำแหน่ง OFF

2) ทดลองรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 1ในหลักนาที โดยการกดสวิตช์ RESET, SET ALARM และสวิตช์ SET MINUTE พร้อมกัน เวลาในหลักนาทีของวงจรตั้งเวลาช่องที่ 1จะเป็น 00 ดังรูปที่ 4.32



รูปที่ 4.32 การรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 1 หลักนาทีให้เป็น 00

3) ทดลองรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 1 ในหลักชั่วโมง โดยกดสวิตช์ RESET, SET ALARM และสวิตช์ SET HOUR พร้อมกัน เวลาในหลักชั่วโมงของวงจรตั้งเวลาช่องที่ 1จะเป็น 00 ดังรูปที่ 4.33



รูปที่ 4.33 การรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 1 หลักชั่วโมงให้เป็น 00

ขั้นตอนที่ 16 การทดลองรีเซ็ทเวลาของวงจรตั้งเวลาช่องที่ 2 ทำได้เช่นเดียวกับวงจรตั้งเวลาช่องที่ 1 โดยเลื่อนดิพสวิตช์ตัวที่ 7 ให้อยู่ในตำแหน่ง OFF และตัวที่ 8 อยู่ในตำแหน่ง ON

ขั้นตอนที่17 ทดลองกดสวิตช์ OFF CH1 จะทำให้วงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 ทำงาน หลอดไฟแสดงผลจะติดสว่าง แต่เมื่อปล่อยสวิตช์ OFF CH1 วงจรควบคุมการทำงานของอุปกรณ์ไฟฟ้าจะหยุดทำงานและหลอดไฟแสดงผลจะดับ ทดลองกับสวิตช์ OFF CH2 และ OFF CH3 ก็ได้ผลการทดลองที่ตรงกัน

ขั้นตอนที่ 18 ทดลองกดสวิตช์ PROG ผลปรากฏว่าวงจรตั้งเวลาขนาด 2 ช่องจะหยุดทำงานและส่วนแสดงผลแบบ 7 ส่วนจะดับ ต้องทำการดาวน์โหลดโปรแกรมใหม่ วงจรตั้งเวลาขนาด 2 ช่องจึงจะทำงานได้ดังเดิม

ขั้นตอนที่ 19 ทดลองนำไฟเลี้ยง +5 V ออกจากบอร์ดตัวอย่างของ FPGAs และ จ่ายไฟเลี้ยงให้วงจรใหม่ปรากฏว่า โปรแกรมที่ทำการดาวน์โหลดไว้ถูกลบไปต้องทำการดาวน์โหลดโปรแกรมใหม่วงจร ตั้งเวลาขนาด 2 ช่องจึงจะทำงานได้ดังเดิม

ขั้นตอนที่ 20 ทดลองกดสวิตช์ SET TIME, SET HOUR และ SET MINUTE พร้อมกัน ผลปรากฏว่าไม่มีการเปลี่ยนแปลงใดๆ เกิดขึ้น

ขั้นตอนที่ 21 ทดลองกดสวิตช์ SET ALARM, SET HOUR และ SET MINUTE พร้อมกัน ผลปรากฏว่าไม่มีการเปลี่ยนแปลงใดๆ เกิดขึ้น

ขั้นตอนที่ 22 ทดลองกดสวิตช์ SET ALARM และ SET TIME พร้อมกัน ผลปรากฏว่าไม่มีการเปลี่ยนแปลงใดๆ เกิดขึ้น

ขั้นตอนที่ 23 ทดลองเทียบเวลาของวงจรตั้งเวลาขนาด 2 ช่องกับเวลาของนาฬิกาดิจิตอล สามารถทำได้ดังนี้

1) รีเซ็ทเวลาของวงจรตั้งเวลาขนาด 2 ช่อง และนาฬิกาดิจิตอลที่นำมาเปรียบเทียบให้เวลาเริ่มต้นที่ 00:00

2) เปรียบเทียบเวลาของวงจรตั้งเวลาขนาด 2 ช่อง และเวลาของนาฬิกาดิจิตอล โดยบันทึกเวลาของนาฬิกาทั้งสองตัวทุก 10 นาที ได้ผลการบันทึกดังตารางที่ 4.2

เวลาของนาฬิกาดิจิตอล	เวลาของวงจรตั้งเวลาขนาด 2 ช่อง
10	8
20	17
30	25
40	33
50	41
60	50

ตารางที่ 4.2 ผลการเปรียบเทียบเวลาของวงจรตั้งเวลาขนาด 2 ช่องกับนาฬิกาดิจิตอล

สำหรับภาพถ่ายการเชื่อมต่อสายดาวน์โหลดเคเบิลกับบอร์ดตัวอย่างของ FPGAs แสดงไว้ดังรูปที่ 4.34

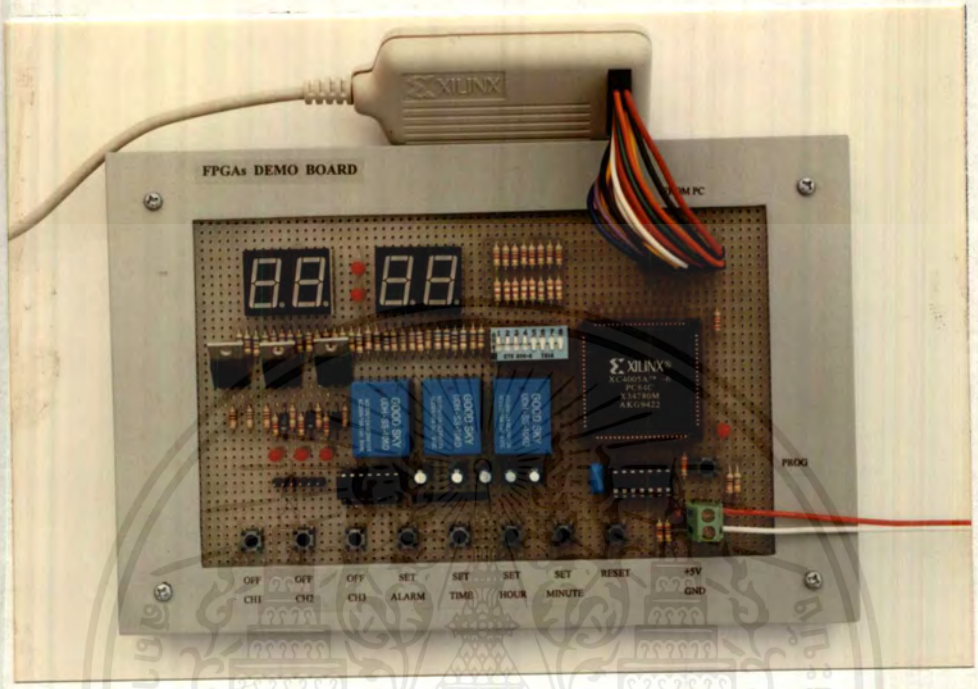
สำหรับภาพถ่ายแสดงเวลาเริ่มต้นการทำงานของวงจรตั้งเวลาขนาด 2 ช่อง เมื่อสิ้นสุดการดาวน์โหลด แสดงไว้ดังรูปที่ 4.35

สำหรับภาพถ่ายแสดงเวลาของวงจรนาฬิกาหลังจากเวลาผ่านไป 59 นาที แสดงไว้ดังรูปที่ 4.36

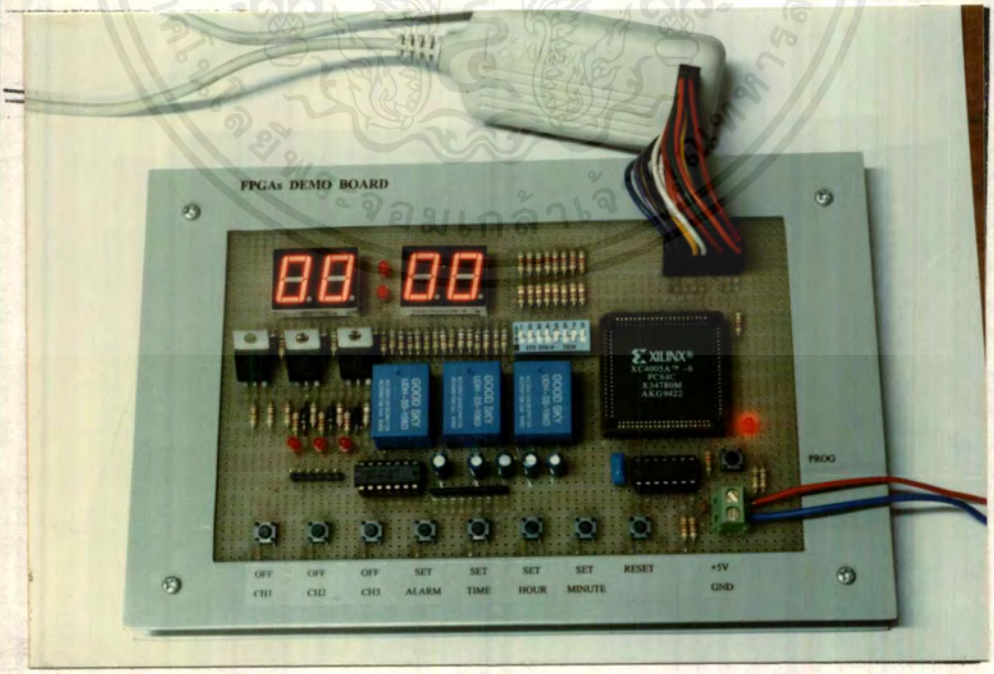
สำหรับภาพถ่ายแสดงเวลาของวงจรนาฬิกาหลังจากเวลาผ่านไป 23 ชั่วโมง 59 นาที แสดงไว้ดังรูปที่ 4.37

สำหรับภาพถ่ายแสดงการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1 เมื่อตั้งเวลาการทำงานไว้ที่ 1 ชั่วโมง 15 นาที แสดงไว้ดังรูปที่ 4.38

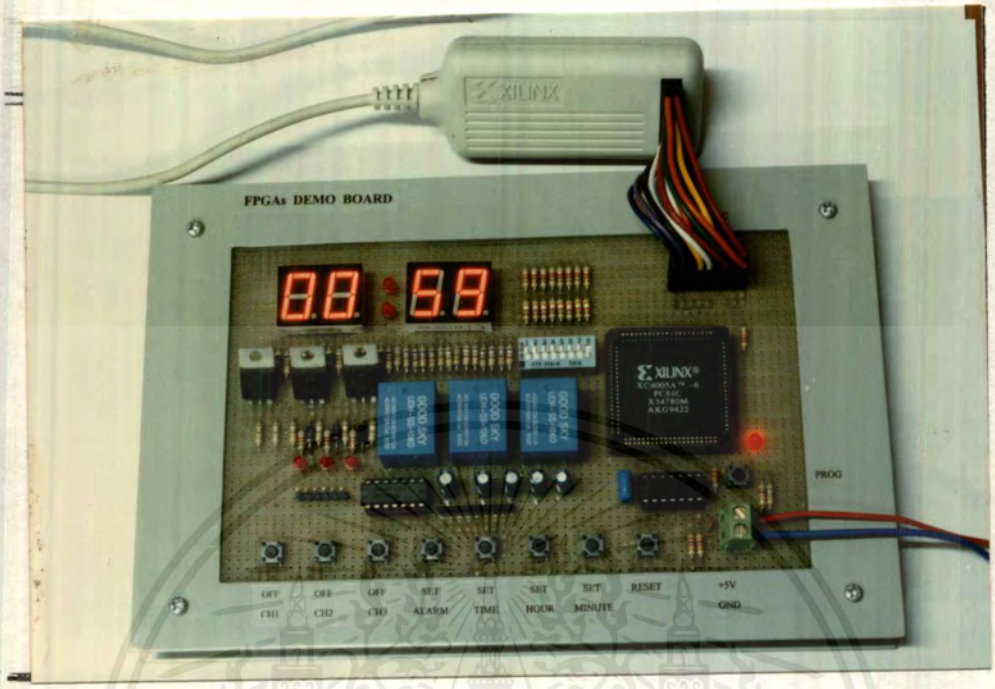
สำหรับภาพถ่ายแสดงการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2 เมื่อตั้งเวลาการทำงานไว้ที่ 2 ชั่วโมง 30 นาที แสดงไว้ดังรูปที่ 4.39



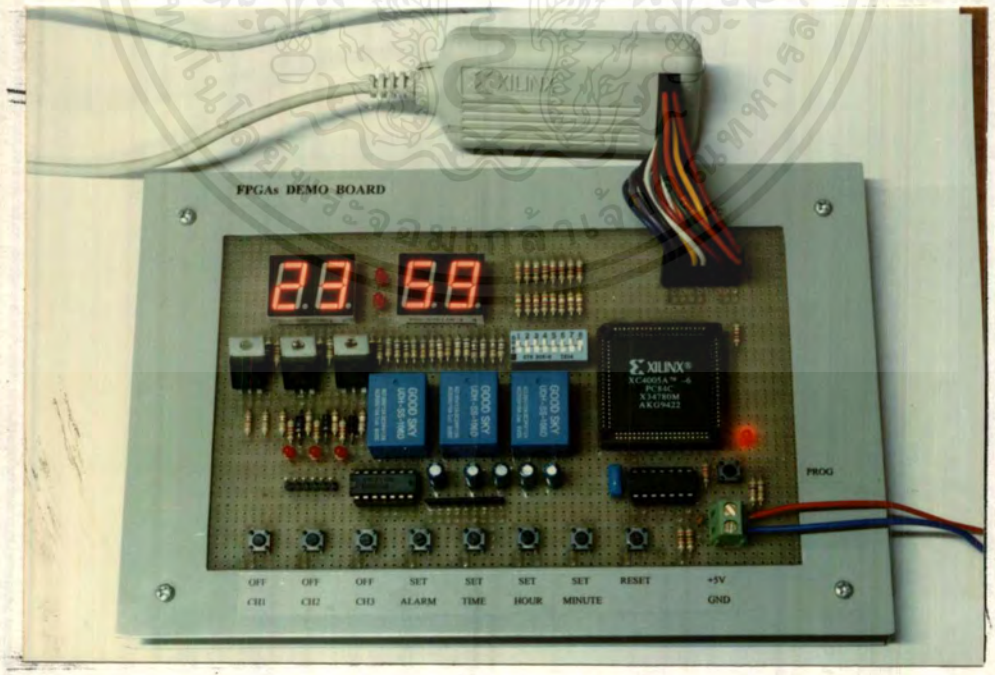
รูปที่ 4.34 ภาพถ่ายการเชื่อมต่อสายดาวน์โหลดเคเบิลกับบอร์ดตัวอย่างของ FPGAs



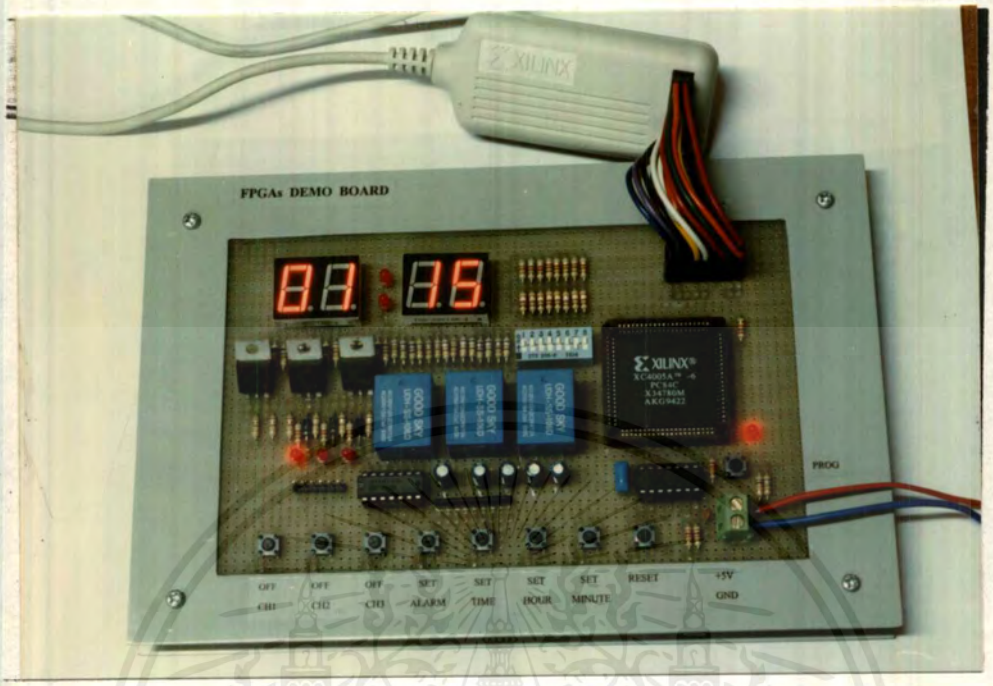
รูปที่ 4.35 ภาพถ่ายแสดงเวลาเริ่มต้นการทำงานของวงจรตั้งเวลาขนาด 2 ช่อง



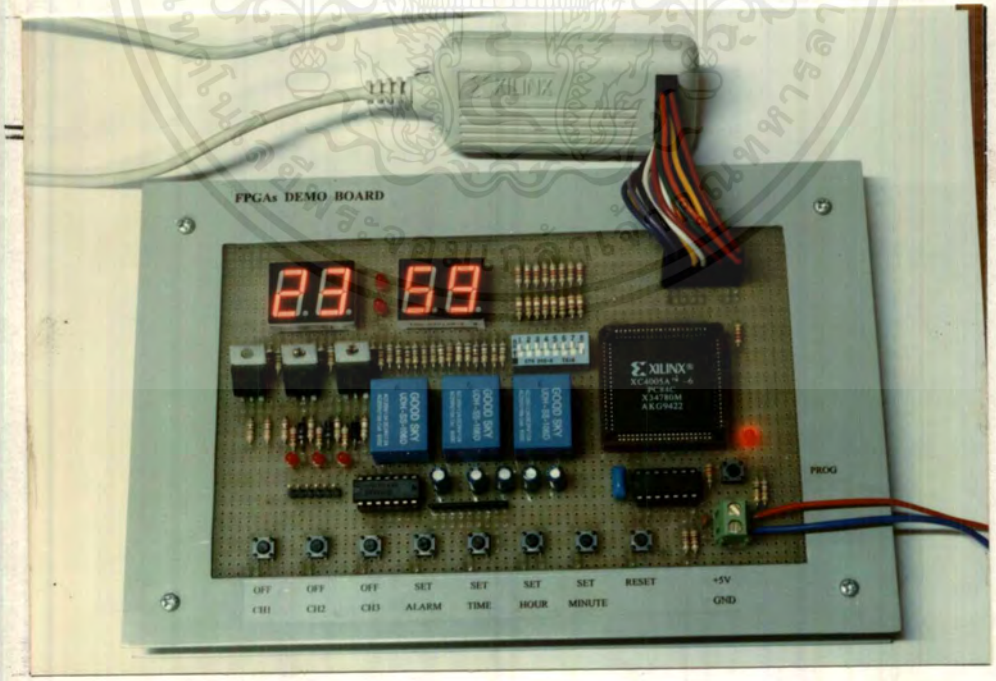
รูปที่ 4.36 ภาพถ่ายแสดงเวลา 59 นาที



รูปที่ 4.37 ภาพถ่ายแสดงเวลา 23 ชั่วโมง 59 นาที



รูปที่ 4.38 ภาพถ่ายแสดงการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 1



รูปที่ 4.39 ภาพถ่ายแสดงการทำงานของวงจรควบคุมอุปกรณ์ไฟฟ้าช่องที่ 2

4.3 ผลการทดลอง

จากการทดลองวงตั้งเวลาขนาด 2 ช่องบนบอร์ดตัวอย่างของ FPGAs สามารถสรุปผลการทดลองได้ดังนี้

1) ต้องจ่ายไฟเลี้ยงให้กับบอร์ดตัวอย่างของ FPGAs และดาวน์โหลดโปรแกรมลงบนไอซี FPGAs ในโหมดสเลฟซีเรียลโดยให้ขา M0, M1, M2 มีสภาวะเป็น '1' ทั้ง 3 ขา

2) เวลาเริ่มแรกภายหลังจากการดาวน์โหลดโปรแกรมเสร็จจะเริ่มต้นที่เวลา 00:00

3) วงจรนาฬิกาจะเริ่มต้นเดินจาก 0 วินาที จนถึงเวลา 23 ชั่วโมง 59 นาที 59 วินาที แล้ววนกลับมาเริ่มต้นที่ 0 วินาทีใหม่

4) การแสดงเวลาในหลักชั่วโมงและหลักนาทีจะแสดงผลบนส่วนแสดงผลแบบ 7 ส่วน หลักวินาทีจะแสดงผลโดยใช้หลอดไฟแสดงผล ซึ่งอยู่ระหว่าง ส่วนแสดงผลแบบ 7 ส่วนของหลักชั่วโมงและหลักนาที

5) วงจรนาฬิกาสามารถตั้งเวลาในหลักชั่วโมงได้โดยการกดสวิตซ์ SET TIME พร้อมกับสวิตซ์ SET HOUR และสามารถตั้งเวลาในหลักนาทีได้โดยการกดสวิตซ์ SET TIME พร้อมกับสวิตซ์ SET MINUTE

6) วงจรตั้งเวลาสามารถตั้งเวลาได้ 2 ช่อง

7) การเลือกช่องของวงจรตั้งเวลาสามารถทำได้โดย วงจรตั้งเวลาช่องที่ 1เลือกให้ดิพสวิตช์ตัวที่ 7 อยู่ที่ตำแหน่ง ON และวงจรตั้งเวลาช่องที่ 2 เลือกให้ดิพสวิตช์ ตัวที่ 8 อยู่ที่ตำแหน่ง ON

8) วงจรตั้งเวลาสามารถตั้งเวลาในหลักชั่วโมงได้ โดยการกดสวิตซ์ SET ALARM พร้อมกับสวิตซ์ SET HOUR และสามารถตั้งเวลาในหลักนาทีได้โดยการกดสวิตซ์ SET ALARM พร้อมกับสวิตซ์ SET MINUTE

9) เมื่อกดสวิตซ์ SET ALARM จะแสดงเวลาของวงจรตั้งเวลาแทนเวลาของวงจรนาฬิกา

10) วงจรตั้งเวลาสามารถตั้งเวลาได้ครั้งละ 1 ช่อง

11) เมื่อเวลาของวงจรนาฬิกาตรงกับเวลาของวงจรตั้งเวลาในช่องใด วงจรควบคุมอุปกรณ์ไฟฟ้าของแต่ละช่องจะทำงาน หลอดไฟแสดงผลก็จะติดสว่างและได้ยินเสียงรีเลย์ทำงาน

12) ถ้าต้องการให้วงจรควบคุมอุปกรณ์ไฟฟ้าหยุดทำงาน ทำได้โดยการกดสวิตซ์ OFF CH ของแต่ละช่อง

13) การรีเซ็ทเวลาของวงจรนาฬิกาหลักนาที ทำได้โดยการกดสวิตช์ RESET, SET TIME และ SET MINUTE พร้อมกัน

14) การรีเซ็ทเวลาของวงจรนาฬิกาหลักชั่วโมง ทำได้โดยการกดสวิตช์ RESET, SET TIME และ SET HOUR พร้อมกัน

16) การรีเซ็ทเวลาของวงจรตั้งเวลาหลักนาที ทำได้โดยการกดสวิตช์ RESET SET ALARM TIME และ SET MINUTE พร้อมกัน

17) การรีเซ็ทเวลาของวงจรตั้งเวลาหลักชั่วโมง ทำได้โดยการกดสวิตช์ RESET SET TIME และ SETHOUR พร้อมกัน

18) เมื่อถอดไฟเลี้ยงออกจากบอร์ดหรือกดสวิตช์ PROG ต้องทำการดาวน์โหลดโปรแกรมลงบนบอร์ดตัวอย่างของ FPGAs ใหม่ทุกครั้ง

19) เวลาของวงจรตั้งเวลาขนาด 2 ช่องเดินช้ากว่าเวลาที่เป็นจริง โดยผลการเทียบเวลาของวงจรตั้งเวลาขนาด 2 ช่องกับนาฬิกาดิจิตอลปรากฏว่า เมื่อเวลาของนาฬิกาดิจิตอลเดินไปได้ 1 ชั่วโมง เวลาของวงจรตั้งเวลาขนาด 2 ช่องจะเดินได้ 50 นาที ช้ากว่าเวลาที่เป็นจริง 10 นาที

บทที่ 5
สรุปอภิปรายข้อเสนอแนะ

5.1 สรุป

วงจรตั้งเวลาขนาด 2 ช่องที่สร้างขึ้นแทนวงจรตั้งเวลาขนาด 3 ช่อง ทำการออกแบบด้วยภาษา VHDL ซึ่งเป็นภาษาที่ใช้ในการบรรยายการทำงานทางฮาร์ดแวร์ของวงจรเชิงเลข โดยตัวภาษา VHDL นั้นมีลักษณะคล้ายกับภาษาคอมพิวเตอร์ทั่วไป ทำให้ง่ายต่อการทำความเข้าใจและการแก้ไข เมื่อทำการเขียนโปรแกรมภาษา VHDL เพื่ออธิบายการทำงานทุกส่วนของวงจรตั้งเวลาขนาด 2 ช่องเสร็จเรียบร้อยแล้ว ขั้นต่อมาคือนำโปรแกรมไปจำลองการทำงาน เพื่อดูการทำงานของวงจรตั้งเวลาขนาด 2 ช่องที่ออกแบบว่ามีการทำงานเป็นไปตามจุดประสงค์ที่ต้องการหรือไม่ ถ้าผลการจำลองการทำงานถูกต้องเป็นไปตามที่ต้องการแล้ว จึงนำโปรแกรมไปสังเคราะห์เป็นวงจรระดับเกทและแปลงไฟล์ที่สังเคราะห์ได้เป็นไฟล์บิตสตรีมโดยใช้ซอฟแวร์ XACT Development Tool เพื่อทำการดาวน์โหลดลงบนบอร์ดตัวอย่างของ FPGAs ผ่านทางดาวน์โหลดเคเบิล

