บอร์ดเก็บข้อมูล

DATA LOGGER

โดย

นาย โสพส โพธิ์ชินณรงค์

นาย อานนท์ จิตต์มงคล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

บอร์ดเก็บข้อมูล

DATA LOGGER

โดย

นายโสพส โพธิ์ชินณรงค์

นาย อานนท์ จิตต์มงคล

อาจารย์ที่ปรึกษา

รศ. ดร. มนัส สังวรศิลป์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ ปีการศึกษา 2545

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่องบอร์ดเก็บข้อมูล *(DATA LOGGER)*

ผู้จัดทำ

1. นายโสพส โพธิ์ชินณรงค์ 43015288
2. นาย อานนท์ จิตต์มงคล 43015293

..... อาจารย์ที่ปรึกษา

(รศ. ดร. มนัส สังวรศิลป์)

บอร์ดเก็บข้อมูล

นายโสพส โพธิ์ชินณรงค์
นาย อานนท์ จิตต์มงคล
รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

บอร์ดเก็บข้อมูล เป็นอุปกรณ์ที่ทำหน้าที่บันทึกสัญญาณอนาลอกที่ได้จากตัวตรวจจับ เพื่อนำข้อมูลความเปลี่ยนแปลงเหล่านี้ ถ่ายโอนให้กับคอมพิวเตอร์ ซึ่งสัญญาณอนาลอกที่ได้จากตัวจับ จะถูกแปลงเป็นข้อมูลดิจิตอลด้วยไอซี เอดีซี 0808 โดยมันจะทำงานร่วมกับ เอฟพีจีเอ ภาษาที่ใช้เขียนโปรแกรม คือ ภาษา วีเอชดีเอล (VHDL : VHSIC Hardware description Language) โดยโปรแกรมทั้งหมดจะถูกสังเคราะห์ แล้วนำผลลัพธ์ที่ได้โปรแกรมลงบน เอฟพีจีเอ ข้อมูลที่ได้จะส่งให้คอมพิวเตอร์ เพื่อนำข้อมูลนี้ไปวิเคราะห์ และใช้งานต่อไป

DATA LOGGER

Mr. Sorot pochinnarong
Mr. Arnon Jitmongkhol
Assoc. Prof. Manus Sangwarasilp (Advisor)
Educational Year 2002

Abstract

The data logger's function is record the alteration of environment such as temperature, light and humidity by records the output signal of sensor for transfers the data of these alteration to a personal computer which analog signal from a sensor are converted to digital signal by IC ADC 0808 . It operate together with FPGA (Filed Programmable gate array) and use VHDL : (VHSIC Hardware description Language). VHDL is used to synthesize this program such that the circuit can be implemented by using chip technology . After finished the convert it will be connected to a personal computer to allow its collected data to be transferred , analyzed and use.

สารบัญ

สารบัญรูป

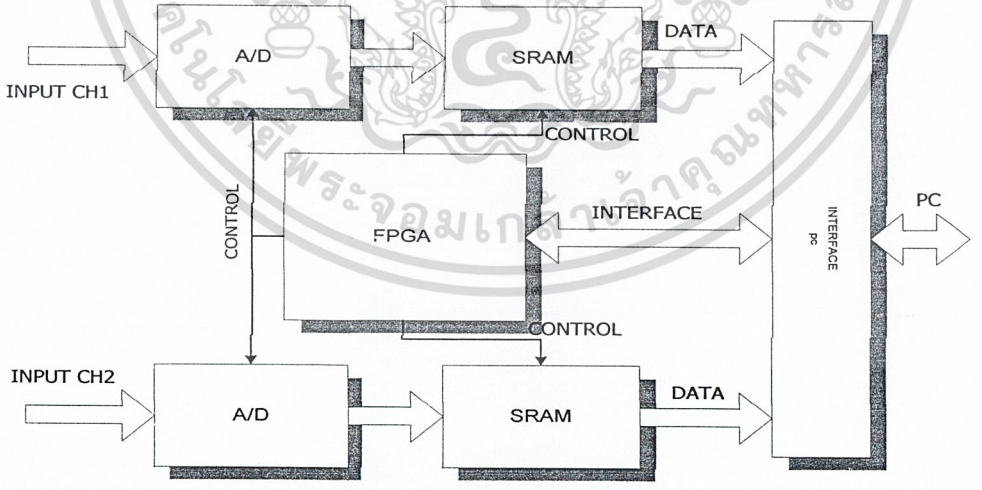
หน้า

บทที่ 1
บทนำ

บอร์ดเก็บข้อมูล (Data Logger) เป็นระบบที่จัดเก็บข้อมูลต่าง ๆ บนตัวตรวจจับ (Sensor) ให้มาแสดงผลบนคอมพิวเตอร์ที่มีการใช้งานอย่างกว้างขวาง ในโรงงานอุตสาหกรรมและการวิจัยทางวิทยาศาสตร์ เพื่อแสดงผลปริมาณทางฟิสิกส์ ซึ่งได้แก่อุณหภูมิ ความดัน ความสว่าง และระดับสัญญาณเสียง เป็นต้น

โดยที่มันจะทำการรับสัญญาณอนาลอกเข้ามาแล้วเปลี่ยนให้เป็นสัญญาณดิจิตอลการจัดเก็บข้อมูลนั้นจะถูกเก็บลงหน่วยความจำแบบ Static Ram ทั้ง 2 ช่อง พร้อมกันซึ่งเวลาในการจัดเก็บได้นานถึง 1-3 วันขึ้นอยู่กับการนำไปใช้งาน แล้วทำการนำบอร์ดมาต่อเข้ากับคอมพิวเตอร์โดยใช้โปรแกรมที่เขียนขึ้นมาเก็บข้อมูลนี้มาตรวจสอบเพื่อศึกษาข้อมูลสภาพแวดล้อมและควบคุมให้เป็นไปตามที่ต้องการได้ สำหรับ Data Logger ตัวนี้จะใช้วงจรแปลงสัญญาณอนาลอกขนาด 8 บิท และต่อสัญญาณอินพุทเข้ามา 2 ช่อง พร้อมกัน โดยมันจะทำงานร่วมกันกับ FPGA ในการควบคุมการทำงานของมันรวมถึง RAM ด้วย และแสดงผลออกคอมพิวเตอร์โดยใช้ภาษา VHDL และในส่วนของการแสดงผลทางคอมพิวเตอร์ใช้ Delphi สำหรับเขียนโปรแกรมเพื่อทำหน้าที่ รับข้อมูลดิจิตอลเพื่อนำไปศึกษาและวิเคราะห์ต่อไป

ซึ่งบล็อกไดอะแกรมของโครงงานนี้ประกอบด้วยส่วนต่าง ๆ ดังรูป

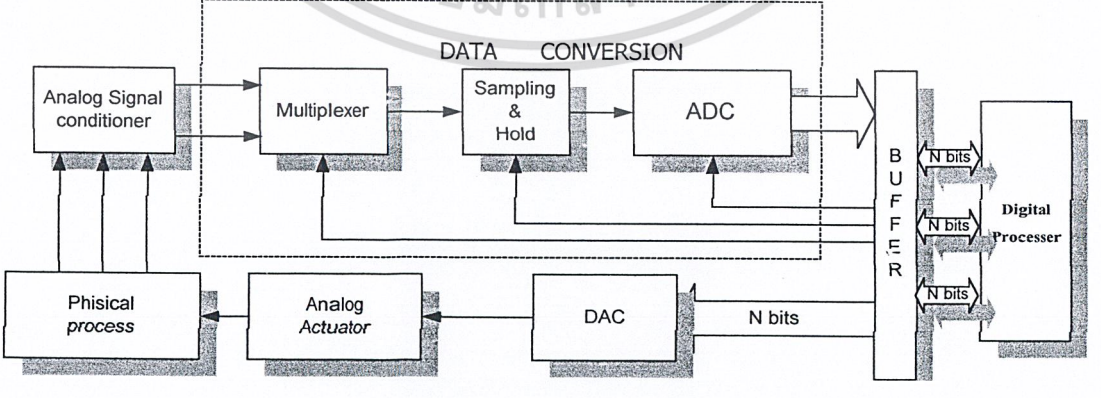


รูปที่ 1.1 บล็อกไดอะแกรมบอร์ดเก็บข้อมูล

บทที่ 2
ทฤษฎีและหลักการทำงาน

2.1 การแปลงสัญญาณอนาลอกเป็นดิจิตอล

รูปแบบสัญญาณไฟฟ้าที่เราพบเห็นและคุ้นเคยในชีวิตประจำวันจะอยู่ในรูปของสัญญาณที่ต่อเนื่องหรือที่เรียกว่า สัญญาณอนาลอก (Analog signal) ซึ่งแต่เดิมการนำเอาสัญญาณไฟฟ้าดังกล่าวมาประมวล (Process) เพื่อให้มีรูปแบบที่เหมาะสมจะกระทำในแบบอนาลอกนั่นเอง แต่เมื่อเทคนิคและอุปกรณ์การประมวลสัญญาณทางสัญญาณดิจิตอลได้รับการพัฒนาขึ้นมา เนื่องจากพบในรูปแบบดิจิตอล การประมวลผล เก็บ สื่อสาร และการนำเสนอ กระทำได้ง่ายและอย่างมีประสิทธิภาพมากกว่า ดังนั้นการเปลี่ยนรูปแบบของสัญญาณ (conversion) จึงได้มีความจำเป็นขึ้นมา ในรูปที่ 1.1 เป็นตัวอย่างแสดงระบบควบคุมที่ใช้ในการประมวลข้อมูลในระบบดิจิตอลในระบบที่ยกมาเป็นตัวอย่างนี้การเปลี่ยนแปลงทางกายภาพในลักษณะใดๆก็ตาม (physical process) เช่น ความดัน อุณหภูมิ ฯลฯ จะถูกเปลี่ยนเป็นสัญญาณไฟฟ้าที่มีความต่อเนื่อง (สัญญาณอนาลอก) โดยทรานสดิวเซอร์ที่มีคุณสมบัติเหมาะสมกับรูปแบบทางกายภาพนั้น สัญญาณไฟฟ้าจะถูกปรับให้อยู่ในรูปและที่ขนาดที่เหมาะสมก่อนโดย Analog signal conditioner ซึ่งอาจจะเป็นจะเป็นวงจรขยายหรือฟิลเตอร์ เป็นต้น ADC จะทำหน้าที่เปลี่ยนรูปแบบของสัญญาณจากอนาลอกเป็นดิจิตอลตัวประมวลผลทางดิจิตอล ตัวประมวลผลทางดิจิตอล (Digital process) เช่น คอมพิวเตอร์จะจัดการกับข้อมูลเพื่อนำเสนอหรือถูกเปลเปลี่ยนกลับมาอยู่ในรูปแบบอนาลอกโดย DAC เพื่อป้อนกลับไปควบคุม Physical Process



รูปที่ 2.1 ระบบควบคุมที่มีการประมวลข้อมูลแบบดิจิตอล

ในระบบที่มีข้อมูลที่ต้องประมวลในเวลาเดียวกันหลายข้อมูลหาก DAC ทำงานได้พอจะไม่จำเป็นต้องใช้ DAC หลายๆ ตัวทำงานแยกกันสำหรับข้อมูลแต่ละชุด แต่จะใช้วิธีแบ่งเวลา (timesharing) โดยวิธี multiplexing (รูป2.1) วงจร sampling and hold (S/H) จะสุ่ม (sample) ขนาดของสัญญาณอนาลอก มาและเก็บ (hold) ไว้ชั่วขณะเพื่อรอให้ ADC รับเปลี่ยนให้เป็นสัญญาณดิจิตอลจนเรียบร้อยแล้วค่อยสุ่มสัญญาณใหม่ ทั้งนี้เพื่อที่ไม่จำเป็นต้องใช้ ADC ที่มีความเร็วสูง ราคาแพง ข้อมูลดิจิตอลจะถูกส่งต่อไปยัง System bus และถูกประมวลโดย Processor ผลของการประมวลผลจะถูกส่งกลับออกมาเพื่อเปลี่ยนกลับเป็นสัญญาณอนาลอกโดย DAC เพื่อไปควบคุมกิจกรรมทางกายภาพของระบบผ่าน Analog actuator

