การออกแบบ PLC โดยใช้ FPGA
PLC Design Using FPGA

นาย ศิริ จงเกษมถาวร 41014417
นาย สุธรรม สุจริตธรรมกุล 41014475

อาจารย์ที่ปรึกษา
รศ. บรรจง ปียธำรง

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
สาขาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระเจ้าจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบ PLC โดยใช้ FPGA

PLC design using FPGA

ผู้จัดทำ

1. นายศิริ จงเกษมถาวร รหัสประจำตัว 41014417
2. นายสุธรรม สุจริตธรรมกุล รหัสประจำตัว 41014475

อาจารย์ที่ปรึกษา

(รศ. บรรจง ปิยธำรง)

การออกแบบ PLC โดยใช้ FPGA

นายศิริ จงเกษมถาวร 41014417
นายสุธรรม สุจริตธรรมกุล 41014475
ร.ศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

โครงงานนี้เป็นการออกแบบ PLC (Programmable Logic Controller) โดยใช้ FPGA (Field Programmable Gate Array) ในการออกแบบส่วนควบคุมหรือ CPU (Central Processing Unit) และใช้ภาษา VHDL ในการบรรยายการทำงานของวงจร

PLC เป็นอุปกรณ์ที่ใช้แทนวงจรรีเลย์ในการควบคุมอุปกรณ์หรือเครื่องจักรในโรงงานอุตสาหกรรม โดยในการออกแบบ CPU ของ PLC จะใช้หลักการของ Top-Down Design โดยทำการเก็บรายละเอียด ข้อมูลของตัวควบคุมที่ต้องการ จากนั้นทำการแบ่งส่วนตัวควบคุม ออกเป็นส่วน ๆ แล้วจึงทำการเขียนแต่ละส่วนด้วยภาษา VHDL และตรวจสอบการทำงานของแต่ละส่วนด้วยซอฟท์แวร์ซิมมูเลชั่น จนกระทั้งแน่ใจว่าแต่ละส่วนที่เขียนขึ้นมีการทำงานที่ถูกต้อง จึงนำแต่ละส่วนที่เขียนขึ้นนี้มารวมเข้าด้วยกัน และทำการทดสอบจนแน่ใจว่าแต่ละส่วนที่สร้างขึ้นนั้นสามารถทำงานร่วมกันได้อย่างไม่มีปัญหา และนำ VHDL ที่เขียนขึ้นมาเข้าสู่กระบวนการ Synthesis เพื่อแปลงให้เป็นวงจรในระดับ Gate-Level และนำลง FPGA ต่อไป

โดยในภาคเรียนนี้ทำการพัฒนาตัวตัวควบคุมของ PLC ให้สามารถประมวลผลคำสั่งพื้นฐานของ PLC ได้ซึ่งตัวควบคุมที่ได้ทำการออกแบบสามารถประมวลผลภาษา Ladder ของ PLC ได้ และมีผลการทำงานที่น่าพอใจ

PLC DESIGN USING FPGA

Mr. Siri Chongasamethaworn 41014417
Mr. Sutham Suitjarritthammakurn 41014475
Assoc.Prof. Banjong Piyathamrong Advisor
2001

Abstract

This project is design PLC (Programmable Logic Controller). In addition, CPU is design by FPGA (Field Programmable Gate Array) and use VHDL as a language for modeling CPU of PLC.

PLC is a device that use as relay circuit for control another device or machine in factory. In the design process, the researchers use top-down design model to develop the CPU. Initially, collect information and specification of PLC and it CPU then coding each part in VHDL language and test each part with simulation software to ensure that these parts are working correctly, then combine them together and test the corrective of the cooperation between these parts. Then bring the derived VHDL code into the synthesis process to convert into gate-level net list. Finally, load all of them into FPGA

In this term, the CPU derived in this project can process Ladder language of PLC and has capability of instructions. In addition, the control unit is working correctly.

สารบัญ

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบัน ในการควบคุมขบวนการผลิตของโรงงานอุตสาหกรรมเกือบทุกประเภท มักใช้อุปกรณ์ควบคุมที่เรียกว่า Programmable Logic Control หรือ PLC ซึ่งเป็นอุปกรณ์ที่พัฒนาขึ้นมาเพื่อใช้แทนระบบควบคุมแบบเก่าที่ใช้รีเลย์ ถึงแม้ว่าการใช้ PLC ในโรงงานอุตสาหกรรมในประเทศไทยกำลังได้รับความนิยมเพิ่มมากขึ้นเรื่อย ๆ แต่เนื่องจากปัจจุบันตัว PLC ยังมีราคาค่อนข้างแพงอยู่ และการนำมาใช้งานต้องมีการนำเข้าจากต่างประเทศ ทำให้ยังคงไม่แพร่หลายในประเทศมากเท่าที่ควรนัก การนำ FPGA มาใช้ในการออกแบบ PLC จะนำมาใช้ในส่วนของ ตัวควบคุมหลัก (Central Processing Unit : CPU) ซึ่งสามารถทำได้ง่าย และมีประสิทธิภาพสูง จึงเป็นทางเลือกหนึ่งในการประหยัดต้นทุนในการใช้งานได้เป็นอย่างดี

การออกแบบวงจรดิจิตอลขนาดใหญ่ที่ระดับเกต (Gate-Level) นั้น เป็นงานที่ยุ่งยาก และซับซ้อนมาก เป็นผลให้งานที่พัฒนามีความล่าช้า และเสร็จไม่ตรงตามกำหนด การนำภาษาบรรยายการทำงานของวงจร (Hardware Description Language: HDL) มาใช้ในการออกแบบจะช่วยแก้ปัญหานี้ได้

การออกแบบวงจรใน FPGA จะเป็นการใช้ภาษาบรรยายการทำงานของวงจร ซึ่งสามารถตรวจสอบความถูกต้องได้ด้วยการใช้ซอฟท์แวร์ซิมมูเลชั่น (Simulation) จนแน่ใจว่างานที่ออกแบบนั้นมีการทำงานที่ถูกต้อง จึงนำงานนั้นไปเข้าสู่กระบวนการสังเคราะห์ (Synthesis) เพื่อแปลงภาษาบรรยายการทำงานของวงจรที่เขียนให้อยู่ในระดับเกต เพื่อนำลง FPGA ต่อไป

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาโครงสร้างและการใช้งานของ FPGA

1.2.2 เพื่อให้สามารถวิเคราะห์ ออกแบบ และพัฒนาวงจรใน FPGA โดยใช้ภาษา VHDL

1.2.3 ทดลองการออกแบบวงจร PLC ที่มีประสิทธิภาพในราคาที่ถูกลง

1.3 ขอบเขตของโครงงาน

ศึกษาโครงสร้างและการทำงานของ PLC รวมถึงภาษาแลดเดอร์ (Ladder) ที่ใช้ในการควบคุมการทำงานของ PLC

ศึกษาและทำความเข้าใจการใช้งานและโครงสร้างของอุปกรณ์ FPGA และภาษาที่ใช้ในการบรรยายการทำงานของวงจรอันได้แก่ภาษา VHDL ซึ่งเป็นภาษาที่เป็นมาตรฐานในการนำมาใช้ในการออกแบบหรือพัฒนาไมโครโพรเซสเซอร์

ออกแบบตัว CPU ของ PLC โดยให้สามารถทำงานคำสั่งพื้นฐานต่าง ๆ ของ PLC ได้

ทำการทดลองตัวควบคุม PLC จนสามารถทำงานได้ตามจุดประสงค์ และที่ออกแบบไว้

สำหรับในโครงงานนี้ตัว PLC ที่เลือกใช้ในการอ้างอิงคือ PLC ของบริษัท Omron รุ่น C20 ซึ่งเป็น PLC ขนาดเล็ก และมีการทำงานตามมาตรฐานคำสั่งต่าง ๆ ของ PLC ได้อย่างครบถ้วน

ตัว FPGA ที่เลือกใช้เป็นตัว FPGA ของบริษัท Xilinx โดยรุ่นที่เลือกใช้คือรุ่น XS40 ซึ่งเป็นรุ่นที่มีราคาถูก และมีขนาดเล็กซึ่งง่ายต่อการศึกษาและใช้งาน

1.4 วิธีการดำเนินงาน

เริ่มจากทำการศึกษาการทำงานของ PLC และภาษา Ladder ที่ใช้ในการสั่งงาน PLC รวมถึงคำสั่งและความสามารถต่าง ๆ ของ PLC ซึ่งจะกล่าวถึงรายละเอียดในบทที่ 2

ศึกษาการทำงาน ความสามารถของ FPGA และบอร์ดที่นำมาใช้งานซึ่งจะกล่าวถึงรายละเอียดในบทที่ 3 จากนั้นทำการศึกษาการใช้ภาษา VHDL ซึ่งเป็นภาษาที่ใช้ในการอธิบายการทำงานซึ่งจะกล่าวถึงรายละเอียดในบทที่ 4 และศึกษาหลักการการออกแบบอุปกรณ์ฮาร์ดแวร์ รวมถึงการใช้ซอฟต์แวร์ทูล (Software Tools) ในการออกแบบ อันได้แก่ ModelSim PE/Plus 5.4 SE, Leonardo Spectrum v2000.1a และ Xilinx Foundation Series 1.4.12 ซึ่งจะกล่าวถึงรายละเอียดต่อไปในบทที่ 5

ออกแบบภาพรวมของตัวควบคุมใน PLC โดยเขียนความต้องการของอุปกรณ์ ออกแบบวงจร (Data Path) ของตัวควบคุม จากนั้นเขียนวงจรโดยใช้ภาษา VHDL โดยแบ่งออกเป็นส่วน ๆ และทำการทดสอบโดยการใช้ Test bench ที่เขียนขึ้นมาเองและกระบวนการสังเคราะห์ จากนั้นทำการรวมแต่ละส่วนเข้าด้วยกัน ทำการทดสอบจนกระทั่งแน่ใจว่าสามารถทำงานได้ถูกต้องจริง

บทที่ 2
PLC

PLC หรือ Programmable Logic Controller เป็นอุปกรณ์ควบคุมอิเล็กทรอนิกส์ที่มีหน่วยความจำในการเก็บโปรแกรมสำหรับควบคุมการทำงานของอุปกรณ์ต่าง ๆ ที่ต่อกับขั้วเข้า และขั้วออกของมัน ซึ่ง PLC นี้ยังมีชื่อเรียกอย่างอื่นอีกเช่น PC (Programmable Controller) และ SC (Sequence Controller) และ Sequencer ก็ได้

PLC ถือเป็นอุปกรณ์ควบคุมตัวสำคัญตัวหนึ่งในการควบคุมเครื่องจักรในโรงงานอุตสาหกรรม ให้ทำงานอย่างอัตโนมัติ ในระบบ FA (Factory Automation) PLC จะถูกใช้ในการปรับปรุงประสิทธิภาพการทำงานของเครื่องจักร ลดของเสีย ทำให้เครื่องจักรทำงานได้เองโดยอัตโนมัติ เป็นการลดงานของคนงาน PLC ที่ใช้ในโรงงานอุตสาหกรรมจะมีขนาดตั้งแต่ใหญ่เป็นระบบควบคุม จนกระทั่งเป็น PLC ตัวเล็ก ๆ ซึ่งใช้ตัวเดียวโดด ๆ โดยในโครงงานนี้จะเป็นเพียงแค่การจำลอง PLC ขนาดเล็กเท่านั้น

2.1 การทำงานของ PLC

PLC เป็นอุปกรณ์ทางอิเล็กทรอนิกส์ทำหน้าที่ควบคุมเครื่องจักรหรือกระบวนการผลิตโดยใช้โปรแกรมในหน่วยความจำกำหนดเงื่อนไขการควบคุมผ่านทางหน่วยอินพุต / เอาต์พุต ดังรูปที่ 2.1

รูปที่ 2.1 โครงสร้างของ PLC

PLC จะรับสัญญาณที่เป็นคำสั่งจากสวิตช์ปุ่มกด สวิตช์เลือก และสวิตช์ตัวเลข ซึ่งอยู่ที่แผงควบคุมเครื่องจักร นอกจากนั้น ยังรับสัญญาณที่มาจากอุปกรณ์ตรวจวัดสภาพการทำงานของเครื่องจักร เช่น ลิมิตสวิทช์ (Limit switch) พล็อกซิมิตีสวิตช์ (Proximity switch) สวิตช์แสง และสวิตช์ตรวจจับชนิดต่าง ๆ จากนั้น PLC จะส่งสัญญาณออกไปที่ขั้วออกเพื่อขับเคลื่อนอุปกรณ์ต่าง ๆ เช่น มอเตอร์โซลินอยด์ และคัตช์แม่เหล็กไฟฟ้า หรือขับพวกหลอดขับแสดง หลอดตัวเลข

PLC รับสัญญาณเข้ามาทางขั้วเข้า และให้สัญญาณออกทางขั้วออก การให้สัญญาณออกนี้จะเป็นไปตามโปรแกรมที่เก็บไว้ในเครื่อง PC การขับอุปกรณ์ซึ่งเป็นโหลดขนาดเล็ก เช่น โซลินอยด์ตัวเล็ก หรือหลอดแสดงนั้น สามารถขับโดยตรงจากขั้วออกได้ แต่ถ้าเป็นอุปกรณ์ที่เป็นโหลดขนาดใหญ่ ใช้ไฟมาก เช่น มอเตอร์สามเฟส หรือโซลินอยด์ตัวใหญ่ จำเป็นต้องต่อผ่านคอนแทคเตอร์ และรีเลย์เพื่อช่วยขยายกำลังขับ พวกคอนแทคเตอร์ รีเลย์ขับเบรกเกอร์ เหล่านี้ จะถูกติดตั้งอยู่ภายในตู้ควบคุมเดียวกับ PLC

การประมวลผลอินพุต (Input Processing)

ก่อนที่ PLC จะทำงานตามโปรแกรมในหน่วยความจำ PLC จะอ่านสัญญาณจากขั้วเข้าทั้งหมดเป็นข้อมูลมาเก็บไว้ในหน่วยความจำอินพุต (Input Image Memory) ในระหว่างที่ PLC ทำงานในโปรแกรม แม้สัญญาณที่ขั้วเข้าจะต่างไป แต่ข้อมูลในหน่วยความจำอินพุตยังเหมือนเดิมจนกว่าจะจบการทำงานในรอบนั้น และอ่านอินพุตในรอบถัดไป

การประมวลผลโปรแกรม (Program Processing)

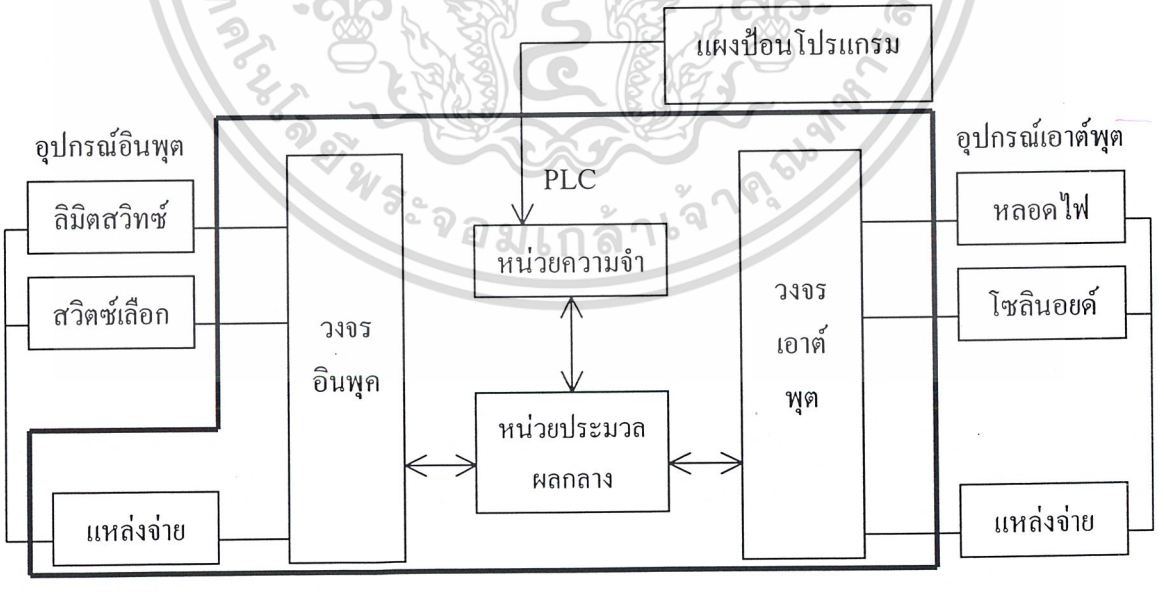
เมื่อ PLC ทำงานตามโปรแกรมในหน่วยความจำจะอ่านข้อมูล ON/OFF จากหน่วยความจำอินพุตและหน่วยความจำที่เก็บข้อมูลของอุปกรณ์ต่าง ๆ ในระหว่างทำงานผลของการคำนวณก็จะไปเก็บไว้ในหน่วยความจำอุปกรณ์เช่นเดียวกัน ดังนั้นจึงกล่าวได้ว่าเมื่อ PLC ทำงานตามโปรแกรม ข้อมูลในหน่วยความจำจะเปลี่ยนแปลงไปตามการทำงานนั้น

การประมวลผลเอาต์พุต (Output Processing)

เมื่อ PLC จบการทำงานตามโปรแกรมจะนำข้อมูลในหน่วย ความจำเอาต์พุตส่งออกไปขับรีเลย์เอาต์พุตทุกตัวในเวลาเดียวกัน

2.2 โครงสร้างของ PLC

ภายใน PLC จะมีไมโครคอมพิวเตอร์ และหน่วยความจำเป็นองค์ประกอบวงจรที่สำคัญ และมีวงจรอินพุตและเอาต์พุตสำหรับต่อกับอุปกรณ์ภายนอก นอกจากนั้นยังมีแผงป้อนโปรแกรมในการอ่านและเขียนโปรแกรมเข้าไปในหน่วยความจำของ PLC



รูปที่ 2.2 แสดงโครงสร้างของ PLC

2.2.1 หน่วยประมวลผลกลาง

หน่วยประมวลผลกลางมีหน้าที่ดูแลการทำงานทั้งหมดของ PLC คือนำโปรแกรมของผู้ใช้มาปฏิบัติเพื่อควบคุมอุปกรณ์ภายนอกตามเงื่อนไขการควบคุมที่ผู้เขียนโปรแกรมต้องการควบคุมการติดต่อรับส่งข้อมูลระหว่างอินพุตและเอาต์พุต

2.2.2 หน่วยความจำ

หน่วยความจำทำหน้าที่เก็บโปรแกรม และข้อมูลต่าง ๆ ที่ PLC ใช้ในการประมวลผล PLC แบ่งหน่วยความจำออกเป็น 2 ส่วนคือ

1. หน่วยความจำระบบ (System Memory) เก็บข้อมูลของระบบ และข้อมูลที่จำเป็นในการทำงานของระบบ
2. หน่วยความจำผู้ใช้ (User Memory) เก็บโปรแกรมผู้ใช้ ข้อมูลของหน่วยอินพุต / เอาต์พุต และอุปกรณ์ภายใน

หน่วยความจำของ PLC ยังมีลักษณะการใช้ที่แตกต่างกัน บางส่วนต้องการหน่วยความจำที่มีความเร็วสูง และไม่อนุญาตให้ผู้ใช้เปลี่ยนแปลงข้อมูลภายในได้ หรือในบางครั้งต้องการหน่วยความจำที่สามารถเปลี่ยนแปลงแก้ไขได้ หรือแม้แต่หน่วยความจำที่เก็บข้อมูลได้ถึงแม้จะไม่มีกระแสไฟเลี้ยง หน่วยความจำที่ใช้กับ PLC แบ่งเป็น 2 ชนิดคือ

1. โวลาไทล์ (Volatile) ข้อมูลในหน่วยความจำชนิดโวลาไทล์จะสูญหายถ้าไม่มีกระแสไฟฟ้า
2. นอนโวลาไทด์ (Nonvolatile) ข้อมูลในหน่วยความจำชนิดนอนโวลาไทล์จะเก็บรักษาข้อมูลไว้ถึงแม้จะไม่มีกระแสไฟฟ้า

หน่วยความจำที่เก็บโปรแกรม

หน่วยความจำที่ใช้ในการเก็บโปรแกรมใน PLC มีอยู่ 3 ชนิดคือ

1. RAM ที่มีแบตเตอรี่

   หน่วยความจำนี้จะมีแบตเตอรี่เล็ก ๆ ต่อไว้ เพื่อใช้ในการเลี้ยงข้อมูลเมื่อเกิดไฟดับ เราสามารถเพิ่มขนาดของ RAM นี้ให้ใหญ่ขึ้นได้โดยการต่อกล่อง RAM เสริมเข้าไปในเครื่อง การอ่านและเขียนโปรแกรมลงใน RAM ทำได้ง่ายมากจึงเหมาะกับการใช้งานในระยะทดลองเครื่องที่มีการเปลี่ยนแปลงแก้ไขโปรแกรมบ่อย

2. EPROM

   หน่วยความจำชนิด EPROM นี้จะต้องใช้เครื่องพิเศษในการเขียนและลบโปรแกรม มีข้อดีตรงที่โปรแกรมจะไม่สูญหายแม้ไฟจะดับ จึงเหมาะสมกับการใช้งานที่ไม่ต้องการเปลี่ยนแปลงโปรแกรมแล้ว ซึ่งการใช้งานเพียงแค่เปลี่ยนกล่อง EPROM ก็สามารถเปลี่ยนโปรแกรมตามที่ต้องการได้

3. EEPROM

   หน่วยความจำชนิด EEPROM นี้ไม่ต้องใช้เครื่องมือพิเศษในการเขียน และลบโปรแกรม ใช้วิธีการทางไฟฟ้าเหมือนกับ RAM นอกจากนั้นก็ไม่จำเป็นต้องมีแบตเตอรี่สำรองไฟในกรณีไฟดับด้วย ซึ่งจะมีราคาที่แพงกว่าชนิด EPROM แต่จะรวมคุณสมบัติที่ดีของทั้ง RAM และ EPROM เอาไว้ด้วยกัน

2.2.3 เครื่องมือในการทำโปรแกรม

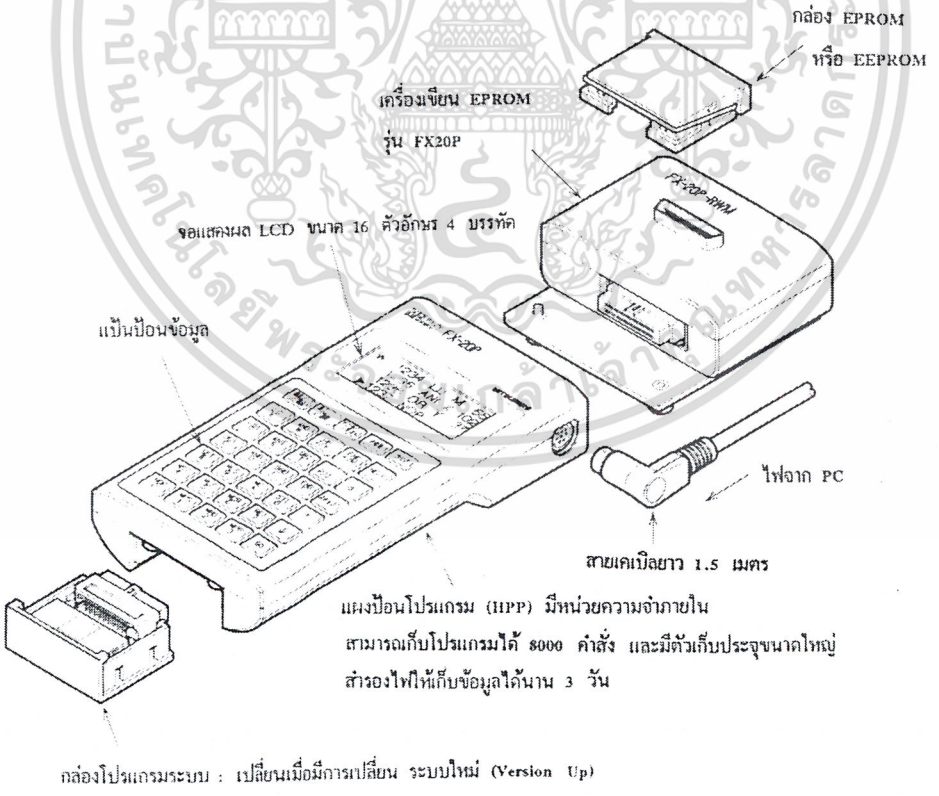
ทำหน้าที่ป้อนโปรแกรมลงในหน่วยความจำ ซึ่งตัวประมวลผลการจะนำไปประมวลผลเพื่อควบคุมอุปกรณ์ภายนอกอีกทีหนึ่ง เครื่องป้อนโปรแกรมที่นิยมใช้งานกันมี 2 ประเภทคือ

1. แผงป้อนโปรแกรม (HPP)

เป็นเครื่องมือป้อนโปรแกรมการต่อวงจรซีเคว้นซ์ที่ง่ายที่สุดของ PLC แผงป้อนโปรแกรมนี้จะต่อเข้ากับ PLC โดยสายเคเบิ้ล และคอนเน็ตเตอร์ เราใช้แผงป้อนโปรแกรมนี้ในการป้อนโปรแกรมเข้าไปในหน่วยความจำของ PLC นอกจากนั้นยังใช้ในการตรวจสอบการทำงานภายใน PLC ได้อีกด้วย

ด้านบนของแผงป้อนโปรแกรมยังสามารถต่อเครื่องเขียน EPROM ซึ่งใช้ในการเขียนและอ่านโปรแกรมลงในกล่อง EPROM หรือ EEPROM ได้

เรายังสามารถใช้แผงป้อนโปรแกรมโดด ๆ ในการทำโปรแกรมเตรียมไว้ก่อน โดยไม่จำเป็นต้องต่อกับ PLC การเตรียมโปรแกรมแบบนี้เรียกว่า การทำโปรแกรมแบบออฟลายน์ (Offline Programming) จะต้องมีอะแดปเตอร์จ่ายไฟตรงให้แผงป้อนโปรแกรมในกรณีออฟลายน์นี้



รูปที่ 2.3 แสดงภาพแผงป้อนโปรแกรม (HPP)

2. เครื่องป้อนโปรแกรม (GPP: Graphic Programming Panel)

เครื่องป้อนโปรแกรมจะมีลักษณะเป็นจอ CRT หรือ LCD โดยการป้อนโปรแกรมจะเป็นการป้อนโปรแกรมที่มีการแสดงผลเป็นกราฟฟิก แตกต่างกับแบบ HPP ที่แสดงผลแบบตัวอักษรเท่านั้น

2.3 ข้อดีของ PLC เมื่อเปรียบเทียบกับรีเลย์

	รีเลย์	PLC
ฟังก์ชั่น	ใช้ในการควบคุมซับซ้อนได้ ถ้าใช้รีเลย์จำนวนมาก	การควบคุมซับซ้อนเพียงใดก็สามารถโปรแกรมได้
การเปลี่ยนแปลงการควบคุม	เปลี่ยนได้โดยการเดินสายใหม่	เปลี่ยนได้โดยการเปลี่ยนโปรแกรม
ความเชื่อถือได้	ปรกติเชื่อถือได้ แต่มีปัญหาเรื่องจุดต่อหลวม และอายุการใช้งานของรีเลย์	องค์ประกอบหลักคือ สารกึ่งตัวนำ จึงไม่มีปัญหาเรื่องจุดต่อสายหลวม มีความน่าเชื่อถือได้สูง
ใช้งานได้เอนกประสงค์	ใช้ได้กับงานที่ออกแบบมาเฉพาะเท่านั้น	ใช้งานได้เอนกประสงค์ โดยการเปลี่ยนโปรแกรม
การขยายระบบ	ทำได้ยากต้องเพิ่มอุปกรณ์ หรือแก้ไขเปลี่ยนแปลงใหม่	ขยายได้เรื่อย ๆ จนเต็มขีดความสามารถ
บำรุงรักษาง่าย	ต้องตรวจเช็คบ่อย ๆ และต้องเปลี่ยนอุปกรณ์ที่มีอายุจำกัด	ซ่อมโดยการเปลี่ยนส่วนประกอบ
การเข้าใจเทคนิค	มีใช้กันแพร่หลาย คนส่วนมากเข้าใจเทคนิคการใช้รีเลย์	ยังไม่แพร่หลาย คนส่วนใหญ่ยังไม่เข้าใจเทคนิคการใช้งาน
ขนาด	ใหญ่	เล็ก และไม่ใหญ่โตตามความซับซ้อนของการควบคุม
เวลาในการออกแบบ	ต้องเขียนแบบจำนวนมาก และต้องใช้เวลาในการประกอบ และการทดสอบ	ออกแบบได้ง่ายแม้จะเป็นการควบคุมที่ซับซ้อน การประกอบวงจรควบคุมทำได้ง่าย และเร็ว
ราคา	ถูก	แพง แต่ถ้าขนาดวงจรใหญ่จะคุ้มกว่าการใช้รีเลย์

ตารางที่ 2.1 แสดงการเปรียบเทียบระหว่างรีเลย์ กับ PLC

2.4 ภาษาแลดเดอร์ (Ladder)

ภาษาแลดเดอร์ประกอบด้วยสัญลักษณ์หน้าสัมผัส มีลักษณะคล้ายวงจรรีเลย์ การเขียนโปรแกรมภาษาแลดเดอร์จึงทำได้ไม่ยากนักถ้ารู้จักกับวงจรควบคุมแบบเก่าที่ใช้รีเลย์

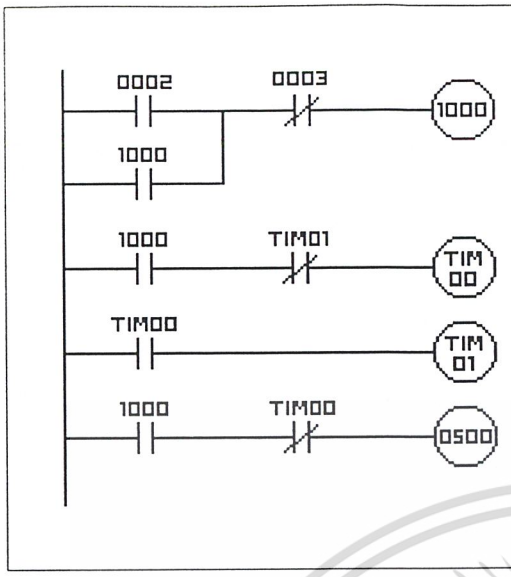
คำสั่งภาษาแลดเดอร์ประกอบด้วยสัญลักษณ์หน้าสัมผัสและขดลวด เพื่อแสดงเงื่อนไขการควบคุมระหว่างอุปกรณ์หน่วยอินพุต เอาต์พุต และอุปกรณ์ภายใน การเขียนโปรแกรมต้องระบุตำแหน่งหรือหมายเลขของอุปกรณ์เหล่านี้ให้ถูกต้องและตรงกันทุกครั้ง หมายเลขของหน้าสัมผัสและขดลวดจะสัมพันธ์กับตำแหน่งภายในหน่วยความจำ ส่วนตารางข้อมูล สัญลักษณ์หน้าสัมผัสหมายถึงการรับสภาวะของหน่วยอินพุตและเอาต์พุต และอุปกรณ์ภายในเพื่อปฏิบัติลอจิกตามเงื่อนไขควบคุม สัญลักษณ์ขดลวดหมายถึงการส่งคำสั่งหรือผลการควบคุมไปยังหน่วยเอาต์พุต และอุปกรณ์ภายใน

รีเลย์แบบปกติปิด จะทำงานเมื่อเป็นจริง หรือมีสภาวะเป็น “1” หรือมีเส้นทางผ่านหน้าสัมผัสปิดจาก ด้านซ้ายของอุปกรณ์มาที่ขดลวดเอาต์พุตและสิ้นสุดที่ปลายด้านขวาของอุปกรณ์นั้นอย่างน้อย 1 เส้นทาง

รีเลย์แบบปกติเปิด จะหยุดทำงานเมื่อเป็นเท็จ หรือมีสภาวะเป็น “0” หรือไม่มีเส้นทางเชื่อมต่อระหว่างปลายทั้งสองด้านของอุปกรณ์

ตารางที่ 2.2 แสดงคำสั่ง PLC ของ Omron รุ่น C20P (เฉพาะที่ใช้ในโครงงานนี้)

Instruction		Symbol	Mnemonic		
Load	LD		LD		Point No.
Load Not	LDI		LD	NOT	Point No.
And	AND		AND		Point No.
And Not	ANI		AND	NOT	Point No.
Or	OR		OR		Point No.
Or Not	ORI		OR	NOT	Point No.
Out	OUT		OUT		Point No.
Out Not	OUI		OUT	NOT	Point No.
Timer	TIM	TIM	TIM		Timer No. Set value
Counter	CNT	CP R CNT	CNT		Counter No. Set value
No Operation	NOP		NOP		.
End	END	END	END		.



Address	Instruction	Data
0000	LD	0002
0001	OR	1000
0002	ANI	0003
0003	OUT	1000
0004	LD	1000
0005	ANI	TIM 01
0006	TIM	00
		#0030
0007	LD	TIM 00
0008	TIM	01
		#0030
0009	LD	1000
0010	ANI	TIM 00
0011	OUT	0500
0012	END	(01)

รูปที่ 2.4 แสดงตัวอย่างของภาษาแลดเดอร์และนิวมอนิคโค้ดของแลดเดอร์ไดอะแกรม

บทที่ 3
FPGA

Field Programmable

อุปกรณ์วงจรรวม ASIC (Application Specific Integrated Circuit) แบบ Field Programmable มีอยู่มากมายหลายชนิด แต่มีลักษณะการสร้างหรือกำหนดการทำงานของวงจรที่เหมือนกัน กล่าวคือ ผู้ใช้งานสามารถออกแบบและสร้างวงจรที่ต้องการใช้ลงในตัวอุปกรณ์ได้เองโดยไม่ต้องไปโรงงานเพื่อผลิต โดยเฉพาะอย่างยิ่งในปัจจุบันนี้มีเครื่องมือที่ช่วยในการออกแบบ และสร้างวงจรร่วมกับไมโครคอมพิวเตอร์ที่มีความสามารถสูงในการพัฒนาตั้งแต่ขั้นการออกแบบ การจำลองการทำงาน จนถึงจัดสร้างวงจรลงในอุปกรณ์ รวมทั้งอุปกรณ์ Field Programmable เหล่านี้สามารถหาซื้อได้ง่ายทำให้การสร้างวงจรอิเล็กทรอนิกส์จนถึงระบบ ไมโครโพรเซสเซอร์หันมาใช้อุปกรณ์จำพวกนี้ เป็นอุปกรณ์ประกอบในวงจรแทนอุปกรณ์ย่อย ๆ แยกชิ้น (Discrete Component) มาก

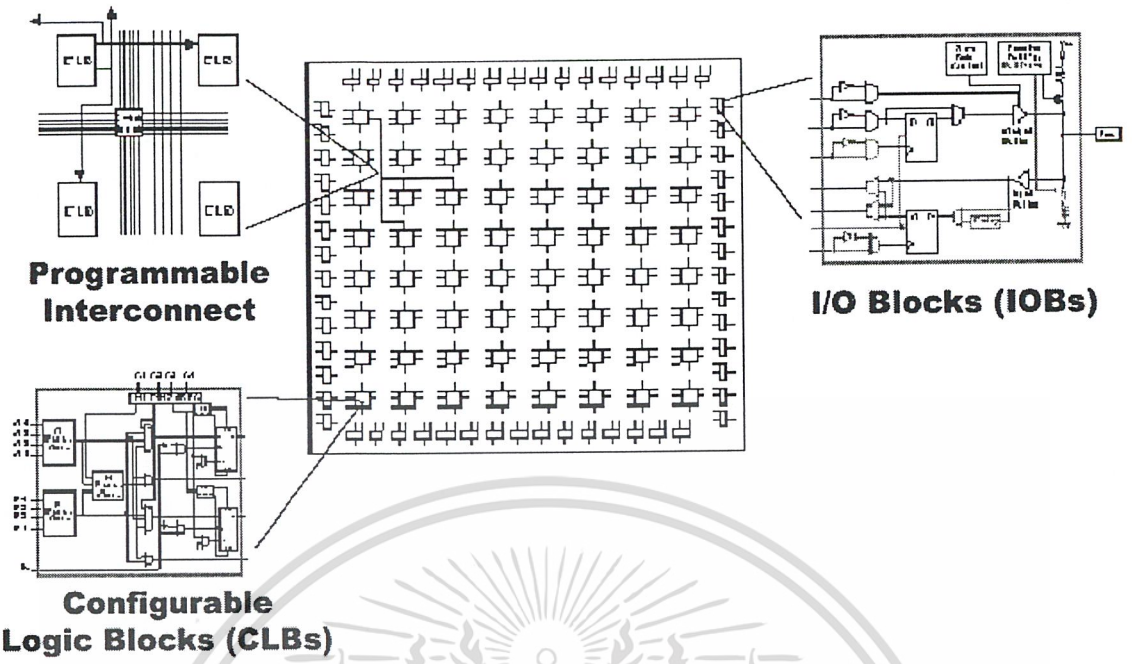
3.1 FPGA (Field Programmable Gate Array)

เป็นอุปกรณ์ที่ถูกพัฒนาต่อจากอุปกรณ์แอลซีเอของบริษัทไซลิงซ์ (XILINX Inc.) โดยมีประสิทธิภาพการทำงานและมีปริมาณความหนาแน่นของเกตสูง สามารถจะกำหนดฟังก์ชั่นของการทำงานได้ตามความต้องการของผู้ใช้โดยผ่านทางโปรแกรม เอฟพีจีเอ (FPGA) ได้รวบรวมข้อดีทั้งหมดของการทำคัสตัมวีแอลเอสไอ (Custom VLSI) มารวมไว้ทั้งหมดได้แก่ การออกแบบการผลิต และการลดเวลาที่จะส่งตัวผลิตภัณฑ์ออกตลาด ซึ่งเป็นประโยชน์ต่อการผลิตวงจรเป็นอย่างมาก นักออกแบบเพียงกำหนดฟังก์ชั่นการทำงานของวงจร ดังนั้นการออกแบบวงจรโดยใช้เอฟพีจีเอ สามารถออกแบบและทดสอบได้ภายในเวลาเพียง 2 –3 วันเท่านั้น ตรงกันข้ามกับการออกแบบโดยใช้เกตอาเรย์ (Gate Array) ซึ่งใช้เวลาหลายสัปดาห์ การเปลี่ยนแปลงแก้ไขแบบก็เช่นเดียวกัน จากประโยชน์ของเอฟพีจีเอดังกล่าวมา ทำให้เกิดการประหยัดค่าใช้จ่ายเป็นอย่างมาก เพราะได้ลดความเสี่ยงในการที่จะต้องแก้ไขวงจร การเลื่อนเวลาการออกผลิตภัณฑ์ ลดค่าเอ็นอาร์อี (NRE : Nonrecurring Engineering Cost) ลงไปด้วย

3.2 สถาปัตยกรรมภายในของเอฟพีจีเอตระกูล XC4000

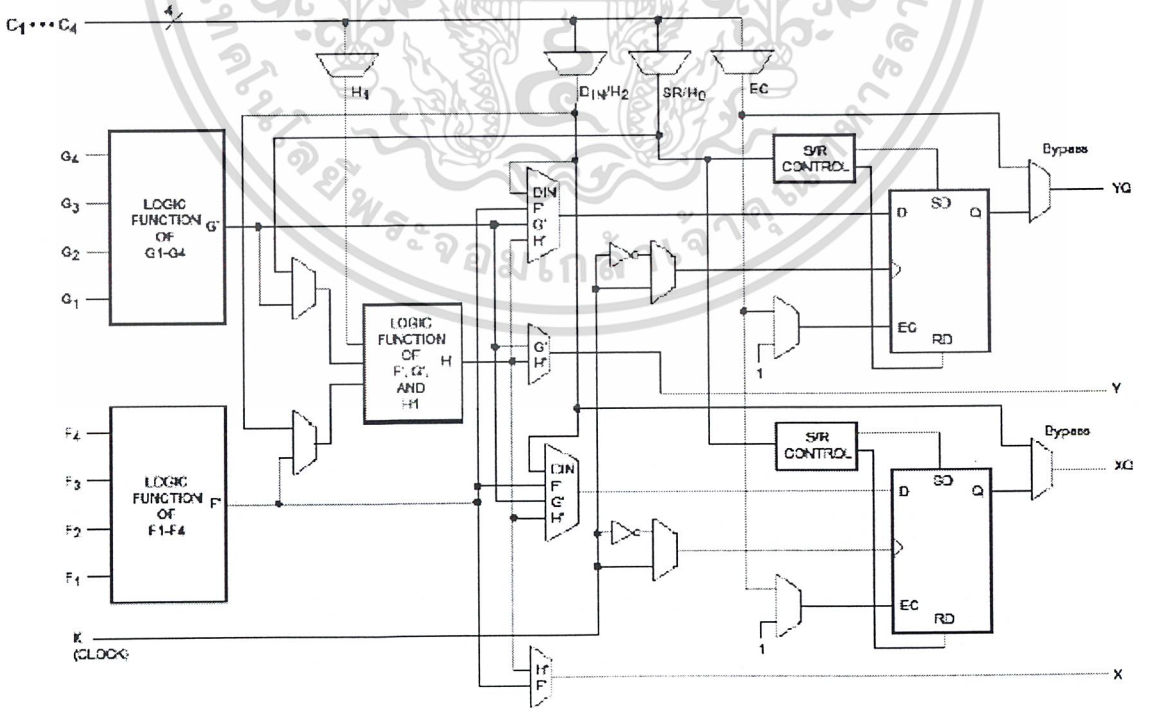
FPGA (Field Programmable Gate Array) เป็นชิพที่มีสนามวงจรขนาดใหญ่อยู่ภายใน ที่เราสามารถนำมาใช้และออกแบบวงจรต่าง ๆ ได้ตามที่เราต้องการ โดยมีโปรแกรมสำเร็จรูปที่ใช้ในการออกแบบอยู่แล้ว มีความเร็วสูง มีสถาปัตยกรรมการออกแบบคล้าย CPLD (Complex Programmable Logic Device) แต่มีส่วนประกอบที่ซับซ้อน และมีประสิทธิภาพมากกว่า ในการออกแบบวงจรสามารถทำได้ง่าย ทั้งในการเชื่อมต่อ และการแก้ไข ดังนั้นเอฟพีจีเอ จึงเหมาะสมสำหรับการออกแบบวงจรเป็นอย่างมาก สถาปัตยกรรมภายในของเอฟพีจีเอ แบ่งเป็น 3 ส่วน คือ

- CLB (Configuration Logic Block)
- IOB (Input Output Block)
- Interconnect



รูปที่ 3.1 แสดงวงจรพื้นฐานภายใน FPGA

ซึ่งภายในสถาปัตยกรรมเอฟพีจีเอจะมีลักษณะเป็นตารางของลอจิกบล็อก (Logic Block) และล้อมรอบไปด้วยบล็อกของไอโอ (I/O Blocks) การเชื่อมต่อระหว่างซีแอลบี (CLB: Configuration Logic Block) ทำได้โดยผ่านช่องว่างระหว่างแถว (Row) และ คอลัมน์ (Column)



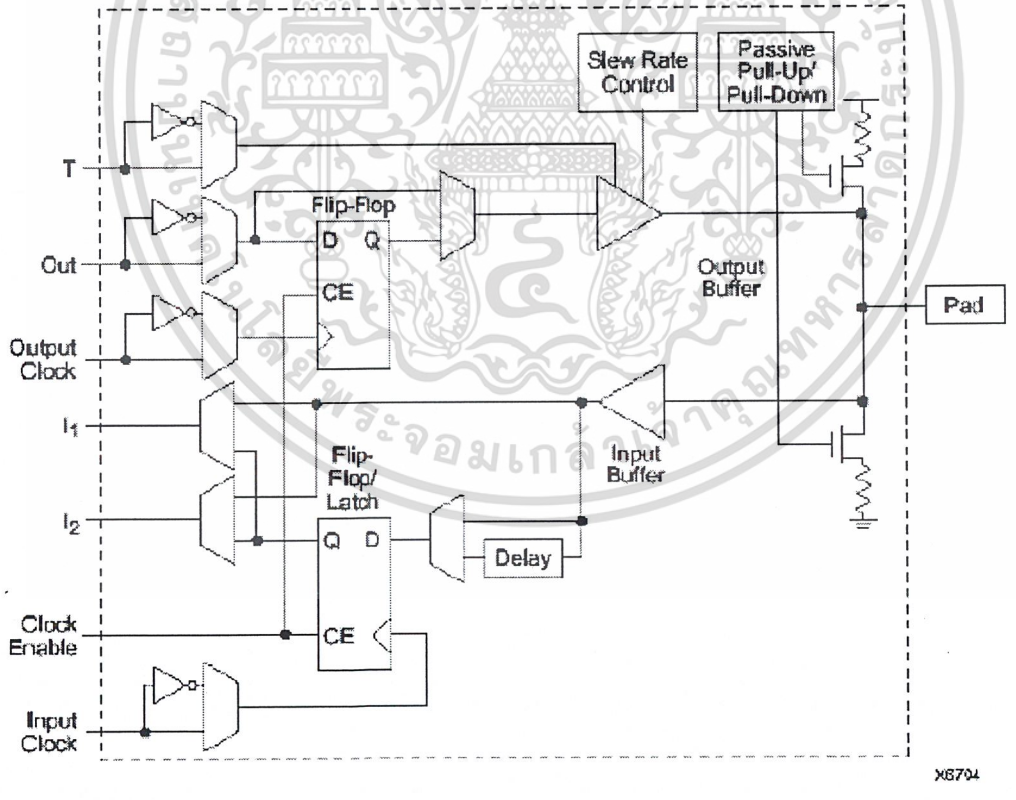
รูปที่ 3.2 แสดงผังวงจรภายในของซีแอลบีของเอฟพีจีเอตระกูล XC4000

ตัวแอลซีเอ (LCA: Logic Cell Array) จะทำงานได้ต้องใช้ Program-driven logic device ซึ่งจะเป็นตัวระบุหน้าที่ของตัวซีแอลบี, ไอโอบี และการเชื่อมต่อภายใน (Interconnection) แต่ละตัว ค่าต่าง ๆ จะถูกกำหนดไว้ในโปรแกรมคอนฟิกกูเรชัน (Configuration program) หรือเก็บไว้ในอีพรอม (EPROM) ภายในแอลซีเอ แอลซีเอจะเริ่มต้นทำงานได้โดยตัวโปรแกรมจะถูกโหลดเข้าสู่แอลซีเอ

ประสิทธิภาพของแอลซีเอจะถูกกำหนดโดย ความเร็วของลอจิก ส่วนประกอบหน่วยความจำและการโปรแกรมการเชื่อมต่อต่าง ๆ ส่วนความเร็วของอัตราของระบบสัญญาณนาฬิกา (System clock rate) จะถูกกำหนดด้วย ทอกเกิลฟลิปฟลอป (Toggle flipflop) สำหรับการประยุกต์ใช้โดยทั่วไปจะตั้งค่าอยู่ที่ประมาณ 1/3 ถึง 1/2 ของค่าสูงสุดของทอกเกิลเกต (Maximum toggle gate)

3.2.1 CLB (Configuration Logic Block)

ภายใน LCA (Logic Cell Array) คือเมทริกซ์ของซีแอลบี แต่ละตัวจะประกอบด้วยหน่วยของคอมบินิชั่นลอจิกที่สามารถโปรแกรมได้ (Programmable combination logic) และส่วนของริจิสเตอร์เก็บข้อมูล (Storage register) ส่วนของวงจรคอมบินิชั่นลอจิกสามารถใช้สร้างวงจรทางด้านฟังก์ชั่นบูลีนของอินพุต ส่วนริจิสเตอร์รับค่าจากส่วนคอมบิเนชั่น หรือรับค่าโดยตรงจากเอาต์พุตของซีแอลบี



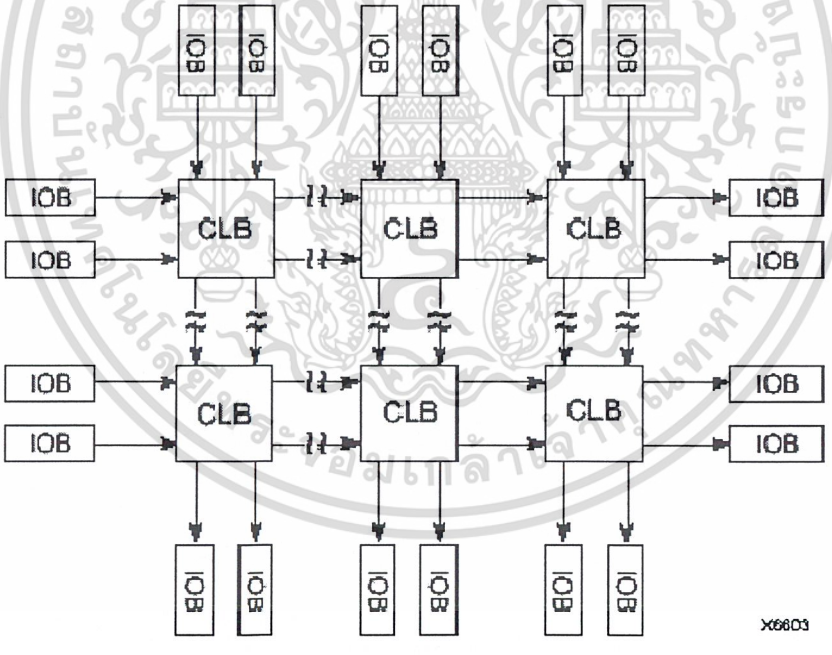
รูปที่ 3.3 แสดงผังวงจรภายในของไอโอบีของเอฟพีจีเอตระกูล XC4000

