เครื่องกำเนิดสัญญาณดิจิตอลหลายช่องสัญญาณแบบโปรแกรมได้

MULTI CHANNEL DATA GENERATOR

โดย

นาย ทิวาชัย เมืองจีน

นาย โชคชัย มาลากุล

นาย ศรศักดิ์ ขุนราช

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เครื่องกำเนิดสัญญาณดิจิตอลหลายช่องสัญญาณแบบโปรแกรมได้

MULTI CHANNEL DATA GENERATOR

โดย

นาย ทิวาชัย เมืองจีน 43015014

นาย โชคชัย มาลากุล 43015065

นาย ศรศักดิ์ ขุนราช 43015104

อาจารย์ที่ปรึกษา

ผศ. เกรียงไกร วงศ์โรจนภรณ์

รศ.ดร. สุวิพล สิทธิชีวภาค

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องกำเนิดสัญญาณดิจิตอลหลายช่องสัญญาณแบบโปรแกรมได้

MULTI CHANNEL DATA GENERATOR

ผู้จัดทำ

1. นาย ทิวาชัย เมืองจีน 43015014
2. นาย โชคชัย มาลากุล 43015065
3. นาย ศรศักดิ์ ขุนราช 43015104

_____ อาจารย์ที่ปรึกษา

(ผศ. เกรียงไกร วงศ์โรจนภรณ์)

_____ อาจารย์ที่ปรึกษา

(รศ.ดร. สุวิพล สิทธิชีวภาค)

เครื่องกำเนิดสัญญาณดิจิตอลหลายช่องสัญญาณแบบโปรแกรมได้

MULTI CHANNEL DATA GENERATOR

ผู้จัดทำ นาย ทิวาชัย เมืองจีน 43015014

นายโชคชัย มาลากุล 43015065

นาย ศรศักดิ์ ขุนราช 43015104

อาจารย์ที่ปรึกษา ผศ. เกรียงไกร วงศ์โรจนภรณ์

รศ.ดร. สุวิพล สิทธิชีวภาค

บทคัดย่อ

โครงงานนี้เป็นการนำเสนอ เครื่องกำเนิดสัญญาณดิจิตอลแบบหลายช่องสัญญาณ ที่ใช้อุปกรณ์อิเล็กทรอนิกส์ประเภท FPGA. มาเป็นตัวสร้างและกำเนิดสัญญาณดิจิตอล ซึ่งสามารถกำหนดความถี่และรูปแบบของสัญญาณดิจิตอลแต่ละช่องสัญญาณได้ โดยสามารถโปรแกรมได้ด้วยคอมพิวเตอร์ สัญญาณข้อมูลจากคอมพิวเตอร์จะถูกส่งมาเก็บไว้ในหน่วยความจำซึ่งใช้ไมโครคอนโทรลเลอร์เป็นตัวส่งผ่านข้อมูลระหว่างคอมพิวเตอร์กับ FPGA. เพื่อเป็นข้อมูลให้วงจรดิจิตอล เพื่อกำหนดความถี่ และรูปแบบของสัญญาณดิจิตอลของแต่ละสัญญาณเพื่อนำไปใช้งาน

ABSTRACT

This project presents a multi channel data generator. FPGA. generates digital signals. Frequencies and patterns of digital signals are determined by computer 's software. Digital signals from computer are sent into memory. Microcontroller is a device that transfers data between computer and FPGA. This data causes FPGA. creates digital signal each channel for application.

สารบัญ

หน้า

หน้า

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

บทที่1
บทนำ

ปัจจุบันวงจรอิเล็กทรอนิกส์ที่เป็นแบบอนาลอกได้ถูกพัฒนาและปรับปรุงมาเป็นแบบดิจิตอลเพิ่มมากขึ้นเรื่อยๆ ซึ่งในการคิดค้นและออกแบบวงจรดิจิตอลนั้น จะต้องทดสอบว่าวงจรที่ได้ออกแบบนั้นทำงานได้ถูกต้องตามที่ได้ออกแบบไว้หรือไม่ หรือในการซ่อมแซมวงจรดิจิตอลก็ต้องทดสอบว่าอุปกรณ์ดิจิตอลนั้นเสียหรือไม่ เพื่อทดลองและทดสอบดังที่กล่าวไว้แล้วนั้น เราต้องทำการป้อนอินพุทที่ออกแบบไว้ให้กับอุปกรณ์และวงจรเหล่านั้นแล้วดูว่าสัญญาณเอาท์พุทที่ออกมานั้นถูกต้องหรือไม่ ซึ่งอุปกรณ์ดิจิตอลหรือวงจรดิจิตอลนั้นส่วนใหญ่จะมีหลายอินพุทและหลายเอาท์พุท

ปริญญานิพนธ์ฉบับนี้จึงมีความคิดที่จะสร้างเครื่องกำเนิดสัญญาณดิจิตอลขึ้น ที่สามารถกำหนดรูปแบบและความถี่ของสัญญาณได้และมีจำนวนของช่องสัญญาณที่เหมาะสม เพื่อสนองความต้องการแก่ผู้ใช้งาน

อุปกรณ์หลักที่นำมาใช้งานในโครงงานนี้คือ ไอซี เอฟพีจีเอ. (FPGA.) ซึ่งนำมาใช้ออกแบบเป็นวงจรดิจิตอลที่ใช้ภายในโครงงานนี้ เพราะสามารถพัฒนาเป็นวงจรดิจิตอลได้หลากหลาย และไมโครคอนโทรลเลอร์ตระกูล เอ็มซีเอส-51 (MCS-51) ซึ่งนำมาใช้ในการออกแบบให้ทำงานร่วมกัน

บทที่ 2
ทฤษฎีหรือหลักการ

2.1 เทคโนโลยีของ FPGA

2.1.1 Field-Programmable Gate Array (FPGA)

FPGA เป็นอุปกรณ์ในลักษณะของ Programmable Device ที่เราสามารถโปรแกรมตัวมันให้เป็นวงจรดิจิตอลใดๆก็ได้ สำหรับ FPGA แล้วนับว่าเป็นอุปกรณ์ตัวใหม่ในตระกูลของ ASIC ซึ่งมีการเจริญเติบโตอย่าง รวดเร็วและมีบทบาทที่สำคัญในการเข้ามาแทนที่ระบบอิเล็กทรอนิกส์ที่ใช้ TTL โครงสร้างภายในของ FPGA ประกอบไปด้วยอะเรย์ของลอจิกเกทต่างๆมากมาย ซึ่งในปัจจุบันความจุเกทภายในตัวชิพ FPGA ได้เพิ่มขึ้น จากระดับไม่กี่พันตัวจนถึงระดับล้านตัวซึ่งสามารถรองรับวงจรดิจิตอลที่มีความสลับซับซ้อนได้เป็นอย่างดี นอกจากนี้ในด้านการออกแบบพัฒนาและทดสอบก็ทำได้ง่ายซึ่งในปัจจุบันการออกแบบวงจรโดยใช้ FPGA กำลังเป็นที่นิยมและมีแนวโน้มที่จะนำมาใช้งานมากขึ้นเรื่อยๆ ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรมที่แตกต่างกันไป นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบคือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรม โดยการใช้หน่วยความจำ

1. การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

1.1 Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียวเท่านั้น ซึ่งหลังจากที่ โปรแกรมแล้วจุดเชื่อมต่อจะขาดจากกัน

1.2 Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรมแล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

2. การโปรแกรมโดยใช้หน่วยความจำ

2.1 EEPROM Based FPGA

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

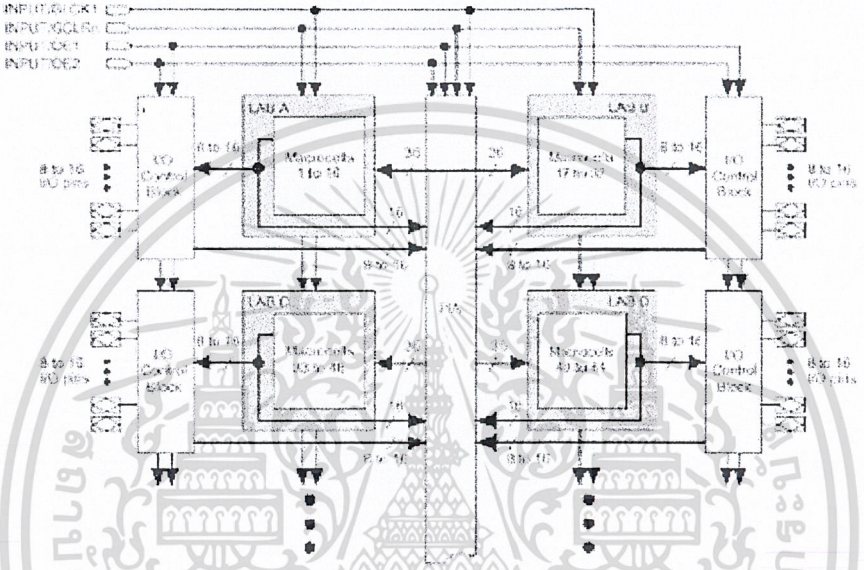
2.2 SRAM Based FPGA

FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ SRAM (Static RAM)ทำให้สามารถโปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมีความจุของเกทในระดับปานกลางถึงสูงมาก (ประมาณ 10,000 - 1,000,000 เกท) ซึ่งข้อดีของ SRAM Based FPGA คือใช้เวลาในการ โปรแกรมน้อย (ระดับ nSec) การโปรแกรมทำได้ง่ายเทียบได้กับการเขียน SRAM ทั่วไป และ เหมาะสำหรับการออกแบบวงจรที่มีความสลับซับซ้อน ส่วนข้อเสียคือไม่สามารถเก็บโปรแกรมใน ภาวะที่ไม่มีไฟเลี้ยงได้ ดังนั้น

FPGA ชนิดนี้จึงมักใช้ควบคู่กับ ROM เพื่อเก็บโปรแกรมและทำการโหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

- **โครงสร้างภายในของ FPGA**

ลักษณะโครงสร้างภายในของ FPGA จะเป็นอะเรย์ของบล็อกลอจิกที่สามารถทำการโปรแกรมได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงโครงสร้างภายในของ FPGA ตระกูล MAX7000S

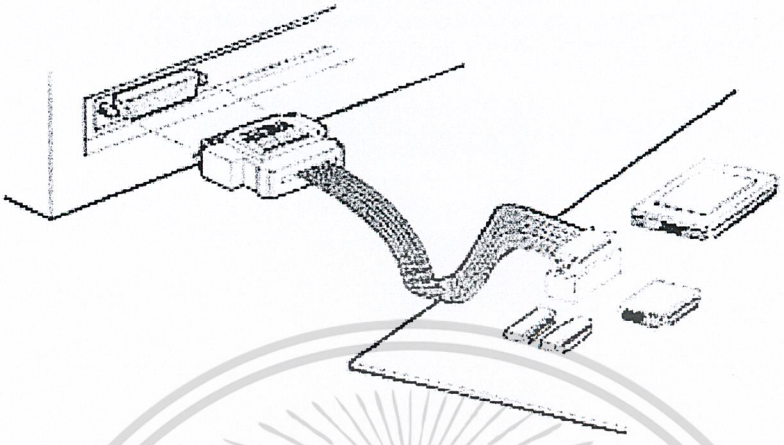
- **ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว**

1.ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออกแบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2.มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language) เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็วและไม่จำเป็นต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตราฐานเดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3.การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมา

โปรแกรมข้างนอก ดังรูปที่ 2.2 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด



รูปที่ 2.2 การโปรแกรมลงในชิพ

- **การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์**

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ ในกรณีของการออก แบบวงจรด้วย ASIC ชนิด Full Custom ผู้ออกแบบจะต้องเขียนวงจรด้วย Schematic จากนั้นจะนำวงจรที่ ออกแบบไว้ไปทำการจำลองการทำงาน (Simulate) ซึ่งหากผลออกมาเป็นที่พอใจก็จะต้อง Layout เป็นชั้นสาร และในการออกแบบ ASIC ชนิดนี้ผู้ออกแบบจำเป็นจะต้องทราบถึงเทคโนโลยีที่ใช้ในการสร้างด้วย หลังจากได้ Layout ที่สมบูรณ์แล้วจึงจะส่งไปเข้ากระบวนการสร้างไอซีหรือ Fabrication เพื่อสร้างเป็นชิพไอซีออกมา แต่ในการออกแบบวงจรด้วย FPGA โดยการใช้ Schematic หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญการออก แบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจรใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verilog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

1. การสังเคราะห์วงจร(Logic Synthesis)

ในขั้นตอนนี้จะใช้ซอฟต์แวร์ในการสังเคราะห์วงจร (Synthesis Tools) ทำการสังเคราะห์พฤติกรรม ของวงจรที่ได้จากการออกแบบด้วย Schematic หรือ VHDL ซึ่งต้องทำการตรวจสอบด้วยว่าซอฟต์แวร์ นั้นสนับสนุนเทคโนโลยี FPGA (FPGA Library) ที่ต้องการหรือไม่ ตัวอย่างเช่น FPGA ของบริษัท XILINX และบริษัท ALTERA จะมีซอฟต์แวร์หลายตัวที่สามารถใช้ได้ เช่น Max Plus II ในขั้น

ตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตามเทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครงสร้างที่มีอยู่ในอุปกรณ์ FPGA ดังนั้นในการ Optimize ซอฟต์แวร์สังเคราะห์วงจร จะต้องทำการ Optimize ให้ได้เป็นวงจรที่ประกอบ ด้วยกลุ่มของลอจิกที่เหมาะสมกับอุปกรณ์ FPGA นั้นๆจึงทำให้ผล ที่ได้มีประสิทธิภาพและในขั้นตอนการสังเคราะห์วงจรนี้ ผู้ออกแบบสามารถกำหนดข้อบังคับสำหรับโมเดล แต่ละตัวได้ เช่น ข้อบังคับในเรื่องเวลา (Timing Constraints) หรือข้อบังคับในเรื่องของพื้นที่ (Area) หรือกำหนดชนิดและตำแหน่งของ I/O ซึ่งข้อบังคับเหล่านี้จะถูกนำไปใช้ในขั้นตอน Optimize เพื่อให้วงจร ที่ได้เป็นไปตามที่กำหนด ส่วนสำคัญในการ Optimize คือการเทียบ (Mapping) โมเดลให้เข้ากับ เทคโนโลยีที่ใช้เพื่อให้ได้วงจรที่เหมาะสมกับโครงสร้างและสถาปัตยกรรมภายในอุปกรณ์ FPGA เมื่อทำ การสังเคราะห์วงจรเสร็จแล้ว ซอฟต์แวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้น เป็นอย่างไร เช่นมีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้น ตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตาม ที่กำหนด

2. การแบ่งวงจร (Partitioning)

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

3. การวางอุปกรณ์ (Placement)

ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควร จะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะ ได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี ความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณได้ไม่หมด การวางอุปกรณ์ที่ดีควรวางส่วนต่างๆให้อยู่ใกล้กันโดยเฉพาะส่วน ที่มีการเชื่อมต่อสัญญาณด้วยกันนอกจากนั้นการกำหนดตำแหน่งขา I/O (I/O pin) ตามตำแหน่งขา I/O ของ FPGA บนแผ่น PCB ก็จะมีผลโดยตรงเลยคือซอฟต์แวร์จะวาง I/O ลงในตำแหน่งที่ผู้ออกแบบกำหนด ซึ่ง บางครั้งตำแหน่งที่กำหนดไปไม่เหมาะสม ดังนั้นการกำหนดขา I/O ควรกำหนดตำแหน่งให้เหมาะสม หรือ ไม่ก็ให้ซอฟต์แวร์จัดการเอง

4. การเชื่อมต่อสัญญาณ (Routing)

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

5. ความหน่วงด้านเวลา (Delay)

ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วง ที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์ FPGA เองและ ความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay) เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์ และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลาเพื่อให้ ซอฟต์แวร์ได้ทำงานอย่างมีประสิทธิภาพเพิ่มขึ้น และเพื่อให้ได้ผลลัพธ์ที่ดีขึ้นค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความหน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้น เป็นไปตามข้อกำหนด หรือไม่

6. การจำลองการทำงานของวงจร (Simulation)

ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการ ทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น Model Sim ของบริษัท Model Technology หรือ Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มีการทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดล เกิดขึ้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้ เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออก แบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจ สอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้ว เพื่อตรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตาม ข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้ จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจ

สอบฟังก์ชันการทำงานแล้วยังต้อง ตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่าความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวม สามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

7. การโปรแกรมอุปกรณ์ FPGA (Configuration)

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

- **เครื่องมือสำหรับการออกแบบ FPGA**

จะเห็นได้ว่าการออกแบบเพื่อทำ FPGA นั้นทำได้สะดวกกว่า ASIC มากเพราะใช้เวลาน้อยกว่ามากด้วย ส่วน สำคัญที่ใช้ในการทำ FPGA คือ ซอฟต์แวร์ที่ใช้ตั้งแต่เขียนโค้ดอธิบายฮาร์ดแวร์จนกระทั่งดาวน์โหลดลงใน อุปกรณ์ FPGA ซึ่งซอฟต์แวร์ที่ใช้ต้องเป็น ซอฟต์แวร์ที่ทำงานต่อเนื่องกันได้ สำหรับซอฟต์แวร์ที่ใช้ทำการ จำลองการทำงานของวงจรนั้น ต้องสามารถใช้งานต่อเนื่องกับซอฟต์แวร์ที่ใช้ทั้งระบบ เพราะโมเดลที่ได้จาก การทำขั้นตอนต่างๆ ด้วยซอฟต์แวร์ ต่างๆ ต้องเอามาจำลองการทำงานได้ และในการจำลองการทำงานของ วงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตลอดทั้งระบบ เพื่อจะได้เปรียบเทียบผลได้ง่าย ในอดีตซอฟต์แวร์ส่วนใหญ่ จะใช้งานอยู่บนคอมพิวเตอร์สมรรถนะสูงอย่างเวิร์คสเตชั่น (Workstation) ในปัจจุบันมีการพัฒนาซอฟต์แวร์ ที่ใช้บนพีซี (PC) มากขึ้นซึ่งสามารถลดค่าใช้จ่ายในด้านอุปกรณ์คอมพิวเตอร์

2.1.2 เทคโนโลยีการออกแบบวงจรลอจิก (VHDL : Logic Design Technology)

เทคโนโลยีของการออกแบบวงจรลอจิกได้มีการพัฒนาไปอยู่เสมอ จนได้มีการพัฒนาภาษาโปรแกรมที่ใช้ในการออกแบบวงจรลอจิกขึ้นมา VHDL เป็นหนึ่งในนั้นที่ได้มีการพัฒนาขึ้นมา VHDL ย่อมาจาก VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit) มันก็คือภาษาที่ใช้ในการออกแบบวงจรลอจิกเพื่อผลิตเป็นชิพไอซีขึ้นมา VHDL เป็นภาษาระดับสูงที่ใช้ใน

การออกแบบระบบและวงจรลอจิกเป็นภาษาที่สามารถออกแบบวงจรได้ในระดับต่างๆ จะประกอบไปด้วยโครงสร้างต่างๆที่ถูกออกแบบขึ้นมา ในระดับที่สูงขึ้นตัวภาษานั้นสามารถที่จะออกแบบระบบโดยไม่คำนึงกระบวนการหรือวิธีการของวงจร เพราะวงจรจะถูกสร้างในรูปแบบของฟังก์ชั่น เราจะพิจารณาเพียงจุดมุ่งหมายของการออกแบบวงจรเท่านั้น เป็นภาษามาตรฐานที่ IEEE ได้ให้การยอมรับ ข้อดีคือเป็นภาษาที่สามารถปรับเปลี่ยนโครงสร้างให้เข้ากับระบบฮาร์ดแวร์ต่างๆได้เป็นอย่างดี และอ่านเข้าใจได้ง่าย

- **องค์ประกอบที่สำคัญของ VHDL**

VHDL มีองค์ประกอบที่สำคัญอยู่ 2 ส่วนคือ Entity และ Architecture

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

2 Architecture ใช้ในการอธิบายโครงสร้างภายในของ Entity

รูปที่ 2.3. แสดง D Flip-Flop

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

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

- **รูปแบบของภาษา VHDL**

- รูปแบบการเขียน Architecture ของ VHDL

VHDL มีรูปแบบการเขียนอยู่ 3 รูปแบบ

1 Structural Model การอธิบายวงจรโดยใช้โมดูลต่างๆมาเชื่อมต่อกันให้เห็นโครงสร้างภายใน

2 Behavioral Model การอธิบายการทำงานของวงจรในระดับลอจิกเกท

3 Sequential Model การอธิบายขั้นตอนการทำงานโดยใช้ Sequential Statement ทำงานทีละขั้นตอน

ต่อไปนี้จะเป็นตัวอย่างการเขียนโปรแกรมเพื่ออธิบายวงจร RS F/F โดยใช้ Architecture ที่แตกต่างกัน

Structural Model

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

Behavioral Model

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

Sequential Model

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

สำหรับรูปแบบการเขียนที่เราจะเลือกใช้นั้น ขึ้นอยู่กับความเหมาะสมของฮาร์ดแวร์ เราสามารถที่จะใช้รูป แบบต่างๆมาร่วมกันได้อยู่ใน Architecture ตัวเดียวกันก็ขึ้นอยู่กับความเหมาะสมเช่นกัน จะเห็นว่าในแต่ละแบบก็มีจุดเด่นของตัวมันเอง แต่ทั้ง 3 แบบก็ให้ผลลัพธ์เหมือนกัน

Sequential มีการอธิบาย Architecture เป็นลำดับขั้นตอนที่ชัดเจน เข้าใจง่าย โดยปกติแล้วโปรแกรมที่อยู่ภายใน Architecture นั้นจะมีการ process พร้อมๆกันทุกบรรทัด (เหมือนวงจรจริงที่มีการเชื่อมต่อสัญญาณกันทั้งวงจร เมื่อมีสัญญาณ input ตัวใดเปลี่ยน output ก็จะมีการเปลี่ยนตามทันที จะมี

delay ก็เพียงเล็กน้อย) แต่แบบ Sequential สามารถจะมองเป็นขั้นตอนได้โดยใส่ Process Statement คร่อมเข้าไปในโปรแกรม ส่วน Structural ก็มีการดึง Component NAND มาจะมองเห็นการเชื่อมต่อที่ชัดเจน ส่วน Behavioral นั้นมีการอธิบายคุณสมบัติระดับลอจิกเกท

- Behavioral Modeling

ในหัวข้อที่แล้วได้กล่าวถึง Structural Model ไปบ้างแล้วในหัวข้อนี้เป็น Behavioral Model ซึ่งจะกล่าวถึง Statement และฟังก์ชั่นต่างๆที่นิยมใช้กับ Behavioral Model

1. การใช้ Selected signal assignment และ Conditional signal assignment statement

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

```
with sel select                    sel<=   0 when a = '0' and b = '0' else
  q<=i0 after 10 ns when 0,                1 when a = '1' and b = '0' else
       i1 after 10 ns when 1,              2 when a = '0' and b = '1' else
       i2 after 10 ns when 2,              3 when a = '1' and b = '1' else
       i3 after 10 ns when 3,              4;
      'x' after 10 ns when others;
```

โปรแกรมทางซ้ายคือ Selected signal assignment โดย q จะเลือกส่งค่า i0, i1, i2, i3 ออกไปขึ้นอยู่กับค่าของ sel (sel เป็น integer) ส่วนโปรแกรมทางขวาก็คือ Condition signal assignment โดยตัวแปร sel จะมีค่าเป็น 0, 1, 2, 3 หรือ 4 ขึ้นอยู่กับเงื่อนไขว่าเงื่อนไขใดถูกต้อง

2. การใช้ delay

การใช้ delay ใน VHDL นั้นเป็นการทำงานเลียนแบบการทำงานจริงๆของวงจรที่จะมี delay ในการทำงานอยู่เช่นกัน

```
a <= b;  ( ส่งค่า b ไป a ทันทีโดยไม่มี delay time )
a <= b after 10 ns; ( ส่งค่า b ไป a หลังจากที่ผ่านไปแล้ว 10 ns )
```

VHDL จะมี delay อยู่ 2 แบบ

1. Inertial delay ก่อนจะส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณทางขวาจะต้องคงค่านั้นได้นานเท่ากับค่า delay ดังนั้น delay แบบนี้สัญญาณของตัวส่งกับตัวรับไม่จำเป็นจะต้องเหมือนกัน ดังตัวอย่างการใช้งาน

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

2 Transport delay เป็นการส่งค่าสัญญาณทางขวาไปยังทางซ้าย โดยสัญญาณของทั้งตัวส่งและตัวรับจะเหมือนกัน แตกต่างกันก็ตรงที่สัญญาณตัวรับจะช้ากว่าเป็นเวลาเท่ากับที่ delay ไว้ ดังตัวอย่างการใช้งาน

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

- การใช้งาน Generics

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

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

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

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

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

เมื่อเราแทรก entity ตัวนั้นลงไปแล้วเราก็สามารถเรียกใช้ได้เสมือนเป็นฟังก์ชั่นตัวหนึ่ง และมีการส่งค่าให้ได้ด้วยดังนี้

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

(เป็นการเรียกใช้ในรูป-แบบ Structure)

- การใช้งาน Block Statements

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

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

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

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

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

- **Sequential Processing**

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

- Process Statement

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

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

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

- Sequential Statements

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

รูปแบบ IF statement

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

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

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

รูปแบบ Case Statements

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

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

รูปแบบของ LOOP Statements

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

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

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

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

- Wait Statements

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

- Wait on signal

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

- Wait until condition

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

- Wait for time_expression

เป็นการหน่วงตามเวลาที่เรากำหนดไว้ เช่น Wait for 10 ns; คือเป็นการหน่วงไว้เป็นเวลา 10 ns;

- Multiple Wait condition

เป็นการใช้การหน่วงทั้ง 3 แบบมาร่วมกันเช่นดังตัวอย่าง

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

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

- **ชนิดของข้อมูล (Data types)**

- Object Types

VHDL ก็มีคุณสมบัติต่างๆเช่นเดียวกับภาษาระดับสูง มีการแบ่งชนิดของตัวแปรให้ใช้ในภาษาอยู่ด้วย 3 ชนิดด้วยกันคือ

1 Signal ใช้เป็นสัญญาณเพื่อแสดงการเชื่อมต่อกันระหว่าง Component ต่างๆ

2 Variable ใช้เป็น Temp เก็บข้อมูลชั่วคราวภายใน entity โดยใช้ในการ process

3 Constant ชื่อที่มีการกำหนดค่าคงที่ให้ ไว้ใช้ใน program

signal มีรูปแบบการใช้งานดังนี้

SIGNAL signal_name : signal_type [:= initial_value];

การใช้งาน Signal นอกจากจะเป็นการประกาศชนิดของสัญญาณแล้ว ยังสามารถจะกำหนดค่าเริ่มต้นให้กับสัญญาณนั้นได้ด้วยดังตัวอย่างเป็นการประกาศสัญญาณ vcc ให้มีชนิดเป็น std_logic และมีค่าเริ่มต้นเป็น 1

SIGNAL vcc : std_logic := '1';

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

Variables มีรูปแบบการใช้งานดังนี้

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

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

VARIABLE delay : time := 2 ns;

Constants มีรูปแบบการใช้งานดังนี้

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

การใช้งาน Constant คือจะต้องมีการกำหนดค่าเริ่มต้นให้กับตัวแปรเพื่อเป็นค่าคงที่นำไปใช้ในโปรแกรมได้ ดังตัวอย่างเป็นการกำหนดตัวแปร pi มีชนิดเป็น real มีค่าคงที่เท่ากับ 3.1414

CONSTANT pi : real := 3.1414;

- **ชนิดของข้อมูล (Data types)**

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

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

1. Scalar Types

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

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

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

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

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

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

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

2. Composite Types

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

Array รูปแบบของ array มิติเดียว

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

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

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

ตัวอย่างการประกาศตัวแปรให้ x มีชนิดเป็น data_bus

```
VARIABLE x : data_bus;
```

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

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

Multidimensional Arrays เป็นการใช้ array หลายมิติ Array 2 มิตินิยมใช้ในการออกแบบ ROM และ RAM

ตัวอย่างการประกาศ Array หลายมิติ

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

ตัวอย่างการเข้าถึง Array หลายมิติ X(1,1) := ‘1’;

Record

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

ตัวอย่างการประกาศ

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

เราสามารถอ้างถึงข้อมูลได้โดยใส่จุดไว้ข้างหลังตัวแปร และตามด้วยชื่อฟิลด์ใน record เช่น

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

3. Access Types

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

ตัวอย่างการใช้

```
TYPE  fifo  IS ARRAY (0 TO 3) OF std_logic;
 TYPE fifo_access IS ACCESS fifo;
 VARIABLE  fifo_ptr : fifo_access := NULL;
 fifo_ptr := new fifo;
 fifo_ptr.ALL := (‘0’, ‘0’, ‘1’, ‘1’);
```

4. File Types

เป็นข้อมูลที่ประกาศขึ้นเพื่อใช้ในการติดต่อกับไฟล์

รูปแบบการประกาศ

TYPE file_ name IS FILE OF file_ type;

file มี Function ให้ใช้งานอยู่ 3 Function

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

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

ENDFILE (file) (file คือชื่อ fileที่ปิด)

5.Subtypes

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

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

- **เทคนิคการใช้งาน VHDL**

- การใช้งาน signal ชนิด std_logic

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

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

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

นอกจากการนำ others มาช่วยแล้ว เรายังสามารถใช้ don't care เข้ามาช่วยได้อีกด้วยในกรณีที่เราสนใจข้อมูลเฉพาะบางบิท เช่น ถ้าเรามีข้อมูลอยู่ 6 บิท แต่ต้องการตรวจสอบเฉพาะ 2 บิทหลังว่าเป็น 1 ทั้งคู่หรือไม่ ก็จะต้องนำข้อมูลนั้นไปเปรียบเทียบกับ "----11" เครื่องหมายลบแทน don't care การใช้งานอย่างนี้จะมีประโยชน์มากในการเปรียบเทียบเพราะเราไม่จำเป็นต้องแยกออกมาเปรียบเทียบกันทีละบิท

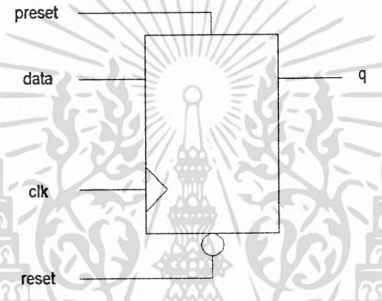
- การใช้งาน VHDL กับ Flip-Flops

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

(clk'event and clk = '1') แทนขอบขาขึ้นของ clk

(clk'event and clk = '0') แทนขอบขาลงของ clk

ยกตัวอย่างการออกแบบ D Flip-Flop ที่มี preset และ reset



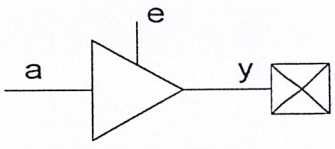
รูปที่ 2.4. แสดง D Flip-Flop มี preset และ reset

```
entity  dff_async  is     ( entity block )
  port (data, clk, reset, preset : in std_logic;
     q : out std_logic);
  end dff_async;
architecture behav of dff_async is
  begin
  process (clk, reset, preset ) begin
     if (reset ='0') begin                ( reset ค่า q เมื่อ reset เป็น '0' )
          q <= '0';
     elsif (preset ='1') then             ( set ค่า q เมื่อ preset เป็น '1' )
          q <= '1';
     elsif(clk'event and clk='1') then  (ส่ง data เมื่อมี clk เข้ามา )
          q <= data;
     end if;
```

```
end process;
end behav;
```

- **การออกแบบ buffer แบบต่างๆ**

1. Tri-State



รูปที่ 2.5 แสดง Tri-State

การออกแบบ buffer ที่มี 3 สถานะโดยมี enable เป็นตัวควบคุมสามารถเขียนเป็น entity ได้ดังนี้

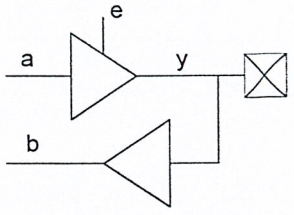
```
entity  tristate  is
port  (e, a : in std_logic;
       y : out std_logic);
end tristate;
```

Architecture นั้นสามารถเขียนได้หลายแบบดังตัวอย่างทั้ง 2 แบบต่อไปนี้

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

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

2. Bi-Directional Buffer



รูปที่ 2.6 แสดง Bi-Directional Buffer

เป็น buffer 2 ทิศทางสามารถเขียนเป็นโปรแกรมส่วน Architecture ได้ดังนี้

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

- **ข้อดีของ VHDL**

1. รูปแบบของภาษาที่เข้าใจได้ง่าย
2. มีโครงสร้างของภาษาที่สามารถปรับเปลี่ยนให้เข้ากับ hardware ได้ง่าย
3. มี statement ให้ใช้งานอยู่หลายตัว
4. สามารถออกแบบในรูปแบบ top-down design ได้
5. มี library ต่างๆให้เลือกใช้งานได้ทำให้ออกแบบได้ง่ายขึ้น เป็นประโยชน์ต่อ programmer

2.2 การใช้งาน MCS –51

2.2.1 จัดขาของไมโครคอนโทรลเลอร์ 8051

V_{cc} : สำหรับแหล่งจ่ายไฟฟ้า (+5v.)

V_{ss} : สำหรับต่อกราวด์

P0 : เป็นขาพอร์ต 0 ของ 8051 ที่มีขนาด 8 บิทชนิดสองทิศทาง ซึ่งแต่ละบิทสามารถกำหนดให้เป็นได้ทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไปหากต้องการให้เป็นอินพุทสามารถทำได้โดยการเขียนข้อมูล “1” ไปยังบิทนั้น โดยแต่ละบิทเมื่อเป็นเอาท์พุทจะสามารถต่อพ่วงกับอุปกรณ์ TTL แบบ LS ได้ 8 ตัว และยังเป็นขาให้สัญญาณ Multiplex ระหว่างสัญญาณข้อมูลกับสัญญาณ Address 8 บิทแรกในกรณีที่ใช้หน่วยความจำภายนอก

P1 : เป็นขาพอร์ต 1 ของ 8051 ขนาด 8 บิท ชนิดสองทิศทางแบบ Quasi bi-directional ซึ่งแต่ละบิทสามารถกำหนดให้เป็นได้ทั้งอินพุทและเอาท์พุทสำหรับใช้งานทั่วไปหากต้องการให้เป็นอินพุท สามารถทำได้โดยการเขียนข้อมูล “1” ไปยังบิทนั้น และสามารถต่อพ่วงกับอุปกรณ์ LS TTL ได้ 4 ตัว

P2 : เป็นขาพอร์ต 2 ของ 8051 ขนาด 8 บิท ชนิดสองทิศทางแบบ Quasi bi-directional เช่นเดียวกับพอร์ต 1 นอกจากนี้พอร์ต2นี้ยังทำหน้าที่ให้สัญญาณ Address 8 บิทบน ในกรณีที่ใช้หน่วยความจำภายนอก ในกรณีอ้าง Address หน่วยความจำขนาด 16 บิท ดังนั้นขณะที่ใช้หน่วยความจำภายนอกจะต้องไม่มีการเขียนข้อมูลใด ๆ ไปที่พอร์ต 2 จะทำให้เกิดความผิดพลาดการทำงานได้

P3 : เป็นขาพอร์ต 3 ของ 8051 ขนาด 8 บิท ชนิดสองทิศทางแบบ Quasi bidirectional เช่นเดียวกับขาพอร์ต 1 และพอร์ต 2 แต่พอร์ต 3 นี้จะมีหน้าที่พิเศษดังตารางที่ 2.1

ขาพอร์ต	หน้าที่พิเศษ
P3.0	RxD (สำหรับรับข้อมูลแบบอนุกรม)
P3.1	TxD (สำหรับส่งข้อมูลแบบอนุกรม)
P3.2	INT0 (ขาอินเทอร์รัพท์ภายนอก 0)
P3.3	INT1 (ขาอินเทอร์รัพท์ภายนอก 1)
P3.4	T0 (ขาอินพุทของ Timer 0)
P3.5	T1 (ขาอินพุทของ Timer 1)
P3.6	WR (สำหรับสัญญาณเขียนหน่วยความจำข้อมูลภายนอก)
P3.7	RD (สำหรับสัญญาณอ่านหน่วยความจำข้อมูลภายนอก)

ตารางที่ 2.1 หน้าที่พิเศษของขาต่าง ๆ ของ PORT 3

ดังนั้น เมื่อมีการใช้สัญญาณดังกล่าว จึงไม่ควรเขียนข้อมูลไปที่พอร์ต 3 จะทำให้การทำงานของ 8051 ผิดพลาดได้

RST : เป็นขาสำหรับรีเซ็ทการทำงานของ 8051 โดยการให้ลอจิกหนึ่งเป็นเวลาอย่างน้อย 2 ช่วง Machine Cycle

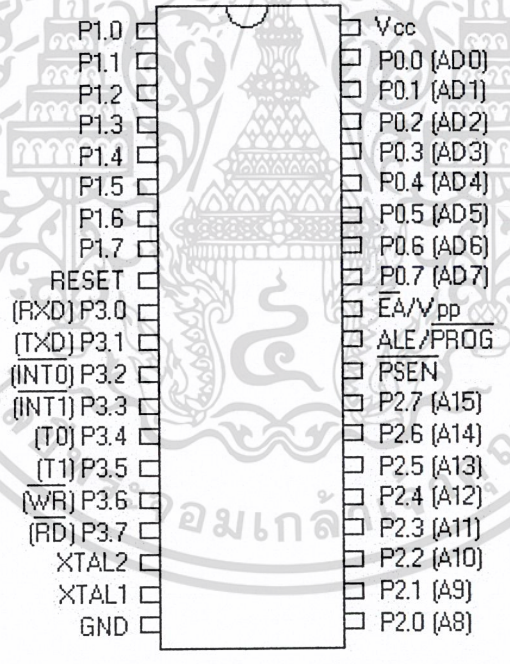
ALE : เป็นขาที่ใช้ในการควบคุมการแลตซ์ของขา พอร์ต 0 เมื่อมีการใช้งานหน่วยความจำภายนอก

PSEN : เป็นขาสัญญาณเพื่อร้องขอติดต่อหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ต้องการอ่านหน่วยความจำโปรแกรมภายนอก

EA : เป็นขาใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมภายนอกหรือภายในไมโครคอนโทรลเลอร์ โดยที่ให้ลอจิก 0 จะอ่านหน่วยความจำโปรแกรมภายนอก และลอจิก 1 จะอ่านหน่วยความจำโปรแกรมภายใน