วงจร A/D มีด้วยกันหลายแบบ เช่น

-แบบสโลปคู่ (Dual Slope)

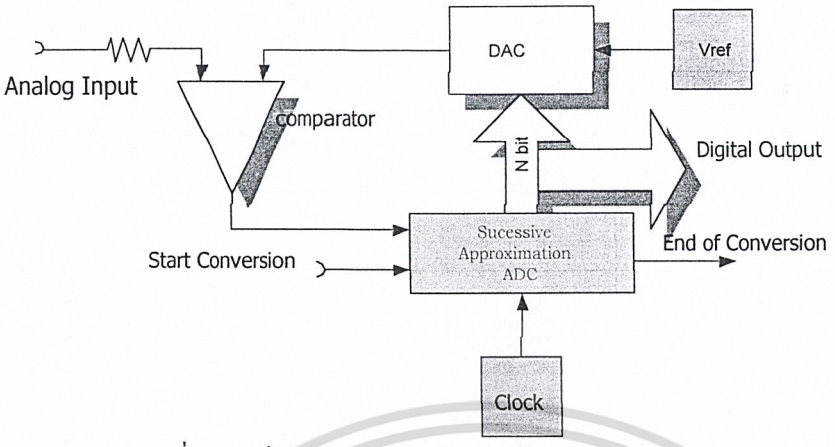
-แบบแปลงแรงดันเป็นความถี่ (V to F Converter)

-แบบประมาณทีละบิด (Successive Approximation)

-แบบแฟลช (Flash)

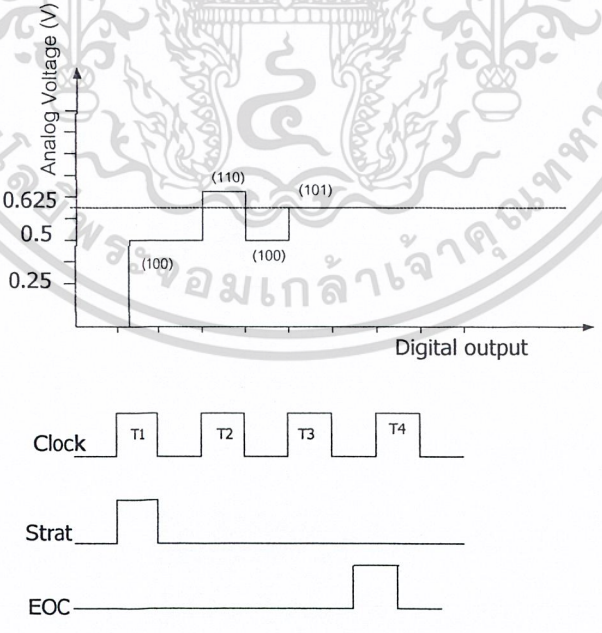
2.2 วงจร A/D แบบประมาณทีละบิด (successive approximation)

วงจร ADC ชนิดนี้ได้รับความนิยมในงานประยุกต์ที่ต้องการความเร็วปานกลางและค่อนข้างสูง การจัดวงจรจะคล้ายกันกับแบบเคาน์เตอร์ ที่ทำงานในลักษณะการป้อนกลับ ซึ่งบล็อกไดอะแกรมในรูปที่ 2.2 แสดงฟังก์ชันต่าง ๆใน ADC ชนิดนี้ คอมพาราเตอร์จะคอยเปรียบเทียบเอาท์พุท จาก DAC กับ อนาลอกอินพุท V_{in} เอาท์พุทจะไปควบคุม Successive Approximation register (SAR) ซึ่งเป็น ไอซี MSI (Medium Scale Integrated circuit) ที่ได้รับการออกแบบเป็นพิเศษเพื่อทำหน้าที่นี้โดยเฉพาะ



รูปที่ 2.2 บล็อกไดอะแกรมของ Sucessive approximation ADC

ในรูปที่ 2.3 แสดงไทมิ่งไดอะแกรมของ ADC ที่มีระดับอนาลอก 0.625 Volt เมื่อ clock เข้าไป 1 ลูก จะทำให้ MSB (most significant bit) (บิท 4) เป็น 1 บิทอื่นยังคงเป็นศูนย์ DAC จะเปลี่ยนเอาท์พุทของ SAR เป็นอนาลอก 1 ไว้ แต่ถ้ามากกว่าจะให้บิทนั้นเป็น 0 จากนั้นทำการทดสอบบิทถัดไปโดยทำให้เป็น 1 หากผลรวมของสองบิทหรือบิทหลังมากกว่าก็ทำให้บิทนั้นเป็น 0 แต่น้อยกว่าให้คง 1 ไว้ แล้วทดสอบบิทถัดไปตามกรรมวิธีดังกล่าวจนครบทุกบิทหรือจนกว่าเอาท์พุทจะต่างจาก V_{in} ไม่เกิน 1 LSB



รูปที่ 2.3 Timing Diagram ของ SAR

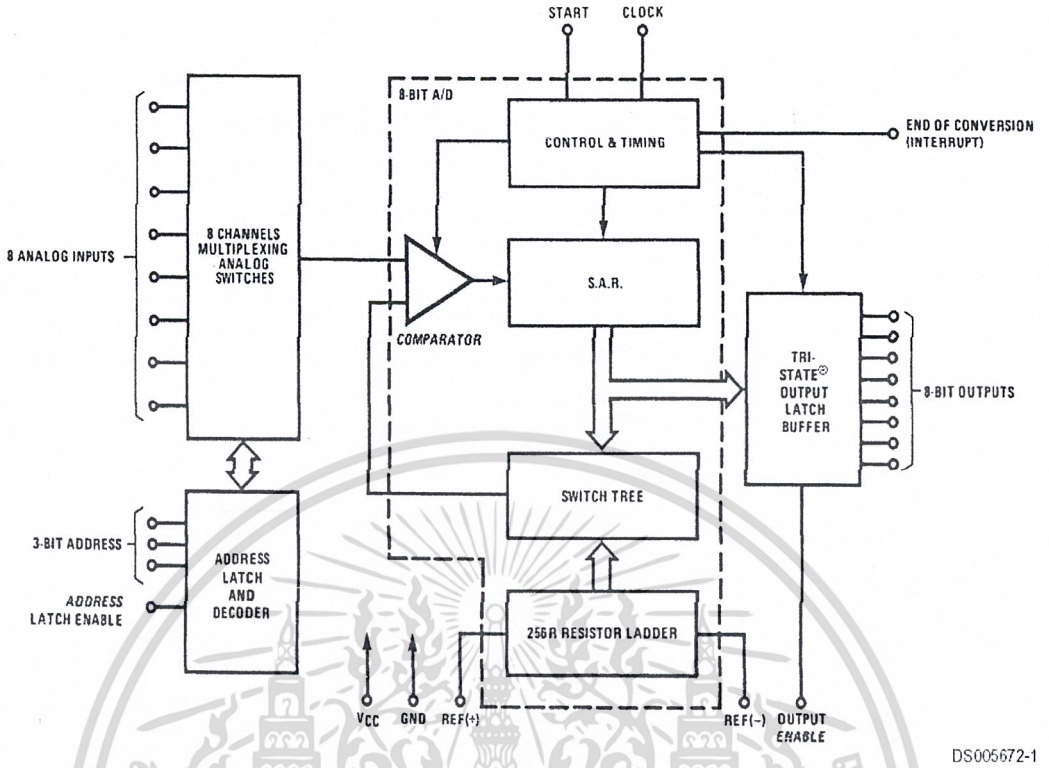
การทำงานของ ADC แบบนี้เปรียบเทียบได้กับการใช้งานของตาชั่งสองแขน เมื่อวัตถุที่ต้องการทราบน้ำหนักเสมือนเป็นอินพุทของ ADC และ เอาท์พุทที่เป็นดิจิตอลบิท เสมือนเป็นตุ้มน้ำหนักมาตรฐานที่จะวางบนจานอีกข้างหนึ่งเมื่อตาชั่งยังไม่สมดุล จะต้องมีการปรับตุ้มน้ำหนักมาตรฐานจนกว่าจะเกิดสมดุล ในรูป 2.2 คอมพาราเตอร์จะเป็นตัวตรวจสอบการสมดุลดังกล่าว และ SAR จะทำหน้าที่ปรับแต่งดิจิตอลบิท (ตุ้มน้ำหนักมาตรฐาน)

มีข้อจำกัดประการหนึ่งสำหรับการ Conversion คือสัญญาณอนาลอกอินพุท จะต้องคงที่ในช่วงเวลาที่ทำการเปลี่ยนแปลงสัญญาณโดยเปลี่ยนได้ไม่เกิน ½ LBS ในช่วงสุดท้ายของการเปลี่ยนสัญญาณดิจิตอลเอาท์พุทจะออกมาขนานกันทุกบิท แต่บางแบบจะให้เอาท์พุทออกมาในลักษณะอนุกรม วงจร ADC แบบนี้สามารถทำงานได้สองโหมด คือ โหมดที่ทำงานโดยอิสระ (Free run) และโหมดที่รอคำสั่ง start conversion จากภายนอก เวลาที่ใช้ในการเปลี่ยนสัญญาณใช้ *(n+1) ลูกของพัลส์ clock ลูกแรกจะใช้ในการ รีเซ็ทรีเจสเตอร์ภายใน สุดท้าย* คุณภาพของระบบจะขึ้นอยู่กับคุณภาพของ DAC ในระบบเป็นอย่างยิ่ง

2.3 ADC 0808

ADC 0808 เป็น IC แบบ CMOS ที่ใช้แปลงสัญญาณ Analog เป็นข้อมูล Digital ที่มีขนาดของข้อมูลที่ *8 bit ซึ่งการทำงานภายในแบบ Successive Approximation ที่มีความเร็วปานกลาง* และความเที่ยงตรงสูง

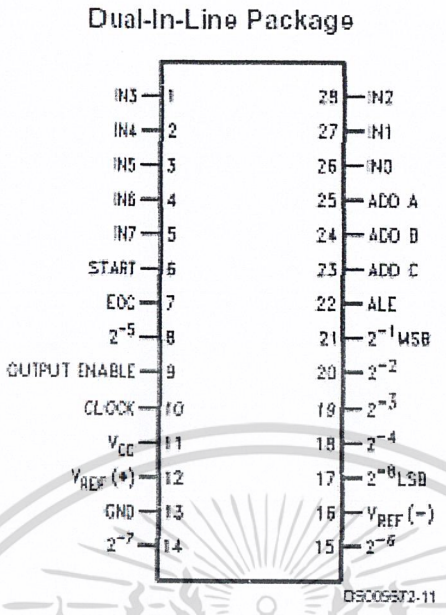
และ ADC 0808 มี Analog input ถึง 8 Channel ซึ่งสามารถต่อสัญญาณ Analog ที่ต้องการแปลงเป็นข้อมูล Digital ได้ถึง 8 ช่องสัญญาณ โดยมีขา multiplex ในการเลือก Analog input อยู่ 3 ขา คือ ADD A,ADD B และ ADD C รูปที่ 2.4 เป็นโครงสร้างภายใน ADC 0808



รูปที่ 2.4 โครงสร้างภายในของ ADC 0808

ลักษณะสำคัญของไอซี ADC 0808

- สามารถเปลี่ยนเป็นสัญญาณดิจิตอลได้ถึง 8 บิท
- ความไม่ลิเนียร์ประมาณ $\pm \frac{1}{2}$ LSB และ ± 1 LSB
- แหล่งจ่ายไฟ 5 VDC
- 8 Channel Analog input และ multiplex with Latched Control Logic
- ง่ายในการต่อใช้งาน และ Interface กับ Microprocessor
- ย่านอุณหภูมิ –20 องศาเซลเซียส ถึง + 85 องศาเซลเซียส หรือ –55 องศาเซลเซียส ถึง +125 องศาเซลเซียส
- Latched TRI-STATE output