3.2.2 IOB (Input Output Block)

เป็นส่วนติดต่อกับวงจรภายนอกของแอลซีเอ สร้างมาจากส่วนของอุปกรณ์อินพุต และเอาต์พุตที่สามารถโปรแกรมได้ (Programmable Input/Output device) แต่ละตัวสามารถโปรแกรมได้อย่างอิสระ โดยจะให้เป็นอินพุต และเอาต์พุตแบบ 3 สถานะ หรือไอโอ (I/O: Input/Output) แบบสองทิศทางก็ได้ โดยอินพุตสามารถโปรแกรมให้รู้จักทั้งระดับสัญญาณทีทีแอล และซีมอสเทรดโฮลของไอโอบี แต่ละตัวมีฟลิปฟลอปสามารถใช้เป็นบัฟเฟอร์สำหรับอินพุต และเอาต์พุต

3.2.3 Interconnect

ความยืดหยุ่นของการใช้แอลซีเอมาทำเป็นอุปกรณ์ขึ้นอยู่กับการโปรแกรม ทรัพยากรต่าง ๆ ที่อยู่ภายในประกอบเข้าด้วยกัน การที่จะควบคุมการเชื่อมต่อระหว่างจุดสองจุดภายในชิปซึ่งเหมือนกับเกตอาเรย์ทั่ว ๆ ไป การเชื่อมต่อภายในแอลซีเอประกอบด้วยเน็ตเวิร์ค 2 ทิศทางคือทางแถว และทางคอลัมน์ ซึ่งวางอยู่ระหว่าง CLB programmable switch และจะทำการเชื่อมต่ออินพุต และเอาต์พุตของไอโอบี และซีแอลบีที่จุดต่อร่วมระหว่างแถวกับคอลัมน์ซึ่งยังสามารถสลับสัญญาณจากเส้นทางไปยังส่วนต่าง ๆ ได้



รูปที่ 3.4 แสดงการ Interconnect ของ เอฟพีจีเอตระกูล XC4000

3.3 คุณสมบัติโดยทั่ว ๆ ไปของเอฟพีจีเอตระกูล XC4000

1. เป็นอุปกรณ์รุ่นที่สามของเอฟพีจีเอ
   - มีฟลิปฟลอปจำนวนมาก
   - ในการผลิตฟังก์ชันของการทำงานมีความยืดหยุ่นสูง

- มีจำนวนเกตภายในจำนวน 2,000 – 10,000 เกต
- เพิ่มความสามารถพิเศษของริจิสเตอร์ และอินพุต / เอาต์พุต
- มีค่าแฟนเอาต์ (fan-out) สูง
- มีบัสภายใน 3 สถานะ
- สามารถทำงานได้กับสัญญาณทีทีแอล และซีมอส
- มีออสซิเลเตอร์แอมปลิฟลายเออร์ภายใน
- มีแรมความเร็วสูงภายใน (< 25 ns)
- ใช้กับงานที่ต้องการความเร็วสูง (ใช้งานได้ที่ความถี่ 70 / 100 / 125 MHz)
- มี Wide edge decoder
- เส้นทางการเชื่อมต่อ (Interconnect line) เป็นแบบลำดับชั้น
- มีการกระจายกำลังงานของสัญญาณต่ำ

2. มีสถาปัตยกรรมภายในที่ยืดหยุ่น
   - มีลอจิกบล็อคและไอโอบล็อคที่สามารถโปรแกรมได้
   - มีอินเตอร์คอนเน็ต และ Wide decoder ที่โปรแกรมได้
3. ทำกระบวนการซับไมครอนชนิดซีมอสได้
   - มีลอจิกและอินเตอร์คอนเน็คที่มีความเร็วสูง
   - ใช้พลังงานต่ำ
4. คุณลักษณะทาง System-Oriented
   - รองรับมาตรฐาน IEEE 1149.1 ในการทำ boundary-scan logic
   - สามารถโปรแกรมค่า Output slew rate ได้
   - สามารถโปรแกรมอินพุตมีลักษณะพูลอัพ (Pull-up) หรือ พูลดาวน์ (Pull-down) รีจิสเตอร์ได้
   - ให้กระแสเอาต์พุตได้ตั้งแต่ 12 – 24 มิลลิแอมป์ (ขึ้นอยู่แต่ละรุ่น)
5. สามารถทำการโหลดเอาแฟ้มข้อมูลประเภทไบนารีได้
   - ไม่จำกัดจำนวนครั้งในการโปรแกรมซ้ำ
   - มีโหมดในการโปรแกรมให้เลือก 6 โหมด

6. มีโปรแกรมช่วยพัฒนาได้แก่ XACT Development System (ปัจจุบัน Foundation series) ที่ทำงานบนคอมพิวเตอร์รุ่นต่าง ๆ เช่น 486 / Pentiums, NEC PC, Apollo, Sun-4, HP700
   - สามารถติดต่อกับโปรแกรมอื่น ๆ ได้ เช่น Viewlogic, Mentor Graphic และ OrCAD เป็นต้น
   - มีโปรแกรมการวางและเชื่อมโยงอุปกรณ์ภายในแบบอัตโนมัติ (Automatic place and routing) ที่ครบสมบูรณ์
   - มี Interactive Design Editor ที่ใช้สำหรับการทำ Optimization
   - มี 288 มาโคร 34 ฮาร์ดมาโคร และ แรม / รอมคอมไพล์เลอร์

ตารางที่ 3.1 แสดงรายละเอียดของอุปกรณ์ภายในเอฟพีจีเอตระกูล XC4000

Device	Max Logic Gate (No RAM)	Max RAM Bits (No Logic)	Typical Gate Range (Logic and RAM)*	CLB Matrix	Total Logic Blocks	Number of Flip-Flops	Max Decode Input per Side	Max User I/O
XC4003E	3,000	3,200	2,000 – 5,000	10x10	100	360	30	80
XC4005E/L	5,000	6,272	3,000 – 9,000	14x14	196	616	42	112
XC4006E	6,000	8,192	4,000 – 12,000	16x16	256	768	48	128
XC4008E	8,000	10,368	6,000 – 15,000	18x18	324	936	54	144
XC4010E/L	10,000	12,800	7,000 – 20,000	20x20	400	1,120	60	100
XC4013E/L	13,000	18,432	10,000 – 30,000	24x24	576	15,36	75	192
XC4020E	20,000	25,088	13,000 – 40,000	28x28	784	2,016	84	224
XC4025E	25,000	32,768	15,000 – 45,000	32x32	1,024	2,560	96	256
XC4028EX/XL	28,000	32,768	18,000 – 50,000	32x32	1,024	2,560	96	256
XC4036EX/XL	36,000	41,472	22,000 – 65,000	36x36	1,296	3,168	106	288
XC4044EX/XL	44,000	51,200	27,000 – 80,000	40x40	1,600	3,840	120	320
XC4052XL	52,000	61,952	33,000 – 100,000	44x44	1,936	4,576	132	352
XC4062XL	62,000	73,728	40,000 – 130,000	48x48	2,304	5,376	144	384
	Larger Devices Available in the First Half of 1997							

* Max value of Typical Gate Range Include 20 – 30% of CLBs used as RAM

3.4 การโปรแกรมเอฟพีจีเอ ตระกูล XC4000

คือกระบวนการในการโหลดข้อมูลในโปรแกรมไปยังแอลซีเอ เพื่อกำหนดหน้าที่ของการทำงานในแต่ละบล็อคภายใน และการเชื่อมต่อต่าง ๆ ซึ่งเอฟพีจีเอตระกูล XC4000 จะต้องใช้ข้อมูลเกี่ยวกับการโปรแกรมประมาณ 350 บิตต่อซีแอลบี โดยแต่ละบิตจะบอกถึงสถานะของหน่วยความจำสเตติกที่ควบคุมในการควบคุม ตารางฟังก์ชั่น (Function table bit) และลักษณะการมัลติเพลกซ์อินพุต หรือการเชื่อมต่อกันระหว่างทรานซิสเตอร์

3.4.1 โหมดการโปรแกรม

เอฟพีจีเอในตระกูล XC4000 มีโหมดการโปรแกรม 6 โหมด ซึ่งแต่ละโหมดจะถูกกำหนดโดยจากบิตโหมด ได้แก่ บิต M0, M1 และ M2 โดยมีโหมดการโปรแกรมดังตารางที่ 3.2

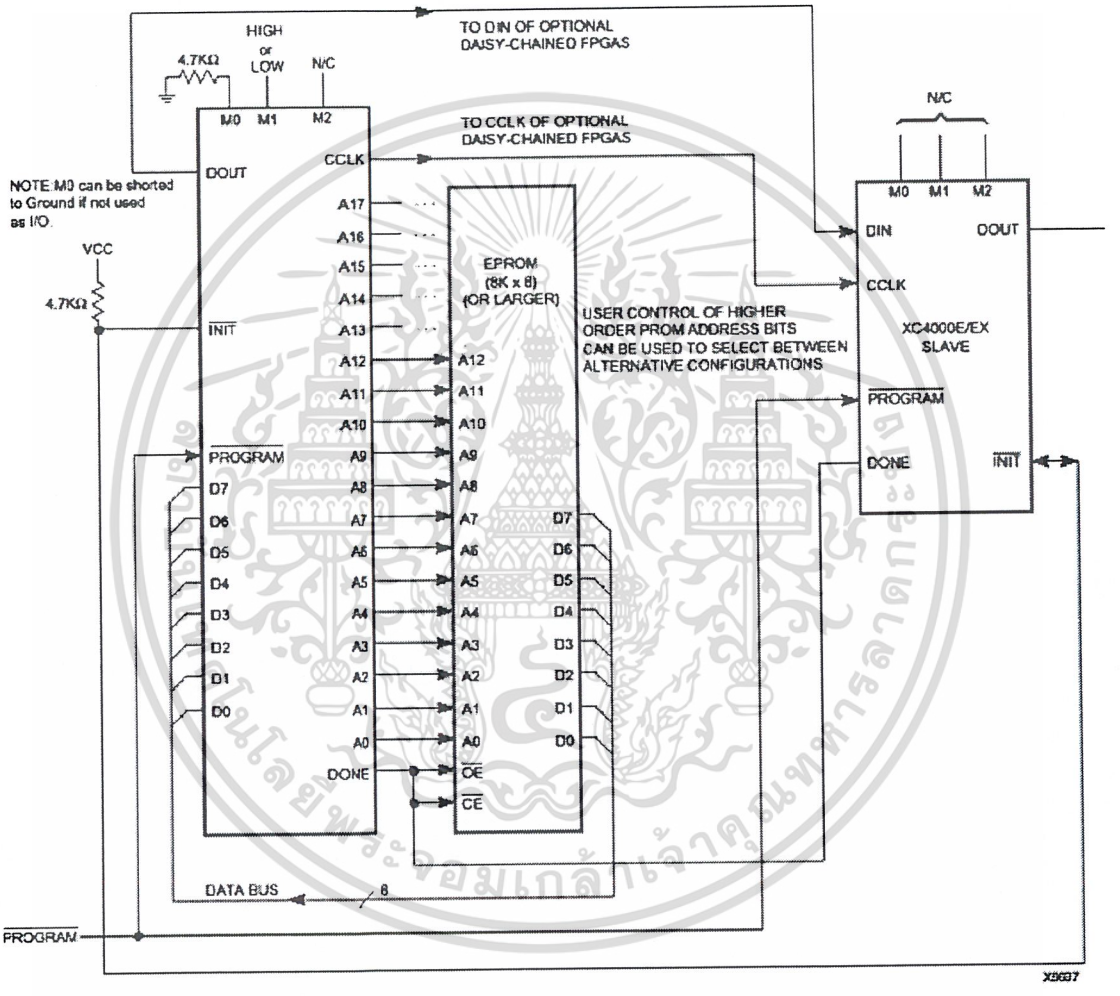
ตารางที่ 3.2 แสดงรูปแบบของโหมดต่าง ๆ ในการโปรแกรมเอฟพีจีเอตระกูล XC4000

Mode	M2	M1	M0	Clock	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
Master Parallel down	1	1	0	Output	Byte-Wide, 3FFFF
Peripheral Synch.	0	1	1	Input	Byte-Wide
Peripheral Asynch.	1	0	1	Output	Byte-Wide

3.4.1.1 โหมดหลัก (Master modes)

ในการทำงานในโหมดนี้ตัวแอลซีเอจะถูกโหลดค่าโครงสร้างที่กำหนดไว้ (Configuration data) จากหน่วยความจำภายนอกเข้ามาโดยอัตโนมัติ มีโหมดที่แตกต่างกัน 8 โหมดโดยใช้ สัญญาณเวลาภายในจ่ายให้ซีคล็อก (CCLK: Configuration Clock) เพื่อที่จะเป็นฐานเวลาในการนำข้อมูลที่เข้ามาทางโหมด

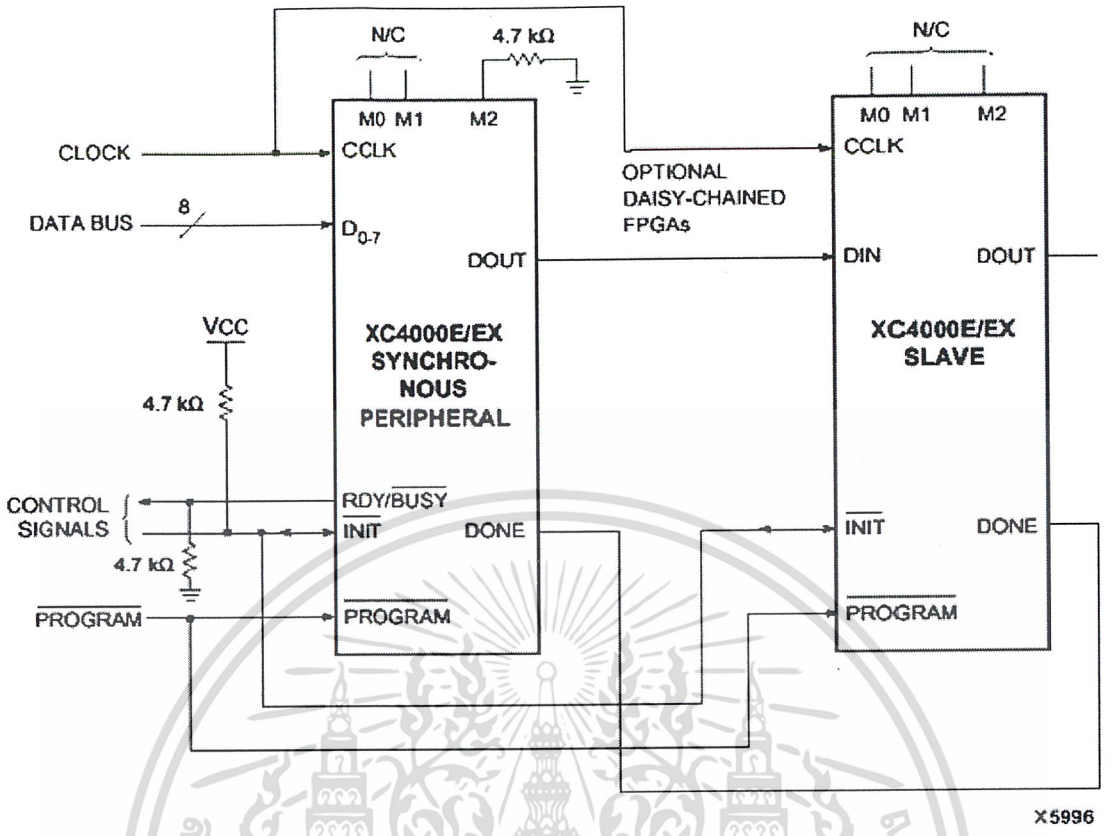
หลักแบบอนุกรม (Serial master mode) และรับค่าโครงสร้างที่กำหนดไว้เข้ามาทางขาสัญญาณดีอิน (DIN: Data in) จากแหล่งสัญญาณซิงโครนัสเช่น Xilinx serial configuration PROM, Parallel master low and Master high mode โดยจะรับข้อมูลแบบขนานมาจากบิต D0 – D7 โดยสัมพันธ์กับแอดเดรสที่กำหนดโดยแอลซีเอ จะเป็นการแสดงตัวอย่างการเชื่อมต่อในโหมดหลักแบบขนาน (Master Parallel mode) โดยเริ่มต้นที่แอดเดรส 0000 ไบต์ข้อมูล (Data byte) จะถูกอ่านเข้ามาแบบขนานทุก ๆ สัญญาณอาร์คล็อก (RCLK: Read Clock) และส่งเข้าไปภายในแบบอนุกรมโดยอาศัยสัญญาณนาฬิกาโครงสร้างที่กำหนดไว้ (Configuration Clock)



รูปที่ 3.5 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดหลักแบบขนาน

3.4.1.2 เพอริเฟอรัลโหมด (Peripheral modes)

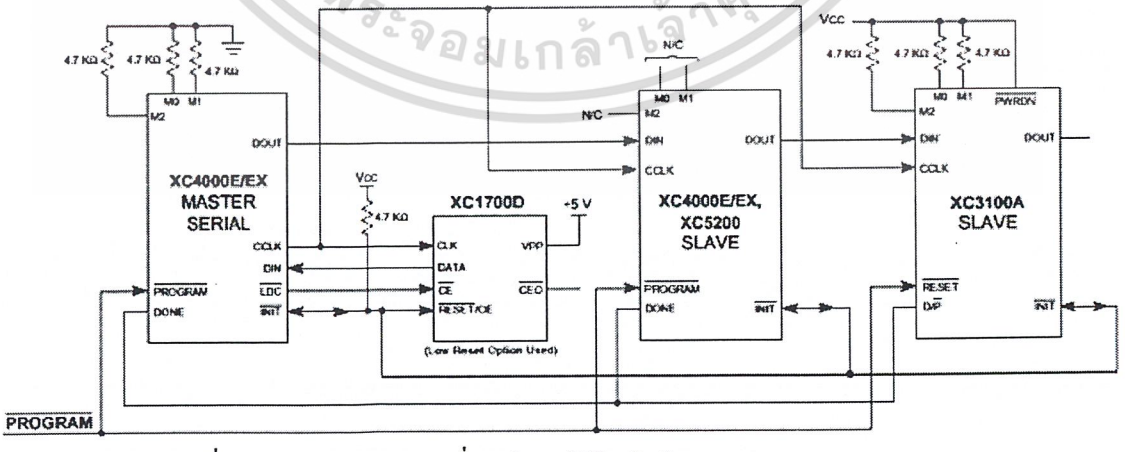
โหมดนี้จะรับข้อมูล สถานะพร้อม (Ready) / กำลังทำงาน (Busy) เป็นไบต์จากบัส เพื่อทำให้สัญญาณต่าง ๆ สื่อสารกันได้มีประสิทธิภาพมากขึ้น ในอะซิงโครนัสโหมดนั้นออสซิเลเตอร์ภายในจะสร้างสัญญาณ CCLK เพื่อรับข้อมูลที่เป็นไบต์ให้อยู่ในรูปแบบที่เหมาะสม และในซิงโครนัสโหมดนั้นสัญญาณนาฬิกาภายนอกจะเป็นตัวควบคุมการรับข้อมูลที่เป็นไบต์



รูปที่ 3.6 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดเพอริเฟอรัลแบบอะซิงโครนัส (Asynchronous Peripheral mode)

3.4.1.3 โหมดรองแบบอนุกรม (Slave Serial modes)

ในโหมดนี้แอลซีเอจะรับค่าโครงสร้างที่กำหนดไว้ที่เป็นแบบอนุกรมในขาขึ้นของสัญญาณ CCLK และหลังจากรับข้อมูลมาแล้วจะส่งข้อมูลเพิ่มเติมออกไปด้วย และแอลซีเอก็จะถูกทำการซิงโครนัสในขาลงถัดไปของสัญญาณ CCLK



รูปที่ 3.7 แสดงผังวงจรการเชื่อมต่อเอฟพีจีเอในโหมดรองแบบอนุกรม

3.5 ความสามารถของแรมในเอฟพีจีเอตระกูล XC4000

แอลซีเอทำงานโดยใช้ตารางการค้นหา (Loop-up table) ซึ่งจะทำการเก็บตารางที่ว่านี้ในสแตติกแรม ซึ่งจะถูกเขียนในระหว่างการโปรแกรมโครงสร้างที่กำหนดลงบนแอลซีเอ และจะถูกอ่านในการทำงาน ดังนั้นแรมภายในจึงควรถูกรวมไว้ในการออกแบบของผู้ใช้ด้วย

หน้าที่ของแรมในเอฟพีจีเอตระกูล XC4000 มีหน้าที่คล้ายแรมโดยทั่ว ๆ ไป เช่น ฟีโฟ (FIFO: First In First Out) ลิโฟ (LIFO: Last In First Out) รีจิสเตอร์ไฟล์ รวมทั้งแอปพลิเคชั่นบางอย่าง เช่น รีจิสเตอร์เลื่อนข้อมูล (Shift register) เป็นต้น แรมของเอฟพีจีเอตระกูล XC4000 มีความเร็วสูงเสมือนเอชแรม (SRAM) จึงไม่จำเป็นต้องคำนึงถึงเวลาหน่วงของการเชื่อมต่อ (Interconnection delay)

ตารางที่ 3.3 จำนวนของแรมภายในเอฟพีจีเอตระกูล XC4000

RAM Module	Equivalent Logic	XC4003	XC4005	XC4010
16x1	4 – Input Function Generators (F or G)	200	392	800
32x1	Two – 4 – Input Function Generators and One 3 – Input Function Generators (F + G + H)	100	196	400

3.6 โครงสร้างและการเชื่อมต่อของอุปกรณ์บนบอร์ด XS40

บอร์ดที่เลือกใช้ในการทำโครงงานนี้คือบอร์ด XS40 ซึ่งเป็นบอร์ดของบริษัท XESS Corp และตัว FPGA ที่ใช้งานคือ FPGA ของบริษัท Xilinx เบอร์ XC40-010E ในหัวข้อนี้จะกล่าวถึงโครงสร้างของบอร์ด XS40 ที่ใช้ในการทำโครงงานเพื่อให้เข้าใจถึงการนำ FPGA มาใช้งานและเข้าใจถึงการทำงานของบอร์ดที่นำมาใช้งาน

การทำงานของบอร์ด XS40 จะเป็นการทำงานร่วมกันระหว่างตัว FPGA และไมโครคอนโทลเลอร์ โดยอุปกรณ์ทั้งสองตัวจะถูกต่อกันอยู่บนบอร์ดของ XS40 เพื่อให้การใช้งานบอร์ดทำได้โดยง่าย ซึ่งก็เป็นการจำกัดการใช้งานของไมโครคอนโทลเลอร์ด้วยเช่นกัน

สัญญาณนาฬิกาที่อยู่ภายในบอร์ด XS40 จะถูกสร้างโดยตัวกำเนิดสัญญาณแบบตั้งค่าได้ (Programmable oscillator) โดยสัญญาณนาฬิกาที่ได้จะถูกนำไปใช้กับ FPGA และ FPGA จะทำการสร้างสัญญาณนาฬิกาใหม่เพื่อส่งให้กับไมโครคอนโทลเลอร์อีกทีหนึ่ง (XTAL1 clock input)

ไมโครคอนโทลเลอร์จะทำหน้าที่ในการเลือกระหว่าง 8 บิตล่างของแอดเดรสของหน่วยความจำกับ 8 บิตของข้อมูลในการส่งออกไปยังพอร์ต P0 ซึ่งทำการเชื่อมต่อกับ SRAM และ FPGA เพื่อนำข้อมูลเข้าและออกจาก SRAM ส่วน FPGA จะถูกตั้งค่าให้เก็บแอดเดรสจาก P0 โดยสัญญาณควบคุม ALE และส่งค่าแอดเดรสที่ได้ไปยัง 8 บิตล่างของ SRAM

แอดเดรส 8 บิตบนของพอร์ต P2 ของไมโครคอนโทลเลอร์ จะถูกใช้โดย SRAM ขนาด 32 Kbytes บนบอร์ด XS40 จะใช้ 7 บิตล่างของแอดเดรสนี้ และบอร์ด XS40+ จะใช้ทั้ง 8 บิต ส่วน FPGA จะใช้แอดเดรส 8 บิตบนนี้ในการตัดสินใจสร้างสัญญาณ PSENB เพื่อควบคุม SRAM โดยจะทำงานควบคู่กับไมโครคอนโทลเลอร์ซึ่งเป็นตัวสร้างสัญญาณ CEB และ OEB ที่ควบคุมการอ่านและเขียนของ SRAM ในการทำงานในการควบคุม SRAM นี้ถ้าสัญญาณ CEB และ OEB ถูกทำให้เป็น High จะเป็นการปิดการทำงานของ SRAM เพื่อป้องกันผลกระทบกับส่วนอื่น ๆ ของวงจรบนบอร์ด XS40

FPGA ยังสร้างสัญญาณควบคุมการรีเซทของไมโครคอนโทลเลอร์ โดยเมื่อต้องการรีเซทจะทำการส่งสัญญาณ RST ให้เป็น High เพื่อป้องกันผลกระทบกับส่วนอื่นของวงจรบนบอร์ด XS40

ขาอินพุตและเอาต์พุตของพอร์ต P1 และ P3 ของไมโครคอนโทลเลอร์จะมีการเชื่อมต่อกับ FPGA เพื่อใช้ในการควบคุมการทำงานของฟังก์ชั่นบางอย่าง ได้แก่ Serial transmitters, receiver, interrupt inputs, timer inputs และ external SRAM read/write ซึ่งในการใช้งานบอร์ด XS40 ถ้าการทำงานของฟังก์ชั่นใดไม่ได้ใช้งานเราสามารถที่จะนำขาอินพุตและเอาต์พุตเหล่านี้มาใช้งานได้ ซึ่งในการนำขาอินพุตและเอาต์พุตเหล่านี้มาใช้งานจำเป็นต้องเขียนดีไซน์ใน FPGA ให้ตำแหน่งขาเหล่านี้เป็น Tristate

Seven-segment LED บนบอร์ด XS40 จะถูกเชื่อมต่อกับ FPGA โดยตรง (ในขาสัญญาณเดียวกันนี้ถูกใช้ในการส่งสัญญาณออกยังจอมอนิเตอร์ด้วย) เนื่องจาก FPGA สามารถที่จะโปรแกรมได้ดังนั้นไมโครคอนโทลเลอร์ก็สามารถที่จะนำควบคุม LED ได้ผ่านพอร์ต P1 หรือ P3 โดยการใช้วิธีตั้งค่าตำแหน่งของ LED เข้ากับตำแหน่งของหน่วยความจำที่ติดต่อกับไมโครคอนโทลเลอร์ได้ (Memory-mapping)

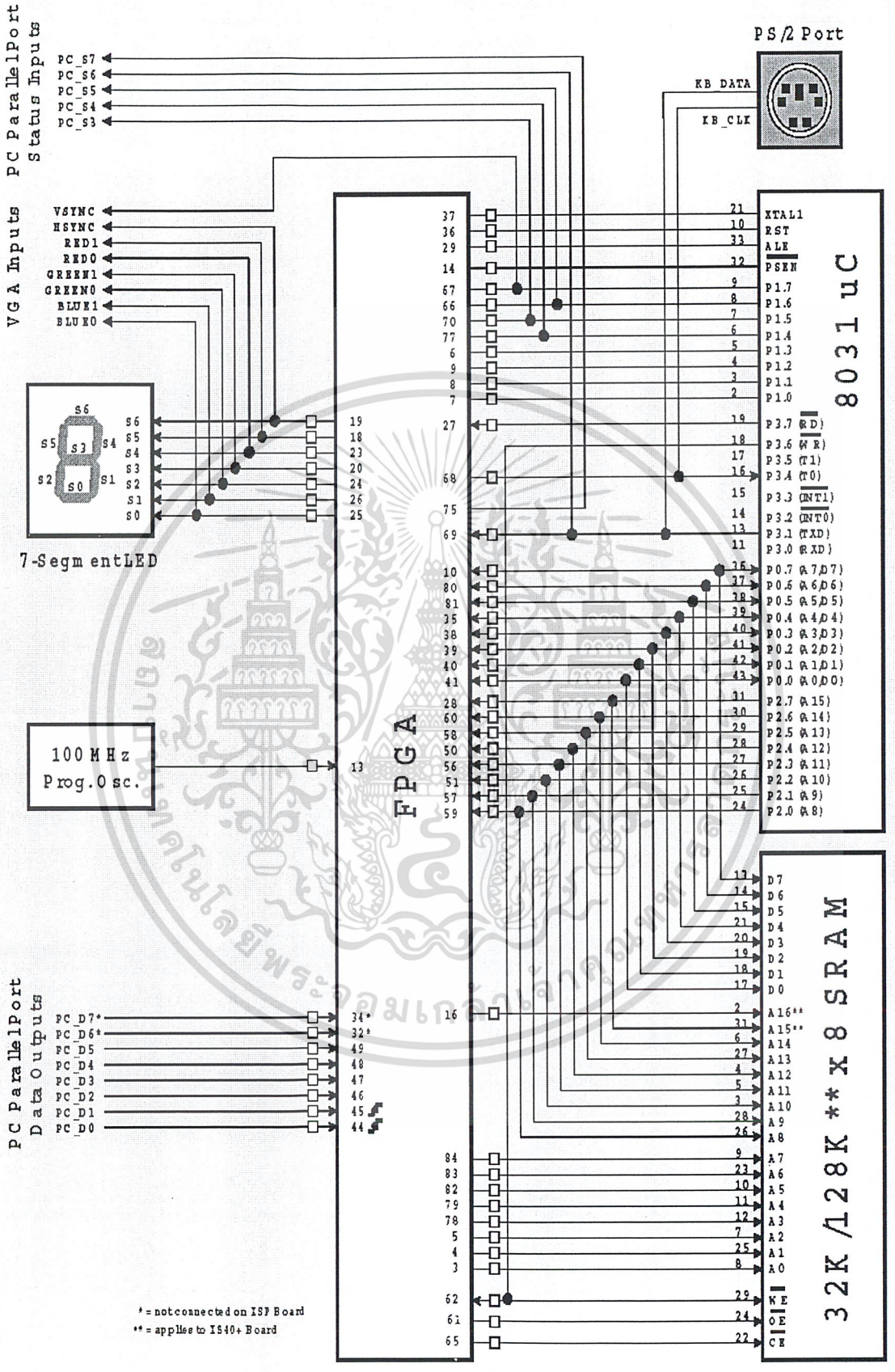
เราสามารถใช้เครื่องคอมพิวเตอร์ส่งข้อมูลมายังบอร์ด XS40 ได้ทางข้อมูล 8 บิตจากพอร์ตขนาน ซึ่งจะทำการต่อตรงกับ FPGA และไมโครคอนโทลเลอร์ก็สามารถนำสัญญาณเหล่านี้ไปใช้ได้เช่นกัน โดยต้องทำการโปรแกรมให้ FPGA ส่งค่าผ่านไปยังไมโครคอนโทลเลอร์

การส่งข้อมูลกลับจากบอร์ด XS40 ไปยังเครื่องตอมพิวเตอร์สามารถทำได้โดยผ่านทางพอร์ตขนานเช่นกัน ขาสัญญาณที่บอกสถานะของพอร์ตขนานจะถูกต่อเข้ากับขา P1และ P3 ของไมโครคอนโทลเลอร์ ซึ่งหมายความว่าเราสามารถโปรแกรมให้ FPGA ส่งสัญญาณกลับไปได้เช่นกัน

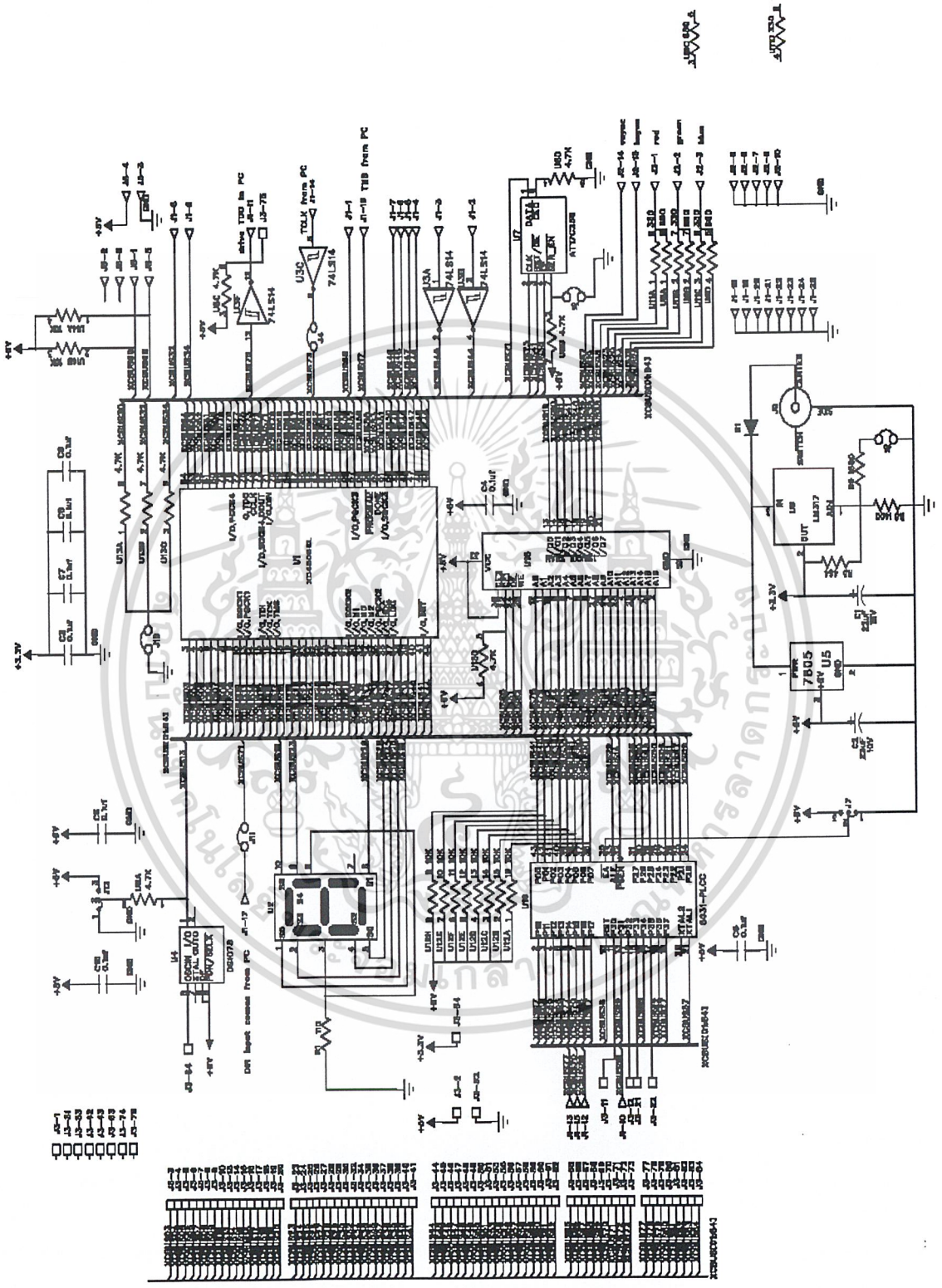
FPGA ยังสามารถที่จะรับค่าจากคีย์บอร์ด และเมาส์ได้โดยผ่านทางพอร์ต PS/2 ของบอร์ด XS40 เช่นกัน

ตารางที่ 3.4 แสดงเบอร์และการใช้งานของพินบนบอร์ด XS40

XS40 Pin	Connects to...	Description
25	S0 BLUE0	These pins drive the individual segments of the LED display (S0 – S6). They also drive the color and horizontal sync signals for a VGA monitors.
26	S1 BLUE1	
24	S2 GREEN0	
20	S3 GREEN1	
23	S4 RED0	
18	S5 RED1	
19	S6 HSYNCB	
13	CLK	An input driven by the 100 MHz programmable oscillator.
44	PC D0	These pins are driven by the data output pins of the PC parallel port. Clocking signals can only be reliably applied through pins 44 and 45 since these have additional hysterisis circuitry. Pins 32 and 34 are mode signals for the FPGA so you must adjust your design to account for the way that the Foundation tools handle these pins. Pins 32 and 34 are not usable as general-purpose I/O on the Spartan FPGA on the XSP Board.
45	PC D1	
46	PC D2	
47	PC D3	
48	PC D4	
49	PC D5	
32	PC D6	
34	PC D7	
37	XTAL1	Pin that drives the uC clock input.
36	RST	Pin that drives the uC reset input.
29	ALER	Pin that monitors the uC address latch enable.
14	PSENR	Pin that monitors the uC program store enable.
7	P1.0	These pins connect to the pins of Port 1 of the uC. Some of the pins are also connected to the status input pins of the PC parallel port. Pin 67 drives the vertical sync signal for a VGA monitor.
8	P1.1	
9	P1.2	
6	P1.3	
77	P1.4 PC S4	
70	P1.5 PC S3	
66	P1.6 PC S5	
67	P1.7 VSYNCB	
69	P3.1 (TXD) PC S6	These pins connect of some of the pins of Port 3 of the uC. The uC has specialized functions for each of the port pins indicated in parentheses. Pin 62 connects to the data write pin of the uC and the write-enable pin of the SRAM. Pin 69 connects to a status input pin of the PC parallel port and the PS/2 data line. Pin 68 connects to the PS/2 clock line.
68	P3.4 (T0) PS/2 CLK	
62	P3.5 (WRR) WER	
27	P3.7 (RDR)	
41	P0.0 (AD0) D0	These pins connect to Port 0 of the uC which is also a multiplexed address/data port. These pins also connect to the data pins of the SRAM
40	P0.1 (AD1) D1	
39	P0.1 (AD2) D2	
38	P0.3 (AD3) D3	
35	P0.4 (AD4) D4	
81	P0.5 (AD5) D5	
80	P0.6 (AD6) D6	
10	P0.7 (AD7) D7	
59	P2.0 (A8) A8	These pins connect to Port 2 of the uC which also outputs the upper address byte. These pins also connect to the upper address bits of the SRAM. Pins 28 and 16 are connected to the 128 KB SRAM address pins only on the XS40+ Board. Pins 28 and 16 do not connect to the 32 KB SRAM on the XS40 Board.
57	P2.1 (A9) A9	
51	P2.2 (A10) A10	
56	P2.3 (A11) A11	
50	P2.4 (A12) A12	
58	P2.5 (A13) A13	
60	P2.6 (A14) A14	
28	P2.7 (A15) A15	
16	A16	
3	A0	These pins drive the 8 lower address bits of the SRAM.
4	A1	
5	A2	
78	A3	
79	A4	
82	A5	
83	A6	
84	A7	
61	OEB	Pin that drives the SRAM output enable.
65	CEB	Pin that drives the SRAM chip enable
75	PC S7	Pin that drives a status input pin of the PC parallel port.



รูปที่ 3.8 แสดงผังการเชื่อมต่อของอุปกรณ์ต่าง ๆ บนบอร์ด XS40



รูปที่ 3.9 แสดงผังวงจร (Schematics) ของบอร์ด XS40

บทที่ 4
ภาษาวีเอชดีแอล (VHDL)

วีเอชดีแอล (VHDL) ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit) ซึ่งเป็นภาษาที่ใช้อธิบายการทำงานของวงจรดิจิตอลฮาร์ดแวร์ ซึ่งสามารถใช้อธิบายฟังก์ชันการทำงานได้หลาย ๆ ระดับตั้งแต่ระดับบล็อก จนถึงระดับเกต ความซับซ้อนของระบบสามารถจะเขียนได้ตั้งแต่ระดับเกต ประกอบกันจนเป็นระบบที่สมบูรณ์ รูปแบบของภาษา VHDL นั้น จะประกอบไปด้วย 2 ส่วนใหญ่ ๆ ได้แก่ ส่วนของภาษาซีเควนเชียล (Sequential Language) และภาษาคอนเคอร์เรนท์ (Concurrent Language) การโปรแกรมด้วยภาษา VHDL สามารถเขียนได้ทั้ง 2 รูปแบบรวมกัน เพราะในการทำงานของระบบใด ๆ ย่อมจะมีการทำงานในแบบซีเควนเชียล และคอนเคอร์เรนท์อยู่ร่วมกัน และนอกจากนี้ตัวภาษา VHDL ยังสามารถอธิบายถึงการเชื่อมต่อระหว่างระบบย่อย ๆ เข้าไว้ด้วยกันเพื่อให้เป็นระบบใหญ่ได้ ตัวภาษา VHDL นอกจากจะกำหนดรูปแบบไวยากรณ์ (Syntax) ของตัวภาษาแล้ว ยังมีการตรวจสอบความหมายของตัวภาษาว่าจะซิมูเลชั่น (Simulation) ได้หรือไม่ เพราะโปรแกรมที่เขียนโดย VHDL ต้องผ่านการซิมูเลชั่น เพื่อตรวจดูการทำงาน ฉะนั้นในการคอมไพล์ (Compile) จะมีการตรวจสอบทั้งไวยากรณ์ และซิมูเลชั่นซีเมนติค (Simulation Semantics) อย่างไรก็ตามแม้ตัวภาษาจะมีความซับซ้อนในรูปแบบ และกฎเกณฑ์ของภาษา แต่การเรียนรู้เพียงบางส่วนของภาษาก็สามารถนำไปใช้งานได้โดยไม่จำเป็นต้องศึกษารายละเอียดทั้งหมด เนื่องจากตัวภาษา VHDL ออกแบบมาให้ใช้สำหรับการออกแบบตั้งแต่วงจรที่มีขนาดเล็กจนถึงวงจรที่มีขนาดใหญ่และซับซ้อน

4.1 ประวัติความเป็นมาของภาษา VHDL

วิวัฒนาการของภาษา VHDL นั้นเริ่มต้นประมาณปี ค.ศ. 1981 โดยที่กระทรวงกลาโหมสหรัฐอเมริกา หรือ Department Of Defense (DOD) มองเห็นว่าอุปกรณ์อิเล็กทรอนิกส์ และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร เป็นอุปกรณ์ที่ได้รับการพัฒนามาเมื่อประมาณ 20 ปีก่อน เพราะเทคโนโลยีในขณะนั้นทำให้การพัฒนาอุปกรณ์อิเล็กทรอนิกส์เป็นไปอย่างล่าช้า ซึ่งเป็นสภาพที่ไม่อาจยอมรับได้ในปัจจุบัน เพราะเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์ ได้รับการพัฒนาไปอย่างรวดเร็ว ดังที่จะเห็นได้ว่ามีวงจรดิจิตอลอิเล็กทรอนิกส์หลายวงจร ที่แต่เดิมถูกสร้างขึ้นมาจากชิ้นส่วนอุปกรณ์อิเล็กทรอนิกส์จำนวนหลายชิ้น ถูกนำประกอบกันอยู่บนแผงวงจรไฟฟ้า (Printed Circuit Board :PCB) ที่มีขนาดใหญ่ แต่ในปัจจุบันสามารถที่จะใช้เทคโนโลยีการออกแบบ และผลิตวงจรรวมขนาดใหญ่มาก (Very Large Scale Integration :VLSI) รวมอุปกรณ์ ต่าง ๆ เหล่านั้นให้อยู่บนชิ้นอุปกรณ์สารกึ่งตัวนำ ที่มีขนาดประมาณ 1 – 2 ตร.ซม. ได้ซึ่งเป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้น (ความเร็วในการทำงานของวงจร) ตลอดจนความน่าเชื่อถือ (Reliability) และความคงทนต่อสภาพแวดล้อมสูง ขณะเดียวกันนั้นในวงการทหาร ได้มีการนำระบบอาวุธ อย่างแพร่หลาย โดยเฉพาะอย่างยิ่งในระบบอาวุธ ดังนั้นอุปกรณ์ที่มีใช้อยู่จึงไม่เหมาะสมกับเทคโนโลยีด้านอาวุธ ของประเทศคู่แข่งขัน การที่จะเปลี่ยนอุปกรณ์ใหม่เป็นสิ่งที่ต้อง

ใช้งบประมาณมาก และก็จะประสบกับปัญหาเช่นเดิมคือ อุปกรณ์ใหม่ก็ได้รับการพัฒนามานานแล้วเช่นกัน เพราะในขณะนั้นขั้นตอนของการออกแบบผลิต และตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ต้องใช้วิศวกร และเวลาสำหรับดำเนินการมาก ฉะนั้นทาง DOD จึงตั้งโครงการขึ้นมาเพื่อศึกษา วิธีการที่จะช่วยพัฒนาวงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามรถนำไปผลิตได้เร็วขึ้น และโครงการดังกล่าวมีชื่อว่า Very High Speed Integrated Circuits หรือ VHSIC ในระยะแรกนั้นโครงการเป็นความลับทางด้านความมั่นคงของประเทศ และอยู่ในความดูและควบคุมของ United States International Traffic and Arms Regulations (ITAR) ในปี ค.ศ. 1983 ตามคำแนะนำของคณะทำงาน ("Woods Hele" workshop) ทาง DOD ได้ออกความต้องการมาตรฐานของภาษาที่ใช้สำหรับบรรยายพฤติกรรมของวงจร หรือฮาร์ดแวร์ของระบบสำหรับโครงการ VHSIC ซึ่งมีสาระพอสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอล และมีคุณสมบัติที่สามารถจะเข้าใจได้ทั้งคน และเครื่องโดยไม่ต้องมีการแปล หรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้ (Project Documentation)
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร (Simulation Language)

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง (High Level Language) เช่นเดียวกับภาษา PASCAL, FORTRAN และ ADA ซึ่งในทางวิศวกรรมการออกแบบฮาร์ดแวร์เรียกว่า Hardware Description Language หรือ HDL ดังนั้นภาษามาตรฐานนี้จึงชื่อว่า VHSIC-HDL หรือ VHDL นั้นเอง

เริ่มต้นโครงการ DOD ได้มอบหมายให้บริษัท IBM, Texas Instruments และ Intermetrics เป็นผู้ศึกษาและพัฒนาการดำเนินการได้กระทำไปอย่างต่อเนื่อง และได้ผลเป็นที่น่าพอใจ จนกระทั่งปี ค.ศ. 1985 ทาง ITAR ได้ยกเลิกข้อจำกัดในการถ่ายทอดเทคโนโลยีทางทหาร ออกจากโครงการนี้ ดังนั้น VHDL จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป จนกระทั่งทาง IEEE ได้รับภาษานี้เข้ามาศึกษาและประมาณปี ค.ศ. 1987 ได้ยอมรับกำหนดมาตรฐานของภาษา โดยให้ชื่อว่า IEEE 1076 – 1987 และมีชื่อเรียกว่า VHDL มาตรฐานนี้ก็ได้รับการปรับปรุงจนปัจจุบัน ได้ชื่อว่า IEEE 1076 – 1993 หรือ VHDL 1993

การที่ทาง DOD ในขณะนั้น เป็นลูกค้ารายใหญ่ของอุตสาหกรรมอิเล็กทรอนิกส์ และคอมพิวเตอร์ จึงมีผู้รับโครงการต่าง ๆ จาก DOD ไปดำเนินการด้านวิจัย และพัฒนามาก เพื่อที่จะให้เป็นมาตรฐานเดียวกันหมดทาง DOD จึงกำหนดว่า ในการส่งโครงการนั้นจะต้องเขียนอยู่ในรูปของภาษา VHDL เท่านั้น ซึ่งทำให้เกิดข้อดีต่อ DOD เองที่เป็นมาตรฐานเดียวกัน สามารถนำไปจำลองกับเครื่องคอมพิวเตอร์ได้หลาย ๆ ระบบ

4.2 ความสามารถของภาษา VHDL

- ตัวภาษา VHDL สามารถอ่าน และทำความเข้าใจได้โดยมนุษย์
- ตัวภาษา VHDL สามารถใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างผู้ผลิตชิปกับผู้ออกแบบ (CAD Tools)

- ใช้เป็นสื่อกลางในการแลกเปลี่ยนระหว่างซีเออี (CAE) และซีเอดีทูลส์ (CAD Tools) เช่นตัวโค้ดของภาษา (Source Code) VHDL สามารถคอมไพล์โดยใช้คอมไพล์เลอร์ (Compiler) และ ซิมมูเลเตอร์ (Simulator) ได้หลายตัวที่แตกต่างกัน
- ภาษา VHDL สนับสนุนการออกแบบ แบบ Top-Down Design และแบบ Bottom-Up Design หรือผสมกันทั้ง 2 แบบ
- สนับสนุนการออกแบบทั้งระบบ Synchronous และ Asynchronous
- สนับสนุนการออกแบบระบบดิจิตอลในหลาย ๆ เทคนิค เช่น Finite State Machine, Algorithmic, Boolean Equation
- ภาษา VHDL สนับสนุนรูปแบบการเขียนได้ถึง 3 รูปแบบ ได้แก่ Behavioral, Structural และ Data Flow หรือสามารถเขียนรวมกันทั้ง 3 รูปแบบ
- สนับสนุนการออกแบบขนาดใหญ่โดยใช้ความสามารถของส่วนประกอบ (Component), Function procedure และ Package
- ภาษา VHDL เป็นมาตรฐานรับรองโดย IEEE และ ANSI ทำให้โมเดลที่ออกแบบโดยภาษา VHDL สามารถเคลื่อนย้ายไปยังระบบใด ๆ ก็ได้ และสามารถนำกลับมาใช้ใหม่ได้
- ภาษา VHDL เป็นแบบทั่วไป คือไม่อิงเทคโนโลยีอันใดอันหนึ่งสามารถอิงเทคโนโลยีใดก็ได้ และในขณะเดียวกันก็สามารถสนับสนุนหลาย ๆ เทคโนโลยี
- ภาษา VHDL เป็นมาตรฐานที่ใช้โดยบริษัทและผู้ออกแบบหลาย ๆ แห่ง ฉะนั้นจึงเป็นการง่ายที่จะทำความเข้าใจ ถึงแม้มาจากแหล่งต่างๆ
- ไม่จำเป็นต้องศึกษาโปรแกรมซิมมูเลท เพราะรูปแบบการซิมมูเลทสามารถเขียนได้โดยใช้ภาษา VHDL เช่นกัน
- สามารถเขียนโมเดลได้ขนาดไม่จำกัด ไม่มีข้อจำกัดในตัวภาษาเรื่องขนาดของโมเดล (ขึ้นอยู่กับโปรแกรม)
- สามารถอธิบายตัวแปรที่เกี่ยวกับฟังก์ชั่นทางด้านเวลาเช่น Propagation Delay, Min-Max Delay, Setup time, Holding time, Spike Detection ได้ภายในตัวภาษา
- มีการประกาศเจนเนอร์ริกส์ (Generics) ช่วยให้เราสามารถสร้างรูปแบบที่เปลี่ยนแปลงได้
- โมเดลที่สร้างด้วยภาษา VHDL นั้น ไม่เพียงแต่จะอธิบายฟังก์ชั่นการทำงานเท่านั้นยังสามารถอธิบายถึงรายละเอียดของตัวโมเดล เช่น ขนาดของโมเดล และ ความเร็วของโมเดล
- โมเดลที่สร้างขึ้นสามารถจำลองการทำงานได้ เพราะตัวแปลได้ตรวจสอบไวยากรณ์ทางด้านซิมมูเลชันซีแมนติคไว้ด้วย
- มีความสามารถที่ให้เราออกแบบข้อมูลใหม่ ๆ ได้ ทำให้ VHDL โมเดล เป็นการออกแบบระดับสูง ที่ไม่ต้องคำนึงถึงว่าจะสร้างตัวโมเดลนั้นขึ้นมาได้อย่างไร