XTAL1 : ขาเข้าของวงจรกำเนิดความถี่อ้างอิงภายในของ 8051

XTAL2 : ขาออกของวงจรกำเนิดความถี่อ้างอิงภายในของ 8051

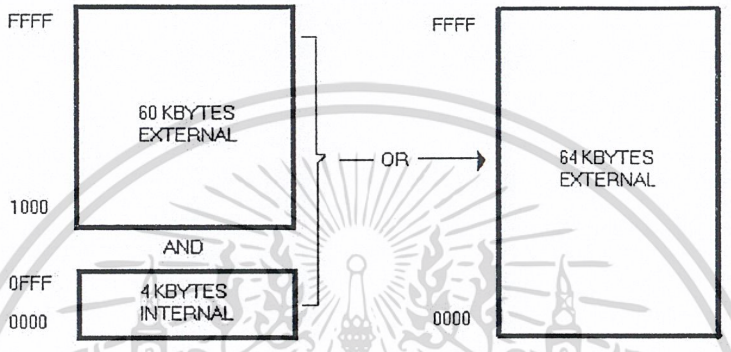


รูปที่ 2.7 การจัดขาของ 8051

2.2.2 โครงสร้างหน่วยความจำของ 8051

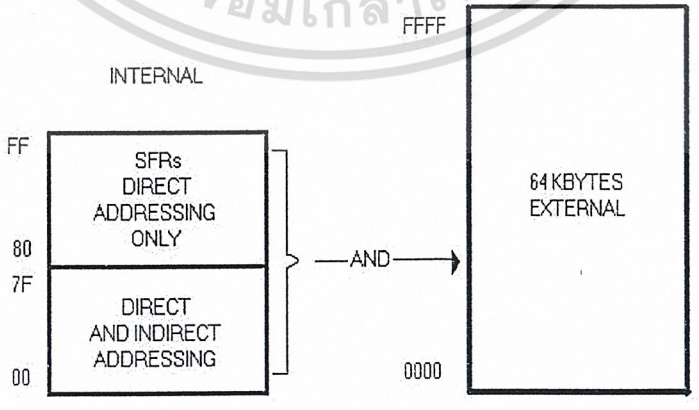
ดังที่กล่าวมาแล้ว 8051 จะแบ่งหน่วยความจำออกเป็นสองส่วน ได้แก่ หน่วยความจำสำหรับโปรแกรมและหน่วยความจำสำหรับเก็บข้อมูล โดยมีขนาดของแต่ละส่วนเท่ากับ 64 กิโลไบต์ ในส่วนของหน่วยความจำโปรแกรมจะเป็นส่วนหน่วยความจำสำหรับอ่านอย่างเดียว โดยที่ 8051 จะใช้สัญญาณ PSEN ในการอ่านเท่านั้น แต่หน่วยความจำข้อมูลของ 8051 จะสามารถอ่านและเขียนได้โดยใช้สัญญาณ

RD และ WR ตามลำดับ แต่อย่างไรก็ตาม ผู้ใช้สามารถรวมหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลเข้าด้วยกันได้ โดยนำสัญญาณ RD และ PSEN มาต่อเข้าวงจรแอนเกต สำหรับสร้างสัญญาณในการอ่านหน่วยความจำ นอกจากนี้หน่วยความจำโปรแกรมยังแบ่งออกเป็นภายนอกและภายในของ 8051 ดังแสดงในรูปที่ 2.8 และ รูปที่ 2.9 โดยรูปที่ 2.8 แสดงหน่วยความจำโปรแกรมในกรณีที่เลือกใช้หน่วยความจำภายนอกและภายใน ในด้านซ้ายมือเป็นส่วนหนึ่งของหน่วยความจำโปรแกรมภายในที่มีขนาด 4 กิโลไบต์ของ 8051 ส่วนที่เหลือจะเป็นหน่วยความจำภายนอก ส่วนด้านขวามือแสดงหน่วยความจำโปรแกรมเมื่อเลือกให้ติดต่อหน่วยความจำภายนอกทั้งหมด



รูปที่ 2.8 แสดงหน่วยความจำโปรแกรมของ 8051

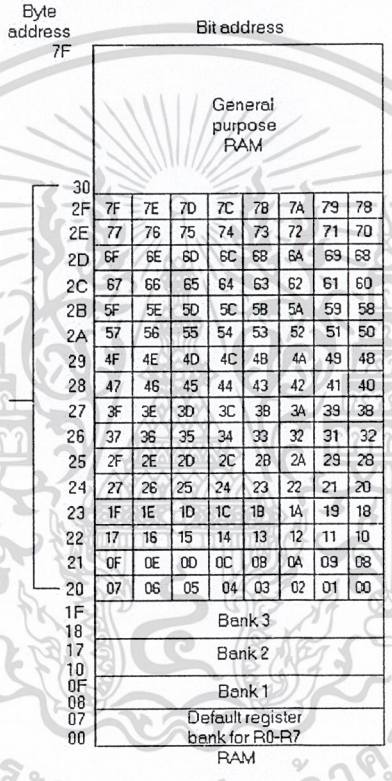
สำหรับหน่วยความจำข้อมูลของ 8051 สามารถแบ่งออกเป็นภายนอกและภายใน โดยหน่วยความจำภายนอกแสดงไว้ด้านขวามือของรูปที่ 2.9 ซึ่งมีขนาด 64 กิโลไบต์ ส่วนหน่วยความจำข้อมูลภายในแสดงไว้ด้านซ้ายของรูปที่ 2.9 โดยหน่วยความจำภายในของ 8051 แบ่งออกเป็นสองส่วน ได้แก่ ส่วนของหน่วยความจำข้อมูลที่สามารถอ้างอิงแบบ Direct และ Indirect ซึ่งมีขนาด 128 ไบต์ กับหน่วยความจำที่อ้างอิงได้เฉพาะแบบ Direct หรือในส่วนนี้จะเรียกอีกแบบหนึ่งว่า SFR (Special Function Register) โดยจะแบ่งกล่าวได้ดังนี้



รูปที่ 2.9 แสดงหน่วยความจำข้อมูลของ 8051

ในส่วนของหน่วยความจำข้อมูลภายในที่อ้างอิงแบบ direct และ Indirect นั้นจะสามารถแบ่งออกได้ 3 ส่วน ดังแสดงในรูปที่ 2.10 โดยมีรายละเอียดดังนี้

- ส่วนที่ 1 เรียกว่า Register Banks 0-3 ซึ่งอยู่ที่ตำแหน่งความจำข้อมูลภายใน ตั้งแต่ 00H ถึง 1FH จำนวน 32 ไบต์ โดยจะแบ่งออกเป็นชุด ชุดละ 8 ไบต์จำนวน 4 ชุด ซึ่งแต่ละชุดจะมีชื่อเรียกเป็น R0 ถึง R7 จะเป็น Register ที่ใช้งาน โดยเมื่อ 8051 ถูกรีเซ็ท Register Bank 0 จะถูกเลือกใช้
- ส่วนที่ 2 เรียกว่า Bit Addressable Area ซึ่งมีขนาด 16 ไบต์ที่ตำแหน่งหน่วยความจำข้อมูล 20H ถึง 2FH ในส่วนนี้สามารถที่จะอ้างอิงข้อมูลได้เป็นระดับบิทถึง 128 บิท โดยการอ้างอิงตำแหน่งโดยตรงในลักษณะบิท ตั้งแต่ตำแหน่ง 00H ถึง 7FH



รูปที่ 2.10 แสดงหน่วยความจำข้อมูลภายใน

- ส่วนที่ 3 เรียกว่า Scratch Pad Area จะอยู่ที่ตำแหน่งตั้งแต่ 30H ถึง 7FH ซึ่งเป็นบริเวณหน่วยความจำข้อมูลภายในเอนกประสงค์ที่ผู้ใช้สามารถใช้ได้โดยตรง นอกจากนี้ยังสามารถใช้หน่วยความจำข้อมูลบริเวณนี้สำหรับการเก็บข้อมูลแบบ Stack ได้ด้วย

ในส่วนของหน่วยความจำข้อมูลภายในที่ใช้อ้างอิงแบบ Direct เพียงอย่างเดียวหรือที่เรียกว่า SFR ซึ่งเป็นส่วนสำหรับเก็บหรือกำหนดการทำงานภายในของ 8051 ดังแสดงในรูปที่ 2.11

ในส่วนของบริเวณนี้จะมีขนาด 128 ไบต์แต่ในการใช้งานนั้นใช้ได้เฉพาะตำแหน่งซึ่งแสดงไว้ในรูปที่ 2.11 เท่านั้น หากผู้ใช้อ้างตำแหน่งนอกเหนือจากนี้นั้นจะได้ข้อมูลที่คาดเดาไม่ได้ โดยแต่ละตำแหน่งจะมีหน้าที่ดังนี้

ACC : เป็น Accumulator ซึ่งเป็นรีจีสเตอร์สำหรับการประมวลผลทางคณิตศาสตร์และลอจิก โดยผู้ใช้สามารถอ้างอิงได้ในรูปแบบของไบต์หรือระดับบิทได้

B : เป็นรีจีสเตอร์พิเศษสำหรับใช้กับคำสั่งในการคูณหรือหาร นอกจากนี้ยังใช้เป็น รีจีสเตอร์สำหรับเก็บพักข้อมูลได้

PSW : เป็นรีจีสเตอร์ Program Status Word หรือแฟลกจะแสดงสถานะการทำงานของ 8051 สำหรับการตรวจสอบซึ่งจะอธิบายรายละเอียดในภายหลัง

	8 Bytes								
F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW[1]								D7
C8	T2CON[1][2]	T2MOD[2]	RCAP2L[2]	RCAP2H[2]	TL2[2]	TH2[2]			CF
C0									C7
B8	IP[1]								BF
B0	P3								B7
A8	IE[1]								AF
A0	P2								A7
98	SCON[1]	SBUF							9F
90	P1								97
88	TCON[1]	TMOD[1]	TL0	TL1	TH1				8F
80	P0	SP	DPL	DPH				PCON[1]	87
	↑ Bit Addressable								

Notes : 1. SFRs converting mode or control bits
2. AT89C52 only

รูปที่ 2.11 แสดงรายละเอียดของ Special Function Register

SP : เป็นรีจีสเตอร์สำหรับชี้หน่วยความจำข้อมูลภายในสำหรับการเก็บแบบ Stack

DPTR : เป็นรีจีสเตอร์ขนาด 16 บิท โดยแบ่งเป็น 8 บิทบนและ 8 บิทล่าง ให้สำหรับชี้ตำแหน่งของหน่วยความจำข้อมูลภายนอกหรือสำหรับการอ่านตารางข้อมูลของหน่วยความจำโปรแกรม

P0 : เป็นรีจีสเตอร์สำหรับพอร์ต 0 ของ 8051

P1 : เป็นรีจีสเตอร์สำหรับพอร์ต 1 ของ 8051

P2 : เป็นรีจีสเตอร์สำหรับพอร์ต 2 ของ 8051

P3 : เป็นรีจีสเตอร์สำหรับพอร์ต 3 ของ 8051

IP : เป็นรีจีสเตอร์สำหรับกำหนดลำดับความสำคัญของการอินเทอร์รัพท์ของ 8051

IE : เป็นรีจีสเตอร์สำหรับกำหนดการรับหรือไม่รับการอินเทอร์รัพท์ของ 8051

TMOD : เป็นรีจีสเตอร์สำหรับควบคุมหน้าที่ของ Timer/Counter ของ 8051

TCON : เป็นรีจีสเตอร์สำหรับควบคุมการทำงานของ Timer/Counter ของ 8051

T2CON : เป็นรีจีสเตอร์สำหรับควบคุมการทำงานของ Timer/Counter 2 ของ 8052

TH0 : เป็นรีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 0 8บิทบน

TL0 : เป็นรีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 0 8บิทล่าง

TH1 : เป็นรีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 1 8บิทบน

TL1 : เป็นรีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 1 8บิทล่าง

TH2 : เป็นรีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 2 8บิทบนของ 8052

TL2 : เป็นรีจีสเตอร์สำหรับเก็บข้อมูลของ Timer/Counter 2 8บิทล่างของ 8052

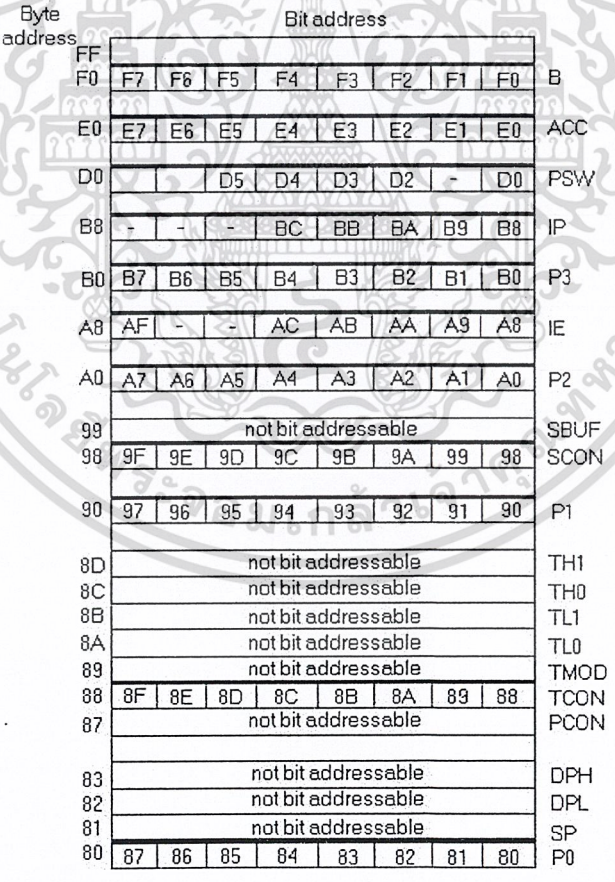
RCAP2H : เป็น Capture Register ของ Timer/Counter 2 8บิทบนของ 8052

SCON : เป็นรีจีสเตอร์สำหรับควบคุมการรับส่งข้อมูลแบบอนุกรมของ 8051

SBUF : เป็นรีจีสเตอร์สำหรับเก็บพักข้อมูลที่ได้จากการรับส่งข้อมูลแบบอนุกรมของ 8051

PCON : เป็นรีจีสเตอร์สำหรับควบคุมการทำงานของ 8051 ด้านเกี่ยวกับการใช้กำลังไฟฟ้า

ในส่วนของรีจีสเตอร์ SFR นี้สามารถที่จะอ้างอิงในระดับบิทได้โดยตำแหน่งการอ้างอิงระดับบิทแสดงไว้ในตารางต่อไปนี้



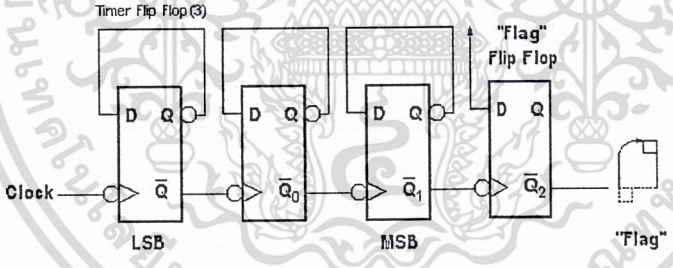
รูปที่ 2.12 แสดงตำแหน่งการอ้างอิงระดับบิทของรีจีสเตอร์ SFR

2.2.3 TIMER

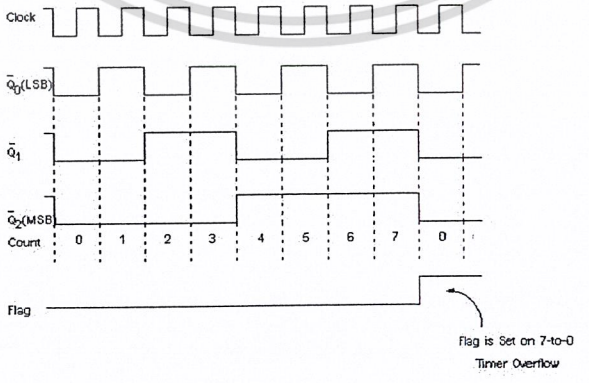
ตัว Timer อาจพิจารณาได้ง่าย ๆ ว่าเป็นตัวฟลิปฟลอปมาต่อเรียงกันโดยมี Clock เป็นอินพุท สำหรับเอาท์พุทที่ออกมาจากฟลิปฟลอปแต่ละตัวจะถูกหารด้วย 2 พิจารณาการต่อฟลิปฟลอปตามรูปที่ 2.13 ถ้าใส่ Clock เข้าไปในฟลิปฟลอปตัวแรก ความถี่ของ Clock ที่ออกจากเอาท์พุทตัวแรกจะถูกหารด้วย 2 และเอาท์พุทนี้จะต่อกับฟลิปฟลอปตัวที่สอง และสัญญาณที่ออกมาจะถูกหารด้วย 2 อีก ดังนั้น ถ้ามีฟลิปฟลอปต่ออยู่ n Stages จะหารสัญญาณนาฬิกาได้ 2^n ถ้าให้เอาท์พุท Stage สุดท้ายของ Timer เป็น Overflow Flip-Flop หรือ Flag และจะให้เอาท์พุทออกมาเมื่อการนับเป็น Overflow เช่น ถ้าเป็นตัวนับแบบ 16 บิท (มีฟลิปฟลอปต่ออยู่ 16 ตัว) วงจรจะนับตั้งแต่ 0000H ถึง FFFFH เมื่อฟลิปฟลอปเปลี่ยนจาก FFFFH เป็น 0000H จะให้บิท Overflow ออกมา

พิจารณารูป 2.13(ก) เป็น 3-bit Timer โดยฟลิปฟลอปแต่ละตัวจะนำขา Q มาต่อกับ D ซึ่งอาจเรียกว่าเป็นการใช้ฟลิปฟลอปแบบ Divide-by-two Mode โดยความถี่ของสัญญาณที่ได้จาก ฟลิปฟลอปแต่ละตัวจะมีค่าหารสองจากสัญญาณนาฬิกาที่เข้ามา เมื่อนับไปถึงค่า 111 (หรือ $Q_2 = 1, Q_1 = 1, Q_0 = 1$) และเปลี่ยนกลับมาเป็น 000 จะให้บิท Flag ออกมา ดังแสดงในรูปที่ 2.13(ข)

ใน MCS – 51 จะมีตัวจับเวลาอยู่ภายในชิพ ถ้าเป็นเบอร์ 8051 หรือ 8031 จะมี 2 ตัว คือ Timer 0 และ Timer 1 แต่ถ้าเป็นเบอร์ 8052 จะมีเพิ่มอีกหนึ่งตัวคือ Timer 2 รีจีสเตอร์ต่าง ๆ ที่เกี่ยวข้องกับการใช้ Timer แสดงได้ดังตารางที่ 2.2 ซึ่งจะเห็นว่ามีรีจีสเตอร์บางตัวสามารถเข้าถึงข้อมูลระดับบิทได้ด้วย นอกจากนี้ตัว Timer สามารถใช้เป็นตัวนับ (Counter) ได้อีกด้วย โดยการ โปรแกรมในรีจีสเตอร์ TMOD



(ก)



(ข)

รูปที่ 2.13 รีจีสเตอร์ที่ใช้เป็น Timer

รีจีสเตอร์	หน้าที่	ตำแหน่ง	สามารถอ้างอิงตำแหน่งบิท
TCON	Control	88H	Yes
TMOD	Mode	89H	No
TL0	Timer 0 Low-byte	8AH	No
TL1	Timer 1 Low-byte	8BH	No
TH0	Timer 0 High-byte	8CH	No
TH1	Timer 1 High-byte	8DH	No
T2CON*	Timer 2 Control	C8H	Yes
RCAP2L*	Timer 2 Low-byte Capture	CAH	No
RCAP2H*	Timer 2 High-byte Capture	CBH	No
TL2*	Timer 2 Low-byte	CCH	No
TH2*	Timer 2 High-byte	CDH	No

* มีในเบอร์ 8032 / 8052

ตารางที่ 2.2 รีจีสเตอร์ที่ใช้เป็น Timer

- **Timer Mode Register (TMOD)**

ตัวรีจีสเตอร์ TMOD เป็นรีจีสเตอร์ควบคุม Timer จะแบ่งออกเป็น 2 กลุ่ม กลุ่มละ 4 บิท โดย 4 บิทบนจะเป็นการควบคุม Timer 1 ส่วน 4 บิทล่างจะเป็นการควบคุม Timer 0 ความหมายของแต่ละบิทดูในตารางที่ 2.3 ซึ่งตัวรีจีสเตอร์นี้เป็นตัวเลือกการทำงานว่าจะให้ตัว Time/Counter ทำงานในโหมดใดและเป็น Timer หรือ Counter รีจีสเตอร์ TCON ไม่สามารถจะโปรแกรมเข้าไปในระดับบิทได้ (Not Bit-Addressable) ซึ่งการใช้งานมักจะโปรแกรมเข้าไปครั้งเดียวในตำแหน่งเริ่มต้นของโปรแกรม

บิท	ชื่อ	Timer	ความหมาย
7	GATE	1	Gate bit ถ้าบิทนี้เซ็ทวงจรจะทำงาน เมื่อ INT1 เป็น High
A	C/T	1	เป็นบิทเลือก Counter / Timer 1 = ใช้เป็น Counter 0 = ใช้เป็น Timer
5	M1	1	Mode bit 1
4	M0	1	Mode bit 0
3	GATE	0	บิท Gate ของ Timer 0
2	C/T	0	บิทเลือก Counter / Timer ของ Timer 0

บิท	ชื่อ	Timer	ความหมาย
1	M1	0	Timer 0 M1 bit
0	M0	0	Timer 0 M0 bit

ตารางที่ 2.3 รีจีสเตอร์ TMOD (Timer Mode)

M1	M0	Mode	ความหมาย
0	0	0	ใช้เป็น Timer แบบ 13-bit (8048 Mode)
0	1	1	ใช้เป็น Timer แบบ 16-bit
1	0	2	ใช้เป็น Timer แบบ 8-bit Auto-reload Mode
1	1	3	Split Timer Mode : แยก Timer 0 ออกเป็น Timer 8 บิทสองตัวคือ TL0 และ TH0 โดยไม่ใช้ Timer 1

ตารางที่ 2.4 การใช้ Timerโหมดต่าง ๆ

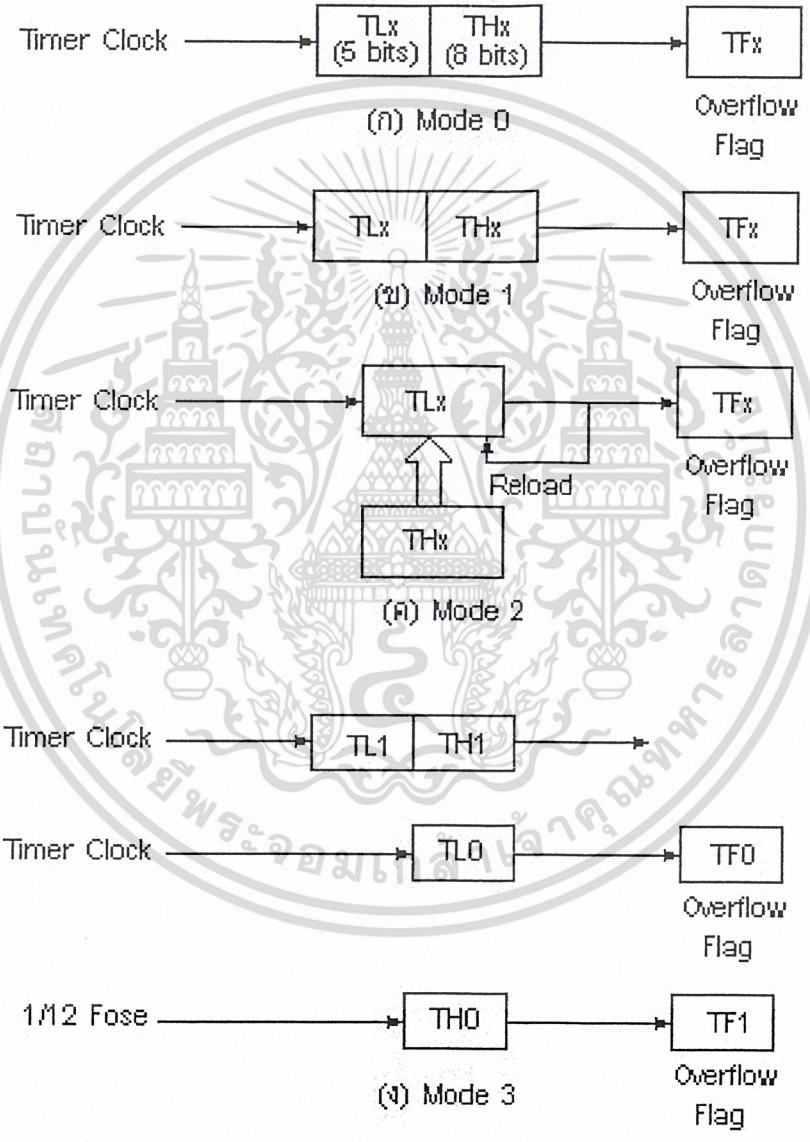
- **Timer Control Register (TCON)**

รีจีสเตอร์ TCON เป็นรีจีสเตอร์ที่บอกสถานะและควบคุมบิท Timer 0 และ Timer 1 ซึ่งดูได้จากตารางที่ 2.5 รีจีสเตอร์นี้สามารถเข้าถึงข้อมูลระดับบิทได้

บิท	ชื่อ	ตำแหน่งบิท	ความหมาย
TCON.7	TF1	8FH	บิทแฟลกแสดงการโอเวอร์โฟลว์ของ Timer 1 จะ Set โดย Hardware และ Clear โดย Software
TCON.6	TR1	8EH	บิทควบคุมการปิด-เปิด Timer 1 Set และ Clear โดย Software
TCON.5	TF0	8DH	แฟลกแสดงการโอเวอร์โฟลว์ของ Timer 0
TCON.4	TR0	8CH	บิทควบคุมการปิด-เปิด Timer 0
TCON.3	IE1	8BH	บิทแฟลกแสดงการอินเทอร์รัพท์จาก INT1 จะ Set โดย Hardware และสามารถ Clear ได้ด้วย Software
TCON.2	IT1	8AH	บิทเลือกชนิดของสัญญาณอินเทอร์รัพท์จากอินเทอร์รัพท์ภายนอก INT1 สามารถ Set และ Clear ได้ด้วย Software
TCON.1	IE0	89H	บิทแฟลกแสดงการอินเทอร์รัพท์จาก INT0

บิท	ชื่อ	ตำแหน่งบิท	ความหมาย
TCON.0	IT0	88H	บิทเลือกชนิดของสัญญาณอินเทอร์รัพท์จากอินเทอร์รัพท์ภายนอก INT0

ตารางที่ 2.5 แสดงความหมายแต่ละบิทของรีจีสเตอร์ TCON (Timer Control)



รูปที่ 2.14 การทำงานของ Timer ในโหมดต่าง

- **Timer Mode And Overflow Flag**

เมื่อใช้ Timer 0 และ Timer 1 จะต้องใช้รีจีสเตอร์คู่ TLx และ THx โดยค่า x จะเป็นตัวบอกว่าเป็น Timer 0 หรือ Timer 1 การใช้ Timer สามารถใช้งานได้หลายโหมด ดังแสดงในรูปที่ 2.14 ซึ่งเราสามารถเซ็ทค่าโหมดการทำงานได้ โดยการโปรแกรมในรีจีสเตอร์ TMOD

13-Bit Timer Mode (Mode 0)

การทำงานในโหมด 0 นี้จะเป็นการใช้ Timer แบบ 13 บิท ดังแสดงในรูป 2.14(ก) ซึ่งจะใช้ 5 บิทล่างของ TLx โดยไม่สนใจ 3 บิทที่เหลือ และ 8 บิท ของ THx การทำงานในโหมดนี้ เมื่อบิทของ TLx นับไปจนเป็น "1" ทุกบิทจะส่ง Clock 1 ลูกให้ หนึ่งลูกให้ THx นับต่อและเมื่อนับเป็น "1" ทุกบิท และเปลี่ยนกลับเป็น "0" จะเกิด Overflow Flag เกิดขึ้น

16-Bit Timer Mode (Mode 1)

การทำงานในโหมดนี้จะเหมือนกับการทำงานในโหมด 0 แต่เป็น Timer แบบ 16 บิท ซึ่งการนับจะเริ่มตั้งแต่ 0000H,0001H,0002H ไปเรื่อย ๆ และจะเกิด Overlflow ขึ้น เมื่อมีการเปลี่ยนจาก FFFFH เป็น 0000H ดังรูปที่ 2.14(ข) ซึ่งเป็นการเซ็ท Overflow Flag และค่านี้จะเกิดขึ้นในบิท TFx ของรีจีสเตอร์ TCON ซึ่งสามารถอ่านและเขียนด้วยโปรแกรม

การใช้ตัว Timer นี้ค่าของบิทสูงสุด (MSB) คือค่าบิท 7 ของ THx ส่วนบิทต่ำสุด (LSB) คือบิท 0 ของ TLx บิท LSB จะเป็น Toggles เมื่อมีสัญญาณอินพุทเข้ามา ถูกหารด้วย 2 ดังนั้นจะพบว่าบิท MSB จะ Toggles ด้วยค่าความถี่ของสัญญาณอินพุทหารด้วย 65,536 (2^{16}) และค่า Timer รีจีสเตอร์นี้ (TLx/THx) สามารถอ่านและเขียนได้ด้วยการโปรแกรม ดังนั้นสามารถนำไปประยุกต์ใช้งานได้ตามต้องการ

8-Bit Auto – Reload Mode (Mode 2)

การทำงานในโหมด 2 เรียกอีกอย่างหนึ่งว่า 8-bit Auto – reload Mode โดยใช้ Timer ไบต์ต่ำ (TLx) เป็น Timer แบบ 8 บิท เมื่อไบต์ต่ำเกิด Overflows หรือเกิดการเปลี่ยนแปลงจาก FFH เป็น 00H จะมีการโหลดค่าที่เก็บไว้ในไบต์สูง (THx) ไปเก็บไว้ในไบต์ต่ำ (TLx) ซึ่งจะเป็นค่าเริ่มต้นของการนับครั้งต่อไป นิยมใช้สร้างเป็นฐานเวลาที่สามารถโปรแกรมได้ การทำงานในโหมดนี้แสดงดังรูปที่ 2.14(ค)

Split Timer Mode (Mode 3)

การทำงานในโหมด 3 นี้ ตัว Timer 1 จะไม่ทำงาน ตัว Timer 0 จะแยกเป็น 2 ตัว ตัวละ 8 บิท คือ TL0 และ TH0 เมื่อ Timer เกิด Overflows จะมีการเซ็ทบิท TF0 และ TF1 ดังแสดงในรูปที่ 2.14(ง)

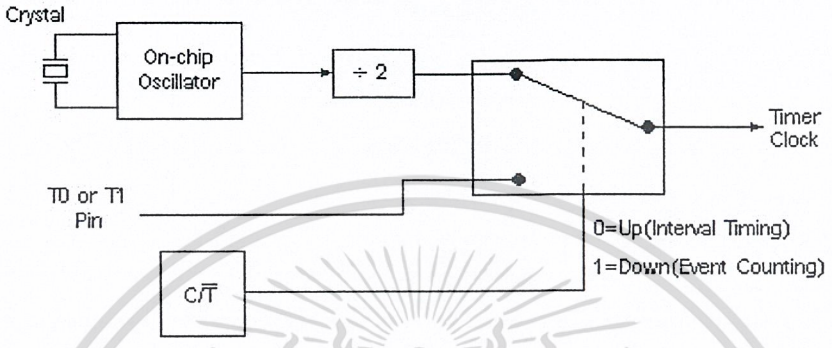
การทำงานในโหมด 3 นี้ Timer 1 จะไม่ถูกใช้งานแต่เราสามารถสวิตช์ให้ Timer 1 ไปทำงานในโหมดอื่นได้ แต่การทำงานของ Timer 1 จะไม่มีการอินเทอร์รัพท์เกิดขึ้น เพราะบิท TF1 ถูกใช้ในการนับของ TH0 ในการทำงานของโหมด 3 ไปแล้ว เราอาจมองว่าถ้าให้ Timer ทำงานในโหมด 3 ทำให้เรามี Timer เพิ่มขึ้น คือ TH0 และ TL0 ใน Timer 0 โหมด 3 และโปรแกรมให้ Timer 1 ไปทำงานในโหมดอื่นๆ

- **Clocking Source**

ในรูปที่ 2.14 ไม่ได้แสดงว่า Timer Clock นำมาจากที่ใดซึ่งการใช้ Timer นี้สามารถใช้ได้ 2 หน้าที่ คือเป็นตัวจับเวลา (Timer) และเป็นตัวนับ (Counter) ซึ่งสามารถโปรแกรมได้โดยการเซ็ทหรือรีเซ็ทบิท C / T ในรีจีสเตอร์ TMOD

การใช้เป็นตัวจับเวลา (Timer)

ถ้าบิท C/ T ใน TMOD เป็นลอจิก "0" จะเป็นการเลือกให้ Timer นำ Clock มาจากวงจร Oscillator ในชิพ ซึ่งสัญญาณนาฬิกาจะเข้ามาทุก ๆ Machine Cycle หรืออาจกล่าวได้ว่าค่าใน THx และ TLx จะมีค่าเพิ่มขึ้นด้วยอัตราการนับแต่ละครั้งใช้เวลาเท่ากับ 1/12 ของความถี่ของสัญญาณนาฬิกาที่ใช้บนชิพ ดังแสดงในรูปที่ 2.15 ถ้า MCS – 51 ใช้สัญญาณนาฬิกา 12 MHz การนับจะมีความถี่เท่ากับ 1 MHz



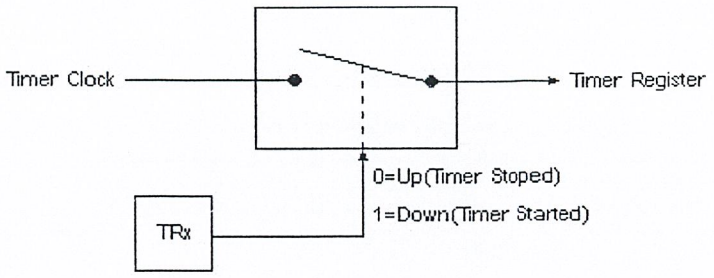
รูปที่ 2.15 ความถี่ของสัญญาณนาฬิกาที่เข้าขา Timer

การใช้เป็นตัวนับ (Counter)

ถ้าบิท C / T เป็น "1" ตัว Timer จะนำ Clock มาจากภายนอกโดยใช้ขา P3.4 หรือ T0 เป็นขา Input Clock ให้กับ Timer 0 และใช้ขา P3.5 หรือ T1 เป็น Input Clock ให้กับ Timer 1 ดังรูปที่ 2.15 หรืออาจมองว่า ถ้าจะให้นับอะไรสัญญาณที่จะนับให้ต่อกับขา T0 และ T1 ในการใช้เป็น Counter สัญญาณที่เข้ามามีการเปลี่ยนแปลงจาก "1" เป็น "0" จะทำให้วงจรนับ TLx มีค่าเพิ่มขึ้น 1 ภายใน MCS – 51 นี้จะตรวจสอบขาอินพุท T0 และ T1 ในช่วงเวลาเฟส 2 ของ State 5 (S5P2) ถ้าพบว่ามีค่าเป็น "1" ต่อมาในอีกหนึ่ง Machine Cycle ที่เฟส 2 ของ State 5 (S5P2) ลอจิกอินพุทเปลี่ยนเป็น "0" จะทำให้ค่าใน Timer เพิ่มขึ้น 1 ดังนั้น จะเห็นได้ว่าการนับ 1 ครั้งจะต้องใช้เวลา 2 Machine Cycles ดังนั้นความถี่สูงสุดที่จะให้ Timer ทำงานเป็น Counter นับได้ จะมีค่ามากที่สุด 500 kHz ถ้า MCS – 51 ทำงานที่ความถี่สัญญาณนาฬิกา 12 MHz

- **การเริ่ม , หยุด และการควบคุม Timer**

ในรูปที่ 2.14 จะแสดงลักษณะของ Timer Registers ซึ่งจะเห็นว่าประกอบด้วย TLx และ THx และเมื่อเกิด Overflow จะเกิดเอาท์พุทที่บิท TFx สำหรับสัญญาณนาฬิกาที่จะเข้าไปใน Time จะมาจาก 2 ส่วนดังแสดงในรูปที่ 2.15 ต่อไปจะกล่าวถึงว่าเราจะควบคุมให้เริ่ม , หยุดตัว Timer ได้อย่างไร วิธีเริ่มและหยุดตัว Timers สามารถควบคุมได้ที่บิท TRx ในรีจีสเตอร์ TCON โดยปกติแล้ว TRx จะเคลียร์หลังจากที่ระบบถูกรีเซ็ท ซึ่งจะเป็นการให้ Timer ไม่นับและ TRx นี้จะเซ็ทได้จากชุดคำสั่ง หรือการโปรแกรม พิจารณารูปที่ 2.16



รูปที่ 2.16 การใช้บิทควบคุม TR

ตัวบิท TRx จะเป็นส่วนที่สามารถเข้าถึงข้อมูลในระดับบิทได้ (Bit Addressable)ในรีจีสเตอร์ TCON ถ้าจะให้ TIMER 0 เริ่มทำงานจะเขียนคำสั่งได้ดังนี้

SETB TR0

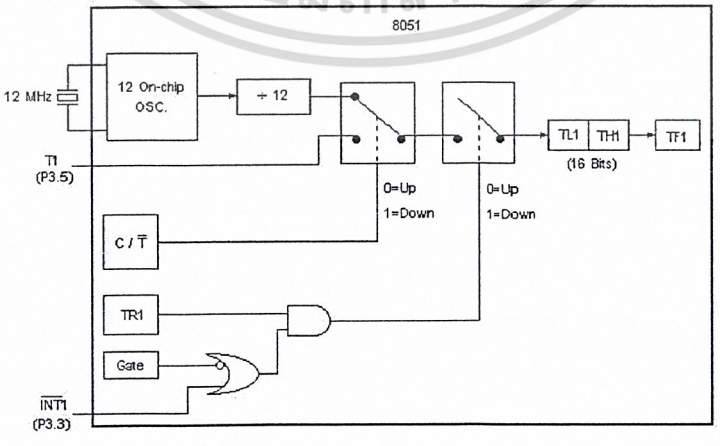
ถ้าจะหยุดทำงานเขียนคำสั่งได้ดังนี้

CLR TR0

ในการเขียนโปรแกรมภาษาแอสแซมบลี สามารถใช้สัญลักษณ์ TR0 ในคำสั่ง SETB TR0 เลยได้ เพราะตัวแอสแซมบลีจะตีความ TR0 เป็น Bit Address ตำแหน่ง 8CH

วิธีควบคุม Timer สามารถควบคุมได้ที่บิท GATE ใน TMOD และขาอินเทอร์รัพท์จากภายนอก INTx ถ้า INT0 เป็นลอจิก “0” และโปรแกรมให้ Timer 0 ทำงานในโหมด 2 เมื่อ TL0/TH0 = 0000H, GATE = 1 และ TR0 = 1 เมื่อ INT0 ขึ้นเป็นลอจิก “1” ตัว Timer จะ “Gate On” และจะให้สัญญาณนาฬิกาความถี่ 1 MHz เมื่อ INT0 ลงเป็น “0” ตัว Timer “ Gate Off “ สัญญาณที่ได้จะมีความกว้างของสัญญาณนาฬิกา 1 μS ส่งเข้าไปใน TL0/TH0

รูปที่ 2.17 จะเป็นระบบที่สมบูรณ์ของ Timer 1 เมื่อทำงานในโหมด 1 ซึ่งเป็น 16-bit Timer โดยใช้รีจีสเตอร์ TL1 / TH1 และ Overflow Flag TF1 ในรูปจะเห็นถึงการควบคุมแหล่งกำเนิด Clock การเริ่มทำงาน และการหยุดทำงาน