รูปที่ 2.5 แสดงขาการใช้งานของ ADC 0808

การทำงานของขา ADC 0808

INO – IN7	เป็นขา Analog signal ที่สามารถต่อสัญญาณ analog ได้ 8 สัญญาณ
ADD A,B และ C	*เป็นขา Multiplex สัญญาณ Analog Input ตามที่ต้องการแปลงสัญญาณ Analog to Digital*
ALE	(ADDRESS LATCH BNABLE) เป็นขา Input ที่รับสัญญาณ เพื่อ Latch ข้อมูลขนาด 3 bit ที่ขา ADD A, ADD B และ ADD C เพื่อในการ Multiplex
START	*เป็นขา input เพื่อรับสัญญาณในการ START ให้ ADC 0808 ทำงานในเปลงสัญญาณ Analog เป็น Digital*
2^{-2}-2^{-1}	เป็นขาของข้อมูล Digital ขนาด 8 bit
OE	(output ENABLE) เป็นขาที่ควบคุมข้อมูล Digital ขนาด 8 bit ว่าต้องการที่จะส่งออกทางขา Output ของ ADC 0808 หรือไม่
EOC	(END OF CONVERSION) เป็นขา output ที่จะส่งสัญญาณเพื่อบอกว่าการทำงานของ ADC 0808 ทำการแปลงข้อมูลเสร็จแล้วหรือยัง

จากโครงสร้างของ ADC 0808 จะพบว่า ADC 0808 มี Analog input อยู่ 8 channel ที่สามารถต่อ Analog signal เพื่อจะแปลงเป็น Digital 8 bit ได้ถึง 8 channel และในการ multiplex channel ที่จะทำการแปลง จะต้องกำหนดข้อมูลขนาด 3 bit ให้กับขา ADD A, ADD B และADD C ดังตารางที่ 1

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

ตารางที่ 2.1 การ Multiplex Channel ADC 0808

ขั้นตอนการทำงาน

1. กำหนด channel ของ analog input ที่ต้องการ โดยป้อนเป็นสัญญาณ digital 3 bit เข้าที่ขา ADD A,ADD B,ADD C
2. ป้อนพัลซ์บวกเข้าที่ขา ALE (ADDRESS LATCH ENABLE) เพื่อทำการ Latch ข้อมูลขนาด 3 bit ที่ป้อนเข้าที่ขา ADD A,ADD B ,ADD C และวงจรจะทำการ multiplex analog channel ตามที่ข้อมูล 3 bit ตามตารางที่ 1
3. ป้อนพัลซ์บวกที่ที่ขา START เพื่อให้วงจรทำงานแปลงข้อมูล
4. เมื่อทำตามขั้นตอนที่1,2 และ 3 เสร็จ ADC 0808 จะทำการเลือก channel ที่เป็น analog signal *ตาม Data 3 bit มายังวงจรภายใน เข้ากับ Comparator in สัญญาณ analog ก็จะถูกทำการแปลง* เป็นสัญญาณ digital โดยมีขนาด 8 bit ขณะที่ทำการแปลงสัญญาณนั้น ที่ขา EOC (END OF CONVERSION) จะ Active Low จนกระทั่งวงจรภายในแปลงสัญญาณเสร็จ EOC ก็จะเปลี่ยนสถานะเป็น High นั้นแสดงว่าข้อมูลที่นำมาแปลงเป็นสัญญาณ Digital เรียบร้อยแล้ว

2.4 ภาษา VHDL

2.4.1 ประวัติความเป็นมาของภาษา VHDL

VHDL ย่อมาจากคำว่า VHSIC Hardware Description Language (VHSIC : Very High Speed Integrated Circuit) เป็นภาษาโปรแกรมระดับสูง (High Level Language) ที่ใช้สำหรับการออกแบบฮาร์ดแวร์ในระบบดิจิตอล ตัวของภาษาสามารถบรรยายพฤติกรรมการทำงานในรูปของลำดับชั้น (Hierarchy) และ สามารถเขียนได้หลายรูปแบบ ด้วยเหตุผลนี้จึงทำให้ภาษา VHDL เป็นเครื่องมือที่ใช้ออกแบบตั้งแต่ขั้นตอนบนสุด คือ แนวความคิดที่จะแก้ปัญหา ลงไปทีละขั้นจนถึงขั้นตอนของการสร้างวงจรจริง และตัวภาษาก็เปิดโอกาสให้วิศวกร ได้พัฒนาและจำลองการทำงานของรูปแบบฟังก์ชันการทำงานของวงจรอย่างสังเขป โดยยังไม่ต้องคำนึงถึงรายละเอียดเกี่ยวกับโครงสร้างวงจรจริง นอกจากนั้น VHDL ยังเป็นภาษาที่สนับสนุนลักษณะต่างๆ ของระบบดิจิตอลที่มี ความซับซ้อนได้ทั้งหมด ดังนั้น VHDL จึงเป็นภาษาที่น่าสนใจในการศึกษาและนำไปใช้งานเป็นอย่างยิ่ง วิวัฒนาการของภาษา VHDL เริ่มต้นประมาณปี ค.ศ. 1981 เมื่อกระทรวงกลาโหมสหรัฐอเมริกา หรือ DoD (Department of Defense) ได้พยายามปรับปรุงอุปกรณ์อิเล็กทรอนิกส์และคอมพิวเตอร์ที่ใช้ในกิจการทางทหาร ให้มีความทันสมัยมากขึ้น ประกอบกับเทคโนโลยีทางด้านไมโครอิเล็กทรอนิกส์มีการพัฒนาไปอย่างรวดเร็วดังจะเห็นได้ จากการนำวงจรดิจิตอลหลายๆ วงจรมาทำการผลิตอยู่บนแผ่นซิลิกอนที่มีพื้นที่เพียง 1 - 2 ตารางเซนติเมตรเท่านั้น ซึ่ง เป็นผลให้ประสิทธิภาพในการทำงานของวงจรสูงขึ้นตลอดจนความน่าเชื่อถือ ในการทำงานและความคงทนต่อสภาพ แวดล้อมสูง แต่เนื่องจากในขณะนั้นขั้นตอนของการออกแบบ การผลิต และการตรวจสอบวงจรต้นแบบ เป็นขบวนการที่ ต้องใช้วิศวกร และเวลาในดำเนินการมาก ฉะนั้น ทาง DoD จึงจัดตั้งโครงการขึ้นมาเพื่อศึกษาวิธีการที่ช่วยในการพัฒนา วงจรอิเล็กทรอนิกส์ โดยเฉพาะอย่างยิ่งวงจรระบบดิจิตอล ให้สามารถนำไปผลิตได้เร็วขึ้น ซึ่งโครงการดังกล่าวมีชื่อว่า "Very High Speed Integrated Circuits" หรือ VHSIC โดยในระยะแรกนั้นโครงการนี้ถือเป็นความลับทาง ด้านความมั่งคงของประเทศ และอยู่ภายใต้ความควบคุมดูแลของ United States International Traffic and Arms Regulations (ITAR)

สำหรับมาตรฐานของภาษาที่ใช้บรรยาย พฤติกรรมวงจรหรือฮาร์ดแวร์ของระบบสำหรับโครงการ VHSIC ที่ DODได้ให้ไว้สามารถสรุปได้ดังนี้

- ต้องเป็นภาษาที่นำไปเขียนรูปแบบระบบดิจิตอลและมีคุณสมบัติที่สามารถเข้าใจได้ทั้งมนุษย์และเครื่องคอมพิวเตอร์โดยไม่ต้องมีการแปลหรือเปลี่ยนแปลงอีก
- สามารถนำไปใช้เป็นเอกสารประกอบโครงการได้
- ต้องเป็นภาษาที่เขียนขึ้นสำหรับใช้จำลองการทำงานของวงจร

ฉะนั้นภาษาดังกล่าวนี้จึงจัดเป็นภาษาโปรแกรมระดับสูง เช่นเดียวกับภาษาปาสคาล หรือภาษาซี ซึ่งในทางวิศวกรรม ภาษาที่ใช้ในการออกแบบฮาร์ดแวร์นี้เรียกว่า "Hardware Description Language" ในตอนเริ่มแรกนั้น DOD ได้มอบหมายให้บริษัทไอบีเอ็ม เท็กซัสอินสตูเมนท์ และอินเตอร์เมทริกซ์ เป็นผู้ศึกษาและพัฒนา โครงการ ซึ่งการดำเนินงานเป็นไปอย่างต่อเนื่อง จนกระทั่งในปี ค.ศ.1985 ทาง ITAR ได้ยกเลิกข้อจำกัดในการถ่ายทอด เทคโนโลยีทางทหารออกจากโครงการนี้ ดังนั้นภาษา VHDL จึงเริ่มเป็นที่รู้จักกันโดยทั่วไป และประมาณปี ค.ศ. 1987 IEEE ได้ทำการกำหนดมาตรฐานของภาษานี้เป็น IEEE 1076-1987 และมีชื่อเรียกว่า VHDL ซึ่งมาตรฐานนี้ได้รับการปรับปรุงจนเป็นมาตรฐาน IEEE 1076-1993 หรือ VHDL 1993 เนื่องจากในขณะนั้น DOD เป็นลูกค้ารายใหญ่ ของอุตสาหกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์ ดังนั้นจึงมีผู้รับโครงการต่างๆ จาก DOD ไปดำเนินการวิจัยและพัฒนา เป็นจำนวนมาก และเพื่อให้ทุกโครงการอยู่ในมาตรฐานเดียวกันหมด ดังนั้นทาง DOD จึงได้กำหนดว่า ทุกๆ โครงการต้อง เขียนอยู่ในรูปของภาษา VHDL เท่านั้น ซึ่งทำให้ DODสามารถนำโครงการเหล่านี้ไปจำลองกับเครื่องคอมพิวเตอร์ได้ หลายๆระบบ

2.4.2 ข้อกำหนดของภาษา VHDL

DOD ได้ตั้งข้อกำหนดสำหรับภาษา VHDL ในเดือนมกราคมปี ค.ศ.1983 ไว้ดังนี้

1. ลักษณะทั่วไปDOD ได้กำหนดให้ VHDL เป็นภาษาสำหรับการออกแบบและบรรยายของฮาร์ดแวร์ ซึ่งหมายถึงความสามารถ ในการอธิบายและออกแบบในระดับสูง การจำลอง (Simulation) การสังเคราะห์ (Synthesis) และการทดสอบ (Testing) นอกจากนั้น VHDL ยังถูกกำหนดไว้สำหรับการบรรยายฮาร์ดแวร์ตั้งแต่ระดับบนซึ่งก็คือระบบจนถึง ระดับเกท อีกด้วย เนื่องจากในการทำงานของระบบดิจิตอลนั้น ทุกๆ องค์ประกอบภายในระบบไม่ว่าเล็กหรือใหญ่ จะทำงานไปพร้อมๆ กัน ซึ่งในเรื่องของความพร้อมเพรียงในการทำงานนี้ก็ถือเป็นข้อกำหนดที่สำคัญ

อย่างหนึ่งของ VHDL ด้วยเช่นกัน (สำหรับในภาษาที่ใช้ในการบรรยายฮาร์ดแวร์นั้นความพร้อมเพรียงจะหมายถึงทุกๆ คำสั่ง องค์ประกอบ เกท หรือวงจรลอจิกจะถูกนำมาปฏิบัติทั้งหมด ดังนั้นในที่สุดแล้วก็จะดูเหมือนว่าได้มีการปฏิบัติไป พร้อมๆ กัน)