4.3 Top-Down Design

ภาษา VHDL สามารถกำหนด และบรรยายพฤติกรรมฟังก์ชันการทำงานของฮาร์ดแวร์ในระบบดิจิตอลนั้น คือความอ่อนตัวของภาษาที่สามารถจำลองการทำงานจากหลักการของรูปแบบ (Simulate

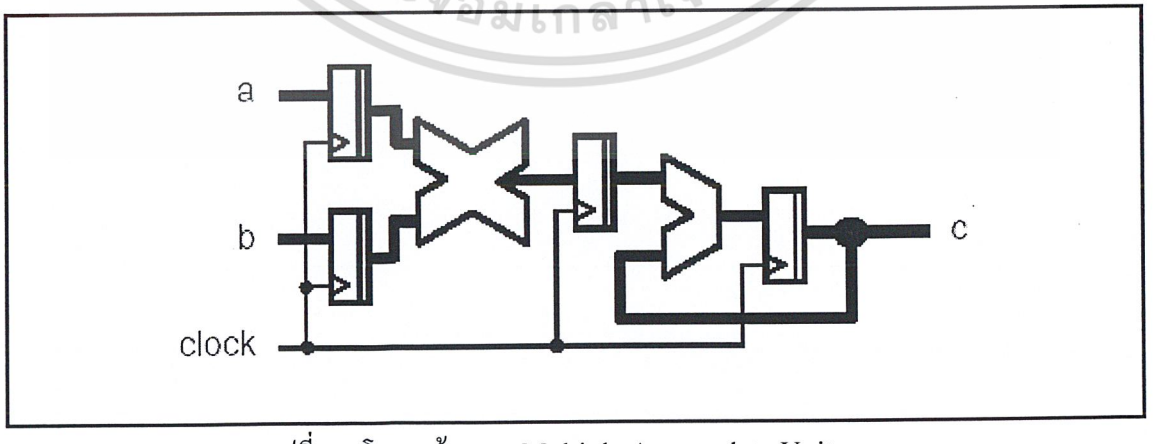
conceptual designs) แต่ในขณะเดียวกันก็สามารถจำลองการทำงานของฮาร์ดแวร์ที่ให้รายละเอียดเกี่ยวกับเวลาอย่างถูกต้อง (Timing based simulations) และจากโครงสร้างของภาษายังสามารถจำลองการทำงานในรูปแบบของลำดับชั้น (Hierarchy of simulation levels) ความสามารถดังกล่าวนี้จึงช่วยให้วิศวกรออกแบบ สามารถที่จะเขียนรูปแบบบรรยาย จากระดับบนสุดของวงจรที่อยู่ในรูปสังเขป (High level of abstraction) ลงสู่รายละเอียดในระดับล่างของวงจรได้เช่น ระดับเกต เป็นต้น ในช่วงเวลานั้นเองวงการอุตสาหกรรมไมโครอิเล็กทรอนิกส์ ตลอดจนสถาบันวิจัยและศึกษา กำลังพัฒนาภาษาที่จะใช้สำหรับการสังเคราะห์วงจรแบบอัตโนมัติ เพื่อลดเวลาในการพัฒนาวงจรลงภาษา VHDL จึงถูกนำเข้าพิจารณาในโครงการนี้ด้วย โดยเพิ่มขีดความสามารถของภาษาขึ้นอีกประการหนึ่ง นอกเหนือจากสิ่งทาง DOD กำหนดในครั้งแรกคือเป็นภาษาที่ใช้สำหรับสังเคราะห์วงจร (Synthesis Language)

ภาษา VHDL เป็นภาษาที่สนับสนุนการเขียนรูปแบบในทุก ๆ ลักษณะและวิธีการ ดังเช่นตัวอย่างที่แสดงในรูปที่ 4.1 คือรูปแบบ (Model) ของลำดับขั้นตอนของการคูณ และหาผลรวม (Multiply accumulate algorithm) ของตัวแปรสองตัว a และ b ส่วนผลลัพธ์คือ c ในลักษณะของเลขฐานสอง (Binary number) รูปแบบนี้แสดงระดับที่สังเขปที่สุดของแนวความคิดที่จะแก้ปัญหา เพื่อผลลัพธ์โดยไม่ได้คำนึงถึงโครงสร้างของวงจรอย่างที่เคยชิน เช่นอุปกรณ์วงจรรวม Arithmetic and Logic Unit (ALU)

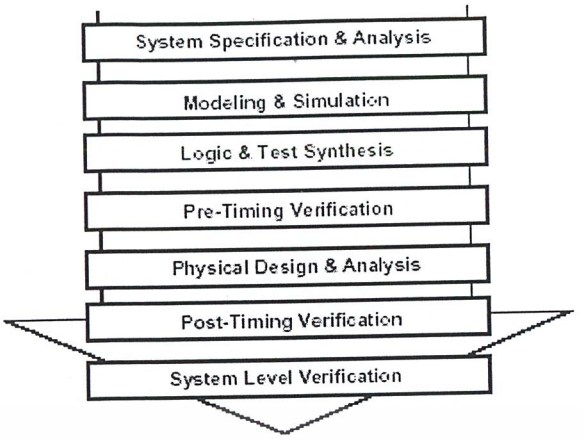
```
FOR i IN 1 TO 16 LOOP
        result := result + a(i) * b(i);
END LOOP;
C <= result;
```

รูปที่ 4.1 VHDL Statement สำหรับ Multiple Accumulate Algorithm

เราสามารถอธิบายภาษา VHDL ที่ใช้บรรยายลำดับขั้นตอนของการคูณ และหาผลรวม (Multiple Accumulate Algorithm) ได้โดยแสดงในรายละเอียดของการสร้างวงจรดังกล่าวจริง ๆ ตามที่เห็นได้จากที่ 4.2



รูปที่ 4.2 โครงสร้างของ Multiple Accumulate Unit



รูปที่ 4.3 ขั้นตอนของ Top-Down Design

ขั้นตอนของขบวนการออกแบบโดยใช้วิธี Top-Down Design มีรายละเอียดดังนี้

1. System Specification and Analysis ขั้นตอนของการสร้างข้อกำหนดของความต้องการ (Specification) และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ในการแก้ปัญหา
2. Modeling and Simulation การเขียนรูปแบบของระบบที่ต้องการออกแบบโดยภาษา VHDL หรือ ภาษา HDL อื่น ๆ จากแนวความคิดอย่างสังเขปที่ได้ สำหรับบรรยายพฤติกรรมการทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด (Specification)
3. Logic and Test Synthesis หลังจากที่ได้หลักการขั้นต้นพร้อมกับแนวคิดที่ผ่านการตรวจสอบแล้ว หลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมาเป็นลำดับขั้นตอนที่เหมือนกัน คือ Modeling and simulation จนกระทั่งอยู่ในระดับที่จะนำไปผลิตวงจรหรือสังเคราะห์ (Synthesis) ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับวงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของวงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ (Gate-level) และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้น หรือไม่ก็อยู่ในรูปของ เน็ตลิสต์ (netlist) ที่สามารถนำไปผลิตลงบนอุปกรณ์อื่นได้ นอกจากนั้นการผลิตบางเทคโนโลยี อาทิเช่น Gate Array หรือ Standard Cell และ Full Custom IC อาจจะมีความจำเป็นที่ต้องสร้างโครงสร้างของวงจรใหม่หลังจากที่สังเคราะห์ครั้งแรกแล้ว เพื่อความสะดวกต่อการตรวจสอบการทำงานหลังจากที่ผลิตเป็นวงจรต้อนแบบแล้ว หรือที่เรียกว่า “Design For Test” (DFT) พร้อมทั้งข้อมูลในการตรวจสอบ (Test pattern) จะถูกกำหนดในขั้นตอนนี้
4. Pre-Timing Verification หลังจากการสังเคราะห์วงจรให้อยู่ในรูป Gate-level หรือเน็ตลิสต์แล้วข้อมูลที่ได้จากผู้ผลิตอุปกรณ์วงจรนั้น นอกจากจะเป็นข้อมูลสำหรับจำลองการทำงาน ในเรื่องของความถูกต้องของฟังก์ชัน (Functional simulation) แล้วยังมีข้อ

มูลที่เกี่ยวกับเวลาด้วย ซึ่งเป็นความจริงที่ว่า อุปกรณ์อิเล็กทรอนิกส์ทุกชิ้นจะมี Propagation delay เสมอ ถึงแม้ว่าจะเป็นเวลาที่น้อยมากในระดับ nanosecond (10^{-9} second) แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกตของฟังก์ชันต่าง ๆ จำนวน 10,000 เกต ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจจะทำให้การทำงานของวงจรรวมทั้งหมดผิดไป หรือไม่สามารถทำงานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

5. Physical Design and Analysis คือขั้นตอนของการผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจจะอยู่ในรูปของแผงวงจรไฟฟ้า (Printed Circuit Board: PCB) ที่ประกอบด้วยอุปกรณ์หลาย ๆ ชิ้น หรืออยู่ในรูปของวงจรรวมเฉพาะงาน (ASIC)
6. Post-Timing Verification หลังจากที่ได้วงจรจริงมาแล้ว ยังต้องมีความจำเป็นที่ต้องตรวจสอบการทำงานที่คำนึงถึงเวลาด้วย เพื่อความถูกต้องของวงจรครั้งสุดท้ายก่อนที่จะนำไปรวมเข้ากับอุปกรณ์อื่น ๆ ให้เป็นระบบดิจิตอล เพราะในขั้นตอนนี้วงจรที่ออกแบบจะประกอบด้วย Input และ Output pad ซึ่ง เป็นจุดต่อสำหรับรับ และส่งสัญญาณกับภายนอก
7. System Level Verification หลังจากที่นำวงจรที่ออกแบบรวมเข้ากับอุปกรณ์อื่น ๆ ให้เป็นระบบดิจิตอลแล้วนั้น จะต้องทดสอบการทำงานรวมทั้งระบบร่วมกับอุปกรณ์อื่น ๆ อีกครั้งเป็นการควบคุมคุณภาพของการผลิต

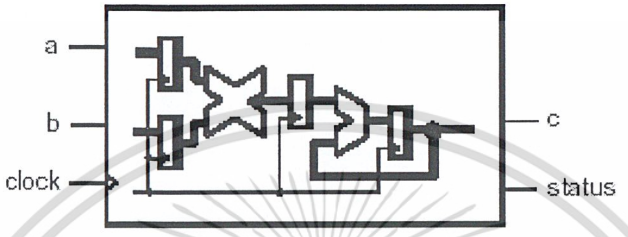
จากความอ่อนตัวของภาษา และความสามารถที่จะเขียนรูปแบบได้หลายลักษณะนี้เอง VHDL จึงเป็นเครื่องมือที่ใช้สำหรับออกแบบตั้งแต่ขั้นตอนบนสุด คือแนวความคิดที่จะแก้ปัญหาลงไปที่ละขั้นจนถึงขั้นตอนของการผลิตวงจรจริง (Form idea to implementation) ข้อดีที่เห็นได้ชัดของการนำ VHDL มาใช้ในการออกแบบลักษณะ Top-Down นี้คือ วิศวกรออกแบบสามารถที่จะสร้างรูปแบบ (Model) และจำลองการทำงาน เพื่อตรวจสอบความถูกต้องกับข้อกำหนด (Specification) ตั้งแต่เริ่มแรกที่มีแนวความคิดอย่างสังเขป จากการจำลองการทำงานในระยะต้น ๆ ของการออกแบบนั้น หลักการต่าง ๆ ที่ถูกกำหนดขึ้นในการแก้ปัญหา (สร้างวงจรให้เป็นตามความต้องการของข้อกำหนด) จะถูกตรวจสอบด้วยทุกครั้งก่อนที่จะมีการลงทุนในขั้นตอนสุดท้ายของการออกแบบ หรือการสร้างวงจรนั้นเอง นั้นหมายความว่าข้อผิดพลาดที่อาจจะเกิดขึ้นได้จากหลักการที่กำหนดขึ้น จะถูกตรวจพบและจัดการแก้ไขให้ถูกต้องได้ ก่อนที่จะทำงานในขั้นตอนต่อ ๆ ไปของขบวนการการออกแบบ

ในตัวอย่างของ Multiply accumulate algorithm สามารถแสดงให้เห็นขบวนการออกแบบในลักษณะของ Top-Down Design ได้โดยการใช้ Multiply accumulate function จุดประสงค์แรกก็คือการเขียนรูปแบบของฟังก์ชันในระดับบนสุดด้วยภาษา VHDL และจำลองการทำงานของรูปแบบที่ได้เพื่อตรวจสอบความถูกต้อง ซึ่งฟังก์ชันนี้อาจจะเป็น Discrete Fourier Transform (DFT) ก็ได้ หลังจากที่ได้ฟังก์ชันที่ต้องการแล้ว สามารถนำไปใช้ได้ในรูปของ Function call คือ

```
C <= multiply_accum(a,b);
```

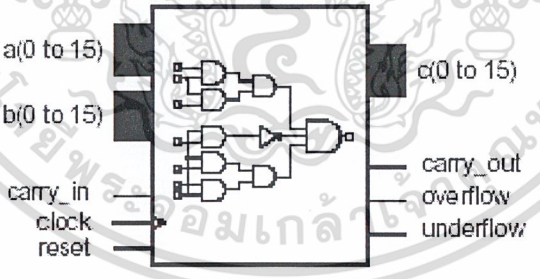
รูปที่ 4.4 แสดงการใช้งาน Function ของภาษา VHDL

ถ้าฟังก์ชันที่ได้ทำงานเป็นที่พอใจแล้ว ในขั้นตอนต่อไปจะเป็นการเริ่มต้นของการแปลงแนวความคิด ไปสู่การสร้างวงจรจริง ขบวนการดังกล่าวเกิดขึ้นโดยการ เพิ่มรายละเอียดให้กับแนวความคิดเดิม รูปที่ 4.5 แสดงให้เห็นวิธีการบรรยายแบบ Dataflow ของ Multiply accumulate function



รูปที่ 4.5 การบรรยายแบบ Dataflow ของ Multiply accumulate function

จากรูปแบบ Dataflow ที่ได้นี้ ขบวนการต่อไปคือการสร้างวงจรให้อยู่ในรูปของอุปกรณ์พื้นฐาน หรือ Gate-level implementation ซึ่งอาจใช้วิธีการสังเคราะห์อัตโนมัติ ผลลัพธ์ที่ได้จากการสังเคราะห์จะเป็นการแสดงภาพวงจรด้วยเกตของฟังก์ชันต่าง ๆ หรือในรูปของเน็ตลิสต์และจะเป็นการกำหนดเทคโนโลยีจำเพาะที่จะนำไปผลิตในภายหลัง ในรูปที่ 4.6 แสดงให้เห็นรูปร่างของวงจรที่ได้จากการสังเคราะห์ในระดับเกต



รูปที่ 4.6 Gate-level Representation ของ Multiply accumulate function

คุณสมบัติหลักของภาษา VHDL คือความสามารถที่จะใช้บรรยายฮาร์ดแวร์ได้ในทุก ๆ ระดับของภาพรวมทั้งระบบ ฉะนั้นวิศวกรออกแบบจึงสามารถใช้เครื่องมือ (ภาษา) เพียงอันเดียวในการบรรยาย ซึ่งก็เช่นเดียวกันกับเครื่องมือจำลองการทำงาน (Simulator)

4.4 ลักษณะของรูปแบบ (Model styles) ของภาษา VHDL

ลักษณะของการเขียนรูปแบบ (Model) ด้วยภาษา VHDL สามารถแบ่งได้เป็น

1. **Behavioral Model** หรือเรียกอีกอย่างได้ว่า **Algorithmic description** เป็นรูปแบบที่บรรยายพฤติกรรมของระบบดิจิตอล ในส่วนที่บรรยายมีโครงสร้างคล้ายกับภาษาชั้นสูง (High level

language) ทั่ว ๆ ไป เช่น PASCAL หรือ C เป็นต้น ในการจำลองการทำงาน (Simulation) คำสั่งแต่ละคำสั่ง (Statement) จะถูกประเมินผลเป็นไปตามลำดับ (Sequential) จากบนลงล่าง ยกเว้นในกรณีของคำสั่ง LOOP หรือการเรียกใช้โปรแกรมย่อย รูปแบบลักษณะนี้จะไม่ให้รายละเอียดที่เกี่ยวกับโครงสร้างของฮาร์ดแวร์แต่ในทางตรงกันข้าม จะให้รายละเอียดที่เกี่ยวกับความสัมพันธ์ระหว่าง Input กับ Output ได้ดี

2. **Dataflow Model** เรียกอีกอย่างหนึ่งได้ว่า “Register Transfer Level” (RTL) เป็น รูปแบบที่ถูกเขียนขึ้น เพื่อจุดประสงค์ที่จะใช้เครื่องมือสำหรับสังเคราะห์วงจรอัตโนมัติ รูปแบบลักษณะนี้ส่วนใหญ่จะเป็น Procedural constructs และ Functional operators (ดูรูปที่ 4.1)
3. **Structural Model** เป็นรูปแบบที่แสดงการเชื่อมต่อกันระหว่างอุปกรณ์ต่าง ๆ ที่ประกอบกันขึ้นเป็นวงจร หรือระบบดิจิตอล และสามารถเรียกอีกอย่างได้ว่า “Netlist Representation” เป็นการเขียนที่แสดงให้เห็นโครงสร้างของฮาร์ดแวร์
4. **Mixed-level Model** จากคุณสมบัติที่อ่อนตัวของภาษา VHDL จึงสามารถที่จะเขียนรูปแบบโดยใช้ลักษณะต่าง ๆ ที่กล่าวมาแล้วข้างต้น บรรยายวงจร หรือระบบดิจิตอลเดียวกันได้ ฉะนั้นรูปแบบเช่นนี้จึงเป็นลักษณะการเขียนแบบผสมทั้ง 3 แบบข้างต้น

บทที่ 5
การออกแบบวงจรดิจิตอลด้วย FPGA และการใช้งานโปรแกรมทูล

5.1 วิธีการออกแบบวงจรดิจิตอล

ตามปกติแล้ว การออกแบบวงจรดิจิตอลเพื่อใช้งานร่วมกับอุปกรณ์ประเภทเอฟพีจีเอ (FPGA: Field Programmable Gate Arrays) นั้นจะมีขั้นตอนที่ไม่แตกต่างจากการออกแบบวงจรดิจิตอลประเภทเอสิก (ASIC: Application Specific Integrated Circuits) มากนักโดยเฉพาะอย่างยิ่งการใช้ภาษาอธิบายฮาร์ดแวร์ในการออกแบบวงจร หรือที่เรียกกันว่า เอชดีแอล (HDL: Hardware Description Language) โดยที่เราสามารถเลือกเทคโนโลยีเป็นหมาย (Target Technology) ได้ตามต้องการ เช่น เลือกเทคโนโลยีเป้าหมายเป็น Standard Cell ASIC หรือ FPGA เป็นต้น แต่จะต้องเลือกใช้ไลบรารี่สำหรับการสังเคราะห์อย่างเหมาะสมด้วย เนื่องจากปัญหาสำคัญส่วนหนึ่งของการออกแบบอยู่ตรงที่การกำหนดเทคโนโลยีเป้าหมาย ดังนั้นอาจจะมีบางจุดในโค้ดที่เราได้เขียนขึ้นต้องได้รับการเปลี่ยนแปลงแก้ไขก่อนเพื่อเหมาะสมและถูกต้อง เมื่อต้องการจะเปลี่ยนจากเทคโนโลยีเป้าหมายหนึ่งไปยังเทคโนโลยีอื่น ๆ ก็ทำได้ไม่ยาก ซึ่งไม่เหมือนกับวิธีการออกแบบที่ใช้ซีมเมติก (Schematic) เพราะสำหรับวิธีหลังนี้ ดีไซน์จะถูกเขียนอธิบายให้อยู่ในรูปของสัญลักษณ์ทางกราฟิกที่ประกอบขึ้นจากองค์ประกอบพื้นฐาน (Primitives) และมีอยู่ในไลบรารี่สำหรับเทคโนโลยีเป้าหมาย (Target Library) แล้วนำมาต่อเข้าด้วยกันเป็นวงจรที่เสร็จสมบูรณ์ ซึ่งการใช้ซีมเมติกนี้เป็นการอธิบายวงจรในเชิงโครงสร้าง (Structural Description) นั้นเอง จุดด้อยของวิธีการออกแบบโดยซีมเมติกนี้ก็คือ การเปลี่ยนแปลงแก้ไขดีไซน์นั้นทำได้ค่อนข้างยากลำบาก และใช้เวลามาก และอีกประการที่สำคัญคือ ดีไซน์ที่ถูกสร้างขึ้นนั้นจะอยู่กับไลบรารี่เป้าหมายที่ได้เลือกใช้ ถ้าจะเปลี่ยนไลบรารี่ก็ย่อมทำได้ลำบาก โดยเฉพาะอย่างยิ่งวงจรที่มีขนาดใหญ่ เพราะจะต้องมาแก้ไขซีมเมติกใหม่ ดังนั้นในการออกแบบจะใช้การออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์เท่านั้น

5.2 ขั้นตอนการออกแบบวงจรโดยใช้ภาษาอธิบายฮาร์ดแวร์

ดังที่ได้กล่าวไปแล้วในปัจจุบันแนวโน้มการออกแบบวงจรดิจิตอลขนาดใหญ่จะหันเหไปทางการใช้งานภาษาอธิบายฮาร์ดแวร์ ซึ่งมีบทบาทมาขึ้นตามลำดับด้วยเหตุผลหลายประการ อาทิเช่น ความสามารถในการใช้อธิบายการทำงานของวงจรที่มีขนาดใหญ่มาก ความสามารถในการนำกลับไปใช้งานได้อีกของดีไซน์ที่ถูกสร้างขึ้น ความสามารถในการนำไปใช้งานได้กับหลาย ๆ เทคโนโลยีเป้าหมาย โดยโค้ดที่ถูกเขียนจะไม่ขึ้นอยู่กับเทคโนโลยีเหล่านั้น หรือเป็นเพียงส่วนน้อยเท่านั้น นอกจากนั้นยังมีซอฟต์แวร์ที่มีประสิทธิภาพอีกมากมายสำหรับใช้ในการสังเคราะห์วงจร

5.2.1 การออกแบบวงจรให้อยู่ในรูปแบบของโค้ดภาษาอธิบายฮาร์ดแวร์

การออกแบบวงจรโดยการใช้ภาษา VHDL จะเริ่มต้นที่การเขียนโค้ดหลังจากที่ได้มีการออกแบบสถาปัตยกรรมของวงจรอย่างดีแล้ว โดยเริ่มจากการแบ่งวงจรรวมออกเป็นวงจรย่อย เพื่อให้ง่ายต่อการออกแบบตามหลักการของ Top-Down Design ที่กล่าวถึงในบทที่ 4 ถ้าวงจร หรือบล็อกย่อยไม่ซับซ้อนจน

เกินไป และง่ายต่อการสังเคราะห์วงจรในภายหลัง ก็สามารถเขียนอธิบายพฤติกรรมการทำงานของวงจรเหล่านั้น (Behavioral Description) เป็นโมเดลในภาษา VHDL และเมื่อวงจรย่อยเหล่านั้นเสร็จแล้ว จึงนำมาประกอบเข้าด้วยกันโดยการเรียกใช้ส่วนประกอบ (Component Instantiation) และมีสายสัญญาณเชื่อมต่อ (Signals) อย่างเหมาะสม ดังนั้นการอธิบายวงจรในระดับนี้จึงเป็นการอธิบายในเชิงโครงสร้าง (Structural Description)

การเขียนโค้ดภาษา VHDL เช่นเดียวกับภาษาสูงอื่น ๆ สามารถใช้โปรแกรมจำพวก Text Editor ทั่ว ๆ ไปได้ แต่อย่างไรก็ตาม ยังมีซอฟต์แวร์หลายตัวที่สนับสนุนการเขียนโค้ดภาษา VHDL ซึ่งทำให้ง่ายขึ้นเวลาเขียนโค้ดในภาษาVHDL รวมทั้งการคอมไพล์โค้ดเพื่อตรวจเช็คความถูกต้องตามหลักไวยากรณ์ด้วย ตัวอย่างของคอมไพล์เลอร์ภาษา VHDL สำหรับเครื่องพีซี เช่น ModelSim PE/Plus ของ Model Technology, Inc. เป็นต้น MEL ก็ได้ใช้ซอฟต์แวร์ตัวนี้ในการคอมไพล์โค้ดภาษา VHDL รวมถึงการทำซิมมูเลชั่น (Simulation) ด้วย (ตัวซอฟต์แวร์และการใช้งานจะกล่าวถึงในภายหลัง)

5.2.2 การตรวจสอบความถูกต้องของวงจรที่สร้างขึ้นในระดับพฤติกรรม

ขั้นตอนอีกขั้นตอนหนึ่งที่สำคัญในการออกแบบวงจร คือ การตรวจสอบว่า โมเดลในภาษา VHDL ที่เราได้สร้างขึ้นนั้นสามารถทำหน้าที่ได้ตามที่กำหนดไว้หรือไม่ ซึ่งสามารถทำได้โดยการทำซิมมูเลชั่นเพื่อดูพฤติกรรมการทำงานของโมเดลเหล่านั้น (Behavioral Simulation) และเพื่อจุดประสงค์นี้เอง จะต้องมีการเขียนโมเดลพิเศษสำหรับการทำซิมมูเลชั่นขึ้นอีก ที่เรียกว่า VHDL Testbench ซึ่งภายในจะมีการเรียกใช้โมดูลหรือส่วนประกอบที่ต้องการจะตรวจสอบพฤติกรรม การทำงาน และมีการกำหนดรูปแบบของสัญญาณขาเข้า (Input Signal) ที่เรียกว่า Stimulus และเมื่อทำซิมมูเลชั่น สัญญาณขาออก (Output Signal) ของโมดูล ก็จะปรากฏให้เห็นบนจอภาพ และสามารถนำไปเปรียบเทียบกับสัญญาณขาเข้าเพื่อตรวจดูว่า การทำงานของโมดูล หรือดีไซน์ที่ถูกตรวจสอบ (Design Under Test : DUT) เป็นไปตามที่คาดหวังไว้หรือไม่

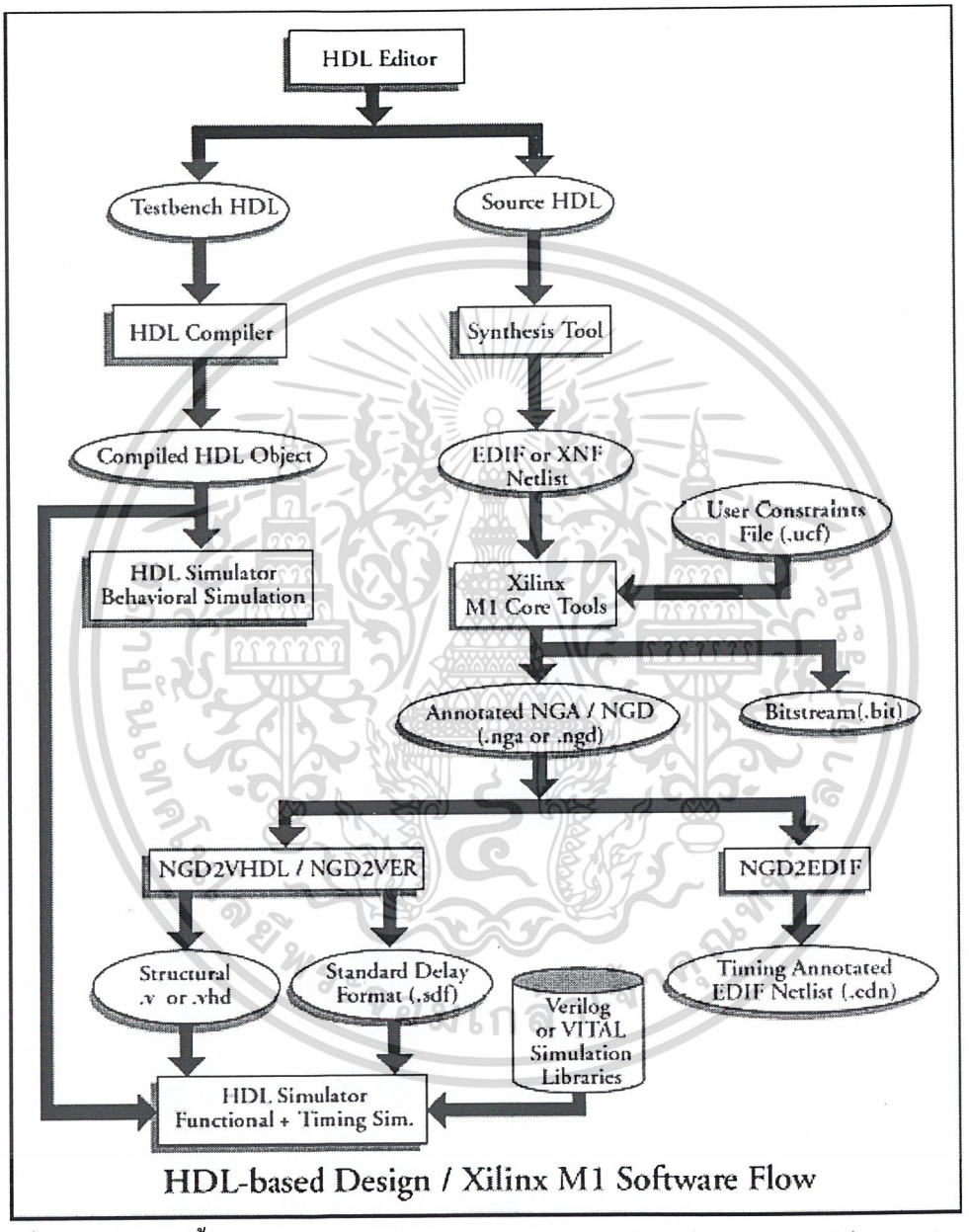
5.2.3 การสังเคราะห์วงจรจากดีไซน์ในภาษาอธิบายฮาร์ดแวร์

เมื่อได้ตรวจสอบเป็นที่แน่ใจแล้วว่า โมเดลต่าง ๆ ที่ได้สร้างขึ้นสามารถทำงานได้อย่างถูกต้องขั้นตอนต่อไปก็คือการผ่านโค้ดของโมเดลเหล่านั้นไปทำการสังเคราะห์วงจรเพื่อให้ได้เน็ตลิสต์ในระดับเกตออกมา เครื่องมือที่ใช้ในการสังเคราะห์โค้ดภาษา VHDL ที่ทาง MEL ใช้อยู่ก็คือ ซอฟต์แวร์ของ Exemplar Logic Inc. ที่มีชื่อว่า Leonardo ซึ่งสามารถใช้ในการสังเคราะห์วงจรให้เป็นเอสิกหรือเอฟพีจีเอก็ได้เพราะมีคลังไลบรารี่ให้เลือกใช้ได้จากหลายผู้ผลิต เนื่องจากเราจะสังเคราะห์วงจรสำหรับใช้งานกับเอฟพีจีเอของ Xilinx ดังนั้นไลบรารี่ที่ใช้ก็จะเป็นของ Xilinx คือไลบรารี่ของตระกูล XC4000E เป็นต้น (ตัวซอฟต์แวร์และการใช้งานจะกล่าวถึงในภายหลัง)

5.2.4 การแปลงดีไซน์ที่สังเคราะห์แล้วเพื่อใช้กับอุปกรณ์เอฟพีจีเอ

ผลที่ได้จากการสังเคราะห์นั้นจะเป็นเน็ตลิสต์ในระดับเกต โดยสามารถนำไปผ่านขั้นตอนการแปลงให้เป็นลอจิกบล็อก และรวมถึงการวางและเชื่อมต่อเส้นทาง (Place & Route) ภายในอุปกรณ์เอฟพีจีเอ โดยใช้ซอฟต์แวร์ M1 Implementation Tool ดังภาพที่ 5.1 แสดงให้เห็นถึงแผนขั้นตอนการออกแบบวงจรดิจิตอลสำหรับอุปกรณ์เอฟพีจีเอโดยใช้ภาษาอธิบายฮาร์ดแวร์ตั้งแต่เริ่มสร้างดีไซน์ให้อยู่ในรูป

ของโมดูลภาษา VHDL หรือ Verilog ผ่านขั้นตอนการทำซิมมูเลชั่น และการสังเคราะห์ ไปจนถึงการเข้าสู่ภาวะ การทำงานของ Xilinx M1 Core Tools อย่างคร่าว ๆ และรวมถึงการตรวจสอบดีไซน์ในขั้นสุดท้ายด้วย การทำงานของ M1 Core Tools จะถูกควบคุมโดย Design Manager Flow Engine โดยใช้ซอฟต์แวร์ Xilninx Design Manager 1.4.12 ซึ่งจะกล่าวถึงต่อไปภายหลัง



รูปที่ 5.1 ภาพแสดงขั้นตอนการออกแบบวงจรดิจิตอลสำหรับอุปกรณ์ FPGA โดยใช้ภาษาอธิบายฮาร์ดแวร์ร่วมกับ M1 Implementation Tools ของ Xilinx

5.2.5 การทำซิมมูเลชั่นหลังผ่านขั้นตอนการวางและเชื่อมเส้นทาง

การทำซิมมูเลชั่นในระดับพฤติกรรมหรือหลังจากการสังเคราะห์วงจรแต่ยังมิได้ผ่านขั้นตอนการวางและเชื่อมเส้นทางนั้นจะใช้ในการตรวจสอบความถูกต้องทางลอจิกของวงจรเท่านั้น แต่ยังไม่มีรายละเอียดที่แน่นอนเกี่ยวกับเรื่องของเวลาหรือไทม์มิ่ง (Timing) มากนัก เช่น ความล่าช้าของสัญญาณไฟฟ้า

ตามเส้นทางต่าง ๆ ภายในวงจร ระยะคาบของสัญญาณเวลา (Clock Period) น้อยที่สุดเท่าที่จะสามารถใช้กับวงจรได้ เป็นต้น เพราะจะต้องให้ผ่านขั้นตอนการวางและเชื่อมเส้นทางก่อน ดังนั้นการทำซิมมูเลชั่นเพื่อตรวจสอบการทำงานของวงจรในอุปกรณ์เอฟพีจีเอโดยคำนึงถึงเรื่องไทม์มิ่งจะกระทำเป็นขั้นตอนสุดท้าย

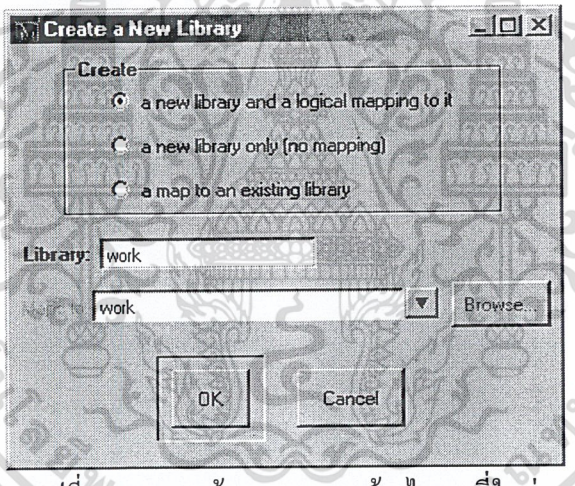
5.3 การใช้งานโปรแกรม ModelSim 5.4 SE

การใช้งานโปรแกรม ModelSim แรกสุดก่อนที่จะได้ทำการคอมไพล์โค้ดภาษา VHDL ผู้ใช้ต้องทำการสร้างไลบรารี่ที่สำหรับใช้งานก่อน โดยไลบรารี่ที่สร้างต้องเป็นชื่อ work เท่านั้น

การสร้างไลบรารี่ใหม่ ขั้นตอนการสร้างไลบรารี่สามารถทำได้ดังนี้

เลือกเมนู Design → Create a New Library

- ในช่อง Create เลือก “a new library and a logical mapping to it”
- ในช่อง Library พิมพ์คำว่า work โดยไลบรารี่ที่สร้างต้องเป็นไลบรารี่ work เท่านั้น
- คลิ้กปุ่มOK



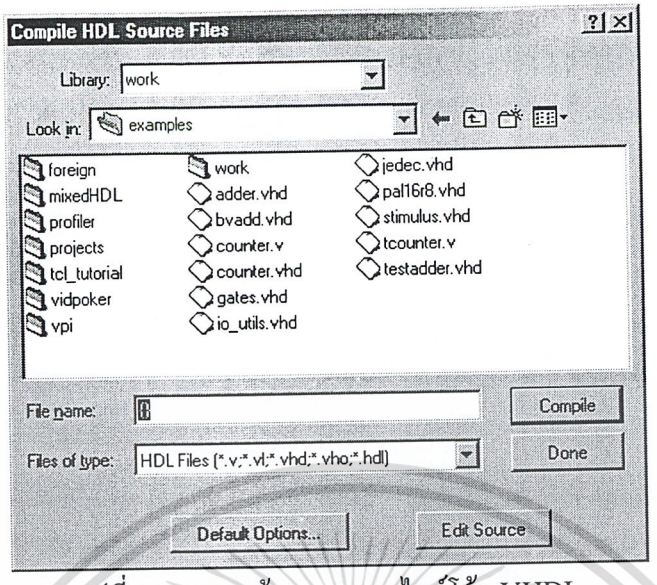
รูปที่ 5.2 แสดงหน้าจอของการสร้างไลบรารี่ใหม่

การคอมไพล์ ขั้นตอนต่อมาในการคอมไพล์โค้ดภาษา VHDL สามารถทำได้โดย

เลือกเมนู Design → Compile หรือคลิ้กที่ไอคอนแรกสุด 

- เลือกไฟล์ที่ต้องการคอมไพล์
- คลิ้กปุ่ม Compile หรือกดปุ่มคลิ้กสองครั้ง ผู้ใช้สามารถคอมไพล์ไฟล์ได้เรื่อย ๆ โดยไม่ต้องเลือกเมนูใหม่โดยทำการเลือกไฟล์ แล้วคลิ้กปุ่ม Compile อีกครั้ง
- เมื่อทำการคอมไพล์เสร็จสิ้นหรือไม่ต้องการคอมไฟล์ต่อให้คลิ้กที่ปุ่ม Done

ถ้า ModelSim ตรวจพบความผิดพลาดใด ๆ เช่น เนื้อหาภายในไฟล์ไม่ถูกต้องตามหลักไวยากรณ์ก็จะแสดงข้อความเตือนพร้อมลักษณะพร้อมลักษณะของความผิดพลาดให้ทราบเช่นกัน ถ้ามีข้อผิดพลาดอยู่ภายในไฟล์ก็จะต้องไปแก้ไขไฟล์ดังกล่าวก่อน



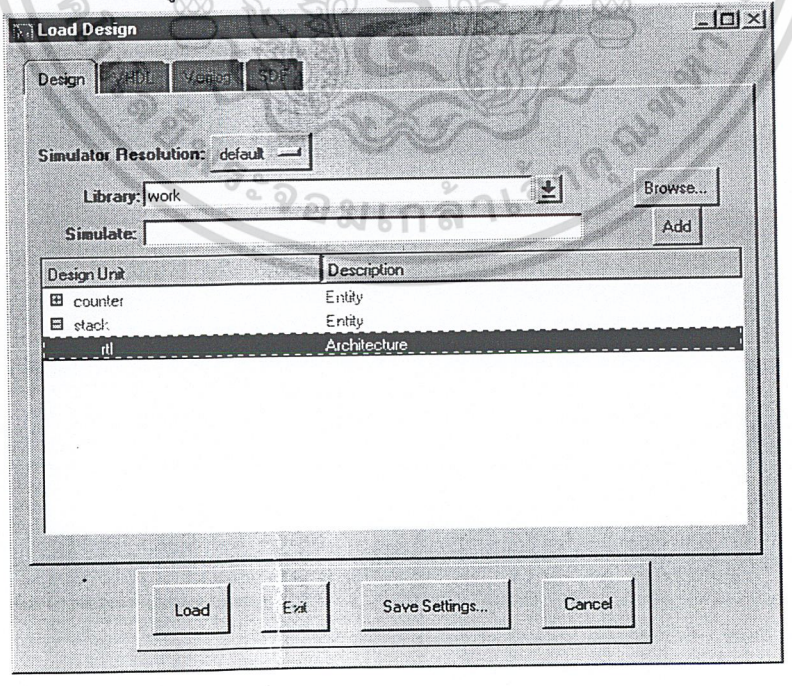
รูปที่ 5.3 แสดงหน้าจอการคอมไพล์โค้ด VHDL

การซิมมูเลชั่น ขั้นตอนต่อมาในการทำซิมมูเลชั่น หรือโหลดดีไซน์ทำได้โดย

เลือกเมนู Design → Load new Design หรือคลิ้กที่ไอคอนที่สอง

- เลือกดีไซน์ที่ต้องการทำการซิมมูเลชั่น
- คลิ้กปุ่ม Load

จากนั้นเริ่มทำการซิมมูเลชั่นโดยการกดที่ปุ่ม RUN โดยผู้ใช้สามารถเลือกทำการซีมูเลชั่นได้ที่ละกี่นาโนวินาที (ns) ก็ได้โดยการใส่ค่าในช่อง Run Length เมื่อได้สัญญาณแล้วผู้ใช้สามารถตรวจสอบผลการทำงานของดีไซน์ได้โดยดูที่แบบคลื่นของสัญญาณ และทำการทดลองใส่ค่าของสัญญาณ



รูปที่ 5.4 แสดงหน้าจอการเลือกดีไซน์ที่ต้องการซิมมูเลชั่น

การดูคลื่นของสัญญาณ และการทดลองใส่ค่าสัญญาณ ขั้นตอนต่อมาในการดูคลื่นของสัญญาณสามารถทำได้โดย

เลือกเมนู View → Signals ในหน้าต่างแสดงค่าสัญญาณของดีไซน์ของผู้ใช้

- เลือกเมนู View → Wave ในหน้าต่างของ Signals โดยจะมีเมนูย่อยให้เลือกอีก 3 เมนูคือ

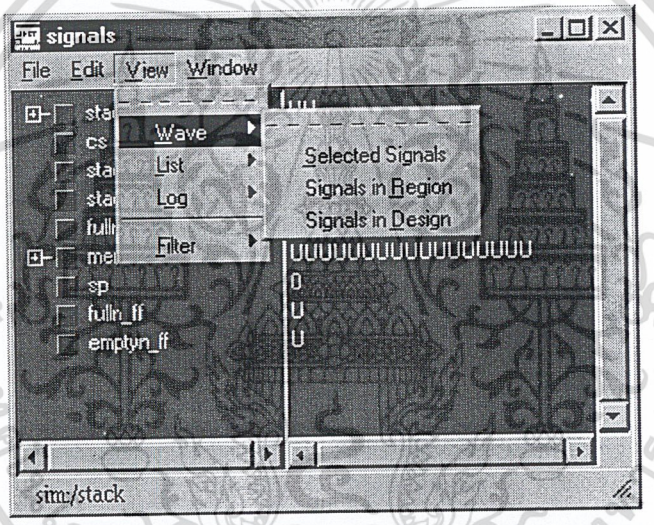
  Selected Signals คือการเลือกเฉพาะสัญญาณที่ต้องการดู

  Signals in Region คือการเลือกดูเฉพาะสัญญาณที่เป็นอินพุต และเอาต์พุต

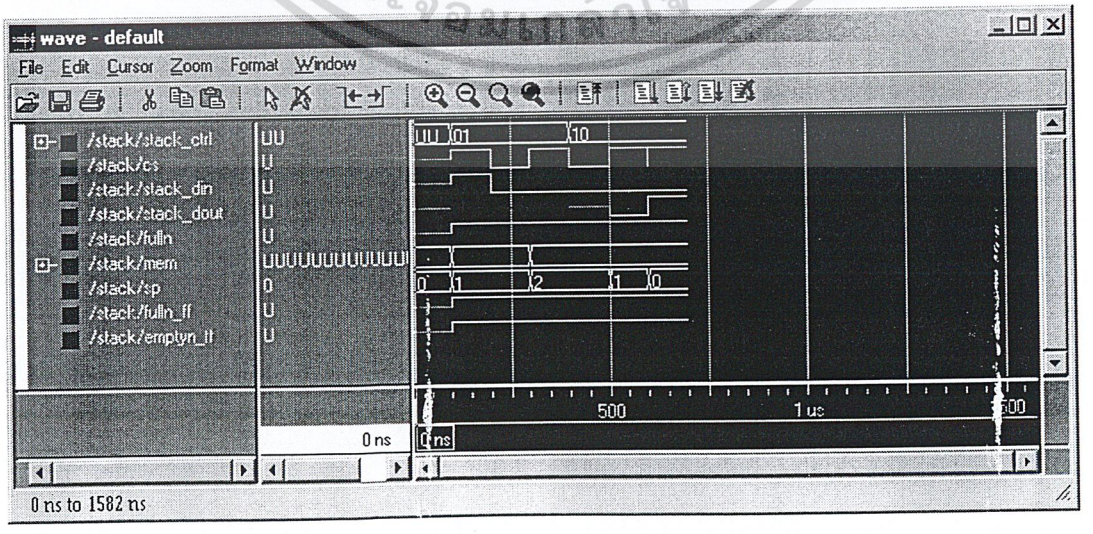
  Signals in Design คือการเลือกดูสัญญาณทั้งหมดในดีไซน์ รวมทั้งสัญญาณ ภายในที่สร้างขึ้นในการดีไซน์ด้วย

การทดลองใส่ค่าสัญญาณลงในดีไซน์สามารถทำได้โดย

เลือกเมนู Edit → Force.... ในหน้าต่างแสดงค่าสัญญาณของดีไซน์ของผู้ใช้



รูปที่ 5.5 แสดงหน้าจอแสดงสัญญาณ และการเลือกดูคลื่นของสัญญาณ



รูปที่ 5.6 แสดงหน้าจอแสดงแบบของคลื่นสัญญาณ

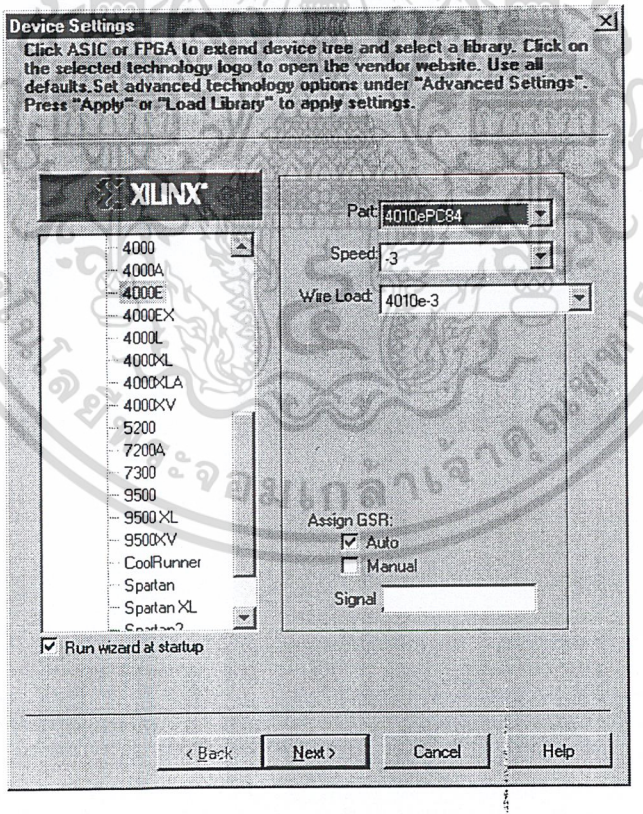
5.4 การใช้งานโปรแกรม Leonardo Spectrum v2000.1a RC

เมื่อได้ออกแบบวงจรดิจิตอลโดยใช้ภาษา VHDL และรวมถึงการตรวจสอบวงจรในระดับพฤติกรรมอย่างเป็นที่น่าพอใจแล้ว ขั้นตอนต่อไปคือคือ การนำไฟล์ภาษา VHDL ของดีไซน์ไปผ่านขบวนการสังเคราะห์วงจร สำหรับขั้นตอนนี้ โดยใช้เครื่องมือสังเคราะห์ (Synthesis Tool) ที่มีชื่อว่า Leonardo Spectrum ซึ่งเป็นซอฟต์แวร์ของบริษัท Exemplar Logic Inc. ตามที่ได้เคยกล่าวถึงไปแล้ว