รูปที่ 2.17 ระบบทั้งหมดของ Timer 1

- **Intializing And Accessing Timer Register**

การใช้งาน Timer เริ่มแรกจะต้องโปรแกรมเพื่อเลือกโหมดการทำงานของ Timer ก่อนเมื่อเริ่มใช้งานก็โปรแกรมให้ เริ่มทำงาน, หยุดทำงาน, อ่าน และ เคลียร์ค่า Flag Bits อ่านค่า Timer Registers ตามลำดับ เพื่อนำไปประยุกต์การใช้งานต่อไป

TMOD คือ รีจีสเตอร์ที่ต้องโปรแกรม โดยเซ็ทโหมดการทำงานก่อน ตัวอย่างเช่น ถ้าให้ Timer 1 เป็น 16-bits Timer (โหมด 1) นับสัญญาณนาฬิกาบนชิพ สามารถเขียนคำสั่งได้ดังนี้

MOV TMOD, #00010000B

ผลที่ได้จากคำสั่งข้างบนคือ เซ็ทบิท M1 = 0 และ M0 = 1 ซึ่งเป็นการเลือกโหมด 1 และให้ C / T= 0 และ GATE = 0 ซึ่งเป็นการใช้สัญญาณนาฬิกาจากภายในหรือใช้เป็น Timer และตัว Timer นี้จะยังไม่ทำงาน ถ้าบิทควบคุม TR1 ยังไม่ได้เซ็ท

ถ้าให้ Timer นี้นับขึ้นโดยใช้รีจีสเตอร์ TL1 / TH1 และจะเซ็ทบิท Overflow Flag เมื่อรีจีสเตอร์เปลี่ยนจาก FFFFH เป็น 0000H โดยให้นับเวลาไป 100 μS หรือให้ TL1 / TH1 นับสัญญาณนาฬิกาได้ 100 ลูก ดังนั้นค่าเริ่มต้นของ TL1 / TH1 จะไม่เริ่มที่ 0000H จะต้องเริ่มที่ FFFFH ลบด้วย 100 ลูก หรือ FF9CH เพื่อให้นับไปถึง FFFFH และเปลี่ยนเป็น 0000H ได้สัญญาณนาฬิกา 100 ลูกพอดี สามารถเขียนคำสั่งได้ดังนี้

MOV TL1 , # 9CH

MOV TH1 , # 0FFH

ถ้าให้ Timer เริ่มทำงานก็ให้บิทควบคุมดังนี้

SETB TR1

จากนั้นบิท Overflow Flag จะส่งออกมาหลังเวลาผ่านไป 100 μS ซึ่งเราสามารถเขียนโปรแกรมเป็นโปรแกรมวนลูป 100 μS ได้ โดยตรวจสอบบิท TF1 ว่าถูกเซ็ทหรือไม่ ถ้าไม่เซ็ทก็ให้วนลูปต่อไปดังนี้

CLR TR1

CLR TF1

การใช้แบบ Readind a Timer “On the Fly”

การใช้งานแบบประยุกต์บางงานจะต้องอ่านค่าจาก Timer Register เนื่องจากตัว Timer Register มีขนาด 2 ไบต์ ถ้าหากไบต์ต่ำเกิด Overflow จะทดเข้าไบต์สูง ถ้าหากเขียนโปรแกรมให้อ่านค่าจากไบต์ต่ำก่อน แล้วจึงอ่านไบต์สูงข้อมูลที่ได้อาจเกิดข้อผิดพลาดได้เนื่องจากไบต์ต่ำมีการเปลี่ยนแปลงเร็วกว่าไบต์สูง การอ่านข้อมูลควรอ่านจากไบต์สูงก่อน แล้วจึงกลับมาอ่านไบต์ต่ำ จากนั้นอ่านข้อมูลไบต์สูงอีกครั้ง ถ้าค่าไบต์สูงที่อ่านได้ไม่มีการเปลี่ยนแปลงให้ใช้ค่านั้นได้เลย แต่ถ้ามีการเปลี่ยนแปลงให้อ่านอีกครั้ง ถ้าต้องการอ่านข้อมูลจาก TL1 / TH1 เข้าในรีจีสเตอร์ R6 / R7 อาจเขียนโปรแกรมได้ดังนี้

AGAIN : MOV A, TH1

MOV R6, TL1

CJNE A, TH1, AGAIN

MOV R7, A

- **Short Intervals And Long Intervals**

ถ้า MCS – 51 ทำงานที่ความถี่สัญญาณนาฬิกา 12 MHz ถ้าให้ Timers ใช้วงจร Oscillator บนชิพ สัญญาณนาฬิกาจะถูกหารด้วย 12 และ Timer จะทำงานด้วยความถี่ 1 MHz ถ้าต้องการใช้โปรแกรมสร้างสัญญาณนาฬิกาออกมาอาจทำได้โดยง่าย ซึ่งพิจารณาจากการทำงานชุดคำสั่งต่าง ๆ ของ MCS – 51 ใน 1 Machine Cycle จะใช้เวลา 1μS ในตารางที่ 2.6 จะแสดงความกว้างของสัญญาณที่สร้างขึ้นจาก MCS – 51 ที่ทำงานด้วย Crystal ความถี่ 12 MHz

Maximum Interval in Microseconps	**Technique**
≈10	Software Tuing
256	8 – bit Timer with Auto-reload
65536	8 – bit Timer
No Limit	16 – bit Timer Plus Software Loops

ตารางที่ 2.6 ค่าสูงสุดของการใช้ Timer โหมดต่าง ๆ

2.2.4 การอินเทอร์รัพท์

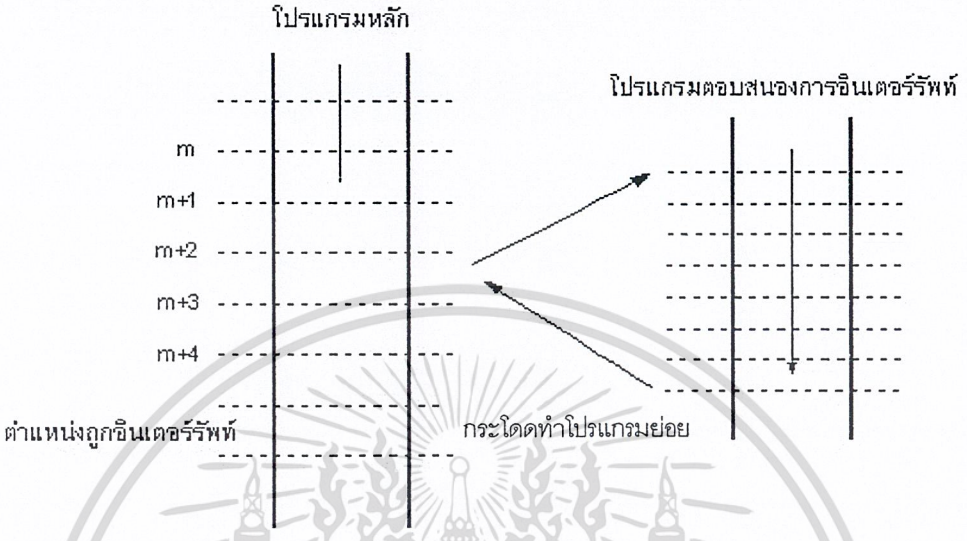
การทำงานของระบบคอมพิวเตอร์ โดยทั่วไปมักมีอุปกรณ์ภายนอกต่อร่วมอยู่ ถ้าคอมพิวเตอร์ต้องการทำงานกับอุปกรณ์ภายนอกจะต้องคอยตรวจสอบอุปกรณ์เหล่านั้นเสมอ ตัวอย่างเช่น ถ้าหากให้คอมพิวเตอร์พอร์ตหนึ่งต่ออยู่กับหลอด LED 7 ส่วน อีกพอร์ตหนึ่งต่อกับสวิตช์ ถ้าระบบของเราทำงานเป็นนาฬิกาเดินไปให้คอยตรวจสอบสวิตช์ด้วยว่ามีการกดหรือยัง การทำงานแบบนี้เรียกว่า Polling Method คือตัวไมโครโปรเซสเซอร์จะต้องคอยตรวจสอบอุปกรณ์อินพุทตลอดเวลาว่ามีข้อมูลเข้ามาหรือยัง การทำงานแบบนี้ ถ้ามีอุปกรณ์ภายนอกหลายตัวระบบต้องตรวจสอบอุปกรณ์ภายนอกหลายตัว ทำให้เสียเวลาในการทำงานหลักไป การทำงานอีกแบบหนึ่งจะให้ CPU ทำงานหลัก ถ้ามีการกดสวิตช์เมื่อไรให้นาฬิกาหยุดเดินทันที การทำงานในลักษณะนี้ CPU ไม่ต้องเสียเวลาในการตรวจอุปกรณ์ภายนอก ถ้าอุปกรณ์ภายนอกต้องการติดต่อกับ CPU อุปกรณ์ภายนอกจะส่งสัญญาณมาบอก CPU เอง ระบบนี้เรียกว่าการอินเทอร์รัพท์ (Interrupt)

- **ขบวนการเกิดอินเทอร์รัพท์**

ถ้าหากคอมพิวเตอร์กำลังทำงานโปรแกรมหลักอยู่เมื่อมีการอินเทอร์รัพท์เข้ามาคอมพิวเตอร์จะละทิ้งโปรแกรมหลัก แต่ไปทำงานโปรแกรมตอบสนองการอินเทอร์รัพท์ (Interrupt Service Routine) เมื่อทำโปรแกรมตอบสนองอินเทอร์รัพท์เสร็จ คอมพิวเตอร์จะกลับมาทำโปรแกรมเดิม พิจารณารูปที่ 2.18

ถ้า CPU กำลังทำงานโปรแกรมหลักอยู่ เช่นกำลังทำคำสั่งในตำแหน่งของหน่วยความจำที่ m, m+1, m+2 ไปเรื่อย ๆ โดย PC จะชี้ที่ตำแหน่งที่จะอ่านค่าคำสั่งถัดมา เมื่อโปรแกรมทำงานมาถึงตำแหน่งที่ m+3 แล้วเกิดการอินเทอร์รัพท์ขึ้น (ขณะนั้น PC อยู่ที่ m+4) โปรแกรมจะต้องทำงานโปรแกรม

ตอบสนองการอินเทอร์รัพท์ โดยย้าย PC ไปที่ตำแหน่งที่เก็บโปรแกรมตอบสนองการ อินเทอร์รัพท์ จากนั้นจะเก็บค่า PC เดิมลงในหน่วยความจำสแตค เมื่อคอมพิวเตอร์ทำงานโปรแกรมตอบสนองการอินเทอร์รัพท์เสร็จสิ้นลง จะคืนค่าใน สแตค (m+4) ให้กับ PC ทำโปรแกรมหลักต่อไป



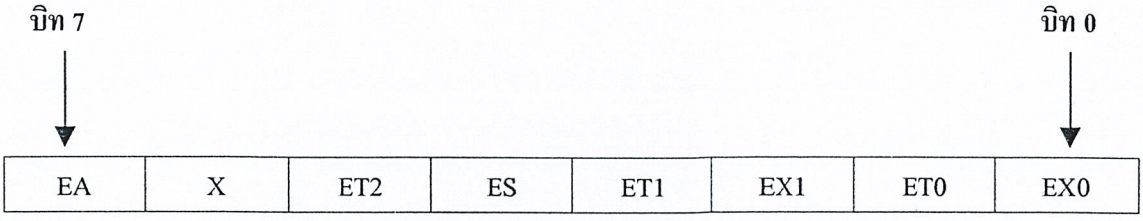
รูปที่ 2.18 ขั้นตอนการทำงานของโปรแกรมเมื่อถูกอินเทอร์รัพท์

- **สัญญาณอินเทอร์รัพท์**

แหล่งกำเนิดสัญญาณอินเทอร์รัพท์ที่ใช้กับ MCS – 51 มีสองชนิดคือ อินเทอร์รัพท์ภายในและภายนอก โดยอินเทอร์รัพท์ภายในจะเกิดขึ้นจากภายในตัว MCS – 51 เอง ได้แก่สัญญาณจาก ไทเมอร์แฟลก 0 (TF0) ไทเมอร์แฟลก 1 (TF1) และพอร์ตอนุกรม สำหรับอินเทอร์รัพท์ภายนอกเกิดจากสัญญาณที่กระตุ้นเข้ามาทางขา INT0 และ INT1 เมื่อมีสัญญาณอินเทอร์รัพท์จากแหล่งต่าง ๆ เข้ามา เราสามารถโปรแกรมได้ว่าจะให้ MCS – 51 ยอมให้มีการอินเทอร์รัพท์ได้หรือไม่ โดยการโปรแกรมไปที่ รีจีสเตอร์ IE (Interrupt Enable) และถ้ามีสัญญาณอินเทอร์รัพท์มากจากแหล่งต่าง ๆ หลายแหล่งพร้อมกันเราสามารถจัดลำดับได้ว่า จะให้อินเทอร์รัพท์ใดเกิดก่อน โดยการโปรแกรมไปที่ อินเทอร์รัพท์ไพออริตี้ IP (Interrupt Priority) รีจีสเตอร์ทั้งสองตัวมีรายละเอียดดังนี้

Interrupt Enables

เป็นรีจีสเตอร์ที่สามารถเข้าถึงข้อมูลระดับบิทได้ ใช้สำหรับกำหนดค่าว่าถ้าเกิดการอินเทอร์รัพท์จากแหล่งต่าง ๆ จะทำอินเทอร์รัพท์เหล่านั้นหรือไม่ โดยรายละเอียดของบิทต่าง ๆ มีดังตาราง ที่ 2.7



บิท	ชื่อบิท	ตำแหน่งบิท	รายละเอียด
IE.7	EA	AFH	ถ้าเซ็ทยอมให้มีการอินเทอร์รัพท์
IE.6	-	AEH	ไม่ใช้งาน
IE.5	ET2	ADH	Enable อินเทอร์รัพท์จาก Timer 2 (ใช้กับ 8052)
IE.4	ES	ACH	Enable อินเทอร์รัพท์จากพอร์ตอนุกรม
IE.3	ET1	ABH	Enable อินเทอร์รัพท์จาก Timer 1
IE.2	EX1	AAH	Enable อินเทอร์รัพท์จาก INT1
IE.1	ET0	A9H	Enable อินเทอร์รัพท์จาก Timer 0
IE.0	EX0	A8H	Enable อินเทอร์รัพท์จาก INT0

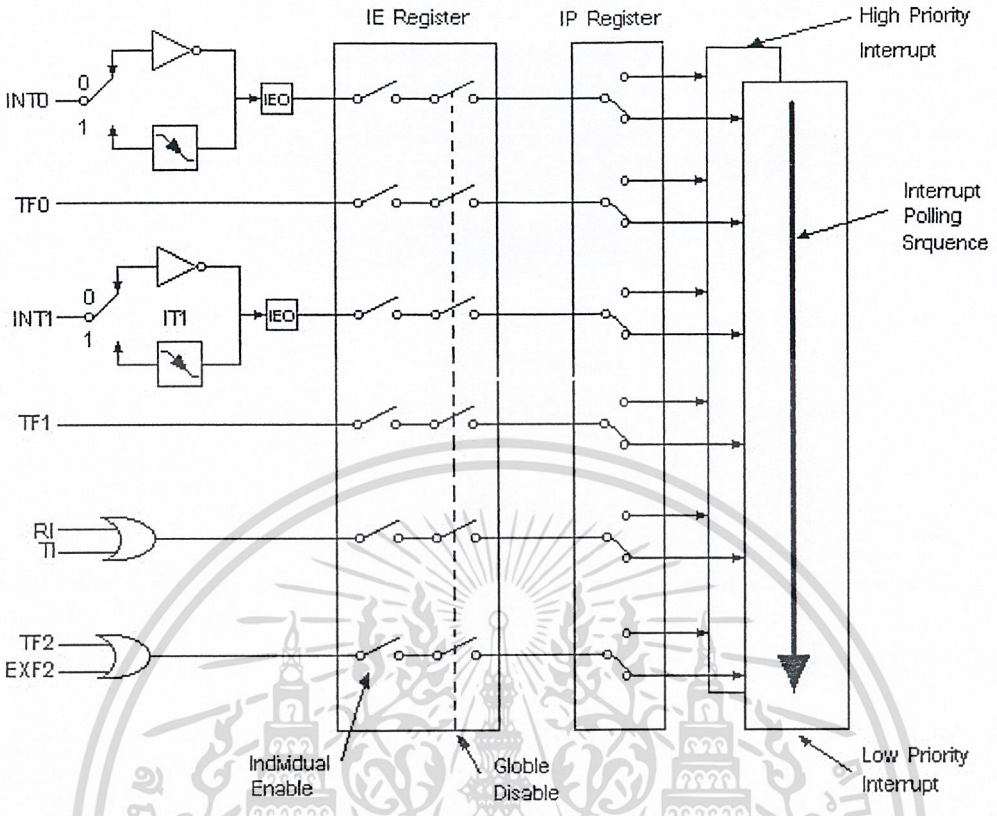
ตารางที่ 2.7 บิทต่าง ๆ ของรีจีสเตอร์ IE

Interrupt Priority

เป็นรีจีสเตอร์ที่สามารถเข้าถึงข้อมูลระดับบิทได้ใช้ในการจัดลำดับความสำคัญของการอินเทอร์-รัพท์ซึ่งสามารถจัดได้สองลำดับ ถ้าเป็น "1" หมายความว่ามีความสำคัญสูงสุด ถ้าเป็น "0" หมายความว่ามีลำดับความสำคัญต่ำสุด ความหมายของบิทต่าง ๆ แสดงได้ดังตารางที่ 2.8 ถ้าหากกำหนดให้มีความสำคัญเป็น "1" เหมือนกันหมด MCS – 51 จะจัดลำดับความสำคัญใหม่ดังนี้

ลำดับ	อินเทอร์รัพท์
1 (สูงสุด)	IE0
2	TF0
3	IE1
4	TF1
5 (ต่ำสุด)	Serial Port

ตารางที่ 2.8 แสดงลำดับความสำคัญของการอินเทอร์รัพท์



รูปที่ 2.19 รีจีสเตอร์ต่าง ๆ ที่เกี่ยวข้องกับการอินเทอร์รัพท์

บิท	ชื่อบิท	ตำแหน่งบิท	รายละเอียด
IP.7	-	-	ไม่ใช้งาน
IP.6	-	-	ไม่ใช้งาน
IP.5	PT2	0BDH	ใช้กับ Timer 2 (8052)
IP.4	PS	0BCH	ใช้กับพอร์ตอนุกรม
IP.3	PT1	0BBH	ใช้กับ Timer 1
IP.2	PX1	0BAH	ใช้กับอินเทอร์รัพท์จาก INT1
IP.1	PT0	0B9H	ใช้กับ Timer 0
IP.0	PX0	0B8H	ใช้กับอินเทอร์รัพท์จาก INT0

ตารางที่ 2.9 บิทและหน้าที่ต่าง ๆ ของรีจีสเตอร์ IP

จากรูปที่ 2.19 แสดงการอินเทอร์รัพท์จากแหล่งต่าง ๆ ที่มีผลกับ MCS – 51 ถ้าเป็นเบอร์ 8051 8031 จะถูกอินเทอร์รัพท์ได้ 5 แหล่ง ถ้าเป็นเบอร์ 8052,8032 จะถูกอินเทอร์รัพท์ได้ 6 แหล่ง โดยเพิ่มอินเทอร์รัพท์จาก Timer 2 ในรูปที่ 2.19 จะแสดงให้เห็นว่า ถ้า MCS – 51 จะถูกอินเทอร์รัพท์ได้จะต้องเซ็ทค่า Global Enable ในรีจีสเตอร์ IE นอกจากนี้ยังกำหนดได้ว่าจะให้อินเทอร์รัพท์ใดเกิดได้ โดยการเซ็ทค่า Interrupt Enable ของอินเทอร์รัพท์จากแหล่งต่าง ๆ ในรีจีสเตอร์ IE จากรูปยังแสดงให้เห็นอีกว่าเมื่อมีการอินเทอร์รัพท์เข้ามาจะมีผลต่อแฟลกใด เช่นถ้า INT0 เป็น “1” บิท IE0 จะเป็น “1” หมายความว่าถูกอินเทอร์รัพท์ โดยแฟลกต่าง ๆ ที่มีผลจากการถูกอินเทอร์รัพท์แสดงได้ดังตารางที่ 2.9

อินเทอร์รัพท์	แฟลก	ประกอบอยู่ในรีจีสเตอร์
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 1	TF1	TCON.7
Timer 0	TF0	TCON.5
Serial port	T1	SCON.1
Serial port	RI	SCON.0
Timer 2	TF2	T2CON.7 (8052)
Timer 2	EXF2	T2CON.6 (8052)

ตารางที่ 2.10 แฟลกที่จะทำงานเมื่อถูกอินเทอร์รัพท์

จากตารางจะเห็นว่า ถ้ามีการอินเทอร์รัพท์จากภายนอกเข้ามา ตัวที่จะอินเทอร์รัพท์ MCS – 51 คือ บิทแฟลก IE0 ซึ่งอยู่ในรีจีสเตอร์ TCON ถ้ามีการสื่อสารแบบอนุกรม เมื่อข้อมูลถูกส่งไปหมดแล้วจะอินเทอร์รัพท์ MCS – 51 ทางบิทแฟลก TI ถ้ารับข้อมูลหมดแล้วจะอินเทอร์รัพท์ MCS – 51 ทางบิทแฟลก RI ซึ่งอยู่ในรีจีสเตอร์ SCON และถ้าใช้ Timer 0 ในการนับเมื่อเกิด Overflow สามารถอินเทอร์รัพท์ MCS – 51 ได้ทางบิท TF0

- **การทำงานของระบบหลังถูกอินเทอร์รัพท์**

เมื่อ MCS – 51 ถูกอินเทอร์รัพท์จะต้องกระโดดไปทำโปรแกรมตอบสนองการอินเทอร์รัพท์โดยตำแหน่งที่จะกระโดดไปเรียกว่า อินเทอร์รัพท์เวกเตอร์ (Interrupt Vectors) เมื่อทำโปรแกรมตอบสนองการอินเทอร์รัพท์เรียบร้อยแล้ว MCS – 51 จะกระโดดมาทำงานยังตำแหน่งเดิม โดยก่อนที่จะกระโดดไปทำโปรแกรมตอบสนองการอินเทอร์รัพท์จะต้องเก็บค่าตำแหน่งเดิมไว้ โดยเก็บค่า PC ลงหน่วยความจำสแตคซึ่งอยู่ที่หน่วยความจำที่ถูกชี้โดยรีจีสเตอร์ SP เมื่อทำโปรแกรมตอบสนองการอินเทอร์รัพท์เสร็จแล้วจะคืนค่าในหน่วยความจำสแตคให้ PC ตามเดิม ค่าอินเทอร์รัพท์เวกเตอร์ของ MCS – 51 แสดงได้ดังตารางที่ 2.10

อินเทอร์รัพท์	อินเทอร์รัพท์เวกเตอร์
System Reset	0000H
External 0	0003H
Timer 0	000BH
External 1	0013H
Timer 1	001BH
Serial Port	0023H
Timer 2	002BH

ตารางที่ 2.11 อินเทอร์รัพท์เวกเตอร์ของอินเทอร์รัพท์ต่าง ๆ

จากตารางจะเห็นว่าถ้าระบบถูกอินเทอร์รัพท์จากภายนอกทาง INT0 ตัว MCS – 51 จะกระโดดไปทำงานที่ตำแหน่ง 0003H ถ้าระบบถูกอินเทอร์รัพท์จาก Timer 0 จะกระโดดไปทำงานตำแหน่ง 000BH

2.2.5 การใช้งานพอร์ตสื่อสารข้อมูลอนุกรมแบบ Single Processor

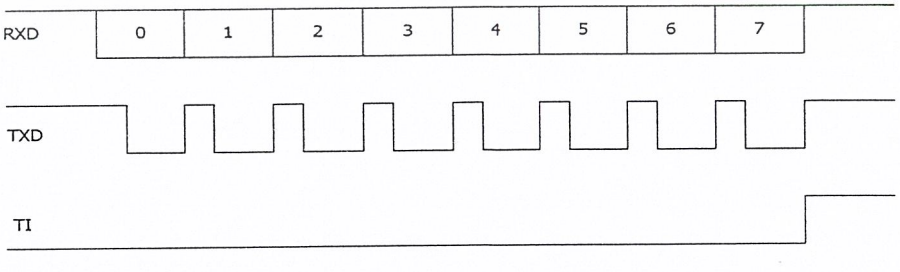
- **พอร์ตสื่อสารอนุกรม**

พอร์ตการสื่อสารมีโครงสร้างการทำงานในแบบที่เรียกว่าฟูลดูเพล็กซ์(Full duplex)ในการรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน โดยทางด้านส่งใช้ขา TxD (พอร์ต 3.1) ทางด้านรับใช้ขาRxD (พอร์ต 3.0) SBUF ใช้เป็นบัฟเฟอร์สำหรับรับและส่งข้อมูลอนุกรม

พอร์ตสื่อสารอนุกรมสามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิท SM1 และ SM0 ซึ่งอยู่ในรีจีสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมด ของพอร์ตสื่อสารอนุกรม มีดังนี้

โหมด 0 : พอร์ตสื่อสารอนุกรม 8 บิท โดยการส่งจะเลื่อนออกทีละบิทโดยส่งบิท D0 ออกไปก่อนทาง

ขา RxD และไม่มีการส่ง Start bit แต่จะส่ง Shift Clock ทางขา TxD (ความเร็ว1/12 เท่าของ CPU Clock)



โหมด 1: พอร์ตสื่อสารอนุกรม 10 บิทข้อมูล 8 บิท 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลง(ความเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิท SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1)

$$\text{Baud Rate Mode 1,3} = \frac{2^{SMOD} \times Oscillator..Frequrency}{32 \times 12 \times [256 - (TH1)]}$$ โดยใช้ Timer 1

$$\text{Baud Rate Mode 1,3} = \frac{Oscillator..Frequrency}{32 \times [65536 - (RCAP2H, RCAP2L)]}$$ โดยใช้ Timer 2

โหมด 2 : พอร์ตสื่อสารอนุกรม 11 บิท ข้อมูล 9 บิท 1 Start bit และ 1 Stop bit(TB8 นิยมนำมาใช้ส่ง Parity bit) ความเร็วในการรับส่งข้อมูลเท่ากับ 1/32 และ 1/64 ของ CPU Clock โดยขึ้นกับบิท SMOD ใน PCON

- Baud Rate Mode2 = (1/32)(Osc Freq) เมื่อ SMOD = 1
- Baud Rate Mode2 = (1/64)(Osc Freq.) เมื่อ SMOD = 0

โหมด 3: พอร์ตสื่อสารแบบ 11 บิท URAT โดยส่งข้อมูล 9 บิท 1 Start bit และ 1 Stop bit เหมือนโหมด 2 ยกเว้นอัตราโอเวอร์โฟลว์ของ Timer1,2 สำหรับ 8051 หรืออัตราโอเวอร์โฟลว์ของ Timer2 (สำหรับ 80C154D)

$$\text{Baud Rate Mode 3} = \frac{2^{SMOD} \times Oscillator.Freq.}{32 \times 12 \times [256 - (TH1)]}$$ โดยใช้ Timer 1

$$\text{Baud Rate Mode 3} = \frac{Oscillator.Freq.}{32 \times [65536 - (RCAP2, RCAP2L)]}$$ โดยใช้ Timer 2

- **การเชื่อมต่อไมโครโปรเซสเซอร์เพื่อรับส่งข้อมูลอนุกรม(UART)**

  มีอยู่ 2 โหมดด้วยกันคือ

  - Single Processor Mode
  - Multiprocessor Mode

Single Processor Mode : ในโหมดนี้เราจะใช้ไมโครคอนโทรลเลอร์ 2 ตัวเชื่อมเข้าหากัน

Multiprocessors Mode : ในโหมดนี้จะใช้ไมโครคอนโทรลเลอร์ 1 ตัวเป็นตัวแม่ (Master) และอีก 0 – 256 ตัวลูก (Slave) รีจีสเตอร์ที่ใช้ควบคุมการรับส่งข้อมูลอนุกรม

SM0	SM1	โหมด	การทำงาน
0	0	0	Shift register อัตราการรับหรือส่งข้อมูลเท่ากับ 1/12 ของ CPU
0	1	1	8 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดได้จาก Timer 1,2
1	0	2	9 bit UART อัตราในการรับหรือส่งข้อมูล = 32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์ ขึ้นกับบิท SMOD ใน PCON
1	1	3	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดที่ Timer 1,2

ตารางที่ 2.12 การเลือกโหมดการทำงานของการสื่อสารแบบอนุกรม

SM2 บิทเลือกการทำงานแบบ Single Processor Mode หรือ Multiprocessor Mode

1 : เลือก Multiprocessor Mode ใช้กับโหมด 2,3

0 : เลือก Single Processor Mode ใช้กับทุกโหมด

เมื่อเลือกการทำงานรับข้อมูลแบบ Multiprocessor Mode แล้ว

ถ้าข้อมูลบิทที่ 9 ที่รับได้มีค่าเป็น 1 RI จะเซ็ท

ถ้าข้อมูลบิทที่ 9 ที่รับได้มีค่าเป็น 0 RI จะไม่เซ็ท

REN (Receive Enable) บิทควบคุมให้รับหรือไม่รับข้อมูล

1 : ให้รับข้อมูลได้

0 : ห้ามรับข้อมูล

TB8 (Transmit bit D8) ข้อมูลบิทที่ 9 ที่จะส่งออกไปโหมด 2,3 ให้ใส่ในบิทนี้ได้เลย

RB8 (Receive bit D8) ข้อมูลบิทที่ 9 ที่รับเข้ามาจะมาเก็บในบิทนี้ (ข้อมูลในบิทที่ 9 ก็คือค่าใน TB8 ทางด้านส่งนั้นเอง)

TI แฟลกซ์ TI จะเป็น 1 เมื่อสิ้นสุดการส่งข้อมูล 1 ไบต์

RI แฟลกซ์ RI จะเป็น 1 เมื่อรับข้อมูลเสร็จ 1 ไบต์(บิท RI, TI ผู้เขียนโปรแกรมจะต้องเคลียร์เอง)

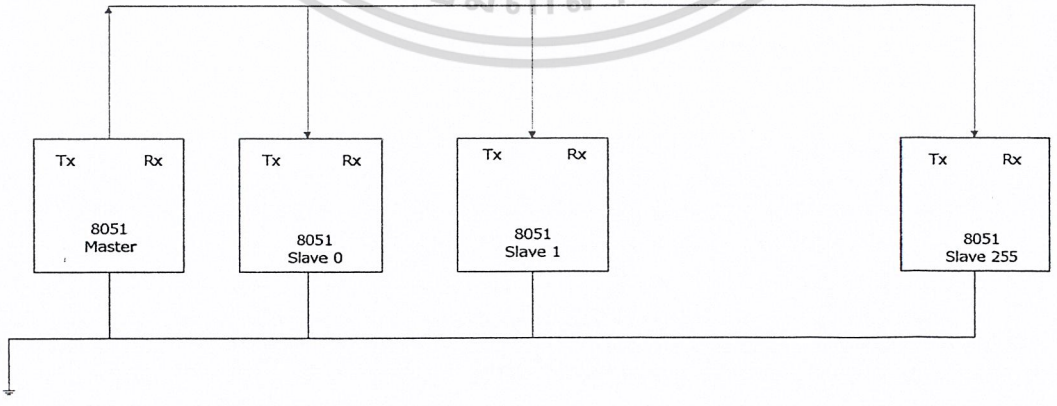
Baud Rate	Fosc	SMOD	TIMER 1		
		ในPCON	C/T	MODE	Reload value
(MODE 0) Max:1 MHz	12 MHz	X	X	X	X
(MODE 2) Max: 375 KHz	12 MHz	1	X	X	X
(MODE 2) Min : 187.5 KHz	12 MHz	0	X	X	X
MODE 1,3 : 62.5K	12 MHz	1	0	2	FFH
19.2K	11.059KHZ	1	0	2	FDH
9.6 K	11.059KHZ	0	0	2	FDH
4.8 K	11.059KHZ	0	0	2	FAH
2.4 K	11.059KHZ	0	0	2	F4H
1.2 K	11.059KHZ	0	0	2	E8H
137.5	11.059KHZ	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12MHz	0	0	1	FEEBH

ตารางที่ 2.13 การใช้ Timer 1 กำหนด บอดเรท

2.2.6 การใช้งานพอร์ตสื่อสารอนุกรมแบบ Multiprocessors

การสื่อสารข้อมูลแบบอนุกรมโดยใช้หลักการของมัลติโปรเซสเซอร์

ในการติดต่อสื่อสารข้อมูลอนุกรมโดยใช้หลักการไมโครโปรเซสเซอร์ จำเป็นที่จะต้องกำหนดค่าแอดเดรสของตัว Slave ออกไป(Slave มีทั้งหมด 256) แล้วตามด้วย ดาต้าไบต์ เป็นไบต์ที่ 2 ซึ่งมีข้อมูลส่งไปยัง Slave การต่อสัญญาณจากตัว Master ไปยัง Slave ทั้ง 256 ตัวดังแสดงในรูปที่ 2.20



รูปที่ 2.20 การต่อสัญญาณจากตัว Master ไปยัง Slave แบบมัลติโปรเซสเซอร์

• **รูปแบบการส่งสัญญาณที่ส่งออกจากตัว Master**

MCS-51 กำหนดรูปแบบการรับส่งข้อมูลได้ 2 แบบคือ Single Processor Mode โดยเซ็ท SM2 = 0 และ Multiprocessor Mode โดยเซ็ท SM2 = 1 การส่งข้อมูลจะต้องเป็น (โหมด 2,3) และโหมด 3 เท่านั้นเพราะการส่งข้อมูลได้ 11 บิท โดยบิทแรกคือ Start bit อีก 8 บิท ข้อมูลตามด้วยบิทที่ 9 ซึ่งใช้เป็นตัวบอกว่าไบต์ที่ส่งไปนี้เป็นแอดเดรสไบต์ หรือ ดาต้าไบต์ ถ้าโปรแกรม Multiprocessor Mode แล้ว

ทางด้านส่ง **บิทที่ 9 (TB8) = 1 หมายถึงเลือกส่ง** แอดเดรสไบต์

บิทที่ 9 (TB8) = 0 หมายถึงเลือกส่ง ดาต้าไบต์

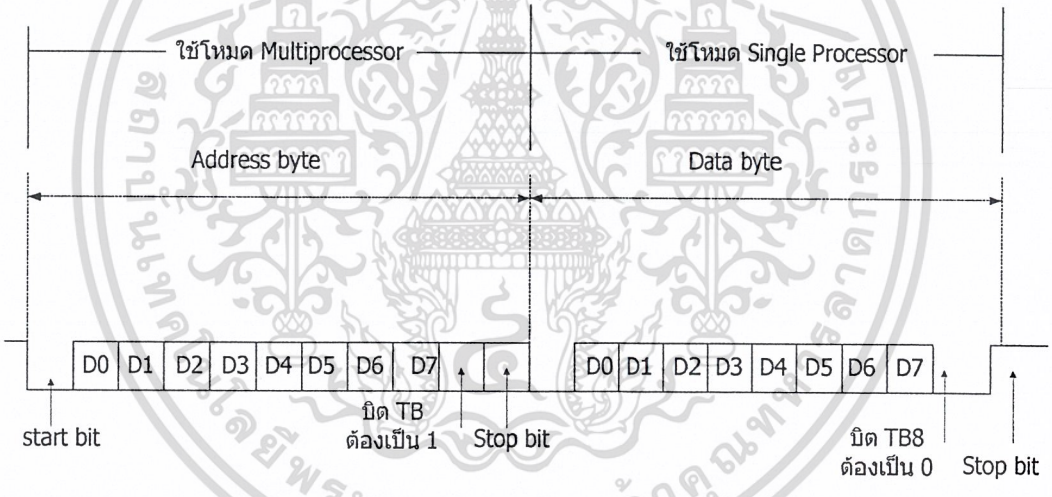
ทางด้านรับ บิทที่ 9 (TB8) = 1 หมายถึงค่าที่รับมาคือ แอดเดรสไบต์

บิทที่ 9 (TB8) = 0 หมายถึงค่าที่รับมาคือ ดาต้าไบต์

การโปรแกรมบิทที่ 9 เป็น 1 หรือ 0 เราจะต้องโปรแกรมที่บิท TB8 ใน SCON

เมื่อการทำงานแบบ Multiprocessor Mode RI จะ SET เมื่อค่าบิทที่ 9 ที่รับมามีค่าเป็น 1

เมื่อการทำงานแบบ Single Processor Mode RI จะ SET โดยไม่สนใจ ค่าบิทที่ 9



รูปที่ 2.21 สัญญาณที่ขา TxD ของตัว Master

หมายเหตุ : บิท TB8 ทางด้านส่งจะเข้าไปเก็บใน RB8 ทางด้านรับ บิทนี้อยู่ใน SCON การส่งข้อมูล โหมดนี้จะต้องส่งแอดเดรสไบต์ ออกไปก่อน บิทที่ 9 (TB8) ต้องเป็น 1 หลังจากนั้นให้ เคลียร์ TB8 เพื่อส่งดาต้าไบต์

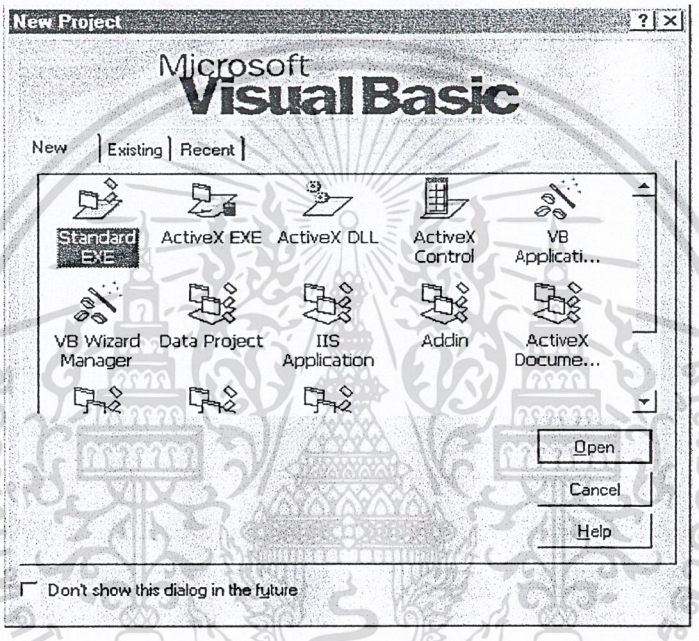
2.3 การเขียนโปรแกรมด้วย Visual Basic

2.3.1 การสร้างแอพพลิเคชันด้วย Visual Basic

- **รูปแบบการสร้างแอพพลิเคชันด้วย Visual Basic**

Visual Basic นั้นสร้างแอพพลิเคชันได้หลายรูปแบบและสร้างได้เร็ว เพราะมีขั้นตอนที่ไม่ยุ่งยากนัก แม้จะไม่เคยมีประสบการณ์ด้านการสร้างซอฟร์แวร์แอพพลิเคชันมาก่อนก็เรียนรู้ได้ไม่ยาก

เมื่อเราเปิด Visual Basic ขึ้นมา เราจะพบกับไดอะล็อกบ็อกซ์ New project ซึ่งแสดงประเภทของแอพพลิเคชันที่สร้างได้