2. สนับสนุนการออกแบบแบบลำดับขั้นการออกแบบแบบลำดับขั้นเป็นลักษณะที่สำคัญอย่างหนึ่งสำหรับการออกแบบระบบที่มีหลายๆ ระดับ โดยในการ ออกแบบจะประกอบด้วยส่วนการบรรยายการเชื่อมต่อ และส่วนการบรรยายหน้าที่การทำงาน ซึ่งหน้าที่การทำงาน ของระบบสามารถกำหนดได้ด้วยตัวเอง หรืออาจถูกกำหนดโดยโครงสร้างที่ประกอบด้วยองค์ประกอบย่อยๆ ลง ไปได้เช่นกัน แต่ที่ระดับล่างสุด องค์ประกอบต้องถูกบรรยายหน้าที่การทำงานด้วยตัวมันเองและไม่สามารถกำหนด การทำงานโดยลักษณะแบบโครงสร้างได้

3. ไลบรารี VHDL ได้สนับสนุนการมีไลบรารีเพื่อระบบการจัดการที่ดี ผู้ออกแบบสามารถกำหนดลักษณะและการทำงานของ อุปกรณ์พื้นฐานไว้ในระบบไลบรารี หรือจะใช้ไลบรารีที่ระบบได้จัดเตรียมไว้แล้วก็ได้ โมเดลและการบรรยายที่ถูก ต้องควรจัดเก็บไว้ในไลบรารีหลังจากที่ได้ผ่านการคอมไพล์เรียบร้อยแล้วเพื่อให้ผู้ออกแบบคนอื่นๆ สามารถนำไป ใช้ได้ด้วย

4. ลำดับคำสั่ง แม้ว่าการปฏิบัติคำสั่งหรือกระบวนการโดยพร้อมเพรียงกันจะเป็นคุณสมบัติที่สำคัญของ VHDL ก็ตาม ตัวภาษา เองก็ยังมีการจัดเตรียมลักษณะการควบคุมแบบลำดับคำสั่งไว้ให้ด้วย เมื่อผู้ออกแบบได้กำหนดหน้าที่และองค์ประกอบ ที่ทำงานพร้อมกันของระบบไว้เรียบร้อยแล้ว ผู้ออกแบบยังสามารถบรรยายหน้าที่การทำงานซึ่งเป็นรายละเอียดภายใน ของแต่ละองค์ประกอบได้ในลักษณะเดียวกับการเขียนโปรแกรมที่ประกอบด้วยโครงสร้างแบบ case, if - then - else และ loop ทั่วๆ ไปได้ การบรรยายแบบลำดับคำสั่งทำให้การออกแบบหน้าที่การทำงานของอุปกรณ์กระทำได้ สะดวกและง่ายขึ้น อย่างไรก็ตามโครงสร้างทั้งหมดของ VHDL ก็ยังคงเป็นการทำงานแบบพร้อมเพรียงกันเช่นเดิม

5. การกำหนดคุณสมบัตินอกจากการกำหนดอินพุทและเอาท์พุทแล้ว เงื่อนไขอื่นๆ ก็มีผลต่อการปฏิบัติหน้าที่ของอุปกรณ์ฮาร์ดแวร์ด้วยเช่นกัน โดยสิ่งนี้รวมถึงสภาพแวดล้อมและลักษณะทางกายภาพของอุปกรณ์นั้นๆ ด้วย ซึ่งภาษาสำหรับการออกแบบที่ดีควร ให้ผู้ออกแบบกำหนดคุณสมบัติของอุปกรณ์ที่ใช้ได้ด้วย เช่น สามารถกำหนดขนาด ลักษณะทางกายภาพเวลา โหลด และเงื่อนไขทางสภาพแวดล้อมอื่นๆ ซึ่งความสามารถในการกำหนดคุณสมบัตินี้ก็เป็นส่วนหนึ่งที่มีอยู่ในภาษา VHDL ด้วยเช่นกัน

6. ชนิดของข้อมูล VHDL สามารถกำหนดชนิดของข้อมูลไม่เพียงแต่ชนิด BIT และ BOOLEAN เท่านั้น แต่ยังสามารถกำหนดชนิด ของข้อมูลเป็นจำนวนเต็ม จำนวนจริง จุดทศนิยม

และชนิดลำดับการนับ (Enumerate Type) หรือแม้แต่ชนิดของ ข้อมูลที่ผู้ออกแบบกำหนดขึ้นมาเองก็ได้

7. โปรแกรมย่อย ความสามารถในการใช้ฟังก์ชันและโพรซีเจอร์ (Procedure) ก็เป็นข้อกำหนดอีกอย่างหนึ่งใน VHDL ซึ่งผู้ออกแบบ สามารถนำโปรแกรมย่อยมาใช้ในการเปลี่ยนแปลงชนิดของข้อมูล การกำหนดหน่วยของลอจิก การกำหนดตัวกระทำต่างๆ หรือหน้าที่อื่นๆ ตามที่ต้องการได้เช่นเดียวกับการเขียนโปรแกรมทั่วไป

8. การควบคุมเวลา VHDL อนุญาตให้ผู้ออกแบบสามารถกำหนดเวลาในการส่งผ่านข้อมูลหรือสัญญาณได้ตามต้องการ การตรวจสอบ การออกแบบเกทหรือการหน่วงเวลาก็สามารถกระทำได้โดยการกำหนดช่วงเวลาที่แน่นอนหรือกำหนดให้มีการรอคอย เหตุการณ์ (Event) นอกจากนี้ก็ยังสามารถกำหนดรูปแบบของสัญญาณมาพิกาได้อีกด้วย

9. การกำหนดแบบโครงสร้าง การกำหนดโครงสร้างขององค์ประกอบต่างๆ สามารถกระทำได้ในทุกระดับของการออกแบบ โดยการกำหนดโครง สร้างขององค์ประกอบร่วมที่เกิดจากองค์ประกอบย่อยซึ่งแตกต่างกันหรือ เหมือนกันก็เป็นข้อกำหนดอย่างหนึ่งของ VHDL เช่นกัน

2.5 องค์ประกอบพื้นฐานของ VHDL

รูปแบบพื้นฐานที่ใช้ในการบรรยายถึงองค์ประกอบของ VHDL จะประกอบไปด้วยส่วนกำหนดการเชื่อมต่อ (Interface) และส่วนกำหนดลักษณะเชิงสถาปัตยกรรม (Architecture) ดังแสดงในรูปที่ 2.6 โดยในการบรรยายการเชื่อมต่อจะขึ้น ต้นด้วยคำว่า ENTITY แล้วตามด้วยชื่อขององค์ประกอบจากนั้นตามด้วยคำว่า IS และถัดมาจะเป็นการบรรยายถึงพอร์ต การติดต่อ อินพุท - เอาท์พุท ขององค์ประกอบ ส่วนลักษณะภายนอกอื่น ๆ เช่น เวลา อุณหภูมิก็สามารถรวมเข้าไปในส่วนนี้ได้เช่นกัน ในส่วนของการกำหนดลักษณะเชิงสถาปัตยกรรมจะขึ้นต้นด้วยคำว่า ARCHITECTURE ซึ่งเป็นส่วนที่ใช้ บรรยายหน้าที่การทำงานขององค์ประกอบ โดยหน้าที่การทำงานนี้จะขึ้นอยู่กับสัญญาณอินพุท เอาท์พุทและพารามิเตอร์ อื่นๆ ที่ได้กำหนดไว้ในส่วนของการเชื่อมต่อดังรูปที่ 2.6 และสำหรับการบรรยายหน้าที่ขององค์ประกอบจะเริ่มต้นหลังจาก คำว่า BEGIN เป็นต้นไป

```
ENTITY component_name IS
    Input and output ports
    Physical and other parameters
END [component_name] ;

ARCHITECTURE identifier OF component_name IS
    [declartion]
BEGIN
    specification of the functionality of the component
    in terms of its input lines and as influenced
    by physical and other parameters
END [identifier];
```

รูปที่ 2.6 การกำหนดการเชื่อมต่อและสถาปัตยกรรม

2.5.1 การกำหนดการเชื่อมต่อ การกำหนดการเชื่อมต่อเป็นระดับบนสุดของการออกแบบ โดยในระดับนี้ต้องกำหนดพอร์ตสำหรับการติดต่อกับองค์ประกอบ ภายนอกอื่นๆ ดังตัวอย่างในรูปที่ 2.7 ซึ่งเป็นบล็อคไดอะแกรม และการบรรยายการเชื่อมต่อขององค์ประกอบสำหรับตัวจ่ายสัญญาณนาฬิกา ในบรรทัดแรกของการบรรยายการเชื่อมต่อเป็นการกำหนดชื่อขององค์ประกอบ ซึ่งกำหนดเป็น clock_ component ตามด้วยคำว่า PORT และชื่อของพอร์ตอยู่ภายในวงเล็บ ส่วน IN และ OUT เป็นการกำหนด โหนดของสัญญาณให้เป็นอินพุทหรือเอาท์พุท และ BIT เป็นการแสดงชนิดของข้อมูล


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

รูปที่ 2.7 บล็อคไดอะแกรมและการบรรยายการเชื่อมต่อของ clock_ component

2.5.2 การกำหนดรูปแบบการบรรยายหน้าที่การทำงานขององค์ประกอบจะถูกบรรยายภายในส่วนนี้ ซึ่งในการบรรยายสามารถกำหนดค่าของสัญญาณ เอาท์พุทในเทอมของอินพุทหรือในรูปขององค์ประกอบอื่นๆ หรือทั้งสองอย่างรวมกันก็ได้ ดังตัวอย่างการบรรยายของ clock_component ในรูปที่ 2.8 ซึ่งเป็นการบรรยายในเชิงพฤติกรรมโดยมี en เป็นอินพุทและ ck เป็นเอาท์พุท PROCESS เป็นคำที่ใช้ในการเริ่มต้นสำหรับการบรรยายในเชิงพฤติกรรม และภายในโปรเซสกำหนดให้ periodic เป็นตัวแปรที่มีคำเริ่มต้นเป็น "0" ถ้าสัญญาณ en มีค่าเป็น "1"' จะทำให้ตัวแปร periodic ถูกคอมพลีเมนท์ (complement) และส่งค่าให้กับ ck ซึ่งเป็นสัญญาณเอาท์พุท และสำหรับคำสั่ง WAIT จะเป็นการกำหนดให้สัญญาณมีคาบเวลาเท่ากับ 1 ไมโครวินาที

```
ARCHITECTURE behavioral OF clock_component IS

BEGIN
   PEOCESS
      VARIABLE periodic : BIT := '0';
   BEGIN
      IF en='1' THEN
       periodic := Not periodic;
      END IF;
      ck <= periodic;
      WAIT FOR 1 US;
  END PROCESS;
END behavioral;
```

รูปที่ 2.8 การบรรยายเชิงพฤติกรรมของ clock _ component

2.5.3 หน่วยการออกแบบแพ็กเก็ตข้อมูลต่างๆ ตลอดจนโปรแกรมย่อย ที่เป็นประโยชน์ต่อการเขียนรูปแบบการบรรยายระบบดิจิตอล สามารถเก็บไว้ใน ส่วนของแพ็กเก็ต ซึ่งหน่วยการออกแบบต่างๆ เช่น หน่วยการออกแบบ Entity หน่วยการออกแบบสถาปัตยกรรมหรือ หน่วยการออกแบบแพ็กเกจอื่นๆ สามารถเรียกข้อมูลเหล่านี้ไปใช้ได้ นอกจากนั้นสิ่งที่นิยมทำกันมากคือการนำรูปแบบ มาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐาน (เช่น ไอซีตระกูล 74XX เป็นต้น) มาเก็บไว้ในรูปของแพ็กเกจ ที่ทุกคนสามารถ เข้าถึงได้ ตามปกติแล้วแพ็กเกจจะแบ่งออกเป็น 2 ส่วนคือ การประกาศแพ็กเกจ (Package declaration)และ ส่วนของบอดี้แพ็กเกจ (Package body) เนื่องจาก แพ็กเกจถูก