การทดสอบการทำงานของวงจรตั้งเวลา 2 ช่องหลังจากทำการดาวน์โหลดแล้ว ปรากฏว่าวงจรตั้งเวลาขนาด 2 ช่องบนบอร์ดตัวอย่างของ FPGAs สามารถทำการตั้งเวลาได้ถูกต้องตามที่ออกแบบไว้ เพียงแต่เวลาของวงจรนาฬิกาเดินช้ากว่าเวลาที่เป็นจริง โดยวงจรตั้งเวลา 2 ช่องที่สร้างขึ้นมีขอบเขตการทำงานตามที่ออกแบบไว้ดังนี้

1) สามารถใช้เป็นนาฬิกาที่นับได้ตั้งแต่ 0 วินาทีจนถึง 23 ชั่วโมง 59 นาที 59 วินาที

2) สามารถตั้งเวลาได้ 2 ช่อง

3) สามารถตั้งเวลาได้ตั้งแต่ 1 นาทีจนถึง 23 ชั่วโมง 59 นาที

4) แสดงผลโดยใช้ส่วนแสดงผลแบบ 7 ส่วนจำนวน 4 หลัก

5) เมื่อเวลาของนาฬิกาเท่ากับเวลาของวงจรตั้งเวลา จะทำให้วงจรควบคุมอุปกรณ์ไฟฟ้าของวงจรตั้งเวลาช่องนั้นๆ ทำงาน

6) สามารถรีเซ็ทเวลาของวงจรนาฬิกาและวงจรตั้งเวลาให้เริ่มต้นที่ 0 นาฬิกาได้

5.2 ปัญหาและแนวทางการแก้ไข

ในการจัดทำโครงงานนี้ สามารถสรุปปัญหาที่เกิดขึ้นระหว่างทำโครงงานได้ดังนี้

1) โปรแกรมภาษา VHDL ที่เขียนโดยใช้ซอฟแวร์เวอร์ชันใหม่ของบริษัทวิวลอจิก จึงทำให้การเขียนโปรแกรมภาษา VHDL ทำได้ลำบาก เนื่องจากไม่มีคู่มือการใช้โปรแกรมให้ศึกษา

แนวทางการแก้ไข คือ ศึกษาการใช้งานของซอฟแวร์โดยดูจากเอกสารการใช้โปรแกรมที่เมนู Help ของโปรแกรมวิวลอจิก และศึกษาจากคู่มือการใช้โปรแกรมของวิวลอจิกเวอร์ชันเก่า

2) โปรแกรมภาษา VHDL ของวงจรตั้งเวลาขนาด 2 ช่องที่ประกาศใช้ไลบรารี synth; และเรียกใช้ไลบรารี synth.vhdlsynth.all ไม่สามารถนำไปจำลองการทำงานของโปรแกรมได้

แนวทางการแก้ไข คือ เรียกใช้ไลบรารีใหม่จากไลบรารี synth.vhdlsynth.all เปลี่ยนเป็นไลบรารี synth.pack1164.all จึงจะนำไปจำลองการทำงานของวงจรตั้งเวลาขนาด 2 ช่องได้ เมื่อต้องการนำไปสังเคราะห์เป็นวงจรระดับเกทต้องเรียกใช้ไลบรารีsynth.vhdlsynth.all โดยไลบรารีทั้ง 2 ตัวไม่สามารถเรียกใช้พร้อมกันได้

3) ไม่สามารถสังเคราะห์โปรแกรม VHDL ที่ใช้การบรรยายแบบโครงสร้างให้เป็นวงจรในระดับเกทได้

แนวทางการแก้ไข คือ สังเคราะห์โปรแกรมที่เป็นส่วนย่อยเสียก่อน เช่น การสังเคราะห์โปรแกรมAclock.vhd จะต้องทำการสังเคราะห์โปรแกรม Timer.vhd , Alarm.vhd และโปรแกรมอื่นๆ เสียก่อน ในการสังเคราะห์โปรแกรมย่อยนั้นขาสัญญาณต่างๆ ต้องกำหนดให้เป็นขาสัญญาณภายใน ส่วนโปรแกรม Aclock.vhd ซึ่งเป็นโปรแกรมหลักนั้นต้องกำหนดขาสัญญาณให้มีอินพุทและเอาท์พุทเพื่อใช้ต่อกับอุปกรณ์ภายนอก เช่น วงจรควบคุมอุปกรณ์ไฟฟ้าหรือวงจรภาคแสดงผล

4) ไม่สามารถออกแบบสร้างวงจรตั้งเวลาขนาด 3 ช่องได้ เนื่องจากอุปกรณ์ FPGAs เบอร์ XC4005APC84C-6 ที่ใช้ในโครงงานมีจำนวนเกทภายในไม่เพียงพอที่จะดาวน์โหลดวงจรตั้งเวลาขนาด 3 ช่องลงไปได้

แนวทางการแก้ไข คือ เปลี่ยนไปใช้อุปกรณ์ FPGAs เบอร์4006 หรือเบอร์ XC 4010 ซึ่งมีจำนวนเกทภายในสูงกว่าอุปกรณ์เบอร์ที่ใช้ในโครงงาน ผู้จัดทำไม่สามารถหาได้จึงลดจำนวนช่องของวงจรตั้งเวลาให้เหลือ 2 ช่อง เพื่อให้สามารถดาวน์โหลดโปรแกรมลงไปได้

5) ไม่สามารถดาวน์โหลดไฟล์ที่เป็นบิตสตรีมลงบนบอร์ดตัวอย่างของ FPGAs โดยผ่านทาง สายดาวน์โหลดเคเบิลได้

แนวทางการแก้ไข คือ ตรวจเช็คที่บอร์ดตัวอย่างของ FPGAs โดยดูว่าต่อวงจรถูกต้องหรือไม่และตั้งดิพสวิตช์ให้ทำการรับข้อมูลในลักษณะสเลฟซีเรียลโดยให้ดิพสวิตช์ตำแหน่ง M0,M1,M2 และ INIT อยู่ในตำแหน่ง ON

6) การใช้งานวงจรตั้งเวลาขนาด 2 ช่องต้องทำการดาวน์โหลดก่อนทุกครั้งที่ถอดไฟเลี้ยงออกจากวงจร และในการใช้งานต้องเสียบสายดาวน์โหลดเคเบิลติดอยู่กับบอร์ดตัวอย่างของ FPGAs ตลอดเวลา

แนวทางการแก้ไข คือ ดาวน์โหลดโปรแกรมลง EPROM และตั้งดิพสวิตช์ให้ใช้งานในลักษณะมาสเตอร์ซีเรียล โดยอุปกรณ์ FPGAs จะรับข้อมูลจาก EPROM แทนการดาวน์โหลดโปรแกรมจากคอมพิวเตอร์

7) เวลาของวงจรตั้งเวลาขนาด 2 ช่องเดินช้ากว่าเวลาที่เป็นจริง

แนวทางการแก้ไข คือ ใช้วงจรสร้างสัญญาณนาฬิกาจากภายนอกเป็นตัวสร้างสัญญาณฐานเวลาให้กับวงจรตั้งเวลาขนาด 2 ช่องแทนการใช้วงจรสร้างสัญญาณนาฬิกาภายในตัวอุปกรณ์ FPGA หรือป้อนสัญญาณนาฬิกาขนาด 1 Hz ให้กับวงจรตั้งเวลาขนาด 2 ช่องโดยตรง

5.3 แนวทางในการพัฒนา

1) วงจรตั้งเวลาขนาด 2 ช่องที่สร้างขึ้นเป็นบอร์ดตัวอย่างของ FPGAs สามารถเปลี่ยนลักษณะการแสดงผลเป็นแบบการสแกน ซึ่งจะช่วยลดจำนวนขาเอาท์พุทของอุปกรณ์ FPGAs ลงได้

2) ภาษา VHDL เป็นภาษาที่มีลักษณะคล้ายกับภาษาคอมพิวเตอร์ทั่วไป เช่น ภาษาซีหรือภาษาปาสคาล ตัวภาษา VHDLทำความเข้าใจและแก้ไขได้ง่าย สามารถเขียนอธิบายการทำงานของวงจรเชิงเลขได้อย่างกว้างขวาง และมีฟังก์ชันการคำนวณทางตรรก จึงสามารถใช้ภาษา VHDL บรรยายการทำงานทางฮาร์ดแวร์ของวงจรเชิงเลขที่มีขนาดใหญ่หรือหน่วยประมวลผลได้

3) สามารถใช้ภาษา VHDL ออกแบบวงจรที่ใช้เฉพาะงานลงบนอุปกรณ์ FPGAs โดยเฉพาะวงจรที่ไม่ต้องการให้เกิดการลอกเลียนแบบ

4) การออกแบบวงจรโดยใช้ภาษา VHDL โดยการเรียกใช้ฟังก์ชันของภาษา VHDL จะช่วยลดจำนวนเกทในการสังเคราะห์วงจรได้

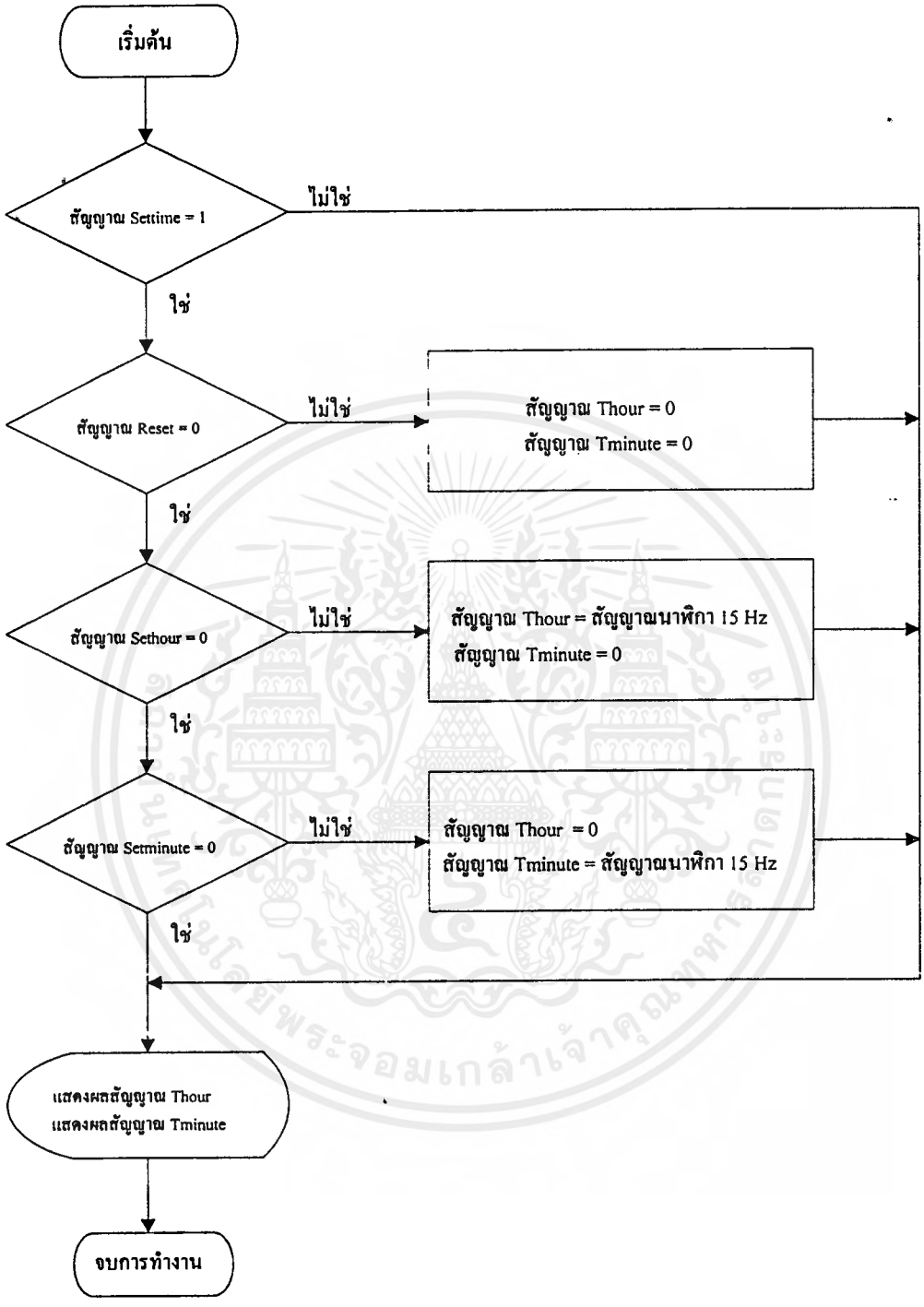
5) ภาษา VHDL สามารถเขียนบรรยายการทำงานของวงจรเชิงเลขได้ทุกวงจร เช่น วงจรนับ วงจรเข้ารหัส, หน่วยความจำ เป็นต้น

6) อุปกรณ์ FPGAs ตระกูล XC 4000 ในไลบรารีของไซลิงค์มีฟังก์ชันการบวก การลบ การเปรียบเทียบ จึงสามารถใช้งานเป็นไมโครโปรเซสเซอร์ได้ โดยขนาดความซับซ้อนของวงจรจะขึ้นอยู่กับจำนวนเกตภายในของอุปกรณ์ FPGAs แต่ละเบอร์

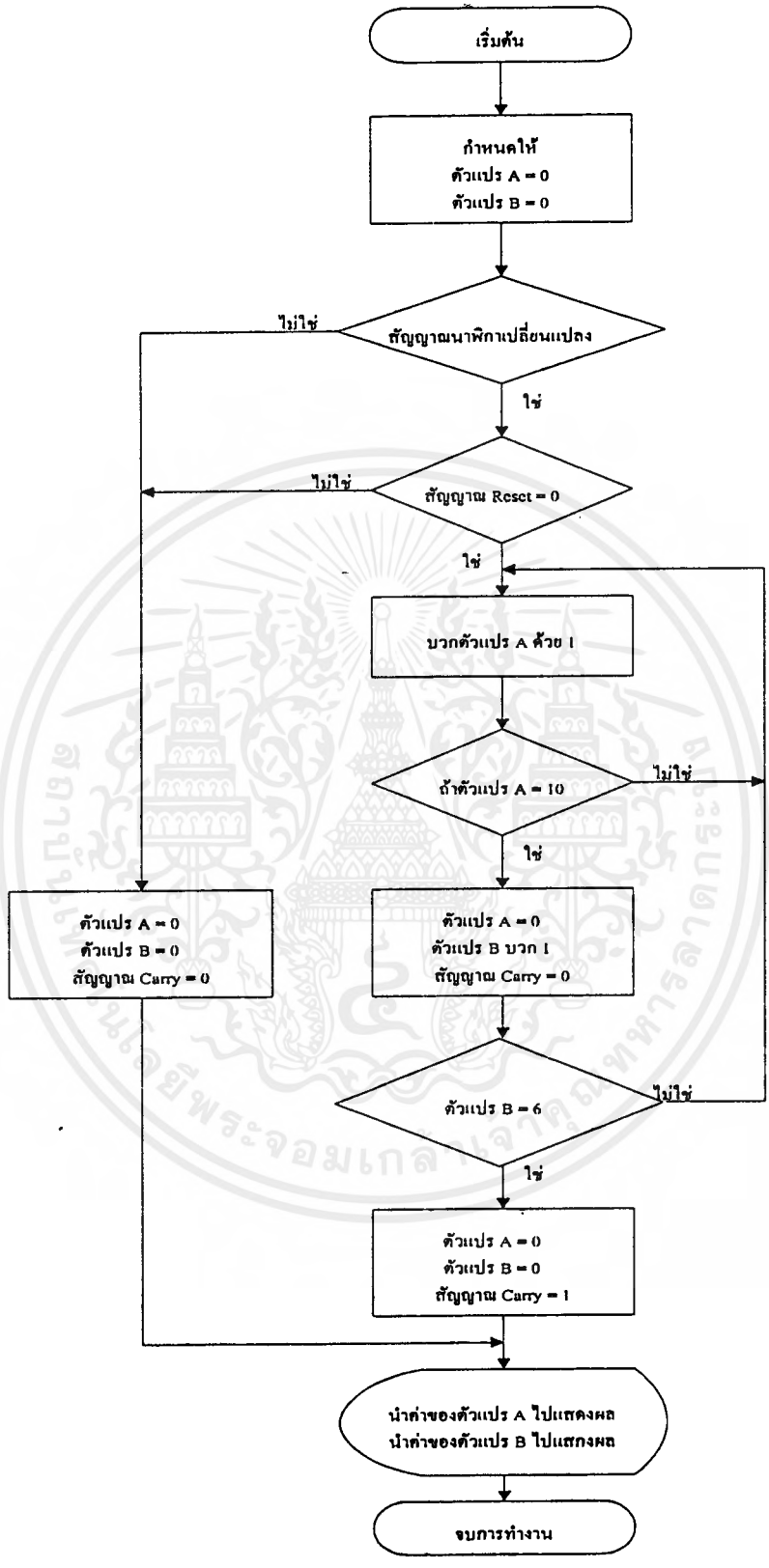
7) วงจรตั้งเวลาขนาด 2 ช่องที่ทำการออกแบบสามารถเพิ่มจำนวนช่องให้มากขึ้นได้ โดยใช้อุปกรณ์ FPGAs ที่มีจำนวนเกทภายในเพิ่มมากขึ้น และเพิ่มเติมส่วนของซอฟแวร์และฮาร์ดแวร์ตามจำนวนช่องที่ต้องการใช้งาน

8) สามารถใช้ภาษา VHDL และอุปกรณ์ FPGAs ออกแบบวงจรไปประยุกต์ใช้งานตามความสามารถและตามความต้องการของผู้ออกแบบแต่ละคนได้อย่างกว้างขวาง

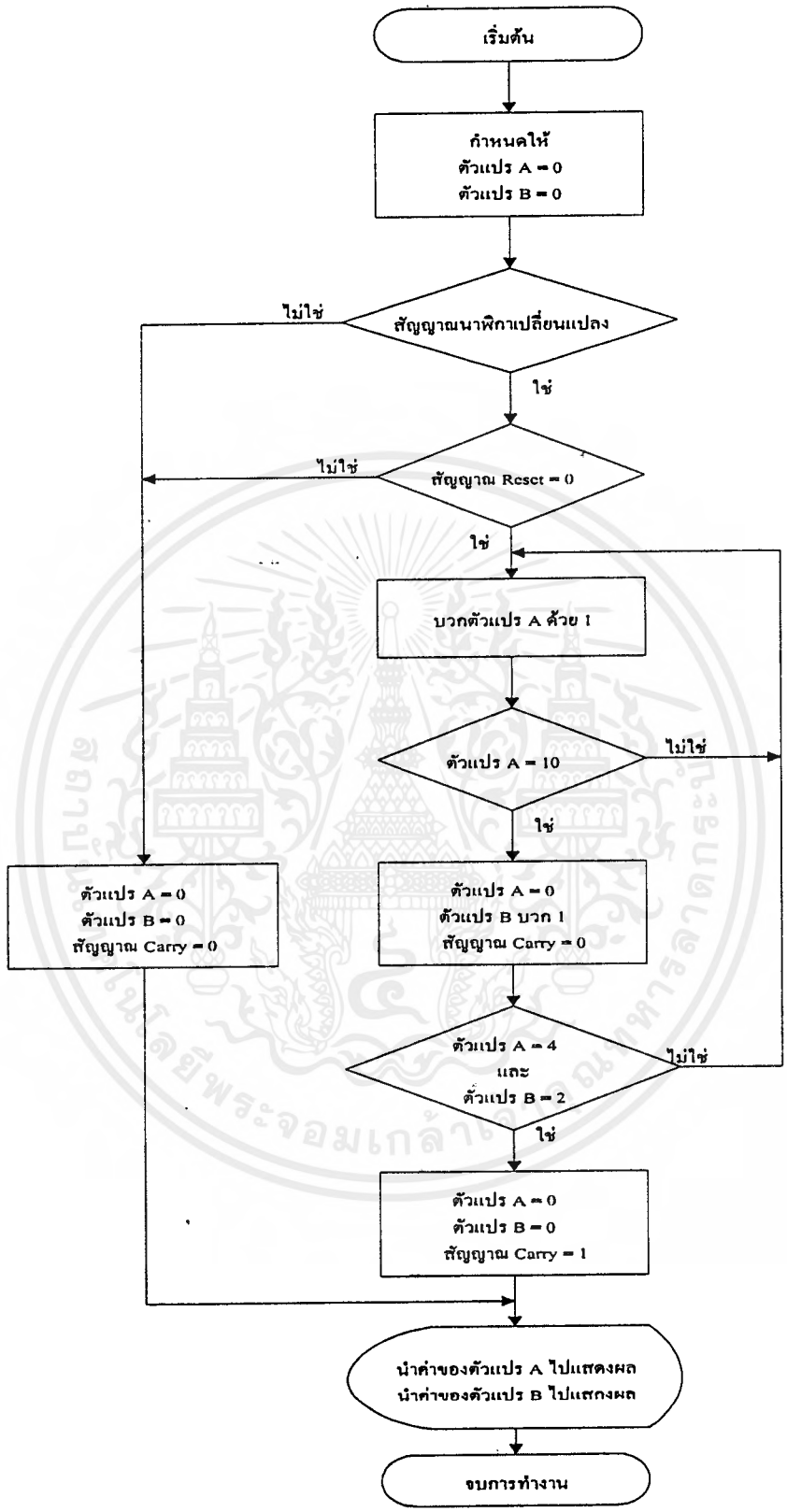
ภาคผนวก ก
ผังงานและโปรแกรมภาษา VHDL ของวงจรตั้งเวลาขนาด 2 ช่อง



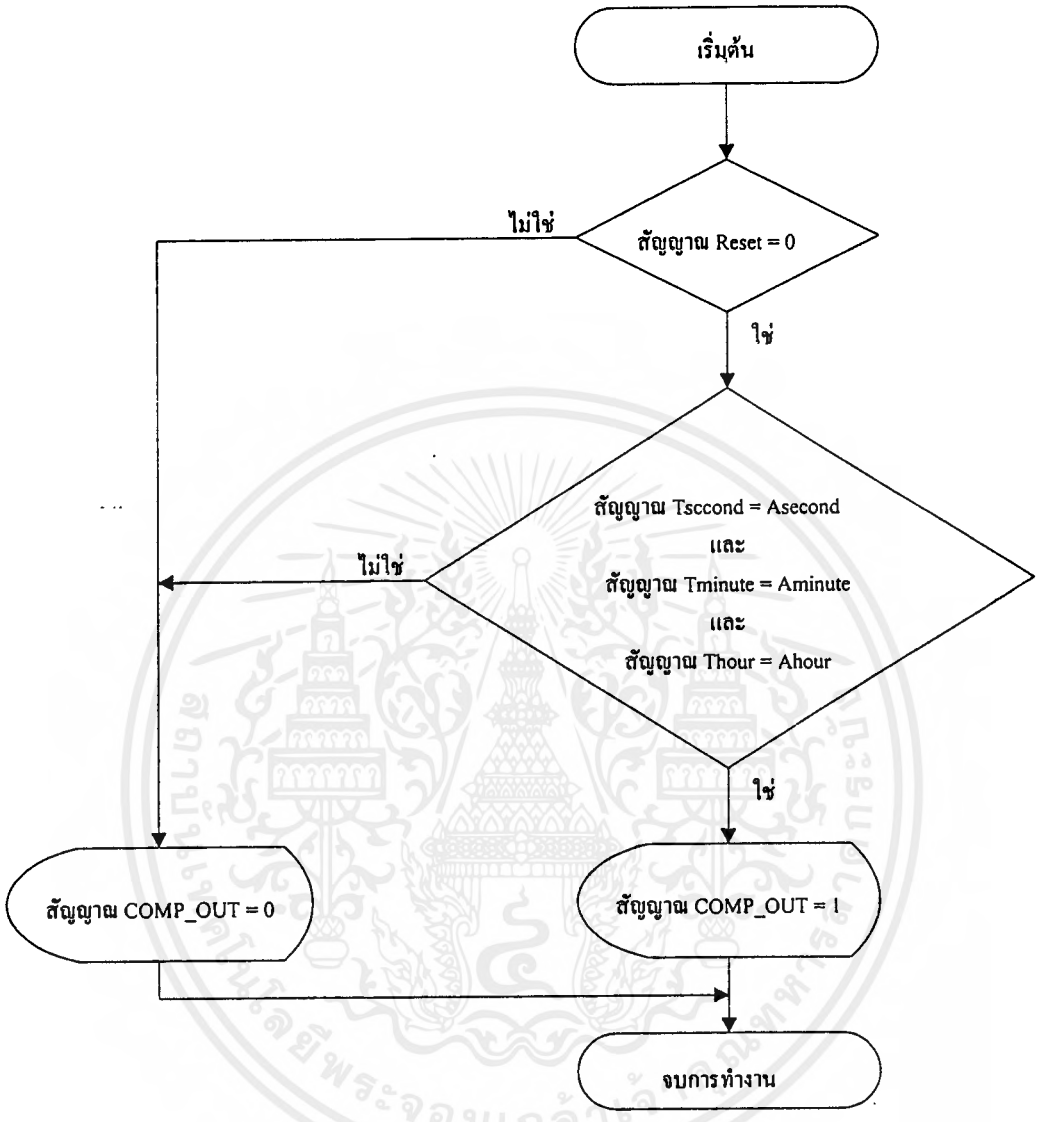
รูปที่ ก.1 ผังงานของโปรแกรมควบคุมการตั้งเวลาของนาฬิกา