แรกสุดในการทำการสังเคราะห์ (Synthesis) ให้ทำการคลิ้กที่ปุ่ม Synthesis Wizard  ที่แถบเครื่องมือจะขึ้นหน้าต่าง Wizard ให้เลือกค่าต่าง ๆ เพื่อทำการสังเคราะห์ดีไซน์ของเรา

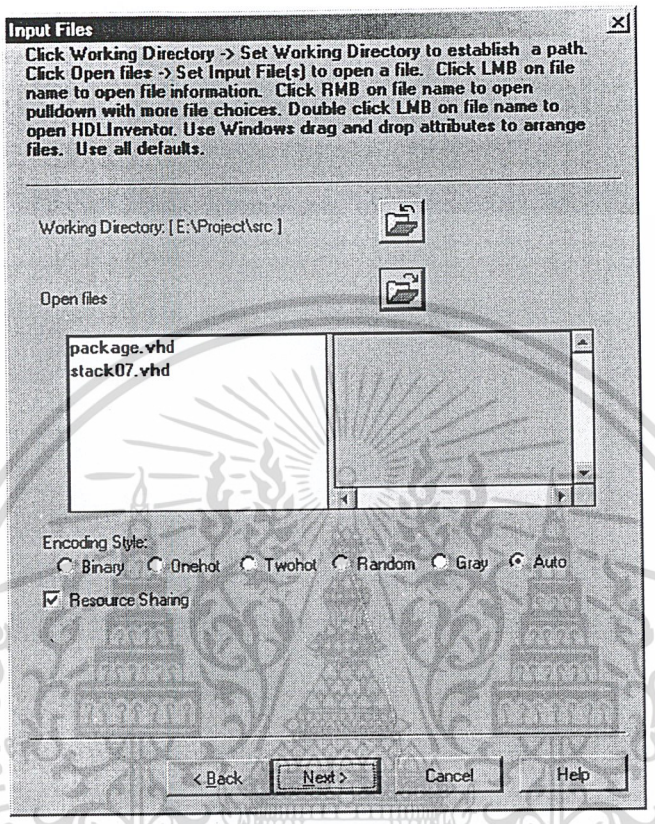
1. เลือกเทคโนโลยีที่ต้องการโดยมีสองไลบรารี่ให้เลือก ได้แก่ ASIC เป็นไลบรารี่ของอุปกรณ์ ASIC (Application Specific Integrated Circuits) เทคโนโลยี และ FPGA/CPLD เป็นไลบรารี่ของอุปกรณ์ FPGA (Field Programmable Gate Array) หรือ CPLD (Complex Programmable Logic Device) ตระกูลต่าง ๆ ดังรูปที่ 5.7

   ให้ทำการเลือกเทคโนโลยี FPGA/CPLD → Xilinx → 400E และที่ช่อง Part ให้เลือก 4010ePC84 ซึ่งเป็นตัวที่ใช้ในโครงงานนี้



รูปที่ 5.7 แสดงหน้าจอการเลือกเทคโนโลยีที่ใช้ในการสังเคราะห์

2. กำหนดไดเรกตอรี่ที่ทำงานและโค้ดที่ต้องการสังเคราะห์ โดยเราสามารถเลือกโค้ดที่ต้องการสังเคราะห์ได้จำนวนมากกว่าหนึ่งโค้ดต่อครั้งโดยโค้ดที่อยู่ล่างสุดจะถือเป็นโค้ดหลักของดีไซน์ ดังรูปที่ 5.8



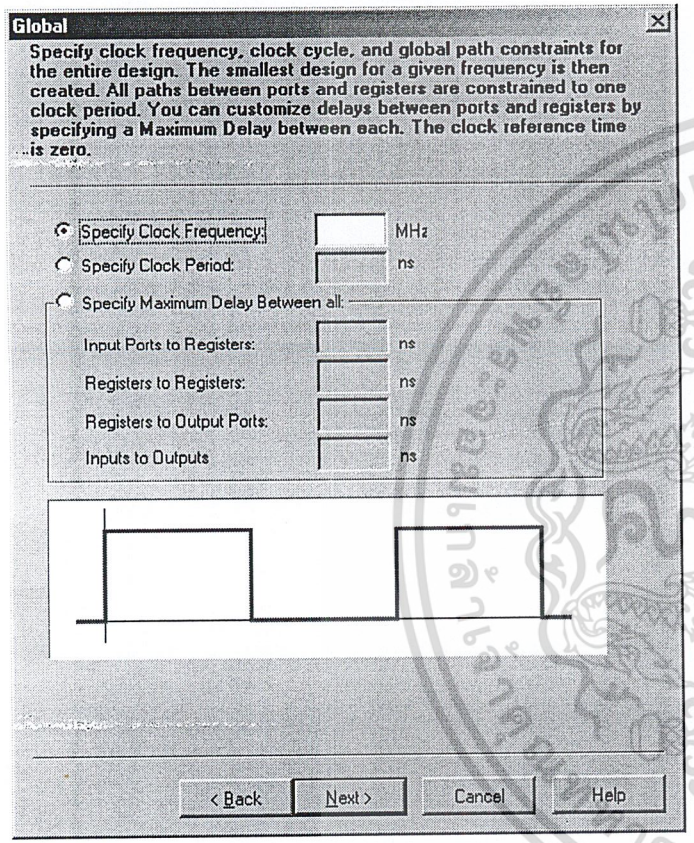
รูปที่ 5.8 แสดงหน้าจอการเลือกโค้ดที่ต้องการสังเคราะห์

3. กำหนดความเร็วของวงจรที่ต้องการสังเคราะห์ โดยในโครงงานนี้ไม่ต้องทำการใส่ค่าลงในช่องนี้ โดยจะเป็นการตั้งค่าตามความสามารถของ FPGA ที่เลือกเอง ดังรูปที่ 5.9

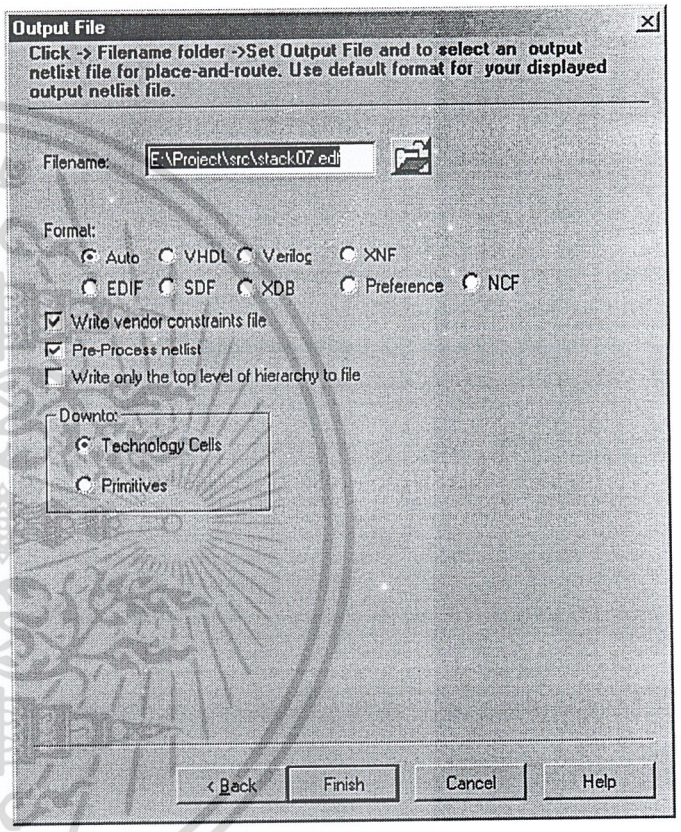
4. กำหนดรูปแบบของไฟล์เอาต์พุต โดยให้เลือก Format เป็น EDIF หรือ Auto เพื่อทำการสังเคราะห์เป็นวงจรในระดับเกต ดังรูปที่ 5.10

   ตัวเลือก Downto ให้เลือกแบบ Technology Cells ซึ่งจะเป็นการเลือก Format แบบ EDIF หรือ XNF ส่วนตัวเลือก Primitives จะเป็นการเลือก Format แบบ VHDL

5. คลิกที่ Finish เพื่อเริ่มทำการสังเคราะห์วงจร ถ้าผลการสังเคราะห์ดีไซน์ที่ออกแบบพบปัญหาใด ๆ โปรแกรมจะทำการแจ้งปัญหาและทำการหยุดการสังเคราะห์ให้เพื่อทำการแก้ไขต่อไป แต่ถ้าผลการสังเคราะห์ดีไซน์ที่ออกแบบไม่พบปัญหาใด ๆ โปรแกรมจะรายงานผลการสังเคราะห์วงจรที่ได้ เช่น ขนาดของวงจรที่สังเคราะห์ ความเร็วของวงจร เป็นต้น



รูปที่ 5.9 แสดงหน้าจอการใส่ค่าเพื่อกำหนดความเร็วของวงจร



รูปที่ 5.10 แสดงหน้าจอการเลือกรูปแบบของไฟล์เอาต์พุต

ผลลัพธ์ที่ได้จากการสังเคราะห์จะเก็บอยู่ในไฟล์ผลลัทธ์ (Summary file) ซึ่งจะมีนามสกุลเป็น ".sum" โดยเราสามารถใช้ซอฟต์แวร์ Text editor ต่าง ๆ เปิดดูได้ในภายหลัง

```
-- Start optimization for design .work.stack.rtl
Using wire table: 4010e-3

      Pass     Area     Delay     DFFs  PIs   POs --CPU--
               (FGs)    (ns)                    min:sec
      1         62       32        24     4     2   00:02
Info, Added global buffer BUFG for port cs
Using wire table: 4010e-3
-- Start timing optimization for design .work.stack.rtl
No critical paths to optimize at this level
```

รูปที่ 5.11 แสดงขนาดและ Delay ของวงจรที่ได้จากการสังเคราะห์

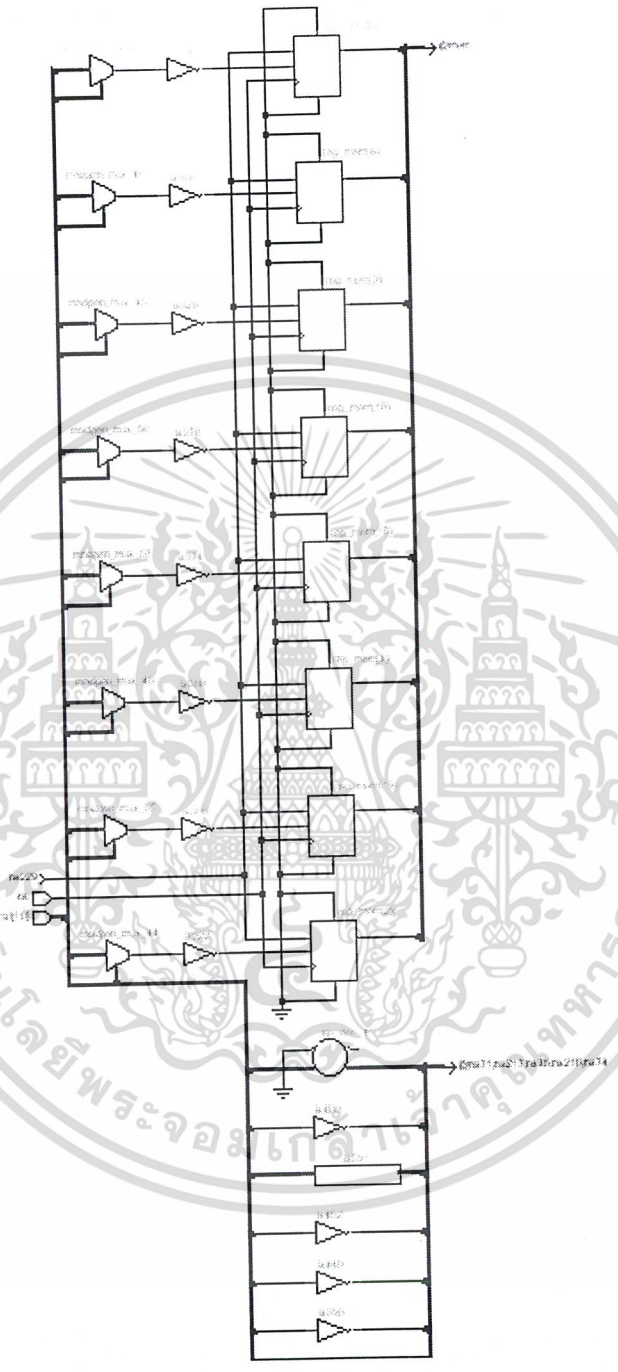
```
*******************************************************

Cell: stack     View: rtl     Library: work

*******************************************************

 Number of ports :                  6
 Number of nets :                 103
 Number of instances :             99
 Number of references to this view :  0

Total accumulated area :
 Number of BUFG :                   1
 Number of CLB Flip Flops :        23
 Number of FG Function Generators :  62
 Number of H Function Generators :   7
 Number of IBUF :                   3
 Number of IOB Input Flip Flops :   1
 Number of OBUF :                   1
 Number of OBUFT :                  1
 Number of Packed CLBs :           32

*********************************************
```

รูปที่ 5.12 แสดงอุปกรณ์ต่าง ๆ ที่ใช้ในการออกแบบวงจรที่สังเคราะห์ขึ้น

```
*********************************************
Device Utilization for 4010ePC84
*********************************************
Resource                Used    Avail    Utilization
-------------------------------------------------
IOs                     6       61         9.84%
FG Function Generators  62      800        7.75%
H Function Generators   7       400        1.75%
CLB Flip Flops          23      800        2.88%
```

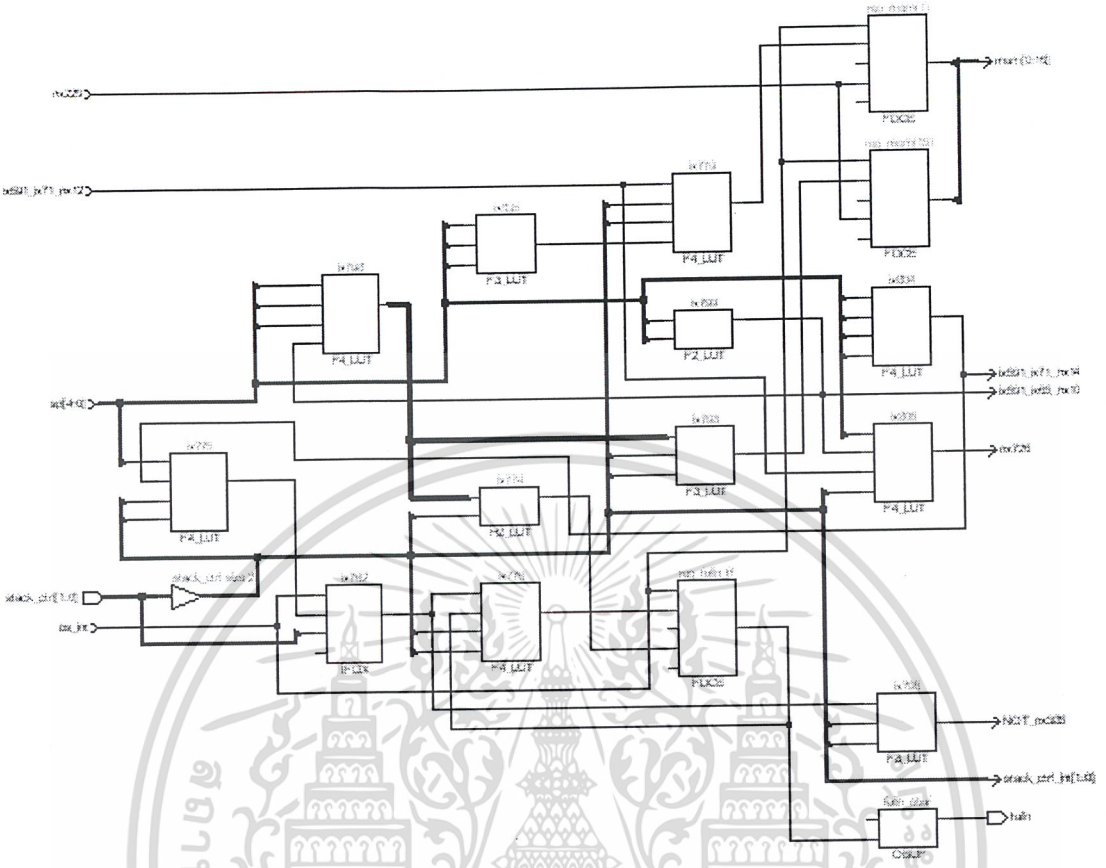
รูปที่ 5.13 แสดงผลรวมของอุปกรณ์ที่ใช้เทียบตามขนาดของชิพที่เลือกใช้

6. การเลือกดูหน้าตาของวงจรระดับ RTL สามารถได้โดยคลิกที่ปุ่ม  (View RTL Schematics)



รูปที่ 5.14 แสดงภาพ RTL ของวงจรที่ถูกสังเคราะห์ขึ้น

7. การเลือกดูหน้าตาวงจรในระดับเทคโนโลยีของ FPGA สามารถเลือกได้โดยคลิ้กที่ปุ่ม  (View Technology Schematics)

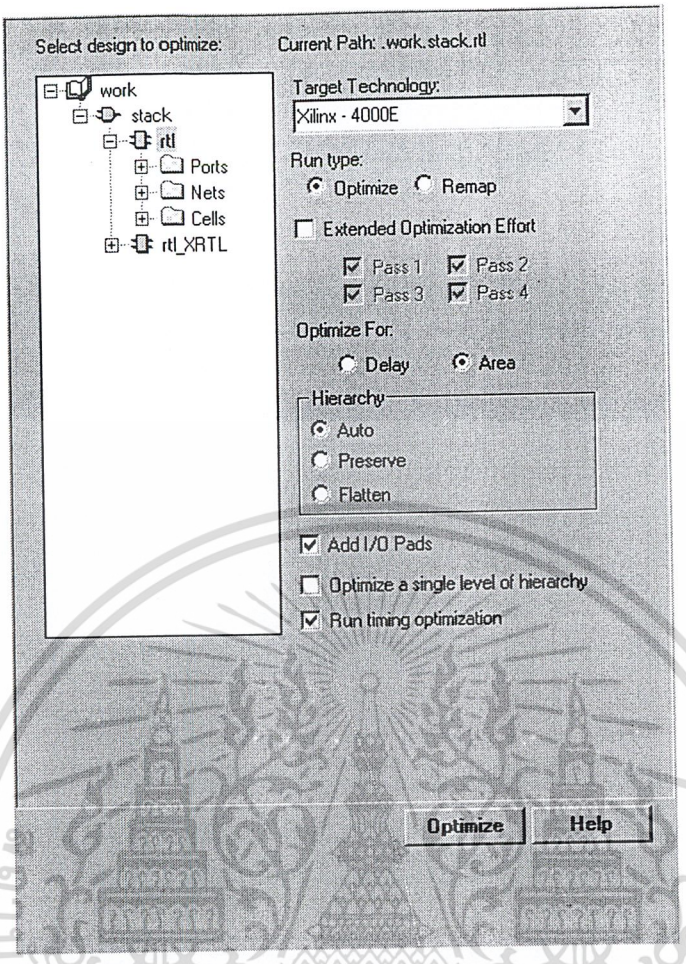


รูปที่ 5.15 แสดงภาพวงจรในระดับเทคโนโลยีของ FPGA

8. การเลือกดูเส้นทางเดินวิกฤต (Critical Path) สามารถทำได้โดยคลิ้กที่ปุ่ม  (View Critical Path Schematics) เส้นทางวิกฤตคือเส้นทางที่มีดีเลย์ไทม์มากที่สุดในวงจร

รูปที่ 5.16 แสดงภาพเส้นทางเดินวิกฤตของวงจร

9. สามารถทำการ Optimize วงจรที่ออกแบบเพื่อให้ได้ผลการสังเคราะห์ที่ได้ออกมาดีที่สุด การ Optimize สามารถทำได้โดยเลือก Optimize ที่ Flow TAB



รูปที่ 5.17 แสดงหน้าจอการ Optimize โดยเลือก Optimize ที่ Flow TAB

- Run type เลือกเป็น Optimize เพื่อต้องการ Optimize หรือเลือกที่ Remap เพื่อต้องการสร้างวงจรใหม่

- Extended Optimization Effort เลือกเพื่อต้องการให้ทำการทดลอง Optimize จำนวน 1 - 4 ครั้งเพื่อหาค่าที่ดีที่สุด

- Optimize For เป็นการเลือกเพื่อให้วงจรที่ได้มีความเร็วสูงสุด (Delay) หรือวงจรที่ได้มีขนาดเล็กที่สุด (Area)

- Hierarchy เลือกเพื่อต้องการ Optimize เฉพาะบางส่วน (Preserve) หรือทั้งหมด (Flatten)

หลังจากการสังเคราะห์วงจรเรียบร้อยแล้วผู้ใช้สามารถนำ Netlist ของวงจรที่ได้เพื่อนำไป Place & Route สู่ชิพต่อไป โดยรูปแบบที่นิยมใช้กันอย่างแพร่หลายจะมี 2 รูปแบบคือ EDIF (Electronics Data Interchange Format) และอีกรูปแบบหนึ่งคือ XNF (Xilinx Netlist Format) โดยในโครงงานนี้จะใช้รูปแบบ XNF เนื่องจากว่าในโครงงานนี้ใช้ชิพ FPGA ของ XILINX นอกจากนี้โปรแกรม Exemplar Leonardo Spectrum ยังสามารถเขียนวงจรที่ผ่านการสังเคราะห์เรียบร้อยแล้วกลับเป็น VHDL หรือ Verilog และเขียนข้อมูล Delay ภายในรูปแบบของ SDF (Standard Delay Format) เพื่อใช้ในการ Simulation ในระดับ Timing ต่อไป

```
LCANET, 5
PART, 4010ePC84-4
USER, LIBRARY_NAME, work
USER, CELL_NAME, alu
USER, VIEW_NAME, rtl
PROG, LeonardoSpectrum Level 3, v2000_1a.45, "09/18/01 08:51:55"
EXT, aop<5>, I,
EXT, aop<4>, I,
EXT, aop<3>, I,
EXT, aop<2>, I,
EXT, aop<1>, I,
EXT, aop<0>, I,
EXT, cs, I,
EXT, alu_src_a, I,
EXT, alu_src_b, I,
EXT, acc, O,
SYM, alu_src_b_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, alu_src_b,
PIN, O, O, alu_src_b_int, 3.375000
END
SYM, alu_src_a_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, alu_src_a,
PIN, O, O, alu_src_a_int, 3.375000
END
SYM, aop<0>_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, aop<0>,
PIN, O, O, aop<0>_int, 3.375000
END
SYM, aop<1>_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, aop<1>,
PIN, O, O, aop<1>_int, 3.375000
END
SYM, aop<2>_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, aop<2>,
PIN, O, O, aop<2>_int, 3.375000
END
SYM, aop<3>_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, aop<3>,
PIN, O, O, aop<3>_int, 3.375000
END
SYM, aop<4>_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, aop<4>,
PIN, O, O, aop<4>_int, 3.375000
END
SYM, aop<5>_ibuf, IBUF, LIBVER=2.0.0, SCHNM=IBUF
PIN, I, I, aop<5>,
PIN, O, O, aop<5>_int, 3.375000
END
SYM, reg_acc_tmp, OUTFF, INIT=R, LIBVER=2.0.0, SCHNM=OFDX
PIN, C, I, cs_int,
PIN, Q, O, acc, 8.775000
PIN, CE, I, nx24,
PIN, D, I, nx28,
END
SYM, cs_ibuf, BUFG, LIBVER=2.0.0
PIN, I, I, cs,
PIN, O, O, cs_int, 6.344998
END
SYM, ix64, EQN, EQN=((~I0*I1)), LIBVER=2.0.0
PIN, I0, I, aop<5>_int,
PIN, I1, I, nx55,
PIN, O, O, nx24,
END
SYM, ix65, EQN, EQN=((~I0*~I1*~I2*I3)), LIBVER=2.0.0
PIN, I0, I, aop<5>_int,
PIN, I1, I, aop<4>_int,
PIN, I2, I, aop<3>_int,
PIN, I3, I, nx52,
```

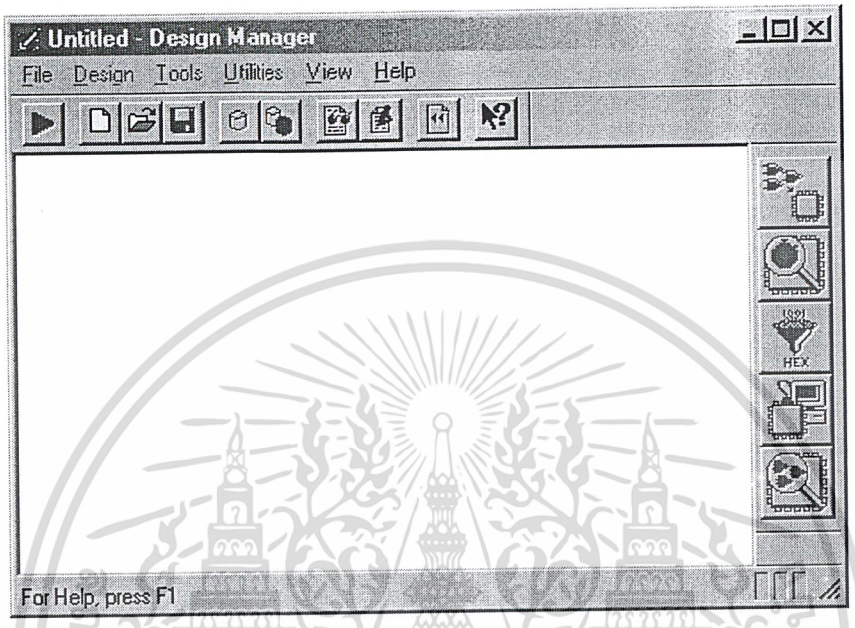
รูปที่ 5.18-1 แสดงตัวอย่างไฟล์ XNF

```
69 PIN, O, O, nx28,
70 END
71 SYM, ix66, EQN, EQN=((I0*I1)+(~I0*I2)), LIBVER=2.0.0
72 PIN, I0, I, aop<2>_int,
73 PIN, I1, I, nx53,
74 PIN, I2, I, nx54,
75 PIN, O, O, nx52,
76 END
77 SYM, ix67, EQN, EQN=((~I0*I2)+(~I0*~I1*I3)+(~I0*I1*~I3)), LIBVER=2.0.0
78 PIN, I0, I, aop<1>_int,
79 PIN, I1, I, aop<0>_int,
80 PIN, I2, I, alu_src_a_int,
81 PIN, I3, I, alu_src_b_int,
82 PIN, O, O, nx53,
83 END
84 SYM, ix68, EQN, EQN=((I0*~I1*I2*I3)+(I0*I1*I2*~I3)), LIBVER=2.0.0
85 PIN, I0, I, aop<1>_int,
86 PIN, I1, I, aop<0>_int,
87 PIN, I2, I, alu_src_a_int,
88 PIN, I3, I, alu_src_b_int,
89 PIN, O, O, nx54,
90 END
91 SYM, ix69, EQN, EQN=((~I0*~I1*~I2*I3)+(~I0*~I1*I2*~I3)), LIBVER=2.0.0
92 PIN, I0, I, aop<4>_int,
93 PIN, I1, I, aop<3>_int,
94 PIN, I2, I, aop<2>_int,
95 PIN, I3, I, aop<1>_int,
96 PIN, O, O, nx55,
97 END
98 SYM, HMAP_0, HMAP, LIBVER=2.0.0
99 PIN, I1, I, aop<5>_int
00 PIN, I2, I, nx55
01 PIN, O, I, nx24
02 END
03 SYM, FMAP_1, FMAP, LIBVER=2.0.0
04 PIN, I1, I, aop<5>_int
05 PIN, I2, I, aop<4>_int
06 PIN, I3, I, aop<3>_int
07 PIN, I4, I, nx52
08 PIN, O, I, nx28
09 END
10 SYM, HMAP_2, HMAP, LIBVER=2.0.0
11 PIN, I1, I, aop<2>_int
12 PIN, I2, I, nx53
13 PIN, I3, I, nx54
14 PIN, O, I, nx52
15 END
16 SYM, FMAP_3, FMAP, LIBVER=2.0.0
17 PIN, I1, I, aop<1>_int
18 PIN, I2, I, aop<0>_int
19 PIN, I3, I, alu_src_a_int
20 PIN, I4, I, alu_src_b_int
21 PIN, O, I, nx53
22 END
23 SYM, FMAP_4, FMAP, LIBVER=2.0.0
24 PIN, I1, I, aop<1>_int
124 PIN, I1, I, aop<1>_int
125 PIN, I2, I, aop<0>_int
126 PIN, I3, I, alu_src_a_int
127 PIN, I4, I, alu_src_b_int
128 PIN, O, I, nx54
129 END
130 SYM, FMAP_5, FMAP, LIBVER=2.0.0
131 PIN, I1, I, aop<4>_int
132 PIN, I2, I, aop<3>_int
133 PIN, I3, I, aop<2>_int
134 PIN, I4, I, aop<1>_int
135 PIN, O, I, nx55
136 END
137 EOF
```

รูปที่ 5.18-2 แสดงตัวอย่างไฟล์ XNF (ต่อ)

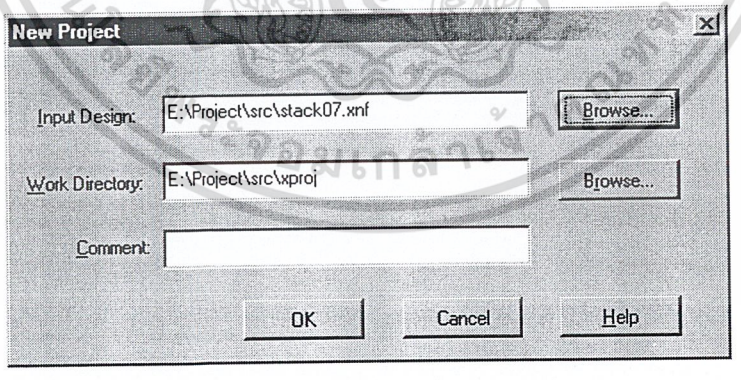
5.5 การใช้งานโปรแกรม Xilinx Design Manager v1.4.12

เมื่อทำการสังเคราะห์วงจรจากโปรแกรม Exemplar logic Leonardo Spectrum โดยผลการสังเคราะห์จะถูกเขียนไว้ในรูปของเน็ตลิสต์แบบ XNF (Xilinx Netlist Format) ขั้นตอนต่อไปจะนำไฟล์เน็ตลิสต์ไปจัดวางวงจร (Place & Route) โดยใช้โปรแกรม Xilinx Design Manager



รูปที่ 5.19 แสดงหน้าจอเริ่มต้นโปรแกรม Xilinx Design Manager

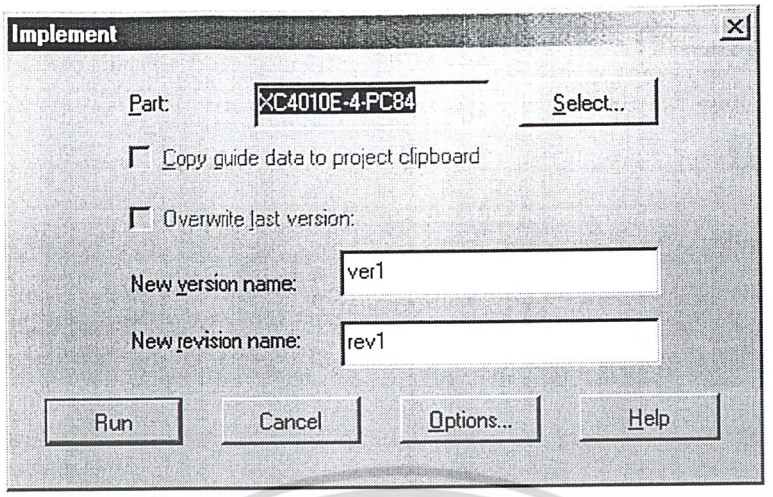
1. สร้างโปรเจคใหม่สำหรับนำดีไซน์ที่ผ่านการสังเคราะห์แล้วเพื่อที่จะนำไป Place & Route ได้โดยเลือก File → New Project ดังรูปที่ 5.20



รูปที่ 5.20 แสดงหน้าจอ New Project

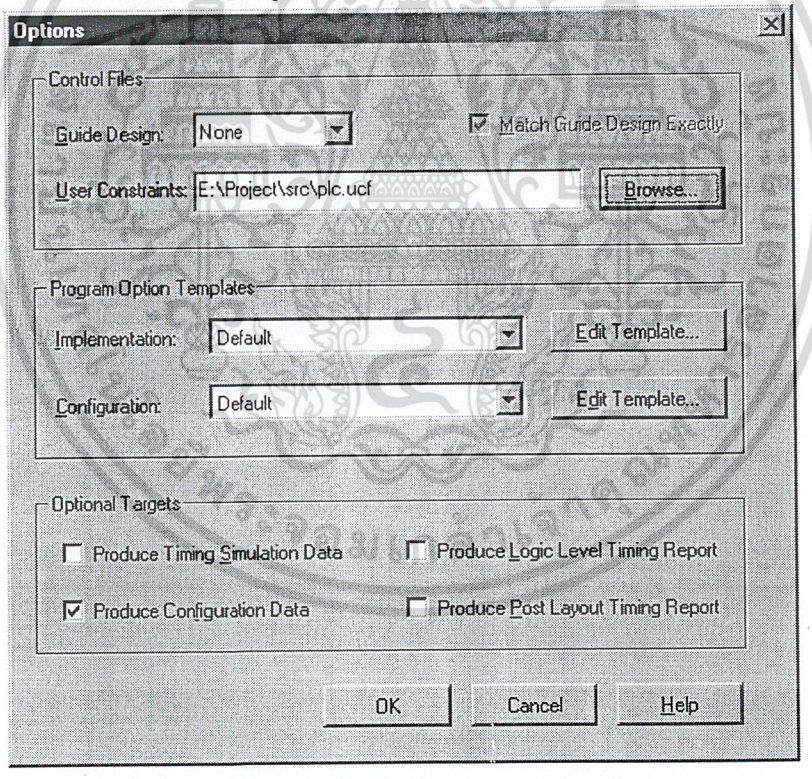
ในช่อง Input Design ให้เลือก Browse เพื่อเลือกดีไซน์ที่ต้องการนำมาใส่โดยดีไซน์ที่เลือกได้ต้องอยู่ในรูปของไฟล์ EDIT, XNF หรือ PLD เท่านั้น

2. ทำการอิมพลีเมนท์ (Implement) โดยเลือกเมนู Design → Implement หรือคลิ้กที่ ▶ จะปรากฏหน้าจอดังรูปที่ 5.21



รูปที่ 5.21 แสดงหน้าจอการเลือกอิมพลีเมนท์

3. ทำการกำหนดข้อบังคับการตั้งขาของอุปกรณ์ต่าง ๆ (Constraint) โดยเลือกที่ Option ในหน้าจอการอิมพลีเมนท์ ดังรูปที่ 5.22



รูปที่ 5.22 แสดงหน้าจอตั้งค่าต่าง ๆ (Option) ของการอิมพลีเมนท์

ในช่อง User Constraints ให้เลือก Browse เพื่อทำการกำหนดค่าบังคับโดยให้เลือกไฟล์ที่มีนามสกุล UCF ซึ่งผู้ใช้ต้องทำการสร้างเอง ตัวอย่างไฟล์ UCF ดังรูปที่ 5.23

```
net clk loc=p13;
net I(0) loc=p44;
net I(1) loc=p45;
net I(2) loc=p46;
net I(3) loc=p47;
net I(4) loc=p48;
net reset loc=p49;
net o(6) loc=p19;
net o(5) loc=p23;
net o(4) loc=p26;
net o(3) loc=p25;
net o(2) loc=p24;
net o(1) loc=p18;
net o(0) loc=p20;
net address(14) loc=p60;
net address(13) loc=p58;
net address(12) loc=p50;
net address(11) loc=p56;
net address(10) loc=p51;
net address(9) loc=p57;
net address(8) loc=p59;
net address(7) loc=p84;
net address(6) loc=p83;
net address(5) loc=p82;
net address(4) loc=p79;
net address(3) loc=p78;
net address(2) loc=p5;
net address(1) loc=p4;
net address(0) loc=p3;
net dbus(7) loc=p10;
net dbus(6) loc=p80;
net dbus(5) loc=p81;
net dbus(4) loc=p35;
net dbus(3) loc=p38;
net dbus(2) loc=p39;
net dbus(1) loc=p40;
net dbus(0) loc=p41;
net cs loc=p65;
net oe loc=p61;
net we loc=p62;
```

รูปที่ 5.23 ตัวอย่างไฟล์ UCF

UCF (User Constraints File) เป็นไฟล์ที่ใช้ในการกำหนดค่าที่ใช้บังคับว่า อินพุต และเอาต์พุตของวงจรที่เราดีไซน์จะให้เป็นขาของอินพุต และเอาต์พุตใดในตัวของ FPGA

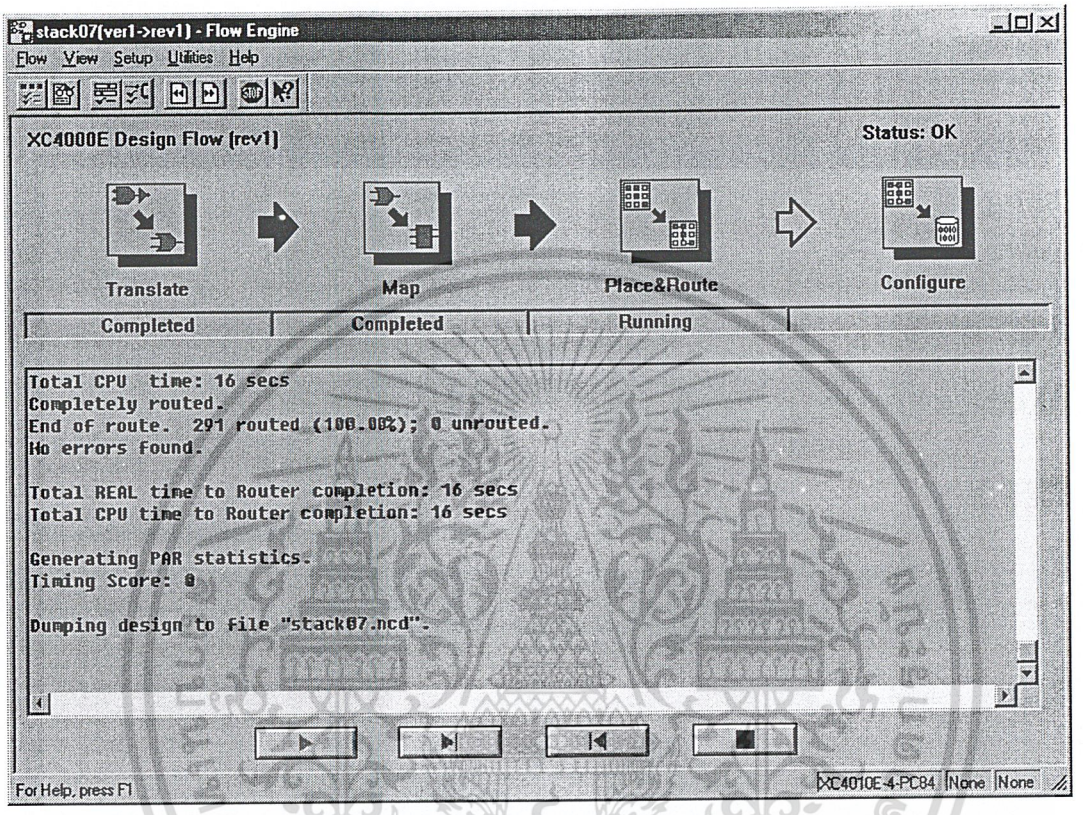
net (I/O ในดีไซน์) loc=(ขาใน FPGA)

รูปที่ 5.24 รูปแบบของไฟล์ UCF

โดยขาของ FPGA ที่จะนำมาใช้งานได้นั้นจะสามารถดูได้จากรูปที่ 3.8 และ จากตาราง ที่ 3.4

เมื่อทำการกำหนดไฟล์ UCF เสร็จแล้วให้เลือก OK แล้วเลือก Run เพื่อทำการอิมพลีเมนท์ต่อไป

4. Flow Engine จะประกอบด้วยขั้นตอนย่อยดังต่อไปนี้ Translate → Map → Place & Route → Configure การดำเนินการจะดำเนินการไปตามลำดับ



รูปที่ 5.25 แสดงหน้าจอของ Flow Engine

Translate

Translate คือขั้นตอนที่โปรแกรมอ่านเน็ตลิสต์ไฟล์ของดีไซน์ตามที่ได้กำหนดไว้แล้วแปลงให้อยู่ในรูปแบบข้อมูลของ Xilinx เอง (ไฟล์ NGO) โดยใช้คำสั่ง ngdbuild ซึ่งจะแปลงให้เป็นเน็ตลิสต์แบบ NGO (Native Generic Database) แล้วผ่านต่อไปยังขั้นตอนต่อไป

Map

ที่เกี่ยวข้องกับการจัดการฮาร์ดแวร์ภายในของอุปกรณ์เป้าหมาย (Resource Allocation) รวมถึงการจัดแบ่งลอจิกในลอจิกบล็อก เช่น CLB (Configuration Logic Block) และ IOB (Input Output Block) และผลการทำงานที่ได้จะถูกเก็บบันทึกไว้ในไฟล์ที่ชื่อว่า map.ncd เพื่อนำไปใช้ในขั้นตอน Place & Route ต่อไป

Place & Route

ขั้นตอนการทำ Place & Route เริ่มจากการอ่านข้อมูลของวงจรไฟล์ map.ncd และไฟล์ PCF (Physical Constraint File) ซึ่งภายในมีข้อบังคับต่าง ๆ ที่โปรแกรม PAR จะต้องคำนึงถึงเมื่อทำขั้นตอน Place & Route งานขั้นแรก คือ การวางลอจิกบล็อกลงใน FPGA ก่อน ซึ่งเป็นหน้าที่ของ Placer และถ้ามีการกำหนดข้อบังคับในเรื่องเวลาแล้วโปรแกรม Placer ก็จะ

พยายามวางบล็อกเหล้านั้นให้อยู่ใกล้ ๆ กันเพื่อลดความล่าช้าของสัญญาณระหว่างบล็อกต่าง ๆ และผลการกระทำในแต่ละครั้งจะมีแต้มคะแนน (Placer Score) และครั้งที่มีแต้มคะแนนน้อยที่สุดถือว่าดีที่สุด และจบท้ายด้วยการเชื่อมโยงบล็อกต่าง ๆ ที่ถูกวางลงในตำแหน่งที่เหมาะสมแล้วโดย Router ซึ่งขั้นตอนนี้จะทำการหลายครั้งเพื่อพยายามเชื่อมโยงเส้นทางภายในของวงจรให้ได้ตามข้อกำหนด

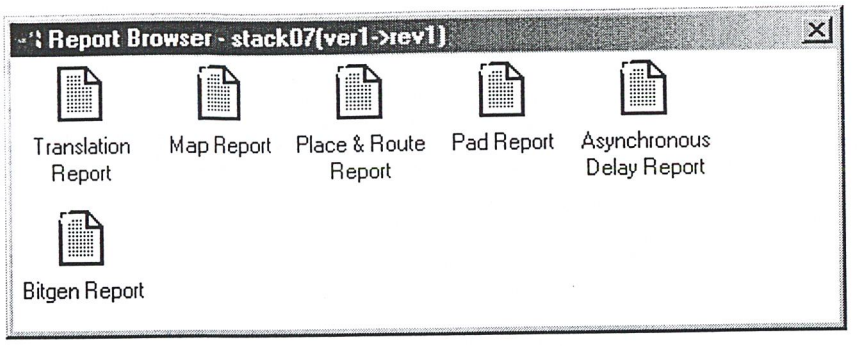
Configure

เมื่อผ่านขั้นตอน Place & Route แล้วถือว่าการออกแบบสร้างดีไซน์สำหรับ FPGA ได้เสร็จสิ้นลงแล้ว และเหลือเพียงขั้นตอน Configure ซึ่งใช้ในการแปลงข้อมูลของวงจรที่ได้ให้อยู่ในรูปแบบของบิตสตรีม (Bit stream) เพื่อนำไปดาว์นโหลดลงใน FPGA ต่อไปโดยข้อมูลนี้จะถูกบันทึกลงในไฟล์ที่มีนามสกุล BIT

รูปที่ 5.26 แสดงหน้าจอของโปรแกรมเมื่อทำงานเสร็จสิ้น

ในระหว่างการทำขั้นตอนแต่ละขั้นตอนเริ่มตั้งแต่ Translate ไปจนถึง Configure ผู้ใช้สามารถหยุดการทำงานของ Flow Engine อย่างชั่วคราวได้ และสามารถสั่งให้ทำต่อไปได้ถ้าต้องการ โดยกดปุ่ม Stop หรือ Run ตามลำดับ

ถ้าผลการอิมพลีเมนท์เกิดความผิดพลาดที่ขั้นตอนใดขั้นตอนหนึ่งใน Flow Engine โปรแกรมจะหยุดการทำงานทันที และเราสามารถที่จะดูผลการทำงานได้โดยคลิ้กที่ปุ่ม Report เพื่อดูผลการรายงาน หรือคลิ้กที่ปุ่ม Flow Engine เพื่อดูปัญหาว่าเกิดที่ขั้นตอนใดก็ได้



รูปที่ 5.27 แสดงหน้าจอการเลือกแสดงผลรายงานแบบต่าง ๆ

5.6 การใช้งานโปรแกรม XSTools Utilities และการใช้งานบอร์ด XS40

เมื่อเราทำการอิมพลีเมนท์ดีไซน์วงจรของเราจนได้ไฟล์สกุล BIT แล้วขั้นตอนต่อไปในการนำลงชิพ FPGA จะเป็นการใช้โปรแกรม XSTools ของบริษัท XESS Corp. ซึ่งการใช้งานโปรแกรม XSTools เราจำเป็นต้องเข้าใจถึงหลักการโปรแกรมของ FPGA (ดังที่กล่าวไปแล้วในบทที่ 3) ซึ่งบอร์ดที่ใช้ในการทำโครงงานคือ บอร์ด XS40 ของบริษัท XESS Corp. และการใช้งานของบอร์ด XS40 ไปพร้อม ๆ กัน

ตารางที่ 5.1 แสดงเบอร์ของพินที่ทำการต่อเข้ากับแหล่งจ่ายไฟขนาดต่าง ๆ

XS40 Board Type	GND Pin	+5V Pin	+3.3V Pin
XS40-005E V1.4	52	2, 54	None
XS40-005XL V1.4	52	2	54
XS40-010E V1.4	52	2, 54	None
XS40-010XL V1.4	52	2	54
XSP-010 V1.4	52	2, 54	none

5.6.1 การจ่ายไฟให้แก่บอร์ด XS40

การจ่ายไฟให้แก่บอร์ด XS40 สามารถทำได้ 2 วิธีการคือ

1. ต่อทางหม้อแปลง

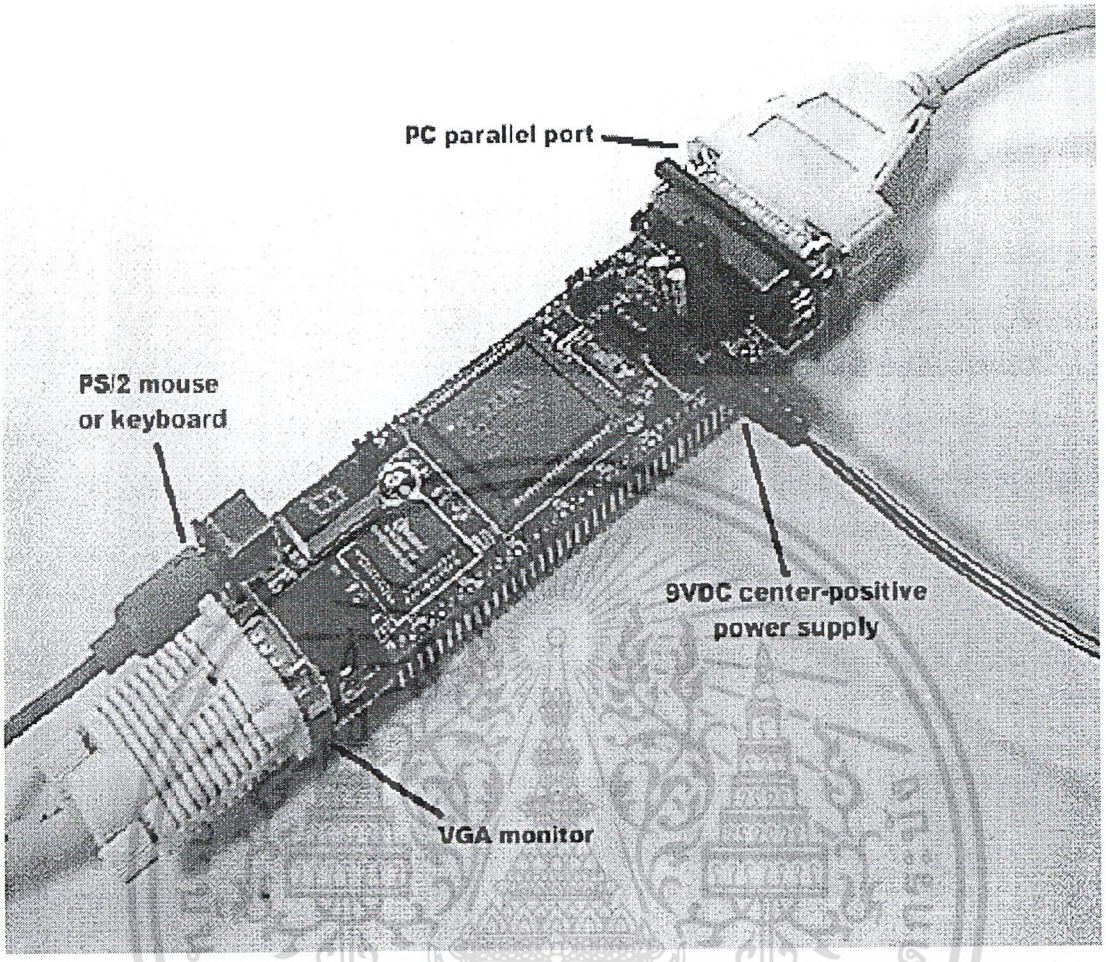
   ในการใช้งานบอร์ด XS40 อย่างแรกสุดไม่ควรวางบอร์ดในพื้นที่ที่มีไฟฟ้าสถิตย์อยู่ จากนั้นต่อไฟเข้าที่จุด J9 ของบอร์ด XS40 (ดูรูปที่ 5.30) โดยใช้สายไฟขนาด 2.1 mm จ่ายไฟ DC 9V จากหม้อแปลงต่อเข้าทีจุดนี้