สร้างขึ้นเป็นส่วนแยกต่างหากออกจากรูปแบบที่ กำลังเขียนอยู่ ฉะนั้นการที่นำแพ็กเกจไปใช้นั้นจะต้องมีการเชื่อมโยงหรืออ้างอิงเสียก่อน ซึ่งในภาษา VHDL สามารถ กระทำได้ด้วยชุดคำสั่ง USE

- PACKAGE DECLARATIONส่วนที่มีความสำคัญที่สุดของแพ็กเกจ (ถ้ามองในแง่ของการนำไปใช้จากภายนอก) ได้แก่ส่วนการประกาศแพ็กเกจ เนื่องจากเป็นส่วนที่ใช้กำหนดชื่อของสิ่งที่ประกาศอยู่ภายในแพ็กเกจ สำหรับนำไปใช้ภายนอกตัวของแพ็กเกจเอง ถ้ามี การประกาศสิ่งใดๆ ในส่วนของส่วนบอดี้แพ็กเกจ แต่ไม่ถูกประกาศในส่วนการประกาศแพ็กเกจจะทำให้ค่าและพฤติกรรมไม่สามารถนำไปใช้งานในส่วนนอกได้ซึ่งเปรียบเทียบได้กับสิ่งที่ประกาศไว้ในส่วนของการประกาศ Entity คือ จุดเชื่อมต่อ หรือ พอร์ต ที่มีหน้าที่ติดต่อกับโลกภายนอก ฉะนั้นโดยทั่วไปแล้วแพ็กเกจสามารถสร้างขึ้นได้โดยไม่จำเป็น ต้องมีส่วนบอดี้ และยังสามารถนำไปใช้งานจากรูปแบบภายนอกได้เช่น ใช้สำหรับประกาศ ชนิด (Type) หรือสัญญาณ เช่นเดียวกับ ส่วนบอดี้แพ็กเกจที่ไม่จำเป็นต้องมี ส่วนของการประกาศแพ็กเกจ แต่แพ็กเกจนั้นจะไม่สามารถนำไปใช้จาก รูปแบบอื่นได้

```
PACKAGE package_name IS
    Package_declarative_part
END package_name;
```

รูปที่ 2.9 โครงสร้างทั่วไปของส่วนการประกาศแพ็กเกจ

- PACKAGE BODYโครงสร้างซึ่งประกอบด้วยลำดับคำสั่งที่ใช้บรรยายฟังก์ชันการทำงานของโปรแกรมย่อยทั้งหลาย ซึ่งชื่อของโปรแกรมย่อยนั้นๆ ได้ถูกประกาศไปแล้วในส่วนของการประกาศแพ็กเกจ จะถูกเก็บไว้ในส่วนของบอดี้แพ็กเกจ ทั้งนี้รวมถึง การกำหนดค่าคงที่ต่างๆ อันได้แก่ค่าคงที่ที่ถูกประกาศชื่อไว้ก่อนในส่วนของการประกาศแพ็กเกจ และถูกกำหนดค่าในส่วนของบอดี้แพ็กเกจ ฉะนั้นในส่วนของบอดี้แพ็กเกจจึงไม่จำเป็นต้องมี ถ้าในส่วนของการประกาศแพ็กเกจไม่มีการ ประกาศชื่อที่เป็นโปรแกรมย่อย หรือค่าคงที่ การเขียนบอดี้แพ็กเกจนั้นจะเป็นไปตามกฎเกณฑ์ดังแสดงในรูปที่ 2.10

```
PACKAGE BODY package_name IS
    declarative part
END package_name;
```

รูปที่ 2.10 โครงสร้างของบอดี้แพ็กเกจ

2.5.4 หน่วยการออกแบบ Configurationดังที่ทราบกันแล้วว่าระบบดิจิตอลรูปแบบหนึ่งไม่ว่าจะเป็นอะไรก็ตาม จะสามารถมีหน่วยการออกแบบ Entity ได้ เพียงหนึ่งเดียวเท่านั้น ซึ่งในหน่วยการออกแบบ Entity หนึ่งหน่วยนี้อาจจะมีสถาปัตยกรรมที่เป็นหน่วยรองได้หลาย หน่วย ดังนั้นจะต้องมีหน่วยการออกแบบ Configuration มาเพื่อกำหนดการใช้ Configuration ของการประกอบ Entity กับหน่วยการออกแบบสถาปัตยกรรมหน่วยใดๆ เข้าด้วยกัน

```
COMFIGURTRION identifier OF entiry_name IS
    Configuretion_declarative_part
END ;
```

รูปที่ 2.11 โครงสร้างโดยทั่วไปของหน่วยการออกแบบโครงแบบ

2.5.5 โปรแกรมย่อย การใช้ฟังก์ชันและโพรซีเจอร์ใน VHDL เปรียบได้กับการใช้โปรแกรมย่อยในการเขียนโปรแกรมภาษาชั้นสูงทั่วๆ ไปค่าที่ถูกส่งกลับหรือถูกเปลี่ยนแปลงโดยโปรแกรมย่อยอาจจะมีหรือไม่มีผลต่อฮาร์ดแวร์โดยตรงก็ได้ เช่นถ้าใช้ฟังก์ชัน แทนการกระทำในสมการบูลีนก็จะมีผลต่อวงจรลอจิกจริงๆในขณะที่ถ้าใช้โปรแกรมย่อยในการเปลี่ยนชนิดของข้อมูล หรือในการคำนวณค่าการหน่วงเวลาแล้วก็จะไม่มีผลต่อโครงสร้างของฮาร์ดแวร์ รูปที่ 2.12 แสดงการใช้โพรซีเจอร์ เพื่อเปลี่ยนข้อมูลชนิด 8 บิตเป็นค่าจำนวนเต็ม และรูปที่ 2.13 แสดงการใช้ฟังก์ชันโดยกำหนดให้ X เป็นตัวแปรชนิด บิตแทนการกระทำในสมการบูลีน

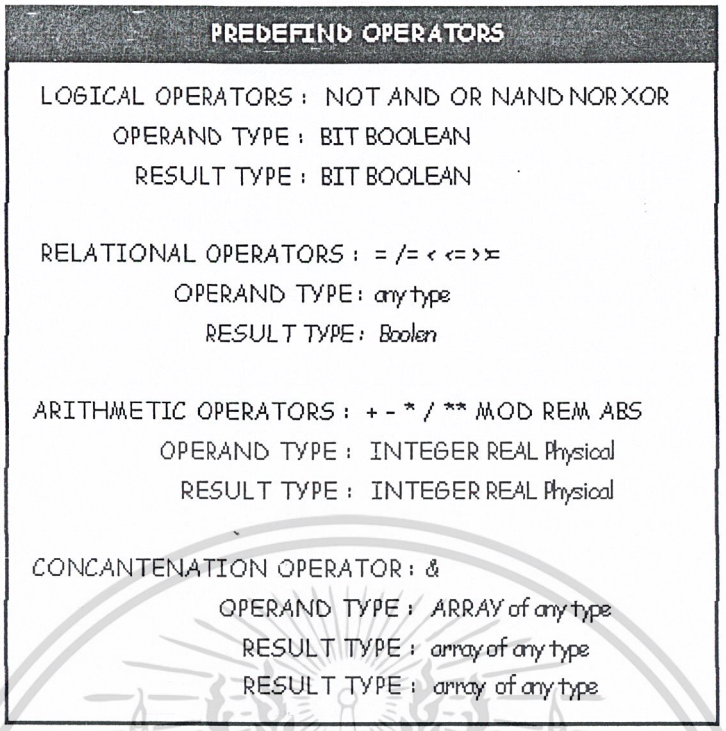
```
TYPE byte IS ARRAY (7 DOWNTO 0) OF BIT;
 ...
PROCEDURE byte_to_integer (ib : IN byte; oi : OUT INTEGER) IS
        VARIABLE result: INTEGER := 0;
BEGIN
        FOR i IN 0 TO 7 LOOP
                IF ib(i) = '1' THEN
                result  := result + 2**i;
                END IF;
        END LOOP;
        oi := result;
END byte_to_integer
```

รูปที่ 2.12 การใช้โพรซีเจอร์

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

รูปที่ 2.13 การใช้ฟังก์ชัน

2.5.6 โอเปอร์เรเตอร์ การบรรยายเชิงพฤติกรรมในภาษา VHDL มีตัวดำเนินการหรือโอเปอร์เรเตอร์ทางลอจิกและคณิตศาสตร์เช่นเดียว กับภาษาซอฟต์แวร์ทั่วไปดังรูปที่ 2.14



รูปที่ 2.14 ตัวดำเนินการใน VHDL

2.5.7 เวลาและความพร้อมเพรียง ในวงจรอิเล็กทรอนิกส์อุปกรณ์ทุกๆ ตัวจะอยู่ในสภาพเตรียมพร้อมเสมอ (Always Active) และจะมีเรื่องของเวลา เข้ามาเกี่ยวข้องในทุกๆเหตุการณ์ที่เกิดขึ้นเสมอ VHDL เป็นภาษาที่ได้รับการออกแบบมาเพื่อให้สามารถบรรยายรูป แบบและการพ้องกันของเวลาสำหรับการทำงานของอุปกรณ์ได้อย่างถูกต้อง การบรรยายการทำงานที่อยู่ภายในส่วนของการบรรยายสถาปัตยกรรม จะมีการทำงานที่พร้อมเพรียงกันเสมอ หรือแม้แต่โปรเซสซึ่งมีการทำงานภายในเป็น แบบลำดับคำสั่งก็ตาม ซึ่งหากมีหลายๆโปรเซสอยู่ภายในโครงสร้างเดียวกันทุกๆโปรเซสก็จะทำงานไปพร้อมๆ กัน ด้วย

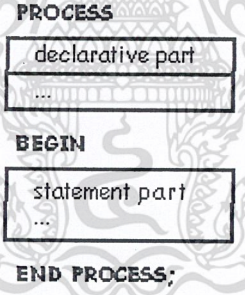
2.5.8 สัญญาณและตัวแปร สัญญาณมีลักษณะเป็นเสมือนตัวกลางฮาร์ดแวร์ที่ใช้ในการส่งผ่านข้อมูลและมีเรื่องของเวลาเข้ามาเกี่ยวข้องด้วยการ กำหนดค่าให้กับสัญญาณจะใช้สัญลักษณ์ <= ในการส่งค่าและสามารถใช้คำสั่ง AFTER เพื่อกำหนดช่วงเวลาในการ ส่งผ่านค่าของสัญญาณ เช่น w <= a AFTER 12 NS หมายถึงการกำหนดค่าสัญญาณ a ให้กับ w หลังจากเวลา ผ่านไป 12 นาโนวินาที ในทางตรงข้ามตัวแปรมีลักษณะเป็นเสมือนตัวกลางที่ใช้ในการส่งผ่านข้อมูลและไม่มีเรื่องของ เวลาเข้ามาเกี่ยวข้องด้วย ซึ่งตัวแปรจะถูกใช้ในส่วนที่มีการทำงานเป็นแบบลำดับคำสั่งเช่นในฟังก์ชัน โพรซีเจอร์ และ โปรเซส สำหรับการกำหนดค่าให้กับตัวแปรจะใช้สัญลักษณ์ :=

2.6 การบรรยายเชิงพฤติกรรม

การบรรยายลักษณะการทำงานของอุปกรณ์ฮาร์ดแวร์ในเชิงพฤติกรรม เป็นการบรรยายลักษณะการเปลี่ยนแปลงของ ข้อมูลในรูปแบบของอัลกอริธึมสำหรับการคำนวณผลลัพธ์ที่เกิดขึ้น ซึ่งสืบเนื่องมาจากการเปลี่ยนแปลงสภาวะของข้อมูล ที่เข้ามาโดยไม่คำนึงถึงลักษณะโครงสร้างหรือความสัมพันธ์ของอุปกรณ์ที่อยู่ภายในว่าจะเป็นอย่างไร