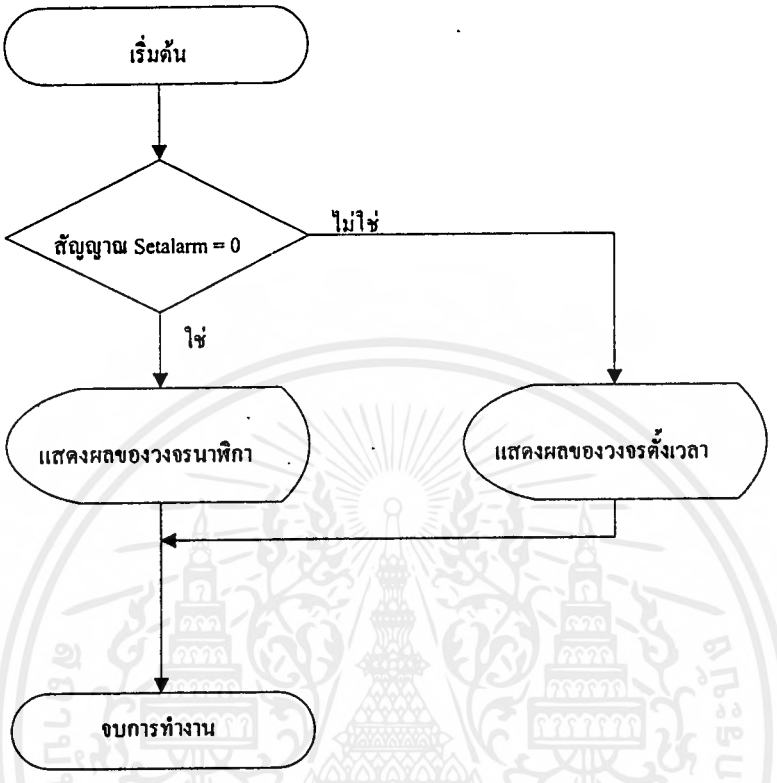
รูปที่ ก.2 ผังงานของโปรแกรมนับหลักนาทีและหลักวินาที



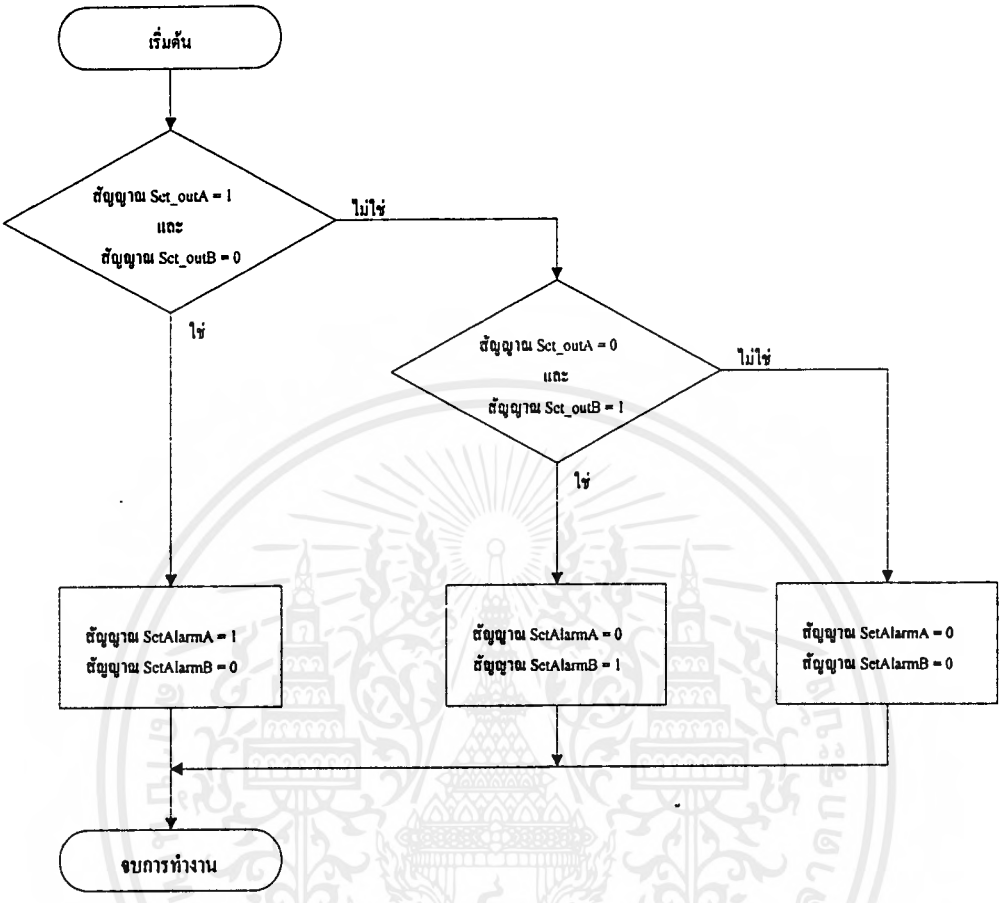
รูปที่ ก.3 ผังงานของโปรแกรมนับหลักชั่งโมง



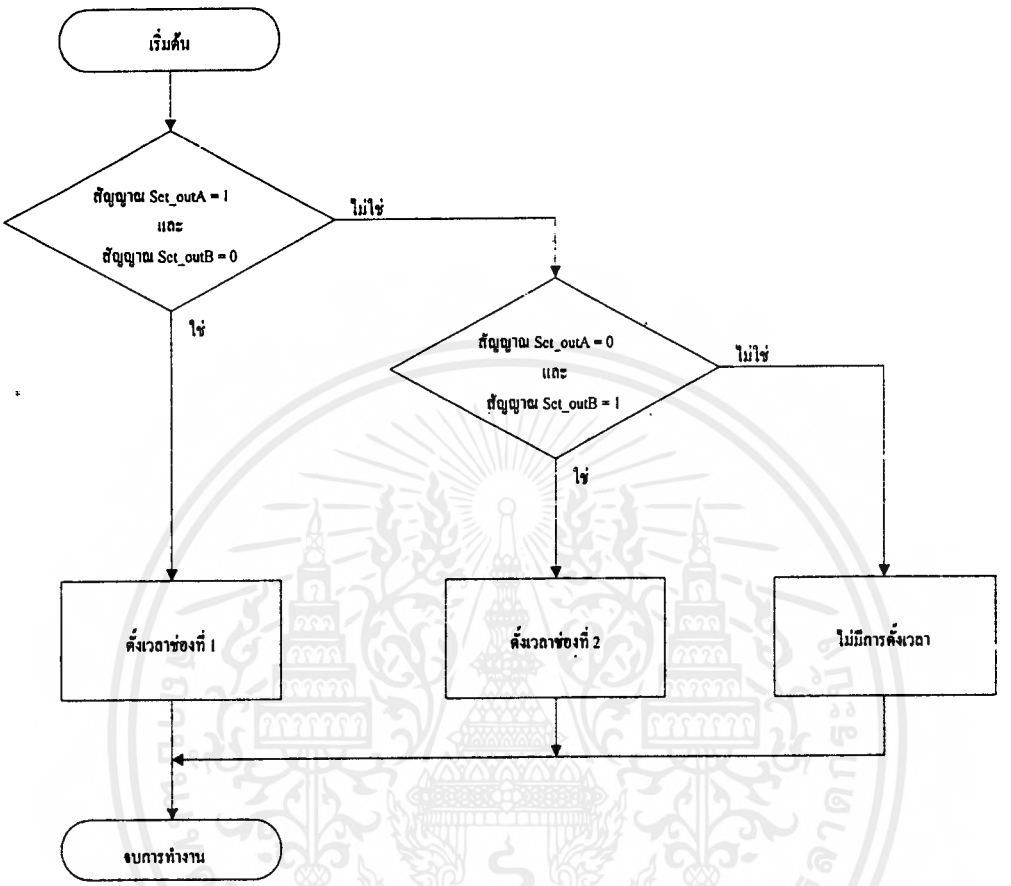
รูปที่ ก.4 ผังงานของโปรแกรมเปรียบเทียบ



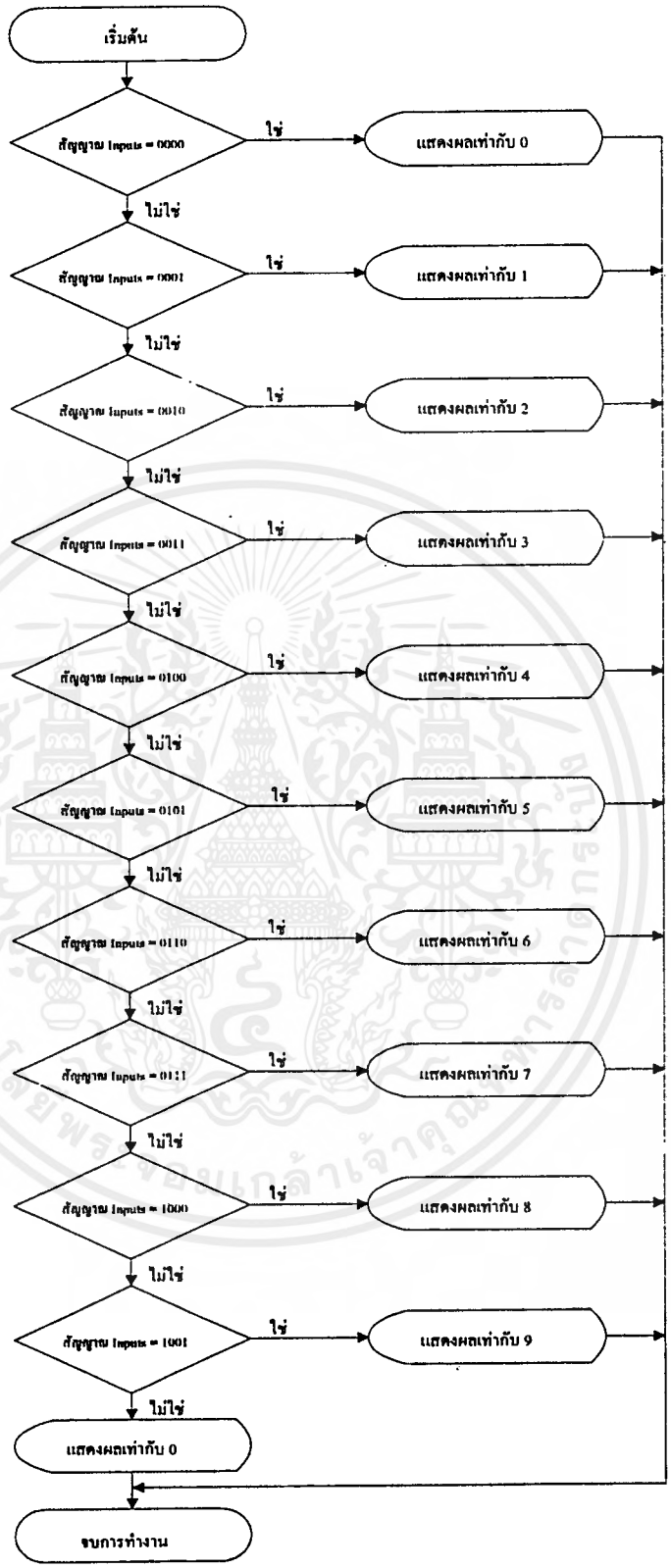
รูปที่ ก.5 ผังงานของโปรแกรมเลือกการแสดงผล



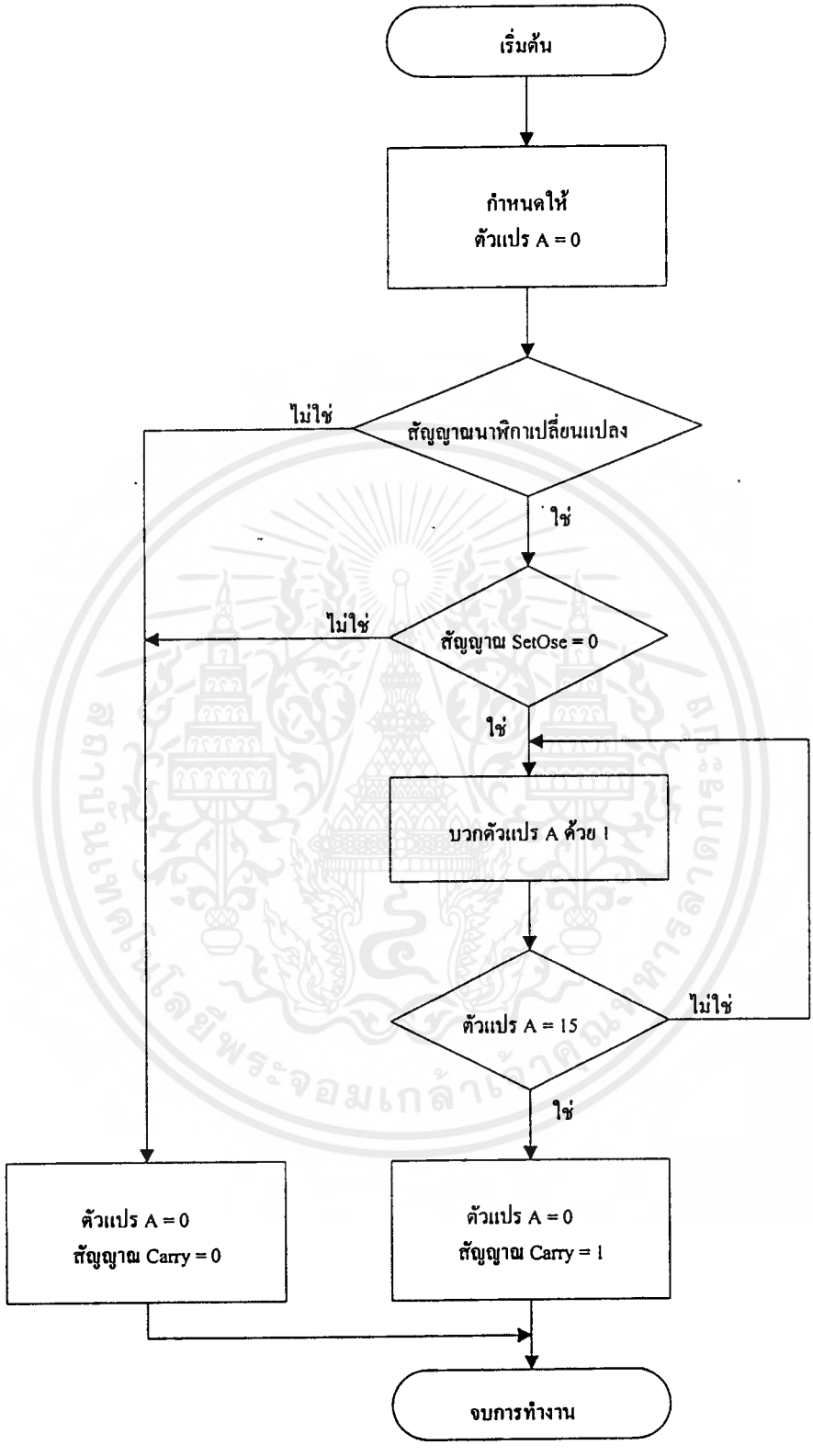
รูปที่ ก.6 ผังงานของโปรแกรมเลือกช่องตั้งเวลา



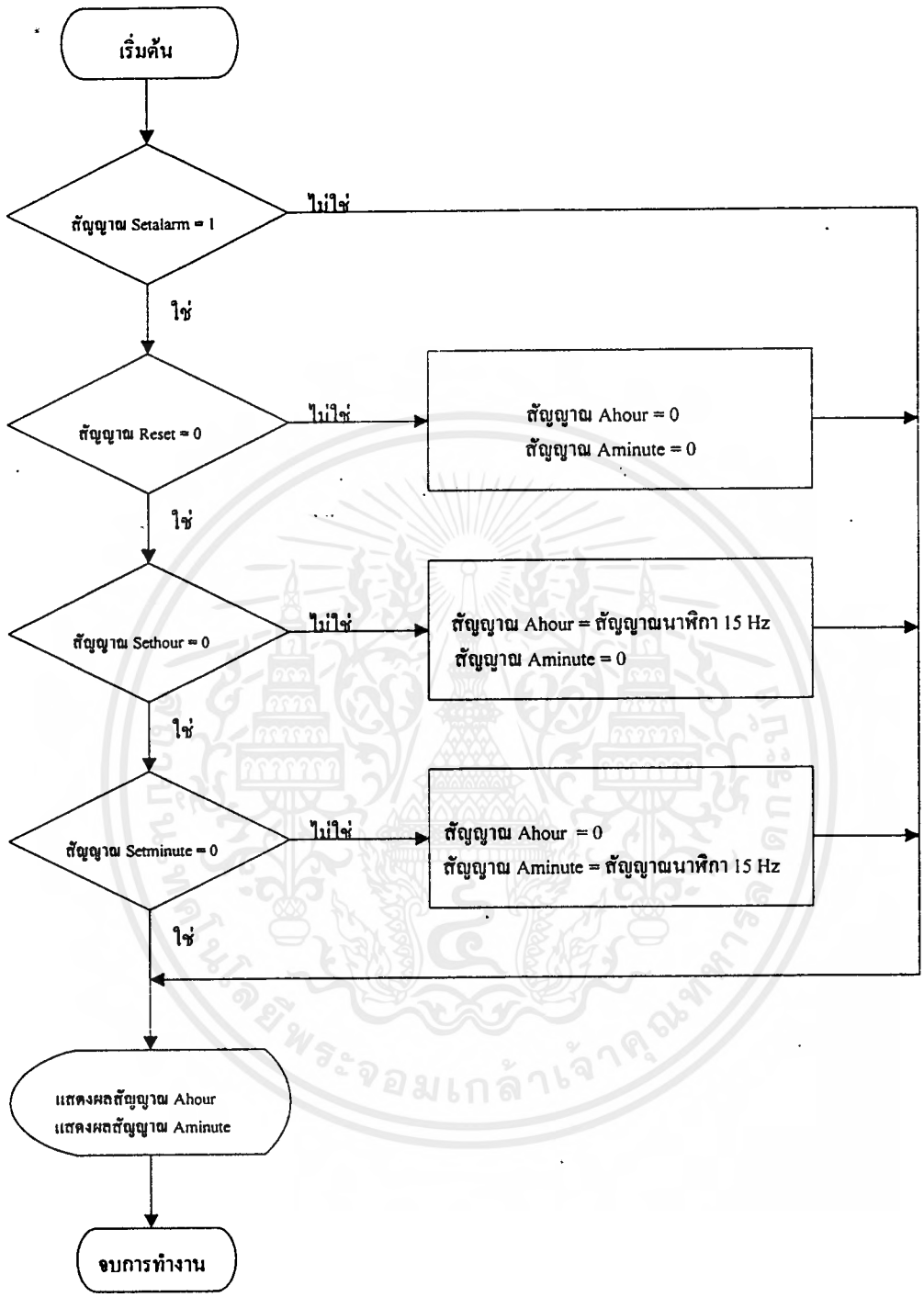
รูปที่ ก.7 ผังงานของโปรแกรมเลือกช่องตั้งเวลาในการแสดงผล