2. ต่อตรง

   เราสามารถจ่ายไฟได้อีกทางหนึ่งโดยทางพินจำนวนสองแถวบนบอร์ด XS40 โดยทำการป้อนไฟโดยตรงลงทางพินเหล่านั้น โดยใช้ไฟขนาด +5V, +3.3V และ GND ซึ่งแตกต่างกันไปตามเบอร์ของชิพ FPGA ที่นำมาใช้ ในที่นี้ใช้ชิพเบอร์ XS40-010E

5.6.2 การต่อบอร์ด XS40 เข้ากับเครื่องคอมพิวเตอร์

เราสามารถต่อบอร์ด XS40 เข้ากับเครื่องคอมพิวเตอร์ได้ทางพอร์ตขนาน (Parallel port) ของเครื่องคอมพิวเตอร์ หรือทางสายเชื่อมต่อ DB-25 โดยต่อเข้ากับทางจุด J1 ทางหัวของบอร์ด XS40 ดังรูปที่ 5.28



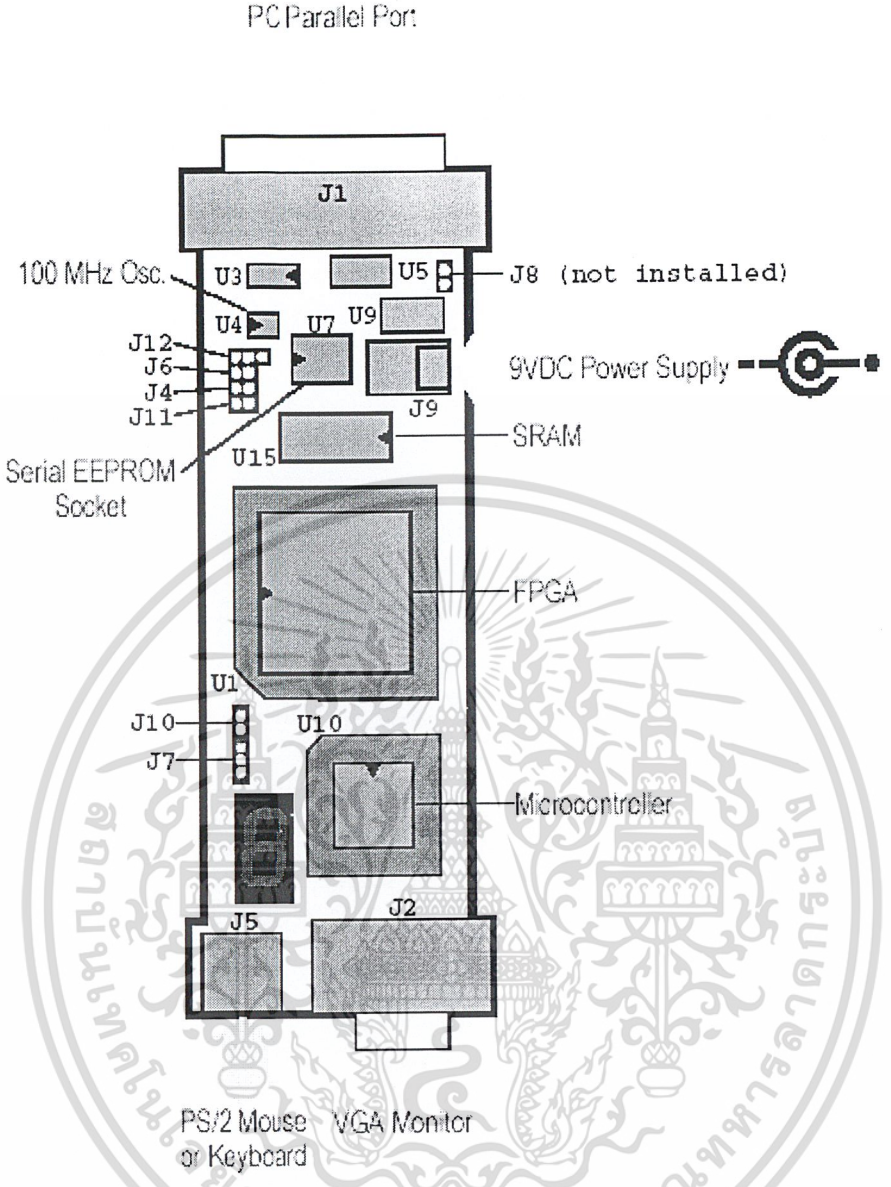
รูปที่ 5.28 แสดงการเชื่อมต่อภายนอกของบอร์ด XS40

5.6.3 การต่อบอร์ด XS40 เข้ากับจอมอนิเตอร์

เราสามารถแสดงผลออกทางจอมอนิเตอร์แบบ VGA (VGA monitor) ได้โดยใช้สายเชื่อมต่อขนาด 15 พินเข้าที่จุด J2 ทางท้ายของบอร์ด XS40 ดังรูปที่ 5.28 การแสดงผลออกเป็นกราฟฟิกต้องทำการติดตั้งไดร์เวอร์ VGA ลงบนบอร์ด XS40 ก่อน โดยเราสามารถหาไดร์เวอร์ได้ที่ http://www.xess.com/ho03000.html

5.6.4 การติดตั้งเมาส์ หรือคีย์บอร์ดเข้ากับบอร์ด XS40

เราสามารถรับอินพุตจากทางคีย์บอร์ด หรือเมาส์ได้โดยทำการต่อเข้าที่จุด J5 (PS/2) ที่ทางท้ายของบอร์ด XS40 ดังรูปที่ 5.28 โดยเราสามารถหาไดร์เวอร์ได้ที่ http://www.xess.com/ho03000.html



รูปที่ 5.29 แสดงการจัดวางส่วนประกอบต่าง ๆ ของบอร์ด XS40

5.6.5 การติดตั้งจัมเปอร์บนบอร์ด XS40

ตำแหน่งมาตรฐานของจัมเปอร์ (Jumper) แสดงดังตารางที่ 5.2 เราสามารถตั้งค่าของจัมเปอร์ใหม่ในการทำงานของเราได้ดังต่อไปนี้

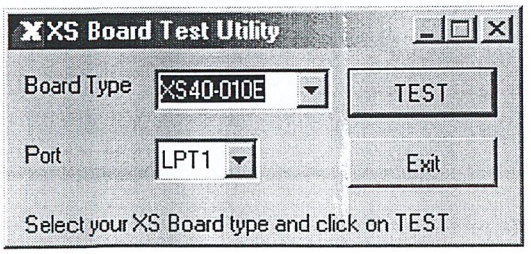
- ใช้บอร์ด XS40 อย่างเดียวโดยไม่ทำการติดต่อกับเครื่องคอมพิวเตอร์ทางพอร์ตขนาน
- ทำการตั้งค่าความถี่นาฬิกา (Clock frequency) ใหม่
- ทำการอ่านโค้ดจาก ROM ภายในแทนการอ่านจาก SRAM ภายนอกบนบอร์ด XS40

ตารางที่ 5.2 แสดงตำแหน่งการใช้งานต่าง ๆ ของจัมเปอร์บนบอร์ด XS40

จัมเปอร์	ตำแหน่ง	การใช้งาน
J4	On (default)	เมื่อต้องการดาว์นโหลดลงบอร์ด XS40 ผ่านทางพอร์ตขนาน
	Off	เมื่อต้องการใช้บอร์ดโดยโหลดค่าจาก EEPROM (U7) แบบอนุกรม
J6	On	ใช้เมื่อ EEPROM (U7) ถูกใช้งาน
	Off (default)	ใช้เมื่อใช้งานบอร์ดตามปกติ
J7	1-2 (ext) (default)	วางที่ตำแหน่งพิน 1 และ 2 (ext) เมื่อมีการใช้งานโปรแกรมไมโครคอนโทรเลอร์ 8031 บน SRAM (U8) ขนาด 32 Kbyte ภายนอกของบอร์ด XS40
	2-3 (int)	วางที่ตำแหน่งพิน 2 และ 3 (in) เมื่อโปรแกรมถูกใช้งานภายใน
J8	On	เมื่อใช้แรงดัน 3.3V บน FPGA เบอร์ XC4000XL
	Off	เมื่อใช้แรงดัน 5V บน FPGA เบอร์ XC400E
J10	On	เมื่อมีการใช้ค่าจาก EEPROM
	Off (default)	เมื่อมีการใช้งานพอร์ตขนานในการติดตั้งค่า
J11	On (default)	เมื่อมีการใช้งานพอร์ตขนานในการติดตั้งค่า
	Off	เมื่อมีการใช้ค่าจาก EEPROM
J12	1-2 (osc) (default)	วางที่ตำแหน่งพิน 1 และ 2 (osc) เมื่อทำการใช้งานสัญญาณนาฬิกาจากออซซิเลเตอร์
	2-3 (set)	วางที่ตำแหน่งพิน 2 และ 3 (set) เมื่อต้องการโปรแกรมค่าความถี่ที่ต้องการลงในออซซิเลเตอร์

5.6.6 การทดสอบบอร์ด XS40

เมื่อเราทำการตั้งค่าต่าง ๆ ของจัมเปอร์บนบอร์ด XS40 เสร็จเรียบร้อยแล้ว และทำการติดตั้งสายเข้ากับพอร์ตขนาน และ ติดตั้งการจ่ายไฟให้กับบอร์ดก่อน เราสามารถทดสอบบอร์ดได้โดยใช้โปรแกรม GXSTEST ของ XSTools



รูปที่ 5.30 แสดงหน้าจอโปรแกรม GXSTEST

ผู้ใช้สามารถเลือกพอร์ตที่จะทำการดาว์นโหลดลงบอร์ด XS40 ได้โดยเลือกที่ Port โดยปกติจะเป็นการใช้งานที่พอร์ต LPT1 เมื่อทำการเลือกพอร์ตเสร็จเรียบร้อยแล้วขั้นตอนต่อไปคือการเลือกชนิดของ XS บอร์ดที่ต้องการทดสอบโดยเลือกที่ Board Type ในที่นี้ทำการเลือกบอร์ดเป็น XS40-010E

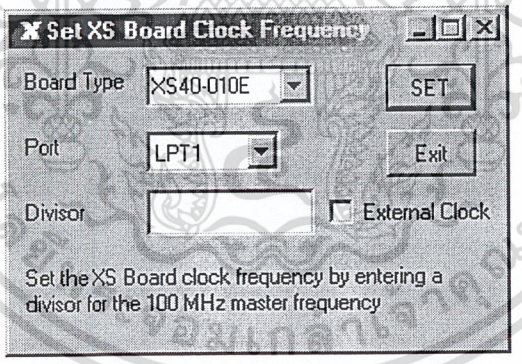
ทำการคลิ้กที่ปุ่ม TEST เพื่อเริ่มต้นทำการทดสอบ โดยโปรแกรม GXSTEST จะทำการโปรแกรมลงบนไมโครคอนโทลเลอร์ และ FPGA บนบอร์ดเพื่อทำการทดสอบ ระยะเวลาในการทดสอบแต่ละครั้งจะประมาณ 15 วินาทีสำหรับบอร์ด XS40 สถานะการทดสอบจะแสดงบนด้านล่างของกรอบโปรแกรม และเมื่อทำการทดสอบเสร็จสิ้นจะแสดง 0 บนหน้าจอ LED และแสดง E เมื่อการทดสอบผิดพลาด และโปรแกรมจะแสดงค่าบนหน้าจอว่าผลการทดสอบบอร์ดสามารถใช้งานได้หรือไม่

ถ้าผลการทดสอบรายงานว่าบอร์ดไม่สามารถใช้งานได้ ให้ทำการตรวจจุดเชื่อมต่อต่าง ๆ รวมถึงการตั้งค่าตำแหน่งของจัมเปอร์ จากนั้นให้ทำการทดสอบกับเครื่องคอมพิวเตอร์เครื่องอื่น ซึ่งผลการทดสอบส่วนใหญ่ปัญหาที่พบมักมาจากสายที่ทำการเชื่อมต่อกับพอร์ตขนาน

5.6.7 การกำหนดค่าความถี่สัญญาณนาฬิกาของบอร์ด XS40

บอร์ด XS40 ภายในจะมีออซซิเลเตอร์ (Oscillator) ที่สามารถโปรแกรมได้ในย่านความถี่ไม่เกิน 100 MHz ซึ่งการกำหนดค่าจะกำหนดโดยการตั้งค่าตัวหารความถี่ตั้งแต่ 1 – 2052 โดยจะได้ค่าความที่ 100 MHz ถึง 48.7 KHz ซึ่งจะถูกนำไปใช้งานใน FPGA ต่อไป

เมื่อมีการนำแหล่งจ่ายไฟออกจากบอร์ด XS40 ผู้ใช้ต้องทำการตั้งค่าความถี่สัญญาณนาฬิกาใหม่ทุกครั้ง การตั้งค่าสามารถตั้งค่าได้โดยโปรแกรม GXSSETCLK

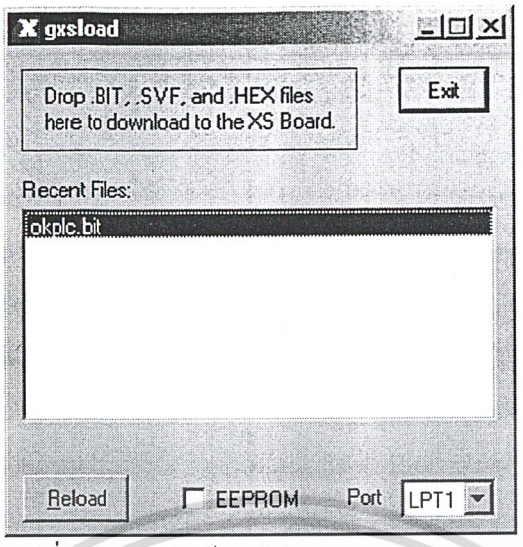


รูปที่ 5.31 แสดงหน้าจอโปรแกรม GXSSETCLK

การใช้งานโปรแกรมทำได้โดย เลือกพอร์ตที่ทำการเชื่อมต่อที่ช่อง Port และชนิดของบอร์ดที่ช่อง Board Type และทำการใส่ค่าที่ต้องการใช้ในการหารความถี่โดยค่าที่ใส่ต้องอยู่ระหว่าง 1 – 2052 ช่อง External Clock ใช้สำหรับเมื่อต้องการใช้สัญญาณนาฬิกาภายนอกในการใช้งาน FPGA แทนสัญญาณนาฬิกาจากออซซิเลเตอร์ภายใน โดยเราสามารถต่อสัญญาณนาฬิกาภายนอกเข้าทีบอร์ด XS40 ทางพิน 64

5.6.8 การนำดีไซน์ที่ออกแบบไว้ลงบอร์ด XS40

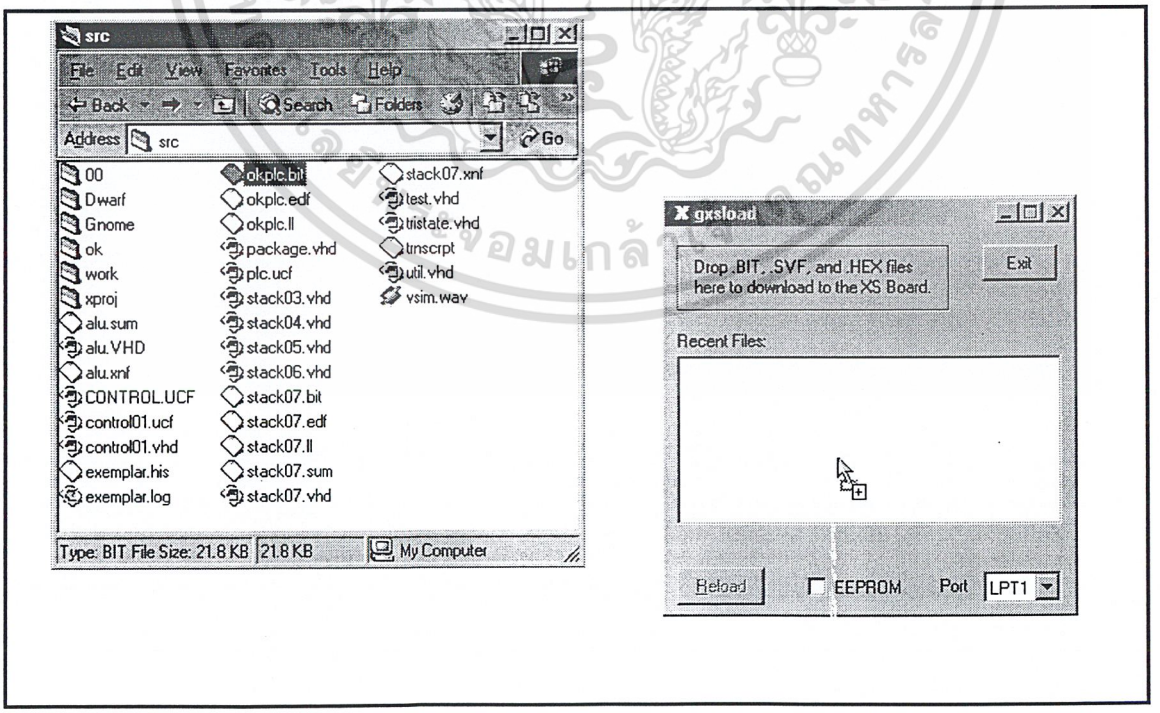
เราสามารถดาว์นโหลดดีไซน์ลงใน FPGA บนบอร์ด XS40 ได้โดยใช้โปรแกรม GXSLOAD



รูปที่ 5.32 แสดงหน้าจอโปรแกรม GXSLOAD

ในการใช้งานโปรแกรม แรกสุดเราต้องทำการเลือกพอร์ตที่ใช้ในการติดต่อกับบอร์ด XS40 โดยปกติเป็นพอร์ต LPT1 ถ้าต้องการใช้งานบอร์ด XS40 โดยใช้งานบอร์ดจาก EEPROM ให้ทำการเลือกที่ช่อง EEPROM

เราสามารถดาว์นโหลดไฟล์สกุล BIT และ HEX ลงยังบอร์ดโดยทำการคลิ้กและลากไฟล์ที่ต้องการจาก Windows Explorer ลงยังหน้าต่างของโปรแกรม GXSLOAD ได้ (drag & drop) ไฟล์สกุล BIT เป็นไฟล์ที่เก็บบิทสตรีม (Bit stream) ซึ่งเป็นตัวบอกลักษณะการกำหนดค่าต่าง ๆ ของ FPGA และไฟล์สกุล HEX จะบรรจุข้อมูลที่ต้องการเก็บลงสน RAM บนบอร์ด XS40 ซึ่งข้อมูลในไฟล์จะต้อง



รูปที่ 5.33 แสดงการลากไฟล์เพื่อโปรแกรมลง GXSLOAD

อยู่ในรูปแบบเลขฐานสิบหก (Intel-format hexadecimal) เมื่อเราทำการปล่อยปุ่มเมาส์ลงบนหน้าจอของโปรแกรม GXSLOAD แล้วโปรแกรมจะทำการส่งข้อมูลในไฟล์ลงไปยังบอร์ด XS40 ผ่านทางพอร์ตขนาน ถ้าไฟล์ที่ถูกนำมาปล่อยไม่ใช่ไฟล์ที่สามารถดาวน์โหลดลงยังบอร์ดได้ (ได้แก่สกุลอื่นที่ไม่ใช่ BIT, SVF หรือ HEX) โปรแกรมจะไม่ทำการส่งไปยังบอร์ด

ระหว่างการทำงาน โปรแกรม GXSLOAD จะแสดงสถานะการทำงาน และเมื่อทำการดาวน์โหลดลงบอร์ดเสร็จสิ้นโปรแกรมจะแสดงปุ่ม Reload ขึ้นที่หน้าต่าง เราสามารถทำการดาวน์โหลดดีไซน์ที่เคยทำการโปรแกรมลงไปแล้วใหม่ได้โดยการคลิ้กที่ปุ่มนี้ และในการดาวน์โหลดไฟล์เราสามารถที่จะทำการโหลดได้มากกว่า 1 ไฟล์พร้อม ๆ กัน

5.6.9 การเก็บดีไซน์ของวงจรไว้ในบอร์ด XS40 ตลอดเวลา

ปกติในบอร์ด XS40 เมื่อทำการดาวน์โหลดดีไซน์ลงยังบอร์ด ค่าต่าง ๆ จะถูกเก็บลงใน SRAM ซึ่งเมื่อทำการถอดแหล่งจ่ายไฟออกจะทำให้ค่าต่าง ๆ เหล่านี้หายไป ซึ่งการเก็บค่าต่าง ๆ ของดีไซน์ไว้เราสามารถเก็บไว้ใน EEPROM (U7) แทนได้ ซึ่งจะถูกนำมาใช้ใหม่ทุกครั้งที่ทำการติดตั้งบอร์ดใหม่ ซึ่งการใช้งานจะใช้ไอซี XILINX XC1700 ซึ่งเป็นไอซี EEPROMs แบบอนุกรม การใช้งานในรูปแบบนี้จะเป็นการใช้งานแบบโปรแกรมครั้งเดียว (OPT: One-Time Programmable) คือต้องทำการเปลี่ยนชิพใหม่ทุกตรั้งที่ต้องการเปลี่ยนดีไซน์ของวงจรใน FPGA ตารางที่ 5.3 จะแสดงชนิดของ EEPROM ที่เหมาะสมในการใช้งานกับบอร์ด XS40 ชนิดต่าง ๆ

XS40 Board Type	Bitstream Size	XILINX EEPROM
XS40-005E	95,008	XC17128E
XS40-005XL	151,960	XC17256E
XS40-010E	178,144	XC17256E
XS40-010XL	283,424	XC1701
XSP-010	95,008	XC17S10

ตารางที่ 5.3 แสดงชนิดของ EEPROM ที่เหมาะกับแต่ละชนิดของบอร์ด

ในการดาวน์โหลดดีไซน์ลงใน EEPROM เราจะใช้โปรแกรม GXSLOAD เช่นกัน โดยต้องทำการเลือกที่ EEPROM และลากไฟล์สกุล BIT ลงในหน้าต่างของ GXSLOAD โปรแกรมจะแสดงขั้นตอนการทำงานต่าง ๆ ขึ้นมา และผู้ใช้จะต้องทำการตั้งค่าใช้งาน EEPROM โดยการกำหนดตำแหน่งของจัมเปอร์บนบอร์ดด้วย

ขั้นตอนการดาวน์โหลดดีไซน์ลงบอร์ด EEPROM

1). ถอดสายจ่ายไฟ และสายต่อขนานออกจากจุดต่อ J1 บนบอร์ด

2). ทำการเปลี่ยนตำแหน่งจัมเปอร์ J10 เพื่อทำการอนุญาตให้ทำงานแบบอนุกรมได้ ซึ่งก็คือการส่งสัญญาณนาฬิกาไปยัง EEPROM เพื่อทำการโหลดค่าจาก EEPROM ลง FPGA ตามสัญญาณนาฬิกา

3). ถอดจัมเปอร์ J4 และ J11 ออกเพื่อป้องกันการชนกันของสัญญาณนาฬิกา และข้อมูลจากคอมพิวเตอร์กับ FPGA

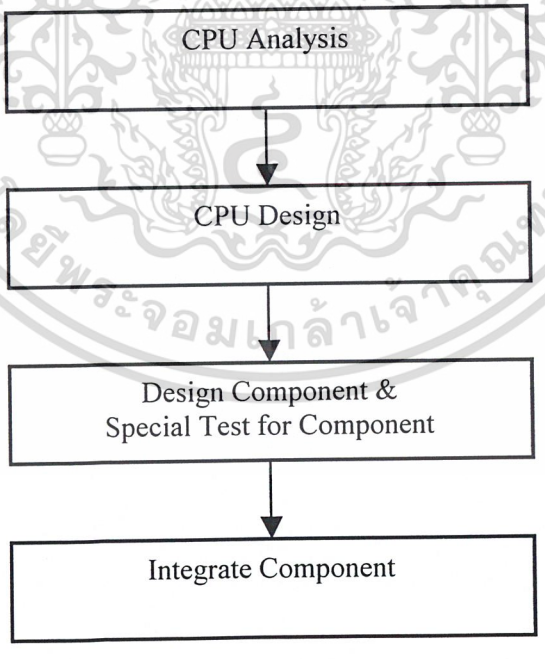
4). ต่อสายต่าง ๆ กลับยังบอร์ด XS40

บทที่ 6
การออกแบบวงจร CPU ของ PLC

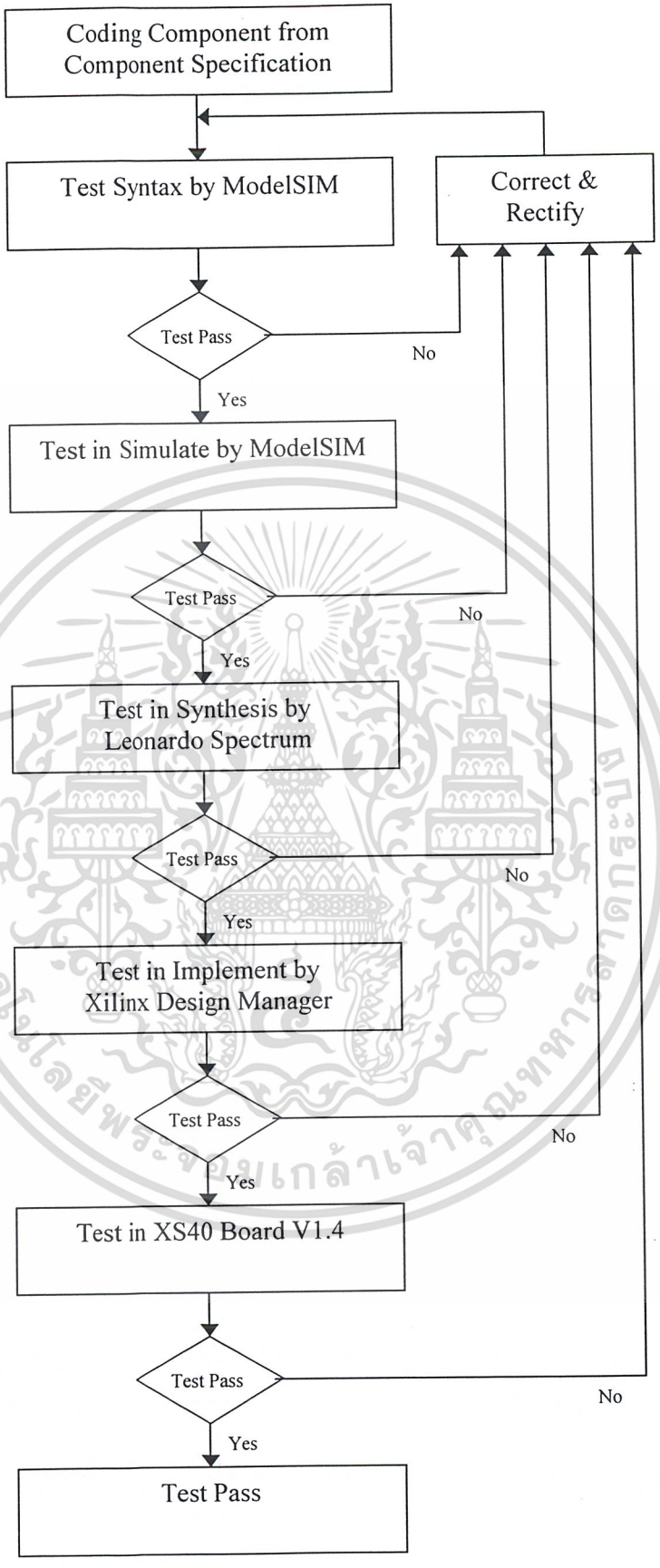
PLC เป็นอุปกรณ์ควบคุมที่ใช้สำหรับควบคุมการทำงานของอุปกรณ์ไฟฟ้า ซึ่งส่วนประกอบที่สำคัญของ PLC ได้แก่ CPU หรือตัวประมวลผลในการทำงาน สำหรับในภาคการศึกษานี้นี้ จะทำการออกแบบ CPU ของ PLC โดยจะอ้างอิงการทำงานต่าง ๆ จาก PLC ของ Omron ในรุ่น C20 โดยใช้ภาษา VHDL เป็นภาษาที่ใช้ในการบรรยายการทำงานของวงจร จากนั้นทำการซิมมูเลทด้วยโปรแกรม ModelSIM และทำการสังเคราะห์ต่อด้วยโปรแกรม Leonardo Spectrum และทำการนำไฟล์เน็ตลิสต์ที่ได้ไปจัดวางวงจรโดยใช้โปรแกรม Xilinx Design Manager จากนั้นในขั้นตอนสุดท้ายจึงเป็นการนำชิพ FPGA โดยใช้ใช้โปรแกรม XSTools โดยบทนี้จะทำการอธิบายถึงขั้นตอนในการออกแบบ CPU ของ PLC

6.1 ขั้นตอนการออกแบบ CPU ของ PLC

ในการออกแบบชิ้นงาน สิ่งแรกคือการทำความเข้าใจในชิ้นงานที่ต้องทำการออกแบบเสียก่อน สำหรับ PLC จะมี CPU เป็นหน่วยประมวลผลกลาง ที่ทำหน้าที่นำโปรแกรมภาษาแลดเดอร์ของผู้ใช้มาปฏิบัติเพื่อควบคุมอุปกรณ์ภายนอกตามเงื่อนไขการควบคุมที่ผู้เขียนโปรแกรมต้องการควบคุมการติดต่อรับส่งข้อมูลระหว่างอินพุตและเอาต์พุต โดยจะมีขั้นตอนสำหรับการออกแบบ CPU ดังรูปที่ 6.1 และ 6.2



รูปที่ 6.1 แสดงขั้นตอนในการออกแบบ CPU



รูปที่ 6.2 แสดงขั้นตอนในการทำ Coding Component

6.2 การวิเคราะห์ส่วนประกอบของ CPU

เป็นการระบุถึงขั้นตอนการทำงาน ส่วนประกอบ และความสามารถในการทำงานของ CPU โดยจะดูจากคำสั่ง (Instructions) ต่าง ๆ ของ PLC

สำหรับ PLC รุ่นที่ทำการศึกษาจะแบ่งการการทำงานหลักออกเป็น 3 โหมด คือ Run, Monitor และ Program ซึ่งเราจะทำการวิเคราะห์เพียงในโหมด Run เท่านั้น เนื่องจาก PLC ที่ทำการออกแบบจะต้องสามารถทำงานได้เหมือน การทำงานของ PLC ในขณะที่อยู่ใน โหมด Run

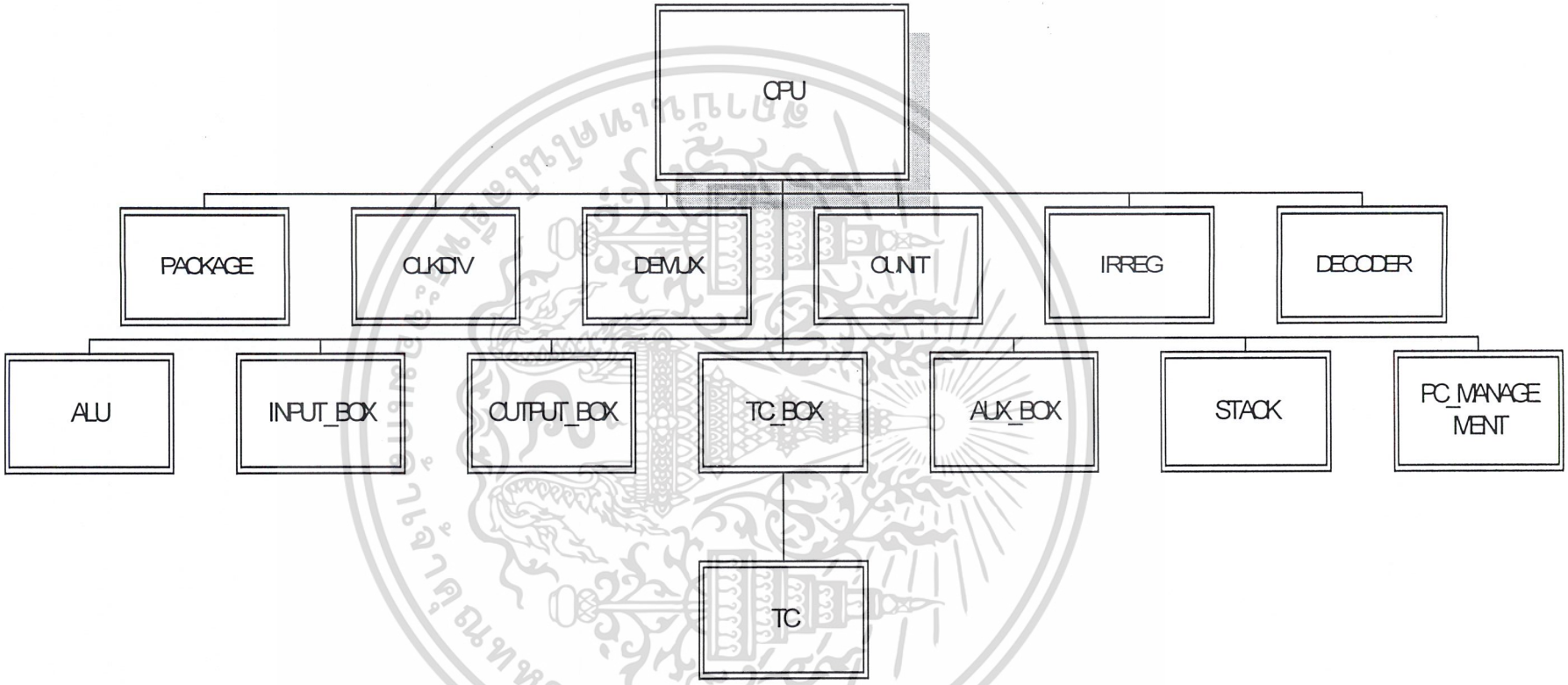
ขนาดและความสามารถของ PLC ที่ทำการออกแบบในโครงงานนี้คือ

- อินพุตจำนวน 5 อินพุต
- เอาต์พุตจำนวน 7 เอาต์พุต
- รีเลย์ช่วย (Auxiliary Relay) จำนวน 64 ตัว
- ไทมเมอร์ หรือเคาท์เตอร์ จำนวน 4 ตัวซึ่งทำงานร่วมกัน โดยทำงานเป็นไทมเมอร์ หรือเคาท์เตอร์ โดยไทมเมอร์ จะต้องสามารถนับเวลาได้ละเอียดสุดที่หน่วย เดซิวินาที (0.1วินาที) ทั้งไทมเมอร์ และ เคาท์เตอร์ สามารถนับได้ตั้งแต่หน่วย 0 ถึง 16383

ความสามารถของส่วนประมวลผล โดยต้องสามารถประมวลผลคำสั่งต่าง ๆ ได้ดังต่อไปนี้

- LD หรือ LOAD ทำการนำค่าในรีเลย์มาใช้เป็นอินพุต
- LDI หรือ LOAD Inverse ทำการนำค่าในรีเลย์ ที่ผ่านการ Inverse แล้ว มาใช้เป็นอินพุต
- AND ทำการ AND ค่าอินพุตหนึ่งกับอีกอินพุตหนึ่ง
- ANI หรือ AND Inverse ทำการ AND ค่าอินพุตหนึ่งกับอีกอินพุตหนึ่งที่ผ่านการ Inverse แล้ว
- OR ทำการ OR ค่าอินพุตหนึ่งกับอีกอินพุตหนึ่ง
- ORI หรือ OR Inverse ทำการ OR ค่าอินพุตหนึ่งกับอีกอินพุตหนึ่งที่ผ่านการ Inverse แล้ว
- OUT ทำการนำค่าในรีเลย์ออกไปยังเอาต์พุต
- OUI หรือ OUT Inverse ทำการนำค่าในรีเลย์ที่ผ่านการ Inverse ออกไปยังเอาต์พุต
- TIM ทำการตั้งค่าในหน่วยไทม์เมอร์
- CNT ทำการตั้งค่าในหน่วยเคาท์เตอร์
- ANB หรือ AND Block ทำการต่ออนุกรมบล็อคของวงจร
- ORB หรือ OR Block ทำการต่อขนานบล็อคของวงจร
- NOP หรือ No Operation ทำการผ่านค่าการทำงานไป หรือทำการระบุให้ CPU ไม่ทำงานคำสั่งนี้
- END คำสั่งจบการทำงาน

ความสามารถในการอ้างอิงหน่วยความจำ (ความจุสูงสุดของคำสั่งที่จะทำงานได้) สามารถอ้างหน่วยความจำภายนอกได้สูงสุด 32 Kbytes

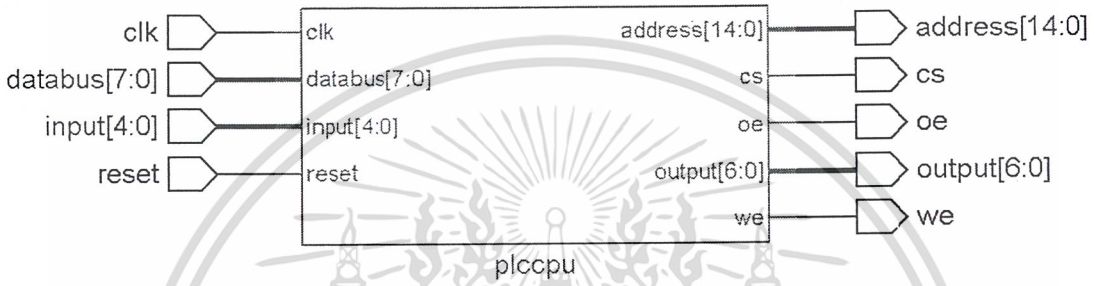


รูปที่ 6.3 แสดงโครงสร้างของ CPU ใน PLC

6.3 การออกแบบ CPU และส่วนประกอบต่าง ๆ ของ CPU

เมื่อได้ลักษณะ (Specification) ของ CPU ที่เราต้องการแล้ว ขั้นตอนต่อมาคือขั้นตอนในการออกแบบในส่วนวงจรของ CPU โดยเริ่มต้นจากการออกแบบภาพรวมของ CPU ก่อนว่าควรประกอบด้วยส่วนประกอบ (Component) อะไรบ้าง และลักษณะของคำสั่งที่ใช้ในการทำงาน

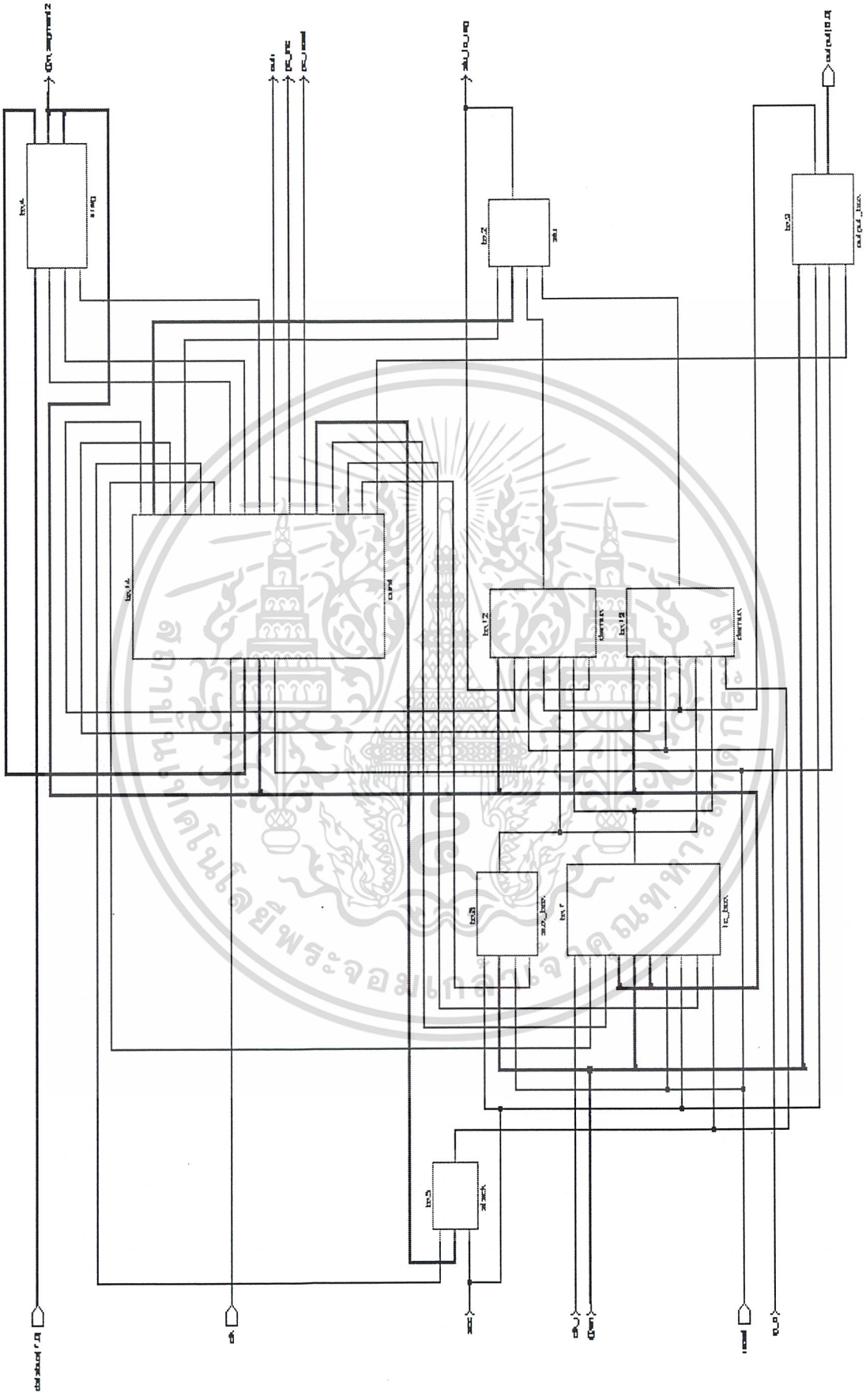
ลักษณะนอกสุดของ CPU ที่ทำการออกแบบจะประกอบด้วยขาสัญญาณอินพุต และเอาต์พุต ได้แก่ ขาสัญญาณ Clock, Reset, Input และ Output และขาสัญญาณที่ทำหน้าที่ติดต่อกับส่วนของหน่วยความจำได้แก่ Address Bus, Data Bus และสัญญาณควบคุม CE, OE และ WE



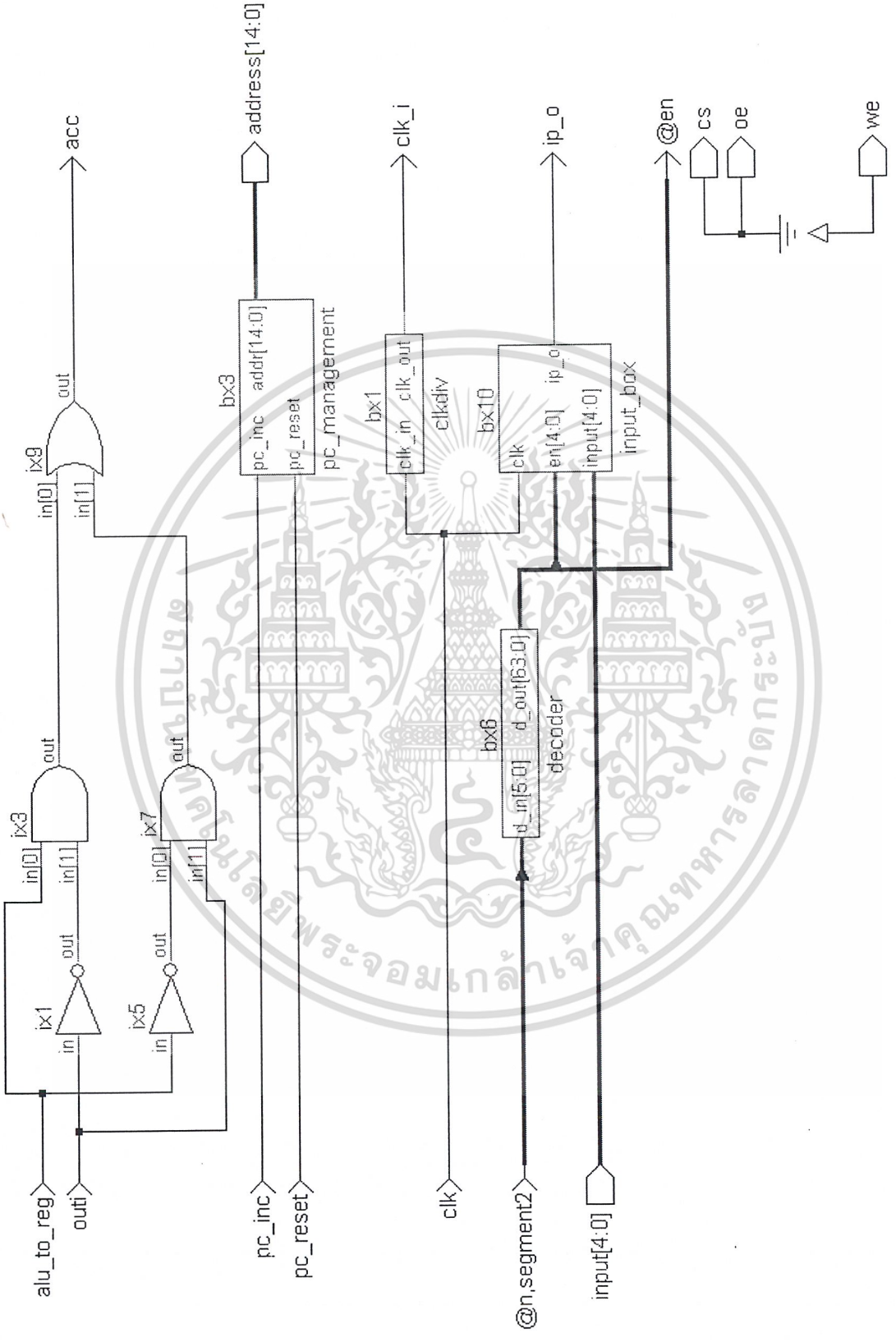
รูปที่ 6.4 แสดงขาอินพุตและเอาต์พุตของ CPU

จากนั้นทำการออกแบบ Datapath ซึ่งเป็นการแสดงถึงลักษณะของวงจรโดยรวมและเส้นทางการเชื่อมต่อของส่วนประกอบแต่ละส่วน ดังรูปที่ 6.3 การออกแบบ Datapath เป็นการออกแบบเพื่อให้เข้าใจถึงภาพรวมของวงจรก่อนตามหลักการออกแบบของ Top-Down Design ในการออกแบบยังต้องระบุถึงความสัมพันธ์ของส่วนประกอบแต่ละส่วนเพื่อให้ทราบถึงลักษณะอินพุตและเอาต์พุตของส่วนประกอบแต่ละส่วน และการใช้งานด้วย

เมื่อได้ Datapath ของวงจร CPU เป็นที่เรียบร้อยแล้วขั้นตอนต่อมาคือการออกแบบแต่ละส่วนของส่วนประกอบภายใน CPU ซึ่งการทำงานของ CPU จะประกอบด้วยส่วนประกอบต่าง ๆ ได้แก่ ส่วนจัดการหน่วยความจำ (Memory Management), รีจีสเตอร์คำสั่ง (Instruction Register), ส่วนควบคุมอินพุต, ส่วนควบคุมเอาต์พุต, ส่วนควบคุมรีเลย์ช่วย, ส่วนควบคุมไทม์เมอร์และเคาท์เตอร์, Stack, ตัวจัดการหารความถี่, ALU และ Control Unit



รูปที่ 6.5-1 แสดง Data path ของ CPU



รูปที่ 6.5-2 แสดง Data path ของ CPU

การออกแบบออฟโค้ด (Opcode) ของคำสั่งต่าง ๆ จะทำการออกแบบโดยแบ่งออกเป็น 3 กลุ่ม ตามลักษณะของแต่ละคำสั่ง โดยใช้ขนาดของคำสั่งเป็น 16 บิต หรือ 2 ไบต์ ตามตารางที่ 6.1

ตารางที่ 6.1 แสดงการออกแบบคำสั่งของ CPU

Type 1 Operation with out Data : 1 byte per instruction, start with '1'			
Mnemonic	Bit 7 – 0 (Opcode)	Instruction	
ANB	10000000	And Block (And Load)	
ORB	10000001	Or Block (Or Load)	
NOP	11111110	Not Operation.	
END	11111111	End of the program.	
Type 2 Operation with Data : 2 bytes per instruction, start with '00'			
Mnemonic	Bit 15 – 8 (Opcode)	Bit 7 – 0 (Operand)	Instruction
LD	00000000	rr nnnnnn	Load
LDI	00000001	rr nnnnnn	Load Inverse
AND	00000010	rr nnnnnn	And
ANI	00000011	rr nnnnnn	And Inverse
OR	00000100	rr nnnnnn	Or
ORI	00000101	rr nnnnnn	Or Inverse
OUT	00000110	rr nnnnnn	OUT
OUI	00000111	rr nnnnnn	OUT Inverse

rr : Type of Register
00:Input 0 to 5
01:Output 0 to 6
10:Auxilary 0 to 63
11:Tim/Cnt 0 to 3
n : Register No.