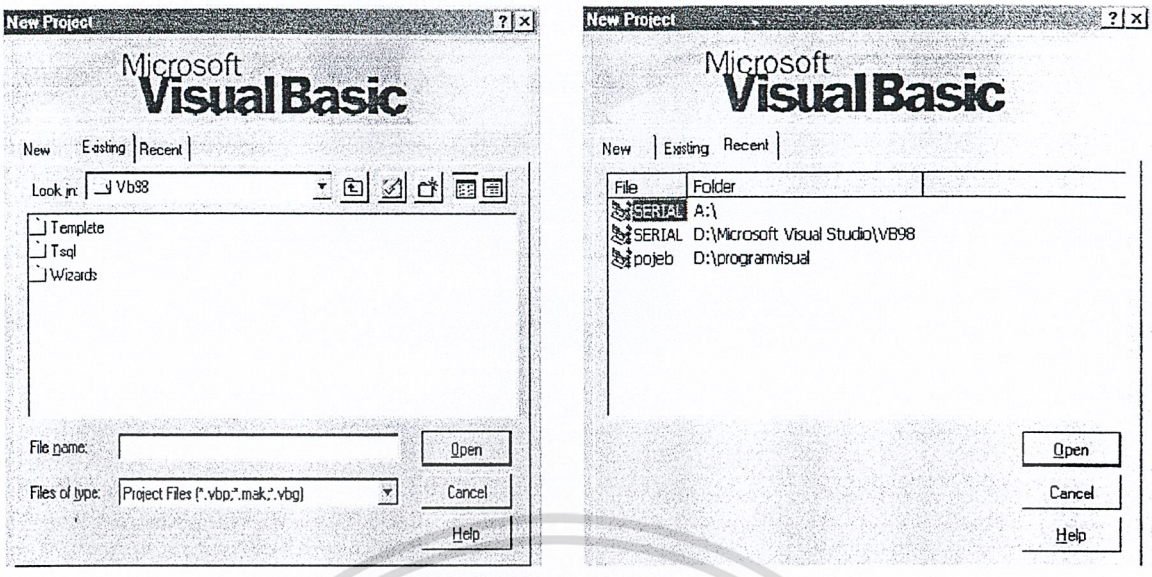


รูปที่ 2.22 ไดอะล็อก New Project

จะสังเกตว่าในไดอะล็อกบ็อก New project ยังมีอีก 2 แท็บซึ่งมีความหมายดังนี้

- แท็บ Exiting แสดงโปรเจ็กต์ที่เคยมีการสร้างมาก่อน ซึ่งจะช่วยให้เราไม่ต้องเสียเวลาค้นหาโฟลเดอร์ที่เก็บโปรเจ็กต์เดิม ๆ

- แท็บ Recent แสดงโปรเจ็กต์ที่เคยมีการสร้างมาก่อน และถูกเรียกมาแก้ไขล่าสุด

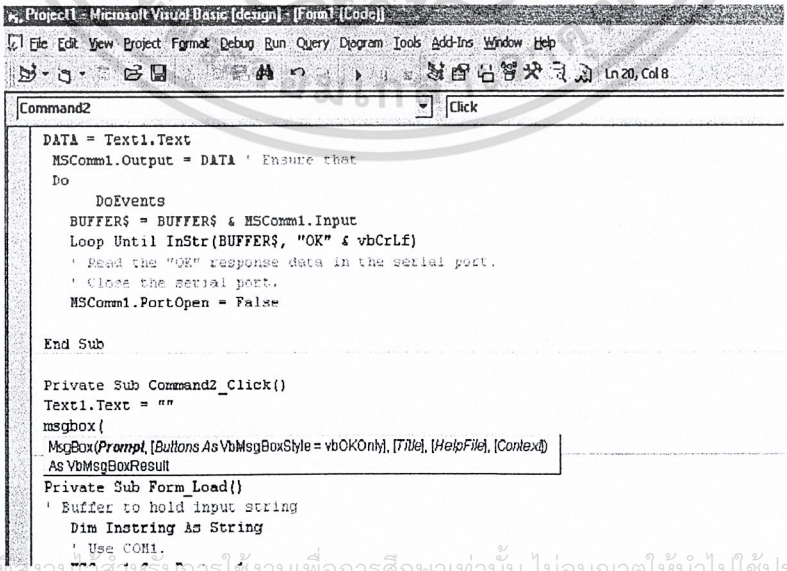


รูปที่ 2.23 แท็บ Existing และ Recent

- **การเขียนโปรแกรมแบบ Event - Drivent**

ในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชันจาก Visual Basic นั้น จะมีวิธีการที่แตกต่างจากการเขียนโปรแกรมโครงสร้างทั่ว ๆ ไปที่เราอาจจะเคยได้เรียนรู้ เช่น จากภาษา Pascal, C นั่นเพราะการเขียนโปรแกรมกับ Visual Basic จะใช้การเขียนแบบ Event – Drivent

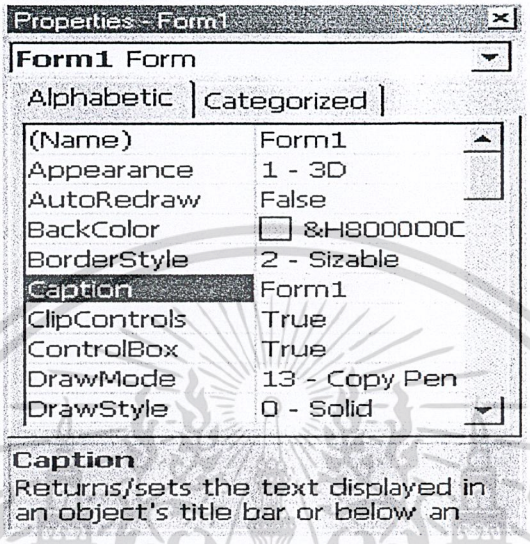
Event – Drivent จริง ๆ ก็คือการเขียนโปรแกรมในลักษณะว่า เหตุการณ์นี้เกิดขึ้น เราจะดำเนินการอย่างไร หรือพูดอีก อย่างว่า เมื่อมีเหตุการณ์ต่างๆ ดำเนินไป เราต้องคิดว่าจะจัดการกับเหตุการณ์ที่เกิดขึ้นอย่างไร เมื่อคิดออกเราก็มาเขียนโปรแกรมรองรับเหตุการณ์ต่างๆ เหล่านั้น Visual Basic นั้นสนับสนุนการเขียนโปรแกรมแบบ Event – Drivent โดยมีเครื่องมือที่ช่วยให้เราจัดการกับเหตุการณ์ต่าง ๆ ที่น่าจะเกิดกับแอพพลิเคชันของเราได้สะดวก



รูปที่ 2.24 ออบเจ็กต์ของ Visual Basic

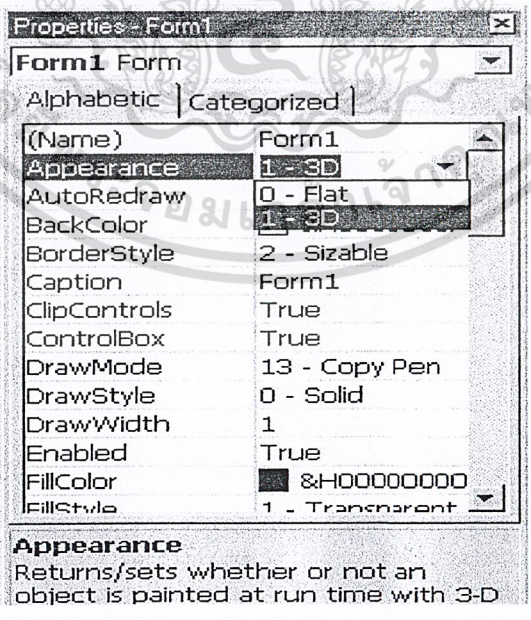
- **การกำหนดพร็อพเพอร์ตี้ให้กับออบเจ็กต์**

จะเห็นได้ว่าแต่ละออบเจ็กต์ จะมีลักษณะเฉพาะที่แตกต่างจากออบเจ็กต์อื่นๆ เช่นปุ่มกด ก็มีข้อความ(caption) ที่แตกต่างกัน เช่น " เปลี่ยนรูป " , " จบการทำงาน " ซึ่งเราเรียกลักษณะเฉพาะตัวนี้ว่า " พร็อพเพอร์ตี้ " (Property)



รูปที่ 2.25 แสดงพร็อพเพอร์ตี้ของฟอร์ม

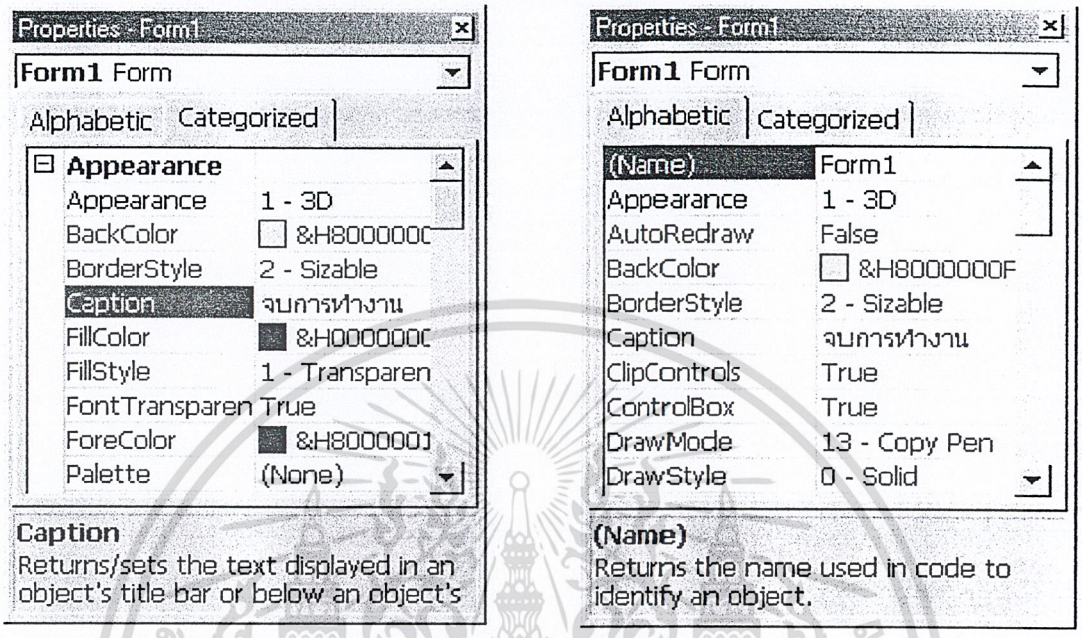
ในขั้นตอนของการสร้างแอพพลิเคชั่นเราสามารถกำหนดค่าพร็อพเพอร์ตี้ได้จาก Property window



รูปที่ 2.26 พร็อพเพอร์ตี้ Appearance ของ image

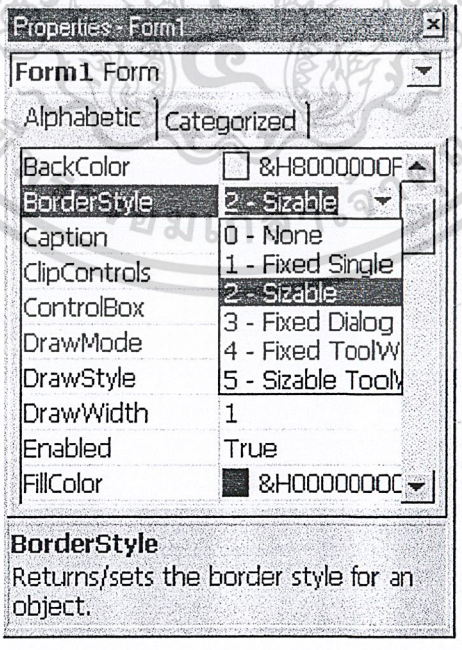
เราจะเห็นว่าค่าของพร็อพเพอร์ตี้ของออบเจ็กต์ต่าง ๆ ใน properties window นั้นเราสามารถกำหนดได้ 3 รูปแบบ

1. กำหนดค่าอย่างอิสระ เช่น พร็อพเพอร์ตี้ Caption พร็อพเพอร์ตี้ Name



รูปที่ 2.27 กำหนดค่าพร็อพเพอร์ตี้ อย่างอิสระ

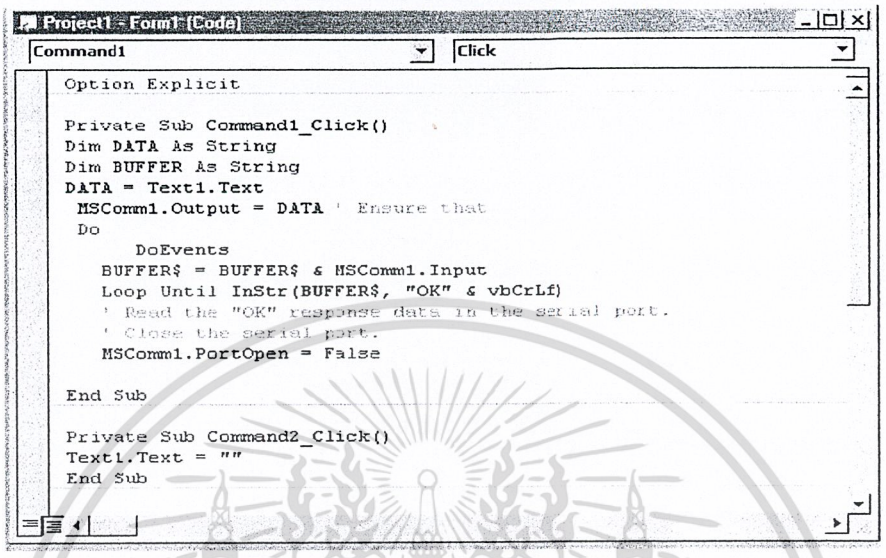
2. กำหนดค่าจากตัวเลือกที่มีใน properties window เช่น Border Style , Backcolor



รูปที่ 2.28 กำหนดค่าพร็อพเพอร์ตี้จากตัวเลือก

- **การเรียกใช้เมธอดของออบเจ็กต์**

นอกเหนือจากพร็อพเพอร์ตี้ที่ใช้ในการบอกความแตกต่างของออบเจ็กต์ต่างๆ แล้ว ยังมีอีกสิ่งหนึ่งที่ออบเจ็กต์ มักจะต้องมีนั่นคือ ความสามารถของออบเจ็กต์ ซึ่งเราจะเรียกว่า " เมธอด " (Method)

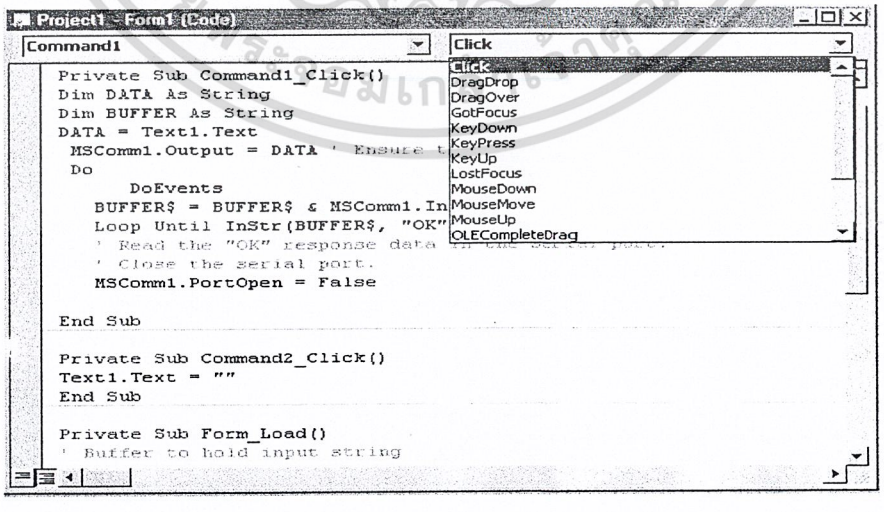


รูปที่ 2.29 ตัวอย่างการเรียกใช้เมธอด

เมธอดนั้นเป็นความสามารถที่ออบเจ็กต์มีอยู่ ซึ่งจะแตกต่างกันในแต่ละออบเจ็กต์ ซึ่งการเรียกใช้งานผ่านการเขียนโปรแกรมเท่านั้น

- **จัดการกับเหตุการณ์ที่เกิดขึ้น**

ออบเจ็กต์แต่ละตัวก็จะมีอีเวนต์ที่สามารถเกิดขึ้นได้จำนวนหนึ่ง ซึ่งเราจะดูได้ว่าจะมีอีเวนต์จาก Code Window



รูปที่ 2.30 แสดงการเลือก code window

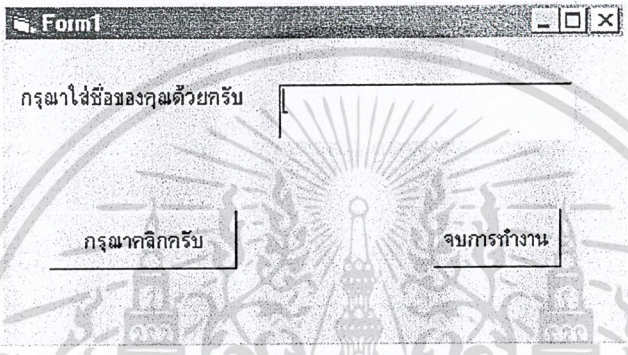
เมื่อเรารู้ว่าจะมีอะไรเกิดขึ้นบ้าง เราก็สามารถจะเขียนคำสั่งของ Visual Basic เพื่อการจัดการกับอีเวนต์ต่างๆ ได้อย่างครบถ้วน ซึ่งมักจะเรียกการจัดการอีเวนต์ที่เกิดขึ้นนี้ว่า อีเวนต์แฮนเดอร์ (Event Hander)

- **เริ่มสร้างแอพพลิเคชันด้วย Visual Basic**

สำหรับขั้นตอนการสร้างแอพพลิเคชั่นนั้นมีขั้นตอนที่ควรจะต้องทำดังนี้

ขั้นที่1 : ออกแบบแอพพลิเคชัน

ก่อนจะสร้างแอพพลิเคชั่น หรือการเขียนโปรแกรมนั้นสิ่งแรกที่ ต้องทำคือ ต้องทราบให้แน่ชัดก่อนว่าแอพพลิเคชั่นที่เราจะสร้างนั้นจะใช้ประโยชน์อะไร ต้องมีความสามารถอะไรบ้าง ต้องการให้มีรูปร่างหน้าตาเป็นอย่างไร ซึ่งจำเป็นอย่างยิ่งที่ต้องคิดให้รอบคอบ และเขียนออกมาให้ชัดเจน

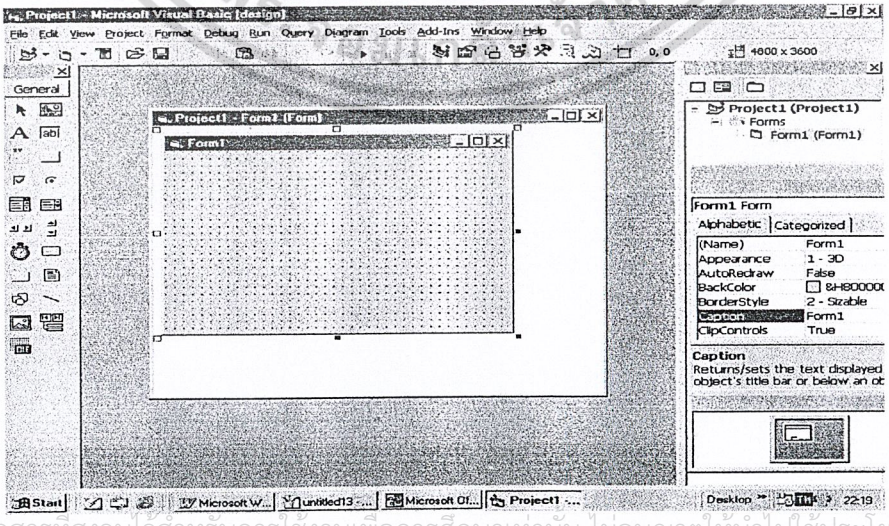


รูปที่ 2.31 แสดงหน้าตาของแอพพลิเคชัน

ขั้นที่ 2 : ตกแต่งหน้าตาแอพพลิเคชัน

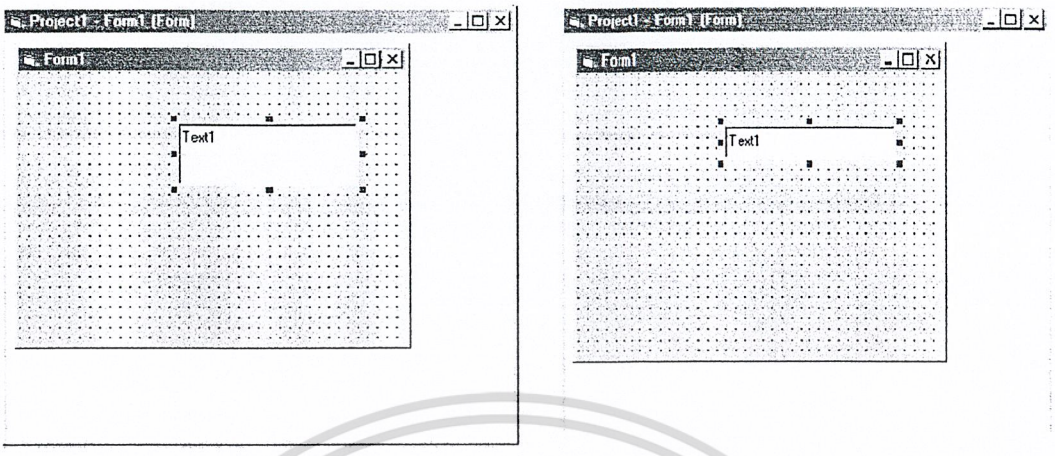
สำหรับขั้นตอนนี้จะเป็นการตกแต่งรูปร่างของแอพพลิเคชันตามที่ได้ออกแบบไว้ พร้อมๆ กับกำหนดค่าพร็อพเพอร์ตี้ต่างๆ ให้กับคอนโทรลแต่ละตัวในแอพพลิเคชัน โดยมีขั้นตอนย่อยๆ ดังนี้

1. เรียกการใช้งาน Visual Basic
2. ดับเบิ้ลคลิกที่คอนโทรลภายใน ToolBox คอนโทรลนั้นจะปรากฏฟอร์มดีไซเนอร์



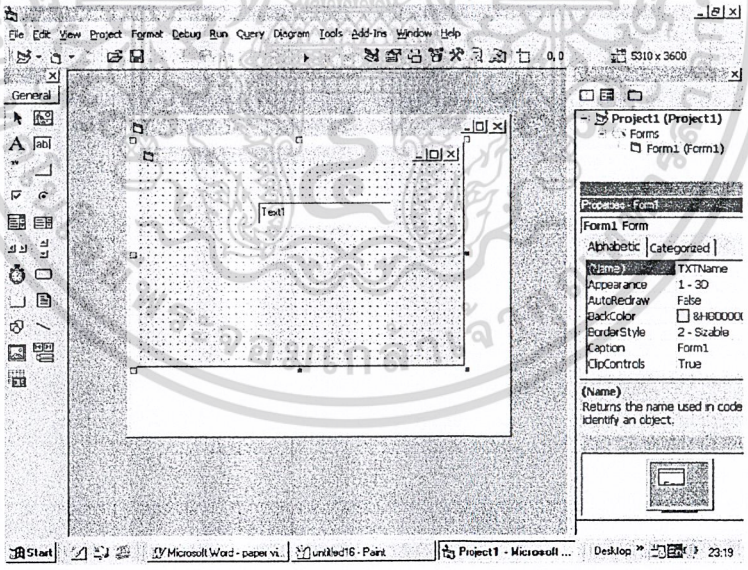
รูปที่ 2.32 แสดงการเลือกคอนโทรลจาก Toolbox

3. จัดวางตำแหน่ง และปรับขนาด ของคอนโทรลให้ได้ตามที่ต้องการ



รูปที่ 2.33 แสดงการปรับแต่งคอนโทรล

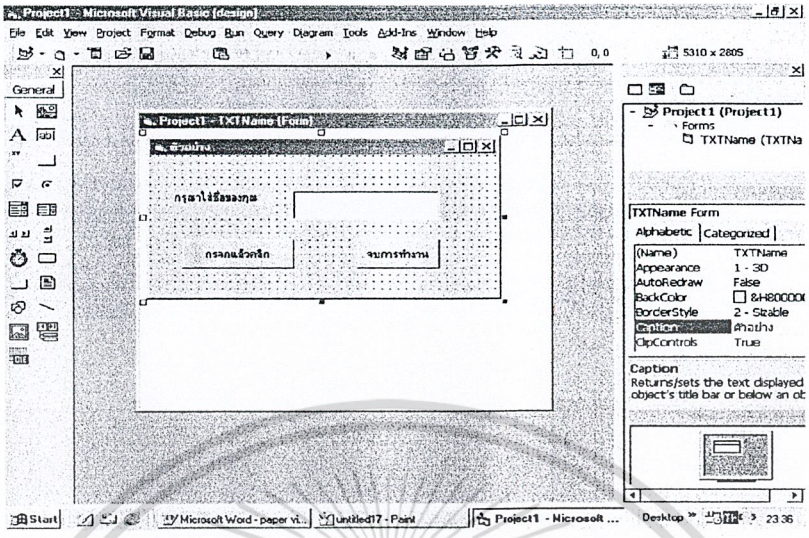
4. กำหนดค่าให้กับพร็อพเพอร์ตี้ของคอนโทรล โดยคลิกที่คอนโทรลที่ต้องการ แล้วเลือกพร็อพเพอร์ตี้ที่ต้องการกำหนดใน Poperties Widow



รูปที่ 2.34 การกำหนดพร็อพเพอร์ตี้ให้กับคอนโทรล

5. นำคอนโทรลต่างๆ เข้ามาในฟอร์มดีไซเนอร์ แล้วปรับหน้าตา จัดวางตำแหน่ง แล้วกำหนดค่าให้พร็อพเพอร์ตี้ของคอนโทรลต่างๆ ดังนี้

6. เมื่อปรับแต่งคอนโทรลต่างๆ เรียบร้อยแล้วจะได้ผลดังนี้

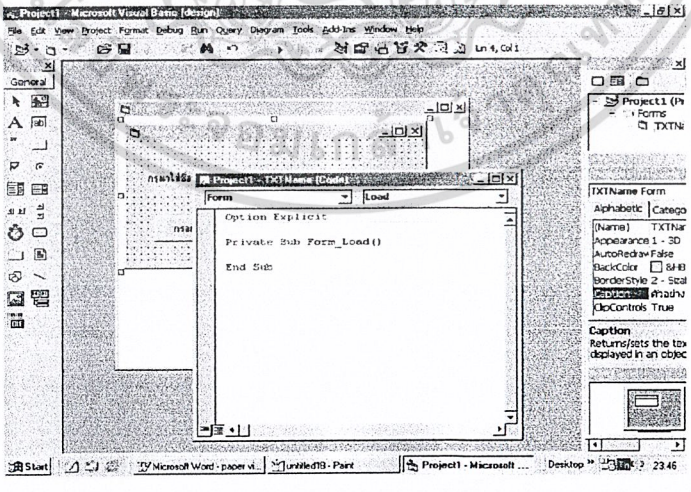


รูปที่ 2.35 ดับเบิลคลิกเลือกคอนโทรล

ขั้นที่ 3 : เขียนโค้ดกำกับการทำงานของแอพพลิเคชั่น

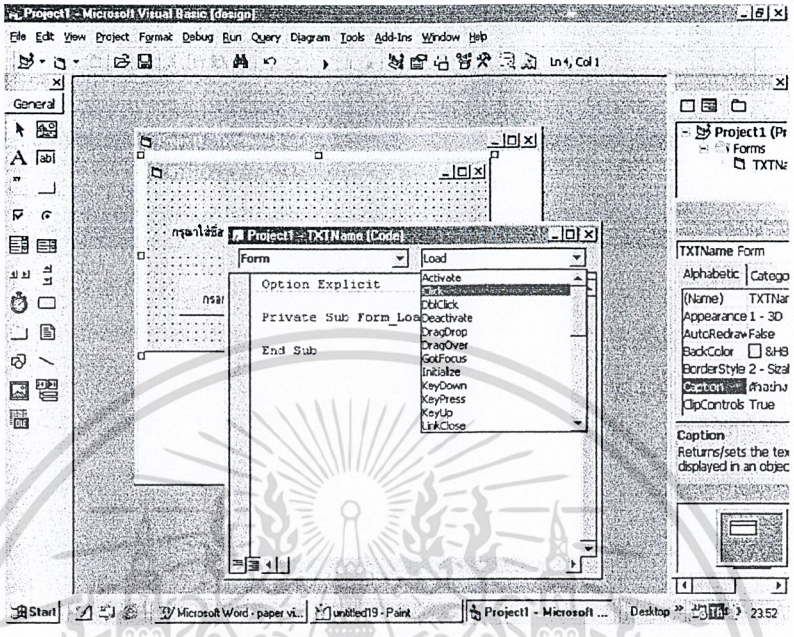
หลังจากตกแต่งหน้าตาเสร็จแล้ว ขั้นต่อไปคือ การเขียนโค้ดหรือเขียนโปรแกรมเพื่อควบคุมการทำงานต่างๆ ซึ่งเราจะใช้การเขียนโปรแกรมแบบ Event Drivent Programing ซึ่งจะเป็นการเขียนโค้ดเพื่อรองรับกับเหตุการณ์ต่างๆ ในแอพพลิเคชันของเรา ซึ่งมีขั้นตอนดังนี้

1. ดับเบิลคลิกที่คอนโทรลที่ต้องการควบคุมซึ่งจะปรากฏ Code window



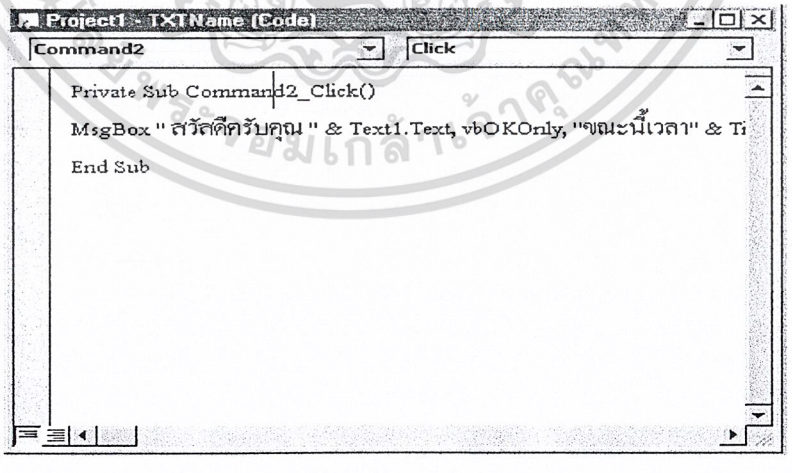
รูปที่ 2.36 การดับเบิลคลิกเพื่อที่จะเขียนโค้ดกำกับการทำงาน

2. ให้เลือกเหตุการณ์ที่เราต้องการจะควบคุม จากลิสต์บ็อกซ์ของ Code Window ในที่นี้เราสนใจตอนที่ผู้ใช้งานคลิกปุ่ม จึงเลือกเหตุการณ์ Click



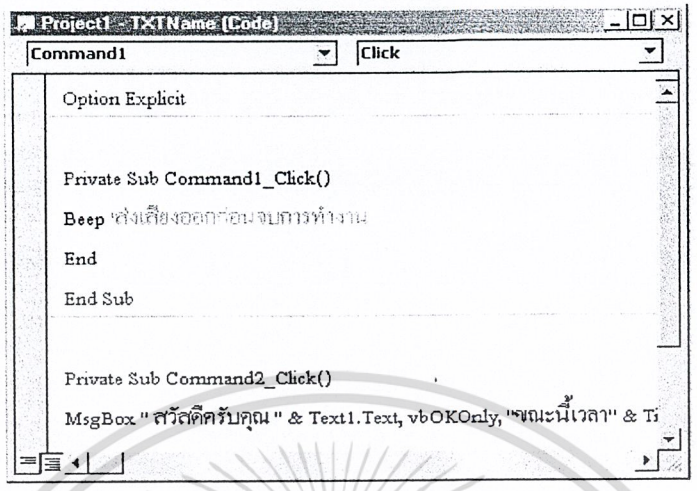
รูปที่ 2.37 การเลือกเหตุการณ์ที่จะควบคุม

3. เขียนโค้ดในภาษา Basic เพื่อจัดการณ์นั้นๆ ในที่นี้เราจะนำเอาชื่อที่ผู้ใช้กรอกเอาไว้มาแสดงพร้อมกับวันเวลา ณ ขณะที่คลิกปุ่มนั้น



รูปที่ 2.38 การเขียนโค้ดกำกับเหตุการณ์

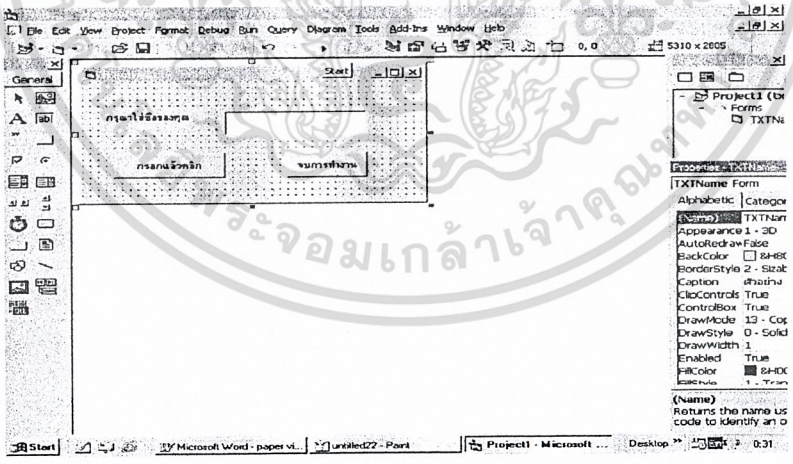
4. เลือกคอนโทรลตัวอื่น มาเขียน โดยแต่ละคอนโทรลก็จะมีเหตุการณ์ที่เราจะต้องจัดการไม่เหมือนกัน



รูปที่ 2.39 การเขียนโค้ดให้กับคอนโทรลตัวอื่น

ขั้นที่ 4 : ทดสอบการทำงานของแอพพลิเคชัน

เมื่อเขียนโค้ดเสร็จแล้ว ก็ถึงเวลาที่จะทดสอบการทำงานของแอพพลิเคชันที่เราสร้างขึ้นซึ่งประกอบไปด้วยคอนโทรลต่างๆ ที่ปรับแต่งไว้ และโค้ดที่เขียนเพื่อจัดการกับเหตุการณ์ต่างๆ โดยเราจะทดสอบการทำงานโดยการกด <f5>



รูปที่ 2.40 การเริ่มคำสั่งรัน

ขั้นที่ 5 : บันทึกเก็บไว้ในคอมพิวเตอร์

หลังจากทดสอบจนแน่ใจแล้วว่าแอพพลิเคชันที่สร้างถูกต้อง เราจึงบันทึกเก็บไว้ซึ่งสามารถแก้ไขและเพิ่มเติมความสามารถอื่นๆ ได้

2.3.2 การใช้งานตัวแปรในการเขียนโปรแกรม

- **ตัวแปรและชนิดของข้อมูล (Variable and Data Type)**

ในการเขียนโปรแกรมเพื่อสร้างแอพพลิเคชันนั้น สิ่งหนึ่งที่เรามักจะใช้งานอยู่บ่อยๆ ก็คือตัวแปร (Variable) ซึ่งเราจะใช้เก็บข้อมูลชั่วคราว หรือนำมาใช้เก็บข้อมูลเพื่อทำการคำนวณ ซึ่งอาจเรียกว่า เราใช้ตัวแปรในการขับเคลื่อนให้แอพพลิเคชันทำงานต่อไปได้

เนื่องจากเราได้ใช้ตัวแปรในการเก็บข้อมูล เราจึงน่าจะรู้จักกับชนิดของข้อมูลต่างๆ ที่สามารถนำมาใช้ได้รายละเอียดดังนี้

ชนิด	คำอธิบาย	ขนาดของหน่วยความจำ
Byte	เป็นขนาดข้อมูลจำนวนเต็มตั้งแต่ 0 ถึง 255	1 ไบต์
Boolean	เป็นข้อมูลทางตรรกะ : จริง (True),เท็จ (False)	2 ไบต์
Integer	เป็นจำนวนเต็มระหว่าง –32,768 ถึง 32,767	2 ไบต์
Long	เป็นจำนวนเต็มระหว่าง –2,147,483,648 ถึง 2,147,483,647	4 ไบต์
Single	เป็นเลขทศนิยมระหว่าง –3.402823E38 ถึง –1.401298E–45 สำหรับค่าลบ และ 1.401298E-45 ถึง 3.402823E38 สำหรับค่าบวก	4 ไบต์
Double	เป็นเลขทศนิยมระหว่าง –1.7976313486232E308 ถึง –4.940656454841247E-324 สำหรับค่าลบ และ 4.94065645841247E-324 ถึง 1.79769313486232E308	8 ไบต์
Currency	เป็นเลขที่มีค่าตั้งแต่ –922,337,203,685,477.5808 ถึง 922,337,203,685,477.5807	8 ไบต์
Date	เป็นวันตั้งแต่ 1 มกราคม ค.ศ.100 ถึง 31 ธันวาคม ค.ศ. 9999	8 ไบต์
Object	เป็นข้อมูลที่อ้างอิงออบเจ็กต์ จึงเก็บแอดเดรสของออบเจ็กต์ไว้	4 ไบต์
String	เก็บสตริง หรือข้อความที่เรียงต่อกัน	64 KB หรือ 2 MB
Variant(ที่มีตัวเลข)	เป็นข้อมูลชนิดพิเศษที่เก็บค่าได้ทุกแบบ (รวมไปถึงค่าพิเศษต่างๆ เช่น EMPTY, NULLเป็นต้น)	16 ไบต์

ตารางที่ 2.13 ชนิดของข้อมูลต่างๆ ที่นำมาใช้งาน

จากตารางจะเห็นว่าชนิดของข้อมูล Variant เป็นข้อมูลที่สามารถแทนชนิดข้อมูลชนิดอื่นๆ ได้ทุกแบบ แต่ก็เป็นแบบใช้เนื้อที่เปลืองที่สุด

- **การประกาศค่าตัวแปร(Variable Declartion)**

การที่เราจะใช้งานตัวแปรในการเขียนโปรแกรม จะเริ่มจากการประกาศตัวแปร (variable declaration) ซึ่งก็คือการตั้งชื่อตัวแปร และการกำหนดชนิดของข้อมูลให้กับตัวแปรแต่ละตัว

ในการประกาศตัวแปร เราจะใช้คำสั่ง Dim (ย่อมาจาก Dimension) ซึ่งมีรูปแบบดังนี้

DIM ชื่อตัวแปร **AS** ชนิดของตัวแปร, ชื่อของตัวแปร **AS** ชนิดตัวแปร,...