รูปที่ ก.8 ผังงานของโปรแกรมแปลงเลขBCDเป็นเลข 7 ส่วน



รูปที่ ก.9 ผังงานของโปรแกรมหารความถี่



รูปที่ ก.10 ผังงานของโปรแกรมควบคุมการตั้งเวลาของวงจรตั้งเวลา

```
--------------------------------------------------------------------------------
-------  Project  Digital Alarm Clock 2 Channel                          -----
------   By       Mr. Satith      Waiyaphan        38031328            ------
------            Mr.Donson       Pongphab         38031304            ------
------            Mr. Prinya      Thongkongthun    38031315            ------
------            Mr.Sophat       Chatsawangwong   38031332            ------
--------------------------------------------------------------------------------
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY synth;
USE synth.vhdlsynth.all;

ENTITY Aclock IS
        PORT ( reset        :IN  std_logic;
               clk_15hz     :IN  std_logic;
               setosc       :IN std_logic;
               set_outa     :IN std_logic;
               set_outb     :IN std_logic;
               settime      :IN std_logic;
               setalarm     :IN std_logic;
               sethour      :IN std_logic;
               setminute    :IN std_logic;
               led_1hz      :OUT std_logic;
               c_out        :OUT std_logic:
               c_out1       :OUT std_logic;
               seg_min0     :OUT std_logic_vector(6 downto 0);
               seg_min1     :OUT std_logic_vector(6 downto 0);
               seg_hr0      :OUT std_logic_vector(6 downto 0);
               seg_hr1      :OUT std_logic_vector(6 downto 0));
        END Aclock;
```

```
ARCHITECTURE behavioral OF Aclock IS

COMPONENT osc
 PORT ( clk            :IN std_logic;
        setin          :IN std_logic;
        clkout         :OUT std_logic);
END COMPONENT;

COMPONENT Timer
 PORT ( clk_1hz        :IN std_logic;
        clk_15hz       :IN std_logic;
        reset          :IN std_logic;
        settime        :IN std_logic;
        sethour        :IN std_logic;
        setminute      :IN std_logic;
        tsec_out0      :OUT std_logic_vector (3 downto 0);
        tsec_out1      :OUT std_logic_vector (3 downto 0);
        tmin_out0      :OUT std_logic_vector (3 downto 0);
        tmin_out1      :OUT std_logic_vector (3 downto 0);
        thr_out0       :OUT std_logic_vector (3 downto 0);
        thr_out1       :OUT std_logic_vector (3 downto 0));
END COMPONENT;

COMPONENT Alarm
 PORT ( clk_15hz       :IN std_logic;
        reset          :IN std_logic;
        setalarm       :IN std_logic;
        sethour        :IN std_logic;
        setminute      :IN std_logic;
        asec_out0      :OUT std_logic_vector(3 downto 0);
```

```
        asec_out1       :OUT std_logic_vector(3 downto 0);
        amin_out0       :OUT std_logic_vector(3 downto 0);
        amin_out1       :OUT std_logic_vector(3 downto 0);
        ahr_out0        :OUT std_logic_vector(3 downto 0);
        ahr_out1        :OUT std_logic_vector(3 downto 0));
END COMPONENT;

COMPONENT Alarm1
 PORT ( clk_15hz        :IN std_logic;
        reset           :IN std_logic;
        setalarm        :IN std_logic;
        sethour         :IN std_logic;
        setminute       :IN std_logic;
        asec1_out0      :OUT std_logic_vector(3 downto 0);
        asec1_out1      :OUT std_logic_vector(3 downto 0);
        amin1_out0      :OUT std_logic_vector(3 downto 0);
        amin1_out1      :OUT std_logic_vector(3 downto 0);
        ahr1_out0       :OUT std_logic_vector(3 downto 0);
        ahr1_out1       :OUT std_logic_vector(3 downto 0));
END COMPONENT;

COMPONENT control
 PORT ( in_puta         :IN std_logic;
        in_putb         :IN std_logic;
        out_0           :OUT std_logic;
        out_1           :OUT std_logic);
END COMPONENT;

COMPONENT Multi
 PORT ( set_a           :IN std_logic;
```

```
        set_b           :IN std_logic;
        a0_min0         :IN std_logic_vector(3 downto 0);
        a0_min1         :IN std_logic_vector(3 downto 0);
        a0_hour         :IN std_logic_vector(3 downto 0);
        a0_hour         :IN std_logic_vector(3 downto 0);
        a1_min0         :IN std_logic_vector(3 downto 0);
        a1_min1         :IN std_logic_vector(3 downto 0);
        a1_hour         :IN std_logic_vector(3 downto 0);
        a1_hour         :IN std_logic_vector(3 downto 0);
        bcd_minte0      :OUT std_logic_vector(3 downto 0);
        bcd_minute1     :OUT std_logic_vector(3 downto 0);
        bcd_hour0       :OUT std_logic_vector(3 downto 0);
        bcd_hour1       :OUT std_logic_vector(3 downto 0));
END COMPONENT;
COMPONENT Multiplx
 PORT ( setalarm        :IN std_logic;
        tbcdmin0        :IN std_logic_vector(3 downto 0);
        tbcdmin1        :IN std_logic_vector(3 downto 0);
        tbcdhour0       :IN std_logic_vector(3 downto 0);
        tbcdhour1       :IN std_logic_vector(3 downto 0);
        abcdmin0        :IN std_logic_vector(3 downto 0);
        abcdmin1        :IN std_logic_vector(3 downto 0);
        abcdhour0       :IN std_logic_vector(3 downto 0);
        abcdhour1       :IN std_logic_vector(3 downto 0);
        bcd_minute0     :OUT std_logic_vector(3 downto 0);
        bcd_minute1     :OUT std_logic_vector(3 downto 0);
        bcd_hour0       :OUT std_logic_vector(3 downto 0);
        bcd_hour1       :OUT std_logic_vector(3 downto 0));
END COMPONENT;
```

```
COMPONENT compara
 PORT ( reset          :IN std_logic;
        tbcdsec0       :IN std_logic_vector(3 downto 0);
        tbcdsec1       :IN std_logic_vector(3 downto 0);
        tbcdmin0       :IN std_logic_vector(3 downto 0);
        tbcdhour0      :IN std_logic_vector(3 downto 0);
        tbcdhour1      :IN std_logic_vector(3 downto 0);
        abcdsec0       :IN std_logic_vector(3 downto 0);
        abcdsec1       :IN std_logic_vector(3 downto 0);
        abcdmin0       :IN std_logic_vector(3 downto 0);
        abcdmin1       :IN std_logic_vector(3 downto 0);
        abcdhour0      :IN std_logic_vector(3 downto 0);
        abcdhour1      :IN std_logic_vector(3 downto 0);
       comp_out        :OUT std_logic);
END COMPONENT;

COMPONENT compra1
 PORT ( reset          :IN std_logic;
        tbcdsec0       :IN std_logic_vector(3 downto 0);
        tbcdsec1       :IN std_logic_vector(3 downto 0);
        tbcdmin0       :IN std_logic_vector(3 downto 0);
        tbcdmin1       :IN std_logic_vector(3 downto 0);
        tbcdhour0      :IN std_logic_vector(3 downto 0);
        tbcdhour1      :IN std_logic_vector(3 downto 0);
        abcdsec0       :IN std_logic_vector(3 downto 0);
        abcdsec1       :IN std_logic_vector(3 downto 0);
        abcdmin0       :IN std_logic_vector(3 downto 0);
        abcdmin1       :IN std_logic_vector(3 downto 0);
        abcdhour0      :IN std_logic_vector(3 downto 0);
        abcdhour1      :IN std_logic_vector(3 downto 0);
```

```
        comp_out1       :OUT std_logic);
END COMPONENT;

COMPONENT seg70
 PORT ( inputs          :IN std_logic_vector (3 downto 0);
        segout          :OUT std_logic_vector (6 downto 0));
END COMPONENT;

COMPONENT seg71
 PORT ( inputs          :IN std_logic_vector (3 downto 0);
        segout          :OUT std_logic_vector (6 downto 0));
END COMPONENT;

COMPONENT seg72
 PORT ( inputs          :IN std_logic_vector (3 downto 0);
        segout          :OUT std_logic_vector (6 downto 0));
END COMPONENT;

COMPONENT seg73
 PORT ( inputs          :IN std_logic_vector (3 downto 0);
        segout          :OUT std_logic_vector (6 downto 0));
END COMPONENT;

SIGNAL bus_clk_1hz          :std_logic;
SIGNAL bus_clk_15hz         :std_logic;
SIGNAL bus_reset            :std_logic;
SIGNAL bus_set_outa         :std_logic;
SIGNAL bus_set_outb         :std_logic;
SIGNAL bus_settime          :std_logic;
SIGNAL bus_setalarm         :std_logic;
```

```
SIGNAL bus_sethour          :std_logic;
SIGNAL bus_setminute        :std_logic;
SIGNAL bus_out_0            :std_logic;
SIGNAL bus_out_1            :std_logic;
SIGNAL t_bcd_sec0           :std_logic_vector (3 downto 0);
SIGNAL t_bcd_sec1           :std_logic_vector (3 downto 0);
SIGNAL t_bcd_minute0        :std_logic_vector (3 downto 0);
SIGNAL t_bcd_minute1        :std_logic_vector (3 downto 0);
SIGNAL t_bcd_hour0          :std_logic_vector (3 downto 0);
SIGNAL t_bcd_hour1          :std_logic_vector (3 downto 0);
SIGNAL a_bcd_minute0        :std_logic_vector (3 downto 0);
SIGNAL a_bcd_minute1        :std_logic_vector (3 downto 0);
SIGNAL a_bcd_hour0          :std_logic_vector (3 downto 0);
SIGNAL a_bcd_hour1          :std_logic_vector (3 downto 0);
SIGNAL bcd_out_min0         :std_logic_vector (3 downto 0);
SIGNAL bcd_out_min1         :std_logic_vector (3 downto 0);
SIGNAL bcd_out_hr0          :std_logic_vector (3 downto 0);
SIGNAL bcd_out_hr1          :std_logic_vector (3 downto 0);
SIGNAL b_asec_out0          :std_logic_vector (3 downto 0);
SIGNAL b_asec_out1          :std_logic_vector (3 downto 0);
SIGNAl b_amin_out0          :std_logic_vector (3 downto 0);
SIGNAL b_amin_out1          :std_logic_vector (3 downto 0);
SIGNAL b_ahr_out0           :std_logic_vector (3 downto 0);
SIGNAl b_ahr_out1           :std_logic_vector (3 downto 0);
SIGNAL b_asec1_out0         :std_logic_vector (3 downto 0);
SIGNAL b_asec1_out1         :std_logic_vector (3 downto 0);
SIGNAL b_amin1_out0         :std_logic_vector (3 downto 0);
SIGNAL b_amin1_out1         :std_logic_vector (3 downto 0);
SIGNAL b_ahr1_out0          :std_logic_vector (3 downto 0);
SIGNAL b_ahr1_out1          :std_logic_vector (3 downto 0);
```

```
BEGIN
 bus_reset      <=     reset;
 bus_clk_15hz   <=     clk_15hz;
 led_1hz        <=     bus_clk_1hz;
 bus_settime    <=     settime;
 bus_setalarm   <=     setalarm;
 bus_sethour    <=     sethour;
 bus_setminute  <=     setminute;
 bus_set_outa   <=     set_outa;
 bus_set_outb   <=     set_outb;

block0: osc
 PORT map ( clk          =>     bus_clk_15hz,
            setin        =>     setosc,
            clkout       =>     bus_clk_1hz);

block1: Timer
 PORT map ( clk_1hz      =>     bus_clk_1hz,
            clk_15hz     =>     bus_clk_15hz,
            reset        =>     bus_reset,
            settime      =>     bus_settime,
            sethour      =>     bus_sethour,
            setminute    =>     bus_setminute,
            tsec_out0    =>     t_bcd_sec0,
            tsec_out1    =>     t_bcd_sec1,
            tmin_out0    =>     t_bcd_minute0,
            tmin_out1    =>     t_bcd_minute1,
            thr_out0     =>     t_bcd_hour0,
            thr_out1     =>     t_bcd_hour1);
```

```
block2: Alarm
 PORT map ( clk_15hz          =>     bus_clk_15hz,
            reset             =>     bus_reset,
            setalarm          =>     bus_out_0,
            sethour           =>     bus_sethour,
            setminute         =>     bus_setminute,
            asec_out0         =>     b_asec_out0,
            asec_out1         =>     b_asec_out1,
            amin_out0         =>     b_amin_out0,
            amin_out1         =>     b_amin_out1,
            ahr_out0          =>     b_ahr_out0,
            ahr_out1          =>     b_ahr_out1);

block3: Alarm1
 PORT map ( clk_15hz          =>     bus_clk_15hz,
            reset             =>     bus_reset,
            setalarm          =>     bus_out_1,
            sethour           =>     bus_sethour,
            setminute         =>     bus_setminute,
            asec1_out0        =>     b_asec1_out0,
            asec1_out1        =>     b_asec1_out1,
            amin1_out0        =>     b_amin1_out0,
            amin1_out1        =>     b_amin1_out1,
            ahr1_out0         =>     b_ahr1_out0,
            ahr1_out1         =>     b_ahr1_out1);

block4: control
 PORT map ( in_puta           =>     bus_set_outa,
            in_putb           =>     bus_set_outb,
            out_0             =>     bus_out_0,
```

```
            out_1          =>    bus_out_1);

block5: multi
 PORT map ( set_a          =>    bus_set_outa,
            set_b          =>    bus_set_outb,
            a0_min0        =>    b_amin_out0,
            a0_min1        =>    b_amin_out1,
            a0_hour0       =>    b_ahr_out0,
            a0_hour1       =>    b_ahr_out1,
            a1_min0        =>    b_amin1_out0,
            a1_min1        =>    b_amin1_out1,
            a1_hour0       =>    b_ahr1_out0,
            a1_hour1       =>    b_ahr1_out1,
            bcd_minute0    =>    a_bcd_minute0,
            bcd_minute1    =>    a_bcd_minute1,
            bcd_hour0      =>    a_bcd_hour0,
            bcd_hour1      =>    a_bcd_hour1);

block6: Multiplx
 PORT map ( setalarm       =>    bus_setalarm,
            tbcdmin0       =>    t_bcd_minute0,
            tbcdmin1       =>    t_bcd_minute1,
            tbcdhour0      =>    t_bcd_hour0,
            tbcdhour1      =>    t_bcd_hour1,
            abcdmin0       =>    a_bcd_minute0,
            abcdmin1       =>    a_bcd_minute1,
            abcdhour0      =>    a_bcd_hour0,
            abcdhour1      =>    a_bcd_hour1,
            bcd_minute0    =>    bcd_out_min0,
            bcd_minute1    =>    bcd_out_min1,
```

```
        bcd_hour0       =>      bcd_out_hr0,
        bcd_hour1       =>      bcd_out_hr1);

block7: compara
 PORT map ( reset       =>      bus_reset,
        tbcdsec0        =>      t_bcd_sec0,
        tbcdsec1        =>      t_bcd_sec1,
        tbcdmin0        =>      t_bcd_minute0,
        tbcdmin1        =>      t_bcd_minute1,
        tbcdhour0       =>      t_bcd_hour0,
        tbcdhour1       =>      t_bcd_hour1,
        abcdsec0        =>      b_asec_out0,
        abcdsec1        =>      b_asec_out1,
        abcdmin0        =>      b_amin_out0,
        abcdmin1        =>      b_amin_out1,
        abcdhour0       =>      b_ahr_out0,
        abcdhour1       =>      b_ahr_out1,
        comp_out        =>      c_out);

block8: compara1
 PORT map ( reset       =>      bus_reset,
        tbcdsec0        =>      t_bcd_sec0,
        tbcdsec1        =>      t_bcd_sec1,
        tbcdmin0        =>      t_bcd_minute0,
        tbcdmin1        =>      t_bcd_minute1,
        tbcdhour0       =>      t_bcd_hour0,
        tbcdhour1       =>      t_bcd_hour1,
        abcdsec0        =>      b_asec1_out0,
        abcdsec1        =>      b_asec1_out1,
        abcdmin0        =>      b_amin1_out0,
```

```
        abcdmin1        =>      b_amin1_out1,
        abcdhour0       =>      b_ahr1_out0,
        abcdhour1       =>      b_ahr1_out1,
        comp_out1       =>      c_out1);

block9: seg70
 PORT map ( inputs      =>      bcd_out_min0,
            segout      =>      seg_min0);

block10: seg71
 PORT map ( inputs      =>      bcd_out_min1,
            segout      =>      seg_min1);

block11: seg72
 PORT map ( inputs      =>      bcd_out_hr0.
            segout      =>      seg_hr0);

block12: seg73
 PORT map ( inputs      =>      bcd_out_hr1,
            segout      =>      seg_hr1);

 END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY Alarm IS
        PORT (  clk_15hz        :IN std_logic;
                reset           :IN  std_logic;
                setalarm        :IN std_logic;
                sethour         :IN std_logic;
                setminute       :IN std_logic;
                asec_out0       :OUT std_logic_vector(3 downto 0);
                asec_out1       :OUT std_logic_vector(3 downto 0);
                amin_out0       :OUT std_logic_vector(3 downto 0);
                amin_out1       :OUT std_logic_vector(3 downto 0);
                ahr_out0        :OUT std_logic_vector(3 downto 0);
                ahr_out1        :OUT std_logic_vector(3 downto 0));
        END Alarm;

ARCHITECTURE behavioral OF Alarm IS

COMPONENT Asecond
PORT ( clk              :IN std_logic;
        reset           :IN std_logic;
       bcdsec0          :OUT std_logic_vector (3 downto 0);
       bcdsec1          :OUT std_logic_vector (3 downto 0);
      carryout          :OUT std_logic);
END COMPONENT;
```

```
COMPONENT Aminute
PORT ( clk          :IN std_logic;
       reset        :IN std_logic;
       bcdmin0      :OUT std_logic_vector (3 downto 0);
       bcdmin1      :OUT std_logic_vector (3 downto 0);
       carryout     :OUT std_logic);
END COMPONENT;

COMPONENT Ahour
PORT ( clk          :IN std_logic;
       reset        :IN std_logic;
       bcdhour0     :OUT std_logic_vector (3 downto 0);
       bcdhour1     :OUT std_logic_vector (3 downto 0));
END COMPONENT;

SIGNAL bus_reset    :std_logic;
SIGNAL gate_reset   :std_logic;
SIGNAL to_clock     :std_logic;
SIGNAL to_clk_min   :std_logic;
SIGNAL to_clk_hr    :std_logic;
SIGNAL clk_minute   :std_logic;
SIGNAL clk_hour     :std_logic;
SIGNAL c_to_minute  :std_logic;
SIGNAL c_to_hour    :std_logic;

BEGIN
bus_reset     <= reset OR gate_reset;
to_clk_min    <= c_to_minute OR clk_minute;
to_clk_hr     <= c_to_hour OR clk_hour;
```

```
PROCESS (setalarm, sethour, setminute, clk_15hz)
BEGIN
IF (setalarm = '1') THEN
        IF0 (sethour = '0') THEN
                IF (setminute = '0') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= '0';
                ELSIF (setminute = '1') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= clk_15hz;
                        clk_hour        <= '0';
                END IF;
        ELSIF (sethour = '1') THEN
                IF (setminute = '0') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= clk_15hz;
                ELSIF (setminute = '1') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= '0';
                END IF;
        END IF;
ELSIF (setalarm = '0') THEN
        to_clock        <= '0';
```

```
        gate_reset      <= '0';
        clk_minute      <= '0';
        clk_hour        <= '0';
END IF;
END PROCESS;

block0: Asecond
PORT map (clk           =>      to_clock,
          reset         =>      bus_reset,
          bcdsec0       =>      asec_out0,
          bcdsec1       =>      asec_out1,
          carryout      =>      c_to_minute);

block1: Aminute
 PORT map (clk          =>      to_clk_min,
           reset        =>      bus_reset,
           bcdmin0      =>      amin_out0,
           bcdmin1      =>      amin_out1,
           carryout     =>      c_to_hour);

block2: Ahour
 PORT map (clk          =>      to_clk_hr,
           reset        =>      bus_reset,
           bcdhour0     =>      ahr_out0,
           bcdhour1             => ahr_out1);
END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY Alarm1 IS
PORT (  clk_15hz          :IN std_logic;
        reset             :IN  std_logic;
        setalarm          :IN std_logic;
        sethour           :IN std_logic;
        setminute         :IN std_logic;
        asec1_out0        :OUT std_logic_vector(3 downto 0);
        asec1_out1        :OUT std_logic_vector(3 downto 0);
        amin1_out0        :OUT std_logic_vector(3 downto 0);
        amin1_out1        :OUT std_logic_vector(3 downto 0);
        ahr1_out0         :OUT std_logic_vector(3 downto 0);
        ahr1_out1         :OUT std_logic_vector(3 downto 0));
END Alarm1;

ARCHITECTURE behavioral OF Alarm1 IS

COMPONENT Asecond
      PORT ( clk          :IN std_logic:
             reset        :IN std_logic;
             bcdsec0      :OUT std_logic_vector (3 downto 0);
             bcdsec1      :OUT std_logic_vector (3 downto 0);
             carryout     :OUT std_logic);
END COMPONENT;
```

```
COMPONENT Aminute
        PORT ( clk          :IN std_logic;
               reset        :IN std_logic;
               bcdmin0      :OUT std_logic_vector (3 downto 0);
               bcdmin1      :OUT std_logic_vector (3 downto 0);
               carryout     :OUT std_logic);
END COMPONENT;

COMPONENT Ahour
        PORT ( clk          :IN std_logic;
               reset        :IN std_logic;
               bcdhour0     :OUT std_logic_vector (3 downto 0);
               bcdhour1     :OUT std_logic_vector (3 downto 0));
END COMPONENT;

SIGNAL bus_reset    :std_logic;
SIGNAL gate_reset   :std_logic;
SIGNAL to_clock     :std_logic;
SIGNAL to_clk_min   :std_logic;
SIGNAL to_clk_hr    :std_logic;
SIGNAL clk_minute   :std_logic;
SIGNAL clk_hour     :std_logic;
SIGNAL c_to_minute  :std_logic;
SIGNAL c_to_hour    :std_logic;

BEGIN
 bus_reset      <= reset OR gate_reset;
 to_clk_min     <= c_to_minute OR clk_minute;
 to_clk_hr      <= c_to_hour OR clk_hour;
PROCESS (setalarm, sethour, setminute, clk_15hz)
```

```
BEGIN
IF (setalarm = '1') THEN
        IF (sethour = '0') THEN
                IF (setminute = '0') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= '0';
                ELSIF (setminute = '1') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= clk_15hz;
                        clk_hour        <= '0';
                END IF;
        ELSIF (sethour = '1') THEN
                IF (setminute = '0') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= clk_15hz;
                ELSIF (setminute = '1') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= '0';
                END IF;
        END IF;
ELSIF (setalarm = '0') THEN
        to_clock        <= '0';
        gate_reset      <= '0';
```

```
        clk_minute      <= '0';
        clk_hour        <= '0';
END IF;
END PROCESS;

block0: Asecond
PORT map ( clk          =>      to_clock,
           reset        =>      bus_reset,
           bcdsc0       =>      asec1_out0,
           bcdsec1      =>      asec1_out1,
           carryout     =>      c_to_minute);

block1: Aminute
 PORT map ( clk         =>      to_clk_min,
            reset       =>      bus_reset,
            bcdmin0     =>      amin1_out0,
            bcdmin1     =>      amin1_out1,
            carryout    =>      c_to_hour);

block2: Ahour
 PORT map ( clk         =>      to_clk_hr,
            reset       =>      bus_reset,
            bcdhour0    =>      ahr1_out0,
            bcdhour1    =>      ahr1_out1);
END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.all;

ENTITY Ahour IS
  PORT (clk          :IN  std_logic;
        reset        :IN  std_logic;
        bcdhour0     :OUT std_logic_vector (3 downto 0);
        bcdhour1     :OUT std_logic_vector (3 downto 0));
END Ahour;

ARCHITECTURE behaveioral OF Ahour IS
BEGIN
      W: PROCESS (clk)
           variable bus0 :  std_logic_vector( 3 downto 0):= "0000";
           variable bus1 :  std_logic_vector( 3 downto 0):= "0000";

      BEGIN
      IF (clk'event and clk = '1') THEN
           IF (reset = '0') THEN
                bus0 := (bus0 + "0001");
                IF ( bus0 = "1010") THEN
                     bus0 := "0000";
                     bus1 := (bus1 + "0001");
                END IF;
                     IF (bus1 = "0010") and (bus0 = "0100") THEN
                       bus0 := "0000";
                       bus1 := "0000";
                     END IF;
           ELSE
```

```
                    bus0 := "0000";
                    bus1 := "0000";
                END IF;
            bcdhour0 <= bus0;
            bcdhour1 <= bus1;
        END IF;
    END PROCESS W;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.all;

ENTITY Aminute IS
        PORT ( clk              :IN  std_logic;
               reset            :IN  std_logic;
               bcdmin0          :OUT std_logic_vector (3 downto 0);
               bcdmin1          :OUT std_logic_vector (3 downto 0);
               carryout         :OUT std_logic);
END Aminute;

ARCHITECTURE behaveioral OF Aminute IS
 BEGIN
        W: PROCESS (clk)
               variable bus0 :  std_logic_vector( 3 downto 0):= "0000";
               variable bus1 :  std_logic_vector( 3 downto 0):= "0000";

               BEGIN
               IF (clk'event and clk = '1') THEN
                        IF ( reset = '0') THEN
                                bus0 := (bus0 + "0001");
                                        IF ( bus0 = "1010") THEN
                                                bus0 := "0000";
                                                bus1 := (bus1 + "0001");
                                                carryout <= '0';
                                          END IF;
                                        IF ( bus1 = "0110") THEN
                                                bus0 := "0000";
```

```
                bus1 := "0000";
                carryout <= '1';
            ELSE
                carryout <= '0';
            END IF;
        ELSE
            bus0 := "0000";
            bus1 := "0000";
            carryout <= '0';
        END IF;
        bcdmin0 <= bus0;
        bcdmin1 <= bus1;
    END IF;
  END PROCESS W;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.all;

ENTITY Asecond IS
  PORT(clk          :IN  std_logic;
       reset        :IN  std_logic;
       bcdsec0      :OUT std_logic_vector (3 downto 0);
       bcdsec1      :OUT std_logic_vector (3 downto 0);
       carryout     :OUT std_logic);
END Asecond;

ARCHITECTURE behaveioral OF Asecond IS
BEGIN
      W: PROCESS (clk)
          variable bus0 :  std_logic_vector( 3 downto 0):= "0000";
          variable bus1 :  std_logic_vector( 3 downto 0):= "0000";

          BEGIN
          IF (clk'event and clk = '1') THEN
              IF (reset = '0') THEN
                      bus0 := (bus0 + "0001");
                          IF ( bus0 = "1010")THEN
                              bus0 := "0000";
                              bus1 := (bus1 + "0001");
                              carryout <= '0';
                          END IF;
                          IF (bus1 = "0110")THEN
                              bus0 := "0000";
```

```
                                        bus1 := "0000";
                                        carryout <= '1';
                                ELSE
                                        carryout <= '0';
                                END IF;
                        ELSE
                                bus0 := "0000";
                                bus1 := "0000";
                        END IF;
                bcdsec0 <= bus0;
                bcdsec1 <= bus1;
        END IF;
        END PROCESS W;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY Timer IS
        PORT ( clk_1hz      :IN  std_logic;
               clk_15hz     :IN std_logic;
               reset        :IN std_logic;
               settime      :IN std_logic;
               sethour      :IN std_logic;
               setminut     :IN std_logic;
               tsec_out     :OUT std_logic_vector(3 downto 0);
               tsec_out     :OUT std_logic_vector(3 downto 0);
               tmin_out0    :OUT std_logic_vector(3 downto 0);
               tmin_out1    :OUT std_logic_vector(3 downto 0);
               thr_out0     :OUT std_logic_vector(3 downto 0);
               thr_out1     :OUT std_logic_vector(3 downto 0));
END Timer;
ARCHITECTURE behavioral OF Timer IS

COMPONENT Tsecond
        port ( clk          :IN std_logic;
               reset        :IN std_logic;
               bcdsec0      :OUT std_logic_vector (3 downto 0);
               bcdsec1      :OUT std_logic_vector (3 downto 0);
               carryout     :OUT std_logic);
END COMPONENT;
```

```
COMPONENT Tminute
        PORT ( clk              :IN std_logic;
               reset            :IN std_logic;
               bcdmin0          :OUT std_logic_vector (3 downto 0);
               bcdmin1          :OUT std_logic_vector (3 downto 0);
               carryout         :OUT std_logic);
END COMPONENT;
COMPONENT thour
        PORT ( clk              :IN std_logic;
               reset            :IN std_logic;
               bcdhour0         :OUT std_logic_vector (3 downto 0);
               bcdhour1         :OUT std_logic_vector (3 downto 0));
END COMPONENT;

SIGNAL bus_reset                :std_logic;
SIGNAL gate_reset               :std_logic;
SIGNAL to_clock                 :std_logic;
SIGNAL to_clk_min               :std_logic;
SIGNAL to_clk_hr                :std_logic;
SIGNAL clk_minute               :std_logic;
SIGNAL clk_hour                 :std_logic;
SIGNAL c_to_minute              :std_logic;
SIGNAL c_to_hour                :std_logic;

BEGIN
        bus_reset       <= reset OR gate_reset;
        to_clk_min      <= c_to_minute OR clk_minute;
        to_clk_hr       <= c_to_hour OR clk_hour;
PROCESS (settime, sethour, setminute. clk_1hz, clk_15hz)
BEGIN
```

```
IF (settime = '1') THEN
        IF (sethour = '0') THEN
                IF (setminute = '0') THEN
                        to_clock        <= clk_1hz;
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= '0';
                ELSIF (setminute = '1') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= clk_15hz;
                        clk_hour        <= '0';
                END IF;
        ELSIF (sethour = '1') THEN
                IF (setminute = '0') THEN
                        to_clock        <= '0';
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= clk_15hz;
                ELSIF (setminute = '1') THEN
                        to_clock        <= clk_1hz;
                        gate_reset      <= '0';
                        clk_minute      <= '0';
                        clk_hour        <= '0';
                END IF;
        END IF;
ELSIF (settime = '0') THEN
        to_clock        <= clk_1hz;
        gate_reset      <= '0';
        clk_minute      <= '0';
```

```
            clk_hour        <= '0';
        END IF;
END PROCESS;

block0: Tsecond
        PORT map (clk           => to_clock,
                  reset         =>      bus_reset,
                  bcdsec0       =>      tsec_out0,
                  bcdsec1       =>      tsec_out1,
                  carryout      =>      c_to_minute);

block1: Tminute
        PORT map ( clk          =>      to_clk_min,
                   reset        =>      bus_reset,
                   bcdmin0      =>      tmin_out0,
                   bcdmin1      =>      tmin_out1,
                   carryout     =>      c_to_hour);

block2: thour
        PORT map (clk           =>      to_clk_hr,
                  reset         =>      bus_reset,
                  bcdhour0      =>      thr_out0,
                  bcdhour1      =>      thr_out1);
END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY Thour IS
        PORT (clk           :IN  std_logic;
              reset         :IN  std_logic;
              bcdhour0      :OUT std_logic_vector (3 downto 0);
              bcdhour1      :OUT std_logic_vector (3 downto 0));
END Thour;

ARCHITECTURE behaveioral OF Thour IS
BEGIN
      W: PROCESS (clk)
            variable bus0   : std_logic_vector( 3 downto 0):= "0000";
            variable bus1   : std_logic_vector( 3 downto 0):= "0000";
            BEGIN
            IF (clk'event and clk = '1') THEN
                IF (reset = '0')THEN
                    bus0 := (bus0 + "0001");
                        IF ( bus0 = "1010") THEN
                            bus0 := "0000";
                            bus1 := (bus1 + "0001");
                        END IF;
                        IF (bus1 = "0010") and (bus0 = "0100") THEN
                            bus0 := "0000";
                            bus1 := "0000";
                        END IF;
```

```
                ELSE
                        bus0 := "0000";
                        bus1 := "0000";
                END IF;
                bcdhour0 <= bus0;
                bcdhour1 <= bus1;
            END IF;
        END PROCESS W;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY Tminute IS
        PORT ( clk              :IN  std_logic;
               reset            :IN  std_logic;
               bcdmin0          :OUT std_logic_vector (3 downto 0);
               bcdmin1          :OUT std_logic_vector (3 downto 0);
               carryout         :OUT std_logic);
END Tminute;

ARCHITECTURE behaveioral OF Tminute IS
 BEGIN
        W: PROCESS (clk)
               variable bus0 :  std_logic_vector(3 downto 0):= "0000";
               variable bus1 :  std_logic_vector( 3 downto 0):= "0000";
               BEGIN
               IF (clk'event and clk = '1') THEN
                        IF (reset = '0')THEN
                                bus0 := (bus0 + "0001");
                                        IF ( bus0 = "1010") THEN
                                                bus0 := "0000";
                                                bus1 := (bus1 + "0001");
                                                carryout <= '0';
                                        END IF;
                                        IF ( bus1 = "0110")THEN
                                                bus0 := "0000";
```

```
                    bus1 := "0000";
                    carryout <= '1';
                ELSE
                    carryout <= '0';
                END IF;
            ELSE
                bus0 := "0000";
                bus1 := "0000";
                carryout <= '0';
            END IF;
            bcdmin0 <= bus0;
            bcdmin1 <= bus1;
        END IF;
    END PROCESS W;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY Tsecond IS
        PORT( clk                   :IN  std_logic;
              reset                 :IN  std_logic;
              bcdsec0               :OUT std_logic_vector (3 downto 0);
              bcdsec1               :OUT std_logic_vector (3 downto 0);
              carryout              :OUT std_logic);
END Tsecond;

ARCHITECTURE behaveioral OF Tsecond IS
BEGIN
        W: PROCESS (CLK)
              variable bus0 :  std_logic_vector( 3 downto 0):= "0000";
              variable bus1 :  std_logic_vector( 3 downto 0):= "0000";
              BEGIN
              IF (clk'event and clk = '1') THEN
                    IF (reset = '0')THEN
                          bus0 := (bus0 + "0001");
                                IF ( bus0 = "1010") THEN
                                      bus0 := "0000";
                                      bus1 := (bus1 + "0001");
                                      carryout <= '0';
                                END IF;
                                IF (bus1 = "0110")THEN
                                      bus0 := "0000";
```

```
                    bus1 := "0000";
                    carryout <= '1';
                ELSE
                    carryout <= '0';
                END IF;
            ELSE
                bus0 := "0000";
                bus1 := "0000";
                carryout <= '0';
            END IF;
            bcdsec0 <= bus0;
            bcdsec1 <= bus1;
        END IF;
    END PROCESS W;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY compara IS
        PORT (  reset           :IN  std_logic;
                tbcdsec0        :IN  std_logic_vector(3 downto 0);
                tbcdsec1        :IN  std_logic_vector(3 downto 0);
                tbcdmin0        :IN  std_logic_vector(3 downto 0);
                tbcdmin1        :IN  std_logic_vector(3 downto 0);
                tbcdhour0       :IN  std_logic_vector(3 downto 0);
                tbcdhour1       :IN  std_logic_vector(3 downto 0);
                abcdsec0        :IN  std_logic_vector(3 downto 0);
                abcdsec1        :IN  std_logic_vector(3 downto 0);
                abcdmin0        :IN  std_logic_vector(3 downto 0);
                abcdmin1        :IN  std_logic_vector(3 downto 0);
                abcdhour0       :IN  std_logic_vector(3 downto 0);
                abcdhour1       :IN  std_logic_vector(3 downto 0);
                comp_out        :OUT std_logic);
END compara;

ARCHITECTURE behavioral OF compara IS
BEGIN
PROCESS (tbcdsec0,tbcdsec1,tbcdmin0,tbcdmin1,tbcdhour0,tbcdhour1,
            abcdsec0,abcdsec1,abcdmin0,abcdmin1,abcdhour0,abcdhour1,reset)
BEGIN
        IF (reset = '1') THEN
                comp_out <= '0';
```

```
        ELSE
                IF (tbcdsec0 = abcdsec0) and
                   (tbcdsec1 = abcdsec1) and
                   (tbcdmin0 = abcdmin0)  and
                   (tbcdmin1 = abcdmin1)  and
                   (tbcdhour0 = abcdhour0) and
                   (tbcdhour1 = abcdhour1) THEN
                        comp_out <= '1';
                ELSE
                        comp_out <= '0';
                END IF;
        END IF;
END PROCESS ;
END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY compara1 IS
        PORT (  reset           :IN  std_logic;
                tbcdsec0        :IN  std_logic_vector(3 downto 0);
                tbcdsec1        :IN  std_logic_vector(3 downto 0);
                tbcdmin0        :IN  std_logic_vector(3 downto 0);
                tbcdmin1        :IN  std_logic_vector(3 downto 0);
                tbcdhour0       :IN  std_logic_vector(3 downto 0);
                tbcdhour1       :IN  std_logic_vector(3 downto 0);
                abcdsec0        :IN  std_logic_vector(3 downto 0);
                abcdsec1        :IN  std_logic_vector(3 downto 0);
                abcdmin0        :IN  std_logic_vector(3 downto 0);
                abcdmin1        :IN  std_logic_vector(3 downto 0);
                abcdhour0       :IN  std_logic_vector(3 downto 0);
                abcdhour1       :IN  std_logic_vector(3 downto 0);
                comp_out1       :OUT std_logic);
END compara1;
ARCHITECTURE behavioral OF compara1 IS
BEGIN
PROCESS (tbcdsec0,tbcdsec1,tbcdmin0,tbcdmin1,tbcdhour0,tbcdhour1,
            abcdsec0,abcdsec1,abcdmin0,abcdmin1,abcdhour0,abcdhour1,reset)
BEGIN
        IF (reset = '1') THEN
                comp_out1 <= '0';
        ELSE
```

```
            IF (tbcdsec0 = abcdsec0) and
              (tbcdsec1 = abcdsec1) and
              (tbcdmin0 = abcdmin0)  and
              (tbcdmin1 = abcdmin1)  and
             (tbcdhour0 = abcdhour0) and
             (tbcdhour1 = abcdhour1) THEN
                    comp_out1 <= '1';
            ELSE
                    comp_out1 <= '0';
            END IF;
        END IF;
    END PROCESS ;
END behavioral;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.All;

LIBRARY synth;
USE synth.vhdlsynth.All;

ENTITY Control IS
    PORT ( in_puta          :IN std_logic;
           in_putb          :IN std_logic;
           out_0            :OUT std_logic;
           out_1            :OUT std_logic);
END Control;

ARCHITECTURE Behaveioral OF Control IS
BEGIN
    PROCESS (in_puta, in_putb)
    BEGIN
    IF (in_puta = '1') AND (in_putb = '0') THEN
            out_0 <= '1';
            out_1 <= '0';
    ELSIF (in_puta = '0') AND (in_putb = '1') THEN
            out_0 <= '0';
            out_1 <= '1';
    ELSE
            out_0 <='0';
            out_1 <='0';
    END IF;
    END PROCESS;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY multi IS
        PORT ( set_a            :IN  std_logic;
               set_b            :IN  std_logic;
               a0_min0          :IN  std_logic_vector(3 downto 0);
               a0_min1          :IN  std_logic_vector(3 downto 0);
               a0_hour0         :IN  std_logic_vector(3 downto 0);
               a0_hour1         :IN  std_logic_vector(3 downto 0);
               a1_min0          :IN  std_logic_vector(3 downto 0);
               a1_min1          :IN  std_logic_vector(3 downto 0);
               a1_hour0         :IN  std_logic_vector(3 downto 0);
               a1_hour1         :IN  std_logic_vector(3 downto 0);
               bcd_minute0      :OUT std_logic_vector(3 downto 0);
               bcd_minute1      :OUT std_logic_vector(3 downto 0);
               bcd_hour0        :OUT std_logic_vector(3 downto 0);
               bcd_hour1        :OUT std_logic_vector(3 downto 0));
END multi;
ARCHITECTURE behavioral OF multi IS
BEGIN
        PROCESS (set_a, set_b, a0_min0,a0_min1,a0_hour0,a0_hour1,
                a1_min0,a1_min1,a1_hour0,a1_hour1)
        BEGIN
        IF (set_a = '1') AND (set_b = '0') THEN
                bcd_minute0 <= a0_min0 ;
                bcd_minute1 <= a0_min1 ;
```

```
            bcd_hour0 <= a0_hour0;
            bcd_hour1 <= a0_hour1;
        ELSIF (set_a = '0') AND (set_b = '1') THEN
            bcd_minute0 <= a1_min0 ;
            bcd_minute1 <= a1_min1 ;
            bcd_hour0 <= a1_hour0;
            bcd_hour1 <= a1_hour1;
        ELSE
            bcd_minute0 <= "0000";
            bcd_minute1 <= "0000";
            bcd_hour0 <= "0000";
            bcd_hour1 <= "0000";
        END IF;
        END PROCESS;
END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY multiplx IS
        PORT ( setalarm         :IN  std_logic;
                tbcdmin0        :IN  std_logic_vector(3 downto 0);
                tbcdmin1        :IN  std_logic_vector(3 downto 0);
                tbcdhour0       :IN  std_logic_vector(3 downto 0);
                tbcdhour1       :IN  std_logic_vector(3 downto 0);
                abcdmin0        :IN  std_logic_vector(3 downto 0);
                abcdmin1        :IN  std_logic_vector(3 downto 0);
                abcdhour0       :IN  std_logic_vector(3 downto 0);
                abcdhour1       :IN  std_logic_vector(3 downto 0);
                bcd_minute0     :OUT std_logic_vector(3 downto 0);
                bcd_minute1     :OUT std_logic_vector(3 downto 0);
                bcd_hour0       :OUT std_logic_vector(3 downto 0);
                bcd_hour1       :OUT std_logic_vector(3 downto 0));
END multiplx;

ARCHITECTURE behavioral OF multiplx IS
BEGIN
        PROCESS (setalarm ,tbcdmin0,tbcdmin1,tbcdhour0,tbcdhour1,
                        abcdmin0,abcdmin1,abcdhour0,abcdhour1)
        BEGIN
        IF (setalarm = '1')THEN
                bcd_minute0 <= abcdmin0 ;
                bcd_minute1 <= abcdmin1 ;
```

```
                bcd_hour0 <= abcdhour0;
                bcd_hour1 <= abcdhour1;
        ELSIF (setalarm = '0') THEN
                bcd_minute0 <= tbcdmin0 ;
                bcd_minute1 <= tbcdmin1 ;
                bcd_hour0 <= tbcdhour0;
                bcd_hour1 <= tbcdhour1;
        END IF;
        END PROCESS;
END behavioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY osc IS
        PORT ( clk              :IN  std_logic;
               setin            :IN std_logic;
               clkout           :OUT std_logic);
END osc;

ARCHITECTURE behave OF osc IS
 BEGIN
        W: PROCESS (clk)
               variable a : std_logic_vector (3 downto 0):= "0000";
               BEGIN
               IF (clk'event and clk = '1') THEN
                      IF (setin = '0') THEN
                             a := (a + "0001");
                             IF ( a = "1111") THEN
                                    a := "0000";
                                    clkout <= '1';
                             ELSE
                                    clkout <= '0';
                             END IF;
                      ELSIF (setin = '1') THEN
                             a := "0000";
                             clkout <= '0';
                      END IF;
```

```
        END IF;
    END PROCESS W;
END behave;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY seg70 IS
        PORT ( inputs           : IN std_logic_vector(3 downto 0);
                segout          : OUT std_logic_vector(6 downto 0) );
END seg70;

ARCHITECTURE behaveioral OF seg70 IS

CONSTANT y0 : std_logic_vector(0 to 6) := "1111110";
CONSTANT y1 : std_logic_vector(0 to 6) := "0110000";
CONSTANT y2 : std_logic_vector(0 to 6) := "1101101";
CONSTANT y3 : std_logic_vector(0 to 6) := "1111001";
CONSTANT y4 : std_logic_vector(0 to 6) := "0110011";
CONSTANT y5 : std_logic_vector(0 to 6) := "1011011";
CONSTANT y6 : std_logic_vector(0 to 6) := "1011111";
CONSTANT y7 : std_logic_vector(0 to 6) := "1110000";
CONSTANT y8 : std_logic_vector(0 to 6) := "1111111";
CONSTANT y9 : std_logic_vector(0 to 6) := "1111011";
CONSTANT i0 : std_logic_vector(0 to 3) := "0000";
CONSTANT i1 : std_logic_vector(0 to 3) := "0001";
CONSTANT i2 : std_logic_vector(0 to 3) := "0010";
CONSTANT i3 : std_logic_vector(0 to 3) := "0011";
CONSTANT i4 : std_logic_vector(0 to 3) := "0100";
CONSTANT i5 : std_logic_vector(0 to 3) := "0101";
CONSTANT i6 : std_logic_vector(0 to 3) := "0110";
```

```
CONSTANT i7 : std_logic_vector(0 to 3) := "0111";
CONSTANT i8 : std_logic_vector(0 to 3) := "1000";
CONSTANT i9 : std_logic_vector(0 to 3) := "1001";
BEGIN
        PROCESS (inputs)
        BEGIN
                CASE inputs IS
                        WHEN i0 => segout <= y0;
                        WHEN i1 => segout <= y1;
                        WHEN i2 => segout <= y2;
                        WHEN i3 => segout <= y3;
                        WHEN i4 => segout <= y4;
                        WHEN i5 => segout <= y5;
                        WHEN i6 => segout <= y6;
                        WHEN i7 => segout <= y7;
                        WHEN i8 => segout <= y8;
                        WHEN i9 => segout <= y9;
                        WHEN OTHERS => segout <=y0;
                END CASE;
        END PROCESS;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY seg71 IS
        PORT ( inputs           : IN std_logic_vector(3 downto 0);
               segout           : OUT std_logic_vector(6 downto 0) );
END seg71;

ARCHITECTURE behaveioral OF seg71 IS
                CONSTANT y0 : std_logic_vector(0 to 6) := "1111110";
                CONSTANT y1 : std_logic_vector(0 to 6) := "0110000";
                CONSTANT y2 : std_logic_vector(0 to 6) := "1101101";
                CONSTANT y3 : std_logic_vector(0 to 6) := "1111001";
                CONSTANT y4 : std_logic_vector(0 to 6) := "0110011";
                CONSTANT y5 : std_logic_vector(0 to 6) := "1011011";
                CONSTANT y6 : std_logic_vector(0 to 6) := "1011111";
                CONSTANT y7 : std_logic_vector(0 to 6) := "1110000";
                CONSTANT y8 : std_logic_vector(0 to 6) := "1111111";
                CONSTANT y9 : std_logic_vector(0 to 6) := "1111011";
                CONSTANT i0 : std_logic_vector(0 to 3) := "0000";
                CONSTANT i1 : std_logic_vector(0 to 3) := "0001";
                CONSTANT i2 : std_logic_vector(0 to 3) := "0010";
                CONSTANT i3 : std_logic_vector(0 to 3) := "0011";
                CONSTANT i4 : std_logic_vector(0 to 3) := "0100";
                CONSTANT i5 : std_logic_vector(0 to 3) := "0101";
                CONSTANT i6 : std_logic_vector(0 to 3) := "0110";
                CONSTANT i7 : std_logic_vector(0 to 3) := "0111";
```

```
        CONSTANT i8 : std_logic_vector(0 to 3) := "1000";
        CONSTANT i9 : std_logic_vector(0 to 3) := "1001";

BEGIN
        PROCESS (inputs)
        BEGIN
            CASE inputs     IS
                WHEN i0 => segout <= y0;
                WHEN i1 => segout <= y1;
                WHEN i2 => segout <= y2;
                WHEN i3 => segout <= y3;
                WHEN i4 => segout <= y4;
                WHEN i5 => segout <= y5;
                WHEN i6 => segout <= y6;
                WHEN i7 => segout <= y7;
                WHEN i8 => segout <= y8;
                WHEN i9 => segout <= y9;
                WHEN others => segout <=y0;
            END CASE;
        END PROCESS;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY seg72 IS
        PORT ( inputs          : IN std_logic_vector(3 downto 0);
               segout          : OUT std_logic_vector(6 downto 0) );
END seg72;

ARCHITECTURE behaveioral OF seg72 IS
                CONSTANT y0 : std_logic_vector(0 to 6) := "1111110";
                CONSTANT y1 : std_logic_vector(0 to 6) := "0110000";
                CONSTANT y2 : std_logic_vector(0 to 6) := "1101101";
                CONSTANT y3 : std_logic_vector(0 to 6) := "1111001";
                CONSTANT y4 : std_logic_vector(0 to 6) := "0110011";
                CONSTANT y5 : std_logic_vector(0 to 6) := "1011011";
                CONSTANT y6 : std_logic_vector(0 to 6) := "1011111";
                CONSTANT y7 : std_logic_vector(0 to 6) := "1110000";
                CONSTANT y8 : std_logic_vector(0 to 6) := "1111111";
                CONSTANT y9 : std_logic_vector(0 to 6) := "1111011";
                CONSTANT i0 : std_logic_vector(0 to 3) := "0000";
                CONSTANT i1 : std_logic_vector(0 to 3) := "0001";
                CONSTANT i2 : std_logic_vector(0 to 3) := "0010";
                CONSTANT i3 : std_logic_vector(0 to 3) := "0011";
                CONSTANT i4 : std_logic_vector(0 to 3) := "0100";
                CONSTANT i5 : std_logic_vector(0 to 3) := "0101";
                CONSTANT i6 : std_logic_vector(0 to 3) := "0110";
                CONSTANT i7 : std_logic_vector(0 to 3) := "0111";
```

```
        CONSTANT i8 : std_logic_vector(0 to 3) := "1000";
        CONSTANT i9 : std_logic_vector(0 to 3) := "1001";

BEGIN
        PROCESS (inputs)
        BEGIN
            CASE inputs     IS
                WHEN i0 => segout <= y0;
                WHEN i1 => segout <= y1;
                WHEN i2 => segout <= y2;
                WHEN i3 => segout <= y3;
                WHEN i4 => segout <= y4;
                WHEN i5 => segout <= y5;
                WHEN i6 => segout <= y6;
                WHEN i7 => segout <= y7;
                WHEN i8 => segout <= y8;
                WHEN i9 => segout <= y9;
                WHEN others => segout <=y0;
            END CASE;
        END PROCESS;
END behaveioral;
```