Type 3 Operation with Multiple Data : 3 bytes per instruction, start with '01'					
Mnemonic	Bit 23 – 16 (Opcode)	Bit 15 – 8 (Data1)	Bit 7 – 0 (Data2)	Instruction	
TIM	01000000	rr nnnnnn	nnnnnnnn	Timer	rr : Tim/Cnt No. 0 to 3 n : value 0 to 16383
CNT	01000001	rr nnnnnn	nnnnnnnn	Counter	

6.4 การออกแบบส่วนประกอบต่าง ๆ ของ CPU

ส่วนประกอบต่าง ๆ ของ CPU จะทำการเขียนโดยใช้ภาษา VHDL ด้วย Text Editor จากนั้นจึงทำการตรวจสอบความถูกต้องในเชิงไวยากรณ์ของการเขียนโดยใช้โปรแกรม ModelSIM ซึ่งสามารถตรวจสอบข้อผิดพลาดและแสดงผลความผิดพลาดได้ หากพบข้อผิดพลาดจะต้องทำการแก้ไขทันที

ในการตรวจสอบความถูกต้องในการทำงานของวงจรที่ทำการออกแบบ สามารถทำได้โดยดูลักษณะของสัญญาณอินพุตและเอาต์พุตของวงจร ซึ่งในการตรวจสอบในบางส่วนประกอบจะใช้วิธีการสร้างส่วนประกอบใหม่ขึ้นมาครอบส่วนประกอบเก่าที่ต้องการทดสอบ หรือเรียกว่า Test Bench โดยตัว Test Bench จะทำหน้าที่ให้กำเนิดสัญญาณอินพุตที่ใช้ในการทดสอบ จากนั้นทำการดูผลลัพธ์ที่ได้จากการทดสอบจากโปรแกรม ModelSIM อีกที

ในการตรวจสอบวงจรที่ทำการออกแบบ ว่าลักษณะของวงจรที่ออกแบบมีความถูกต้องของวงจรอย่างไรสามารถทำได้โดยใช้โปรแกรม Leonardo Spectrum ทำการสังเคราะห์วงจรแล้วดูผลการสังเคราะห์ในระดับเกต (Gate-Level)

ในการตรวจสอบขั้นตอนสุดท้ายคือการนำไปใช้งานจริง จะกระทำในตัวของ FPGA โดยใช้โปรแกรม Xilinx Design Manager ทำการสร้างข้อมูลเพื่อนำลงบอร์ดทดสอบ XS40 อีกทีหนึ่ง จากนั้นทำการทดลองป้อนค่าอินพุตและสังเกตค่าเอาต์พุตที่ได้

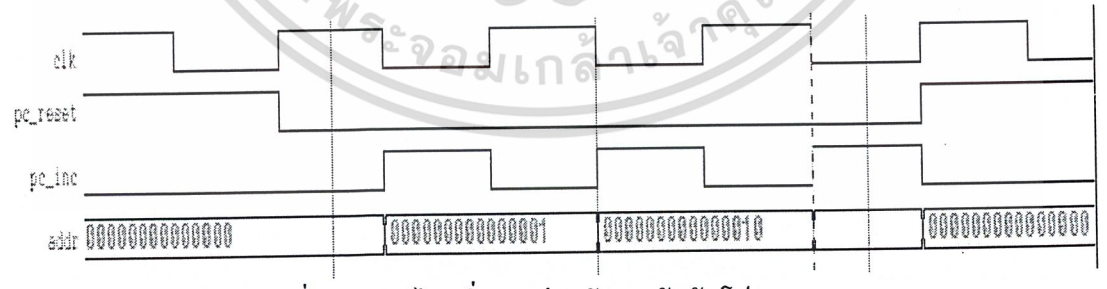
ในการตรวจสอบทุกขั้นตอนหากพบว่าเกิดปัญหาที่ขั้นตอนใด จะต้องทำการแก้ไขที่ขั้นตอนนั้นในทันที และเมื่อส่วนประกอบทุกส่วนมีความเรียบร้อยและถูกต้องแล้วจึงจะทำการรวมส่วนประกอบทุกส่วนเข้าด้วยกันเพื่อทำการทดสอบการเชื่อมต่อ และการทำงานโดยรวมอีกที

6.4.1 ส่วนจัดการตัวนับโปรแกรม (PC Management)

เป็นส่วนที่ทำหน้าที่จัดการกับการค่าตัวนับโปรแกรม (Program Counter : PC) โดยทำหน้าที่เพิ่มค่า PC รวมทั้งจัดการควบคุมให้การทำงานของ PC ให้มีการทำงานอย่างถูกต้อง ในการติดต่อกับหน่วยความจำค่าของสัญญาณ OE (Output Enable) จะมีค่าเป็น '0' เสมอ เนื่องจากในการทำงานจะต้องมีการอ่านข้อมูลจากหน่วยความจำตลอดเวลา และเมื่อได้รับสัญญาณรีเซทเข้ามาจะทำการกำหนดค่าใน PC ให้มีค่าเป็นแอดเดรสเริ่มต้น

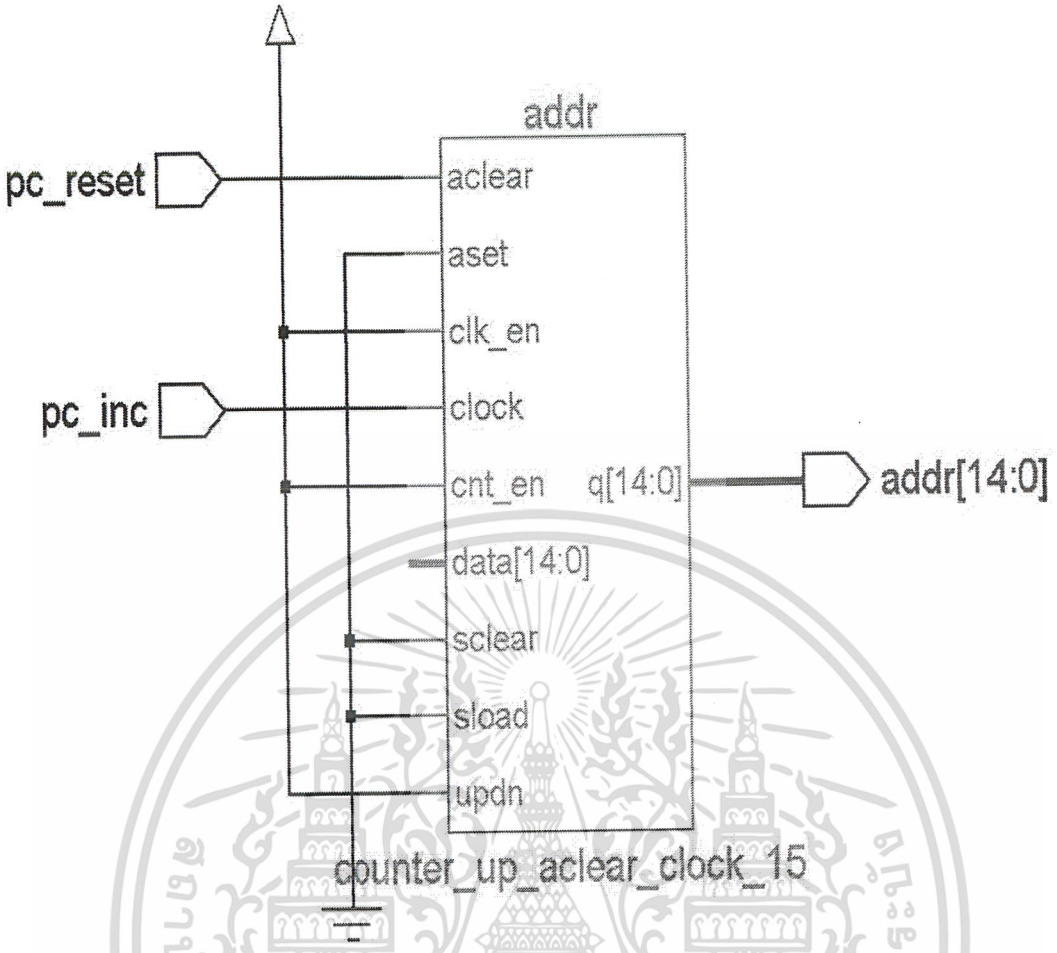
รูปที่ 6.6 แสดงขาอินพุตและเอาต์พุตของส่วนจัดการตัวนับโปรแกรม

ส่วนจัดการค่าตัวนับโปรแกรมมีการทำงานโดยขึ้นกับรับอินพุตจากสัญญาณ 2 สัญญาณ คือ PC_RESET และ สัญญาณ PC_INC โดยหาก สัญญาณ PC_RESET = '1' ค่าข้อมูลของ PC จะถูกกำหนดให้เป็นค่าเริ่มต้น และเมื่อสัญญาณ PC_INC อยู่ในสถานะขอบขาขึ้น ค่าข้อมูลของ PC จะถูกเพิ่มค่าขึ้นครั้งละ 1

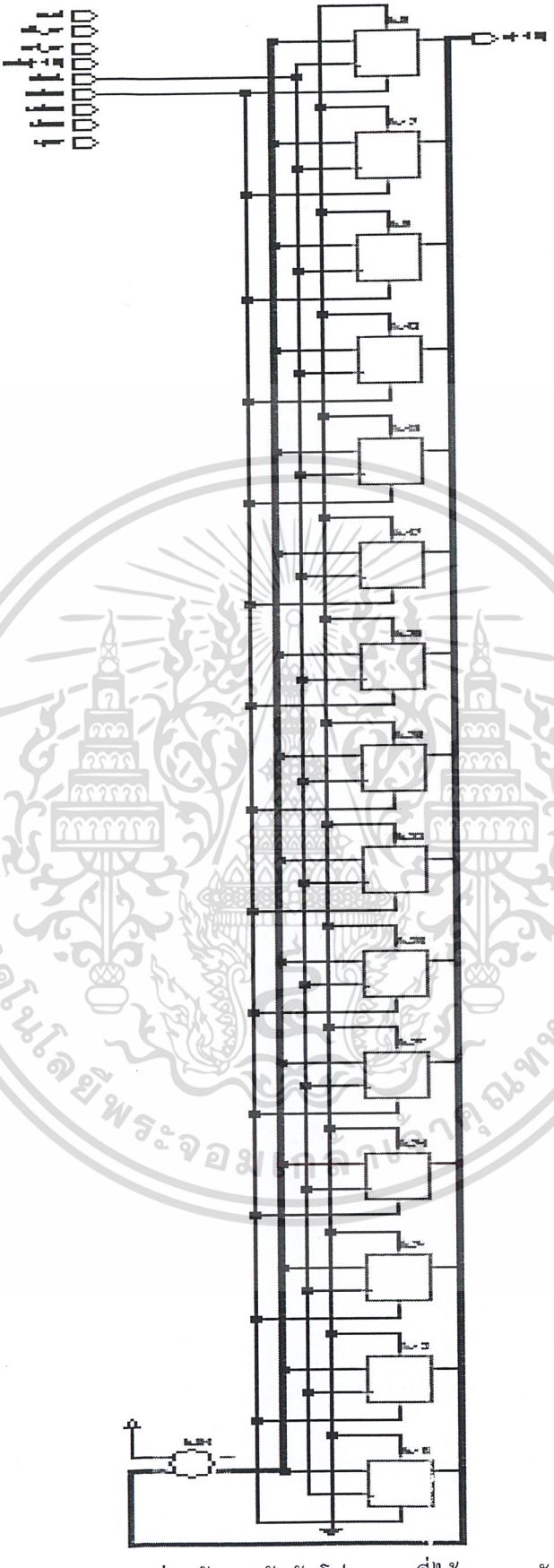


รูปที่ 6.7 แสดงไทมมิ่งของส่วนจัดการตัวนับโปรแกรม

จากรูปที่ 6.7 แสดงไทมมิ่งของส่วนจัดการตัวนับโปรแกรม (Timing Diagram) จะสังเกตได้ว่าสัญญาณ PC_INC จะอยู่ในสถานะขอบขาขึ้น เมื่อ สัญญาณ CLK อยู่ในสถานะขอบขาลง ซึ่งจะทำให้ชั่วขณะที่สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น จะทำให้อ่านได้ค่าข้อมูลจากหน่วยความจำ ที่ถูกชี้ด้วย PC ได้อย่างถูกต้องและแน่นอน



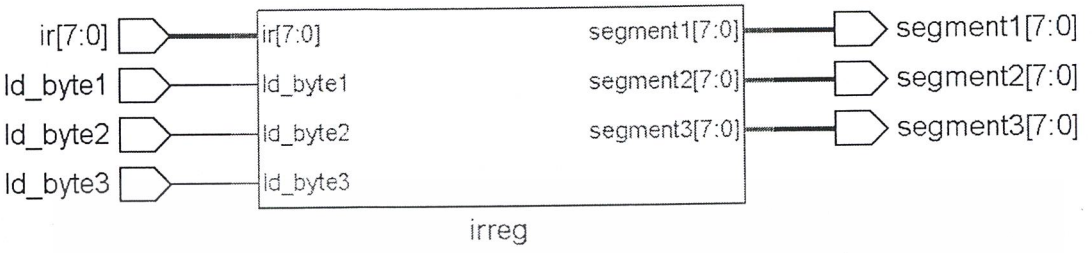
รูปที่ 6.8 แสดงผลของลายวงจรของส่วนจัดการตัวนับโปรแกรมที่ได้จากการสังเคราะห์



รูปที่ 6.9 แสดงผลของลายวงจรของส่วนจัดการตัวนับโปรแกรมที่ได้จากการสังเคราะห์อีกระดับหนึ่ง

6.4.2 รีจีสเตอร์คำสั่ง (Instruction Register)

เป็นส่วนที่มีหน้าที่รับผิดชอบเกี่ยวกับ การควบคุมการทำงานของ รีจีสเตอร์คำสั่ง (Instruction Register) ซึ่งเป็นข้อมูลที่ได้มาจากหน่วยความจำภายนอก โดยมีขนาดและโครงสร้างของแต่ละชนิดคำสั่ง ดังนี้

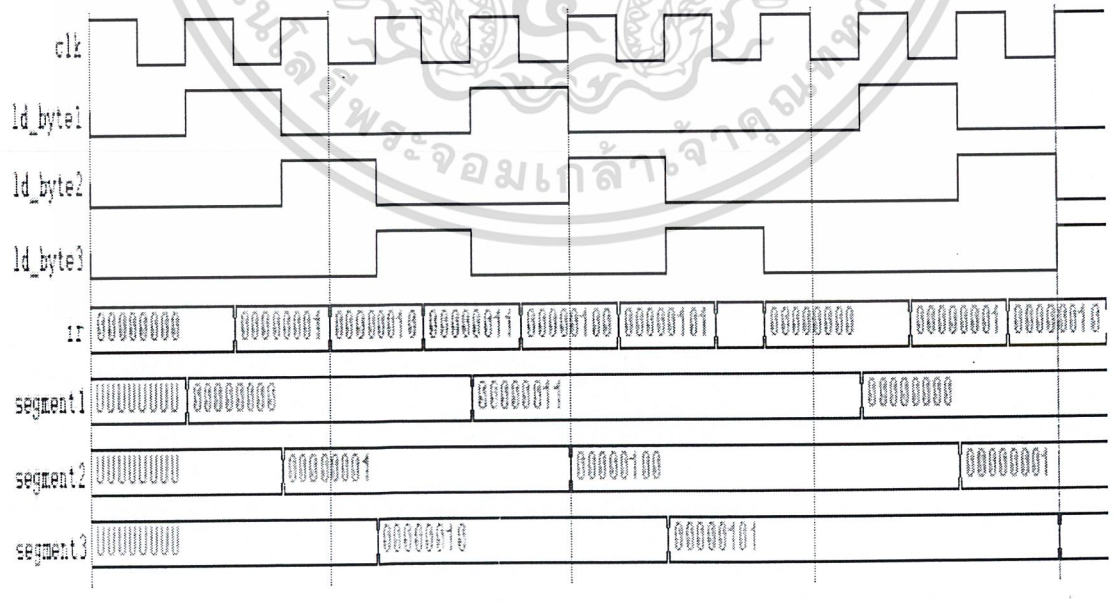


รูปที่ 6.10 แสดงขาอินพุตและเอาต์พุตของส่วนรีจีสเตอร์คำสั่ง

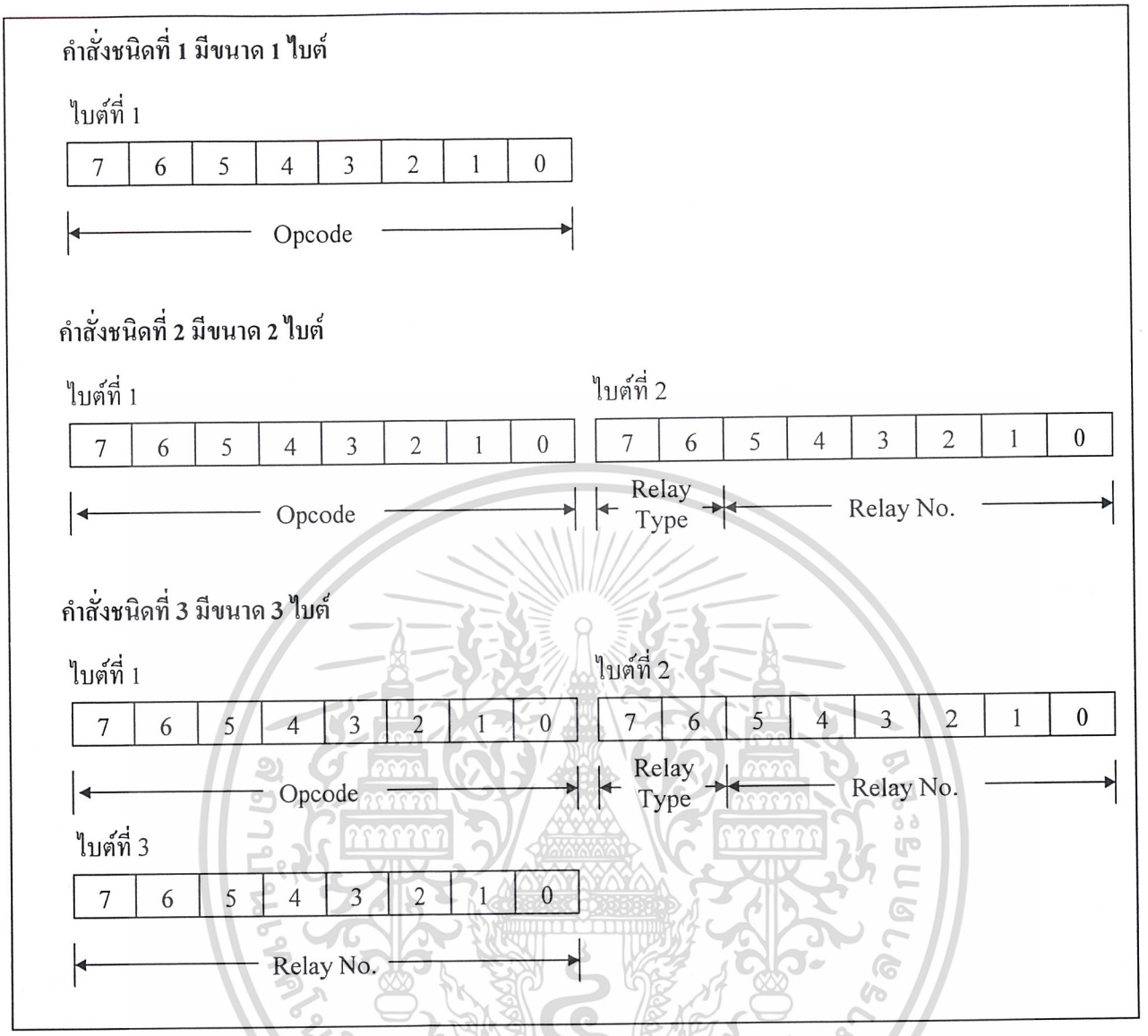
คำสั่งที่เป็นชนิดที่ 1 มีขนาด 8 บิต (Bits) หรือ 1 ไบต์ (Byte) โดยทั้ง 8 บิต เป็นตัวระบุชนิดของคำสั่ง (Opcode)

คำสั่งที่เป็นชนิดที่ 2 มีขนาด 16 บิตหรือ 2 ไบต์สามารถแบ่งข้อมูลในคำสั่งออกเป็นไบต์ที่หนึ่ง 8 บิตแรกเป็นชนิดของคำสั่ง ไบต์ที่สอง 2 บิตแรกเป็นชนิดของรีเลย์ และ 6 บิตถัดมาจะเป็นค่าที่ระบุหมายเลขตำแหน่งที่ใช้งานของรีเลย์

คำสั่งที่เป็นชนิดที่ 3 มีขนาด 24 บิตหรือ 3 ไบต์ สามารถแบ่งข้อมูลในคำสั่งออกเป็นไบต์ที่หนึ่ง 8 บิตแรกเป็นชนิดของคำสั่ง และไบต์ที่สอง 2 บิตแรกจะเป็นตัวระบุตำแหน่งที่ใช้งาน ของรีเลย์ประเภทไทม์เมอร์และเคาท์เตอร์ และ 6 บิตสุดท้ายของไบต์ที่ 2 จะใช้งานร่วมกับ 8 บิตของไบต์ที่สามรวมเป็น 14 บิต จะเป็นค่าที่ระบุจำนวนที่ต้องการให้ รีเลย์ประเภทไทม์เมอร์และเคาท์เตอร์ นั้น ๆ ทำการนับหรือ จับเวลา



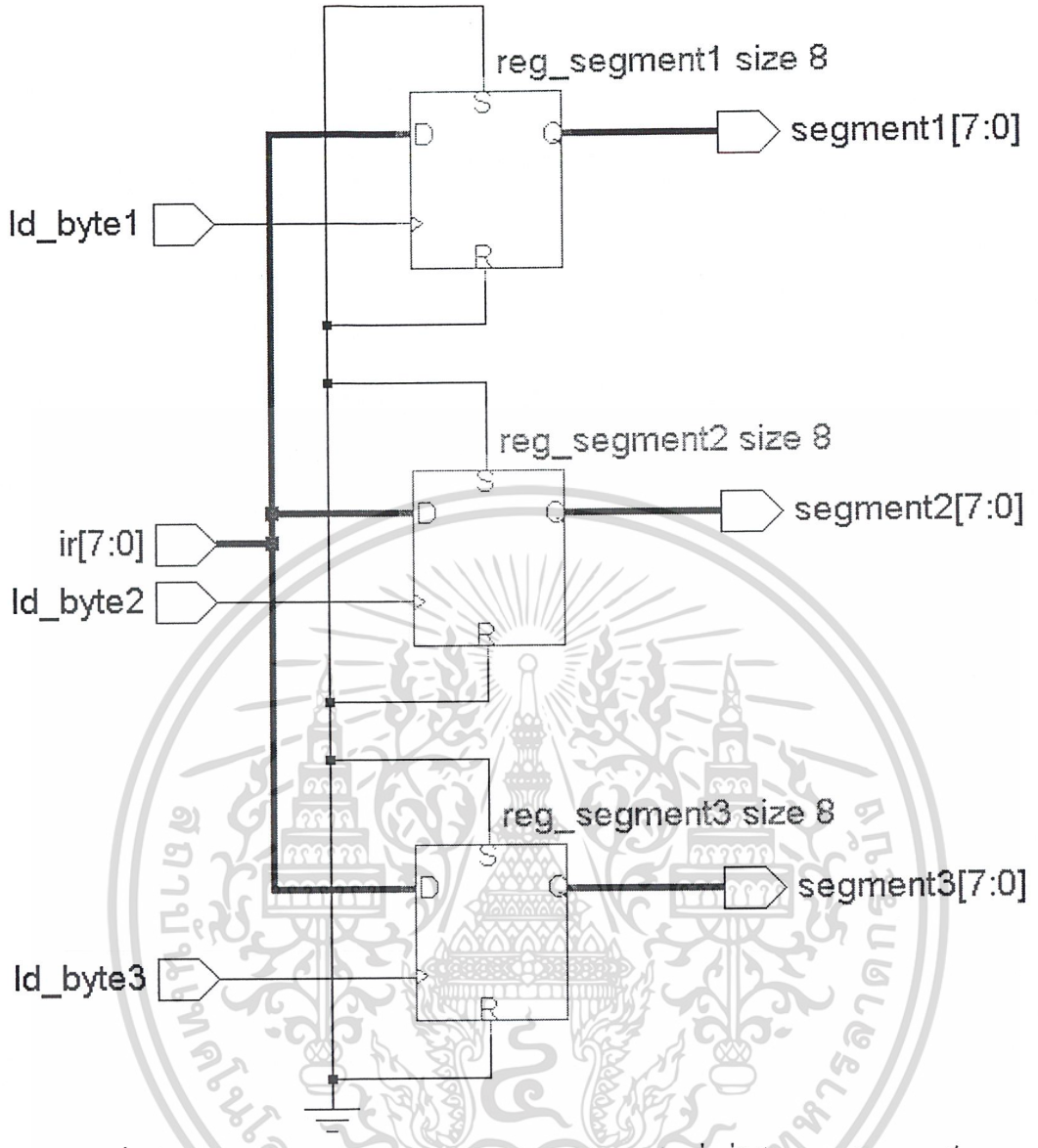
รูปที่ 6.11 แสดงไทมมิ่งของส่วนจัดการรีจีสเตอร์คำสั่ง



รูปที่ 6.12 แสดงภาพของข้อมูลในแต่ละไบต์คำสั่ง

ส่วนจัดการรีจิสเตอร์คำสั่ง มีการทำงานโดยขึ้นกับสัญญาณ 3 สัญญาณคือ สัญญาณ LD_BYTE1 สัญญาณ LD_BYTE2 และ สัญญาณ LD_BYTE3 โดยหากสัญญาณ LD_BYTE ใด อยู่ในสถานะขอบขาขึ้น ค่าข้อมูลจากสัญญาณ IR ทั้ง 8 bits จะถูกเก็บที่เซกเมนท์ (Segment) นั้น โดยใช้ฟลิปฟลอป (Flip Flop)

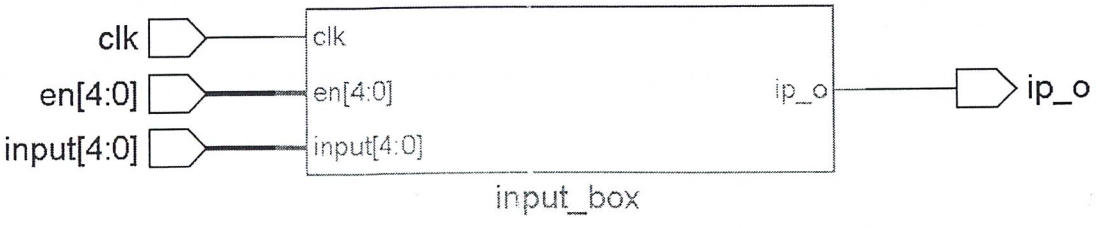
จากรูปที่ 6.11 จะสังเกตได้ว่า สัญญาณ IR ซึ่งเป็นสัญญาณข้อมูลที่อ่านได้จากหน่วยความจำ จะมีการเปลี่ยนแปลงขณะ สัญญาณ CLK อยู่ในสถานะขอบขาลง เนื่องจากค่า PC จะถูกเปลี่ยนเมื่อสัญญาณ CLK อยู่ในสถานะขอบขาลง สัญญาณ LD_BYTE1 สัญญาณ LD_BYTE2 และ สัญญาณ LD_BYTE3 จะอยู่ในสถานะขอบขาขึ้น ขณะที่สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น ซึ่งเป็นช่วงเวลาที่ สัญญาณข้อมูล IR มีความเสถียร



รูปที่ 6.13 แสดงผลของลายวงจรของส่วนจัดการรีจีสเตอร์คำสั่งที่ได้จากการสังเคราะห์

6.4.3 ส่วนควบคุมอินพุต

เป็นส่วนที่มีหน้าที่รับผิดชอบเกี่ยวกับการจัดการกับสัญญาณอินพุตจากภายนอก การรับค่าสัญญาณที่เข้ามาภายในวงจรจะใช้ฟลิปฟลอป (Flip Flop) เป็นตัวเก็บค่า และภายในอุปกรณ์จะใช้สัญญาณนาฬิกาที่มีความถี่ 5 MHz ในการควบคุมอุปกรณ์

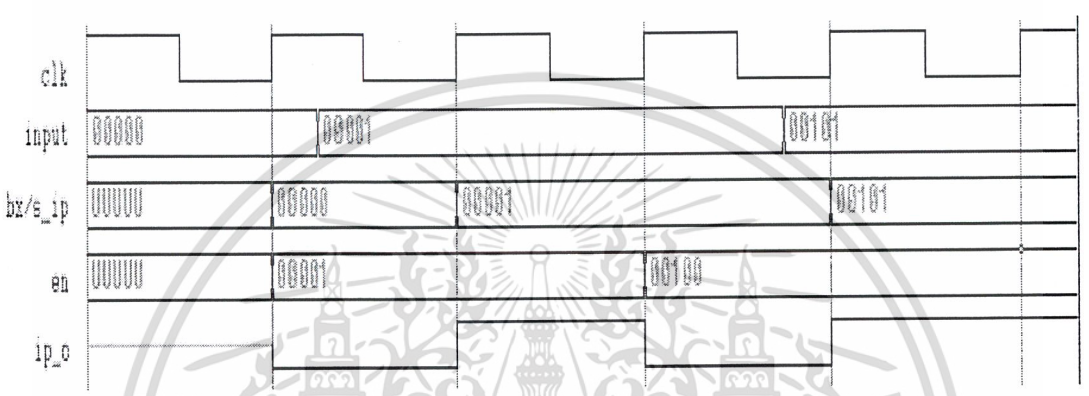


รูปที่ 6.14 แสดงขาอินพุตและเอาต์พุตของส่วนควบคุมอินพุต

ส่วนควบคุมอินพุตจะแบ่งการทำงานออกได้เป็น 2 ส่วนคือ ส่วนเก็บค่าข้อมูล และส่วนนำค่าข้อมูลออกไปใช้ โดยทั้ง 2 ส่วนมีการทำงานดังนี้

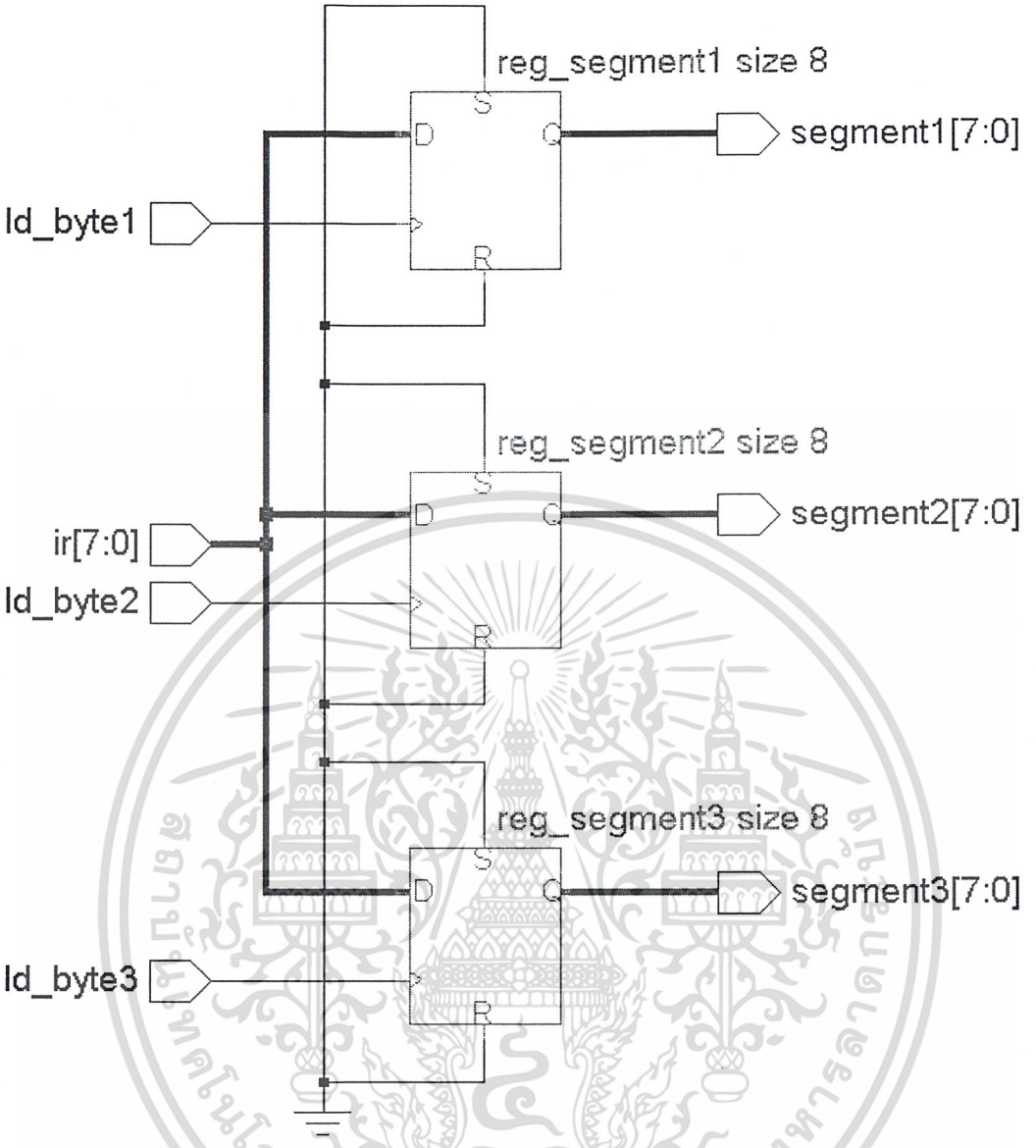
ส่วนเก็บค่าข้อมูลจะใช้สัญญาณนาฬิกาในการควบคุมฟลิปฟลอปเพื่อใช้ในการเก็บค่าอินพุต มีการทำงานดังนี้คือ ทุก ๆ 200 ns (Clock = 5 Mhz) จะทำการรับสัญญาณอินพุตจากภายนอก เพื่อจัดเก็บลงฟลิปฟลอป (Flip Flop)

ส่วนนำค่าข้อมูลออกไปใช้ มีการทำงานโดยขึ้นกับสัญญาณเลือก (EN) จำนวน 5 bits โดยในแต่ละบิต จะเป็นตัวบ่งชี้ว่า จะให้ ข้อมูลที่ถูกเก็บไว้ตัวไหนถูกนำออกไปใช้



รูปที่ 6.15 แสดงไทมมิ่งของส่วนควบคุมอินพุต

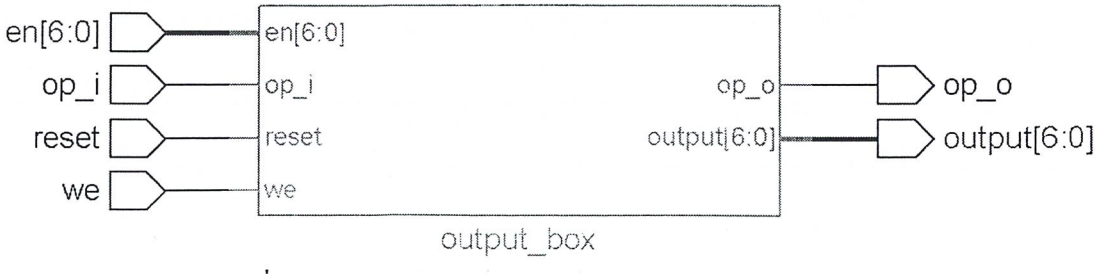
จากรูปที่ 6.15 แสดงไทมมิ่งของส่วนควบคุมอินพุต จะสังเกตได้ว่า สัญญาณ CLK มีคาบเป็น 200 ns โดยหากสัญญาณอินพุต มีการเปลี่ยนแปลงหลังจาก สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น ข้อมูลอินพุตจะถูกรอและเปลี่ยนแปลงเมื่อ สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น ครั้งถัดไป ทำให้เกิดค่าดีเลย์ (Delay time) ซึ่งค่าดีเลย์จะมีค่าไม่เกิน 200 ns ซึ่งเป็นค่าที่ส่งผลน้อยมาก หรือไม่ส่งผลเลยสำหรับการประมวลผลของ CPU



รูปที่ 6.16 แสดงผลของลายวงจรของส่วนควบคุมอินพุตที่ได้จากการสังเคราะห์

6.4.4 ส่วนควบคุมเอาต์พุต

เป็นส่วนที่มีหน้าที่รับผิดชอบเกี่ยวกับการควบคุมสัญญาณเอาต์พุต ซึ่งเป็นสัญญาณที่ส่งออกสู่ภายนอกของ PLC ในการส่งค่าออกจะมีการเก็บไว้ในฟลิปฟลอปก่อนแล้วค่อยทำการส่งออกอีกที เนื่องจากมีการเก็บค่าไว้ในฟลิปฟลอปทำให้สามารถ ทำการเรียกค่าของเอาต์พุตรีจิสเตอร์ (Ouput Register) ตัวนั้น ๆ เพื่อไปทำการประมวลผลต่าง ๆ ได้

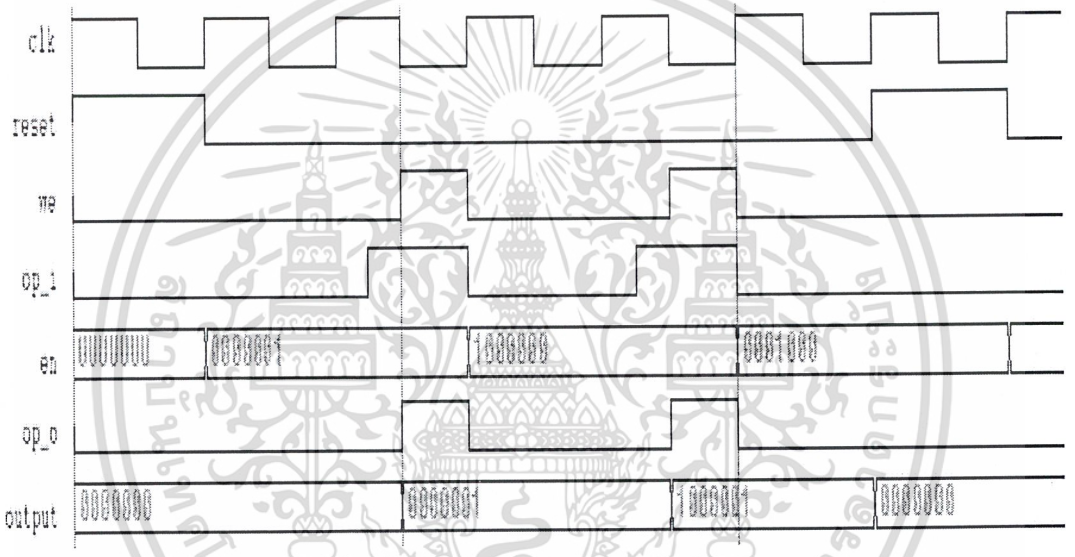


รูปที่ 6.17 แสดงขาอินพุตและเอาต์พุตของส่วนควบคุมเอาต์พุต

ส่วนควบคุมเอาต์พุต มีการทำงาน แบ่งได้เป็น 3 ส่วน คือ ส่วนเก็บค่าข้อมูล ส่วนนำค่าข้อมูลออกไปใช้ และส่วนในการนำค่าข้อมูลที่เก็บไว้ทั้งหมดไปเป็นสัญญาณภายนอก โดยทั้ง 3 ส่วนมีการทำงานดังต่อไปนี้

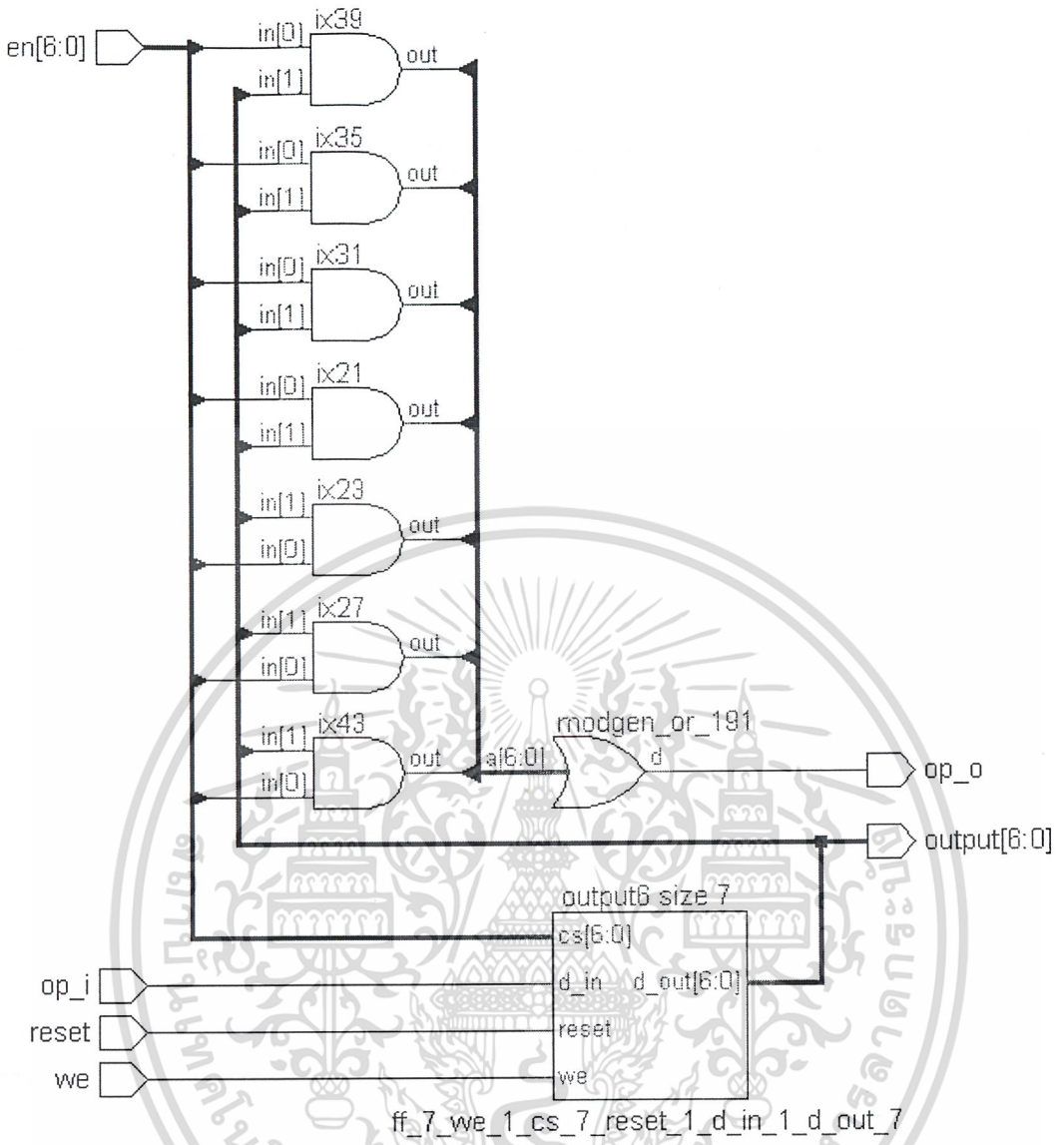
ส่วนเก็บค่าข้อมูล มีการทำงานโดยขึ้นกับสัญญาณ 4 สัญญาณคือ สัญญาณรีเซท (reset) สัญญาณเลือก (en) สัญญาณเก็บข้อมูล (we) และ สัญญาณข้อมูล (OP_i) โดยหาก สัญญาณรีเซทมีค่าเท่ากับ '1' ค่าของข้อมูลทุกค่าจะถูกกำหนดเป็นค่าเริ่มต้นหรือ '0' ในการจะนำค่าเก็บลงฟลิปฟลอปจะต้องมีสัญญาณเลือก และสัญญาณข้อมูล ที่ถูกต้องแน่นอน ในขณะที่ สัญญาณเก็บข้อมูลอยู่ในสถานะขอบขาขึ้น

ส่วนนำค่าข้อมูลออกไปใช้ มีการทำงานโดยขึ้นกับสัญญาณเลือกจำนวน 7 บิตโดยในแต่ละบิตจะเป็นตัวบ่งชี้ว่า จะให้ข้อมูลที่ถูกเก็บไว้ตัวไหนถูกนำออกไปใช้



รูปที่ 6.18 แสดงไทมมิ่งของส่วนควบคุมเอาต์พุต

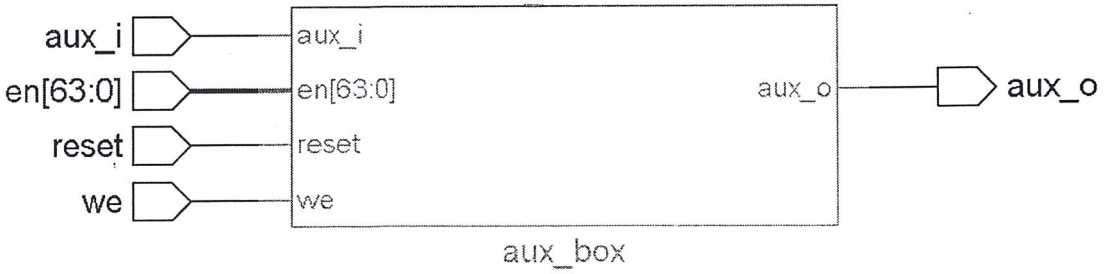
จากรูปที่ 6.18 แสดงไทมมิ่งของส่วนควบคุมเอาต์พุต จะสังเกตได้ว่า สัญญาณเลือก (en) มีการเปลี่ยนแปลงขณะที่ สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น เนื่องจากสัญญาณดังกล่าว เป็นข้อมูลที่มีการเปลี่ยนแปลงตามสัญญาณ LD_BYTE2 ของส่วนจัดการรีจีสเตอร์คำสั่ง ซึ่งจะทำงานขณะที่ สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น สัญญาณข้อมูล (OP_i) จะมีการเปลี่ยนแปลงหลังจากสัญญาณ CLK อยู่ในสถานะขอบขาขึ้น เนื่องจากสัญญาณดังกล่าว จะต้องผ่านการประมวลผลให้เป็นสัญญาณที่ถูกต้อง และจะต้องเป็นสัญญาณที่เสถียรก่อนที่สัญญาณ CLK จะอยู่ในสถานะขอบขาลง สัญญาณเก็บข้อมูล (we) จะมีการเปลี่ยนแปลงในขณะที่สัญญาณ CLK อยู่ในสถานะขอบขาลง ซึ่งสัญญาณเลือก (en) และสัญญาณข้อมูล (OP_i) มีความเสถียรสำหรับนำไปใช้เป็นสัญญาณสัญญาณภายนอก ส่วนสัญญาณที่จะนำไปใช้สำหรับส่วนอื่น จะมีการเปลี่ยนแปลงทันทีข้อมูลที่ถูกเก็บไว้เปลี่ยนแปลง



รูปที่ 6.19 แสดงผลของลายวงจรของส่วนควบคุมเอาต์พุตที่ได้จากการสังเคราะห์

6.4.5 ส่วนควบคุมรีเลย์ช่วย

เป็นส่วนที่มีหน้าที่รับผิดชอบเกี่ยวกับ การควบคุมการทำงานของรีเลย์ช่วย ซึ่งรีเลย์ช่วยจะมีอยู่ทั้งหมด 64 ตัวโดยในการเก็บค่าของข้อมูลจะใช้ฟลิปฟลอป

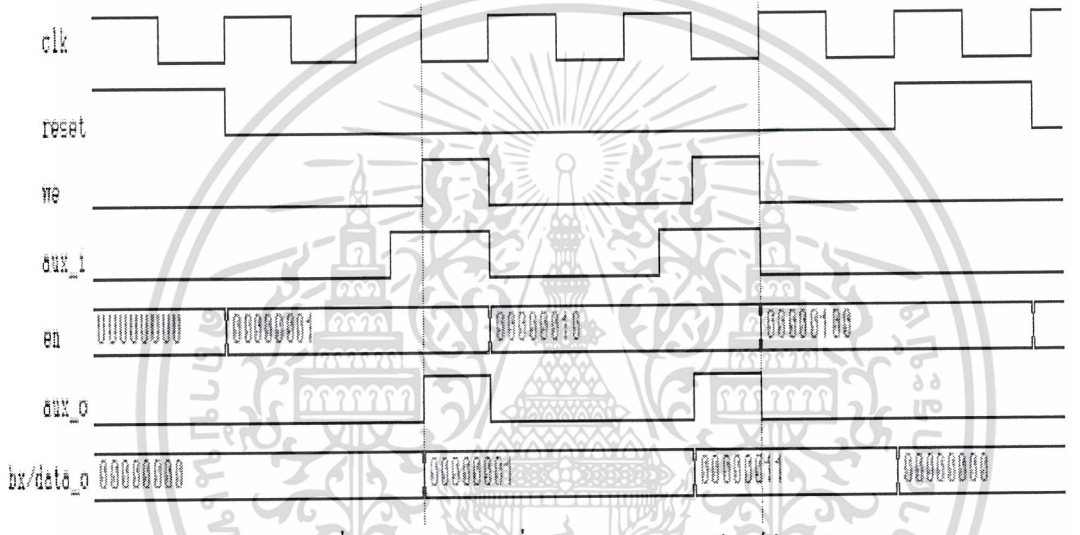


รูปที่ 6.20 แสดงขาอินพุตและเอาต์พุตของส่วนควบคุมรีเลย์ช่วย

ส่วนควบคุมรีเลย์ช่วย มีการทำงาน แบ่งได้เป็น 2 ส่วน คือ ส่วนเก็บค่าข้อมูล และส่วนนำค่าของข้อมูลออกไปใช้ โดยทั้ง 2 ส่วนมีการทำงานดังต่อนี้