ปกติเรามักจะเลือกชนิดข้อมูลให้เหมาะสมกับงาน แต่ถ้าเราไม่ได้เลือกชนิดของข้อมูลไว้ก่อนตัว Visual Basic เองจะตัดสินให้เองว่าควรจะเก็บข้อมูลที่เป็น Varaint แบบใด โดยการพิจารณาจากข้อมูลที่จะเก็บ หรือยึดสภาวะรอบข้างในการตัดสินใจ ว่าจะเป็นข้อมูลแบบใด

ค่าคงที่(Constant)

ในการประกาศค่าคงที่เราต้องใช้ข้อมูลที่มีค่าไม่เปลี่ยนแปลงตลอด เราจะใช้กำหนดให้เป็นค่าคงที่ ซึ่งการกำหนดให้เป็นค่าคงที่ ช่วยให้เราสะดวกในการเขียนโปรแกรมเพราะจดจำง่าย เป็นชื่อที่ความหมายแล้วเข้าใจได้ดี

สำหรับค่าคงที่ที่ใช้งานใน Visual Basic นั้นแบ่งได้ 2 ประเภท ได้แก่

- ค่าคงที่แบบ User Defined
- ค่าคงที่แบบ Pre Defined

ค่าคงที่แบบ User Defined

เป็นค่าคงที่ที่เราเป็นผู้กำหนดเอง เป็นค่าที่ใช้เฉพาะในแอพพลิเคชันของเรา ซึ่งมีรูปแบบดังนี้

CONST ชื่อค่าคงที่ **AS** ชนิดข้อมูล = ค่าคงที่นั้น

2.3.3 การตัดสินใจในการเขียนโปรแกรม

การเขียนโปรแกรมบ่อยครั้งที่เราต้องการทางเลือกทำงาน อย่างใดอย่างหนึ่ง ซึ่งเรียกว่าการตัดสินใจ โดยลักษณะการตัดสินใจในการเขียนโปรแกรมนั้นมีหลายรูปแบบ แล้วแต่ต้องการในการกำหนดทิศทางการทำงานของแอพพลิเคชั่นนั้นเป็นอย่างไร

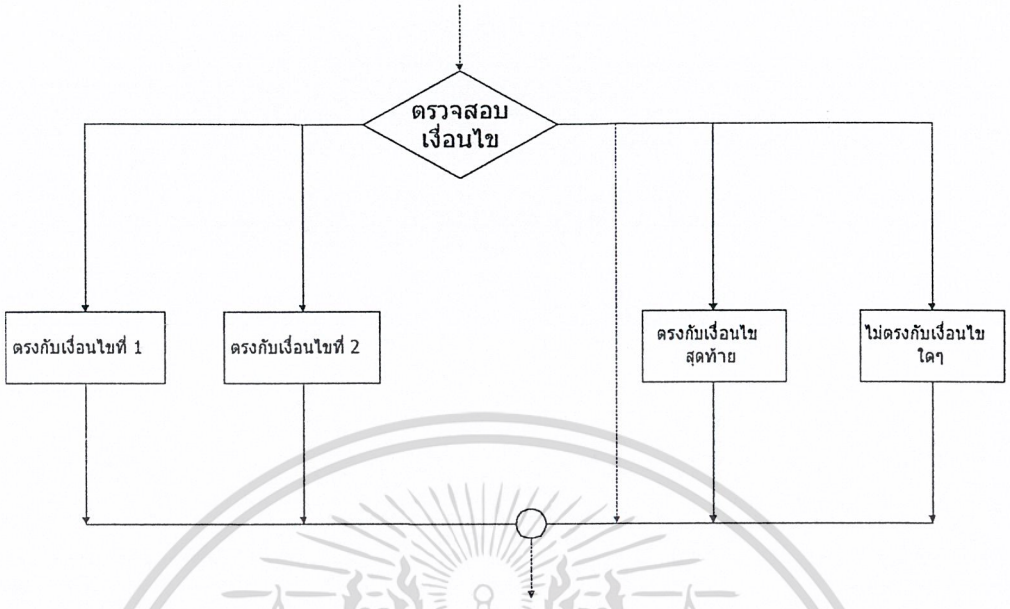
รูปแบบการตัดสินใจ

ในการเขียนโปรแกรม บางครั้งจำเป็นต้องตัดสินใจให้แอพพลิเคชันทำงานอย่างใดอย่างหนึ่ง จากทางเลือกมากกว่า 1 ทางเลือก โดยการตัดสินใจในโปรแกรม จะมี 2 ประเภท

1. ตัดสินใจเลือกจากทางเลือก 2 ทางเลือก
2. ตัดสินใจเลือกจากทางเลือกมากกว่า 2 ทางเลือก

If...Then...Else : ตัดสินใจเลือกจากทางเลือก 2 ทางเลือก

เราจะใช้งาน If..Then...Else ในการตัดสินใจเมื่อมีทางเลือก 2 ทางเลือก(yes/no) ซึ่งในการใช้งานมีการตรวจสอบเงื่อนไขก่อนว่าจริงหรือเท็จ แล้วค่อยเลือกว่าจะไปทางใด



รูปที่ 2.41 ไดอะแกรม ของ If...Then...Else

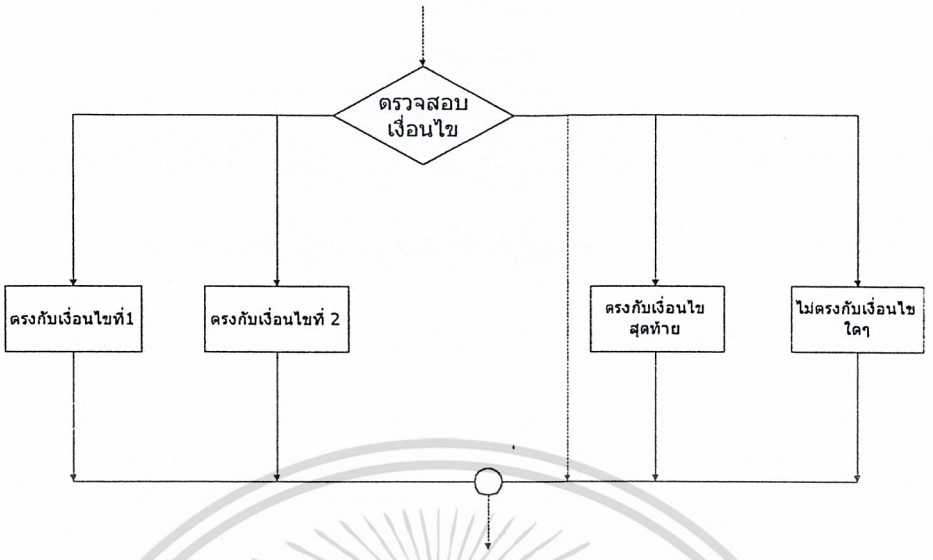
ในการใช้งานนั้นมีรูปแบบดังนี้

```
If ( ทดสอบเงื่อนไขว่าจริง หรือเท็จ ) Then
        ถ้าเงื่อนไขเป็นจริงให้ทำงานหลังคำว่า Then
    Else
        ถ้าเป็นเท็จให้ทำงานหลังคำว่า Else
End If
```

Select...Case: ตัดสินใจเลือกมากกว่า 2 ทางเลือก

การใช้งาน If...Then...Else If เพื่อตัดสินใจเลือกจากตัวเลือกมากกว่า 2 ตัวเลือก คงเป็นวิธีการที่ไม่เหมาะสมดังนั้นจึงควรจะมีวิธีการเฉพาะจะดีกว่า

ใน Visual basic นั้นการตัดสินใจเลือกจากทางเลือกมากกว่า 2 ทางเลือกจะใช้ Select...Case ซึ่งจะมีการตรวจสอบเงื่อนไขในการเลือกก่อนว่าตรงกับตัวเลือกใด แล้วจึงทำงานตามคำสั่งที่ต่อท้ายตัวเลือกนั้นซึ่งเขียนเป็นโฟลว์ชาร์ต ได้ดังนี้



รูปที่ 2.42 ไดอะแกรม ของ Select...Case

ในการใช้งานนั้นมีรูปดังนี้

```
Select case < ทดสอบเงื่อนไข >
        Case เงื่อนไขแรก : < ทำตามเงื่อนไขแรก >
        Case เงื่อนไขที่สอง : < ทำตามเงื่อนไขที่สอง >
        Case สุดท้าย : < ทำตามเงื่อนไขสุดท้าย >
        Case Else < เมื่อไม่ตรงกับเงื่อนไขใดๆ ทำงานหลังคำว่า Else >
End Select
```

จากรูปแบบจะเห็นว่า ถ้าทดสอบเงื่อนไขแล้วตรงกับเงื่อนไขใดก็จะทำตามชุดคำสั่งที่อยู่หลังเงื่อนไขนั้นๆ แต่ไม่ตรงกับเงื่อนไขใด ก็จะมีตัวเลือกเผื่อเอาไว้เช่นกัน

2.3.4 การทำงานแบบวนซ้ำ

หลังจากที่เราได้ใช้การตัดสินใจในรูปแบบต่างๆ เพื่อเปลี่ยนทิศทางการทำงานของแอพพลิเคชั่น แล้วเราจะมาเรียนรู้อีกลักษณะการควบคุมการทำงานของแอพพลิเคชั่นนั่นคือ การทำงานแบบวนซ้ำ

- **รูปแบบการทำงานแบบวนซ้ำ**

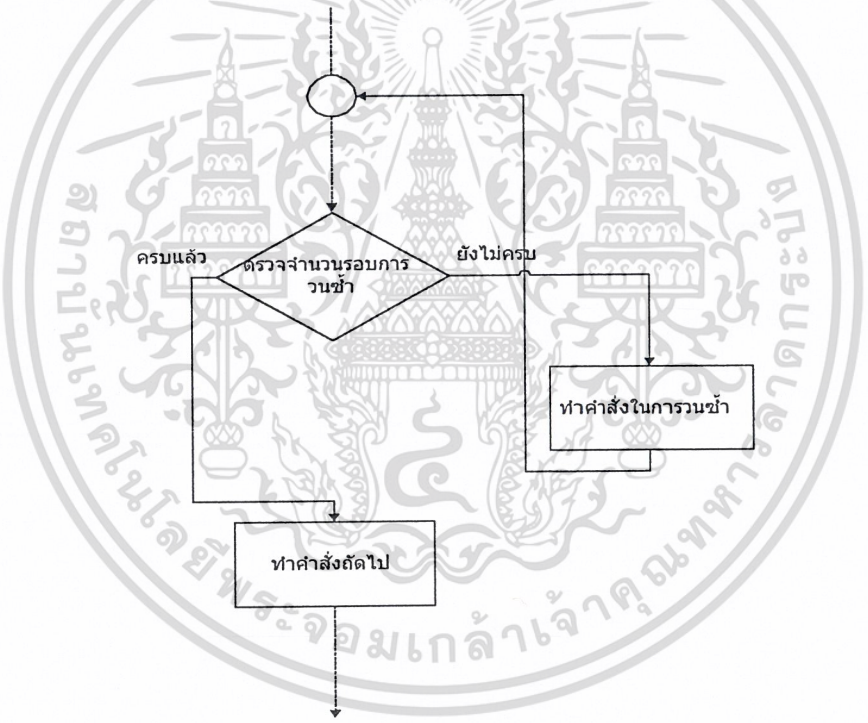
ในการเขียนโปรแกรม เราอาจจำเป็นต้องสั่งให้แอพพลิเคชันทำงานแบบวนซ้ำตามจำนวนครั้งที่เราต้องการได้ ซึ่งรูปแบบการวนซ้ำมี 2 แบบ ได้แก่

1. การวนซ้ำด้วยจำนวนรอบที่นอน
2. การวนซ้ำด้วยจำนวนรอบที่ไม่แน่นอน

การทำงานแบบวนรอบนี้ช่วยเพิ่มความสะดวกให้แก่เราเพราะไม่ต้องเขียนโปรแกรมที่ซ้ำซ้อน เพราะใช้การเลือกรูปแบบการวนซ้ำที่เหมาะสมก็สามารถเขียนโปรแกรมที่สั้น และทำงานได้อย่างมีประสิทธิภาพได้

- **การวนซ้ำด้วยจำนวนรอบที่นอน**

สำหรับการวนซ้ำด้วยจำนวนรอบที่แน่นอนเราจะใช้ For...Next ในการใช้งานวนซ้ำ ซึ่งจะต้องใช้ตัวแปร 1 ตัวที่ใช้นับจำนวนรอบว่า วนซ้ำครบตามที่กำหนดหรือไม่



รูปที่ 2.43 ไดอะแกรม ของ For...Next

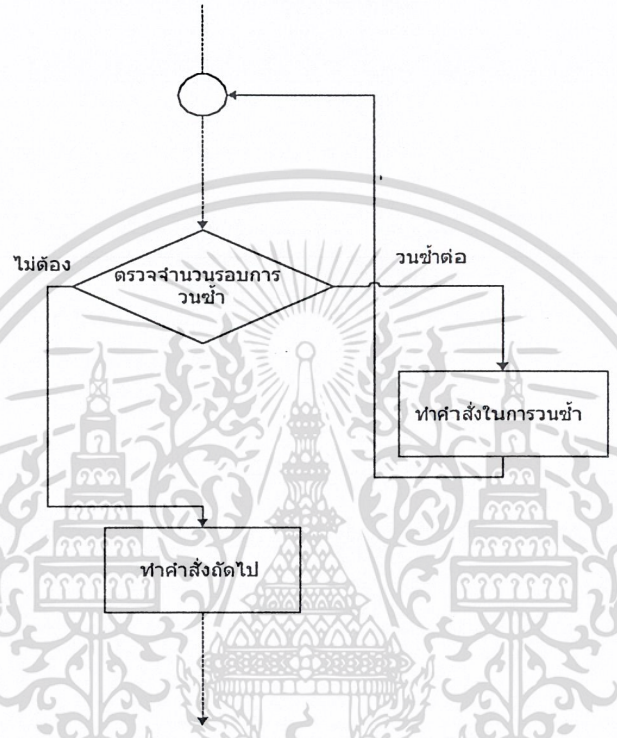
ในการใช้งาน For...Next นั้นมีรูปแบบดังนี้

For ตัวแปรใช้นับจำนวนรอบ = จำนวนรอบเริ่มต้น **To** จำนวนรอบสุดท้าย[step ขั้นของการนับ]
< ทำงานตามคำสั่ง >
Next ตัวแปรที่ใช้วัดจำนวนรอบ

- **การวนซ้ำด้วยจำนวนรอบที่ไม่แน่นอน**

การวนซ้ำด้วยการใช้ While...Wend

while...wend เป็นรูปแบบการวนซ้ำที่ไม่สามารถบอกจำนวนรอบที่แน่นอนของการวนซ้ำได้ ซึ่งจะทำการวนซ้ำถึงเมื่อใดนั้น จะใช้การตรวจสอบเงื่อนไขก่อนการวนซ้ำว่าต้องการวนซ้ำอีกหรือไม่ ถ้าเงื่อนไขยังคงเป็นจริงอยู่ก็จะทำการวนซ้ำต่อ แต่ถ้าเงื่อนไขเป็นเท็จก็จะหลุดจากการวนซ้ำ



รูปที่ 2.44 ไดอะแกรม ของ While...Wend

ในการใช้งาน While...Wend นั้นมีรูปแบบดังนี้

```
While < ทดสอบเงื่อนไข จริงหรือเท็จ >
    < ถ้าเงื่อนไขเป็นจริง ให้ทำงานตามคำสั่ง >
Wend
```

วนซ้ำด้วยการใช้ Do...Loop

นอกเหนือจากการใช้ While...Wend ในการวนซ้ำแล้ว เรายังใช้การวนซ้ำแบบตรวจสอบเงื่อนไขได้อีกหลายแบบได้แก่

```
Do While        < ทดสอบเงื่อนไข จริงหรือเท็จ >
                < ถ้าเงื่อนไขเป็นจริง ให้ทำงานตามคำสั่ง >
Loop
```

หรือ

```
Do
                < ทำงานตามคำสั่ง >
Loop While      < ทดสอบเงื่อนไข จริงหรือเท็จ ถ้าเป็นจริงให้กลับไปทำงานอีกรอบ >
```

และ

```
Do Until        < ทดสอบเงื่อนไข จริงหรือเท็จ >
                < ถ้าเงื่อนไข จริงหรือเท็จ ให้ทำงานตามคำสั่ง >
```

จะเห็นว่าเราสามารถตรวจสอบเงื่อนไขได้ 2 แบบ

While จะหลุดจากการวนซ้ำได้เมื่อทดสอบแล้วเป็นเท็จ

Until จะหลุดจากการวนซ้ำได้เมื่อทดสอบเงื่อนไขแล้วเป็นจริง

2.3.5 การลดความซ้ำซ้อนด้วยโปรแกรมย่อย

- **โปรแกรมย่อยใน Visual Basic**

ในการเขียนโปรแกรม ย่อมเกิดการทำงานที่ซ้ำๆ กัน เราสามารถลดการทำงานซ้ำๆ นั้นการเขียนโปรแกรมย่อยเก็บไว้ เมื่อเราต้องการทำงานแบบเดิมอีก เราก็เพียงแต่เรียกโปรแกรมย่อยนั้นมาใช้งาน ทำให้ลดความยุ่งยากของโปรแกรมที่เขียนด้วย

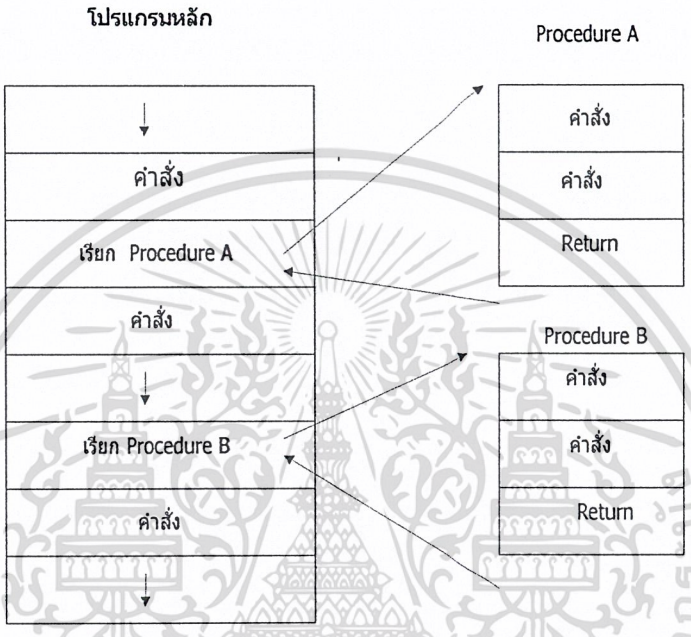
โปรแกรมย่อยใน Visual Basic มีอยู่ 2 ประเภท

- Procedure โพรซีเจอร์ เป็นโปรแกรมย่อยที่เมื่อจบการทำงานแล้ว จะไม่มีการส่งผลการทำงานกลับมาให้เราทราบ เราจึงมักเขียนโพรซีเจอร์ที่ต้องการแต่ผลของงาน แต่ไม่ต้องการเน้นผลการทำงานกลับมาให้

- Function ฟังก์ชันเป็นโปรแกรมย่อยที่เมื่อจบการทำงานแล้ว จะต้องส่งผ่านการทำงานกลับมาให้ผู้ที่ใช้งาน ฟังก์ชันนั้นบางครั้งเราอาจเรียกโพรซีเจอร์ว่า ซับรูทีน(Sub Routine) ก็ได้ ซึ่งก็หมายถึง โปรแกรมย่อยเหมือนกัน ซึ่งต้องการการส่งข้อมูลกลับมายังผู้เรียก

- **ใช้งานโปรแกรมย่อยชนิด Procedure**

โพรซีเจอร์นั้น เป็นโปรแกรมย่อยที่เราเขียนขึ้นมาเพื่อทำหน้าที่อย่างใดอย่างหนึ่ง โดยที่โพรซีเจอร์นั้นจะถูกเรียกใช้งานตามจุดต่างๆ ของโปรแกรม และเมื่อมันทำงานเสร็จแล้วจะไม่มีการคืนค่าใดๆ กลับมายังผู้เรียกใช้งาน



รูปที่ 2.45 ไดอะแกรม การใช้งานของโพรซีเจอร์

โพรซีเจอร์ที่เราจะสร้างขึ้นมีรูปแบบการใช้งานดังนี้

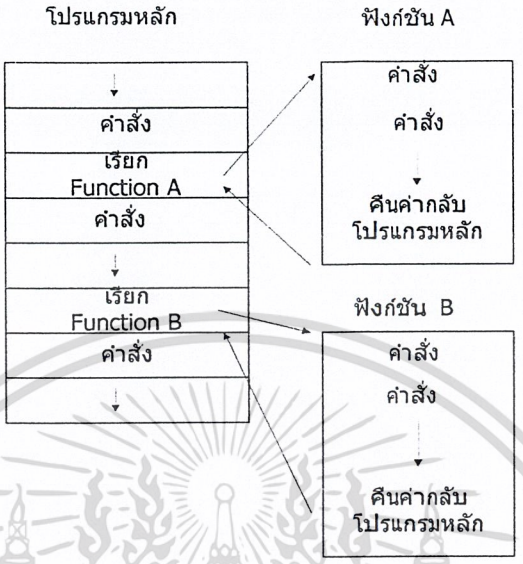
Sub ชื่อโพรซีเจอร์ (รายการพารามิเตอร์ จะมีหรือไม่มีก็ได้)

< คำสั่งใน Visual basic >

End sub

- **การใช้งานโปรแกรมย่อยชนิด Function**

ฟังก์ชันเป็นโปรแกรมย่อยที่จะต้องคืนค่ากลับมาหาผู้เรียกใช้ หลังจากโปรแกรมย่อยทำงานเสร็จ



รูปที่ 2.46 ไดอะแกรม การใช้งานของฟังก์ชัน

ฟังก์ชันที่เราจะสร้างขึ้นมีรูปแบบการใช้งานดังนี้

```
Function ชื่อฟังก์ชัน ( รายการพารามิเตอร์ จะมีหรือไม่มีก็ได้ ) As ชนิดของข้อมูลที่คืนให้ผู้เรียก
        < คำสั่งใน Visual basic >
        ชื่อฟังก์งชัน  = ค่าที่คืนกลับ
End Function
```

จากรูปแบบการทำงานจะเห็นว่าจะต้องมีการกำหนดชนิดข้อมูลที่จะส่งกลับเสมอ ส่วนจะมีข้อมูลส่งไปประกอบการทำงานของฟังก์ชันนั้นก็สุดแล้วแต่การทำงาน

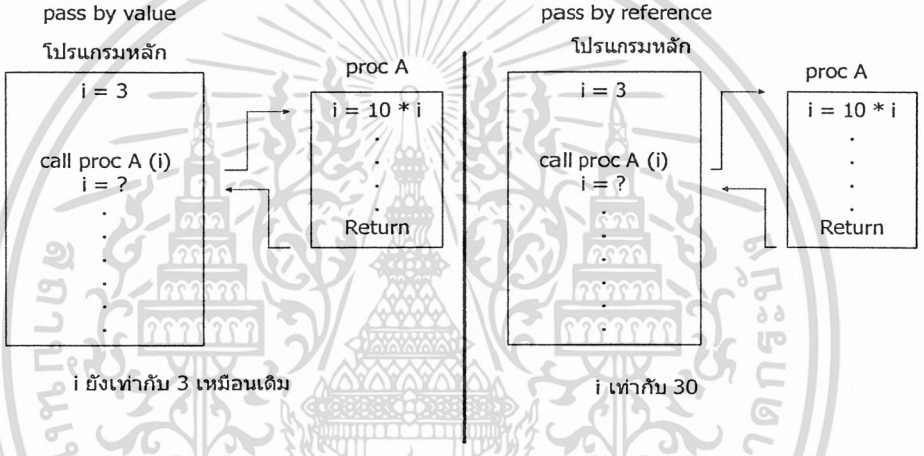
ฟังก์ชันนั้นจะต้องมีการคืนค่ากลับให้ผู้เรียกใช้งาน โดยปกติเราจะระบุชื่อของฟังก์ชันไว้ในบรรทัดสุดท้ายก่อนจบการทำงานของฟังก์ชันเสมอ และกำหนดค่าให้เพื่อที่คืนกลับให้ผู้เรียกใช้ แต่บางฟังก์ชันที่มีการจบการทำงานได้หลายจุด เช่นฟังก์ชันที่มีการตัดสินใจ ซึ่งแต่ละทางเลือกอาจจบไม่เหมือนกัน ก็ต้องมีการระบุชื่อฟังก์ชัน พร้อมด้วยการกำหนดค่าให้เรียบร้อยในทุก ๆ จุดที่จะจบการใช้งานของฟังก์ชัน

- **การผ่านค่าให้โปรแกรมย่อย (Passing Argument)**

ที่ผ่านมาเราจะเห็นว่าในการใช้งานโพรซีเจอร์หรือฟังก์ชัน เราจะต้องส่งข้อมูลบางอย่างไปประกอบในการทำงานของโปรแกรมย่อย เราเรียกข้อมูลที่ส่งไปให้โปรแกรมย่อยหรือฟังก์ชันว่าอาร์กิวเมนต์ อยู่ 2 รูปแบบ ได้แก่

- Pass by value เป็นการผ่านค่าโดยจะก็อปปี้ค่าที่จะผ่าน แล้วส่งไปให้โปรแกรมย่อยที่ต้องการ แม้ว่าภายในโปรแกรมย่อยจะมีการเปลี่ยนแปลงค่าที่ผ่านไปให้นั้น ก็จะไม่มีผลใดๆ ต่อค่าที่ผ่านซึ่งอยู่ภายนอกโปรแกรมย่อยนั้น

- Pass by Reference เป็นการผ่านค่าโดยการบอกตำแหน่งในหน่วยความจำที่เก็บค่าที่ผ่านนั้นไว้ เพราะฉะนั้นถ้าภายในโปรแกรมย่อยมีการเปลี่ยนแปลงค่าที่ผ่านไปให้ย่อมส่งผล โดยตรงของค่าที่ซึ่งอยู่ภายนอกโปรแกรมย่อยนั้น



รูปที่ 2.47 ไดอะแกรม การส่งผ่านค่าให้กับโปรแกรมย่อยทั้งสองแบบ

การผ่านค่าแบบ pass by value นั้นจะเหมือนกับเราสำเนาเอกสารก่อน แล้วค่อยส่งสำเนานั้นไป เพราะฉะนั้นจะทำอะไรกับสำเนาย่อมไม่มีผลกับเอกสารตัวจริง แต่ pass by reference นั้นเหมือนกับการส่งเอกสารตัวจริงไปเลยทีเดียวจึงระมัดระวังในการใช้งาน

- **ขอบเขตของตัวแปรและค่าคงที่ (Scoping)**

ตัวแปร หรือค่าคงที่ที่เราประกาศใช้งานนั้น จะมีขอบเขตการใช้งานมัน โดยเราแบ่งขอบเขตของตัวแปร และค่าคงที่ใน Visual Basic ออกเป็นดังนี้

- Local เป็นขอบเขตที่เล็กที่สุด อาจจะอยู่ใน Sub หรือ Function ที่เราเขียนขึ้นมาใช้งาน เมื่อจบการทำงานของ Sub หรือ Function ตัวแปรหรือค่าคงที่นั้นก็จะไม่ถูกเก็บไว้

- Module เป็นขอบเขตที่กว้างกว่า Local เช่นอยู่ภายในฟอร์มเดียวกัน หรือ Module เดียวกัน(โมดูลจะประกอบด้วย Sub หรือ Function ต่างๆ รวมกันอยู่) โดยค่าตัวแปรหรือค่าที่นั้นจะเก็บค่าใช้งานตลอดทั้งโมดูล

- Global เป็นขอบเขตที่ใหญ่ที่สุดคือประกอบไปด้วยโมดูลต่างๆ

2.3.6 คอนโทรล MSComm

MSComm จัดเตรียมทางเลือกเอาไว้ 2 ทางเพื่อความสะดวกในการสื่อสารข้อมูล ทางแรกคือการสื่อสารข้อมูลที่กระตุ้นด้วยเหตุการณ์ (event – drivent communications) เป็นรูปแบบการใช้งานที่มีประสิทธิภาพมากสำหรับการตอบสนองแบบทันทีทันใด เช่น เมื่อตัวอักษร ถูกส่งมาที่พอร์ตอนุกรมหรือเกิดการเปลี่ยนแปลงที่ขา Data Carrier Detect (DCD) หรือขา Request To Send (RTS) เหตุการณ์ Oncomm ของ Mscomm จะสามารถตรวจจับสัญญาณนั้นได้ทันที ทางเลือกที่สองเป็นการคอยตรวจสอบค่าเหตุการณ์และความผิดพลาดที่เกิดขึ้นด้วยการดูที่ค่าเปลี่ยนแปลงภายในคุณสมบัติ CommEvent หลังจากการให้โปรแกรมทำงานที่ฟังก์ชั่นต่างๆ ไปเรียบร้อยแล้ว ซึ่งวิธีนี้ใช้งานได้ดีในกรณีที่เขียนโปรแกรมมีขนาดเล็กมาก

คอนโทรลของ MSComm 1 ตัวสามารถควบคุมการทำงานของพอร์ตอนุกรมได้ 1 พอร์ต ถ้าในโปรแกรมที่ใช้งานต้องการติดต่อกับพอร์ตอนุกรมมากกว่า 1 พอร์ตจะต้องใช้คอนโทรล MSComm มากกว่า 1 ตัวเพื่อควบคุมพอร์ตอนุกรมในแต่ละพอร์ต แอดเดรสของพอร์ตอนุกรมและแอดเดรสของการเกิดอินเตอร์รัพ สามารถเปลี่ยนแปลงได้จากการแก้ไขค่าที่ Control Panel

ถึงแม้ว่า คอนโทรล MSComm จะมีคุณสมบัติมากมายแต่สามารถทำความเข้าใจได้ดังนี้

CommPort

ใช้ในการกำหนดและอ่านค่าของพอร์ตอนุกรมที่ต่อติดอยู่ (COM1,COM2)

รูปแบบการใช้งาน

object . Commport [= value]

โดย value เป็นค่าของพอร์ตอนุกรม ชนิดของข้อมูลเป็น Integer ค่า Value สามารถกำหนดได้ในช่วง 1 – 16 (ค่าเริ่มต้นกำหนดไว้ที่ 1) เมื่อมีการกำหนดค่าแล้วทำการเปิดพอร์ตโดยใช้คุณสมบัติ PortOpen แต่ว่าพอร์ตนั้นไม่มีอยู่ในระบบ MSComm จะสร้าง สัญญาณแสดงข้อผิดพลาด error 68 ขึ้นมา ซึ่งหมายถึงอุปกรณ์ ตัวนั้นไม่ได้อยู่ในระบบ ดังนั้นการเขียนโปรแกรมจึงจำเป็นต้องกำหนดตำแหน่งของพอร์ตอนุกรมก่อนที่ใช้คำสั่ง OpenPort

Setting

ใช้ในการกำหนดและอ่านค่าอัตราบอดเรท, พาริตี้, จำนวนของบิตข้อมูล, จำนวนของบิตปิดท้าย

รูปแบบการใช้งาน

object . settings [= value]

ค่าของ Value มีชนิดข้อมูลเป็นแบบ String มีรูปแบบเป็น “ BBBB, P ,D,S ” โดย BBBB เป็นค่าอัตราบอดเรท, P เป็นค่าพาริตี้ , D เป็นจำนวนของบิตข้อมูล และ S เป็นจำนวนของบิตปิดท้ายปกติแล้วค่านี้ถูกกำหนดไว้เป็น “ 9600 , N , 8 , 1 ”

อัตราบอดเรทมาตรฐานที่ใช้กับ MSComm มีดังนี้

110 บิตต่อวินาที

300 บิตต่อวินาที

600 บิตต่อวินาที

1,200 บิตต่อวินาที

2,400 บิตต่อวินาที

9,600 บิตต่อวินาที

14,400 บิตต่อวินาที

19,200 บิตต่อวินาที

28,800 บิตต่อวินาที

38,400 บิตต่อวินาที (สงวน)

56,000 บิตต่อวินาที (สงวน)

128,000 บิตต่อวินาที (สงวน)

256,000 บิตต่อวินาที (สงวน)

สำหรับค่ามาตรฐานในการกำหนดค่าพาริตี้มีดังนี้

สัญลักษณ์	รายละเอียด
E	พาริตี้คู่ (Even)
M	ลอจิก "1"(MARK)
N	ไม่ใช้ (ค่าปกติ)
O	พาริตี้คี่ (Odd)
S	ลอจิก "0" (Space)

ค่าที่ใช้ในการกำหดจำนวนบิตมี 5 ค่าคือ 4, 5 , 6 , 7 และ 8 ค่าที่ระบุจำนวนบิตปิดท้ายมี 3 ค่าคือ 1, 1.5 และ 2

ตัวอย่างการใช้งานคำสั่ง Settings โดยจะเป็นการกำหนดค่าบอดเรทเท่ากับ 9600 ไม่มีพาริตี้ จำนวนบิต ข้อมูล 8 บิต และบิตปิดท้าย 1 บิต สามารถเขียนโปรแกรมได้ดังนี้

MSComm1 . Setting = " 9600, N , 8, 1"

PortOpen

ใช้ในการกำหนดและอ่านค่าสถานะของพอร์ตอนุกรม เพื่อเปิดและปิดพอร์ตอนุกรม

รูปแบบการใช้งาน

object . Portopen [= Value]

ค่า Value มีชนิดข้อมูลเป็นแบบบูลีนคือ True กับ False โดย True หมายถึงการเปิดพอร์ตอนุกรม และ False หมายถึงการปิดพอร์ตอนุกรมสำหรับการปิดนั้นจะมีการเคลียร์บัฟเฟอร์รับ ข้อมูลและบัฟเฟอร์ส่งข้อมูลด้วย คอนโทรล MSComm จะปิดพอร์ตอนุกรมโดยอัตโนมัติเมื่อออกจากโปรแกรม ก่อนที่จะใช้ คุณสมบัติ PortOpen ต้องตรวจสอบให้แน่ใจก่อนว่าคุณสมบัติ CommPort นั้นได้ทำการ

กำหนดตำแหน่งของพอร์ตอนุกรมไว้ถูกต้องหรือไม่ มิเช่นนั้น MSComm จะแสดงข้อผิดพลาด Error 68 แจ้งแก่ผู้ใช้งาน หรือถ้าพอร์ตอนุกรมนั้นถูกเปิดไว้แล้ว โปรแกรมก็จะแจ้งข้อผิดพลาดออกมาเช่นกัน

ตัวอย่างการใช้คำสั่งเปิดพอร์ต เพื่อการติดต่อสื่อสารอนุกรม COM1 และมีอัตราบอดเรท 9600 บิตต่อวินาที ไม่มีพาริตี้บิต จำนวนบิตข้อมูล 8 บิต และบิตปิดท้าย 1 บิต ดังนี้

MSComm1 . Settings = " 9600, n, 8, 1"

MSComm1 . Commport = 1

MSComm1. Portopen = true

Input

อ่านค่าและลบค่าขบวนข้อมูลจากภาครับ

รูปแบบการใช้งาน

object . Input

คุณสมบัติ InputLen เป็นตัวกำหนดจำนวนของตัวอักษรที่จะอ่านโดยคุณสมบัติ Input การกำหนดค่าให้ InputLen เท่ากับ 0 เป็นการกำหนดให้คุณสมบัติ Input ทำการอ่านข้อข้อมูลในบัฟเฟอร์รับข้อมูลทั้งหมด

คุณสมบัติ InputMode เป็นตัวกำหนดชนิดของข้อมูลที่คุณสมบัติ Input รับเข้ามา ถ้า InputMode ถูกกำหนดให้เป็น ComInputMode Text คุณสมบัติ Input จะส่งค่าข้อมูลกลับมาในรูปแบบข้อความชนิดข้อมูลเป็นแบบ Varaint ถ้า InputMode กำหนดเป็น comInputModeBinary คุณสมบัติ Input จะส่งข้อมูลกลับมาในรูปของไบนารี่และชนิดข้อมูลเป็นแบบ Varaint

InBufferCount

ส่งค่าจำนวนของตัวอักษรที่อยู่ในบัฟเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

object.inbuffercount [= value]

คำสั่ง InBufferCount จะแสดงค่าจำนวนของตัวอักษร ซึ่งรับมาจากภายนอกและยังเก็บอยู่ภายในบัฟเฟอร์ภาครับ เพื่อให้ผู้ใช้งานอ่านค่าออกไป สำหรับการเคลียร์ค่าบัฟเฟอร์ภาครับโดยกำหนดให้ InBufferCount มีค่าเป็น 0

InbufferSize

กำหนดและคืนค่าขนาดของบัฟเฟอร์ภาครับในหน่วยเป็นไบต์

รูปแบบการใช้งานคำสั่ง

object.InBufferSize [= value]

คำสั่ง InbufferSize ใช้เพื่อกำหนดขนาดของบัฟเฟอร์ภาครับ ค่าเริ่มต้นกำหนดไว้ที่ 1,024 ไบต์ การกำหนดค่าบัฟเฟอร์ภาครับขนาดใหญ่จะทำให้ หน่วยความจำที่เหลือสำหรับการใช้งานส่วนอื่นๆ จะน้อยอย่างไรก็ตามการกำหนดค่า บัฟเฟอร์ภาครับที่น้อยเกินไปจะทำให้ เกิดการโอเวอร์โฟลว์หรือข้อมูลล้นบัฟเฟอร์

InputLen

กำหนดค่าและคืนค่าจำนวนของตัวอักษรที่อ่านจากบัฟเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

object.InputLen [= value]

ค่าเริ่มต้นของคุณสมบัติ InputLen มีค่าเท่ากับ “0” การกำหนดค่าเท่ากับ “0” จะทำให้คำสั่ง Input ของ MSComm อ่านค่าข้อมูลที่อยู่ในบัฟเฟอร์ภาครับทั้งหมด

ถ้าไม่มีข้อมูลอยู่ในบัฟเฟอร์ภาครับมากเท่ากับจำนวน InputLen คำสั่ง Input จะส่งค่าว่าง(“”) กลับออกมา ผู้ใช้งานสามารถตรวจสอบข้อมูลอยู่ในภาครับได้โดยใช้คุณสมบัติ InbufferCount โดยกำหนดให้มีข้อมูลอยู่ในบัฟเฟอร์ภาครับก่อนแล้วจึงค่อยอ่านข้อมูล จากบัฟเฟอร์ภาครับ

InputMode

กำหนดค่าและคืนค่าชนิดของข้อมูลที่รับโดยคำสั่ง Input

รูปแบบการใช้งานคำสั่ง

object.InputMode [= value]

คุณสมบัติ InputMode ใช้กำหนดว่าข้อมูลชนิดไหนที่รับเข้ามาผ่านคำสั่ง Input โดยข้อมูลจะเลือกได้ 2 ประเภทคือ

comInputModeText สำหรับข้อมูลที่อยู่ในรูปข้อความตัวอักษรตามมาตรฐาน ANSI โดยจะต้องกำหนดค่าเป็น “0” และค่าเริ่มต้นของการรับข้อมูลก็จะเป็นค่านี้

comInputModeBinary สำหรับข้อมูลมูลอื่นๆ ซึ่งจะเก็บในรูปไบนารี่รวมกันอยู่เป็นไบต์ข้อมูล Outputใช้ในการส่งขบวนของข้อมูลไปยังบัฟเฟอร์ส่งข้อมูล

รูปแบบการใช้งาน

object.output [= value]