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

LIBRARY synth;
USE synth.vhdlsynth.ALL;

ENTITY seg73 IS
        PORT ( inputs           : IN std_logic_vector(3 downto 0);
               segout           : OUT std_logic_vector(6 downto 0) );
END seg73;

ARCHITECTURE behaveioral OF seg73 IS
                CONSTANT y0 : std_logic_vector(0 to 6) := "1111110";
                CONSTANT y1 : std_logic_vector(0 to 6) := "0110000";
                CONSTANT y2 : std_logic_vector(0 to 6) := "1101101";
                CONSTANT y3 : std_logic_vector(0 to 6) := "1111001";
                CONSTANT y4 : std_logic_vector(0 to 6) := "0110011";
                CONSTANT y5 : std_logic_vector(0 to 6) := "1011011";
                CONSTANT y6 : std_logic_vector(0 to 6) := "1011111";
                CONSTANT y7 : std_logic_vector(0 to 6) := "1110000";
                CONSTANT y8 : std_logic_vector(0 to 6) := "1111111";
                CONSTANT y9 : std_logic_vector(0 to 6) := "1111011";
                CONSTANT i0 : std_logic_vector(0 to 3) := "0000";
                CONSTANT i1 : std_logic_vector(0 to 3) := "0001";
                CONSTANT i2 : std_logic_vector(0 to 3) := "0010";
                CONSTANT i3 : std_logic_vector(0 to 3) := "0011";
                CONSTANT i4 : std_logic_vector(0 to 3) := "0100";
                CONSTANT i5 : std_logic_vector(0 to 3) := "0101";
                CONSTANT i6 : std_logic_vector(0 to 3) := "0110";
                CONSTANT i7 : std_logic_vector(0 to 3) := "0111";
```

```
        CONSTANT i8 : std_logic_vector(0 to 3) := "1000";
        CONSTANT i9 : std_logic_vector(0 to 3) := "1001";