ส่วนที่หนึ่ง ส่วนเก็บค่าข้อมูล มีการทำงานโดยขึ้นกับสัญญาณ 4 สัญญาณคือ สัญญาณรีเซท (reset) สัญญาณเลือก (en) สัญญาณเก็บข้อมูล (we) และ สัญญาณข้อมูล (AUX_i) โดยหาก สัญญาณรีเซทมีค่าเท่ากับ '1' ค่าของข้อมูลทุกค่าจะถูกกำหนดเป็นค่าเริ่มต้นหรือ '0' ในการจะนำค่าเก็บลงฟลิปฟลอปจะต้องมีสัญญาณเลือก และสัญญาณข้อมูล ที่ถูกต้องแน่นอน ในขณะที่ สัญญาณเก็บข้อมูลอยู่ในสถานะขอบขาขึ้น

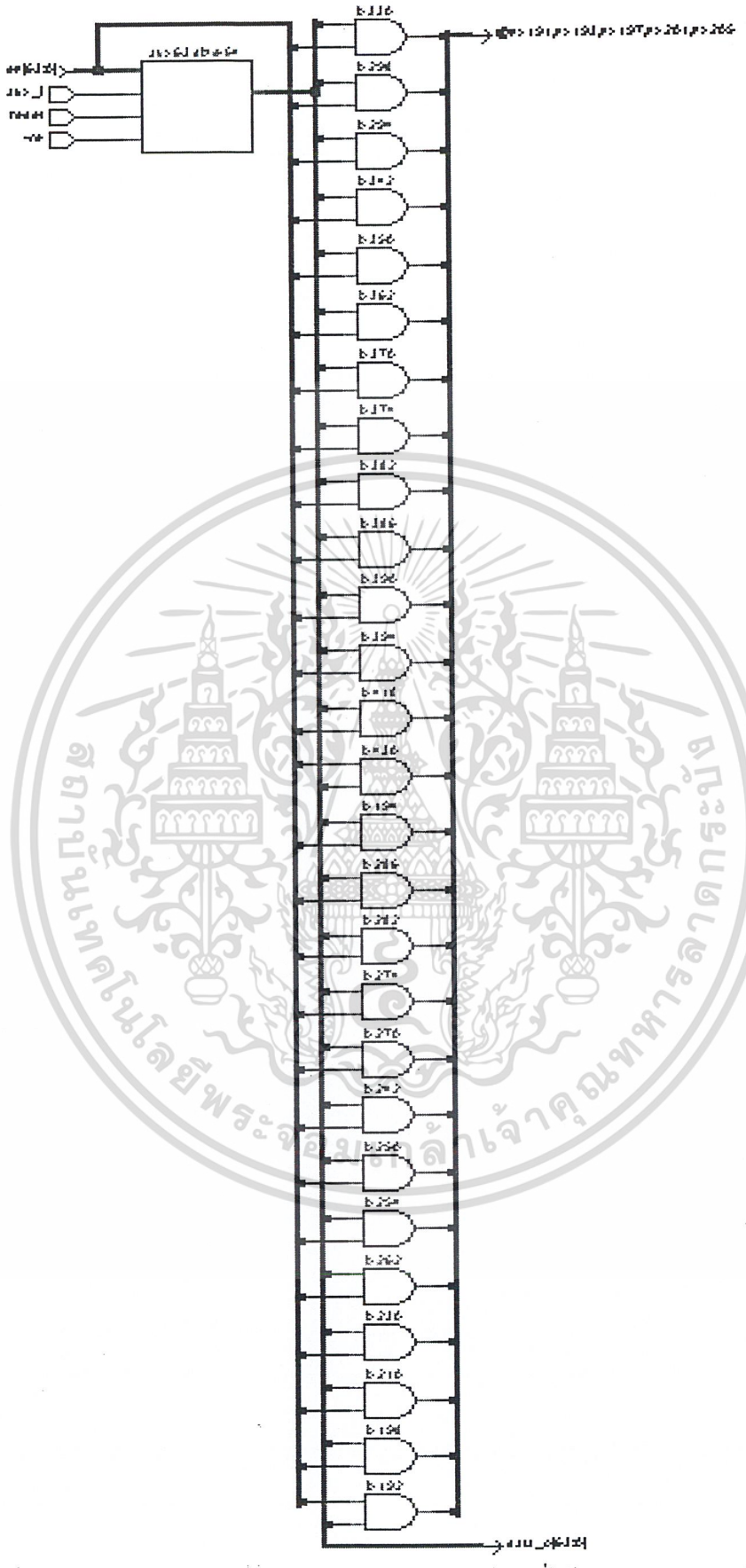
ส่วนที่สอง ส่วนนำค่าข้อมูลออกไปใช้ มีการทำงานโดยขึ้นกับสัญญาณเลือกจำนวน 7 บิตโดยในแต่ละบิต จะเป็นตัวบ่งชี้ว่า จะให้ข้อมูลที่ถูกเก็บไว้ตัวไหนถูกนำออกไปใช้



รูปที่ 6.21 แสดงไทมมิ่งของส่วนควบคุมรีเลย์ช่วย

จากรูปที่ 6.21 แสดงไทมมิ่งของส่วนควบคุมเอาต์พุต จะสังเกตได้ว่า สัญญาณเลือก (en) มีการเปลี่ยนแปลงขณะที่ สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น เนื่องจากสัญญาณดังกล่าว เป็นข้อมูลที่มีการเปลี่ยนแปลงตามสัญญาณ LD_BYTE2 ของส่วนจัดการรีจีสเตอร์คำสั่ง ซึ่งจะทำงานขณะที่ สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น สัญญาณข้อมูล (AUX_i) จะมีการเปลี่ยนแปลงหลังจากสัญญาณ CLK อยู่ในสถานะขอบขาขึ้น เนื่องจากสัญญาณดังกล่าว จะต้องผ่านการประมวลผลให้เป็นสัญญาณที่ถูกต้อง และจะต้องเป็นสัญญาณที่เสถียรก่อนที่สัญญาณ CLK จะอยู่ในสถานะขอบขาลง สัญญาณเก็บข้อมูล (we) จะมีการเปลี่ยนแปลงในขณะที่สัญญาณ CLK อยู่ในสถานะขอบขาลง ซึ่งสัญญาณเลือก (en) และสัญญาณข้อมูล (AUX_i) มีความเสถียรสำหรับนำไปใช้เป็นสัญญาณสัญญาณภายนอก ส่วนสัญญาณที่จะนำไปใช้สำหรับส่วนอื่น จะมีการเปลี่ยนแปลงทันที่ข้อมูลที่ถูกเก็บไว้เปลี่ยนแปลง

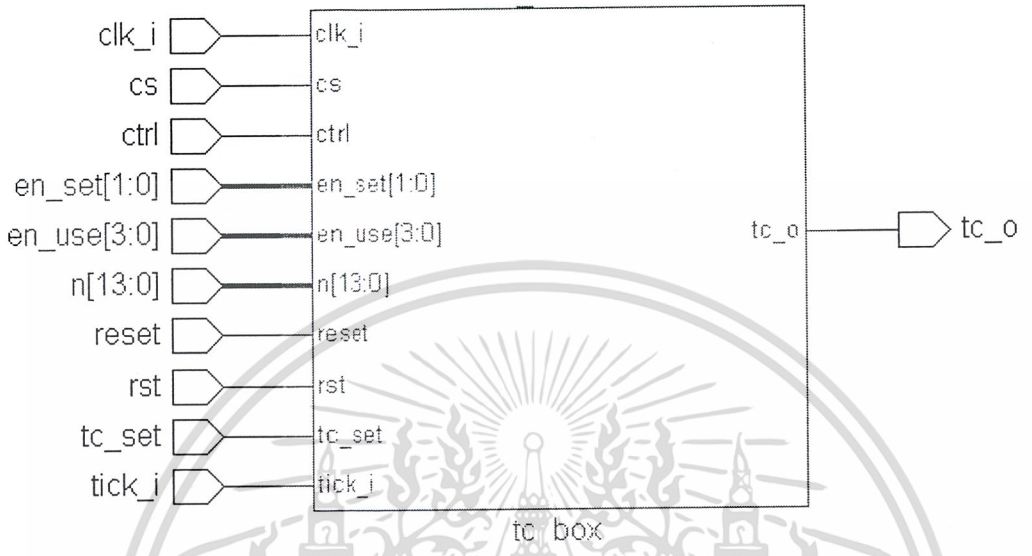
รูปที่ 6.22-1 แสดงผลของลายวงจรของส่วนควบคุมรีเลย์ช่วยที่ได้จากการสังเคราะห์



รูปที่ 6.22-2 แสดงผลของลายวงจรของส่วนควบคุมรีเลย์ช่วยที่ได้จากการสังเคราะห์ (ต่อ)

6.4.6 ส่วนควบคุมไทม์เมอร์และเคาท์เตอร์

เป็นส่วนที่มีหน้าที่รับผิดชอบเกี่ยวกับการควบคุมการทำงานของอุปกรณ์ไทม์เมอร์และเคาท์เตอร์ โดยจะมีจำนวนอุปกรณ์ไทม์เมอร์และเคาท์เตอร์รวมกันอยู่จำนวน 4 ชุดการทำงาน โดยในแต่ละชุดการทำงานจะมีการทำงานร่วมกันได้

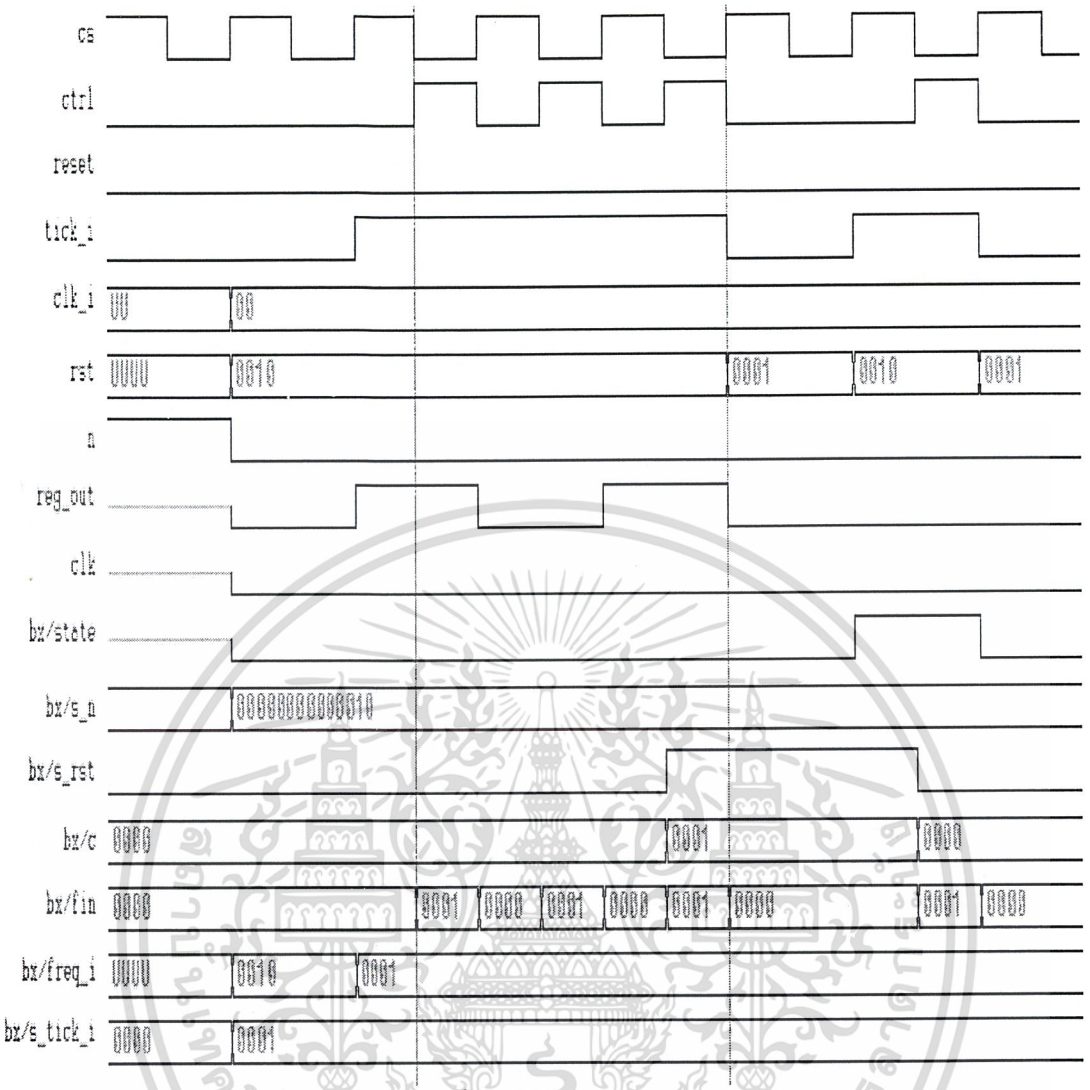


รูปที่ 6.23 แสดงขาอินพุตและเอาต์พุตของส่วนควบคุมไทม์เมอร์และเคาท์เตอร์

ส่วนควบคุมไทม์เมอร์และเคาท์เตอร์ มีการทำงานแบ่งได้เป็น 2 ส่วน คือ ส่วนพิจารณาสัญญาณเลือก และส่วนพิจารณาข้อมูลออก

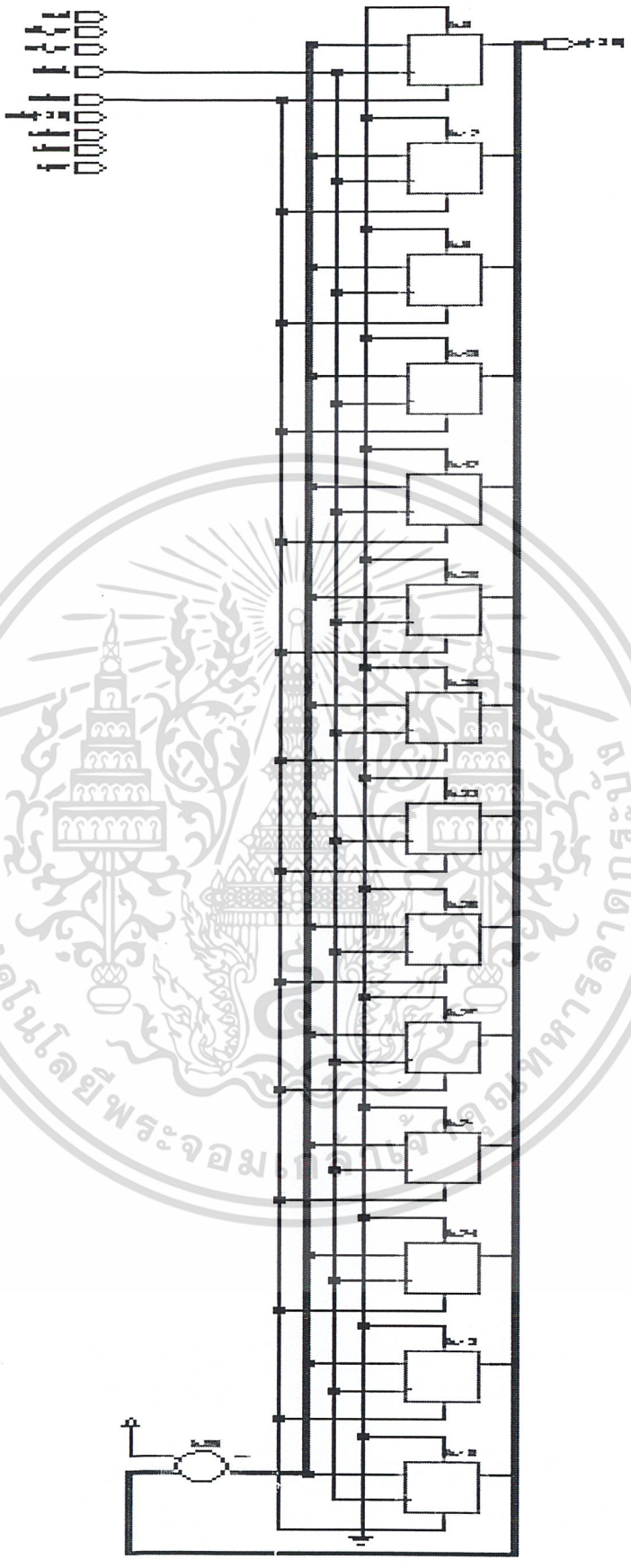
ส่วนพิจารณาสัญญาณเลือก มีการทำงานโดยพิจารณาว่า สัญญาณเลือกที่ใช้ภายในจะเป็นสัญญาณจาก สัญญาณเลือกขณะกำหนดค่า หรือสัญญาณเลือกขณะใช้ค่า โดยไม่ว่าจะเลือกสัญญาณใด ยังจะต้องพิจารณาร่วมกับสัญญาณเก็บข้อมูลด้วยเสมอ

ส่วนการพิจารณาข้อมูลออก หลังจากได้สัญญาณเลือกข้างต้น จำนวน 4 บิต ในแต่ละบิต จะเป็นตัวบ่งชี้ว่า จะให้ ข้อมูลที่ถูกเก็บไว้จากอุปกรณ์ตัวไหนถูกนำออกไปใช้



รูปที่ 6.24 แสดงไทมมิ่งของส่วนไทมเมอร์และเคาท์เตอร์

จากรูปที่ 6.24 แสดงไทมมิ่งของส่วนไทมเมอร์และเคาท์เตอร์ จะสังเกตได้ว่า สัญญาณเลือกจากภายนอก (sc) จะมีการเปลี่ยนแปลงขณะที่สัญญาณ CLK อยู่ในสถานะขอบขาลง เนื่องจากเป็นช่วงเวลาที่สัญญาณข้อมูลที่ต้องทำงานร่วมด้วยมีความเสถียร เพราะสัญญาณดังกล่าวมีการเปลี่ยนเฉพาะเวลาที่สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น



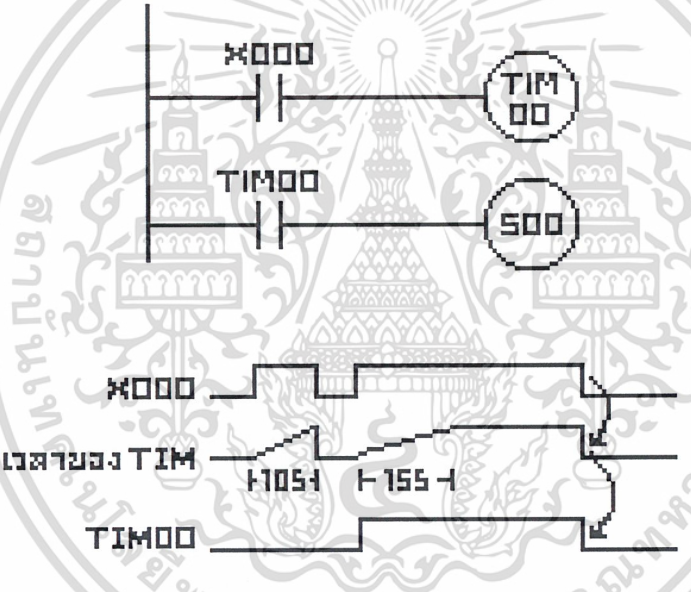
รูปที่ 6.25 แสดงผลของลายวงจรของส่วนไทม์เมอร์และเคาท์เตอร์ที่ได้จากการสังเคราะห์

ในแต่ละอุปกรณ์ จะมีการทำงานดังนี้

เมื่อมีสัญญาณกำหนดค่าเริ่มต้น (Reset) อุปกรณ์จะมีการทำงานในสถานะเริ่มต้น คืออุปกรณ์จะถูกตั้งให้เป็นเคาท์เตอร์ โดยการกำหนดค่าข้อมูลตัวนับภายในเป็นค่าเริ่มต้นเสมอ และกำหนดค่าจำนวนที่ต้องนับถึงเป็น 16383

เมื่ออยู่ในสถานะการทำงานแบบไทม์เมอร์ จะทำการนับค่าเวลาโดยมีหน่วยเป็น 1/10 วินาทีหรือ 1 เดซิวินาที (0.1s) หากทำการนับเสร็จสิ้นจะให้ค่าสัญญาณเอาต์พุตออกเป็น '1' และหากสัญญาณหยุดนับมีค่าเป็น '1' จะทำการหยุดนับและกำหนดค่าข้อมูลตัวนับภายในเป็นค่าเริ่มต้น

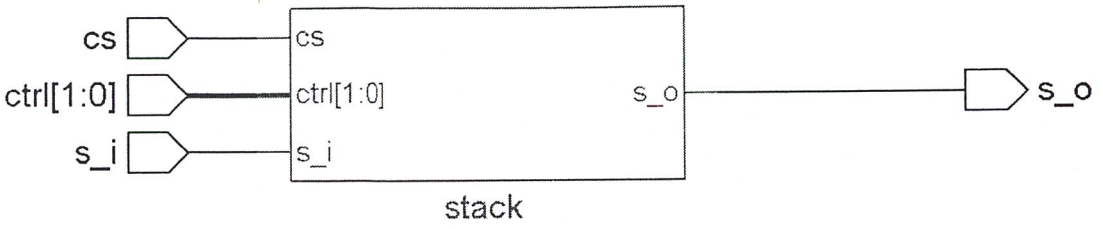
เมื่ออยู่ในสถานะการทำงานแบบเคาท์เตอร์ จะทำหน้าที่เป็นตัวนับสัญญาณ โดยจะนับสัญญาณตามจำนวนครั้งที่มีสัญญาณเข้ามา หากสัญญาณมีจำนวนครบตามที่ตั้งค่าไว้จะทำการส่งสัญญาณเอาต์พุตออกเป็น '1' และหากสัญญาณหยุดนับมีค่าเป็น '1' จะทำการหยุดนับและกำหนดค่าข้อมูลตัวนับภายในเป็นค่าเริ่มต้น



รูปที่ 6.26 แสดงตัวอย่างการทำงานของไทม์เมอร์และเคาท์เตอร์

6.4.7 Stack

เป็นส่วนที่รับผิดชอบหน้าที่ในการจัดการกับ Stack ในการทำงานของ CPU ค่าที่ได้จากการประมวลผลจะถูกเก็บไว้ใน Stack เพื่อนำมาประมวลผลกับคำสั่งต่อไป โดยความลึกของ Stack จะมีขนาด 32 ชั้น



รูปที่ 6.27 แสดงขาอินพุตและเอาต์พุตของ Stack

การทำงานของ Stack จะถูกควบคุมด้วยสัญญาณควบคุม (STACK_CTRL) ซึ่งมีคำสั่งดังต่อไปนี้

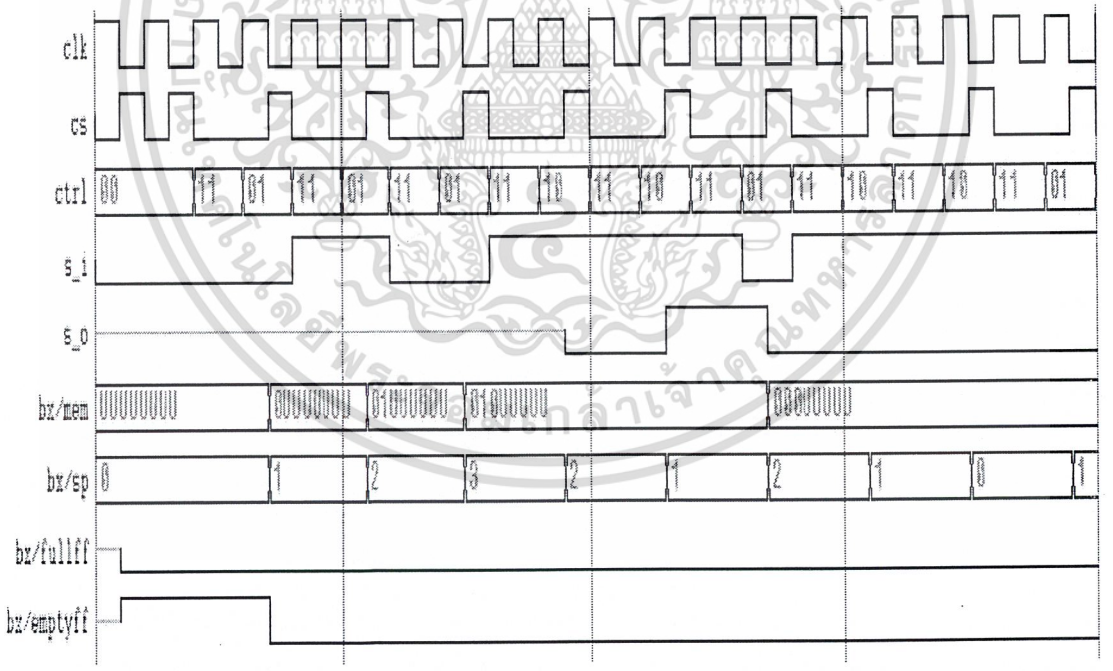
sCLR : จะทำการ ตั้งค่าตัวชี้ตำแหน่งบนสุดของ Stack ให้ชี้ไปที่ตำแหน่งล่างสุดของ Stack

sPUSH : จะทำการ เลื่อนตัวชี้ตำแหน่งบนสุดของ Stack ขึ้นไป 1 ตำแหน่ง แล้วทำการเก็บค่าของข้อมูลลงยังตำแหน่ง ที่ถูกชี้ด้วยตัวชี้ตำแหน่งบนสุดของ Stack

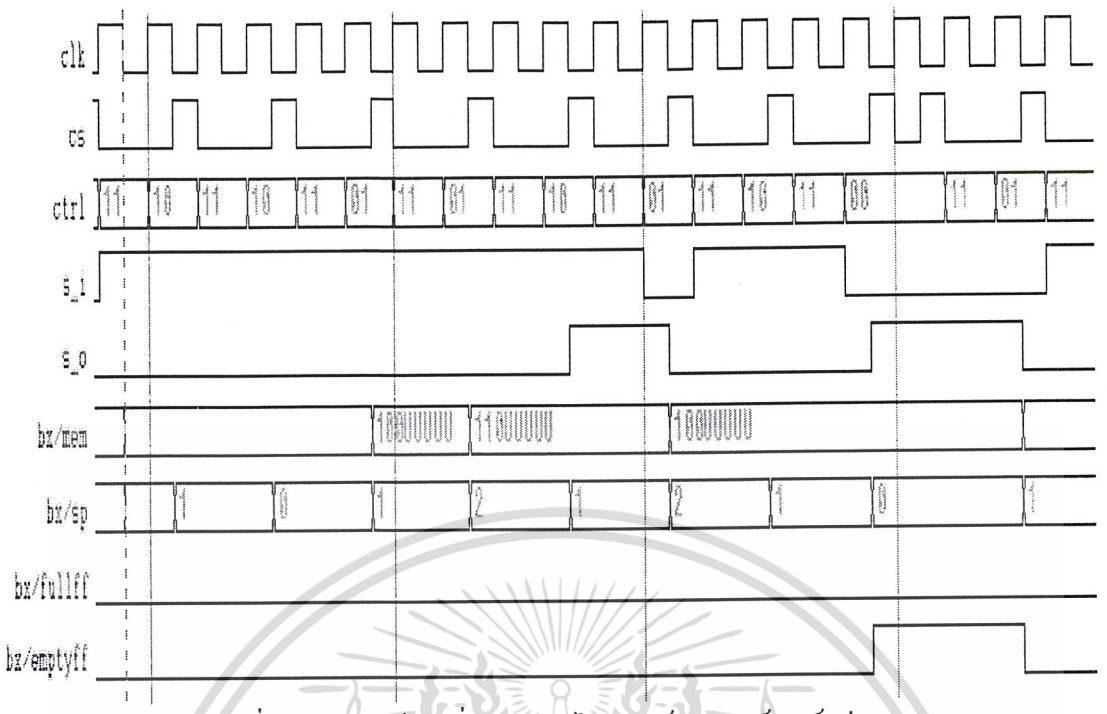
sPOP : จะทำการนำค่าของข้อมูลที่ถูกชี้ด้วยตัวชี้ตำแหน่งบนสุดของ Stack ออกมา

sNOP : ไม่มีการกระทำใดๆ ทั้งสิ้น

คำสั่งควบคุมการทำงานของ Stack จะถูกประมวลผล เมื่อสัญญาณเลือก (CS) อยู่ในสถานะขอบขาขึ้น ซึ่งเป็นการทำให้การควบคุมการทำงานของ Stack กระทำได้ดีขึ้น

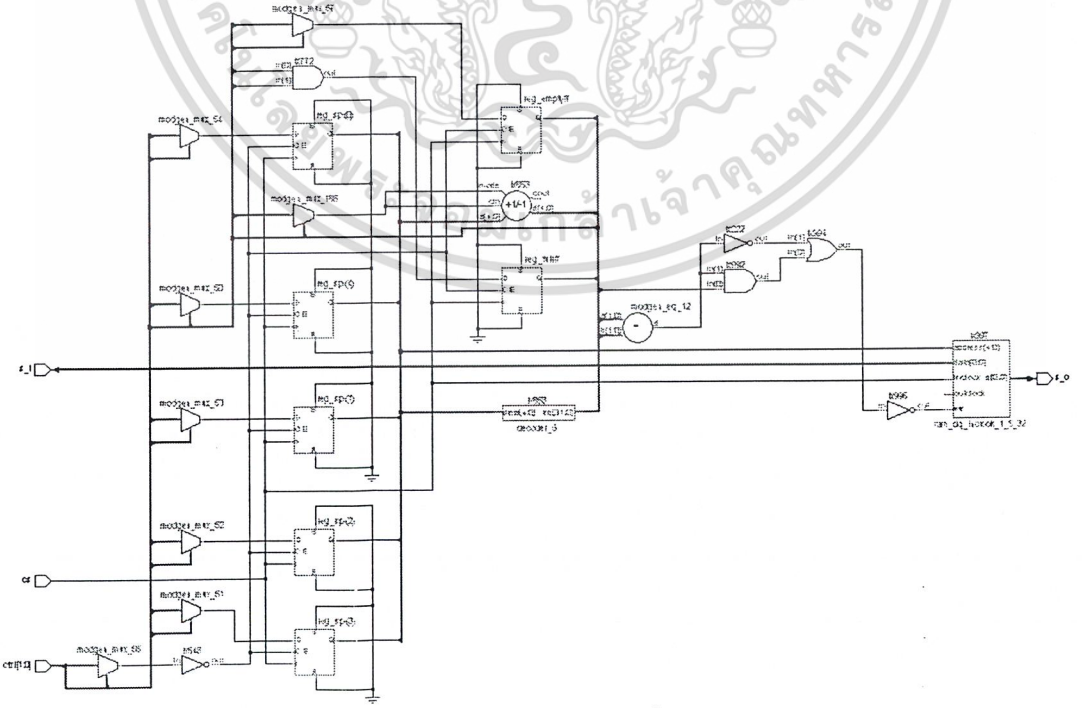


รูปที่ 6.28-1 แสดงไทมมิ่งของส่วนไทมเมอร์และเคาท์เตอร์

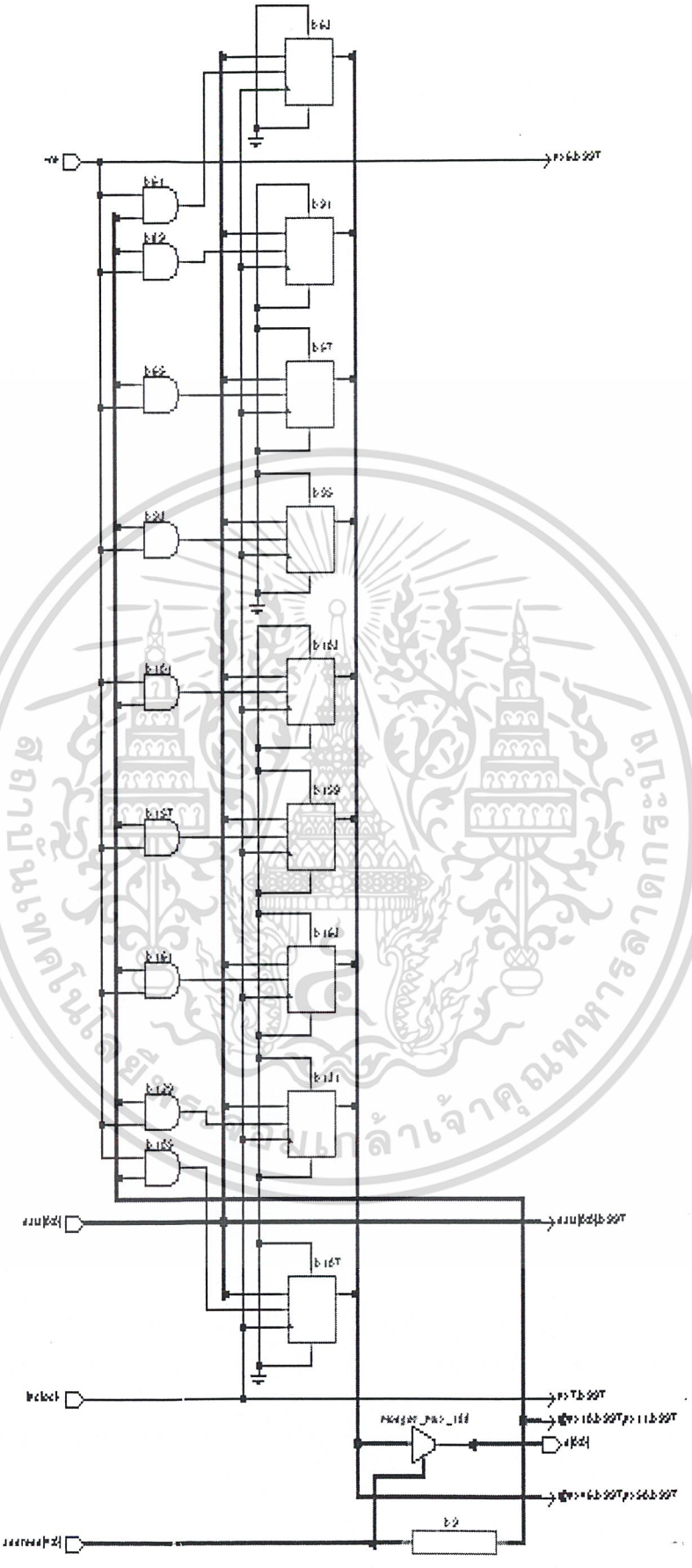


รูปที่ 6.28-2 แสดงไทมมิ่งของส่วนไทมเมอร์และเคาท์เตอร์ (ต่อ)

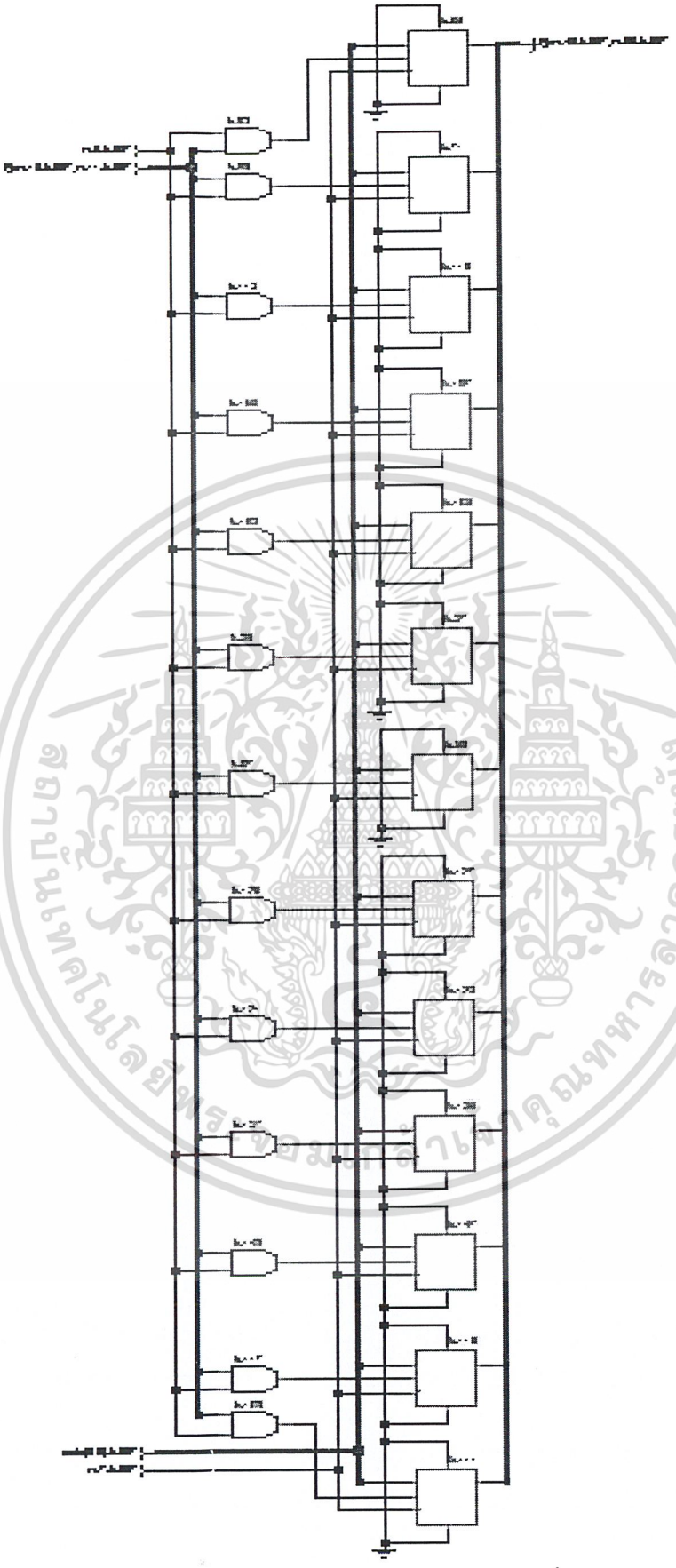
จากรูป 6.28-1 และ 6.28-2 แสดงไทมมิ่งของส่วนไทมเมอร์และเคาท์เตอร์ จะสังเกตได้ว่า สัญญาณเลือกจากภายนอก (sc) จะมีการเปลี่ยนแปลงในขณะที่สัญญาณ CLK อยู่ในสถานะขอบขาลง เนื่องจากเป็นช่วงเวลาที่ สัญญาณข้อมูลที่ต้องทำงานร่วมด้วยมีความเสถียร เพราะสัญญาณดังกล่าวมีการเปลี่ยนเฉพาะเวลาที่สัญญาณ CLK อยู่ในสถานะขอบขาขึ้น



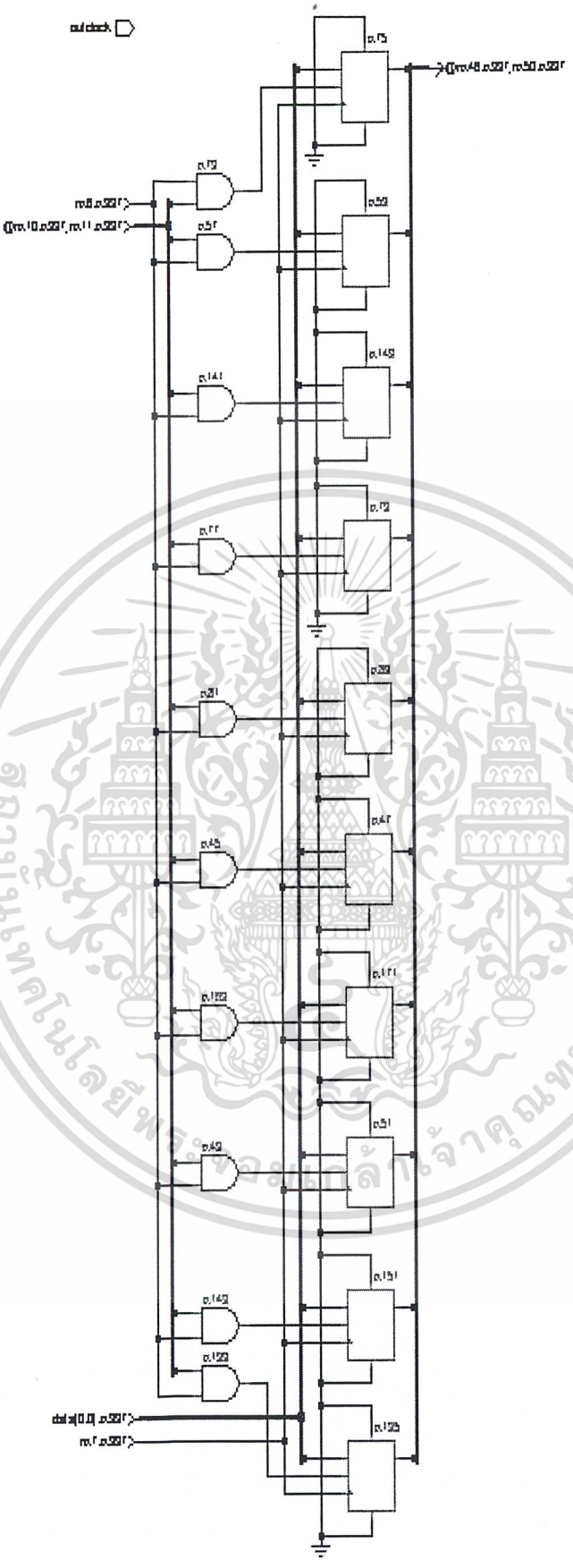
รูปที่ 6.29 แสดงผลของลายวงจรของ Stack ที่ได้จากการสังเคราะห์



รูปที่ 6.30-1 แสดงผลของลายวงจรของส่วนหน่วยความจำของ Stack ที่ได้จากการสังเคราะห์



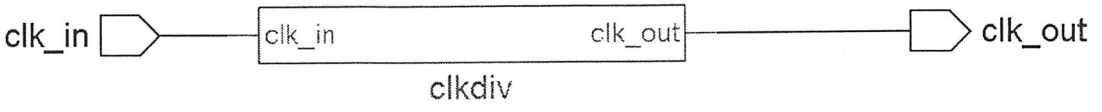
รูปที่ ค.20 แสดงผลลัพธ์ของส่วนหน่วยความจำของ Stack ที่ได้จากการสังเคราะห์ (ต่อ)



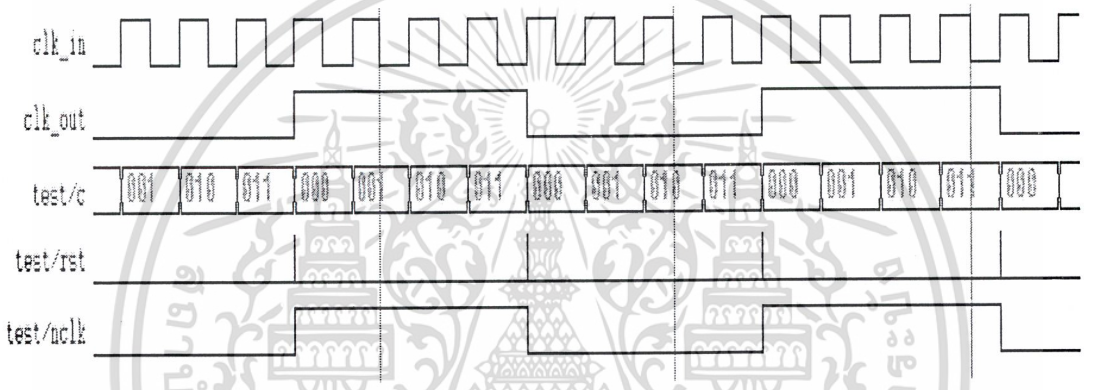
รูปที่ 6.30-3 แสดงผลของลายวงจรของส่วนหน่วยความจำของ Stack ที่ได้จากการสังเคราะห์ (ต่อ)

6.4.8 ตัวจัดการหารความถี่

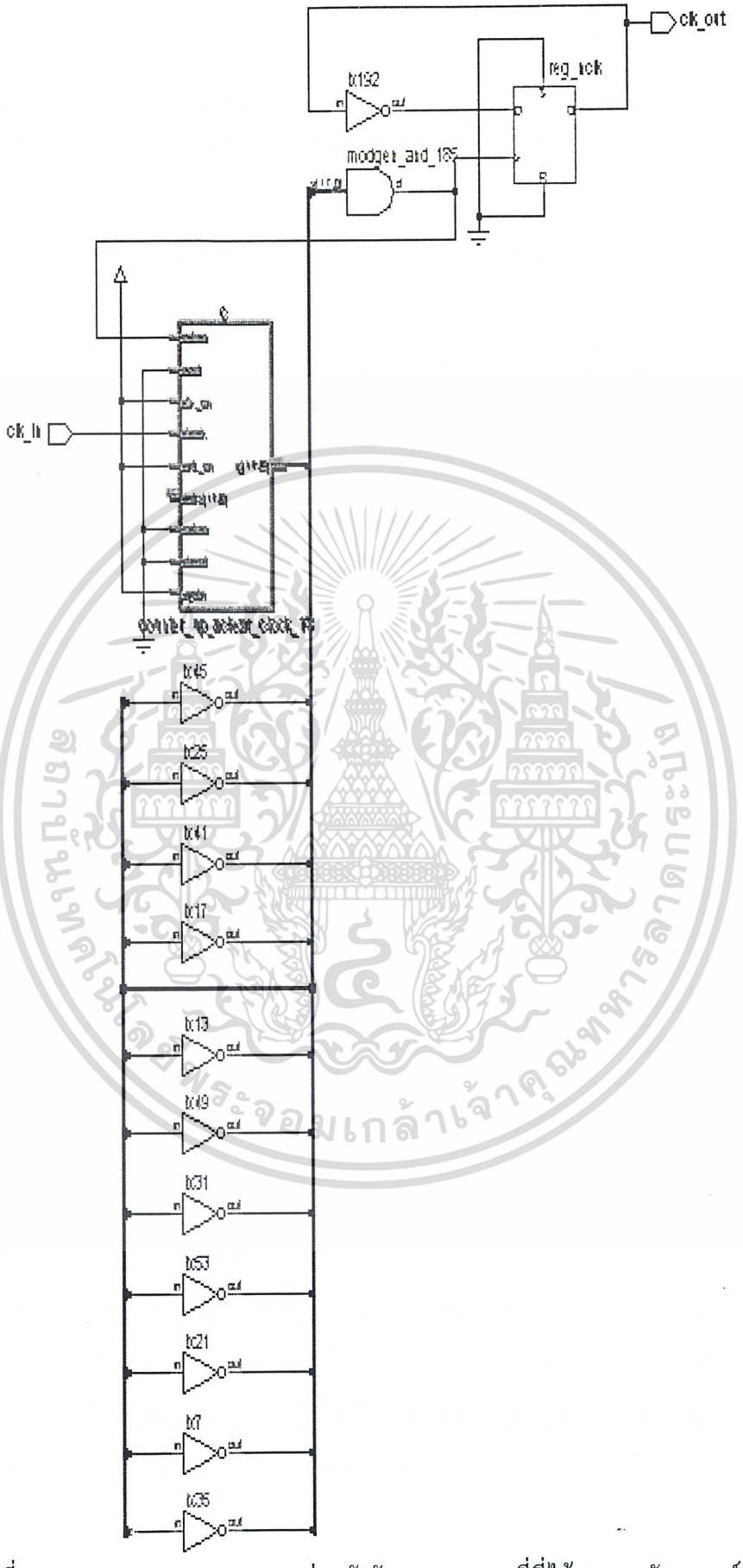
เป็นส่วนที่มีหน้าที่รับความถี่จากสัญญาณนาฬิกาภายนอก เพื่อสร้างความถี่เฉพาะสำหรับอุปกรณ์ภายใน โดยจะรับสัญญาณนาฬิกาภายนอก ซึ่งมีความถี่ 5 MHz ทำการหารความถี่ ด้วย 500,000 เพื่อสร้างสัญญาณนาฬิกาภายใน มีความถี่ 10 Hz (0.1s per clock) ให้กับส่วนควบคุมไทม์เมอร์และเคาท์เตอร์



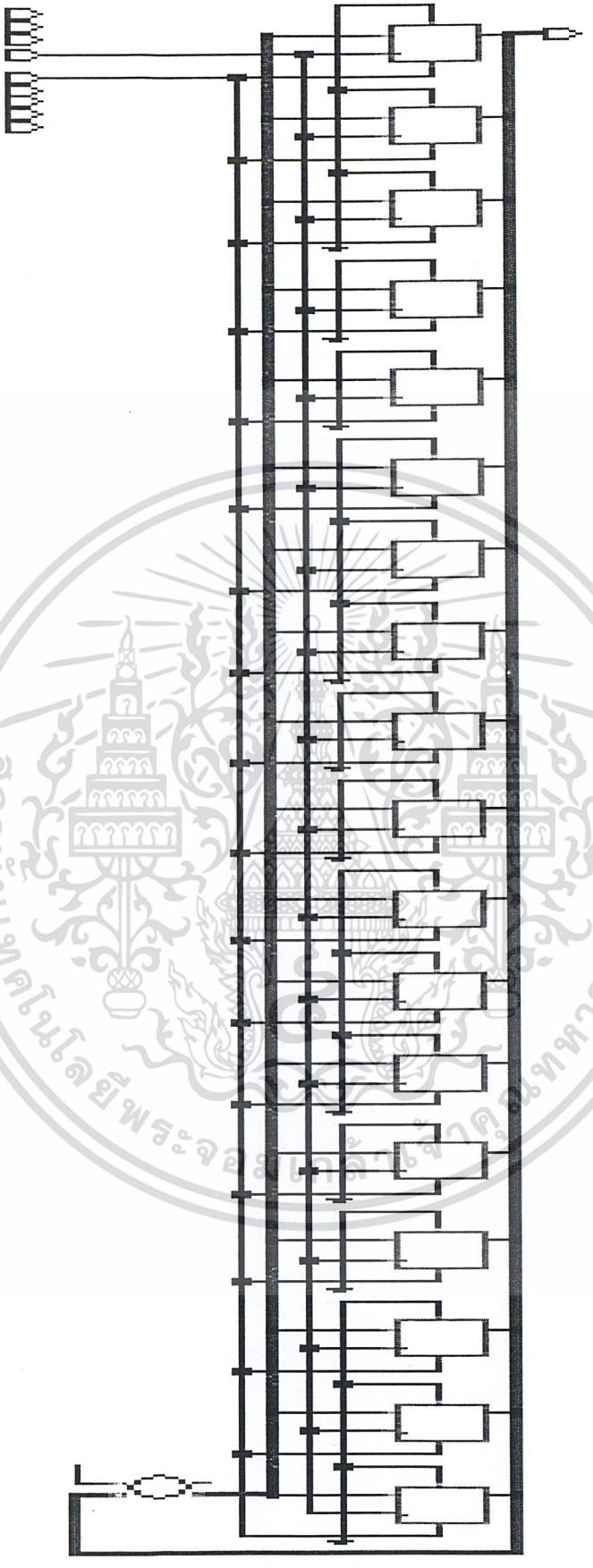
รูปที่ 6.31 แสดงขาอินพุตและเอาต์พุตของตัวจัดการหารความถี่