ค่า value เป็นค่าของตัวอักษรที่เขียนไปยังบัฟเฟอร์ส่งข้อมูล คุณสมบัติ Output สามารถใช้ในการส่งข้อมูลตัวอักษรหรือ ข้อมูลไบนารี่ก็ได้ โดยการส่งข้อมูลเป็นรูปแบบตัวอักษรจะต้องกำหนดข้อมูลเป็นแบบ variant และมีข้อมูลภายในแบบ String สำหรับการส่งข้อมูลไบนารี่จะต้องกำหนด ชนิดข้อมูลเป็นแบบ variant และข้อมูลภายในเป็นแบบ byte

OutBufferCount

คืนค่าจำนวนของข้อมูลตัวอักษรที่เก็บอยู่ในบัฟเฟอร์ภาคส่ง และสามารถใช้คำสั่งนี้เคลียร์บัฟเฟอร์ภาคส่งได้ด้วย

รูปแบบการใช้งานคำสั่ง

object.Outbuffercount [= value]

ผู้ใช้งานสามารถเคลียร์บัฟเฟอร์ภาคส่งได้โดยการกำหนดค่า OutbufferCount เท่ากับ “0”

OutBufferSize

กำหนดค่าและคืนค่าขนาดบัฟเฟอร์ภาคส่ง ชนิดของตัวแปรเป็นแบบไบต์

รูปแบบการใช้งานคำสั่ง

Object.OutBufferSize [= value]

คุณสมบัติ OutbufferSize ใช้สำหรับกำหนดขนาดของภาคส่ง โดยค่าปกติที่ใช้งานจะมีค่าเท่ากับ 512 ไบต์

ParityReplace

กำหนดและคืนค่าตัวอักษรที่ไปวางแทนในตำแหน่งที่เกิดข้อผิดพลาดจากพาริตี้

รูปแบบการใช้งานคำสั่ง

Object.ParityReplace [= value]

บิตพาริตี้ เป็นบิตที่ทางภาคส่งข้อมูลทำการส่งมาพร้อมกับข้อมูล เพื่อตรวจสอบข้อผิดพลาดของข้อมูล โดยเมื่อมีการใช้บิตพาริตี้ คอนโทรล MSComm จะทำการบวกบิตทุกบิตที่มีค่าลอจิก “1” ในแต่ละไบต์ และทำการตรวจสอบผลลัพธ์ ว่าบิตที่อ่านได้นั้นมีจำนวนลอจิก “1” เป็นเลขคู่หรือคี่ และตรงกับค่าที่กำหนดไว้แต่ต้นหรือไม่ ถ้าค่าที่นำมาบวกแล้วมีพาริตี้ไม่ตรงแสดงว่าการรับส่งข้อมูลผิดพลาด

บทที่ 3
การคำนวณและการสร้าง

จากทฤษฎีสามารถนำมาออกแบบโครงสร้างการทำงานของโครงงาน เครื่องกำเนิดสัญญาณดิจิตอลหลายช่องสัญญาณแบบโปรแกรมได้ (Multi Channel Data Generator) โดยจะแบ่งออกเป็น 3 ส่วนหลักคือ

1. ส่วนกำหนดข้อมูล
2. ส่วนส่งผ่านข้อมูล
3. ส่วนควบคุมและกำเนิดสัญญาณ

รูปที่ 3.1 บล็อกไดอะแกรมแสดงการทำงานของ Multi Channel Data Generator

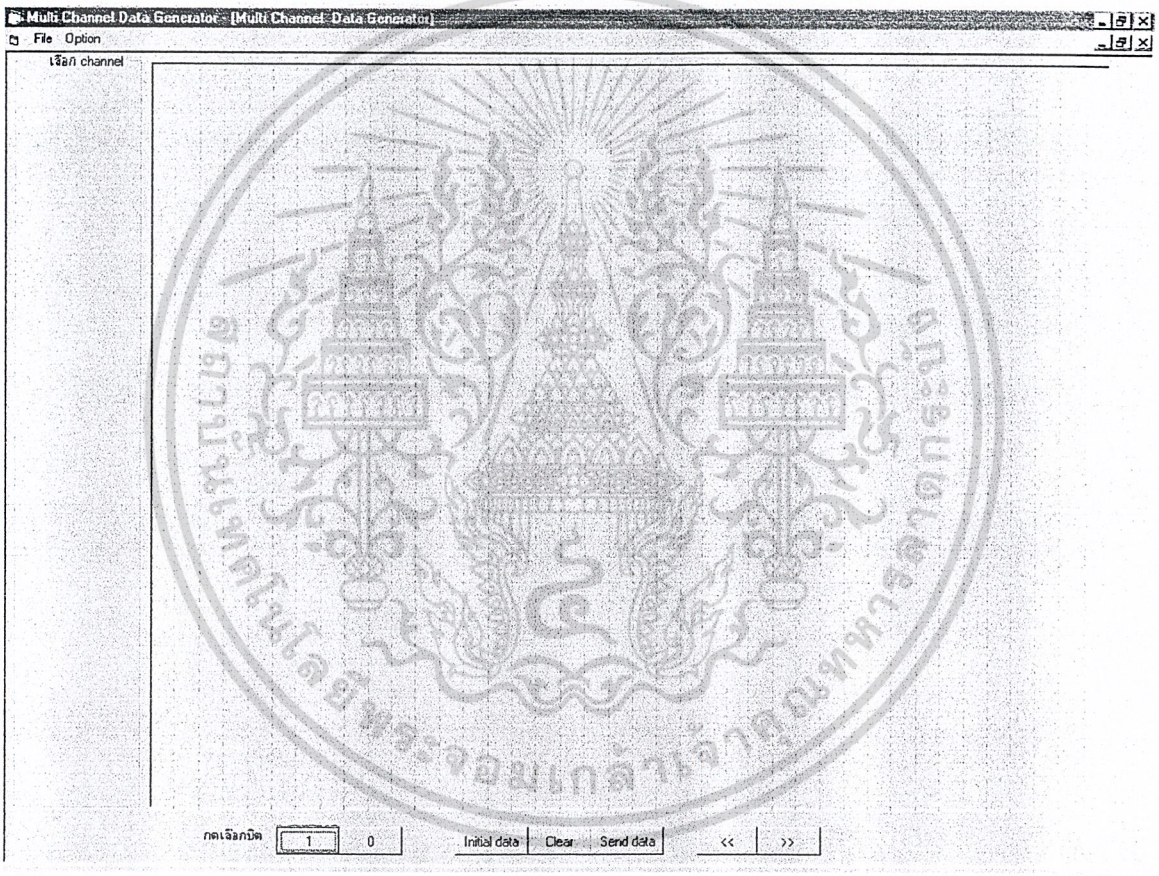
ซึ่งส่วนกำหนดข้อมูลนี้ผู้ใช้จะกำหนดลักษณะของข้อมูลได้จากคอมพิวเตอร์ เมื่อกำหนดข้อมูลเสร็จ ข้อมูลจะถูกส่งจากคอมพิวเตอร์ทางพอร์ทสื่อสารอนุกรม โดยมีส่วนส่งผ่านข้อมูลเป็นตัวกลางในการรับข้อมูลจากคอมพิวเตอร์ แล้วส่งไปยังส่วนควบคุมและกำเนิดสัญญาณต่อไป ซึ่งส่วนควบคุมและกำเนิดสัญญาณเมื่อได้รับข้อมูลมาแล้ว ก็จะนำไปเก็บในหน่วยความจำ และจะเริ่มอ่านข้อมูล หยุดอ่านข้อมูล หรือเปลี่ยนความถี่ในการอ่านข้อมูล ซึ่งกระบวนการเหล่านี้จะถูกควบคุมโดยส่วนควบคุมและกำเนิดสัญญาณ

3.1 การออกแบบวงจร

3.1.1 ส่วนกำหนดข้อมูล

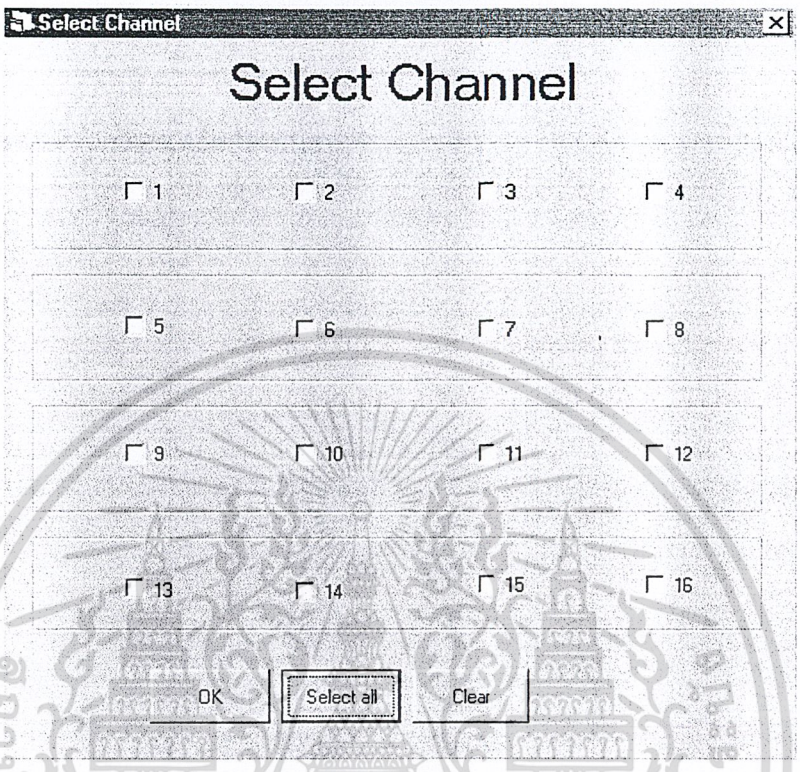
ผู้ใช้สามารถกำหนดข้อมูลได้จากคอมพิวเตอร์ โดยใช้โปรแกรมวิชวลเบสิก การใช้งานมีขั้นตอนดังต่อไปนี้

1. เลือก File แล้วเลือก New project โดยเมื่อเลือกแล้วหน้าจอคอมพิวเตอร์จะมีลักษณะดังรูปที่ 3.2 ตอนนี้ยังไม่สามารถสร้างหรือกำหนดข้อมูลได้ เพราะยังไม่ได้เลือกช่องสัญญาณ (Channel)



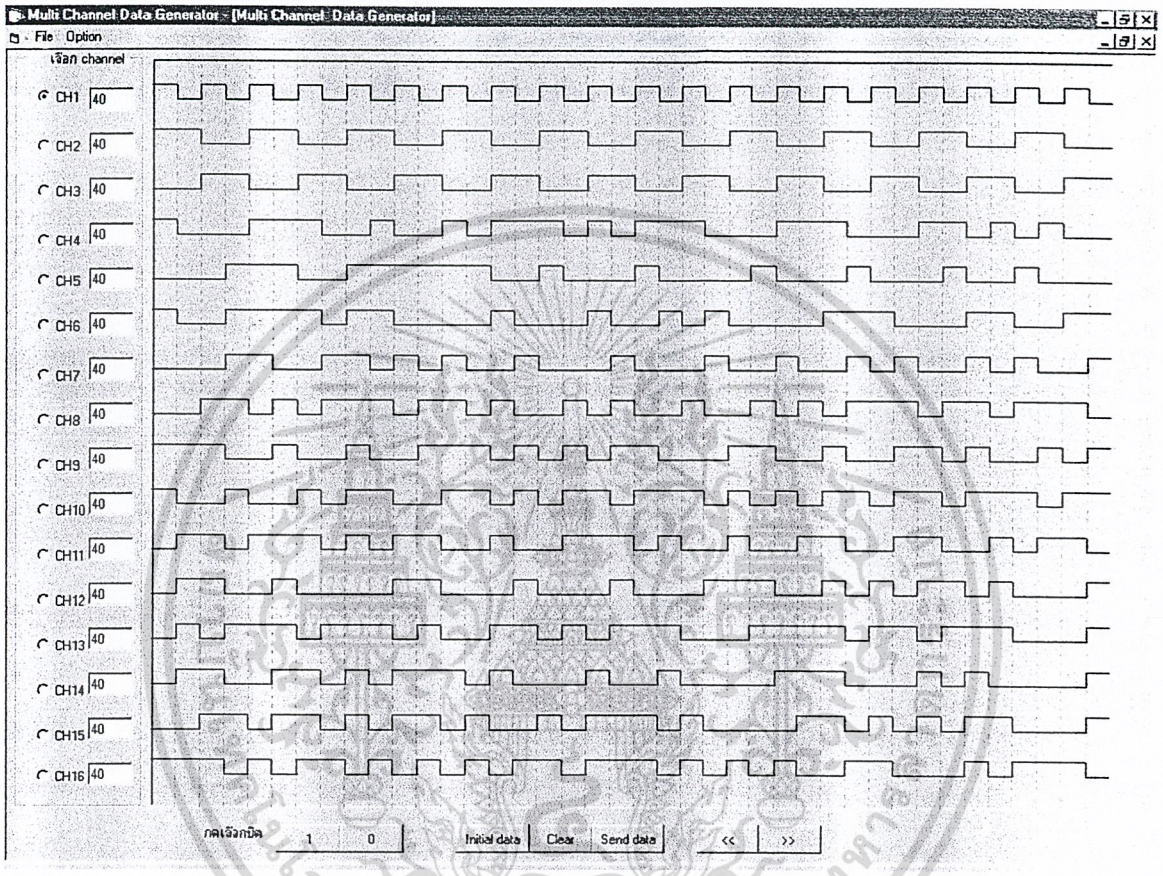
รูปที่ 3.2 แสดงลักษณะหน้าจอคอมพิวเตอร์ตอนที่ยังไม่ได้กำหนดช่องสัญญาณและข้อมูล

2. เลือก Option แล้วเลือก Select channel เพื่อเลือกช่องสัญญาณที่ต้องการใช้งาน หรือเลือกช่องสัญญาณทั้งหมดโดยคลิ๊กปุ่ม Select all แล้วคลิ๊กปุ่ม Ok แสดงดังรูปที่ 3.3



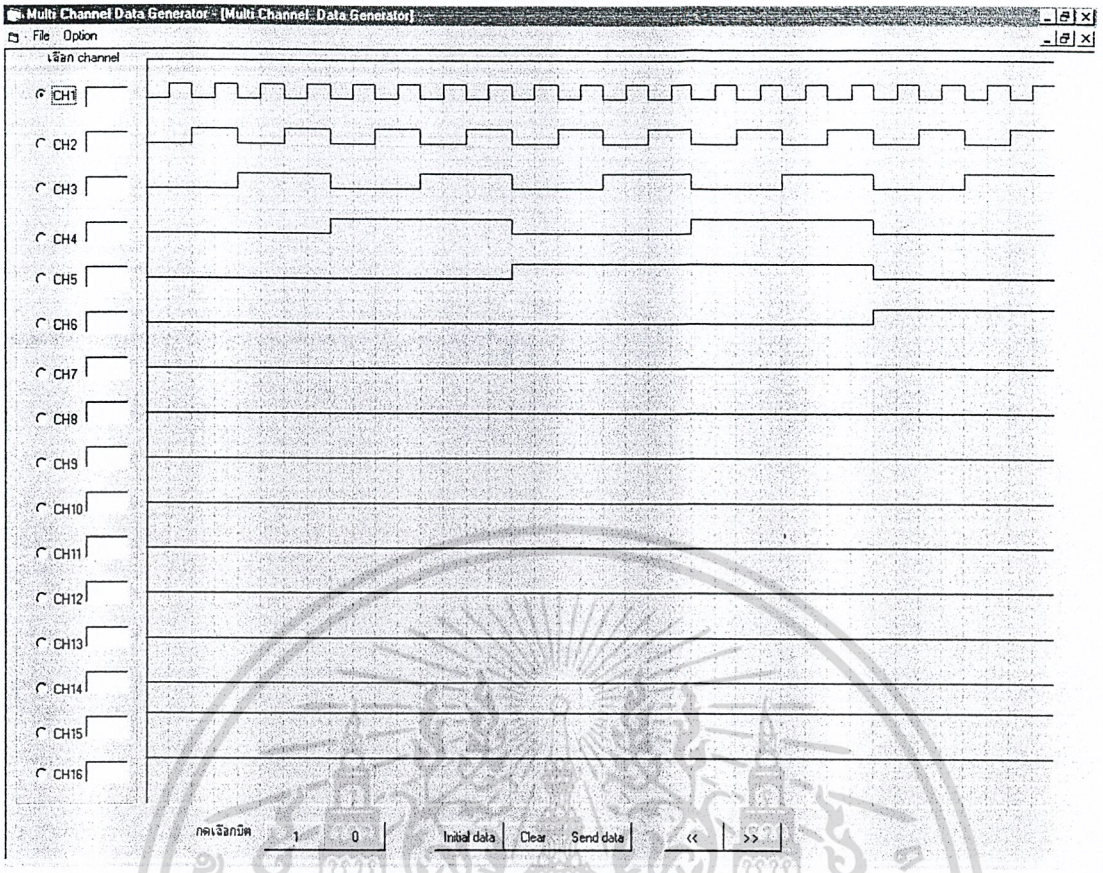
รูปที่ 3.3 แสดงการเลือกช่องสัญญาณในการใช้งาน

3. สร้างสัญญาณข้อมูลตามช่องสัญญาณที่ได้กำหนดไว้ โดยสามารถกำหนดได้โดยการใช้แป้นพิมพ์ ปุ่ม 0 หรือ 1 หรือคลิ๊กจากปุ่มสร้างสัญญาณบนหน้าจอคอมพิวเตอร์(ปุ่ม 0 หรือ 1) โดยสามารถกำหนดข้อมูลได้มากถึง 8 กิโลบิตต่อช่องสัญญาณ โดยสัญญาณที่กำหนดเองจะมีลักษณะดังรูปที่ 3.4



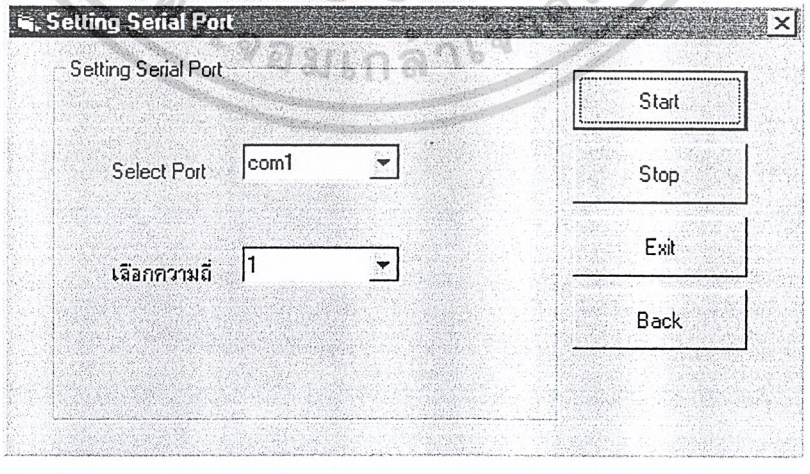
รูปที่ 3.4 แสดงข้อมูลที่ได้กำหนดเองในแต่ละช่องสัญญาณ

หรือหากไม่ต้องการกำหนดข้อมูลเอง ก็สามารถใช้ข้อมูลที่ได้กำหนดไว้แล้วโดยคลิ๊กปุ่ม Initial data ก็ได้ แสดงดังรูปที่ 3.5 โดยสัญญาณที่ได้จะมีลักษณะคือ สัญญาณจากช่องสัญญาณก่อนหน้าจะเป็นสัญญาณนาฬิกา กระตุ้นให้เกิดสัญญาณช่องถัดไป เป็นอย่างนี้จนครบทั้ง 16 ช่องสัญญาณ



รูปที่ 3.5 แสดงข้อมูลที่ได้กำหนดไว้แล้ว

4. เมื่อกำหนดข้อมูลเรียบร้อยแล้ว ก็จะเป็นการส่งข้อมูลผ่านทางพอร์ตอนุกรม โดยคลิ๊กที่ปุ่ม Send data เพื่อกำหนดรูปแบบในการส่งข้อมูล แสดงดังรูปที่ 3.6 เมื่อกำหนดรูปแบบในการส่งข้อมูลเรียบร้อยแล้ว คลิ๊กปุ่ม Start เพื่อทำการส่งข้อมูล

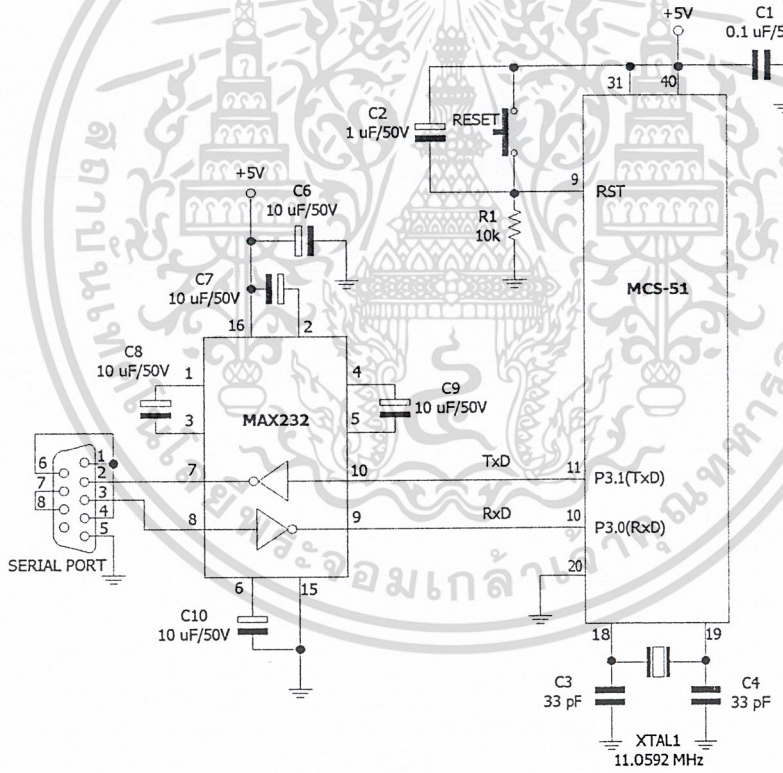


รูปที่ 3.6 แสดงการกำหนดพอร์ตและเลือกความถี่เพื่อเริ่มส่งข้อมูล

3.1.2 ส่วนส่งผ่านข้อมูล

การออกแบบส่วนส่งผ่านข้อมูลนั้น จะมีการต่อวงจรดังรูปที่ 3.7 โดยเป็นการเชื่อมต่อระหว่างพอร์ทอนุกรมของคอมพิวเตอร์และไมโครคอนโทรลเลอร์

ซึ่งการกำหนดอัตราบอด (baud Rate) ที่ใช้ติดต่อสื่อสารคือ 19200 โดยการกำหนดให้พอร์ทอนุกรมทำการรับส่งข้อมูลในโหมด 1 คือ มีการส่งข้อมูลครั้งละ 10 บิต จะแบ่งเป็น บิตเริ่มต้น 1 บิต บิตข้อมูล 8 บิต บิตปิดท้าย 1 บิต โดยการกำหนดให้รีจิสเตอร์ SCON มีค่า 50H และการกำหนดอัตราบอดนั้นจะทำโดยการกำหนดบิต SMOD ในรีจิสเตอร์ PCON ให้มีค่าเป็น "1" โดยการกำหนดให้รีจิสเตอร์ PCON มีค่าเท่ากับ 80H และจะต้องมีการใช้อัตราการเกิดโอเวอร์โฟลว์ของไทเมอร์ 1 เป็นตัวกำหนดอัตราบอดด้วย โดยการเลือกใช้ไทเมอร์ 1 ในโหมด 2 ซึ่งก็จะเป็นการทำหน้าที่เป็นไทเมอร์ขนาด 8 บิต และทำการโหลดค่าเริ่มต้นอัตโนมัติ เมื่อมีการโอเวอร์โฟลว์ โดยการกำหนดค่าของรีจีสเตอร์ TMOD ให้มีค่า 20H และค่าเริ่มต้นจะเก็บไว้ในรีจิสเตอร์ TH1 คือ FDH

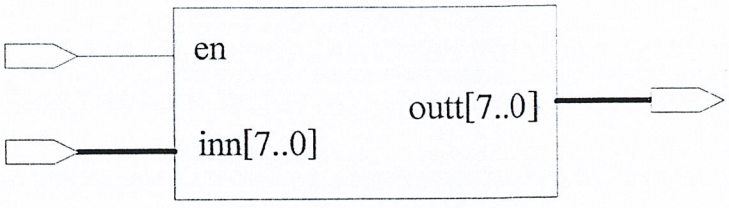


รูปที่ 3.7 แสดงวงจรในส่วนส่งผ่านข้อมูล

รูปที่ 3.8 โฟลว์ชาร์ตแสดงหลักการทำงานของส่วนส่งผ่านข้อมูล

3.1.3 ส่วนควบคุมและกำเนิดสัญญาณ

3.1.3.1 วงจร SINGLE _ BUFF



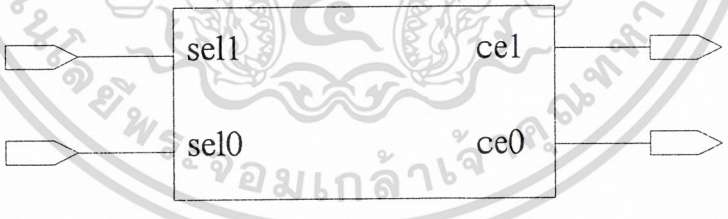
รูปที่ 3.9 แสดงวงจรเสมือนของวงจร SINGLE _BUFF

en	outt[7..0]
0	inn[7..0]
1	Z

ตารางที่ 3.1 แสดงตารางความจริงแสดงสถานะการทำงานของวงจร SINGLE _BUFF

การทำงานของวงจรนี้คือ เมื่อขา en มีสถานะเป็น “0” สัญญาณข้อมูลที่เข้ามาทางขาอินพุท inn[7..0] จะปรากฏที่ขาเอาท์พุท outt[7..0] แต่เมื่อขา en มีสถานะเป็น “1” เอาท์พุทของวงจรที่ขา outt[7..0] จะมีสถานะเป็น High Impedance

3.1.3.2 วงจร CE



รูปที่ 3.10 แสดงวงจรเสมือนของวงจร CE

sel1	sel0	ce0	ce1
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

ตารางที่ 3.2 แสดงตารางความจริงแสดงสถานะการทำงานของวงจร CE

วงจร CE จะทำหน้าที่เลือกไอซีหน่วยความจำ ที่ใช้ตอนการเขียนข้อมูล และอ่านข้อมูลจากหน่วยความจำ โดยเมื่อขา sel1 และ sel0 มีสถานะเป็น “0” จะไม่มีการเลือกไอซีหน่วยความจำในการใช้งาน เมื่อ sel1 มีสถานะเป็น “0” และ sel0 มีสถานะเป็น “1” จะเป็นการเลือกหน่วยความจำตัวที่1 เพื่อที่จะเขียนข้อมูลเข้าไป เมื่อ sel1 มีสถานะเป็น “1” และ sel0 มีสถานะเป็น “0” จะเป็นการเลือกหน่วยความจำตัวที่2 เพื่อที่จะเขียนข้อมูลเข้าไป และเมื่อ sel1 และ sel0 มีสถานะเป็น “1” ทั้งคู่ จะเป็นการเลือกหน่วยความจำทั้งสอง เพื่อใช้ในการอ่านข้อมูลจากหน่วยความจำเพื่อนำข้อมูลไปใช้งาน โดยขา ce0 จะต่ออยู่กับขา CE ของหน่วยความจำตัวที่ 1 และ ce1 จะต่ออยู่กับขา CE ของหน่วยความจำตัวที่ 2

3.1.3.3 วงจร MUX21

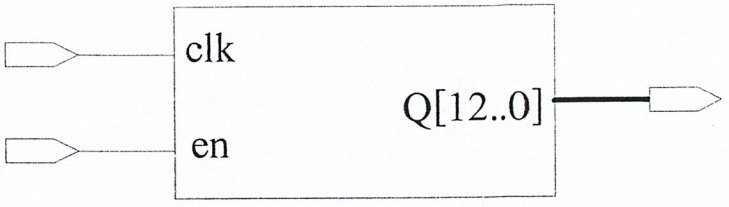
รูปที่ 3.11 แสดงวงจรเสมือนของวงจร MUX21

SEL	A	B	Q
L	X	H	H
L	X	L	L
H	H	X	H
H	L	X	L

ตารางที่ 3.3 แสดงตารางความจริงแสดงสถานะของขาต่างๆ ของวงจร MUX21

ลักษณะการทำงานของวงจรคือ 2 to 1 Multiplex การทำงานของวงจร MUX21 จะใช้ขา SEL ควบคุมขา อินพุท A และ B ว่าจะให้สัญญาณของอินพุทใดไปปรากฏที่ เอาท์พุท Q โดยถ้า SEL มีสถานะเป็น “0” จะเป็นการทำให้ อินพุท B ไปปรากฏที่ เอาท์พุท Q ส่วนถ้า SEL มีสถานะเป็น “1” จะเป็นการทำให้ อินพุท A ไปปรากฏที่ เอาท์พุท Q

3.1.3.4 วงจร COUNT8KBYTE



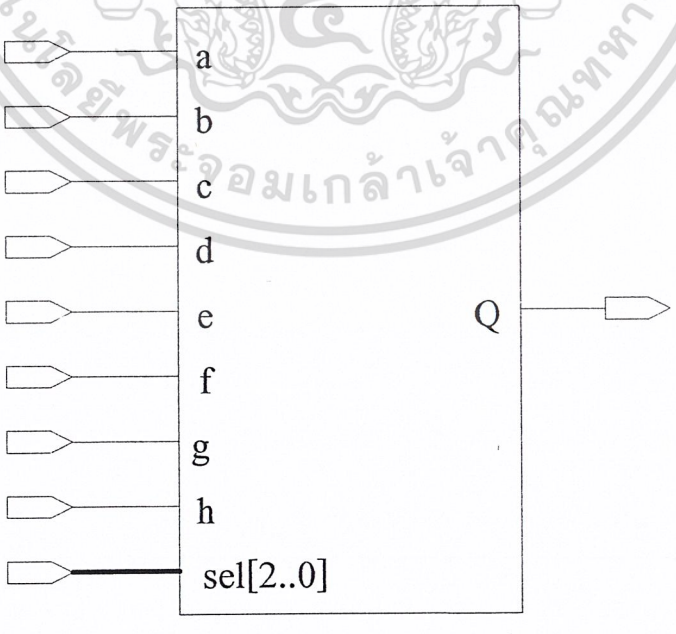
รูปที่ 3.12 แสดงวงจรเสมือนของวงจร COUNT8KBYTE

en	clk	Q [12..0]
0	↑	Q = Q+1
1	x	0

ตารางที่ 3.4 แสดงตารางความจริงแสดงสถานะของขาต่างๆ ของวงจร COUNT8KBYTE

หลักการทำงานของวงจร COUNT8KBYTE คือ จะนับขึ้นที่ขอบขาของสัญญาณ clock และจะนับเมื่อขา en มีสถานะเป็น " 0 " แต่ถ้าขา en มีสถานะเป็น " 1 " วงจรนับจะหยุดนับแล้วทำให้เอาท์พุท Q [12..0] มีสถานะเป็น " 0 " โดยตัววงจรถูกออกแบบให้เป็นวงจรนับ 8192 คือ จาก 0 ถึง 8191 โดยมีขาเอาท์พุท ขนาด 13 บิต คือ Q [12..0]

3.1.3.5 วงจร MUX8



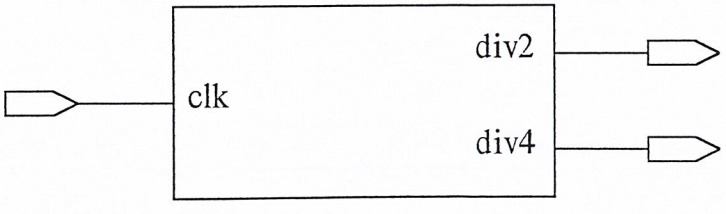
รูปที่ 3.13 แสดงวงจรเสมือนของวงจร MUX8

sel2	sel1	sel0	Q
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h

ตารางที่ 3.5 แสดงตารางความจริงแสดงสถานะของขาต่างๆ ของวงจร MUX8

ลักษณะการทำงานของวงจร 8 to 1 Multiplex การทำงานของวงจร MUX8 คือ จะใช้ขา sel2, sel1 และ sel0 เป็นตัวเลือกสัญญาณอินพุท ให้ไปปรากฏที่ขาสัญญาณเอาท์พุท โดย เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 000 " สัญญาณที่ขา อินพุท a ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 001 " สัญญาณที่ขา อินพุท b ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 010 " สัญญาณที่ขา อินพุท c ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 011 " สัญญาณที่ขา อินพุท d ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 100 " สัญญาณที่ขา อินพุท e ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 101 " สัญญาณที่ขา อินพุท f ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 110 " สัญญาณที่ขา อินพุท g ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q เมื่อขา sel2, sel1 และ sel0 มีสถานะเป็น " 111 " สัญญาณที่ขา อินพุท h ก็จะถูกเลือกให้ไปปรากฏที่ เอาท์พุท Q

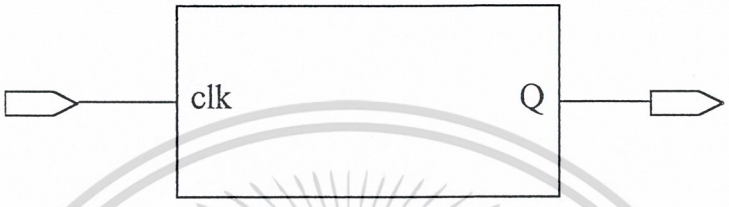
3.1.3.6 วงจร DIV2



รูปที่ 3.14 แสดงวงจรเสมือนของวงจร DIV2

วงจร DIV2 จะทำงานเป็นวงจรหารความถี่ โดยทำงานเหมือนวงจรนับขนาด 2 บิต คือ นับจาก 0 ถึง 3 โดยเอาท์พุทบิตต่ำ คือขาสัญญาณ div2 โดยความถี่ของสัญญาณนาฬิกาที่ขานี้ จะมีความถี่ลดลงเป็น 2 เท่า ของสัญญาณนาฬิกาที่ป้อนเข้ามาที่ขาอินพุท clk และที่ขาสัญญาณเอาท์พุทบิตสูง div4 สัญญาณนาฬิกาที่ขานี้จะมีความถี่ลดลงเป็น 4 เท่า ของสัญญาณนาฬิกาที่ป้อนเข้ามาที่ขาอินพุท clk

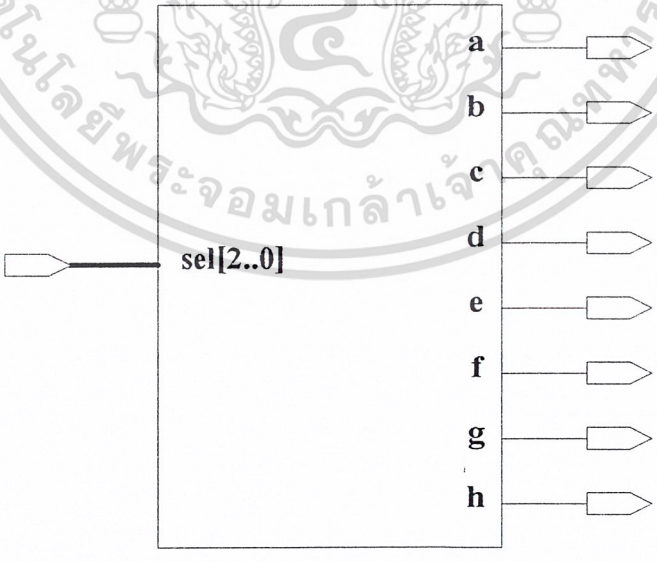
3.1.3.7 วงจร DIV10



รูปที่ 3.15 แสดงวงจรเสมือนของวงจร DIV10

วงจร DIV10 จะทำงานเป็นวงจรหารความถี่ โดยเอาท์พุทที่ขา Q จะกลับสถานะจาก “0” เป็น “1” หรือจาก “1” เป็น “0” โดยจะกลับสถานะทุกๆ 10 คาบเวลา ของสัญญาณนาฬิกาที่ป้อนเข้ามาทางขา clk ซึ่งจะทำให้สัญญาณนาฬิกาที่ออกที่เอาท์พุท Q จะลดลง 10 เท่า ของสัญญาณนาฬิกาที่อินพุท clk

3.1.3.8 วงจร FREQUENCY



รูปที่ 3.16 แสดงวงจรเสมือนของวงจร FREQUENCY

sel[2..0]	a	b	c	d	e	f	g	h
0 0 0	1	0	0	0	0	0	0	0
0 0 1	0	1	0	0	0	0	0	0
0 1 0	0	0	1	0	0	0	0	0
0 1 1	0	0	0	1	0	0	0	0
1 0 0	0	0	0	0	1	0	0	0
1 0 1	0	0	0	0	0	1	0	0
1 1 0	0	0	0	0	0	0	1	0
1 1 1	0	0	0	0	0	0	0	1

ตารางที่ 3.6 แสดงตารางความจริงแสดงสถานะการทำงานของวงจร FREQUENCY

การทำงานของวงจร FREQUENCY จะทำการถอดรหัสสัญญาณการเลือกความถี่มาแสดงผล โดยเมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “000” จะทำให้สัญญาณเอาท์พุทที่ขา a มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “001” จะทำให้สัญญาณเอาท์พุทที่ขา b มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “010” จะทำให้สัญญาณเอาท์พุทที่ขา c มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “011” จะทำให้สัญญาณเอาท์พุทที่ขา d มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “100” จะทำให้สัญญาณเอาท์พุทที่ขา e มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “101” จะทำให้สัญญาณเอาท์พุทที่ขา f มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “110” จะทำให้สัญญาณเอาท์พุทที่ขา g มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0” เมื่อสัญญาณที่ขาอินพุท sel[2..0] มีสถานะ “111” จะทำให้สัญญาณเอาท์พุทที่ขา h มีสถานะเป็น “1” ขาเอาท์พุทอื่นมีสถานะเป็น “0”

3.2 หลักการทำงาน

เมื่อผู้ใช้ได้กำหนดข้อมูลที่ต้องการแล้ว โปรแกรมคอมพิวเตอร์ก็จะนำข้อมูลไปเก็บในอะเรย์ภายในโปรแกรม โดยจะมี 2 อะเรย์ คือ wave1 และ wave2 โดยอะเรย์ wave1 จะเก็บข้อมูลของช่องสัญญาณที่ 1 ถึง 8 และ อะเรย์ wave2 จะเก็บข้อมูลของช่องสัญญาณที่ 9 ถึง 16 เมื่อต้องการส่งข้อมูลก็กดปุ่มส่งข้อมูล โดยในขณะนั้นทางไมโครคอนโทรลเลอร์ก็เตรียมพร้อมที่จะรับข้อมูลเพื่อนำไปเก็บยังหน่วยความจำตัวที่1โดยมีการกำหนดสภาวะเริ่มต้นคือใช้ขาEN1,EN0,SEL2,SEL1,SEL0,START_LED, CLK เป็น “0” และ WE,LOADING เป็น “1” จะทำให้แอลอีดี D2 ติดเพื่อแสดงว่ากำลังรับข้อมูล