2.7 โปรเซส

โปรเซสเป็นรูปแบบพื้นฐานอย่างหนึ่งที่ใช้ในการกำหนดให้กับสัญญาณ โปรเซสจะอยู่ในสถานะที่เตรียมพร้อมอยู่เสมอ และจะปฏิบัติคำสั่งพร้อมๆ กันกับโปรเซสอื่นๆ ที่อยู่ในสถาปัตตกรรมบรรยายเดียวกัน โดยโปรเซสจะปฏิบัติงานตามคำ สั่งทันทีที่มีเหตุการณ์เกิดขึ้นกับญญาณที่อยู่ทางด้านขวามือของสัญลักษณ์กำหนดค่าให้กับสัญญาณ (<=) การบรรยาย โปรเซสจะเริ่มต้นด้วยคำสั่ง PROCESS และจบด้วยคำสั่ง END PROCESS ในรูปที่ 2.15 เป็นการแสดงส่วน ประกอบของการบรรยายแบบโปรเซส ซึ่งประกอบด้วยส่วนของการประกาศตัวแปรที่ต้องใช้และส่วนของการปฏิบัติ คำสั่งเพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 2.15 รูปแบบของการบรรยายแบบโปรเซส

การกำหนดตัวดำเนินการภายในโปรเซส ตัวดำเนินการภายในโปรเซสมี 3 ชนิดคือ ตัวแปร (Variable) ไฟล์ (File) และตัวคงที่ (Constant) ซึ่งตัวดำเนิน การทั้งสามชนิดนี้หากมีการประกาศไว้ในโปรเซสใดก็จะใช้ได้เฉพาะภายในโปรเซสนั้นเท่านั้นสำหรับการติดต่อกับภาย นอกหรือระหว่างโปรเซสสามารถทำได้โดยใช้สัญญาณ (Signal) หรือตัวคงที่ที่ได้ประกาศไว้ในส่วนของ ARCHITECTURE ในรูปที่ 2.16 แสดงตัวอย่างการประกาศตัวกระทำภายในโปรเซส ซึ่งจะอยู่

ระหว่างคำสั่ง PROCESS และ BEGIN และค่าเริ่มต้นที่ถูกกำหนดให้กับตัวดำเนินการภายในโปรเซสจะถูกนำมาใช้ในตอนเริ่มต้น ของการปฏิบัติเพียงครั้งเดียวเท่านั้น ต่างกับค่าเริ่มต้นที่อยู่ภายในโปรแกรมย่อยจะถูกนำมาใช้ทุกครั้งที่มีการเรียกใช้ โปรแกรมย่อยนั้น ๆ

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

รูปที่ 2.16 ตัวอย่างการประกาศตัวดำเนินการภายในโปรเซส

การกำหนดการกระทำภายในโปรเซส การกระทำใดๆ ภายในโปรเซสจะเป็นการปฏิบัติแบบลำดับ (Sequential) เสมอ ซึ่งภายในโปรเซสสามารถใช้ประโยค เงื่อนไขหรือการทำซ้ำได้เช่น *IF-THEN - ELSE,CASE - WHEN, FOR LOOP และ WHILE LOOP* ดังตัวอย่างในรูปที่ 2.17 และ 2.18

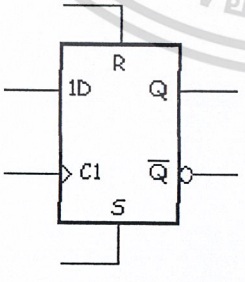
```
ARCHITECTURE demo OF partial_process IS
  ....
BEGIN
  PROCESS
  ....
  BEGIN
        ....
        x <= '1';
        IF x = '1' THEN
          perform action_1
        ELSE
          perform action_2
        END IF;
        ...
  END PROCESS;
END demo;
```

รูปที่ 2.17 เงื่อนไขการกระทำในโปรเซส

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

รูปที่ 2.18 แสดงการกระทำในโปรเซส

การกระตุ้นและยับยั้งการกระทำของโปรเซสการกระทำภายในโปรเซสจะอยู่ในสภาพเตรียมพร้อม และมีการปฏิบัติงานอยู่ตลอดเวลาที่มีการเปลี่ยนแปลงของเหตุการณ์ เกิดขึ้น อย่างไรก็ดีเราสามารถกระตุ้นหรือยับยั้งการกระทำภายในโปรเซสได้โดยการกำหนดรายการของสัญญาณที่ต้อง การให้โปรเซสปฏิบัติงานเมื่อมีเหตุการณ์เกิดขึ้นกับสัญญาณที่กำหนดไว้เท่านั้น ส่วนเหตุการณ์ใดๆ ที่เกิดขึ้นกับสัญญาณ ที่ไม่ได้กำหนดไว้ในรายการก็จะไม่ส่งผลให้มีการกระทำภายในโปรเซส ซึ่งรายการของสัญญาณนี้เรียกว่า Sensitivity List และจะกำหนดไว้ภายในวงเล็บหลังคำสั่ง PROCESS รูปที่ 2.19(a) แสดงตัวอย่างโมเดล และรูปที่ 2.19 (b) เป็นตัวอย่างการบรรยายการเชื่อมต่อของ D-Flip Flop



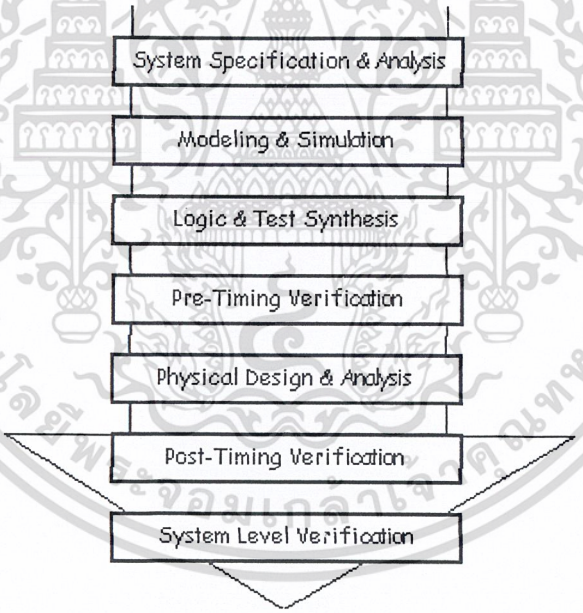

```
ENTITY component_name IS
   Input and output ports
   Physical and other parameters
END component_name;
```

(a) (b)

รูปที่ 2.19 ตัวอย่างโมเดล D-Flip Flop

2.8 การออกแบบจากบนลงล่าง

ภาษา VHDL นั้นอนุญาตให้อธิบายและวิเคราะห์การ ทำงานของแต่ละบล็อก รวมถึงการปรับปรุงการทำงานจากผลที่วิเคราะห์เพื่อให้ได้การทำงานตามต้องการ นอกจากนี้ยัง สามารถเพิ่มเติมในรายละเอียดในแต่ละขั้นตอนได้ ซึ่งหลักการนี้สอดคล้องกับหลักการออกแบบจากบนลงล่าง (Top - Down Design) นั่นเอง ถ้าทดลองเปรียบเทียบกับการออกแบบจากล่างขึ้นบน (Bottom - Up Design) จะเห็นได้ว่า การออกแบบจากล่างขึ้นบนจะใช้เวลาการออกแบบมากกว่า 90% เนื่องจากเป็นการวาดวงจรด้วยอุปกรณ์ต่างๆ (Schematic capture) ที่ประกอบกันเข้าเป็นวงจรที่ต้องการออกแบบ ก่อนแล้วจึงทำการจำลองการทำงาน และตรวจ สอบความถูกต้อง ดังนั้นการใช้ภาษา VHDL กับหลักการออกแบบจากบนลงล่างจึงเป็นทางออกให้กับวิศวกรให้สามารถ ออกแบบและพัฒนาวงจรที่มีความซับซ้อนได้มากขึ้น ทั้งยังช่วยลดเวลาและค่าใช้จ่ายในการออกแบบด้วย



รูปที่ 2.20 ขั้นตอนการออกแบบจากบนลงล่าง

จากรูปที่ 2.20 แสดงถึงขั้นตอนของการออกแบบจากบนลงล่าง ทั้งนี้ในทางปฏิบัติอาจมีข้อแตกต่างไปจากนี้บ้าง เล็กน้อยเนื่องจากขั้นตอนของการผลิต (Implementation) สามารถกระทำได้หลายเทคโนโลยี สำหรับรายละเอียดของขั้นตอน

2.8.1 การออกแบบจากบนลงล่างในแต่ละขั้นตอนมีดังนี้

1.สร้างข้อกำหนดของความต้องการ และวิเคราะห์ระบบ เพื่อหาแนวความคิดและหลักการ (Idea and Concept) ใน การแก้ปัญหา

2.เขียนรูปแบบของระบบที่ต้องการออกแบบโดยใช้ภาษา VHDL หรือ ภาษา HDL อื่น ๆ สำหรับบรรยายพฤติกรรมการ ทำงาน พร้อมทั้งจำลองการทำงาน เพื่อเปรียบเทียบและตรวจสอบความถูกต้องกับข้อกำหนด

3.หลังจากที่ได้หลักการขั้นต้นพร้อมแนวความคิดที่ผ่านการตรวจสอบแล้วหลักการนี้จะถูกเพิ่มเติมในรายละเอียดลงมา เป็นลำดับขั้นที่สอง จนกระทั้งอยู่ในระดับที่จะนำไปผลิตวงจริง หรือสังเคราะห์ในขั้นตอนนี้เองเทคโนโลยีที่จะมารองรับ วงจรออกแบบจะถูกกำหนดขึ้น และระบบช่วยการออกแบบจะสังเคราะห์วงจรที่ได้จากรูปแบบที่เขียนขึ้นให้อยู่ในรูปของ วงจรที่ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ หรือวงจรในระดับเกท และการเชื่อมต่อระหว่างกันของอุปกรณ์เหล่านั้นหรือไม่ก็อยู่ในรูปของ Netlist ที่สามารถนำไปผลิตในอุปกรณ์อื่นได้

4.หลังจากการสังเคราะห์วงจรให้อยู่ในระดับเกทหรือ Netlist แล้ว ข้อมูลนี้จะถูกใช้สำหรับจำลองการทำงานในเรื่อง ความถูกต้องของฟังก์ชัน พร้อมกับนำข้อมูลที่เกี่ยวข้องกับเวลาเข้ามาประกอบการพิจารณาด้วย ซึ่งตามปกติแล้วอุปกรณ์ ทางอิเล็กทรอนิกส์ทุกชิ้นจะมีเวลาหน่วงของการแพร่กระจาย (Propagation Delay Time) เสมอ ถึงแม้ว่าจะเป็น เวลาที่น้อยมากในระดับนาโนวินาทีก็ตาม แต่ถ้าภายในวงจรหนึ่งประกอบด้วยเกทของฟังก์ชันต่างๆ จำนวน 10,000 เกท ขึ้นไป เวลาดังกล่าวนี้จะสะสมกันมากขึ้น จนอาจทำให้การทำงานของวงจรรวมทั้งหมดผิดพลาดไป หรือไม่สามารถทำ งานในย่านความถี่สัญญาณนาฬิกาที่สูงได้

5.ผลิตเป็นวงจรจริง (Technology and device mapping) โดยนำข้อมูลที่ได้จากการสังเคราะห์มาผลิต ซึ่งอาจ จะอยู่ในรูปของแผงวงจรไฟฟ้า ที่ประกอบด้วยอุปกรณ์หลายๆ ชิ้นหรืออยู่ในรูปของวงจรรวม ASIC

6.ทำการตรวจสอบการทำงานและตัวแปรทางด้านเวลาทั้งหมด เพื่อความถูกต้องของวงจรเป็นครั้งสุดท้ายก่อนนำไปรวมเข้ากับอุปกรณ์อื่นๆ ให้เป็นระบบดิจิตอล เนื่องจากในขั้นตอนนี้ วงจรที่ออกแบบ จะประกอบด้วยจุดต่อทางอินพุทและเอาท์พุท ซึ่งเป็นจุดต่อสำหรับการรับและส่งสัญญาณกับภายนอก