BEGIN
        PROCESS (inputs)
        BEGIN
            CASE inputs     IS
                WHEN i0 => segout <= y0;
                WHEN i1 => segout <= y1;
                WHEN i2 => segout <= y2;
                WHEN i3 => segout <= y3;
                WHEN i4 => segout <= y4;
                WHEN i5 => segout <= y5;
                WHEN i6 => segout <= y6;
                WHEN i7 => segout <= y7;
                WHEN i8 => segout <= y8;
                WHEN i9 => segout <= y9;
                WHEN others => segout <=y0;
            END CASE;
        END PROCESS;
END behaveioral;
```

ภาคผนวก ข

เอกสารของอุปกรณ์ FPGAs ตระกูล XC4000A

XILINX®

XC4000A
Logic Cell Array Family

Product Specifications

Features

- Third Generation Field-Programmable Gate Arrays
  - Abundant flip-flops
  - Flexible function generators
  - On-chip ultra-fast RAM
  - Dedicated high-speed carry-propagation circuit
  - Wide edge decoders (two per edge)
  - Hierarchy of interconnect lines
  - Internal 3-state bus capability
  - Eight global low-skew clock or signal distribution network
- Flexible Array Architecture
  - Programmable logic blocks and I/O blocks
  - Programmable interconnects and wide decoders
- Sub-micron CMOS Process
  - High-speed logic and Interconnect
  - Low power consumption
- Systems-Oriented Features
  - IEEE 1149.1-compatible boundary-scan logic support
  - Programmable output slew rate (4 modes)
  - Programmable input pull-up or pull-down resistors
  - 24-mA sink current per output (48 per pair)
- Configured by Loading Binary File
  - Unlimited reprogrammability
  - Six programming modes
- XACT Development System runs on '386/'486-type PC, NEC PC, Apollo, Sun-4, and Hewlett-Packard 700 Series
  - Interfaces to popular design environments like Viewlogic, Mentor Graphics and OrCAD
  - Fully automatic partitioning, placement and routing
  - Interactive design editor for design optimization
  - 288 macros, 34 hard macros, RAM/ROM compiler

Description

The XC4000A family of FPGAs offers four devices at the low end of the XC4000 family complexity range. XC4000A differs from XC4000 in four areas: fewer routing resources, fewer wide-edge decoders, higher output sink current, and improved output slew-rate control.

- The XC4000 routing structure is optimized for smaller designs, naturally requiring fewer routing resources. The XC4000A devices have four Longlines and four single-length lines per row and column, while the XC4000 devices have six Longlines and eight single-length lines per row and column. This results in a smaller chip area and lower cost per device.
- XC4000A has two wide-edge decoders on every device edge, while the XC4000 has four. All other wide-decoder features are identical in XC4000 and XC4000A.
- XC4000A outputs are specified at 24 mA, sink current, while XC4000 outputs are specified at 12 mA. The source current is the same 4 mA for both families.
- The XC4000A family offers a more sophisticated output slew-rate control structure with four configurable options for each individual output driver: fast, medium fast, medium slow, and slow. Slew-rate control can alleviate ground-bounce problems when multiple outputs switch simultaneously, and it can reduce or eliminate crosstalk and transmission-line effects on printed circuit boards.

Note that the XC4003 and XC4005 devices are available in both flavors, the lower-priced XC4003A/XC4005A with reduced routing, and the higher-priced XC4003/XC4005 with more abundant routing resources. The XC4000A devices are intended for less demanding and more structured designs, and the XC4000 devices for more random designs requiring additional routing resources.

The equivalent devices are pin-compatible and are available in identical packages, but they are not bitstream compatible. In order to move from a XC4000A to a XC4000, or vice versa, the design must be recompiled.

Table 1. The XC4000A Family of Field-Programmable Gate Arrays

Device	XC4002A	XC4003A	XC4004A	XC4005A
Appr. Gate Count	2,000	3,000	4,000	5,000
CLB Matrix	8 x 8	10 x 10	12 x 12	14 x 14
Number of CLBs	64	100	144	196
Number of Flip-Flops	256	360	480	616
Max Decode Inputs (per side)	24	30	36	42
Max RAM Bits	2,048	3,200	4,608	6,272
Number of IOBs	64	80	96	112

Absolute Maximum Ratings

Symbol	Description		Units
V_{CC}	Supply voltage relative to GND	–0.5 to +7.0	V
V_{IN}	Input voltage with respect to GND	–0.5 to V_{CC} +0.5	V
V_{TS}	Voltage applied to 3-state output	–0.5 to V_{CC} +0.5	V
T_{STG}	Storage temperature (ambient)	–65 to + 150	°C
T_{SOL}	Maximum soldering temperature (10 s @ 1/16 in. = 1.5 mm)	+ 260	°C
T_J	Junction temperature	+ 150	°C

Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Operating Conditions

Symbol	Description	Min	Max	Units
V_{CC}	Supply voltage relative to GND Commercial 0°C to 85°C junction	4.75	5.25	V
	Supply voltage relative to GND Industrial -40°C to 100°C junction	4.5	5.5	V
	Supply voltage relative to GND Military –55°C to 125°C case	4.5	5.5	V
V_{IH}	High-level input voltage (XC4000 has TTL-like input thresholds)	2.0	Vcc	V
V_{IL}	Low-level input voltage (XC4000 has TTL-like input thresholds)	0	0.8	V
T_{IN}	Input signal transition time		250	ns

At junction temperatures above those listed as Operating conditions, all delay parameters increase by 0.35% per °C.

DC Characteristics Over Operating Conditions

Symbol	Description	Min	Max	Units
V_{OH}	High-level output voltage @ I_{OH} = –4.0 mA, V_{CC} min	2.4		V
V_{OL}	Low-level output voltage @ I_{OL} = 24 mA, V_{CC} min (Note 1)		0.4	V
I_{CCO}	Quiescent LCA supply current (Note 2)		10	mA
I_{IL}	Leakage current	–10	+10	µA
C_{IN}	Input capacitance (sample tested)		15	pF
I_{RIN}	Pad pull-up (when selected) @ V_{IN} = 0V (sample tested)	0.02	0.25	mA
I_{RLL}	Horizontal Long Line pull-up (when selected) @ logic Low	0.2	2.5	mA

Note: 1. With 50% of the outputs simultaneously sinking 24 mA.
2. With no output current loads, no active input or longline pull-up resistors, all package pins at V_{CC} or GND, and the LCA configured with a MakeBits tie option.

Wide Decoder Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, there derived from benchmark timing patterns. The following guidelines reiflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description	Symbol	Speed Grade / Device	-6 Max	-5 Max	-4 Max	Units
Full length, both pull-ups, inputs from IOB I-pins	T_{WAF}	XC4002A	8.5	7.5	[illegible]	ns
		XC4003A	9.0	8.0	5.0	ns
		XC4004A	9.5	8.5	[illegible]	ns
		XC4005A	10.0	9.0	6.0	ns
Full length, both pull-ups inputs from internal logic	T_{WAFL}	XC4002A	11.5	10.5	[illegible]	ns
		XC4003A	12.0	11.0	7.0	ns
		XC4004A	12.5	11.5	[illegible]	ns
		XC4005A	13.0	12.0	8.0	ns
Half length, one pull-up inputs from IOB I-pins	T_{WAO}	XC4002A	8.5	7.5	[illegible]	ns
		XC4003A	9.0	8.0	6.0	ns
		XC4004A	9.5	8.5	[illegible]	ns
		XC4005A	10.0	9.0	7.0	ns
Half length, one pull-up inputs from internal logic	T_{WAOL}	XC4002A	11.5	10.5	[illegible]	ns
		XC4003A	12.0	11.0	8.0	ns
		XC4004A	12.5	11.5	[illegible]	ns
		XC4005A	13.0	12.0	9.0	ns

Note: These delays are specified from the decoder input to the decoder output. For pin-to-pin delays, add the input delay (T_{PIO}) and output delay (one of 4 modes), as listed on page 2-70.

Global Buffer Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Description	Symbol	Speed Grade / Device	-6 Max	-5 Max	-4 Max	Units
Global Signal Distribution						
From pad through primary buffer, to any clock k	T_{PG}	XC4002A	7.7	5.7	[illegible]	ns
		XC4003A	7.8	5.8	[illegible]	ns
		XC4004A	7.9	5.9	[illegible]	ns
		XC4005A	8.0	6.0	[illegible]	ns
From pad through secondary buffer, to any clock k	T_{SG}	XC4002A	8.7	6.7	[illegible]	ns
		XC4003A	8.8	6.8	[illegible]	ns
		XC4004A	8.9	6.9	[illegible]	ns
		XC4005A	9.0	7.0	[illegible]	ns

Horizontal Longline Switching Characteristic Guidelines

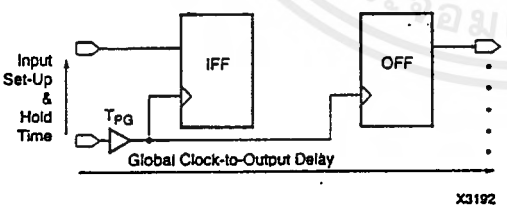
Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

	Speed Grade		-6	-5	-4	
Description	Symbol	Device	Max	Max	Max	Units
TBUF driving a Horizontal Longline (L.L.) I going High or Low to L.L. going High or Low, while T is Low, i.e. buffer is constantly active	T_{IO1}	XC4002A	8.2	6.0		ns
		XC4003A	8.8	6.2	4.4	ns
		XC4004A	9.4	6.6		ns
		XC4005A	10.0	7.0	5.5	ns
I going Low to L.L. going from resistive pull-up High to active Low, (TBUF configured as open drain)	T_{IO2}	XC4002A	8.7	6.5		ns
		XC4003A	9.3	6.7	5.0	ns
		XC4004A	9.9	7.1		ns
		XC4005A	10.5	7.5	6.0	ns
T going Low to L.L. going from resistive pull-up or floating High to active Low, (TUBF configured as open drain)	T_{ON}	XC4002A	10.1	8.4		ns
		XC4003A	10.7	9.0	7.2	ns
		XC4004A	11.4	9.5		ns
		XC4005A	12.0	10.0	8.0	ns
T going High to TBUF going inactive, not driving L.L.	T_{OFF}	All devices	3.0	2.0	1.8	ns
T going High to L.L. going from Low to High, pulled up by a single resistor	T_{PUS}	XC4002A	23.0	19.0		ns
		XC4003A	24.0	20.0	14.0	ns
		XC4004A	25.0	21.0		ns
		XC4005A	26.0	22.0	16.0	ns
T going High to L.L. going from Low to High, pulled up by two resistors	T_{PUF}	XC4002A	10.5	8.5		ns
		XC4003A	11.0	9.0	7.0	ns
		XC4004A	11.5	9.5		ns
		XC4005A	12.0	10.0	8.0	ns

Guaranteed Input and Output Parameters (Pin-to-Pin)

All values listed below are tested directly. and guaranteed over the operating conditions. The same parameters can also be derived indirectly from the IOB and Global Buffer specifications. The XACT delay calculator uses this indirect method. When there is a discrepancy between these two methods, the directly tested values listed below should be used, and the derived values should be ignored.

	Speed Grade		-6	-5	-4	
Description	Symbol	Device				Units
Global Clock to Output (fast)	T_{ICKOF}	XC4002A	14.9	12.2		ns
		XC4003A	15.1	12.5	11.6	ns
	(Max)	XC4004A	15.3	12.8		ns
		XC4005A	15.5	13.0	12.0	ns
Global Clock to Output (slew limited)	T_{ICKO}	XC4002A	19.9	15.2		ns
		XC4003A	20.1	15.5	14.6	ns
	(Max)	XC4004A	20.3	15.8		ns
		XC4005A	20.5	16.0	15.0	ns
Input Set-up Time, using IFF (no delay)	T_{PSUF}	XC4002A	2.6	2.3		ns
		XC4003A	2.4	2.0	1.6	ns
	(Min)	XC4004A	2.2	1.7		ns
		XC4005A	2.0	1.5	1.2	ns
Input Hold time, using IFF (no delay)	T_{PHF}	XC4002A	4.9	3.7		ns
		XC4003A	5.1	4.0	4.0	ns
	(Min)	XC4004A	5.3	4.3		ns
		XC4005A	5.5	4.5	4.5	ns
Input Set-up Time, using IFF (with delay)	T_{PSU}	XC4002A	21.8	18.8		ns
		XC4003A	21.5	18.5	12.0	ns
	(Min)	XC4004A	21.2	18.2		ns
		XC4005A	21.0	18.0	12.0	ns
Input Hold Time, using IFF (with delay)	T_{PH}	XC4002A	0	0		ns
		XC4003A	0	0	0	ns
	(Min)	XC4004A	0	0		ns
		XC4005A	0	0	0	ns



X3192

Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture). When testing fast outputs, only one output switches. When testing slew-rate limited outputs, half the number of outputs on one side of the device are switching. These parameter values are tested and guaranteed for worst-case conditions of supply voltage and temperature, and also with the most unfavorable clock polarity choice.

T_{PDLI} for -4 Speed Grade

Pad to I1, I2 via transparent latch, with delay	XC4003A	17.6 ns
	XC4005A	17.9 ns

PRELIMINARY

See page 2-76

T_{PICKD} for -4 Speed Grade

Input set-up time pad to clock (IK) with delay	XC4003A	15.6 ns
	XC4005A	15.9 ns

PRELIMINARY

X6091

IOB Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

		-6		-5		XC4003A XC4005A -4		
Description	Symbol	Min	Max	Min	Max	Min	Max	Units
INPUT								
Propagation Delays								
Pad to I1, I2	T_{PID}		4.0		3.0		2.8	ns
Pad to I1, I2, via transparent latch (no delay)	T_{PLI}		8.0		7.0		6.0	ns
Pad to I1, I2, via transparent latch (with delay)	T_{PDLI}		26.0		24.0			ns
Clock (IK) toI1, I2, (flip-flop)	T_{IKRI}		8.0		7.0		6.0	ns
Clock (IK) to I1, I2 (latch enable, active Low)	T_{IKLI}		8.0		7.0		6.0	ns
Set-up Time (Note 3)								
Pad to Clock (IK), no delay	T_{PICK}	7.0		6.0		4.0		ns
Pad to Clock (IK) with delay	T_{PICKD}	25.0		24 .0				ns
Hold Time (Note 3)								
Pad to Clock (IK), no delay	T_{IKPI}	1.0		1.0		1.0		ns
Pad to Clock (IK) with delay	T_{IKPID}	neg		neg		neg		ns
OUTPUT								
Propagation Delays								
Clock (OK) to Pad (fast)	T_{OKPOF}		7.5		7.0		6.5	ns
Output (O) to Pad (fast)	T_{OPF}		9.0		7.0		5.5	ns
3-state to Pad begin hi-Z (slew-rate independent)	T_{TSHZ}		9.0		7.0		6.5	ns
3-state to Pad active and valid (fast)	T_{TSONF}		13.0		10.0		9.5	ns
Additional Delay								
For medium fast outputs			2.0		1.5		1.0	ns
For medium slow outputs			4.0		3.0		2.0	ns
For slow outputs			6.0		4.5		3.0	ns
Set-up and Hold Times								
Output (O) to clock (OK) set-up time	T_{OOK}	8.0		6.0		5.5		ns
Output (O) to clock (OK) hold time	T_{OKO}	0.0		0.0		0		ns
Clock								
Clock High or Low time	T_{CH}/T_{CL}	5.0		4.0		4.0		ns
Global Set/Reset								
Delay from GSR net through Q to I1, I2	T_{RRI}		14.5		13.5		13.5	ns
Delay from GSR net to Pad	T_{RPO}		18.0		17.0		14.6	ns
GSR width*	T_{MRW}	21.0		18.0		18.0		ns

* Timing is based on the XC4005. For other devices see XACT timing calculator.
** See preceding page.

Notes: 1. Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture).

2. Voltage levels of unused (bonded and unbonded) pads must be valid logic levels. Each can be configured with the internal pull-up or pull-down resistor or alternatively configured as a driven output or be driven from an external source.

3. Input pad setup times and hold times are specified with respect to the internal clock (IK). To calculate system setup time, subtract clock delay (clock pad to IK) from the specified input pad setup time value, but do not subtract below zero. Negative hold time means that the delay in the input data is adequate for the external system hold time to be zero, provided the input clock uses the Global signal distribution from pad to IK.

CLB Switching Characteristic Guidelines

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

Speed Grade		-6		-5		XC4003A XC4005A -4		
Description	**Symbol**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Units**
Combinatorial Delays								
F/G inputs to X/Y outputs	T_{ILO}		6.0		4.5		4.0	ns
F/G inputs via H' to X/Y outputs	T_{IHO}		8.0		7.0		6.0	ns
C inputs via H' to X/Y outputs	T_{HHO}		7.0		5.0		4.5	ns
CLB Fast Carry Logic								
Operand inputs (F1,F2,G1,G4) to C_{OUT}	T_{OPCY}		7.0		5.5		5.0	ns
Add/Subtract input (F3) to C_{OUT}	T_{ASCY}		8.0		6.0		5.5	ns
Initialization inputs (F1,F3) to C_{OUT}	T_{INCY}		6.0		4.0		3.5	ns
C_{IN} through function generators to X/Y outputs	T_{SUM}		8.0		6.0		5.5	ns
C_{IN} to C_{OUT}, bypass function generators.	T_{BYP}		2.0		1.5		1.5	ns
Sequential Delays								
Clock K to outputs Q	T_{CKO}		5.0		3.0		3.0	ns
Set-up Time before Clock K								
F/G inputs	T_{ICK}	6.0		4.5		4.5		ns
F/G inputs via H'	T_{IHCK}	8.0		6.0		6.0		ns
C inputs via H1	T_{HHCK}	7.0		5.0		5.0		ns
C inputs via DIN	T_{DICK}	4.0		3.0		3.0		ns
C inputs via EC	T_{ECCK}	7.0		4.0		3.0		ns
C inputs via S/R, going Low (inactive)	T_{RCK}	6.0		4.5		4.0		ns
C_{IN} input via F'/G'		8.0		6.0		5.5		ns
C_{IN} input via F'/G' and H'		10.0		7.5		7.3		ns
Hold Time after Clock K								
F/G inputs	T_{CKI}	0		0		0		ns
F/G inputs via H'	T_{CKIH}	0		0		0		ns
C inputs via H1	T_{CKHH}	0		0		0		ns
C inputs via DIN	T_{CKDI}	0		0		0		ns
C inputs via EC	T_{CKEC}	0		0		0		ns
C inputs via S/R, going Low (inactive)	T_{CKR}	0		0		0		ns
Clock								
Clock High time	T_{CH}	5.0		4.0		4.0		ns
Clock Low time	T_{CL}	5.0		4.0		4.0		ns
Set/Reset Direct								
Width (High)	T_{RPW}	5.0		4.0		4.0		ns
Delay from C inputs via S/R, going High to Q	T_{RIO}		9.0		8.0		7.0	ns
Master Set/Reset*								
Width (High or Low)	T_{MRW}	21.0		18.0		18.0		ns
Delay from Global Set/Reset net to Q	T_{MRQ}		33.0		31.0		28.0	ns

* Timing is based on the XC4005. For other devices see XACT timing calculator.

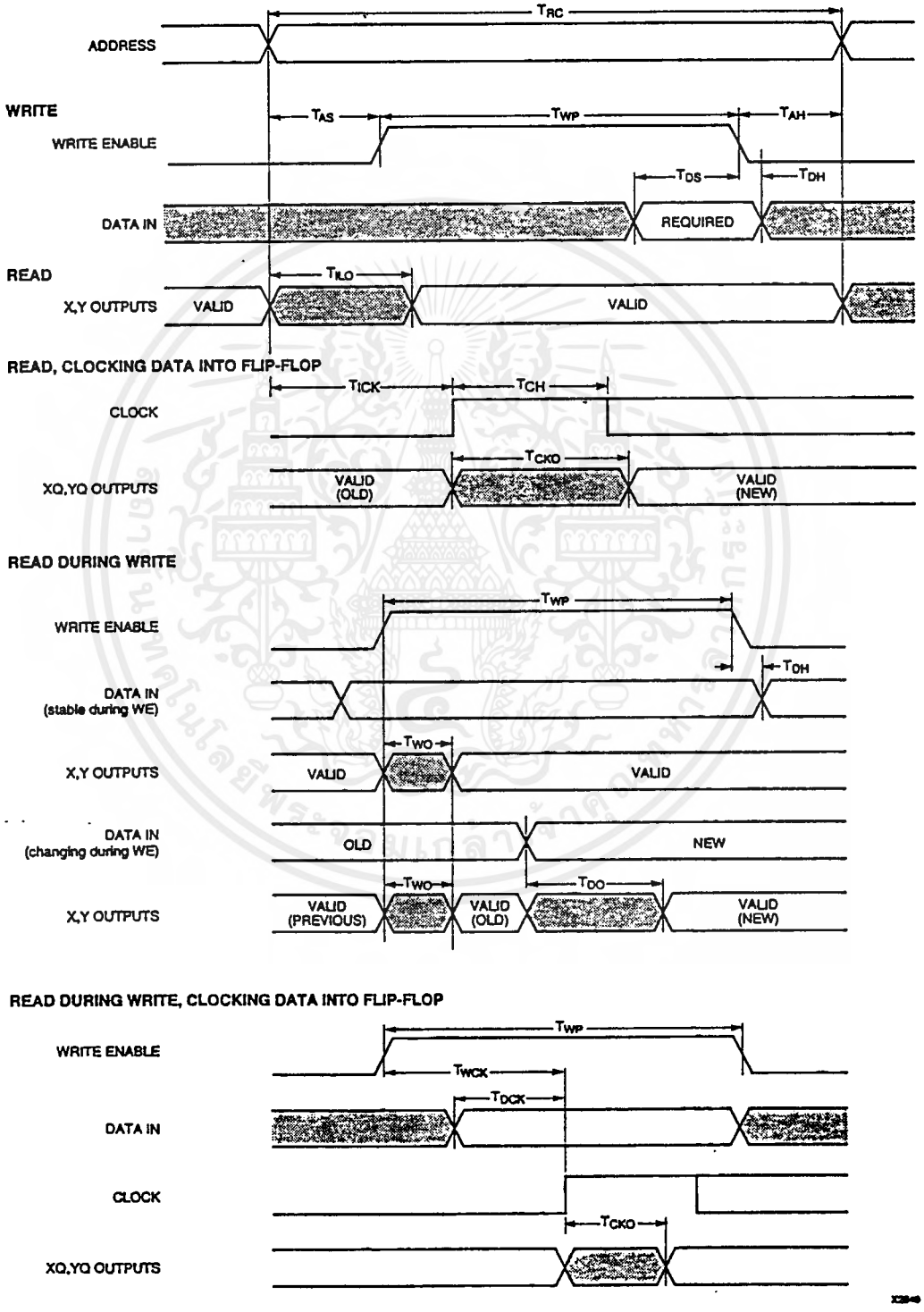
CLB Switching Characteristic Guidelines (continued)

Testing of the switching parameters is modeled after testing methods specified by MIL-M-38510/605. All devices are 100% functionally tested. Since many internal timing parameters cannot be measured directly, they are derived from benchmark timing patterns. The following guidelines reflect worst-case values over the recommended operating conditions. For more detailed, more precise, and more up-to-date timing information, use the values provided by the XACT timing calculator and used in the simulator.

CLB RAM OPTION	Speed Grade		-6		-5		XC4003A XC4005A -4		
Description	Symbol		Min	Max	Min	Max	Min	Max	Units
Write Operation									
Address write cycle time	16 x 2	T_{WC}	9.0		8.0		8.0		ns
	32 x 1	T_{WCT}	9.0		8.0		8.0		ns
Write Enable pulse width (High)	16 x 2	T_{WP}	5.0		4.0		4.0		ns
	32 x 1	T_{WPT}	5.0		4.0		4.0		ns
Address set-up time before beginning of WE	16 x 2	T_{AS}	2.0		2.0		2.0		ns
	32 x 1	T_{AST}	2.0		2.0		2.0		ns
Address hold time after end of WE	16 x 2	T_{AH}	2.0		2.0		2.0		ns
	32 x 1	T_{AHT}	2.0		2.0		2.0		ns
DIN set-up time before end of WE	16 x 2	T_{DS}	4.0		4.0		4.0		ns
	32 x 1	T_{DST}	5.0		5.0		5.0		ns
DIN hold time after end of WE	both	T_{DHT}	2.0		2.0		2.0		ns
Read Operation									
Address read cycle time	16 x 2	T_{RC}	7.0		5.5		5.0		ns
	32 x 1	T_{RCT}	10.0		7.5		7.0		ns
Data valid after address change (no Write Enable)	16 x 2	T_{ILO}		6.0		4.5		4.0	ns
	32 x 1	T_{IHO}		8.0		7.0		6.0	ns
Read Operation, Clocking Data into Flip-Flop									
Address setup time before clock K	16 x 2	T_{ICK}	6.0		4.5		4.5		ns
	32 x 1	T_{IHCK}	8.0		6.0		6.0		ns