เมื่อผ่านกระบวนการกำหนดสภาวะเริ่มต้นแล้ว ไมโครคอนโทรลเลอร์ก็จะกำหนดให้ขา EN0 มีสถานะเป็น “1” จะทำให้วงจร SINGLE_BUFF ตัวที่ 1 พร้อมที่จะทำการรับข้อมูล DATA[7..0] จากไมโครคอนโทรลเลอร์ ส่งไปเก็บยังหน่วยความจำตัวที่ 1 และจากขา EN[1..0] มีสถานะเป็น “01” เมื่อป้อนให้กับวงจร CE ทางขา sel1 และ sel0 ของวงจร CE จะทำให้ขา ce0 มีสถานะเป็น “0” ซึ่งขานี้จะต่ออยู่กับขา CE ของหน่วยความจำตัวที่ 1 เพื่อเลือกใช้หน่วยความจำตัวที่1 ทั้งหมดนี้เป็นกระบวนการเตรียมความพร้อมที่จะนำข้อมูลไปเก็บยังหน่วยความจำตัวที่ 1 จากนั้นคอมพิวเตอร์ ก็จะส่งข้อมูลของอะเรย์ wave1 ไปยังไมโครคอนโทรลเลอร์ครั้งละฟิลด์ แล้วไมโครคอนโทรลเลอร์ก็จะส่งค่าเดิมกลับเพื่อให้คอมพิวเตอร์รู้ว่า ไมโครคอนโทรลเลอร์ได้รับข้อมูลแล้ว และให้ส่งข้อมูลฟิลด์ต่อไปจนครบ 8192 ฟิลด์ ที่อยู่ในอะเรย์ wave1 โดยตอนที่ไมโครคอนโทรลเลอร์ได้รับข้อมูลแต่ละฟิลด์ ไมโครคอนโทรลเลอร์จะทำการเปลี่ยนสถานะขาสัญญาณ WE เป็น “0” และ CLK เป็น “1” ประมาณ 1.085 μS ตามลำดับ แล้วก็จะกลับไปอยู่ในสภาวะเดิม โดยใช้สัญญาณ WE เพื่อใช้เป็นสัญญาณในการเขียนข้อมูลเก็บไว้ในหน่วยความจำ และใช้สัญญาณ CLKเป็นสัญญาณนาฬิกา เพื่อกระตุ้นวงจร COUNT8KBYTE เพื่อให้นับขึ้นเพื่อชี้แอดเดรสถัดไปของหน่วยความจำเป็นอย่างนี้จนครบ 8192 ฟิลด์

ในวงจร COUNT8KBYTE ซึ่งเป็นวงจรชี้แอดเดรสให้กับหน่วยความจำ เมื่อ EN[1..0] มีสถานะเป็น “01” , “10”, “11” โดยเมื่อ EN[1..0] มีสถานะเป็น “01” , “10” เมื่อจะทำการนำข้อมูลไปเก็บยังหน่วยความจำตัวที่ 1 และ 2 จะทำให้วงจร MUX21 เลือกสัญญาณนาฬิกาที่กำเนิดจากไมโครคอนโทรลเลอร์ป้อนให้กับวงจร COUNT8KBYTE และเมื่อ EN มีสถานะเป็น “11” วงจร MUX21 ก็จะเลือกสัญญาณนาฬิกาที่ได้จากวงจร MUX8 เพื่อใช้ในการอ่านข้อมูล

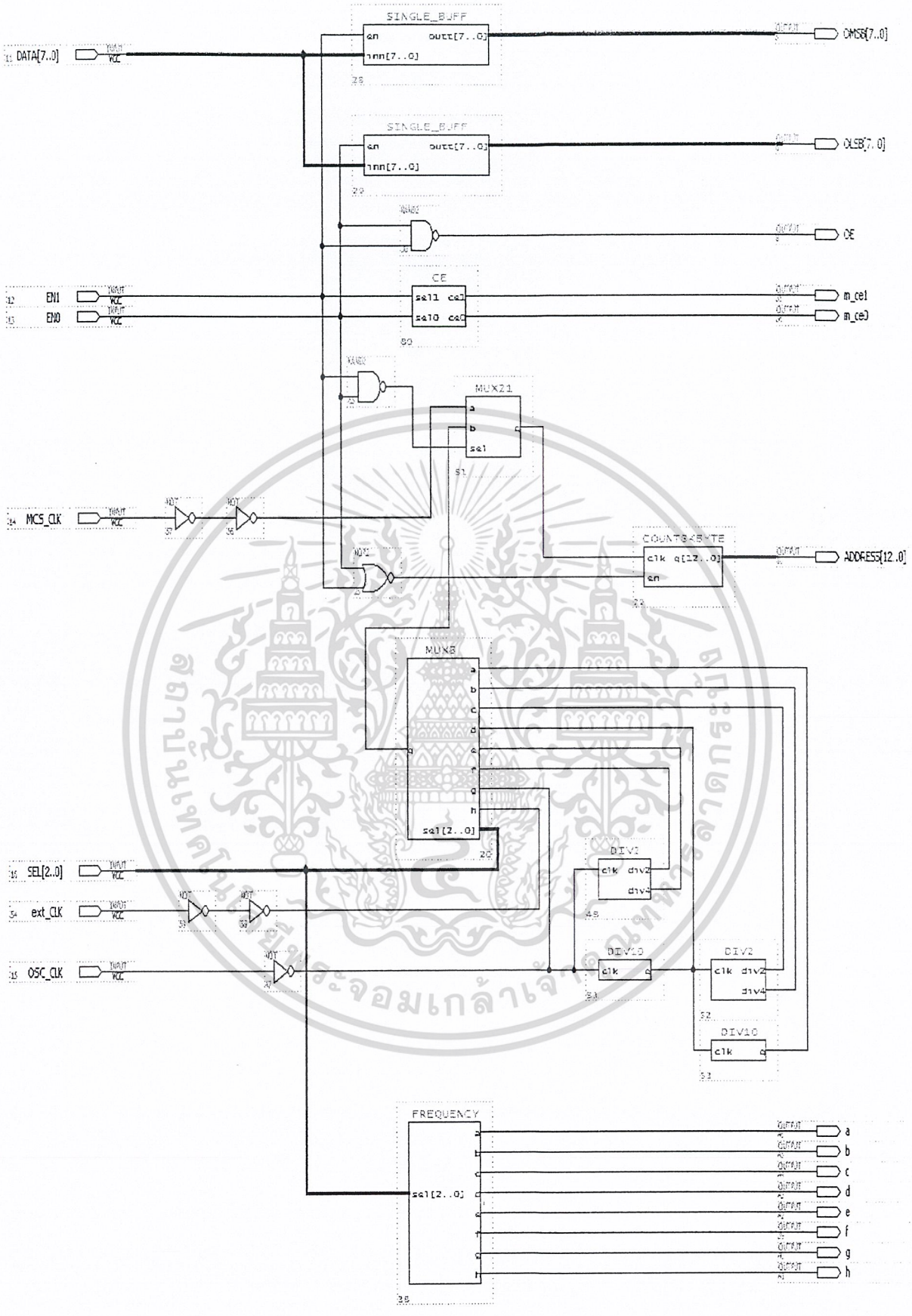
เมื่อส่งข้อมูลเสร็จแล้ว ขาสัญญาณ EN[1..0] ก็จะเปลี่ยนสถานะเป็น “00” เพื่อเคลียร์เอาท์พุทของวงจร COUNT8KBYTE

และจากนั้นขาสัญญาณ EN[1..0] ถูกกำหนดสถานะเป็น “10” เพื่อเลือกหน่วยความจำตัวที่ 2 เพื่อใช้ในการเก็บข้อมูลในอะเรย์ wave2 ที่ส่งมาจากคอมพิวเตอร์ ซึ่งระหว่างที่มีการส่งข้อมูลมาครั้งละฟิลด์นั้น ก็จะมีกระบวนการกำหนดสัญญาณ WE และ CLK เหมือนกับตอนนำข้อมูลไปเก็บที่หน่วยความจำตัวที่ 1

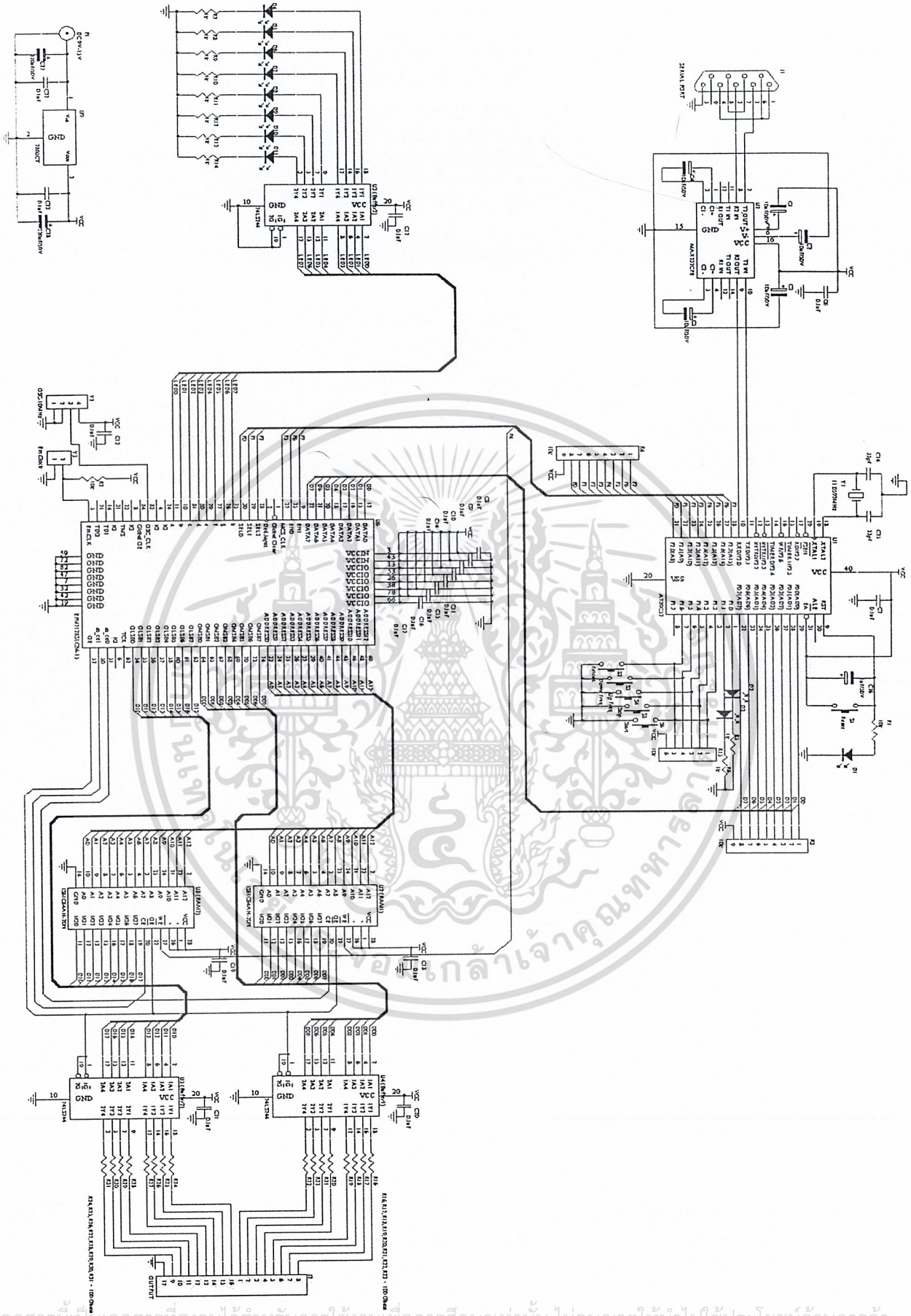
เมื่อนำข้อมูลมาเก็บในหน่วยความจำทั้งสองเสร็จแล้ว ไมโครคอนโทรลเลอร์ก็จะทำการกำหนดค่าของ EN[1..0] ให้มีสถานะเป็น “00” เพื่อรอการสั่งให้อ่านข้อมูล และเคลียร์ขาสัญญาณ LOADING จะทำให้แอลอีดี D2 ดับ เพื่อบอกว่ารับข้อมูลเสร็จแล้ว จากนั้นไมโครคอนโทรลเลอร์ก็จะรับค่าของการเลือกความถี่เริ่มต้นในการอ่านข้อมูลมาเก็บไว้ในรีจิสเตอร์ R1 โดยค่าของการเลือกความถี่การนับ สามารถปรับโดยไมโครคอนโทรลเลอร์อีกครั้ง โดยปุ่ม UP และ DOWN โดยมีให้เลือกทั้งหมด 7 ความถี่ คือ 10 MHz, 5 MHz, 2.5 MHz, 1 MHz ,0.5 MHz, 0.25 MHz, 0.1 MHz และอีก 1 ช่องสัญญาณนาฬิกาจากภายนอก โดยสัญญาณนาฬิกาหลักจะมาจากออสซิลเลเตอร์ 10 MHz แล้วผ่านวงจร DIV10, DIV2 เพื่อลดความถี่ลงป้อนให้กับวงจร MUX8 โดยที่อินพุท a ของวงจร MUX8 ถูกป้อนด้วยความถี่ 0.1 MHz อินพุท b ถูกป้อนด้วยความถี่ 0.25 MHz อินพุท c ถูกป้อนด้วยความถี่ 0.5 MHz อินพุท d ถูกป้อนด้วยความถี่ 1 MHz อินพุท e ถูกป้อนด้วยความถี่ 2.5 MHz อินพุท f ถูกป้อนด้วยความถี่ 5 MHz อินพุท g ถูกป้อนด้วยความถี่

10 MHz และอินพุท h เป็นสัญญาณนาฬิกาจากภายนอก การเลือกความถี่ใช้สัญญาณ SEL[2..0] ในการควบคุม และสัญญาณที่ใช้เลือกความถี่นี้ จะถูกส่งไปป้อนให้วงจร FREQUENCY เพื่อแสดงผล โดยจะมีเอาท์พุท A,B,C,D,F,F,G และ H ซึ่งต่อกับแอลอีดี เพื่อแสดงผล ซึ่งแอลอีดี D4 จะติดเมื่อความถี่ 0.1 MHz ถูกเลือกใช้แอลอีดี D5 จะติดเมื่อความถี่ 0.25 MHz ถูกเลือกใช้ แอลอีดี D6 จะติดเมื่อความถี่ 0.5 MHz ถูกเลือกใช้ แอลอีดี D7 จะติดเมื่อความถี่ 1 MHz ถูกเลือกใช้ แอลอีดี D8 จะติดเมื่อความถี่ 2.5 MHz ถูกเลือกใช้ แอลอีดี D9 จะติดเมื่อความถี่ 5 MHz ถูกเลือกใช้ แอลอีดี D10 จะติดเมื่อความถี่ 10 MHz ถูกเลือกใช้ และ แอลอีดี D11 จะติดเมื่อเลือกใช้สัญญาณนาฬิกาจากภายนอก

เมื่อต้องการให้เริ่มต้นการอ่านข้อมูลก็จะต้องกดปุ่ม START(แอลอีดี D3 จะติด) ไมโครคอนโทรลเลอร์จะกำหนดให้ขา EN[1..0] เป็น “11” ทำให้วงจรเริ่มอ่านข้อมูล และกดปุ่ม STOP(แอลอีดี D3 จะดับ) เพื่อหยุดการอ่านข้อมูล โดยไมโครคอนโทรลเลอร์จะกำหนดให้ขา EN[1..0] เป็น “00” เพื่อหยุดการอ่านข้อมูล และมีปุ่ม RELOAD เพื่อเตรียมพร้อมในการรับข้อมูลใหม่ โดยตอนที่เริ่มอ่านข้อมูลหยุดอ่านข้อมูล จะมีสัญญาณ OE ซึ่งเป็นสัญญาณที่ออกมาจากขา OE ของ ไอซี. เอฟพีจีเอ. ซึ่งต่ออยู่กับขา 1G,2G ของบัฟเฟอร์ตัวที่1 และ 2 ไปควบคุมบัฟเฟอร์ ที่ใช้ในส่งผ่านข้อมูลออกเอาท์พุทของวงจร โดยในขณะที่หน่วยความจำกำลังรับข้อมูลอยู่หรือตอนหยุดการทำงานอยู่นั้น ขาสัญญาณ OE จะเป็น “1” จะทำให้ไม่มีสัญญาณออกไปที่เอาท์พุท แต่เวลาที่อ่านข้อมูล OE จะเป็น “0” เพื่อเลือกสัญญาณออกเอาท์พุทเพื่อนำไปใช้งานต่อไป



รูปที่ 3.17 แสดงวงจรภายในของ FPGA

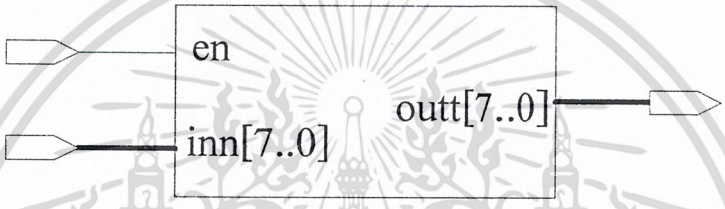


รูปที่ 3.18 แสดงวงจรการทำงานของ Multi Channel Data Generator

บทที่ 4
การทดลองและผลการทดลอง

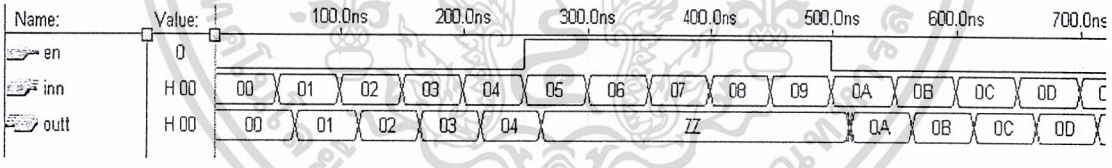
ผลการทดลองของวงจรดิจิตอลส่วนต่างๆ ภายในตัว ไอซี. เอฟพีจีเอ. จะเป็นการจำลองการทำงาน(simulate)ภายใต้สภาวะเงื่อนไขต่างๆของวงจรโดยใช้โปรแกรม แม๊ก+พลัส ทู 10.0(MAX+plus II 10.0) ซึ่งการจำลองการทำงานของวงจรใดๆสามารถทำได้โดยเปิดโปรแกรมของวงจรที่ต้องการทดลองขึ้นมา แล้วทำการจำลองการทำงานโดยกำหนดสัญญาณที่ขาอินพุทต่างๆตามเงื่อนไขที่ได้ออกแบบมา เมื่อสั่งให้จำลองการทำงานแล้วก็จะได้รูปสัญญาณเอาท์พุทของวงจรนั้นออกมา

4.1 วงจร SINGLE_BUFF



รูปที่ 4.1 แสดงวงจรเสมือนของวงจร SINGLE_BUFF

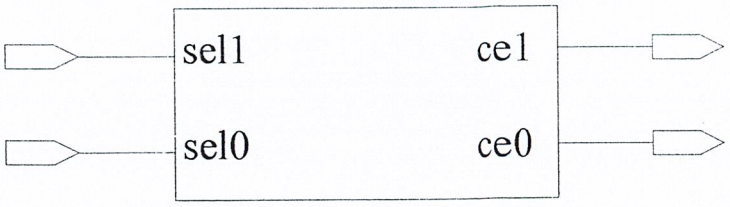
การทดลองและผลการทดลอง



รูปที่ 4.2 แสดงผลการทดลองของวงจร SINGLE_BUFF

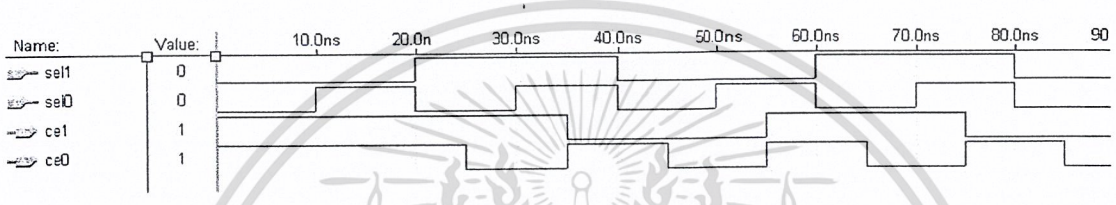
รูปที่ 4.2 แสดงผลการทดลองของวงจร SINGLE_BUFF โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ en และ inn[7..0] ซึ่งจะได้เอาท์พุทที่ขา outt[7..0] ดังรูป

4.2 วงจร CE



รูปที่ 4.3 แสดงวงจรเสมือนของวงจร CE

การทดลองและผลการทดลอง



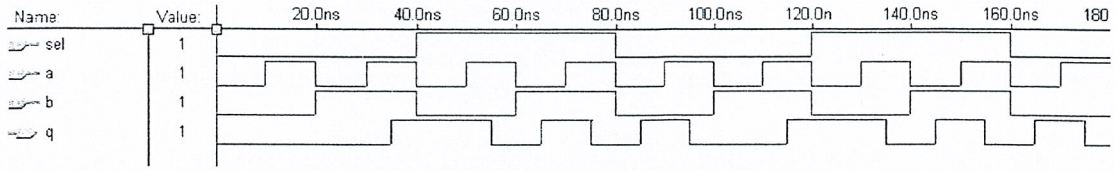
รูปที่ 4.4 แสดงผลการทดลองของวงจร CE

รูปที่ 4.4 แสดงผลการทดลองของวงจร CE โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ sel1 และ sel0 ซึ่งจะได้เอาท์พุทที่ขา ce1 และ ce0 ดังรูป

4.3 วงจร MUX21

รูปที่ 4.5 แสดงวงจรเสมือนของวงจร MUX21

การทดลองและผลการทดลอง



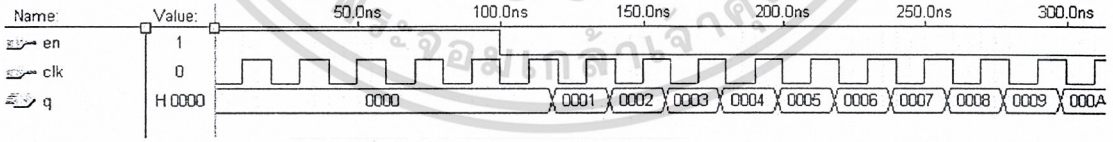
รูปที่ 4.6 แสดงผลการทดลองของวงจร MUX21

รูปที่ 4.6 แสดงผลการทดลองของวงจร MUX21 โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ a,b และ sel ซึ่งจะได้เอาท์พุทที่ขา q ดังรูป

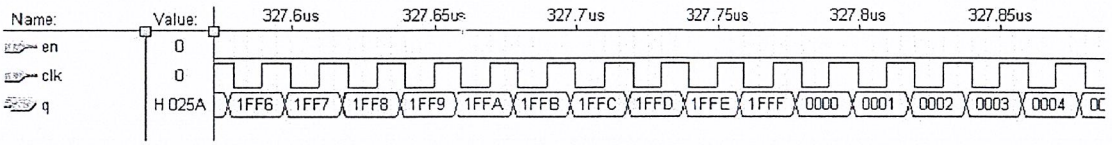
4.4 วงจร COUNT8KBYTE

รูปที่ 4.7 แสดงวงจรเสมือนของวงจร COUNT8KBYTE

การทดลองและผลการทดลอง



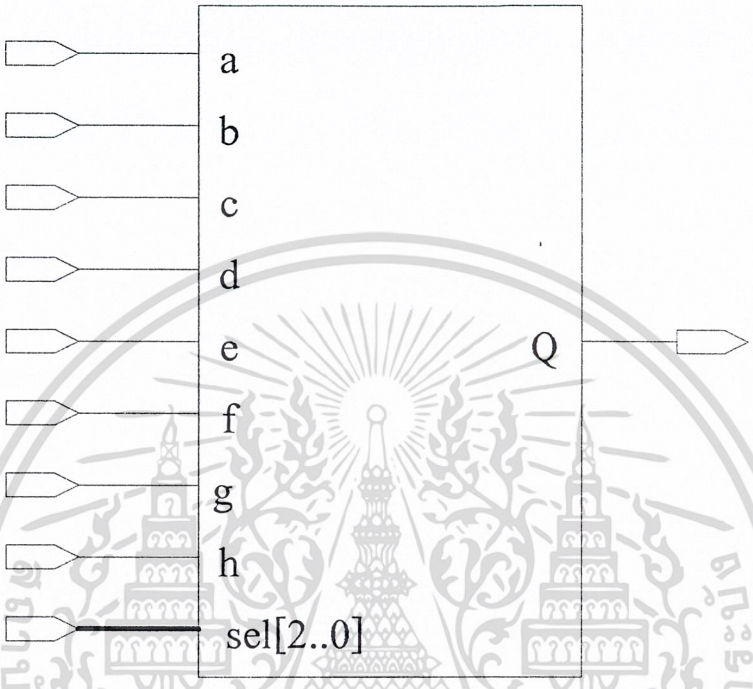
รูปที่ 4.8 แสดงผลการทดลองของวงจร COUNT8KBYTE (ขณะเริ่มนับ)



รูปที่ 4.9 แสดงผลการทดลองของวงจร COUNT8KBYTE (ขณะนับเสร็จ)

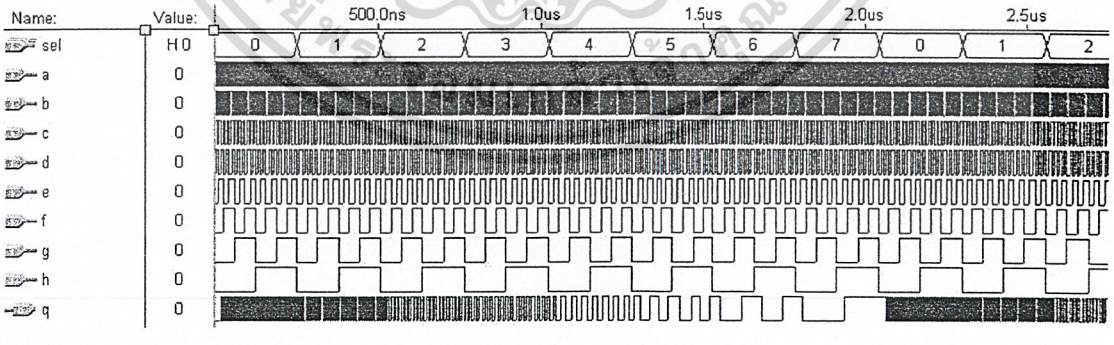
รูปที่ 4.8 และ 4.9 แสดงผลการทดลองของวงจร COUNT8KBYTE โดยการกำหนดสัญญาณอิน-พุทให้กับขาสัญญาณ en และ clk ซึ่งจะได้เอาท์พุทที่ขา q[12..0] ดังรูป

4.5 วงจรMUX8



รูปที่ 4.10 แสดงวงจรเสมือนของวงจรMUX8

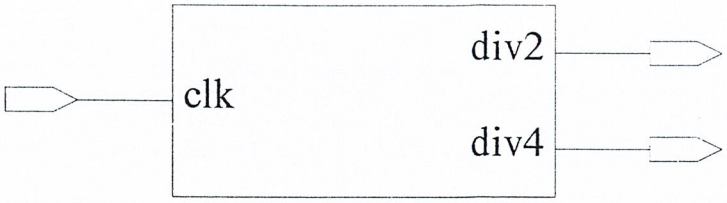
การทดลองและผลการทดลอง



รูปที่ 4.11 แสดงผลการทดลองของวงจร MUX8

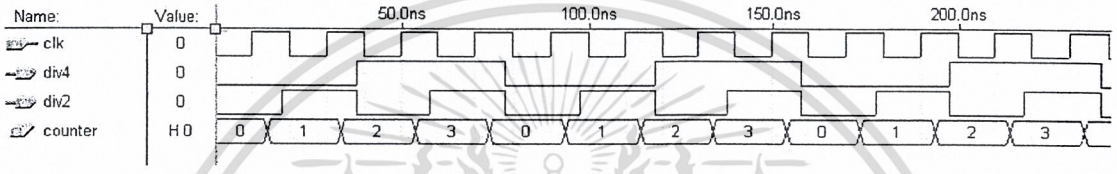
รูปที่ 4.11 แสดงผลการทดลองของวงจร MUX8 โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ sel[2..0],a,b,c,d,e,f,g และ h ซึ่งจะได้เอาท์พุทที่ขา q ดังรูป

4.6 วงจร DIV2



รูปที่ 4.12 แสดงวงจรเสมือนของวงจร DIV2

การทดลองและผลการทดลอง



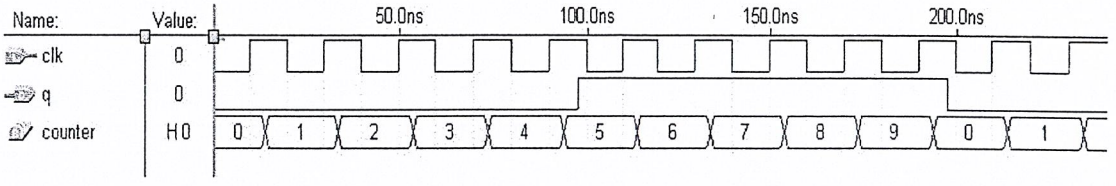
รูปที่ 4.13 แสดงผลการทดลองของวงจร DIV2

รูปที่ 4.13 แสดงผลการทดลองของวงจร DIV2 โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ clk ซึ่งจะได้เอาท์พุททีขา div4 และ div2 ดังรูป

4.7 วงจร DIV10

รูปที่ 4.14 แสดงวงจรเสมือนของวงจร DIV10

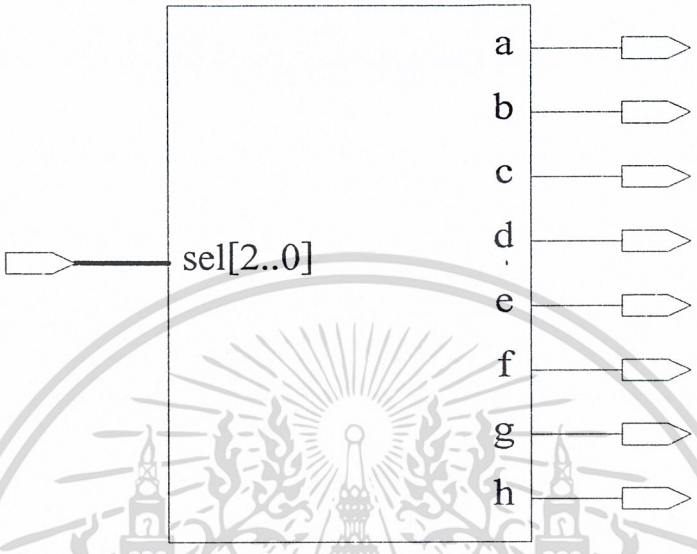
การทดลองและผลการทดลอง



รูปที่ 4.15 แสดงผลการทดลองของวงจร DIV10

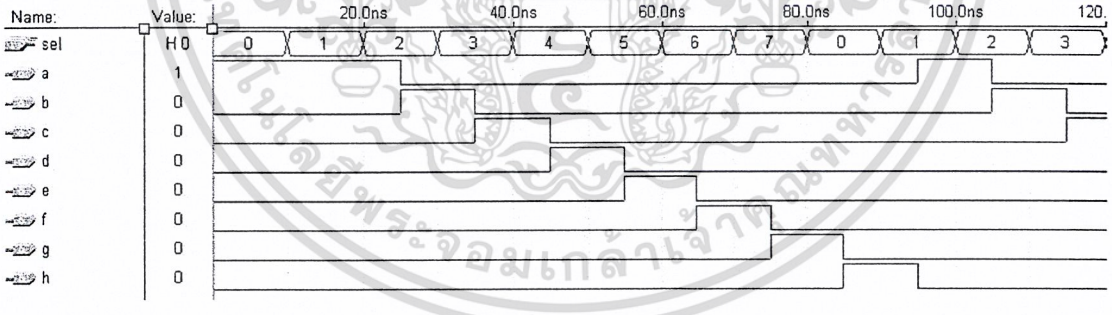
รูปที่ 4.15 แสดงผลการทดลองของวงจร DIV10 โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ clk ซึ่งจะได้เอาท์พุทที่ขา div10 ดังรูป

4.8 วงจร FREQUENCY



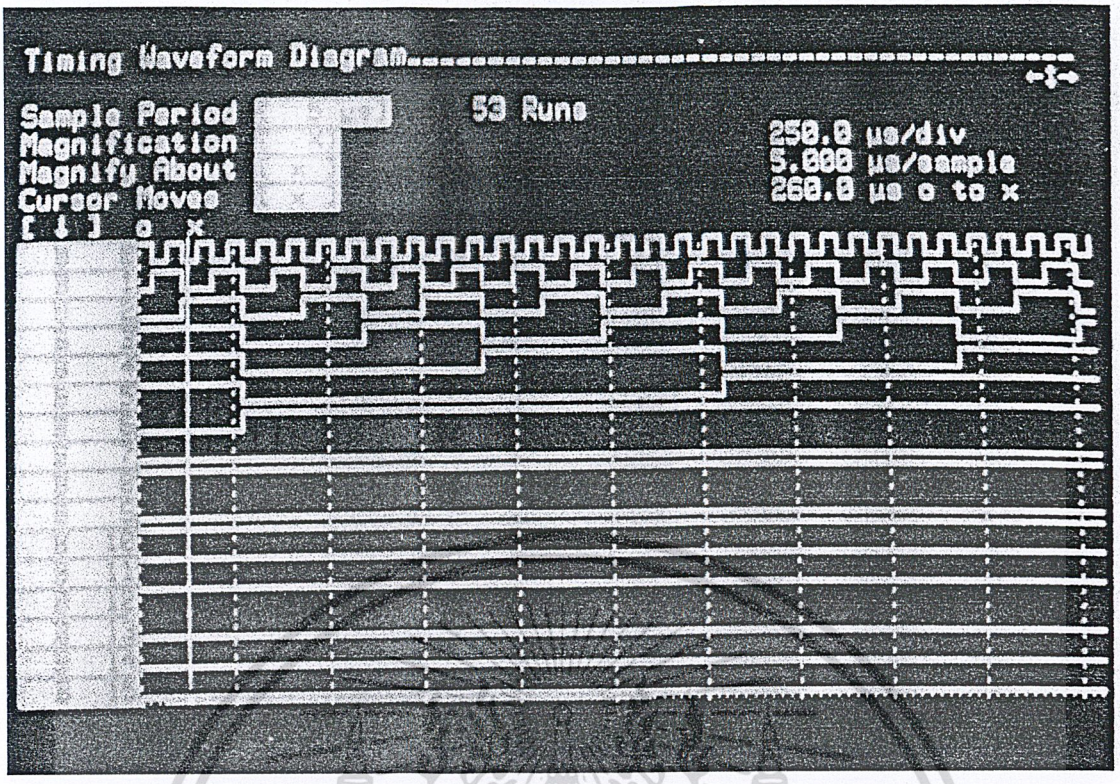
รูปที่ 4.16 แสดงวงจรเสมือนของวงจร FREQUENCY

การทดลองและผลการทดลอง

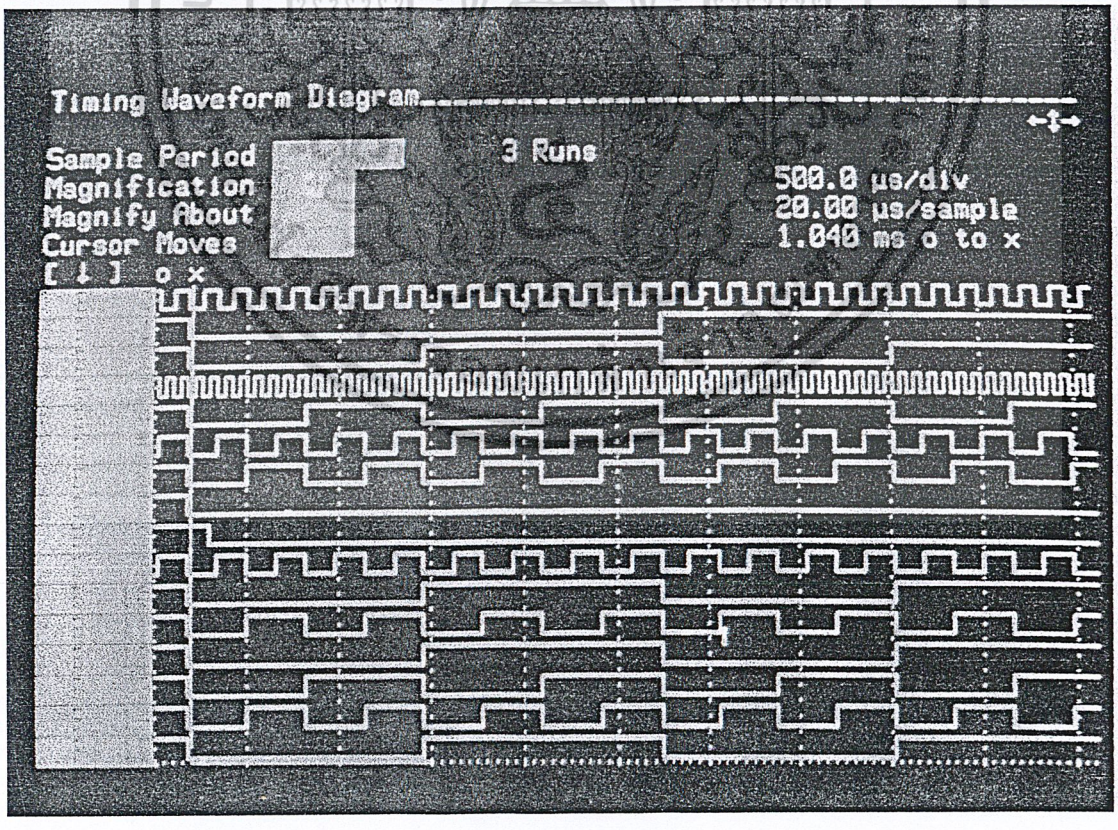


รูปที่ 4.17 แสดงผลการทดลองของวงจร FREQUENCY

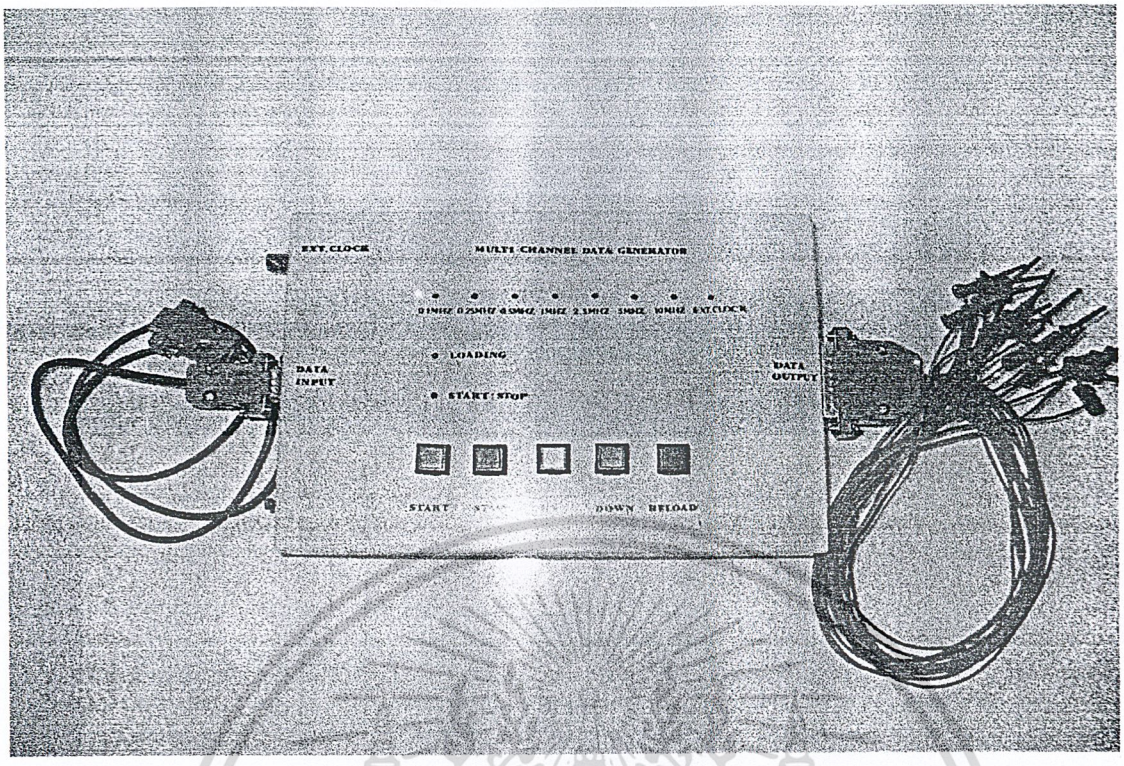
รูปที่ 4.17 แสดงผลการทดลองของวงจร FREQUENCY โดยการกำหนดสัญญาณอินพุทให้กับขาสัญญาณ sel[2..0] ซึ่งจะได้เอาท์พุทที่ขา a,b,c,d,e,f,g และ h ดังรูป



รูปที่ 4.18 แสดงรูปสัญญาณเอาท์พุทที่ได้จากการกำหนดข้อมูลไว้ก่อนแล้ว



รูปที่ 4.19 แสดงรูปสัญญาณเอาท์พุทที่ได้ทดลองกำหนดข้อมูลขึ้นเอง



รูปที่ 4.20 แสดงชิ้นงานที่ได้ประกอบเสร็จแล้ว

บทที่ 5
บทวิจารณ์และบทสรุป

5.1 บทวิจารณ์

ผลการทดลองของวงจรในส่วนต่างๆภายใน ไอซี. เอฟพีจีเอ. สามารถจำลองการทำงานได้บนโปรแกรม แม็ก+พลัส ทู 10.0 การจำลองการทำงานสามารถกำหนดเงื่อนไขของสัญญาณอินพุทตามที่ต้องการ และสัญญาณเอาท์พุทที่ออกมานั้น สามารถนำเวลามาเปรียบเทียบกับเวลาของสัญญาณอินพุท เพื่อหาเวลาที่สัญญาณถูกหน่วงไว้ได้อย่างละเอียด และความเร็วในการทำงานของ เอฟพีจีเอ. มีความเร็วกว่าไมโครคอนโทรลเลอร์มาก จึงไม่มีปัญหาในการเชื่อมต่อ เมื่อใช้ไมโครคอนโทรลเลอร์มาควบคุม ไอซี.เอฟพีจีเอ.