2.9 FPGA

ในปัจจุบันมี FPGA อยู่ 4 ชนิดที่วางขายอยู่ในท้องตลาดได้แก่ Symmetrical Array, Row-Based, Hierarchical PLD และ Sea-of-Gates ซึ่งแต่ละชนิดก็มีลักษณะการเชื่อมต่อภายในและการโปรแกรม ที่แตกต่างกันไป นอกจากนี้ในการแบ่งประเภทของ FPGA อาจแบ่งได้ตามเทคโนโลยีที่ใช้ในการโปรแกรม ซึ่งมีอยู่ 2 แบบคือ การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพของตัวชิพ และการโปรแกรม โดยการใช้หน่วยความจำ

2.9.1 การโปรแกรมโดยการทำให้เกิดการเปลี่ยนแปลงทางกายภาพ

- Fuse เป็นวิธีการโปรแกรมที่สามารถทำได้เพียงครั้งเดียว ซึ่งหลังจากที่โปรแกรมแล้วจุดเชื่อม ต่อจะขาดจากกัน

- Anti Fuse เป็นวิธีการโปรแกรมที่คล้ายกับแบบ Fuse แต่ต่างกันที่หลังจากทำการโปรแกรม แล้วจุดเชื่อมต่อจะเชื่อมถึงกัน

2.9.2 การโปรแกรมโดยใช้หน่วยความจำ

- EEPROM Based FPGA FPGA ที่ใช้การโปรแกรมแบบนี้มักเรียกว่า CPLD ซึ่งเทคโนโลยีที่ใช้จะเหมือนกับ EEPROM ทำให้มีความจของเกทต่ำ โดยทั่วไปจะน้อยกว่า 20,000 เกท แต่ข้อดีของ EEPROM Based FPGA คือสามารถเก็บข้อมูลที่โปรแกรมลงไปได้โดยไม่จำเป็นต้องมีไฟเลี้ยง และในการโปรแกรมจะใช้ทรานซิสเตอร์ 1 ตัวต่อ 1 บิท ซึ่งการโปรแกรมสามารถทำได้ประมาณ 10,000 ครั้ง

- SRAM Based FPGA FPGA แบบนี้จะใช้เทคโนโลยีในการโปรแกรมเหมือนกับ *SRAM (Static RAM) ทำให้สามารถ โปรแกรมซ้ำได้โดยไม่จำกัดจำนวนครั้ง นอกจากนี้ยังมี* ความจุของเกทในระดับปานกลางถึงสูงมาก (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 เพื่อเก็บโปรแกรมและทำการ โหลดโปรแกรมลงในตัวชิพในขณะที่เริ่มต้นใช้งาน

2.9.3 ปัจจัยที่ทำให้การออกแบบ FPGA ทำได้ง่ายและสะดวกรวดเร็ว

1.ผู้ออกแบบไม่จำเป็นต้องทราบถึงโครงสร้างภายในของตัวชิพ เพียงแต่มีความรู้เกี่ยวกับขั้นตอนการออก แบบลอจิกก็เพียงพอแล้ว ต่างกับการใช้ไมโครโปรเซสเซอร์ซึ่งจำเป็นต้องศึกษาโครงสร้างภายในรวมถึง ภาษา Assembly ของไมโครโปรเซสเซอร์ตัวนั้นด้วย

2.มีการออกแบบโดยใช้ภาษาในการอธิบายการทำงานของวงจร หรือ HDL (Hardware Description Language)เป็นเครื่องมือในการออกแบบ ซึ่งเป็นวิธีการที่มีความยืดหยุ่นสูง ทำได้รวดเร็ว และไม่จำเป็น ต้องทราบถึงลักษณะของวงจรที่ต้องการว่าจะเชื่อมต่อกันอย่างไร เพียงแต่กำหนดลักษณะการทำงานให้มัน จากนั้นตัวซอฟต์แวร์จะทำ Synthesis and Optimize ให้ทั้งหมด นอกจากนี้ภาษาที่ใช้ยังเป็นมาตราฐาน เดียวกันสามารถใช้ได้กับชิพทุกตัวและทุกบริษัท

3.การโปรแกรมสามารถทำได้เองและใช้เวลาไม่นาน เพียงแค่ส่งข้อมูลผ่านสายดาวน์โหลดทางพอร์ตของ คอมพิวเตอร์ก็สามารถโปรแกรมตัวชิพขณะที่อยู่ในระบบได้ โดยไม่จำเป็นต้องถอดมาโปรแกรมข้างนอก ดังรูปที่ 2.3 และที่สำคัญสามารถโปรแกรมได้หลายครั้ง จึงทำให้ง่ายในการแก้ไขและพัฒนาโดยไม่ต้องเสีย ค่าใช้จ่ายเพิ่มแต่อย่างใด

2.10 การออกแบบโดยใช้ภาษาอธิบายพฤติกรรมของฮาร์ดแวร์

ในการออกแบบวงจรดิจิตอลนั้นสามารถทำได้โดยการวาดวงจร (Schematic) หรือใช้ภาษาอธิบายพฤติกรรม (Hardware Description Language) ของฮาร์ดแวร์ จากที่ได้กล่าวไปแล้วในบทที่ 1 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 หรือใช้ภาษาอธิบายการทำงานของวงจรจะทำ ได้สะดวกกว่า เนื่องจากวิธีการนี้ผู้ออกแบบไม่จำเป็นต้องคำนึงถึงเทคโนโลยีที่จะใช้สร้างไอซีและที่สำคัญ การออกแบบโดยวิธีนี้สามารถแก้ไขโมเดล (Model) หรือเปลี่ยนแปลงเทคโนโลยีได้สะดวกกว่า เพราะไม่ต้องวาดวงจร ใหม่ นั่นคือการออกแบบโดยใช้ภาษาอธิบายฮาร์ดแวร์ จะทำให้โมเดลที่ได้ไม่ขึ้นกับเทคโนโลยีสำหรับภาษาที่ใช้ สำหรับอธิบายพฤติกรรมของฮาร์ดแวร์ที่ใช้กันก็มี VHDL, AHDL และ Verylog เป็นต้น ส่วนรายละเอียด ของขั้นตอนในการออกแบบสามารถอธิบายได้ดังนี้

2.10.1 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 ในขั้นตอนนี้ ซอฟต์แวร์สังเคราะห์วงจรจะทำการแปลงโค้ด VHDL และทำการ Optimize เพื่อให้ได้วงจรตาม เทคโนโลยีที่เลือกใช้ในการสังเคราะห์วงจรนั้นวงจรระดับเกท (Gate Level) จะไม่เหมาะสมกับโครง สร้างที่มีอยู่ในอุปกรณ์ 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 ซอฟแวร์การสังเคราะห์วงจรก็จะมีการรายงานผลว่าโมเดลที่ออกแบบไปนั้น เป็นอย่างไร เช่นมีค่าความหน่วง (Delay) เท่าใด ใช้ทรัพยากรต่างๆใน FPGA อะไรบ้าง เมื่อมาถึงขั้น ตอนนี้ ผู้ออกแบบก็จะทราบว่าโมเดลเป็นไปตามข้อบังคับหรือไม่ ถ้าไม่ก็สังเคราะห์ใหม่จนกว่าจะเป็นไปตาม ที่กำหนด

2.10.2 การแบ่งวงจร (Partitioning) ขั้นตอนนี้เป็นการแบ่งวงจรที่ได้จากการสังเคราะห์เป็นส่วนย่อยๆ สำหรับลงใน CLBs, IOBs หรือองค์ ประกอบอื่นๆ ภายในอุปกรณ์ FPGA สำหรับเกณฑ์ที่ใช้ในการแบ่งคือให้แต่ละส่วนที่จะแยกออกจากกัน มีจำนวนสัญญาณที่เชื่อมต่อระหว่างกันน้อยที่สุดเท่าที่จะทำได้ เพื่อลดความหนาแน่นในตอนทำการเชื่อม ต่อสัญญาณ (routing) ในขั้นตอนนี้จะใช้ซอฟต์แวร์ทำโดยซอฟต์แวร์จะเทียบส่วนประกอบของวงจรเช่น เกท (gate), ฟลิป-ฟลอบ (flip-flop) ลงในทรัพยากรต่างๆ ที่มีอยู่ในอุปกรณ์ FPGAหลังจากทำขั้นตอน นี้เสร็จแล้วผู้ออกแบบสามารถที่จะทราบว่าวงจรใช้จำนวนทรัพยากรภายในอุปกรณ์ FPGA ไปเท่าไร ส่วนข้อมูลทางเวลานั้นผู้ออกแบบจะทราบเฉพาะความหน่วงภายในแต่ละส่วนเท่านั้น หรือที่เรียกว่าความหน่วงลอจิก(logic delay) ส่วนซอฟต์แวร์จะรวมเอาซอฟต์แวร์ย่อยอื่นๆ อีก เพื่อให้การทำ PPR (Partitioning Placement & Routing) เป็นไปอย่างต่อเนื่อง

2.10.3 การวางอุปกรณ์ (Placement)ขั้นตอนนี้เป็นการเลือกทำเลที่ตั้งของแต่ละส่วนของวงจรที่ผ่านการแบ่งวงจร (Partitioning) มาแล้วว่าควร จะอยู่ ณ ตำแหน่งไหนในอุปกรณ์ FPGA เพื่อให้ได้ผลลัพธ์ดีที่สุด เช่นวงจรส่วนไหนควรอยู่ใกล้กัน เพื่อจะ ได้ค้นหาเส้นทางได้ (route) ง่ายหรือช่วยลดความหน่วง จะเห็นได้ว่าตำแหน่งภายในอุปกรณ์ FPGA นั้นมี ความสำคัญเพราะถ้าจัดวางวงจรลงในตำแหน่งที่ไม่เหมาะสมแล้วจะทำให้ความหน่วงเพิ่มขึ้นหรือ Router ทำการค้นหาเส้นทางสัญญาณไดไม่หมด 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

2.10.4 การเชื่อมต่อสัญญาณ (Routing)ในขั้นตอนนี้เป็นการเชื่อมต่อสัญญาณระหว่างองค์ประกอบต่างๆ ภายในอุปกรณ์ FPGA ขั้นตอนนี้จะทำ ต่อเนื่องจากการวางอุปกรณ์ ในกรณีที่ทำการวางอุปกรณ์ไว้ไม่ดีซอฟต์แวร์ก็จะทำการเชื่อมต่อสัญญาณ ได้ไม่หมด (เนื่องจากจำนวนทรัพยากรสำหรับเชื่อมต่อสัญญาณนั้นมีอยู่จำกัด) หรือเกิดความหน่วงเกิน ค่าที่กำหนดในข้อบังคับ ผู้ออกแบบสามารถทำขั้นตอนนี้ได้โดยใช้ซอฟท์แวร์หรือผู้ออกแบบจะทำการ เชื่อมต่อสัญญาณด้วยตนเองก็ได้ แต่ทางที่ดีควรใช้ซอฟต์แวร์ทำดีกว่า นอกจากนั้นการกำหนดข้อบังคับทางเวลา จะช่วยให้ผลที่ได้จากการเชื่อมต่อสัญญาณดีขึ้นได้

2.10.5ความหน่วงด้านเวลา (Delay) ในการทำ FPGA นั้นความหน่วงที่เกิดขึ้นเป็นความหน่วงที่เกิดจากการวางตำแหน่ง (layout) ของอุปกรณ์ ซึ่งผู้ออกแบบไม่สามารถเข้าไปแก้ไขได้ แต่สามารถทำให้มีความหน่วงน้อยที่สุดได้ สำหรับความหน่วง ที่เกิดขึ้นนั้นแยกได้เป็นสองประเภทคือ

- ความหน่วงลอจิก (Logic delay) เป็นความหน่วงภายในองค์ประกอบของอุปกรณ์ FPGA เอง
- ความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณ (Routing delay)