Read During Write									
Data valid after WE going active (DIN stable before WE)	16 x 2	T_{WO}		12.0		10.0		9.0	ns
	32 x 1	T_{WOT}		15.0		12.0		11.0	ns
Data valid after DIN (DIN change during WE)	16 x 2	T_{DO}		11.0		9.0		8.5	ns
	32 x 1	T_{DOT}		14.0		11.0		11.0	ns
Read During Write, Clocking Data into Flip-Flop									
WE setup time before clock K	16 x 2	T_{WCK}	12.0		10.0		9.5		ns
	32 x 1	T_{WCKT}	15.0		12.0		11.5		ns
Data setup time before clock K	16 x 2	T_{DCK}	11.0		9.0		9.0		ns
	32 x 1	T_{DCKT}	14.0		11.0		11.0		ns

Note: Timing for the 16 x 1 RAM option is identical to 16 x 2 RAM timing

CLB RAM Timing Characteristics



XC4002A Pinouts

Pin Description	PC 84	PQ100	VQ100	PG120	Bound Scan
VCC	2	92	89	G3	–
I/O (A8)	3	93	90	G1	26
I/O (A9)	4	94	91	F1	29
–	–	95*	92*	E1*	–
–	–	96*	93*	F2*	–
I/O (A10)	5	97	94	F3	32
I/O (A11)	6	98	95	D1	35
–	–	–	–	E2*	–
I/O (A12)	7	99	96	C1	38
I/O (A13)	8	100	97	D2	41
–	–	–	–	E3*	–
–	–	–	–	B1*	–
I/O (A14)	9	1	98	C2	44
SGCK1 (A15, I/O)	10	2	99	D3	47
VCC	11	3	100	C3	–
GND	12	4	1	C4	–
PGCK1 (A16, I/O)	13	5	2	B2	50
I/O (A17)	14	6	3	B3	53
–	–	–	–	A1*	–
–	–	–	–	A2*	–
I/O (TDI)	15	7	4	C5	56
I/O (TCK)	16	8	5	B4	59
–	–	–	–	A3*	–
I/O (TMS)	17	9	6	B5	62
I/O	18	10	7	A4	65
–	–	–	–	C6*	–
–	–	11*	8*	A5*	–
I/O	19	12	9	B6	68
I/O	20	13	10	A6	71
GND	21	14	11	B7	–
VCC	22	15	12	C7	–
I/O	23	16	13	A7	74
I/O	24	17	14	A8	77
–	–	18*	15*	A9*	–
–	–	–	–	B8*	–
I/O	25	19	16	C8	80
I/O	26	20	17	A10	83
I/O	27	21	18	B9	86
I/O	–	22	19	A11	89
–	–	–	–	B10*	–

Pin Description	PC 84	PQ100	VQ100	PG120	Bound Scan
I/O	28	23	20	C9	92
SGCK2 (I/O)	29	24	21	A12	95
O (M1)	30	25	22	B11	98
GND	31	26	23	C10	–
I (M0)	32	27	24	C11	101†
VCC	33	28	25	D11	–
I (M2)	34	29	26	B12	102†
PGCK2 (I/O)	35	30	27	C12	103
I/O (HDC)	36	31	28	A13	106
–	–	–	–	B13*	–
–	–	–	–	E11*	–
I/O	–	32	29	D12	109
I/O (LDC)	37	33	30	C13	112
I/O	38	34	31	E12	115
I/O	39	35	32	D13	118
–	–	36*	33*	F11*	–
–	–	37*	34*	E13*	–
I/O	40	38	35	F12	121
I/O (ERR, INIT)	41	39	36	F13	124
VCC	42	40	37	G12	–
GND	43	41	38	G11	–
I/O	44	42	39	G13	127
I/O	45	43	40	H13	130
–	–	44*	41*	J13*	–
–	–	45*	42*	H12*	–
I/O	46	46	43	H11	133
I/O	47	47	44	K13	136
I/O	48	48	45	J12	139
I/O	49	49	46	L13	142
–	–	–	–	K12*	–
–	–	–	–	J11*	–
I/O	50	50	47	M13	145
SGCK3 (I/O)	51	51	48	L12	148
GND	52	52	49	K11	–
DONE	53	53	50	L11	–
VCC	54	54	51	L10	–
PROG	55	55	52	M12	–
I/O (D7)	56	56	53	M11	151
PGCK3 (I/O)	57	57	54	N13	154
–	–	–	–	N12*	–

Pin Description	PC 84	PQ100	VQ100	PG120	Bound Scan
–	–	–	–	L9	–
I/O (D6)	58	58	55	M10	157
I/O	–	59	56	N11	160
I/O (D5)	59	60	57	M9	163
I/O (CSO)	60	61	58	N10	166
–	–	62*	59*	L8*	–
–	–	63*	60*	N9*	–
I/O (D4)	61	64	61	M8	169
I/O	62	65	62	N8	172
VCC	63	66	63	M7	–
GND	64	67	64	L7	–
I/O (D3)	65	68	65	N7	175
I/O (RS)	66	69	66	N6	178
–	–	70*	67*	N5*	–
–	–	–	–	M6*	–
I/O (D2)	67	71	68	L6	181
I/O	68	72	69	N4	184
I/O (D1)	69	73	70	M5	187
I/O (RCLK-BUSY/RDY)	70	74	71	N3	190
–	–	–	–	M4*	–
–	–	–	–	L5*	–
I/O (D0, DIN)	71	75	72	N2	193
SGCK4 (DOUT, I/O)	72	76	73	M3	196
CCLK	73	77	74	L4	–
VCC	74	78	75	L3	–
O (TDO)	75	79	76	M2	–
GND	76	80	77	K3	–
I/O (A0, WS)	77	81	78	L2	2
PGCK4 (I/O,A1)	78	82	79	N1	5
–	–	–	–	M1*	–
–	–	–	–	J3*	–
I/O (CS1, A2)	79	83	80	K2	8
I/O (A3)	80	84	81	L1	11
I/O (A4)	81	85	82	J2	14
I/O (A5)	82	86	83	K1	17
–	–	87*	84*	H3*	–
–	–	88*	85*	J1*	–
I/O (A6)	83	89	86	H2	20
I/O (A7)	84	90	87	H1	23
GND	1	91	88	G2	–

* Indicates unconnected package pins.
† Contributes only one bit (J) to the boundary scan register.
Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 199 = BSCANT.UPD

XC4003A Pinouts

Pin Description	PC84	VQ100	PQ100	PG120	Bound Scan
VCC	2	89	92	G3	–
I/O (A8)	3	90	93	G1	32
I/O (A9)	4	91	94	F1	35
I/O	–	92	95	E1	38
I/O	–	93	96	F2	41
I/O (A10)	5	94	97	F3	44
I/O (A11)	6	95	98	D1	47
–	–	–	–	E2*	–
I/O (A12)	7	96	99	C1	50
I/O (A13)	8	97	100	D2	53
–	–	–	–	E3*	–
–	–	–	–	B1*	–
I/O (A14)	9	98	1	C2	56
SGCK1 (A15,I/O)	10	99	2	D3	59
VCC	11	100	3	C3	–
GND	12	1	4	C4	–
PGCK1 (A16, I/O)	13	2	5	B2	62
I/O (A17)	14	3	6	B3	65
–	–	–	–	A1*	–
–	–	–	–	A2*	–
I/O (TDI)	15	4	7	C5	68
I/O (TCK)	16	5	8	B4	71
–	–	–	–	A3*	–
I/O (TMS)	17	6	9	B5	74
I/O	18	7	10	A4	77
I/O	–	–	–	C6	80
I/O	–	8	11	A5	83
I/O	19	9	12	B6	86
I/O	20	10	13	A6	89
GND	21	11	14	B7	–
VCC	22	12	15	C7	–
I/O	23	13	16	A7	92
I/O	24	14	17	A8	95
I/O	–	15	18	A9	98
I/O	–	–	–	B8	101
I/O	25	16	19	C8	104
I/O	26	17	20	A10	107
I/O	27	18	21	B9	110
I/O	–	19	22	A11	113
–	–	–	–	B10*	–
I/O	28	20	23	C9	116
SGCK2 (I/O)	29	21	24	A12	119
O (M1)	30	22	25	B11	122
GND	31	23	26	C10	–
I (M0)	32	24	27	C11	125†
VCC	33	25	28	D11	–
I (M2)	34	26	29	B12	126†
PGCK2 (I/O)	35	27	30	C12	127
I/O (HDC)	36	28	31	A13	130
–	–	–	–	B13*	–
–	–	–	–	E11*	–
I/O	–	29	32	D12	133
I/O (LDC)	37	30	33	C13	136
I/O	38	31	34	E12	139
I/O	39	32	35	D13	142
I/O	–	33	36	F11	145
I/O	–	34	37	E13	148
I/O	40	35	38	F12	151
I/O (ERR, INIT)	41	36	39	F13	154
VCC	42	37	40	G12	–

Pin Description	PC84	VQ100	PQ100	PG120	Bound Scan
GND	43	38	41	G11	–
I/O	44	39	42	G13	157
I/O	45	40	43	H13	160
I/O	–	41	44	J13	163
I/O	–	42	45	H12	166
I/O	46	43	46	H11	169
I/O	47	44	47	K13	172
I/O	48	45	48	J12	175
I/O	49	46	49	L13	178
–	–	–	–	K12*	–
–	–	–	–	J11*	–
I/O	50	47	50	M13	181
SGCK3 (I/O)	51	48	51	L12	184
GND	52	49	52	K11	–
DONE	53	50	53	L11	–
VCC	54	51	54	L10	–
PROG	55	52	55	M12	–
I/O (D7)	56	53	56	M11	187
PGCK3 (I/O)	57	54	57	N13	190
–	–	–	–	N12*	–
–	–	–	–	L9*	–
I/O (D6)	58	55	58	M10	193
I/O	–	56	59	N11	196
I/O (D5)	59	57	60	M9	199
I/O (CS0)	60	58	61	N10	202
I/O	–	59	62	L8	205
I/O	–	60	63	N9	208
I/O (D4)	61	61	64	M8	211
I/O	62	62	65	N8	214
VCC	63	63	66	M7	–
GND	64	64	67	L7	–
I/O (D3)	65	65	68	N7	217
I/O (RS)	66	66	69	N6	220
I/O	–	67	70	N5	223
I/O	–	–	–	M6	226
I/O (D2)	67	68	71	L6	229
I/O	68	69	72	N4	232
I/O (D1)	69	70	73	M5	235
I/O (RCLK-BUSY/RDY)	70	71	74	N3	238
–	–	–	–	M4*	–
–	–	–	–	L5*	–
I/O (D0, DIN)	71	72	75	N2	241
SGCK4 (DOUT, I/O)	72	73	76	M3	244
CCLK	73	74	77	L4	–
VCC	74	75	78	L3	–
O (TDO)	75	76	79	M2	–
GND	76	77	80	K3	–
I/O (A0, WS)	77	78	81	L2	2
PGCK4 (A1, I/O)	78	79	82	N1	5
–	–	–	–	M1*	–
–	–	–	–	J3*	–
I/O (CS1, A2)	79	80	83	K2	8
I/O (A3)	80	81	84	L1	11
I/O (A4)	81	82	85	J2	14
I/O (A5)	82	83	86	K1	17
I/O	–	84	87	H3	20
I/O	–	85	88	J1	23
I/O (A6)	83	86	89	H2	26
I/O (A7)	84	87	90	H1	29
GND	1	88	91	G2	–

* Indicates unconnected package pins.
† Contributes only one bit (.I) to the boundary scan register.
Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 247 = BSCANT.UPD

XC4004A Pinouts

Pin Description	PC84	TQ144	PQ160	PG120	Bound Scan
VCC	2	128	142	G3	–
I/O (A8)	3	129	143	G1	38
I/O (A9)	4	130	144	F1	41
I/O	–	131	145	E1	44
I/O	–	132	146	F2	47
I/O (A10)	5	133	147	F3	50
I/O (A11)	6	134	148	D1	53
–	–	135*	149*	–	–
–	–	136*	150*	–	–
GND	–	137	151	E2	–
–	–	–	152*	–	–
–	–	–	153*	–	–
I/O (A12)	7	138	154	C1	56
I/O (A13)	8	139	155	D2	59
I/O	–	140	156	E3	62
I/O	–	141	157	B1	65
I/O (A14)	9	142	158	C2	68
SGCK1 (A15, I/O)	10	143	159	D3	71
VCC	11	144	160	C3	–
GND	12	1	1	C4	–
PGCK1 (A16, I/O)	13	2	2	B2	74
I/O (A17)	14	3	3	B3	77
I/O	–	4	4	A1	80
I/O	–	5	5	A2	83
I/O (TDI)	15	6	6	C5	86
I/O (TCK)	16	7	7	B4	89
–	–	–	8*	–	–
–	–	–	9*	–	–
GND	–	8	10	A3	–
–	–	9*	11*	–	–
–	–	10*	12*	–	–
I/O (TMS)	17	11	13	B5	92
I/O	18	12	14	A4	95
I/O	–	13	15	C6	98
I/O	–	14	16	A5	101
I/O	19	15	17	B6	104
I/O	20	16	18	A6	107
GND	21	17	19	B7	–
VCC	22	18	20	C7	–
I/O	23	19	21	A7	110
I/O	24	20	22	A8	113
I/O	–	21	23	A9	116
I/O	–	22	24	B8	119
I/O	25	23	25	C8	122
I/O	26	24	26	A10	125
–	–	25*	27*	–	–
–	–	26*	28*	–	–
GND	–	27	29	–	–
–	–	–	30*	–	–
–	–	–	31*	–	–
I/O	27	28	32	B9	128
I/O	–	29	33	A11	131
I/O	–	30	34	B10	134
I/O	–	31	35	–	137

Pin Description	PC84	TQ144	PQ160	PG120	Bound Scan
I/O	28	32	36	C9	140
SGCK2 (I/O)	29	33	37	A12	143
O (M1)	30	34	38	B11	146
GND	31	35	39	C10	–
I (M0)	32	36	40	C11	149†
VCC	33	37	41	D11	–
I (M2)	34	38	42	B12	150†
PGCK2 (I/O)	35	39	43	C12	151
I/O (HDC)	38	40	44	A13	154
I/O	–	41	45	B13	157
I/O	–	42	46	E11	160
I/O	–	43	47	D12	163
I/O (LDC)	37	44	48	C13	166
–	–	–	49*	–	–
–	–	–	50*	–	–
GND	–	45	51	–	–
–	–	46*	52*	–	–
–	–	47*	53*	–	–
I/O	38	48	54	E12	169
I/O	39	49	55	D13	172
I/O	–	50	56	F11	175
I/O	–	51	57	E13	178
I/O	40	52	58	F12	181
I/O (ERR, INIT)	41	53	59	F13	184
VCC	42	54	60	G12	–
GND	43	55	61	G11	–
I/O	44	56	62	G13	187
I/O	45	57	63	H13	190
I/O	–	58	64	J13	193
I/O	–	59	65	H12	196
I/O	46	60	66	H11	199
I/O	47	61	67	K13	202
–	–	62*	68*	–	–
–	–	63*	69*	–	–
GND	–	64	70	–	–
–	–	–	71*	–	–
–	–	–	72*	–	–
I/O	48	65	73	J12	205
I/O	49	66	74	L13	201
I/O	–	67	75	K12	211
I/O	–	68	76	J11	214
I/O	50	69	77	M13	217
SGCK3 (I/O)	51	70	78	L12	220
GND	52	71	79	K11	–
DONE	53	72	80	L11	–
VCC	54	73	81	L10	–
PROG	55	74	82	M12	–
I/O (D7)	56	75	83	M11	223
PGCK3 (I/O)	57	76	84	N13	226
I/O	-	77	85	N12	229
I/O	-	78	86	L9	232
I/O (D6)	58	79	87	M10	235
I/O	-	80	88	N11	238
-	-	-	89*	-	–

Pin Description	PC84	TQ144	PQ160	PG120	Bound Scan
–	–	–	90*	–	–
GND	–	81	91	–	–
–	–	82*	92*	–	–
–	–	83*	93*	–	–
I/O (D5)	59	84	94	M9	241
I/O (CSO)	60	85	95	N10	244
I/O	–	86	96	L8	247
I/O	–	87	97	N9	250
I/O (D4)	61	88	98	M8	253
I/O	62	89	99	N8	256
VCC	63	90	100	M7	–
GND	64	91	101	L7	–
I/O (D3)	65	92	102	N7	259
I/O (RS)	66	93	103	N6	262
I/O	–	94	104	N5	265
I/O	–	95	105	M6	268
I/O (D2)	67	96	106	L6	271
I/O	68	97	107	N4	274
–	–	98*	108*	–	–
–	–	99*	109*	–	–
GND	–	100	110	–	–
–	–	–	111*	–	–
–	–	–	112*	–	–
I/O (D1)	69	101	113	M5	277
I/O (RCLK-BUSY/RDY)	70	102	114	N3	280
I/O	–	103	115	M4	283
I/O	–	104	116	L5	286
I/O (D0, DIN)	71	105	117	N2	289
SGCK4 (DOUT, I/O)	72	106	118	M3	292
CCLK	73	107	119	L4	–
VCC	74	108	120	L3	–
O (TDO)	75	109	121	M2	–
GND	76	110	122	K3	–
I/O (A0, WS)	77	111	123	L2	2
PGCK4 (I/O,A1)	78	112	124	N1	5
I/O	–	113	125	M1	8
I/O	–	114	126	J3	11
I/O (CS1, A2)	79	115	127	K2	14
I/O (A3)	80	116	128	L1	17
–	–	117*	129*	–	–
–	–	–	130*	–	–
GND	–	118	131	–	–
–	–	119*	132*	–	–
–	–	120*	133*	–	–
I/O (A4)	81	121	134	J2	20
I/O (A5)	82	122	135	K1	23
–	–	–	136*	–	–
I/O	–	123	137	H3	26
I/O	–	124	138	J1	29
I/O (A6)	83	125	139	H2	32
I/O (A7)	84	126	140	H1	35
GND	1	127	141	G2	–

* Indicates unconnected package pins.
† Contributes only one bit (J) to the boundary scan register.
Boundary Scan Bit 0 = TDO.T
Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 295 = BSCANT.UPD

XC4005A Pinouts

Pin Description	PC84	TQ144	PQ160	PQ208	PG156	Bound Scan
VCC	2	128	142	183	H3	–
I/O (A8)	3	129	143	184	H1	44
I/O (A9)	4	130	144	185	G1	47
I/O	–	131	145	186	G2	50
I/O	–	132	146	187	G3	53
–	–	–	–	188*	–	–
–	–	–	–	189*	–	–
I/O (A10)	5	133	147	190	F1	56
I/O (A11)	6	134	148	191	F2	59
I/O	–	135	149	192	E1	62
I/O	–	136	150	193	E2	65
GND	–	137	151	194	F3	–
–	–	–	–	195*	–	–
–	–	–	–	196*	–	–
–	–	–	152*	197*	D1*	–
–	–	–	153*	198*	D2*	–
I/O (A12)	7	138	154	199	E3	68
I/O (A13)	8	139	155	200	C1	71
–	–	–	–	–	–	–
I/O	–	140	156	201	C2	74
I/O	–	141	157	202	D3	77
I/O (A14)	9	142	158	203	B1	80
SGCK1 (A15, I/O)	10	143	159	204	B2	83
VCC	11	144	160	205	C3	–
–	–	–	–	206*	–	–
–	–	–	–	207*	–	–
–	–	–	–	208*	–	–
–	–	–	–	1*	–	–
GND	12	1	1	2	C4	–
–	–	–	–	3*	–	–
PGCK1 (A16, I/O)	13	2	2	4	B3	86
I/O (A17)	14	3	3	5	A1	89
I/O	–	4	4	6	A2	92
I/O	–	5	5	7	C5	95
–	–	–	–	–	–	–
I/O (TDI)	15	6	6	8	B4	98
I/O (TCK)	16	7	7	9	A3	101
–	–	–	8*	10*	A4*	–
–	–	–	9*	11*	–	–
–	–	–	–	12*	–	–
–	–	–	–	13*	–	–
GND	–	8	10	14	C6	–
I/O	–	9	11	15	B5	104
I/O	–	10	12	16	B6	107
I/O (TMS)	17	11	13	17	A5	110
I/O	18	12	14	18	C7	113
–	–	–	–	19*	–	–
–	–	–	–	20*	–	–
I/O	–	13	15	21	B7	116
I/O	–	14	16	22	A6	119
I/O	19	15	17	23	A7	122
I/O	20	16	18	24	A8	125
GND	21	17	19	25	C8	–
VCC	22	18	20	26	B8	–
I/O	23	19	21	27	C9	128
I/O	24	20	22	28	B9	131
I/O	–	21	23	29	A9	134
I/O	–	22	24	30	B10	137
–	–	–	–	31*	–	–
–	–	–	–	32*	–	–
I/O	25	23	25	33	C10	140
I/O	26	24	26	34	A10	143
I/O	–	25	27	35	A11	146
I/O	–	26	28	36	B11	149
GND	–	27	29	37	C11	–
–	–	–	–	38*	–	–
–	–	–	–	39*	–	–
–	–	–	30*	40*	A12*	–
–	–	–	31*	41*	–	–
I/O	27	28	32	42	B12	152
I/O	–	29	33	43	A13	155
I/O	–	30	34	44	A14	158

Pin Description	PC84	TQ144	PQ160	PQ208	PG156	Bound Scan
I/O	–	31	35	45	C12	161
–	–	–	–	–	–	–
I/O	28	32	36	46	B13	164
SGCK2 (I/O)	29	33	37	47	B14	167
O (M1)	30	34	38	48	A15	170
GND	31	35	39	49	C13	–
I (M0)	32	36	40	50	A16	173†
–	–	–	–	51*	–	–
–	–	–	–	52*	–	–
–	–	–	–	53*	–	–
–	–	–	–	54*	–	–
VCC	33	37	41	55	C14	–
I (M2)	34	38	42	56	B15	174†
PGCK2 (I/O)	35	39	43	57	B16	175
I/O (HDC)	36	40	44	58	D14	178
I/O	–	41	45	59	C15	181
–	–	–	–	–	–	–
I/O	–	42	46	60	D15	184
I/O	–	43	47	61	E14	187
I/O (LDC)	37	44	48	62	C16	190
–	–	–	49*	63*	E15*	–
–	–	–	50*	64*	D16*	–
–	–	–	–	65*	–	–
–	–	–	–	66*	–	–
GND	–	45	51	67	F14	–
I/O	–	46	52	68	F15	193
I/O	–	47	53	69	E16	196
I/O	38	48	54	70	F16	199
I/O	39	49	55	71	G14	202
–	–	–	–	72*	–	–
–	–	–	–	73*	–	–
I/O	–	50	56	74	G15	205
I/O	–	51	57	75	G16	208
I/O	40	52	58	76	H16	211
I/O (ERR, INIT)	41	53	59	77	H15	214
VCC	42	54	60	78	H14	–
GND	43	55	61	79	J14	–
I/O	44	56	62	80	J15	217
I/O	45	57	63	81	J16	220
I/O	–	58	64	82	K16	223
I/O	–	59	65	83	K15	226
–	–	–	–	84*	–	–
–	–	–	–	85*	–	–
I/O	46	60	66	86	K14	229
I/O	47	61	67	87	L16	232
I/O	–	62	68	88	M16	235
I/O	–	63	69	89	L15	238
GND	–	64	70	90	L14	–
–	–	–	–	91*	–	–
–	–	–	–	92*	–	–
–	–	–	71*	93*	N16*	–
–	–	–	72*	94*	M15*	–
I/O	48	65	73	95	P16	241
I/O	49	66	74	96	M14	244
I/O	–	67	75	97	N15	247
I/O	–	68	76	98	P15	250
I/O	50	69	77	99	N14	253
SGCK3 (I/O)	51	70	78	100	R16	256
GND	52	71	79	101	P14	–
–	–	–	–	102*	–	–
DONE	53	72	80	103	R15	–
–	–	–	–	104*	–	–
–	–	–	–	105*	–	–
VCC	54	73	81	106	P13	–
–	–	–	–	107*	–	–
PROG	55	74	82	108	R14	–
I/O (D7)	56	75	83	109	T16	259
PGCK3 (I/O)	57	76	84	110	T15	262
I/O	–	77	85	111	R13	265
–	–	–	–	–	–	–
I/O	–	78	86	112	P12	268
I/O(D6)	58	79	87	113	T14	271

* Indicates unconnected package pins.
† Contributes only one bit (.i) to the boundary scan register.

XC4005A Pinouts (continued)

Pin Descriptions	PC84	TQ144	PQ160	PQ208	PG156	Bound Scan
I/O	–	80	88	114	T13	274
–	–	–	89*	115*	R12*	–
–	–	–	90*	116*	T12*	–
–	–	–	–	117*	–	–
–	–	–	–	118*	–	–
GND	–	81	91	119	P11	–
I/O	–	82	92	120	R11	277
I/O	–	83	93	121	T11	280
I/O (D5)	59	84	94	122	T10	283
I/O ($\overline{CS0}$)	60	85	95	123	P10	286
–	–	–	–	124*	–	–
–	–	–	–	125*	–	–
I/O	–	86	96	126	R10	289
I/O	–	87	97	127	T9	292
I/O (D4)	61	88	98	128	R9	295
I/O	62	89	99	129	P9	298
VCC	63	90	100	130	R8	–
GND	64	91	101	131	P8	–
I/O (D3)	65	92	102	132	T8	301
I/O ($\overline{RS}$)	66	93	103	133	T7	304
I/O	–	94	104	134	T6	307
I/O	–	95	105	135	R7	310
–	–	–	–	136*	–	–
–	–	–	–	137*	–	–
I/O (D2)	67	96	106	138	P7	313
I/O	68	97	107	139	T5	316
I/O	–	98	108	140	R6	319
I/O	–	99	109	141	T4	322
GND	–	100	110	142	P6	–
–	–	–	–	143*	–	–
–	–	–	–	144*	–	–
–	–	–	111*	145*	R5*	–
–	–	–	112*	146*	–	–
I/O (D1)	69	101	113	147	T3	325
I/O (RCLK-BUSY/RDY)	70	102	114	148	P5	328
I/O	–	103	115	149	R4	331
–	–	–	–	–	–	–
I/O	–	104	116	150	R3	334
I/O (D0, DIN)	71	105	117	151	P4	337
SGCK4 (DOUT, I/O)	72	106	118	152	T2	340
CCLK	73	107	119	153	R2	–
VCC	74	108	120	154	P3	–
–	–	–	–	155*	–	–
–	–	–	–	156*	–	–
–	–	–	–	157*	–	–
–	–	–	–	158*	–	–
O (TDO)	75	109	121	159	T1	–
GND	76	110	122	160	N3	–
I/O (A0,$\overline{WS}$)	77	111	123	161	R1	2
PGCK4 (A1,I/O)	78	112	124	162	P2	5
I/O	–	113	125	163	N2	8
–	–	–	–	–	–	–
I/O	–	114	126	164	M3	11
I/O (CS1,A2)	79	115	127	165	P1	14
I/O (A3)	80	116	128	166	N1	17
–	–	117*	129*	167*	M2*	–
–	–	–	130*	168*	M1*	–
–	–	–	–	169*	–	–
–	–	–	–	170*	–	–
GND	–	118	131	171	L3	–
I/O	–	119	132	172	L2	20
I/O	–	120	133	173	L1	23
I/O (A4)	81	121	134	174	K3	26
I/O (A5)	82	122	135	175	K2	29
–	–	–	–	176*	–	–
–	–	–	136*	177*	–	–
I/O	–	123	137	178	K1	32
I/O	–	124	138	179	J1	35
I/O (A6)	83	125	139	180	J2	38
I/O (A7)	84	126	140	181	J3	41
GND	1	127	141	182	H2	–

* Indicates unconnected package pins.
Boundary Scan Bit 0 = TDO.T

Boundary Scan Bit 1 = TDO.O
Boundary Scan Bit 343 = BSCANT.UPD

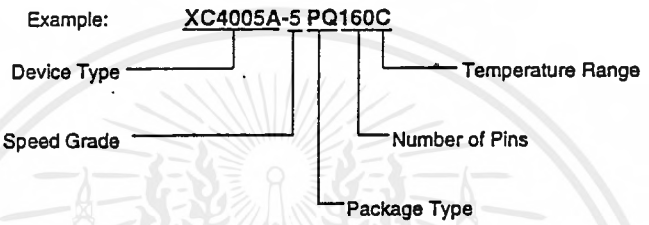
For a detailed description of the device architecture, see pages 2-9 through 2-31.

For a detailed description of the configuration modes and their timing, see pages 2-32 through 2-55.

For detailed lists of package pinouts, see pages 2-57 through 2-81 through 2-85.

For package physical dimensions and thermal data, see Section 4.

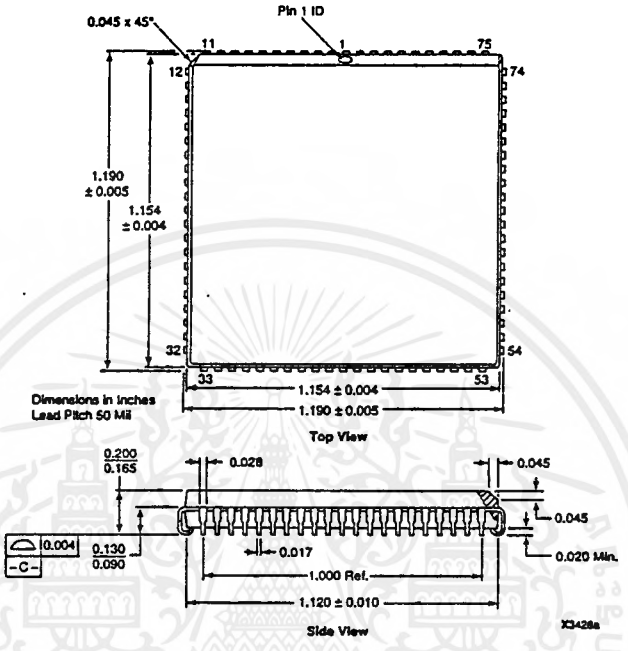
Ordering Information



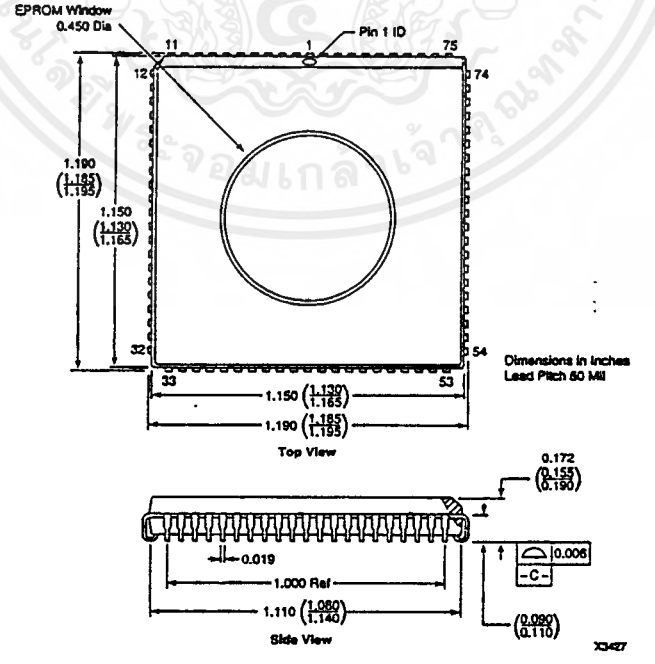
Component Availability

PINS		84	100			120	144	156	160	164	191	196	208		223	225	240		299
TYPE		PLAST. PLCC	PLAST. PQFP	PLAST. VQFP	TOP BRAZED CQFP	CERAM. PGA	PLAST. TQFP	CERAM PGA	PLAST. PQFP	TOP BRAZED CQFP	CERAM. PGA	TOP BRAZED CQFP	PLAST. PQFP	METAL PQFP	CERAM. PGA	PLAST. BGA	PLAST. PQFP	METAL PQFP	METAL PQFP
CODE		PC84	PQ100	VQ100	CB100	PG120	TQ144	PG156	PQ160	CB164	PG191	CB196	PQ208	MQ208	PG223	BG225	PQ240	MQ240	PG299
XC4002A	-6	C I	C I	C I		C I													
	-5	C	C	C		C													
	-4																		
XC4003A	-10				M B	M B													
	-6	C I	C I	C I	M B	C I M B													
	-5	C	C	C		C													
	-4	C	C	C		C													
XC4004A	-6	C I				C I	C I		C I										
	-5	C				C	C		C										
	-4																		
XC4005A	-6	C I					C I	C I	C I				C I						
	-5	C I					C I	C I	C I				C I						
	-4	C					C	C	C				C						

C = Commercial = 0° to +85° C I = Industrial = -40° to +100° C M = Mil Temp = -55° to +125° C
B = MIL-STD-883C Class B Parentheses indicate future product plans



84-Pin Plastic PLCC (PC84)



84-Pin Windowed CLCC (WC84)

Thermal Resistance, Junction-to-Ambient (θ_{JA} in °C/W)

Pins	44	48	64	68		84		100				120	132	
Type	PLCC	PDIP	VQFP	PLCC	PGA	PLCC	PGA	PQFP	TQFP	VQFP	COFP	PGA	PPGA	PGA
XC2064/L	46.5	42.2		44.9	39.8									
XC2018/L	47.2			43.7	37.4	38.4	37.4				46.0			
XC3020A/L				42.4		37.1	36.1	72.0						
XC3030A/L	44.6			41.1		35.6	34.6	68.0			45.4			
XC3042A/L						35.4	34.4	65.0					34.5	33.5