การทำงานของโครงงานนี้ ใช้เวลาในการส่งข้อมูลจากคอมพิวเตอร์มายังไมโครคอนโทรลเลอร์ เพื่อนำไปเก็บจะใช้เวลานาน เนื่องจากข้อมูลที่ถูกส่งมานั้นมีจำนวนมาก

5.2 บทสรุป

โครงงานนี้ถูกออกแบบมาเพื่อตอบสนองความต้องการของผู้ใช้ โดยสามารถกำหนดข้อมูลได้เอง ซึ่งจะสามารถนำไปประยุกต์ใช้งานได้ตามความต้องการ และการออกแบบวงจรดิจิตอล โดยใช้โปรแกรม แม็ก+พลัส ทู 10.0 นั้น แม้ว่าจะออกแบบได้ง่าย แต่เพื่อที่จะให้ได้วงจรที่มีประสิทธิภาพ ผู้ออกแบบควรจะมีพื้นฐานทางด้านวงจรดิจิตอลที่ดี

ภาคผนวก

โปรแกรมส่วนส่งผ่านข้อมูล

```
                    EN_1        BIT P2.7
                    EN_0        BIT P2.6
                    CLK         BIT P2.5
                    WE          BIT P2.4
                    SEL2        BIT P2.2
                    SEL1        BIT P2.1
                    SEL0        BIT P2.0
                    LOADING     BIT P1.0
                    START_LED   BIT P1.1
                    START_KEY   BIT P1.2
                    STOP_KEY    BIT P1.3
                    UP_KEY      BIT P1.4
                    DOWN_KEY    BIT P1.5
                    RELOAD_KEY  BIT P1.6
;*********************************************************************
;*********************************************************************
                    ORG         0000H
        MAIN:       MOV         TMOD,#20H
                    MOV         TH1,#0FDH
                    MOV         PCON,#80H
                    MOV         SCON,#50H
                    SETB        TR1
                    MOV         P0,#00H
                    MOV         R4,#00H
                    MOV         DPTR,#0000H
                    CLR         EN_1
                    CLR         EN_0
                    CLR         CLK
                    SET         WE
                    CLR         SEL2
                    CLR         SEL1
                    CLR         SEL0
                    SET         LOADING
                    CLR         START_LED
                    SETB        START_KEY
                    SETB        STOP_KEY
                    SETB        UP_KEY
                    SETB        DOWN_KEY
                    SETB        RELOAD_KEY
;*********************************************************************
;*********************************************************************
                    ACALL       RX_BYTE
                    MOV         R1,A
        LOOP:       ACALL       SENT_C
                    ACALL       RX_BYTE
                    CJNE        R4,#00H,SENT_LSB
;*********************************************************************
;*********************************************************************
     SENT_MSB:      SETB        EN_0
                    MOV         P0,A
                    CLR         WE
                    SETB        WE
                    SETB        CLK
                    CLR         CLK
                    INC         DPTR
                    MOV         R0,DPH
                    CJNE        R0,#20H,LOOP
```

```
                    MOV         R4,#0FH
                    CLR         EN_0
                    SETB        EN_1
                    MOV         DPTR,#0000H
                    SJMP        LOOP
;*********************************************************************
;*********************************************************************
    SENT_LSB:       MOV         P0,A
                    CLR         WE
                    SETB        WE
                    SETB        CLK
                    CLR         CLK
                    INC         DPTR
                    MOV         R0,DPH
                    CJNE        R0,#20H,LOOP
                    CLR         EN_1
                    CLR         EN_0
                    SETB        WE
                    CLR         LOADING
                    MOV         R1,#00H

;*********************************************************************
;*********************************************************************
       START:       JB          START_KEY,STOP
          AA:       ACALL       DELAY_10mS
                    JNB         START_KEY,AA
                    SETB        START_LED
                    SETB        EN_1
                    SETB        EN_0
                    SJMP        START
;*********************************************************************
;*********************************************************************
        STOP:       JB          STOP_KEY,UP
          BB:       ACALL       DELAY_10mS
                    JNB         STOP_KEY,BB
                    CLR         START_LED
                    CLR         EN_1
                    CLR         EN_0
                    SJMP        START

;*********************************************************************
;*********************************************************************
          UP:       JB          UP_KEY,DOWN
          CC:       ACALL       DELAY_10mS
                    JNB         UP_KEY,CC
                    CJNE        R1,#07H,UP_DO
                    SJMP        START
;*********************************************************************
;*********************************************************************
       UP_DO:       INC         R1
                    MOV         A,P2
                    ANL         A,#0F8H
                    ORL         A,R1
                    MOV         P2,A
                    SJMP        START
;*********************************************************************
;*********************************************************************
        DOWN:       JB          DOWN_KEY,RELOAD
          DD:       ACALL       DELAY_10mS
                    JNB         DOWN_KEY,DD
                    CJNE        R1,#00H,DOWN_DO
```

```
                    SJMP        START
      DOWN_DO:      DEC         R1
                    MOV         A,P2
                    ANL         A,#0F8H
                    ORL         A,R1
                    MOV         P2,A
                    SJMP        START
;*********************************************************************
;*********************************************************************
       RELOAD:      JB          RELOAD_KEY,START
                    LJMP        MAIN
;*********************************************************************
;*********************************************************************
   DELAY_10mS:      MOV         R4,#10
       DELAY1:      MOV         R3,#0E6H
       DELAY2:      NOP
                    NOP
                    DJNZ        R3,DELAY2
                    DJNZ        R4,DELAY1
                    RET
;*********************************************************************
;*********************************************************************
      RX_BYTE:      JNB         RI,$
                    CLR         RI
                    MOV         A,SBUF
                    RET
;*********************************************************************
;*********************************************************************
       SENT_C:      MOV          SBUF,A
                    JNB          TI,$
                    CLR          TI
                    RET

                    END
```

โปรแกรมส่วนควบคุมและกำเนิดสัญญาณ

1. โปรแกรม SINGLE_BUFF

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity single_buff is
      port(en    : in  std_logic;
           inn   : in  std_logic_vector( 7 downto 0);
           outt  : out std_logic_vector(7 downto 0));
end single_buff;

 Architecture behav of single_buff is
      begin
            process (en,inn)
                  begin
                        if (en = '1') then
                                    outt<= (others=>'Z');
                        else
                                    outt<= inn;
                  end if;
            end process;
 end behav;
```

2. โปรแกรม CE

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
Entity ce is
      port(ce1,ce0 :out  std_logic;
               sel1:in   bit;
               sel0:in   bit);
end ce;
Architecture behav of ce is
      begin
            process(sel1,sel0)
                  begin
                        if sel1='0' then
                              if sel0='0'
                                    then ce1<='1';
                                         ce0<='1';
                                    else ce1<='1';
                                         ce0<='0';
                              end if;
                         else
                              if sel0='0'
                                    then ce1<='0';
                                         ce0<='1';
                                    else ce1<='0';
                                         ce0<='0';
                                    end if;
                              end if;
            end process;
end behav;
```

3. โปรแกรม MUX21

```
Library ieee;
use ieee.std_logic_1164.all;

Entity mux21 is
      port(a,b : in   std_logic;
           sel : in   std_logic ;
           q   : out  std_logic);
end mux21;

Architecture behav of mux21 is
      begin
            process(a,b,sel)
                  begin
                        case sel is
                              when '1'=>q<=a;
                              when others=>q<=b;
                        end case;
            end process;
end behav;
```

4. โปรแกรม COUNT8KBYTE

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity count8kbyte is
       port(clk : in    bit;
            q   : out   std_logic_vector(12 downto 0);
            en  : in    bit );
end count8kbyte;

Architecture behav of count8kbyte is
      begin
            process(clk,en)
                        variable counter  : integer range 0 to 8191;
                  begin
                        if (clk'event) and (clk = '1') then
                         if (en = '1') then
                            counter := 0;
                            q <= conv_std_logic_vector(counter,13);
                         else
                            counter := counter+1;
                            q <= conv_std_logic_vector(counter,13);
                         end if;
                        end if;
            end process;
end behav;
```

5.โปรแกรม MUX8

```
Library ieee;
use ieee.std_logic_1164.all;

Entity mux8 is
      port(a,b,c,d,e,f,g,h : in std_logic;
                        sel : in std_logic_vector(2 downto 0);
                        q   : out std_logic);
end mux8;

Architecture behav of mux8 is
      begin
            process(a,b,c,d,e,f,g,h,sel)
                  begin
                        case sel is
                              when "000"=>q<=a;
                              when "001"=>q<=b;
                              when "010"=>q<=c;
                              when "011"=>q<=d;
                              when "100"=>q<=e;
                              when "101"=>q<=f;
                              when "110"=>q<=g;
                              when others=>q<=h;
                        end case;
            end process;
end behav;
```

6. โปรแกรม DIV2

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

Entity div2 is
      port(clk : in   bit;
          div2 : out std_logic;
          div4 : out std_logic);
end div2;

Architecture behav of div2 is
      begin
       process(clk)
            variable counter  : integer range 0 to 3;
            variable q :std_logic_vector (1 downto 0);
             begin
                  if (clk'event) and (clk = '1') then
                     counter := counter+1;
                  end if;
                  q:=conv_std_logic_vector(counter,2);
                  div2<=q(0);
                  div4<=q(1);
       end process;
end behav;
```

7. โปรแกรม DIV10

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

Entity div10 is
      port(clk : in  bit;
           q   : out std_logic);
end div10;

Architecture behav of div10 is
      begin
            process(clk)
                  variable counter : integer range 0 to 9;
                begin
                  if (clk'event) and (clk = '1') then
                        counter := counter+1;
                     if  counter = 10 then
                        counter := 0;
                     end if;
                     if  counter>4 then
                        q<= '1';
                     else
                        q<= '0';
                     end if;
                  end if;
            end process;
end behav;
```

8. โปรแกรม FREQUENCY

```
Library ieee;
use ieee.std_logic_1164.all;

Entity frequency is
      port(a,b,c,d,e,f,g,h : out  std_logic;
                        sel : in   std_logic_vector(2 downto 0));
end frequency;

Architecture behav of frequency is
  begin
     process(sel)
       begin
         case sel is
when "000"=> a<='1';b<='0';c<='0';d<='0';e<='0';f<='0';g<='0';h<='0';
when "001"=> a<='0';b<='1';c<='0';d<='0';e<='0';f<='0';g<='0';h<='0';
when "010"=> a<='0';b<='0';c<='1';d<='0';e<='0';f<='0';g<='0';h<='0';
when "011"=> a<='0';b<='0';c<='0';d<='1';e<='0';f<='0';g<='0';h<='0';
when "100"=> a<='0';b<='0';c<='0';d<='0';e<='1';f<='0';g<='0';h<='0';
when "101"=> a<='0';b<='0';c<='0';d<='0';e<='0';f<='1';g<='0';h<='0';
when "110"=> a<='0';b<='0';c<='0';d<='0';e<='0';f<='0';g<='1';h<='0';
when others=>a<='0';b<='0';c<='0';d<='0';e<='0';f<='0';g<='0';h<='1';
         end case;
end process;

end behav;
```

IS61C64B

ISSI®

8K x 8 HIGH-SPEED CMOS STATIC RAM

July 2002

FEATURES

- High-speed access time: 10, 12, and 15 ns
- Automatic power-down when chip is deselected
- CMOS low power operation
  - — 450 mW (typical) operating
  - — 250 µW (typical) standby
- TTL compatible interface levels
- Single 5V power supply
- Fully static operation: no clock or refresh required
- Three state outputs
- One Chip Enables ($\overline{CE}$) for increased speed

DESCRIPTION

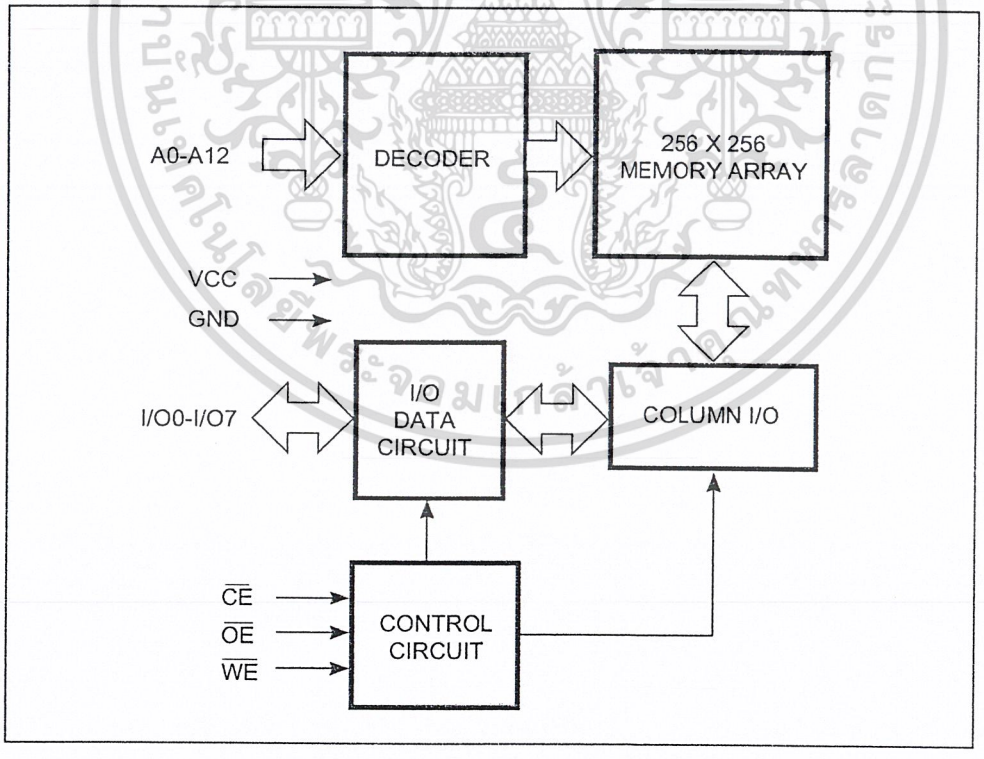
The *ISSI* IS61C64B is a very high-speed, low power, 8192-word by 8-bit static RAM. It is fabricated using *ISSI*'s high-performance CMOS technology. This highly reliable process coupled with innovative circuit design techniques, yields access times as fast as 10 ns with low power consumption.

When $\overline{CE}$ is HIGH (deselected), the device assumes a standby mode at which the power dissipation can be reduced down to 250 µW (typical) with CMOS input levels.

Easy memory expansion is provided by using one Chip Enable input, $\overline{CE}$. The active LOW Write Enable ($\overline{WE}$) controls both writing and reading of the memory.

The IS61C64B is packaged in the JEDEC standard 28-pin, 300-mil SOJ, and TSOP.

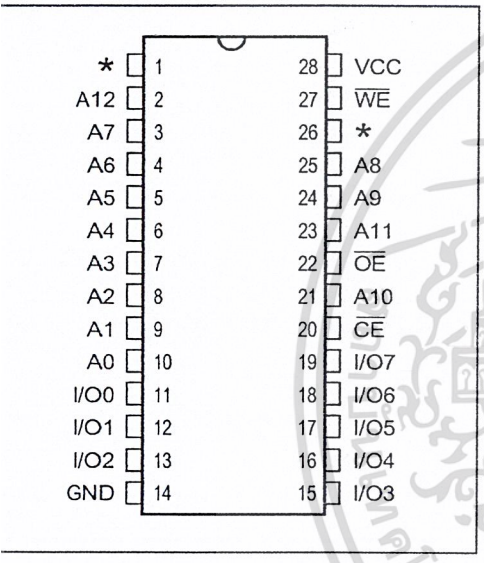
FUNCTIONAL BLOCK DIAGRAM



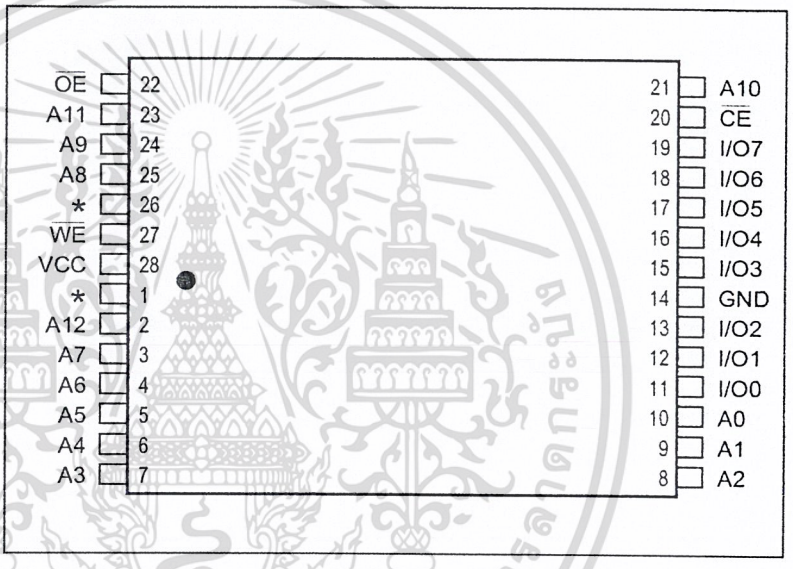
TRUTH TABLE

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	I/O Operation	Vcc Current
Not Selected	X	H	X	High-Z	I_{SB1}, I_{SB2}
(Power-down)	X	X	X	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	High-Z	I_{CC}
Read	H	L	L	D_{OUT}	I_{CC}
Write	L	L	X	D_{IN}	I_{CC}

PIN CONFIGURATION
28-Pin SOJ



PIN CONFIGURATION
28-Pin TSOP (Type 1)



PIN DESCRIPTIONS

A0-A12	Address Inputs
$\overline{CE}$	Chip Enable 1 Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
I/O0-I/O7	Input/Output
*	Must be tied to either Vcc or GND
Vcc	Power
GND	Ground

ABSOLUTE MAXIMUM RATINGS[1]

Symbol	Parameter	Value	Unit
V_{TERM}	Terminal Voltage with Respect to GND	–0.5 to +7.0	V
T_{BIAS}	Temperature Under Bias	–10 to +85	°C
T_{STG}	Storage Temperature	–65 to +150	°C
P_T	Power Dissipation	1.0	W
I_{OUT}	DC Output Current (LOW)	20	mA

Notes:

1. Stress greater than those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

OPERATING RANGE

Range	Ambient Temperature	Speed	V_{CC}
Commercial	0°C to +70°C	10 ns	5V ± 5%
		12 ns	5V ± 10%
		15 ns	5V ± 10%

DC ELECTRICAL CHARACTERISTICS (Over Operating Range)

Symbol	Parameter	Test Conditions	Min.	Max.	Unit
V_{OH}	Output HIGH Voltage	V_{CC} = Min., I_{OH} = –4.0 mA	2.4	—	V
V_{OL}	Output LOW Voltage	V_{CC} = Min., I_{OL} = 8.0 mA	—	0.4	V
V_{IH}	Input HIGH Voltage		2.2	V_{CC} + 0.5	V
V_{IL}	Input LOW Voltage[1]		–0.5	0.8	V
I_{LI}	Input Leakage	GND - V_{IN} - V_{CC}	–2	2	µA
I_{LO}	Output Leakage	GND - V_{OUT} - V_{CC}, Outputs Disabled	–2	2	µA

Notes:

1. V_{IL} = –3.0V for pulse width less than 10 ns.

POWER SUPPLY CHARACTERISTICS[1] (Over Operating Range)

Symbol	Parameter	Test Conditions	-10ns Min.	-10ns Max.	-12ns Min.	-12ns Max.	-15ns Min.	-15ns Max.	Unit
I_{CC}	Vcc Dynamic Operating Supply Current	Vcc = Max., I_{OUT} = 0 mA, f = f_{MAX}	—	185	—	175	—	135	mA
I_{SB1}	TTL Standby Current (TTL Inputs)	Vcc = Max., V_{IN} = V_{IH} or V_{IL} $\overline{CE1}$ • V_{IH} or CE2 - V_{IL}, f = 0	—	30	—	30	—	30	mA
I_{SB2}	CMOS Standby Current (CMOS Inputs)	Vcc = Max., $\overline{CE1}$ • Vcc – 0.2V, CE2 - 0.2V, V_{IN} • Vcc – 0.2V, or V_{IN} - 0.2V, f = 0	—	10	—	10	—	10	mA

Notes:
1. At f = f_{MAX}, address and data inputs are cycling at the maximum frequency, f = 0 means no input lines change.

CAPACITANCE[1,2]

Symbol	Parameter	Conditions	Max.	Unit
C_{IN}	Input Capacitance	V_{IN} = 0V	8	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V	10	pF

Notes:
1. Tested initially and after any design or process changes that may affect these parameters.
2. Test conditions: T_A = 25°C, f = 1 MHz, Vcc = 5.0V.

READ CYCLE SWITCHING CHARACTERISTICS[1] (Over Operating Range)

Symbol	Parameter	-10ns Min.	-10ns Max.	-12ns Min.	-12ns Max.	-15ns Min.	-15ns Max.	Unit
t_{RC}	Read Cycle Time	10	—	12	—	15	—	ns
t_{AA}	Address Access Time	—	10	—	12	—	15	ns
t_{OHA}	Output Hold Time	2	—	2	—	2	—	ns
t_{ACE}	$\overline{CE}$ Access Time	—	10	—	12	—	15	ns
t_{DOE}	$\overline{OE}$ Access Time	—	5	—	6	—	7	ns
t_{LZOE}[2]	$\overline{OE}$ to Low-Z Output	0	—	0	—	0	—	ns
t_{HZOE}[2]	$\overline{OE}$ to High-Z Output	—	5	—	6	—	6	ns
t_{LZCE1}[2]	$\overline{CE}$ to Low-Z Output	2	—	3	—	3	—	ns
t_{HZCE}[2]	$\overline{CE}$ to High-Z Output	—	5	—	7	—	8	ns

Notes:
1. Test conditions assume signal transition times of 5 ns or less, timing reference levels of 1.5V, input pulse levels of 0 to 3.0V and output loading specified in Figure 1a.
2. Tested with the load in Figure 1b. Transition is measured ±500 mV from steady-state voltage. Not 100% tested.

AC TEST CONDITIONS

Parameter	Unit
Input Pulse Level	0V to 3.0V
Input Rise and Fall Times	3 ns
Input and Output Timing and Reference Level	1.5V
Output Load	See Figures 1a and 1b

AC TEST LOADS

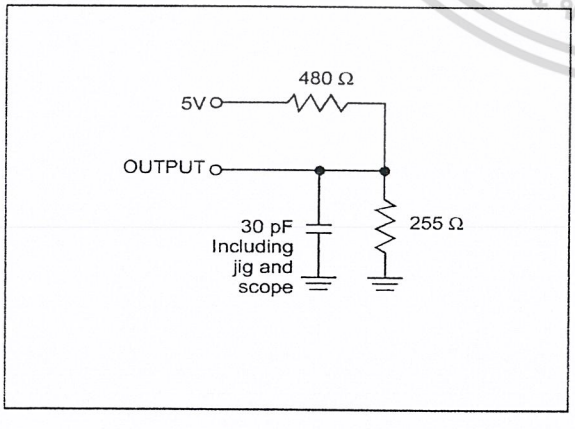


Figure 1a.

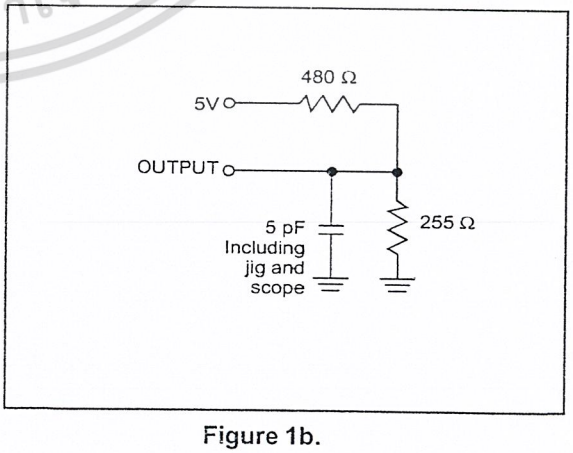
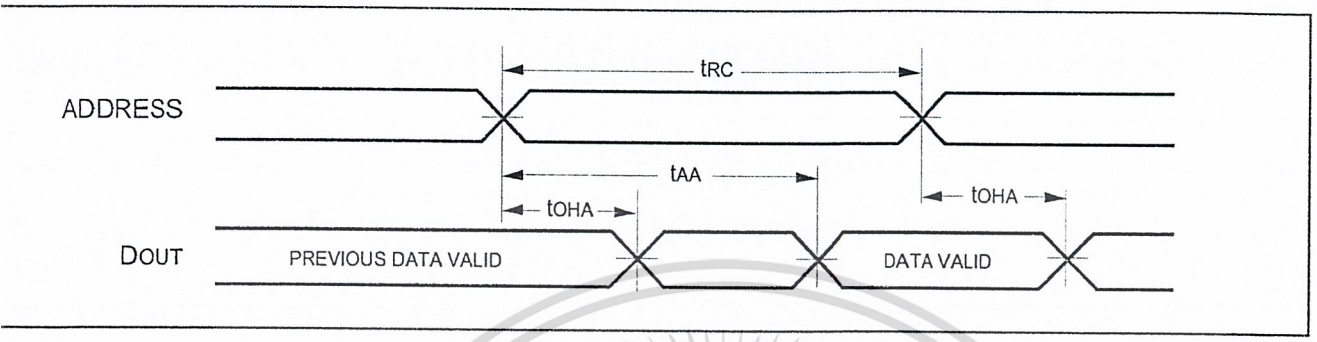


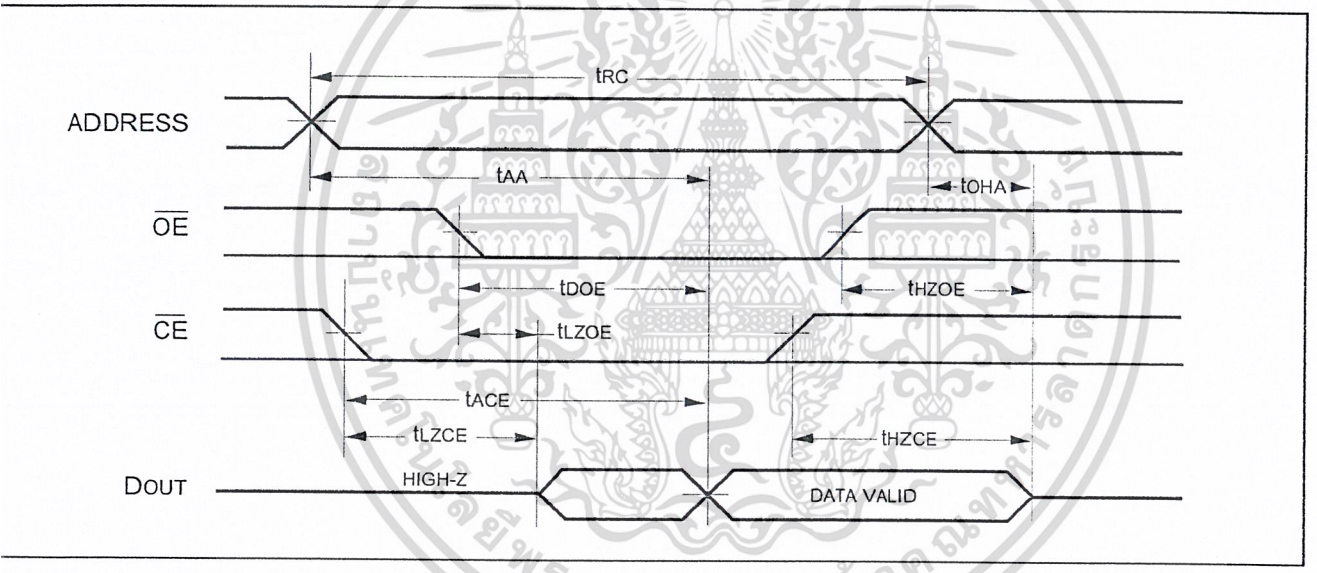
Figure 1b.

AC WAVEFORMS

READ CYCLE NO. 1[1,2]



READ CYCLE NO. 2[1,3]



Notes:
1. $\overline{WE}$ is HIGH for a Read Cycle.
2. The device is continuously selected. $\overline{OE}$, $\overline{CE}$ = V_{IL}.
3. Address is valid prior to or coincident with $\overline{CE}$ LOW transitions.

WRITE CYCLE SWITCHING CHARACTERISTICS[1,3] (Over Operating Range)

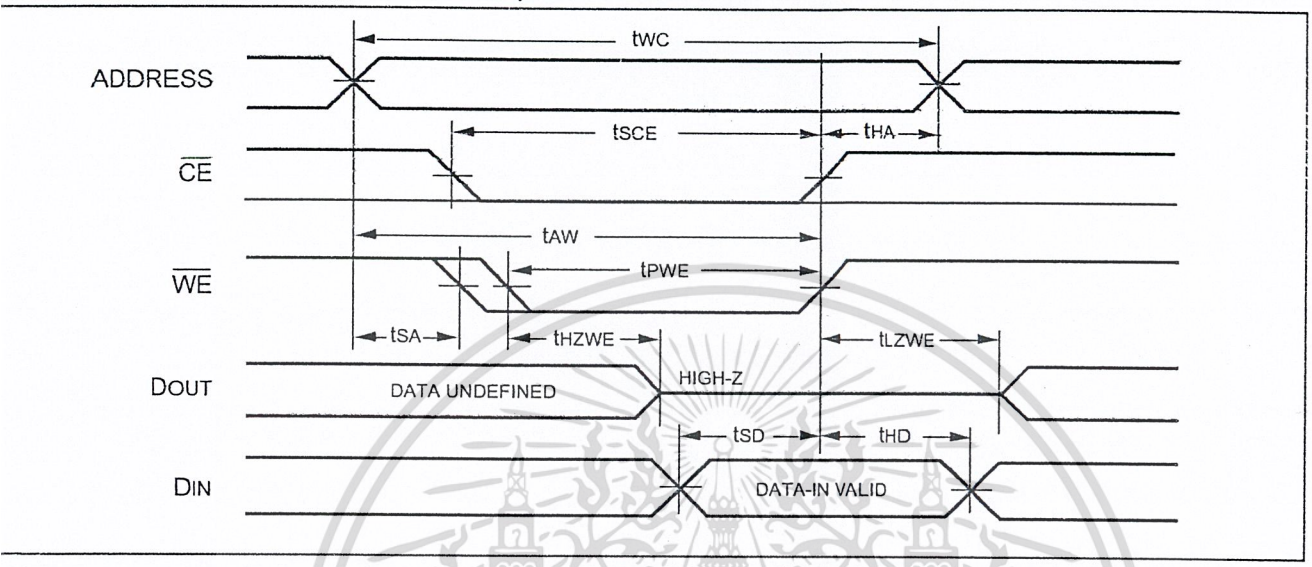
Symbol	Parameter	-10ns Min.	-10ns Max.	-12ns Min.	-12ns Max.	-15ns Min.	-15ns Max.	Unit
t_{WC}	Write Cycle Time	10	—	12	—	15	—	ns
t_{SCE}	$\overline{CE}$ to Write End	9	—	10	—	12	—	ns
t_{AW}	Address Setup Time to Write End	9	—	10	—	12	—	ns
t_{HA}	Address Hold from Write End	0	—	0	—	0	—	ns
t_{SA}	Address Setup Time	0	—	0	—	0	—	ns
t_{PWE}[4]	$\overline{WE}$ Pulse Width	8	—	8	—	10	—	ns
t_{SD}	Data Setup to Write End	8	—	8	—	9	—	ns
t_{HD}	Data Hold from Write End	0	—	0	—	0	—	ns
t_{HZWE}[2]	$\overline{WE}$ LOW to High-Z Output	—	6	—	6	—	7	ns
t_{LZWE}[2]	$\overline{WE}$ HIGH to Low-Z Output	0	—	0	—	0	—	ns

Notes:

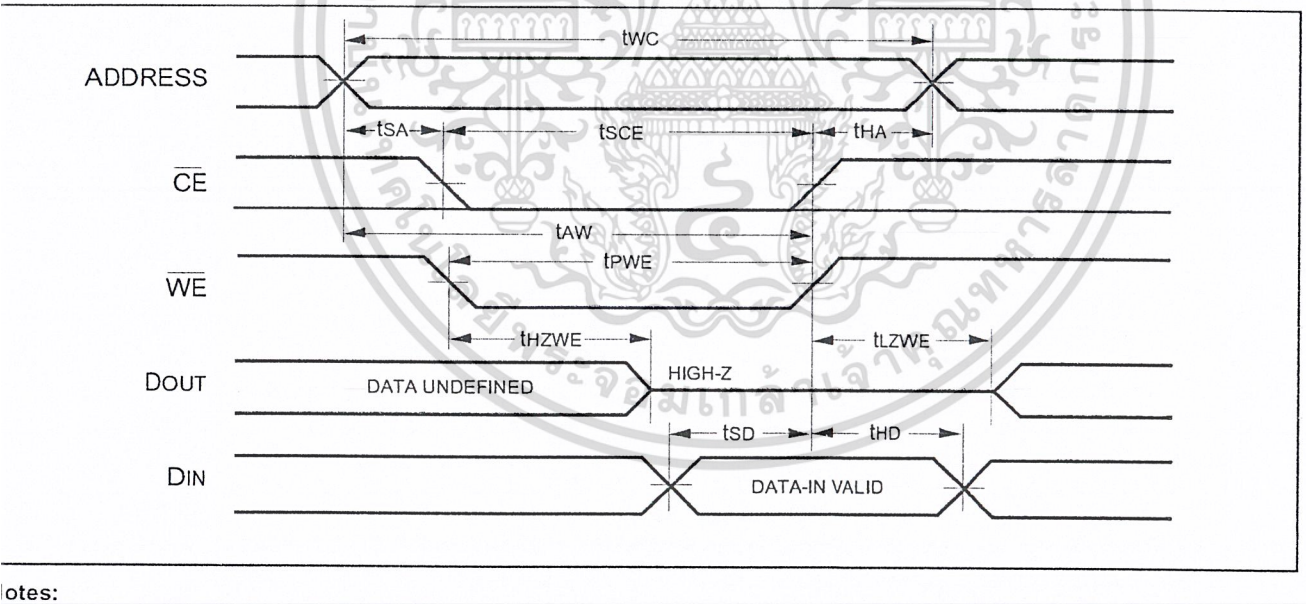
1. Test conditions assume signal transition times of 5 ns or less, timing reference levels of 1.5V, input pulse levels of 0 to 3.0V and output loading specified in Figure 1a.
2. Tested with the load in Figure 1b. Transition is measured ±500 mV from steady-state voltage. Not 100% tested.
3. The internal write time is defined by the overlap of $\overline{CE}$ LOW and $\overline{WE}$ LOW. All signals must be in valid states to initiate a Write, but any one can go inactive to terminate the Write. The Data Input Setup and Hold timing are referenced to the rising or falling edge of the signal that terminates the write.

AC WAVEFORMS

WRITE CYCLE NO. 1 ($\overline{WE}$ Controlled)[1,2]



WRITE CYCLE NO. 2 ($\overline{CE1}$, CE2 Controlled)[1,2]



Notes:

1. The internal write time is defined by the overlap of $\overline{CE}$ LOW and $\overline{WE}$ LOW. All signals must be in valid states to initiate a Write, but any one can go inactive to terminate the Write. The Data Input Setup and Hold timing are referenced to the rising or falling edge of the signal that terminates the write.
2. I/O will assume the High-Z state if $\overline{OE}$ = V_{IH}.

ORDERING INFORMATION

Commercial Range: 0°C to +70°C

Speed (ns)	Order Part No.	Package
10	IS61C64B-10J	300-mil Plastic SOJ
	IS61C64B-10T	Plastic TSOP
12	IS61C64B-12J	300-mil Plastic SOJ
	IS61C64B-12T	Plastic TSOP
15	IS61C64B-15J	300-mil Plastic SOJ
	IS61C64B-15T	Plastic TSOP

หนังสืออ้างอิง

[1] Stafan sjonhom and Lenart Lindh, "VHDL for Designers" : Prentice Hall : Europe,1997.

[2] สัจจะ จรัสรุ่งวรีวร , " Visual Basic 6" , อินโฟเพรส : กรุงเทพ , พ.ศ. 2544.

[3] สมยศ จุณณะปิยะ , "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS – 51" : สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง : กรุงเทพ , 2543.