รูปที่ 6.32 แสดงไทมมิ่งของตัวจัดการหารความถี่



รูปที่ 6.33 แสดงผลของลายวงจรของส่วนตัวจัดการหารความถี่ที่ได้จากการสังเคราะห์



รูปที่ 6.34 แสดงผลของลายวงจรของส่วนตัวจัดการหารความถี่ที่ได้จากการสังเคราะห์อีกระดับหนึ่ง

6.4.9 ALU (Arithmetic and Logic Unit)

เป็นส่วนที่ทำหน้าที่ในการประมวลผลการทำงานหลักของ CPU โดยสามารถทำการประมวลผลคำสั่ง ดังต่อไปนี้ได้

LD จะทำการนำค่าจากสัญญาณอินพุต scr_a ไปเก็บที่ acc เพื่อเป็นสัญญาณเอาต์พุตต่อไป

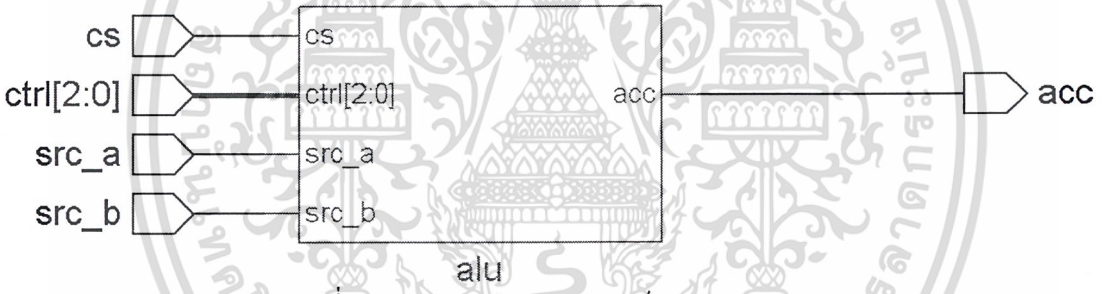
LDI จะทำการนำค่าจากสัญญาณอินพุต scr_a ไปทำการ inverse แล้วเก็บที่ acc เพื่อเป็นสัญญาณเอาต์พุตต่อไป

AND จะทำการนำค่าจากสัญญาณอินพุต scr_a ไปทำการ AND กับค่าจากสัญญาณอินพุต scr_b แล้วนำผลที่ได้เก็บที่ acc เพื่อเป็นสัญญาณเอาต์พุตต่อไป

ANI จะทำการนำค่าจากสัญญาณอินพุต scr_a ไปทำการ AND กับค่าจากการ inverse ค่าจากสัญญาณอินพุต scr_b แล้วนำผลที่ได้เก็บที่ acc เพื่อเป็นสัญญาณเอาต์พุตต่อไป

OR จะทำการนำค่าจากสัญญาณอินพุต scr_a ไปทำการ OR กับค่าจากสัญญาณอินพุต scr_b แล้วนำผลที่ได้เก็บที่ acc เพื่อเป็นสัญญาณเอาต์พุตต่อไป

ORI จะทำการนำค่าจากสัญญาณอินพุต scr_a ไปทำการ OR กับค่าจากการ inverse ค่าจากสัญญาณอินพุต scr_b แล้วนำผลที่ได้เก็บที่ acc เพื่อเป็นสัญญาณเอาต์พุตต่อไป

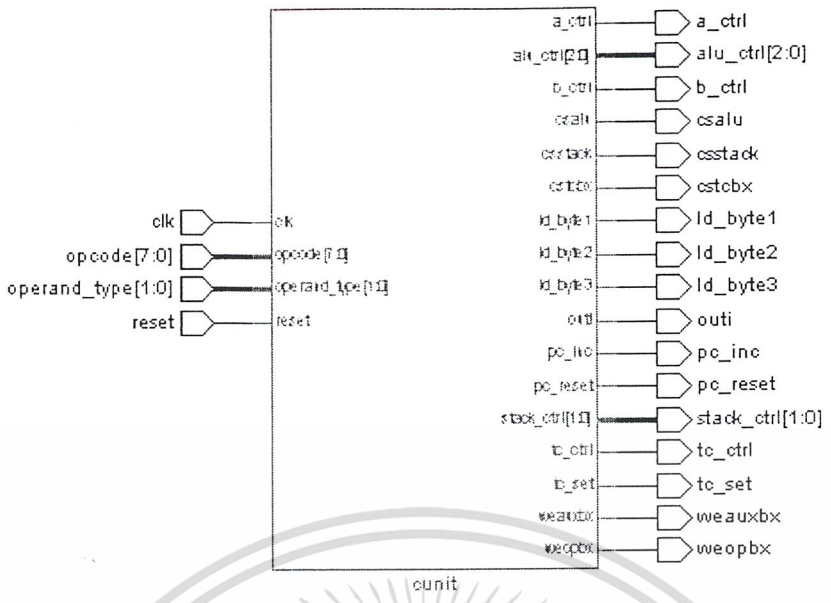


รูปที่ 6.35 แสดงขาอินพุตและเอาต์พุตของ ALU

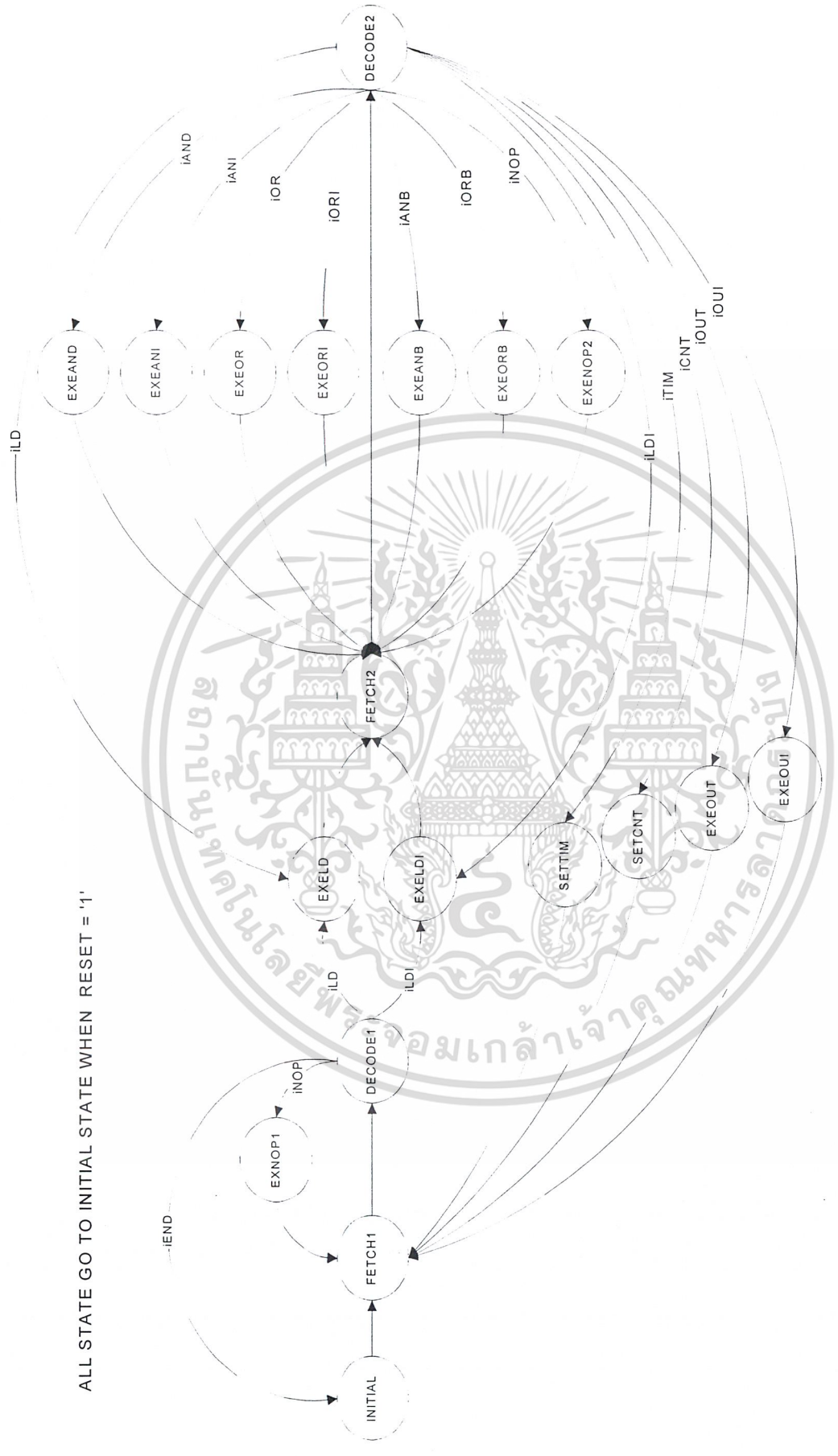
และในส่วนนี้ยังประกอบด้วยส่วนที่ใช้เก็บผลลัพธ์จากการประมวลผลข้างต้น โดยจะใช้ฟลิปฟลอปในการเก็บค่าของข้อมูล ซึ่งในการเก็บข้อมูล สัญญาณเลือก (CS) จะต้องอยู่ในสถานะขอบขาขึ้น และสัญญาณควบคุม (ALU_CTRL) จะต้องทำงานเป็น No Operation หรือสัญญาณควบคุมจะต้องมีค่าเป็น ‘110’ หรือ ’111’

6.4.10 Control Unit

เป็นส่วนที่ทำหน้าที่ควบคุมดูแลการทำงานของอุปกรณ์ต่าง ๆ ภายในตัว CPU โดยจะสร้างสัญญาณควบคุมไปยังส่วนอื่น ๆ เพื่อให้การทำงานเป็นไปตามขั้นตอนดังที่ได้ออกแบบไว้



รูปที่ 6.36 แสดงขาอินพุตและเอาต์พุตของ Control Unit



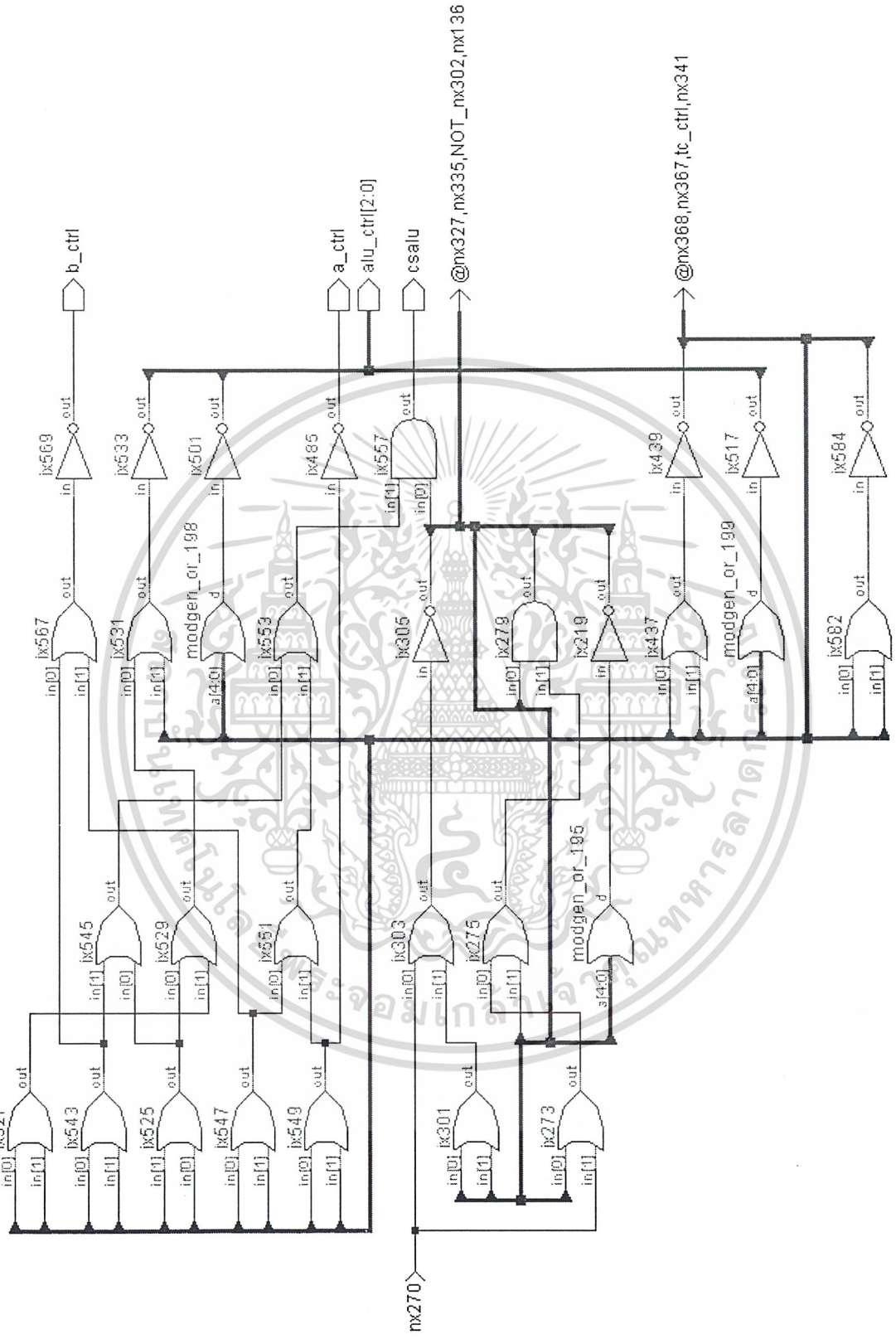
รูปที่ 6.37 แสดง State Diagram ของ Control Unit

ตารางที่ 6.2 แสดงการทำงานของแต่ละ State ใน State Diagram ของ Control Unit

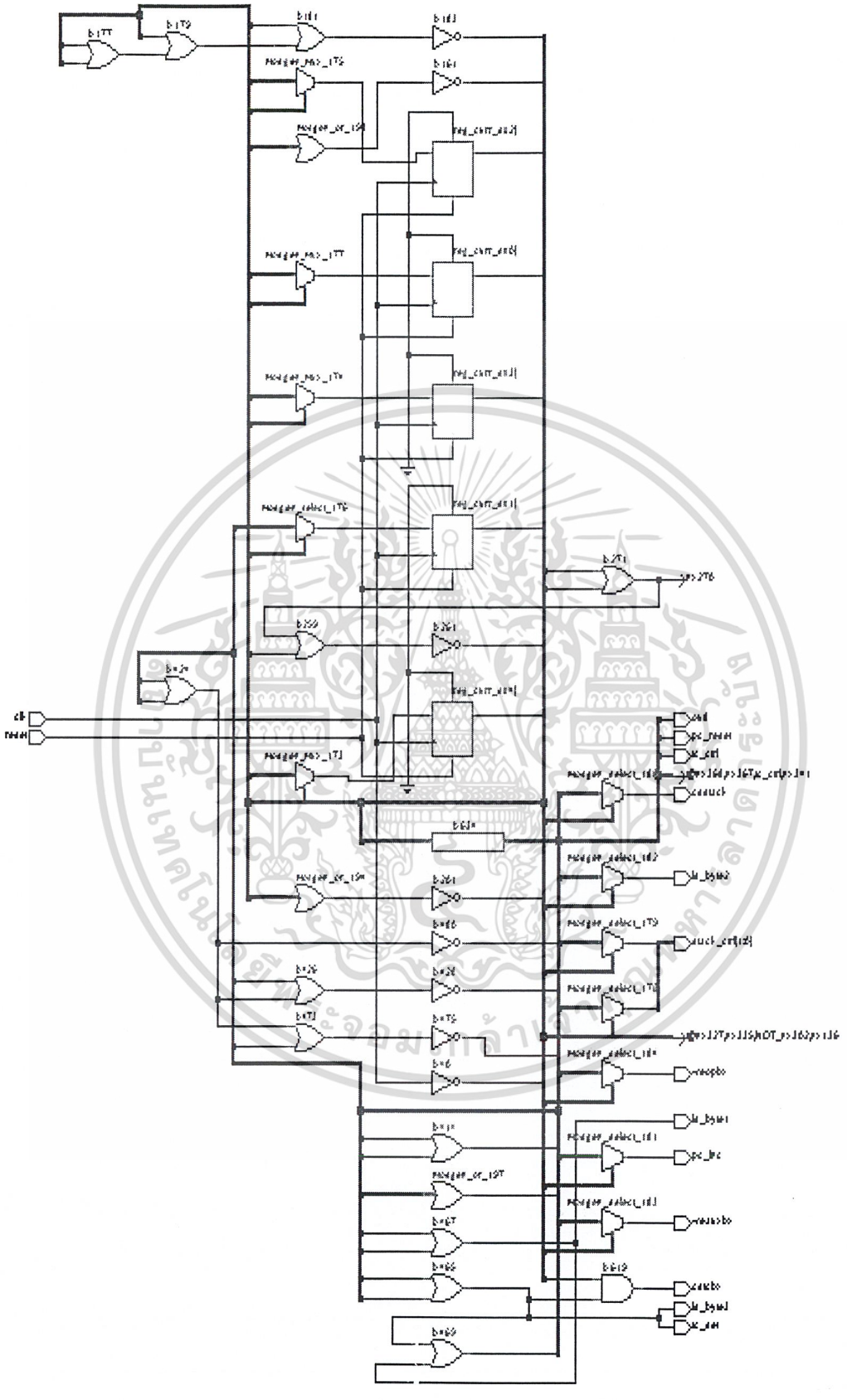
State	Detail	Events	CSalu	ALU_CTRL	PC_RESET	PC_INC	LD_BYTE1	LD_BYTE2	LD_BYTE3	CSstack	STACK_CTRL	CStcbx	TC_CTRL	TC_SET	WEauxbx	WEopbx	A_CTRL	B_CTRL	OUTI	Next state
INIT	Initial State to set initial configuration and set initial address		0	aNOP	1*	0	0	0	0	N*	sCLR*	0	tc_C	0	0	0	1	1	0	FETCH1
FETCH1	Fetch first byte of each instruction		0	aNOP	0	N*	1*	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	DECODE1
FETCH2	Fetch first byte of each instruction		0	aNOP	0	N*	1*	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	DECODE2
DECODER1	For Decode first byte of code and Fetch second byte of code	iLD	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXLD
		iLDI	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXLDI
		iEND	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	INIT
		iNOP	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXNOP1
		OTHER	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	ERR
DECODER2	For Decode first byte of code and Fetch second byte of code	iLD	0	aNOP	0	N*	0	1*	0	N*	sPUSH*	0	tc_C	0	0	0	1	1	0	EXLD
		iLDI	0	aNOP	0	N*	0	1*	0	N*	sPUSH*	0	tc_C	0	0	0	1	1	0	EXLDI
		iAND	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXAND
		iANI	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXANI
		iOR	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXOR
		iORI	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXORI
		iOUT	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXOUT
		iOUI	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXOUI
		iANB	0	aNOP	0	0	0	0	0	N*	sPOP*	0	tc_C	0	0	0	1	1	0	EXANB
		iORB	0	aNOP	0	0	0	0	0	N*	sPOP*	0	tc_C	0	0	0	1	1	0	EXORB
		iTIM	0	aNOP	0	N*	0	1*	0	0	sNOP	0	tc_C	0	0	0	1	1	0	SETTIM
		iCNT	0	aNOP	0	N*	0	1*	0	N*	sPOP*	0	tc_C	0	0	0	1	1	0	SETCNT
		iNOP	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	EXNOP2
		OTHER	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	ERR

State	Detail	Events	CSalu	ALU_CTRL	PC_RESET	PC_INC	LD_BYTE1	LD_BYTE2	LD_BYTE3	CSstack	STACK_CTRL	CStcbx	TC_CTRL	TC_SET	WEauxbx	WEopbx	A_CTRL	B_CTRL	OUTI	Next state
EXLD	State to Execute LD		N*	aLD	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	0*	1	0	FETCH2
EXLDI	State to Execute LDI		N*	aLDI	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	0*	1	0	FETCH2
EXAND	State to Execute AND		N*	aAND	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	0*	0	FETCH2
EXANI	State to Execute ANI		N*	aANI	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	0*	0	FETCH2
EXOR	State to Execute OR		N*	aOR	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	0*	0	FETCH2
EXORI	State to Execute ORI		N*	aORI	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	0*	0	FETCH2
EXOUT	State to Execute OUT	rOP	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	N*	1	1	0	FETCH1
		rAUX	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	N*	0	1	1	0	FETCH1
EXOUI	State to Execute OUI	rOP	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	N*	1	1	1*	FETCH1
		rAUX	0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	N*	0	1	1	1*	FETCH1
EXANB	State to Execute ANB		N*	aAND	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	FETCH2
EXORB	State to Execute ORB		N*	aOR	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	FETCH2
SETTIM	State to Execute TIM		0	aNOP	0	N*	0	0	1*	0	sNOP	N*	tc_T	1*	0	0	1	1	1*	FETCH1
SETCNT	State to Execute CNT		0	aNOP	0	N*	0	0	1*	0	sNOP	N*	tc_C	1*	0	0	1	1	0	FETCH1
EXNOP1	State to Execute NOP		0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	FETCH1
EXNOP2	State to Execute NOP		0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	FETCH2
OTHER	Have error		0	aNOP	0	0	0	0	0	0	sNOP	0	tc_C	0	0	0	1	1	0	ERR

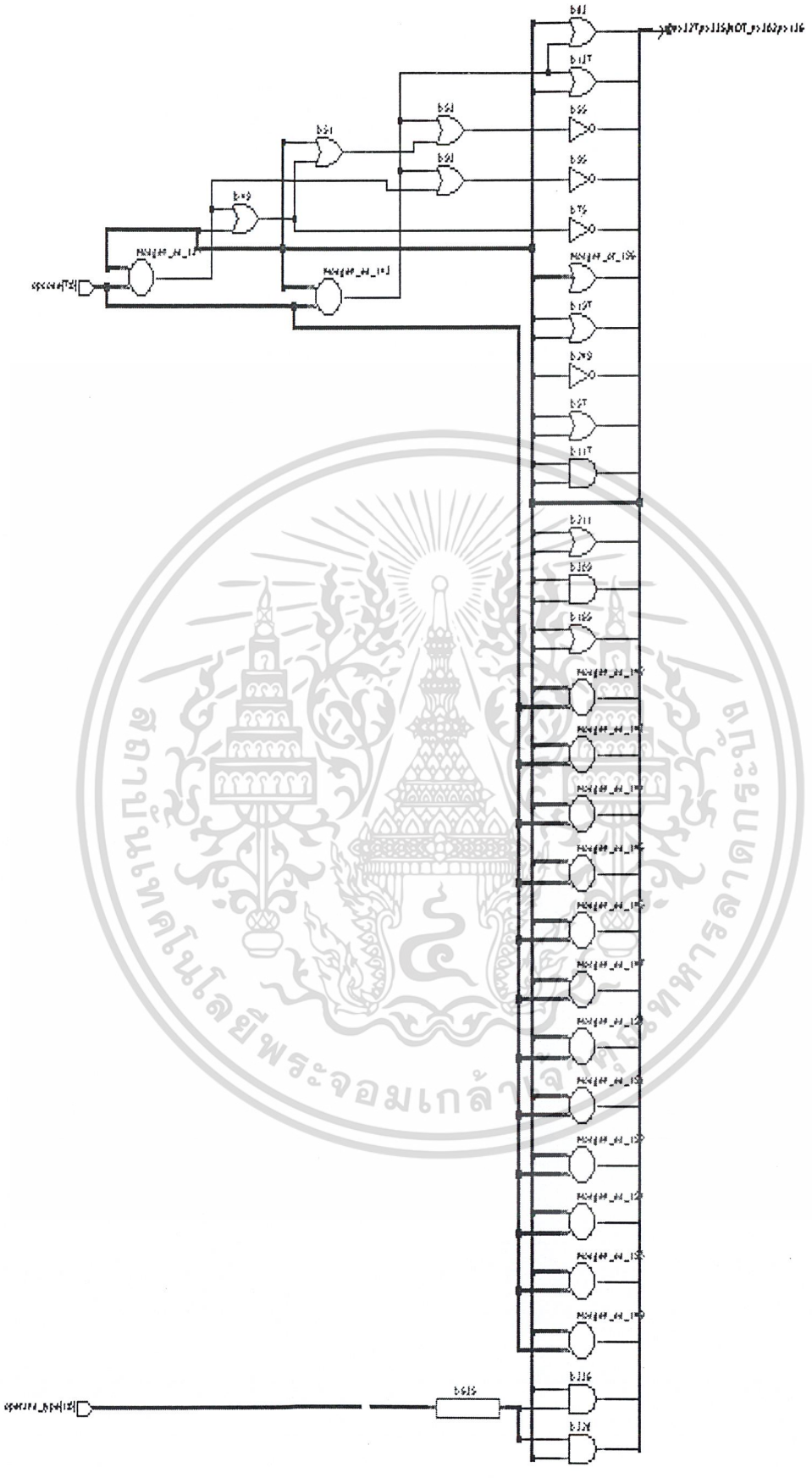
* signal change
- N : not clk
- If reset then go to INITIAL State



รูปที่ 6.38-1 แสดงผลของลายวงจรของ Control Unit ที่ได้จากการสังเคราะห์



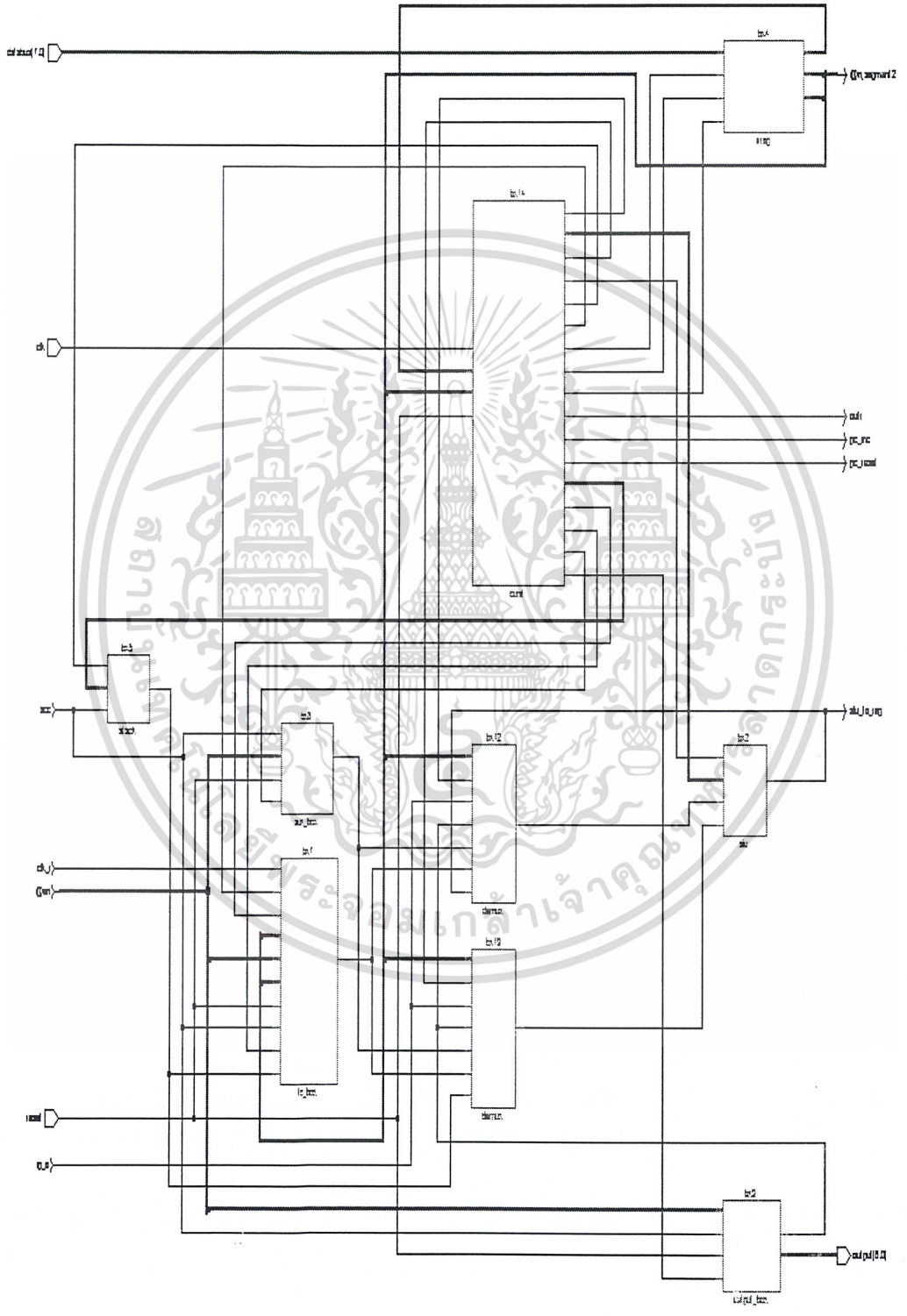
รูปที่ 6.38-2 แสดงผลของลายวงจรของ Control Unit ที่ได้จากการสังเคราะห์ (ต่อ)



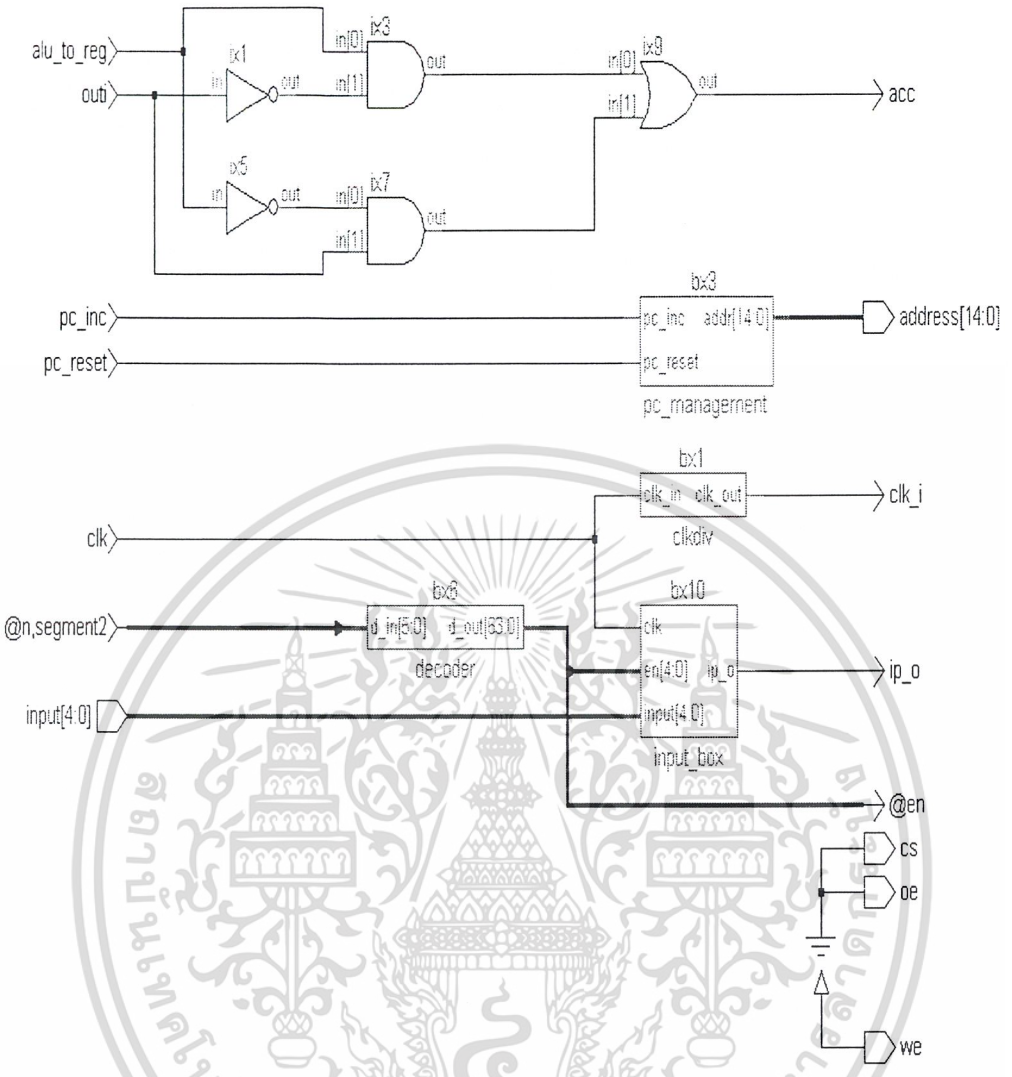
รูปที่ 6.38-3 แสดงผลของลายวงจรของ Control Unit ที่ได้จากการสังเคราะห์ (ต่อ)

6.5 การประกอบวงจรรวม

เมื่อทำการสร้างและทดสอบแต่ละส่วนประกอบเรียบร้อยแล้ว ขั้นตอนสุดท้ายคือการนำส่วนประกอบทั้งหมดมาทำการประกอบเข้าด้วยกัน แล้วทำการทดสอบอีกทีหนึ่ง



รูปที่ 6.39-1 แสดงผลของลายวงจรของ CPU ที่ได้จากการสังเคราะห์



รูปที่ 6.39-2 แสดงผลของลายวงจรของ CPU ที่ได้จากการสังเคราะห์ (ต่อ)

บทที่ 7
การทดสอบ

7.1 วัตถุประสงค์

วัตถุประสงค์ในการทดสอบของโครงงานนี้เป็นการทำเพื่อทดสอบการทำงานของตัวควบคุมหลักของ PLC โดยในการทดสอบต้องทำการทดสอบทุกฟังก์ชั่นการใช้งานที่ได้ออกแบบตามที่ระบุไว้ในบทที่ 6 ดังนี้

ขนาดและความสามารถของ PLC

- อินพุตจำนวน 5 อินพุต โดยทำการเชื่อมต่อ (Map) เข้ากับพอร์ตขนานของบอร์ดทดสอบ
- เอาต์พุตจำนวน 7 เอาต์พุต โดยทำการเชื่อมต่อเข้ากับ 7 Segment ของบอร์ดทดสอบ
- รีเลย์ช่วย (Auxiliary Relay) จำนวน 64 ตัว
- ไทมเมอร์ หรือเคาท์เตอร์ จำนวน 4 ตัว

คำสั่งที่สามารถประมวลผลได้ตามคำสั่งพื้นฐานของ PLC ได้แก่คำสั่ง LD, LDI, AND, ANI, OR, ORI, OUT, OUI, ANB, ORB, TIM, CNT, NOP, END

ในการทดสอบจะเน้นไปที่การทดสอบการประมวลผลของชุดคำสั่งต่าง ๆ ว่าสามารถทำงานได้อย่างถูกต้อง

7.2 ขั้นตอนการทดสอบและผลการทดสอบ

ในการทดสอบจะทำการทดสอบโดยเป็นชุดคำสั่งเพื่อทดสอบการใช้งานทุกคำสั่ง และทดสอบการประมวลผลของคำสั่งมากกว่าหนึ่งคำสั่ง ความสามารถในการประมวลผลคำสั่งแบบต่อเนื่องกันเพื่อมีสภาพเหมือนกันการนำไปใช้งานจริง

ทำการทดสอบแต่ละชุดคำสั่งดังนี้

1. กลุ่มชุดคำสั่งที่ 1 คำสั่ง LD (Load) คำสั่ง LDI (Load Inverse) คำสั่ง OUT (Out) คำสั่ง OUI (Out Inverse) และคำสั่ง END (End) โดยมีชุดคำสั่งดังนี้

HEX CODE		MNUMONIC	
00	01	LD	X01
06	41	OUT	Y01
01	02	LDI	X02
06	42	OUT	Y02
00	03	LD	Y03
07	43	OUI	Y03
01	04	LDI	X04
07	44	OUI	Y04
ff		END	

2. กลุ่มชุดคำสั่งที่ 2 คำสั่ง LD (Load) คำสั่ง OUT (Out) คำสั่ง AND (And) คำสั่ง ANI (And Inverse) คำสั่ง OR (Or) คำสั่ง ORI (Or Inverse) และคำสั่ง END (End) โดยมีชุดคำสั่งดังนี้

HEX CODE		MNUMONIC	
00	00	LD	X00
02	81	AND	M00
06	40	OUT	Y00
00	00	LD	X00
03	81	ANI	M01
06	41	OUT	Y01
00	00	LD	X00
04	81	OR	M02
06	42	OUT	Y02
00	00	LD	X00
05	81	ORI	M03
06	43	OUT	Y03
ff		END	

3. กลุ่มชุดคำสั่งที่ 3 คำสั่ง LD (Load) คำสั่ง OUT (Out) คำสั่ง AND (And) คำสั่ง ANI (And Inverse) คำสั่ง OR (Or) คำสั่ง ORI (Or Inverse) และ คำสั่ง END (End) โดยมีชุดคำสั่งดังนี้

HEX CODE		MNUMONIC	
00	00	LD	X00
02	01	AND	X01
02	02	AND	X02
06	40	OUT	Y00
00	00	LD	X00
03	01	ANI	X01
03	02	ANI	X02
06	41	OUT	Y01
00	00	LD	X00
04	01	OR	X01
04	02	OR	X02
06	42	OUT	Y02
00	00	LD	X00
05	01	ORI	X01
05	02	ORI	X02
06	43	OUT	Y03
ff		END	

4. กลุ่มชุดคำสั่งที่ 4 คำสั่ง LD (Load) คำสั่ง OUT (Out) คำสั่ง AND (And) คำสั่ง ANI (And Inverse) คำสั่ง OR (Or) คำสั่ง ORI (Or Inverse) คำสั่ง ORB (Or Block) คำสั่ง ANB (And Block) และคำสั่ง END (End) โดยมีชุดคำสั่งดังนี้

HEX CODE		MNUMONIC	
00	00	LD	X00
02	01	AND	X01
00	02	LD	X02
03	03	ANI	X03
80		ANB	
06	40	OUT	Y00
00	00	LD	X00
04	01	OR	X01
00	02	LD	X02
05	03	ORI	X03
81		ORB	
06	41	OUT	Y01
ff		END	

5. กลุ่มชุดคำสั่งที่ 5 คำสั่ง LD (Load) คำสั่ง OUT (Out) คำสั่ง TIM (Timer) และคำสั่ง END (End) ทำการทดสอบโดยป้อนคำสั่งต่อไปนี้

HEX CODE		MNUMONIC	
00	00	LD	X00
02	81	AND	M00
40	00	TIM	00
0a			10
00	c0	LD	T00
06	40	OUT	Y00
ff		END	

6. กลุ่มชุดคำสั่งที่ 6 คำสั่ง LD (Load) คำสั่ง OUT (Out) คำสั่ง CNT (Counter) และ คำสั่ง END (End) ทำการทดสอบโดยป้อนคำสั่งต่อไปนี้

HEX CODE		MNUMONIC	
00	00	LD	X00
04	81	OR	M00
00	01	LD	X01
41	c0	CNT	01
03			3
00	c3	LD	T01
06	40	OUT	Y00
ff		END	

7.3 สรุปผลการทดสอบ

ผลการทดสอบแต่ล่ะคำสั่งสามารถทำงานได้ผลดีตรงตามที่ได้ทำการออกแบบไว้ตามมาตรฐานของ PLC ที่ได้ออกแบบไว้ดังข้างต้น

โดยคำสั่ง LD และ LDI สามารถนำค่าสถานะจากอินพุตรีเลย์ หรือรีเลย์ช่วย หรือจากเอาต์พุตรีเลย์ เข้ามาเพื่อทำการเปิดแถวการเชื่อมต่อรีเลย์ใด ๆ ใหม่ และทำการประมวลผลได้

คำสั่ง AND, ANI สามารถนำค่าสถานะจากอินพุตรีเลย์หรือรีเลย์ช่วย หรือจากเอาต์พุตรีเลย์ เข้ามาเพื่อทำการกระทำทางลอจิกแอนด์ หรือการนำรีเลย์มาเชื่อมต่อวงจรแบบอนุกรมได้

คำสั่ง OR, ORI สามารถนำค่าสถานะจากอินพุตรีเลย์หรือรีเลย์ช่วย หรือจากเอาต์พุตรีเลย์ เข้ามาเพื่อทำการกระทำทางลอจิกออ หรือการนำรีเลย์มาเชื่อมต่อวงจรแบบขนานได้

คำสั่ง ANB และ ORB สามารถทำการเชื่อมต่อวงจรแบบอนุกรม และแบบขนานได้

คำสั่ง TIM สามารถทำการนับค่าเวลาได้โดยค่าเวลาที่น้อยที่สุดที่สามารถนับได้คือ 1 เดซิวินาที และเมื่อทำการให้สัญญาณรีเซทจะทำการเริ่มต้นนับเวลาใหม่ โดยจะไม่เริ่มต้นนับค่าต่อจนกว่าสัญญาณรีเซทจะมีการเปลี่ยนแปลง

คำสั่ง CNT สามารถทำการนับค่าสัญญาณที่เข้ามาจากรีเลย์ใด ๆ ได้และเมื่อทำการให้สัญญาณรีเซทจะทำการเริ่มต้นนับเวลาใหม่ โดยจะไม่เริ่มต้นนับค่าต่อจนกว่าสัญญาณรีเซทจะมีการเปลี่ยนแปลงเช่นเดียวกันคำสั่ง TIM

7.4 บทวิจารณ์

โครงงานนี้เป็นโครงงานที่มีจุดประสงค์ในการทดลองสร้างตัวควบคุมของ PLC โดยใช้ FPGA ซึ่งต้องสามารถทำงานตามคำสั่งพื้นฐานของ PLC ได้ และต้องพิจารณาถึงวัตถุประสงค์ของโครงงานดังนี้

1 เพื่อศึกษาโครงสร้างและการใช้งานของ FPGA

2 เพื่อให้สามารถวิเคราะห์ ออกแบบ และพัฒนาวงจรใน FPGA โดยใช้ภาษา VHDL

3 ทดลองการออกแบบวงจร PLC ที่มีประสิทธิภาพในราคาที่ถูกลง

และจากการออกแบบและทดสอบในโครงงานนี้ ซึ่งสามารถสรุปผลตามจุดประสงค์ของโครงงานได้คือ มีความเข้าใจในการออกแบบวงจรโดยใช้ภาษา VHDL ใน FPGA ได้เป็นอย่างดี และพิจารณาถึงตัวชิ้นงานได้ดังนี้

จุดเด่น

- มีขนาดเล็กเมื่อเปรียบเทียบกับ PLC ในรุ่นที่นำมาเป็นตัวอย่าง
- สามารถทำการโปรแกรมภาษาแลดเดอร์ลงไปในตัวควบคุมได้ง่ายกว่า โดยสามารถทำการป้อนโดยผ่านทางพอรต์ขนานของคอมพิวเตอร์ และสามารถใช้ตัวซอฟต์แวร์แอสแซมเบลอร์ที่สร้างทำการแปลงโค้ดภาษาแลดเดอร์เป็น HEX โค้ดเพื่อโปรแกรมลง FPGA ได้ (ดูภาคผนวก ก. ซอฟต์แวร์แอสแซมเบลอร์ภาษาแลดเดอร์)
- มีความสามารถในการโปรแกรมแบบออฟลายน์ (Off line) ได้เช่นเดียวกับ PLC บางรุ่นโดยจากความสามารถในการโปรแกรมผ่านทางคอมพิวเตอร์ข้างต้น
- ถ้ามีการนำไปผลิตจริงโดยนำลายวงจรจากภายในตัว FPGA ไปทำการสร้างจะสามารถสร้างตัวควบคุมที่มีราคาถูกกว่าได้

จุดด้อย

- จำนวนคำสั่งที่สามารถประมวลผลได้มีจำนวนน้อยกว่า PLC รุ่นที่นำมาเป็นตัวอย่าง โดยตัดส่วนคำสั่งที่เป็นการคำนวณนอกเหนือจากการเชื่อมต่อรีเลย์ออกไป
- มีความสะดวกในการนำไปใช้งานน้อยจริงกว่า เนื่องจากถูกต่ออยู่กับบอร์ดทดลอง
- มีจำนวนรีเลย์ที่ใช้งานน้อยกว่าเนื่องจากขีดจำกัดของตัวบอร์ดทดสอบและในการทดสอบ แต่สามารถขยายเพิ่มได้ง่ายตามขนาดของ FPGA

- มีจำนวนไทม์เมอร์และเคาท์เตอร์น้อยกว่า

ภาคผนวก ก. ซอฟต์แวร์แอสแซมเบลอร์ภาษาแลดเดอร์

ซอฟต์แวร์แอสเซมเบลอร์ (Assembler) คือโปรแกรมที่ทำหน้าที่ในการแปลงโค้ดโปรแกรมในลักษณะ Mnemonic Code ให้กลายเป็น Machine Code ที่หน่วยประมวลผลสามารถนำไปประมวลผลได้ต่อไป

ในโครงงานนี้มีการจัดทำส่วนซอฟต์แวร์แอสเซมเบลอร์ภาษาแลดเดอร์ ขึ้นมาเพื่ออำนวยความสะดวกในการแปลงจากโค้ดภาษาแลดเดอร์ให้เป็น HEX โค้ดเพื่อนำลงบอร์ดทดลองอีกต่อไป โดยความสามารถของซอฟ์แวร์จะมีการทำงานใกล้เคียงกับซอฟต์แวร์แอสเซมเบลอร์ทั่ว ๆ ไป

การใช้งานซอฟต์แวร์แอสแซมเบลอร์ภาษาแลดเดอร์

การใช้งานสามารถทำได้โดยพิมพ์คำสั่งที่ Shell Command ของระบบปฏิบัติการซึ่งซอฟต์แวร์ชุดนี้สามารถทำงานได้ทั้งระบบปฏิบัติการวินโดวส์ (Windows) และลินุกส์ (Linux)

ในระบบปฏิบัติการวินโดวส์ NT, 2000 หรือ XP ให้เข้า Shell Command ที่ Start Menu → Accessories → Command Prompt ส่วนระบบปฏิบัติการวินโดวส์ 95, 98 หรือ ME ให้เข้า Shell Command ที่ Start Menu → Command Prompt ส่วนระบบปฏิบัติการลินุกส์ให้เข้า Shell Command ที่ Terminal

รูปแบบคำสั่งของโปรแกรมได้แก่

```
lad2hex <option> filename…
```

คำสั่งใช้งานในกรณีต้องการแอสเซมเบลอร์อย่างเดียวดังตัวอย่าง

```
lad2hex input.asm
```

จะได้ไฟล์เอาต์พุตเป็นชื่อเดียวกันกับไฟล์อินพุตแต่จะมีนามสกุลของไฟล์เป็น .HEX

ถ้าต้องการกำหนดชื่อของไฟล์เอาต์พุตเองให้ระบุออฟชั่น (Option) –o ดังตัวอย่าง

```
lad2hex -o output.hex input.asm
```

นอกจากนี้ยังมีออฟชั่นอื่น ๆ อีกได้แก่

-o <filename> เป็นการกำหนดชื่อไฟล์เอาต์พุต

-b เป็นการกำหนดให้โปรแกรมหยุดการทำงานทันทีที่พบความผิดพลาดของโค้ด

-a <maxbuffer> เป็นการกำหนดขนาดของบัฟเฟอร์ที่ใช้งาน โดยขนาดของบัฟเฟอรีจะมีผลต่อความเร็วในการแปลงโค้ด ค่าปกติเป็น 969 ไบต์

-h เป็นการแสดงหน้าจอแสดงความช่วยเหลือของโปรแกรม

/? เป็นการแสดงหน้าจอแสดงความช่วยเหลือของโปรแกรม

บรรณานุกรม

หนังสืออ้างอิง

[1] กฤษดา วิศวธีรานนท์ : PC ตัวควบคุมซีเคว้นซ์ หลักการทำงานและการประยุกต์, จุฬาลงกรณ์มหาวิทยาลัย, 2539

[2] Operation Manual SYSMAC C20P/C28P/C40P Programmable Controllers, OMRON

[3] FPGA Design Workshop, ฝ่ายออกแบบวงจรรวม ศูนย์วิจัยและพัฒนาเทคโนโลยีไมโครอิเล็กทรอนิกส์ (TMEC), ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC), สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ

[4] ชาติชาย ดิษฐกุล, คู่มือการใช้ภาษา VHDL

[5] XS40, XSP Board V1.4 User Manual, XESS Cooperation, 2001

[6] XC4000E and XC4000X Series Field Programmable Gate Arrays, XILNX, 1999

[7] HDL Synthesis for design FPGAs design guide, XACT STEP, Xilinx, 1995

โฮมเพจอ้างอิง

[1] Xilinx, Inc
http://www.xilinx.com

[2] XESS Corp.
http://www.xess.com