XC3064A/L						32.3							36.5	33.7
XC3090A/L						32.4								
XC3195A						29.4								
XC4002A						32.9		69.0						
XC4003						32.1		65.0						
XC4003A						32.7		65.0						
XC4003H														
XC4004A						30.3								
XC4005						31.4								
XC4005A						32.0								
XC4005H														
XC4006						30.0								
XC4008						29.1								
XC4010/D						28.7								
XC4013/D														
XC7236	48.5													
XC7354	45.6			43.1										
XC7372				40.5		35.0	34.0							
XC73108						34.2		66.0						
XC73144														
XC7236A	47.3													
XC7272A				40.5		34.9	33.9							

Measured between 22° and 25° C. Values indicated for zero air flow. Additional values may be approximated for plastic packages. X6053
For 250 linear feet per minute (LFM) or 1.3 m/s, multiply value by 0.75. For 500 LFM or 2.5 m/s, multiply value by 0.67.
For 750 LFM or 3.8 m/s, multiply value by 0.63.

Thermal Resistance, Junction-to-Case (θ_{JC} in °C/W)

Pins	44	48	64	68		84		100				120	132	
Type	PLCC	PDIP	VQFP	PLCC	PGA	PLCC	PGA	PQFP	TQFP	VQFP	COFP	PGA	PPGA	PGA
XC2064/L	17.5	11.6		13.0	9.1									
XC2018/L	15.2			11.3	8.1	9.1	8.1							
XC3020A/L				9.9		8.2	7.2	12.0			8.6			
XC3030A/L	11.5			8.5		7.1	6.1	10.0						
XC3042A/L						6.2	5.2	8.0			6.8		6.1	2.9
XC3064A/L						4.8							5.2	2.5
XC3090A/L						4.0								
XC3195A						3.0								
XC4002A						5.2		8.0						
XC4003						4.7		4.0						
XC4003A						5.1		6.0						
XC4003H														
XC4004A						3.5								
XC4005						4.2								
XC4005A						4.6								
XC4005H														
XC4006						3.3								
XC4008						2.8								
XC4010/D						2.6								
XC4013/D														
XC73318/36	18.4													
XC7354	12.9			9.6										
XC7372				8.0		6.7	5.7							
XC73108						6.1		5.0						
XC73144														
XC7236A	15.5													
XC7272A				8.0		6.6	5.6							

Measured at 25° C with FC40 conditions. X6054

Board Conditions

Through-hole packages were socketed. Surface-mount packages were mounted on a single-layer 6.0 inch by 4.5 inch by 0.0625 inch FR4 board with less than 10% copper traces. Indirect electrical method used.

Thermal Resistance, Junction-to-Ambient (θ_{JA} in °C/W)

Pins	144	156	160	164	175		176	191	196	208		223	240	
Type	TQFP	PGA	PQFP	CQFP	PPGA	PGA	TQFP	PQFP	TQFP	PQFP	MQFP	PGA	PQFP	MQFP
XC2064A														
XC2018A														
XC3020A/L														
XC3030A/L														
XC3042A/L	41.6													
XC3064A/L	39.6		34.5											
XC3090A/L	39.7		34.6		28.5	28.5				33.6				
XC3195A			32.1		27.8	27.8				31.1				
XC4005A														
XC4003														
XC4003A														
XC4003H								20E		32.0				
XC4004A	38.0		32.9											
XC4005	36.9	27.2	33.8							32.8				
XC4005A	39.2		34.4							33.1				
XC4005H												20E	28.0	16.0
XC4006		LP 40.0	32.6							31.6				
XC4008			31.7							30.7	16.0			
XC4010D			31.3							30.3	16.0			
XC4013D			29.3							28.7	17.7	20E	28.0	16.0
XC7236														
XC7254														
XC7272														
XC73108			35.8											
XC73144			34.4											
XC7236A														
XC7272A														

Measured between 22°C and 25°C. E=Estimated value. Values indicated for zero air flow. Additional values may be approximated for plastic packages. X6050
For 250 linear feet per minute (LFPM), multiply value by 0.75. For 500 LFPM, multiply value by 0.67. For 750 LFPM, multiply value by 0.63.

Thermal Resistance, Junction-to-Case (θ_{JC} in °C/W)

Pins	144	156	160	164	175		176	191	196	208		223	240	
Type	TQFP	PGA	PQFP	CQFP	PPGA	PGA	TQFP	PQFP	TQFP	PQFP	MQFP	PGA	PQFP	MQFP
XC2064A														
XC2018A														
XC3020A/L														
XC3030A/L														
XC3042A/L	8.1													
XC3064A/L	6.9		5.6											
XC3090A/L	6.9		5.6		4.9	4.9				5.6				
XC3195A			5.7		4.1	4.1				3.6				
XC4005A														
XC4003														
XC4003A														
XC4003H								1.9		4.2				
XC4004A	4.1		4.3											
XC4005	4.6	3.9	5.0							4.9				
XC4005A	4.7		5.2							5.1				
XC4005H												1.8	6.0E	6.0E
XC4006			4.0							4.0				
XC4008			3.5							3.4	1.3			
XC4010D			3.3							3.2	1.3			
XC4013D			2.3							2.4	1.2	1.8	6.0E	6.0E
XC7236														
XC7254														
XC7272														
XC73108			6.9											
XC73144			6.5											
XC7236A														
XC7272A														

Measured at 25° C with FC40 conditions. X6052

Board Conditions

Through-hole packages were socketed. Surface-mount packages were mounted on a single-layer 6.0 inch by 4.5 inch by 0.0625 inch FR4 board with less than 10% copper traces. Indirect electrical method used.

XILINX®

Additional XC4000 Data

XAPP 045.000

Application Note By PHILIP FREIDIN

Summary

This Application Note contains additional information that may be of use when designing with the XC4000 families of devices. This information supplements the product descriptions and specifications, and is provided for guidance only.

Xilinx Family

XC4000/XC4000A/XC4000H

Introduction

This application note describes the electrical characteristics of the output drivers, their static output characteristics or I/V curves, the additional delay caused by capacitive loading, and the ground bounce created when many outputs switch simultaneously.

Voltage/Current Characteristics of XC4000-Family Outputs

Figures 1 through 4 show the output source and sink currents, both drawn as absolute values. Note that the XC4000 families have an n-channel only, totem-pole like output structure that pulls a High output to a voltage level that is one threshold drop lower than V_{CC}. When driving inputs that have a 1.4-V threshold, this lower V_{OH} offers faster speed and more symmetrical switching delays. The XC4000H outputs offer an optional p-channel output driver and thus rail-to-rail switching a configuration option for each individual pin.

These curves represent typical devices. Measurements were taken at V_{CC}= 5 V, T = 25°C. These characteristics vary by manufacturing lot, and will be affected by future changes in minimum device geometries, notably a change from 0.8 µ to 0.6 µ. These characteristics are not production-tested as part of the normal device test procedure; they can, therefore, not be guaranteed. Although these measurements show that the output sink and source capability far exceeds the guaranteed data sheet limits, continuous high-current operation beyond the data sheet limits can cause metal migration of the on-chip metal traces, permanently damaging the device. Output currents in excess of the data-sheet limits are, therefore, not recommended for continuous operation. These output characteristics can, however, be used to calculate or model output transient behavior.

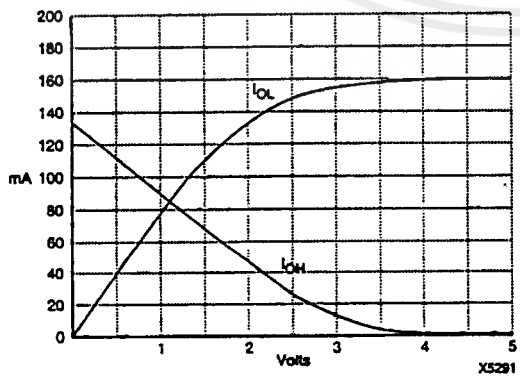


Figure 1. Output Voltage/Current Characteristics for XC4005-5

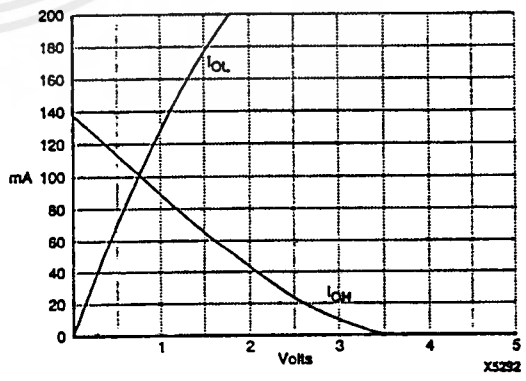


Figure 2. Output Voltage/Current Characteristics for XC4002A

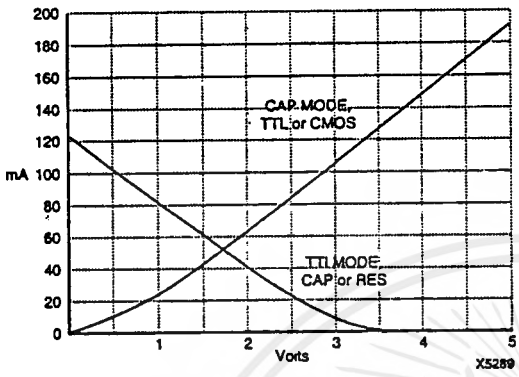


Figure 3. Output Voltage/Current Characteristics for XC4005H

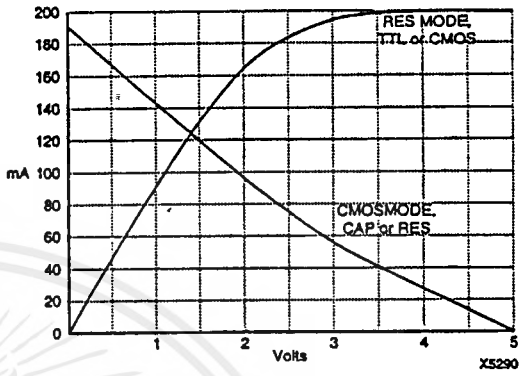


Figure 4. Output Voltage/Current Characteristics for XC4005H

Additional Output Delays When Driving Capacitive Load

Xilinx Product Specifications in Section 2 give guaranteed worst-case output delays with a 50-pF load.

The values given in Table 1 are actual measurements on a small number of mid-93 production XC4005-5, XC4005A-5 and XC4003H-5 devices, all in PQ208 packages, measured at room temperature and V_{CC} = 5.5 V. Listed is the additional output delay, measured crossing 1.5 V, relative to the delays specified in this 1994 Data Book, Section 2.

These parameters are not part of the normal production test flow, and can, therefore, not be guaranteed.

Table 2 lists the additional output delay, measured crossing 1.5-V, relative to the delay with 100 pF load shown in Table 1.

Example:

ΔT High-to-Low for XC4005-5 with Fast-mode output driving 250 pF:

1.2 ns (from Table 1) plus (250-100) pF • 1.5 ns/100 pF = 1.2 ns + 2.25 ns = 3.45 ns

Total propagation delay, clock OK to pad:

T_{OKPOF} + 3.45 ns = 7.0 ns + 3.45 ns = 10.45 ns

Table 1. Increase in Output Delay When Driving Light Capacitive Loads (<150 pF)

Family	Slew Mode	High-to-Low 10	50	100	Low-to-High 10	50	100	pF
XC4000	Slow	-1.6	0*	1.4	-1.4	0*	1.4	ns
	Fast	-1.6	0*	1.2	-1.2	0*	1.1	ns
XC4000A	Slow	-2.2	0*	1.7	-1.5	0*	1.4	ns
	MedSlow	-1.8	0*	1.6	-1.3	0*	1.1	ns
	MedFast	-1.8	0*	1.3	-1.3	0*	1.1	ns
	Fast	-2.0	0*	1.2	-1.0	0*	1.3	ns
XC4000H	Cap-CMOS	-2.2	0*	1.9	-0.5	0*	0.7	ns
	Res-CMOS	-1.4	0*	1.2	-1.0	0*	0.8	ns
	Cap-TTL	-1.9	0*	1.6	-1.2	0*	1.2	ns
	Res-TTL	-1.1	0*	1.0	-1.1	0*	1.0	ns

*Zero by definition.

Table 2. Increase in Output Delay When Driving Heavy Capacitive Loads (>150 pF)

Family	Slew Mode	High-to-Low	Low-to-High	
XC4000	Slow	1.7	1.2	ns/100 pF
	Fast	1.5	1.2	ns/100 pF
XC4000A	Slow	2.1	1.2	ns/100 pF
	MedSlow	1.5	1.1	ns/100 pF
	MedFast	1.0	1.1	ns/100 pF
	Fast	0.9	1.1	ns/100 pF
XC4000H	Cap-CMOS	2.7	0.9	ns/100 pF
	Res-CMOS	1.8	1.0	ns/100 pF
	Cap-TTL	2.1	1.3	ns/100 pF

Ground Bounce in XC4000 Devices

Ground-bounce is a problem with high-speed digital ICs, when multiple outputs change state simultaneously causing undesired transient behavior on an output, or in the internal logic. This is also referred to as the Simultaneous Switching Output (SSO) problem. Ground bounce is primarily due to current changes in the combined inductance of ground pins, bond wires, and ground metallization. The IC-internal ground level deviates from the external system ground level for a short duration (a few nanoseconds) after multiple outputs change state simultaneously. Ground bounce affects outputs that are supposed to be stable Low, and it also affects all inputs since they interpret the incoming level by referencing it to the internal ground. If the ground bounce amplitude exceeds the actual instantaneous noise margin, then a non-changing input will be interpreted as a short pulse with a polarity opposite to the ground bounce.

V_{CC} bounce is not as important as ground bounce, because it is of lower magnitude due to the weaker pull-up transistors. Also, the noise immunity in the High state is usually better than in the Low state, and input levels are referenced to ground, not V_{CC}. All this is the result of our industry's TTL heritage.

Test Method

Data was taken on XC40005-5, XC40005A-5, and XC40003H-5 devices, all in the PQ208 package, soldered to the Xilinx Ground Bounce Test Board. Pin 82, two pins away from the nearest ground pin, was configured as a permanently Low output driver, effectively monitoring the internal ground level. The simultaneously switching outputs were on pins 80 and 83, for two outputs switching; additionally, pins 80 and 86 were used for four outputs switching (81 and 84 on the XC40003H). The closest ground pins are 79 and 90.

Four ground-bounce parameters were measured at room temperature, with Vcc set at 5.5 V as shown in Figure 1.

V_{OLP-HL} Peak ground noise when outputs switch High-to-Low

V_{OLV-HL} Valley ground noise when outputs switch High-to-Low

V_{OLP-LH} Peak ground noise when outputs switch Low-to-High

V_{OLV-LH} Valley ground noise when outputs switch Low-to-High

All four parameters can affect system reliability.

The two positive peak values can cause problems with a signal leaving the ground bounce chip, driving another chip. The positive ground bounce voltage is added to the V_{OL}, and may exceed the receiving input's noise margin. A continuously logic Low input may thus be interpreted as a short-duration High pulse.

The two negative valley parameters can cause problems with a signal arriving at the ground-bounce chip, where the on-chip ground reference is negative, reducing the Low-level noise immunity. The incoming voltage may not be Low enough, and may, therefore, be interpreted as a short-duration High input pulse.

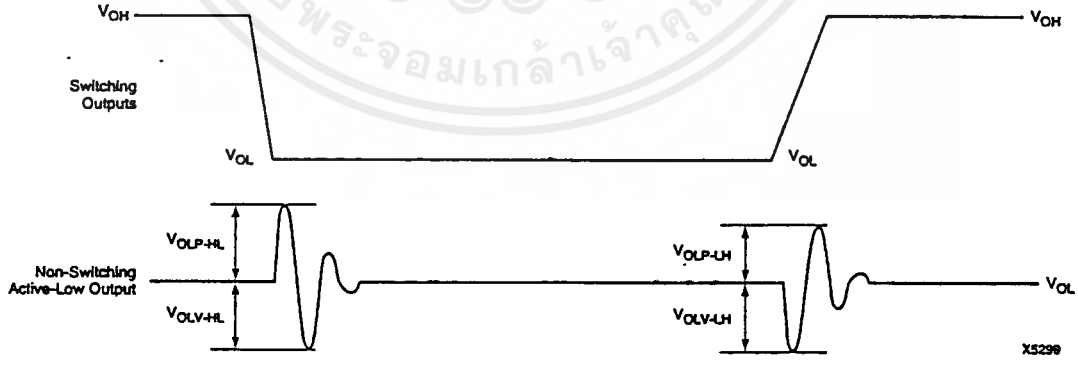


Figure 5. Ground Bounce

Table 3. Ground Bounce, 16 Outputs Switching, Each With 50 pF Load, V_{CC} = 5.5 V

Family	Slew Mode	High-to-Low		Low-to-High		
		V_{OLP}	V_{OLV}	V_{OLP}	V_{OLV}	
XC4000	Slow	670	480	240	240	mV
	Fast	1,170	710	480	660	mV
XC4000A	Slow	565	425	290	310	mV
	MedSlow	950	610	500	780	mV
	MedFast	1,140	860	500	780	mV
	Fast	1,240	910	500	810	mV
XC4000H	Cap-CMOS	940	660	660	770	mV
	Res-CMOS	1,250	1,210	590	480	mV
	Cap-TTL	830	460	450	570	mV
	Res-TTL	1060	980	440	350	mV

Table 4. Ground Bounce, 16 Outputs Switching, Each With 150 pF Load, V_{CC} = 5.5 V

Family	Slew Mode	High-to-Low		Low-to-High		
		V_{OLP}	V_{OLV}	V_{OLP}	V_{OLV}	
XC4000	Slow	740	330	210	280	mV
	Fast	1,180	420	350	710	mV
XC4000A	Slow	615	270	245	330	mV
	MedSlow	960	310	820	370	mV
	MedFast	1,140	620	370	790	mV
	Fast	1,200	640	370	810	mV
XC4000H	Cap-CMOS	1,080	390	470	860	mV
	Res-CMOS	1,500	820	420	590	mV
	Cap-TTL	900	250	320	610	mV
	Res-TTL	1,170	660	300	470	mV

Interpretation of the results

Ground bounce is a linear phenomenon. When multiple outputs switch, the total ground bounce is the sum of the ground-bounce values caused by individual outputs switching. Since the actual switching of multiple outputs is usually not quite simultaneous, small timing differences between the switching outputs, caused by routing delays, can indirectly affect the amplitude. With low capacitive loading, < 50 pF, the peaks and valleys might even partially cancel each other. With larger capacitive loads, the tendency is for valleys to combine with valleys and peaks to combine with peaks.

In most devices tested, the load capacitance does not directly affect the ground-bounce amplitude, but it does affect the duration of the ground-bounce signals.

On the fastest outputs, minimal load capacitance created a ground-bounce resonant frequency of 340 MHz, with a half-cycle time of 1.5 ns. Such a signal exceeds 90% of its peak amplitude for about 0.4 ns.

With a 50 pF load on the switching outputs, the ground bounce resonant frequency is 85 to 97 MHz, with a half-cycle time of 5 to 6 ns, staying 1.7 ns above 90% of peak amplitude.

With a 150 pF load on the switching outputs, the ground bounce resonant frequency is 40 to 60 MHz, with a half-cycle time of 8 to 12 ns, staying 3 ns above 90% of peak amplitude.

The main problem with large load capacitances is not an increase in amplitude, but rather an increase in duration of the ground-bounce signal. The amplitude is mainly affected by the number of outputs switching simultaneously, and by the slew-rate mode of these outputs. Switching outputs closer to the monitoring output also cause larger peaks and valleys than outputs further away.

Guidelines for reducing ground-bounce effects

- Minimize the impedance of the system ground distribution network and its connection to the IC pins. PQFPs are best suited, PGAs are worst, and PLCCs are in-between.
- Use PC-boards with ground- and V_{CC}-planes, connected directly to the ICs' supply pins. Place decoupling capacitors very close to these ground and V_{CC} pins.
- Keep the ground plane as undisturbed as possible. A row of vias can easily cause a dynamic ground-voltage drop.
- Keep the clock inputs physically away from the outputs that create ground bounce, and connect clocks to input pins that are close to a ground pin. Make sure that all clock and asynchronous inputs have ample noise margin, especially in the Low state.
- If possible, avoid simultaneous switching by staggering output delays, e.g. through additional local routing of signals or clocks.
- Spread simultaneously switching outputs around the IC periphery. For a 16-bit bus, use two outputs each on either side of four ground pins.

Ground-Bounce vs Delay Trade-Off

After the external sources of ground bounce have been reduced or eliminated. the designer can trade reduced ground bounce for additional delay by selecting between families and slew-rate options. Figures 2 and 3 show the available choices, based on 16 outputs switching simultaneously High-to-Low.

Summary

For light capacitive loads, the XC4000 and XC4000A, both in slow mode perform well, with ground bounce below 800 mV; the additional delay, compared to fast mode, is only 4 to 5 ns. For larger capacitive loads, the XC4000H in Capacitive-TTL mode offers the best trade-off.

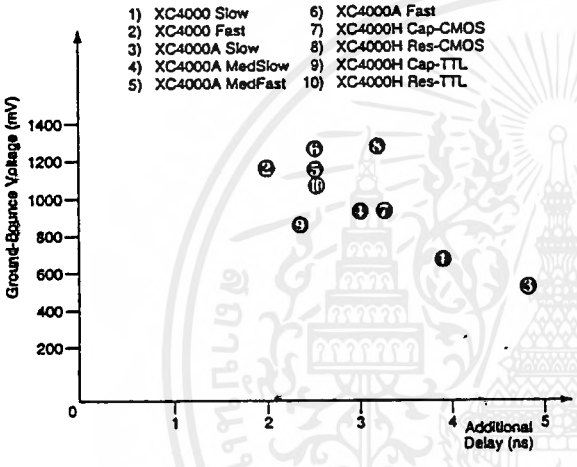


Figure 6. Ground-Bounce vs Delay Trade-off for 16 Outputs Switching 50 pF each

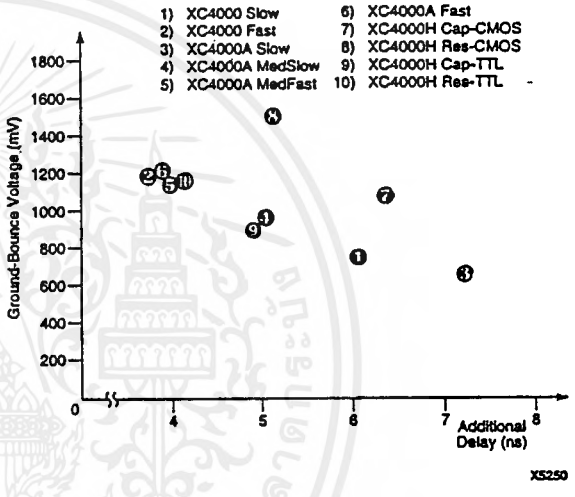


Figure 7. Ground-Bounce vs Delay Trade-off for 16 Outputs Switching 150 pF each

บรรณานุกรม

[1] นรินทร์ วัฒนกุล ดิจิตอลเบื้องต้นและไมโครคอมพิวเตอร์ชิพเดี่ยว : แผนกไฟฟ้ากำลัง สถาบันเทคโนโลยีราชมงคลวิทยาเขตพระนครเหนือ

[2] บรรจง ปิยธำรง และ สุรเชษฐ์ ศรีพลกรัง การออกแบบระบบดิจิตอลโดยใช้ VHDL : การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 18

[3] Jean-Michael Berge and Rolan Airiau. Circuit Synthesis with VHDL : Kluwer Acadamic Publishers. 1994

[4] Richard Larry Ukeiley. Field Programmable Gate Arrays (FPGAs) The 3000 series : Practice-Hall Inc.1993

[5] Viewlogic Systems Inc. VHDL Modeling :Viewlogic System Inc.1994

[6] Xilinx Inc. The Programmable Logic Data Book : Xilinx Inc.1994

[7] Xilinx Inc. Viewlogic Tutorials : Xilinx Inc.1994

[8] Xilinx Inc. XACT Hardware & Peripherals Guide : Xilinx Inc.1994

[9] Xilinx Inc. XACT Reference Guide, Volume II : Xilinx Inc.1994

[10] Zainalabedin Navabi. VHDL Analysis and Modeling of Digital System : McGRAW-HILL. 1993