เป็นความหน่วงที่เกิดจากการเชื่อมต่อสัญญาณระหว่างองค์ประกอบภายในอุปกรณ์ FPGA โดยปกติแล้ว ค่าความหน่วงลอจิกไม่ควรเกิน 50% ของค่าความหน่วงที่ยอมรับได้ เพราะความหน่วงที่เกิดจากการ เชื่อมต่อสัญญาณมักจะมีค่ามากกว่าค่าความหน่วงลอจิก ดังนั้นในการวางอุปกรณ์ และเชื่อมต่อสัญญาณ ผู้ออกแบบควรกำหนดข้อบังคับทางเวลาเพื่อให้ ซอฟต์แวร์ได้ทำงานอย่างมีประ

สิทธิภาพเพิ่มขึ้น และเพื่อ ให้ได้ผลลัพธ์ที่ดีขึ้นค่าความหน่วงที่ได้หลังจากการวางอุปกรณ์ และเชื่อมต่อสัญญาณแล้วจะมีค่าความ หน่วงที่ค่อนข้างแน่นอน ซึ่งผู้ออกแบบสามารถทราบได้ว่าโมเดลที่ออกแบบนั้น เป็นไปตามข้อกำหนด หรือไม่

2.10.6การจำลองการทำงานของวงจร (Simulation)ในขั้นตอนนี้เป็นขั้นตอนที่สำคัญอีกขั้นตอนหนึ่ง เพราะเป็นขั้นตอนที่ผู้ออกแบบตรวจสอบฟังก์ชันการ ทำงานของโมเดลว่าถูกต้องหรือไม่ มีข้อผิดพลาดตรงไหนเพื่อจะได้ทำการแก้ไขให้ถูกต้อง ในขั้นตอนนี้ จะมีซอฟต์แวร์ที่ใช้สำหรับทำการจำลองการทำงานของวงจรที่ใช้อยู่ เช่น Model Sim ของบริษัท Model Technology หรือ *Max Plus II ของบริษัท Altera ในการจำลองการทำงานของวงจร ควรทำทุกครั้งหลังจากที่มี* การทำแต่ละขั้นตอนหลักเสร็จแล้ว เพื่อจะได้ทราบว่าข้อผิดพลาดของโมเดล เกิดขึ้นตอนไหน จะได้แก้ไขข้อผิดพลาดตรงขั้นตอนนั้นๆ ได้เลย ไม่ต้องมาคอยตรวจหาขั้นตอนที่ทำให้ เกิดข้อผิดพลาด นั่นคือการทำการจำลองการทำงานของวงจร ต้องทำทั้งหลังการเขียนโค้ด, การสังเคราะห์วงจร และการทำ PPR การจำลองการทำงานของวงจรหลังจากที่เขียนโค้ดเสร็จแล้วนั้น ผู้ออก แบบสามารถทราบได้แค่โมเดลทำงานถูกต้องหรือไม่เท่านั้น (functional test) ยังไม่สามารถตรวจ สอบการทำงานในเชิงเวลาได้ถูกต้อง ในการจำลองการทำงานของวงจรหลังจากที่สังเคราะห์เป็นวงจรแล้ว เพื่อจรวจสอบว่าฟังก์ชันการทำงานยังคงถูกต้องหรือไม่ และค่าความหน่วงที่เกิดขึ้นเป็นไปตาม ข้อบังคับหรือไม่ มีข้อผิดพลาดเกิดขึ้นหรือไม่ถ้ามีจะแก้ไขให้ถูกต้อง

ในการจำลองการทำงานของวงจรหลังจากที่ทำการวางอุปกรณ์ การเชื่อมต่อสัญญาณ (post layout simulation) แล้วก็มีความสำคัญเช่นกันเพราะผลที่ได้จากการจำลองการทำงานของวงจรในตอนนี้ จะเป็นผลลัพธ์ของโมเดลเลย ซึ่งผู้ออกแบบนอกจากจะตรวจสอบฟังก์ชันการทำงาน *แล้วยังต้อง ตรวจสอบคุณสมบัติอื่นๆ เช่น ความหน่วงที่ได้จากการทำ PPR ในรูปแบบค่า* ความหน่วงมาตรฐาน (Standard Delay Format : SDF) ว่าตรงตามที่กำหนดหรือไม่ หรือตรวจสอบว่าวงจรรวม สามารถใช้งานที่ความถี่สูงสุดเท่าไรนั่นเอง ในการจำลองการทำงาน ของวงจรควรใช้ซอฟต์แวร์ตัวเดียวกันตอลดเพื่อจะได้เปรียบเทียบผลที่ได้จากขั้นตอนต่างๆ

2.10.7 การโปรแกรมอุปกรณ์ FPGA (Configuration)หลังจากที่โมเดลผ่านขั้นตอนต่างๆ จนกระทั่งผ่านการทำ PPR (Partitioning, Placement & Routing) แล้วนั้น ถึงตอนนี้ก็สามารถที่จะดาวน์โหลด (download) ลงในอุปกรณ์ FPGAได้แล้วในการดาวน์โหลดนี้ก่อนอื่นต้องแปลง *แบบวงจรรวมที่ได้เป็นข้อมูลวงจร* (configuration data) ซึ่งอยู่ในรูปของบิตสตรีม (bit stream) ก่อนแล้วจึงดาวน์โหลดลงไปเพื่อให้อุปกรณ์ FPGA มี ฟังก์ชันการทำงานตามโมเดลที่ผู้ออกแบบ

ต้องการ 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 ก็ไม่จำ เป็นต้องมีหน่วยความจำสำหรับเก็บข้อมูลวงจร เพราะว่าอุปกรณ์ FPGA ชนิดนี้เมื่อดาวน์โหลดข้อมูล วงจรลงไป ข้อมูลที่ดาวน์โหลดลงไปก็ยังคงอยู่ในอุปกรณ์ FPGA และครั้งต่อไปก็ใช้งานโมเดลที่ออก แบบไว้ได้เลย

2.11 การโปรแกรมแบบวิชวล (Visual Programming)

วิชวล (Visual) หมายถึง ภาพหรือสิ่งที่เรามองเห็น ดังนั้น วิชวลโปรแกรมมิง (Visual Programming- การโปรแกรมแบบวิชวล) จึงหมายถึง การโปรแกรมด้วยภาพหรือการเขียนโปรแกรมด้วยสิ่งที่เรามองเห็น

ภาพประกอบเบื้องต้นสำหรับใช้เขียนโปรแกรมแบบวิชวลมี 4 รายการดังต่อไปนี้

1. ฟอร์ม (Form) ฟอร์มเป็นวินโดว์ว่างใช้สำหรับออกแบบโปรแกรม การทำงานต่าง ๆ ของโปรแกรมจะปรากฏอยู่บนฟอร์ม
2. คอมโพเนนต์ (Component) คอมโพเนนต์เป็นส่วนประกอบต่าง ๆ ที่จะต้องใช้ในโปรแกรม เช่น เมนู (Menu) ,ปุ่มหรือบัตทอน (Button – ปุ่มสำหรับกดให้โปรแกรมทำงานชนิดหนึ่ง) เมสเสจบ็อกซ์ (Message Box–กรอบแสดงข้อความไดอะล็อกบ็อกซ์) กรอบแสดงข้อความไดอะล็อกบ็อกซ์ (Dialog Box - กรอบสำหรับโต้ตอบกับโปรแกรม)
3. คำสั่งสำหรับจัดลักษณะและการทำงานของคอมโพเนนต์
4. คำสั่งสำหรับควบคุมการทำงานของโปรแกรม

วิชวลโปรแกรมมิ่งช่วยให้เราสามารถเขียนโปรแกรมได้ง่าย สะดวก รวดเร็ว และถูกต้อง

ปาสคาล และเดลไฟล

ปาสคาล (Pascal) เป็นภาษาคอมพิวเตอร์ที่สร้างขึ้นโดยมีจุดมุ่งหมายเพื่อให้ผู้เขียนโปรแกรมได้เรียนรู้เกี่ยวกับวิธีการเขียนโปรแกรมที่ดี ซึ่งจะเป็นพื้นฐานในการเขียนโปรแกรมขนาดใหญ่ที่มีความซับซ้อนได้โดยสะดวก ปาสคาลที่นิยมใช้กันมากในไมโครคอมพิวเตอร์ ได้แก่ เทอร์โบปาสคาล (Terbo Pascal) ซึ่งเป็นผลงานของบริษัท บอร์แลนด์อินเตอร์เนชันแนล (Borland International Inc) ประเทศสหรัฐอเมริกา เทอร์โบปาสคาล เวอร์ชัน 1.0 วางจำหน่ายในปี 1984 ตั้งแต่เวอร์ชัน 5.5 ซึ่งวางจำหน่ายในปี 1999 เป็นต้นมา เทอร์โบปาสคาลได้เพิ่มวิธีการเขียนโปรแกรมแบบออบเจ็กต์ ซึ่งเรียกว่า การเขียนโปรแกรมแบบ OOP (Object – Oriented Programming) และเรียกชื่อปาสคาลชนิดนี้ว่า ออบเจกต์ปาสคาล (Object Pascal) ออบเจ็กต์ปาสคาลมีทั้งชนิดที่รันในดอสและรันในวินโดวส์

ออบเจ็กต์ปาสคาลและเดลไฟ (Delhi) ต่างก็เป็นภาษาคอมพิวเตอร์ซึ่งใช้สำหรับเขียนโปรแกรมเพื่อสั่งให้คอมพิวเตอร์ทำงานต่าง ๆตามที่เราต้องการ ออบเจ็กต์ปาสคาลและเดลไฟลมีคำและกฎเกณฑ์ในการเขียนเหมือนกันเกือบทุกประการ แต่มีวิธีเขียนโปรแกรมต่างกัน กล่าวคือ

ออบเจ็กปาสคาลใช้วิธีเขียนโปรแกรมด้วยอักษรและเครื่องหมาย แต่เดลไฟใช้วิธีเขียนแบบวิชวล ดังนั้นผู้ที่เขียนโปรแกรมด้วยออบเจ็กต์ปาสคาลจึงสามารถเปลี่ยนมาเขียนโปรแกรมด้วยเดลไฟลได้อย่างรวดเร็ว เพราะเปลี่ยนแปลงเฉพาะวิธีการเท่านั้น ซึ่งวิธีการของเดลไฟลจะทำให้เราสามารถเขียนโปรแกรมได้ง่าย และรวดเร็วกว่าวิธีการของออบเจ็กต์ปาสคาล เดลไฟลผลิต โดยบริษัทบอร์แลนด์อินเตอร์เนชันแนล เดลไฟล เวอร์ชันแรกนั้นวางจำหน่ายในปี 1995 โดยแบ่งการจำหน่ายเป็น 4 ชุด คือ

1. Delphi Standard (ชุดมาตรฐาน)
2. Delphi Professional (ชุดระดับมืออาชีพ)
3. Delphi Client/Server Suite (ชุดเครือข่ายคอมพิวเตอร์)
4. *Delphi Enterprise (ชุดสำหรับองค์การ)*

เดลไฟล เป็นวิชวลโปรแกรมมิ่ง ซึ่งมีวิธีการเขียนโปรแกรมโดยการนำคอมโพเนนต์ที่เรามองเห็นมาวางในฟอร์ม แล้วกำหนดลักษณะและการทำงานให้กับคอมโพเนนต์นั้น ในบางโปรแกรมอาจจะต้องเขียนคำสั่งควบคุมการทำงานของโปรแกรมเพิ่มเติม ต่อจากนั้นจึง คอมไพล์ (Compile) และรัน (Run) โปรแกรมนั้นได้ ผลจากการคอมไพล์โปรแกรมจะได้ไฟล์ชนิด .EXE ซึ่งสามารถนำไปรันในคอมพิวเตอร์เครื่องอื่นได้อย่างอิสระ โดยไม่ต้องเกี่ยวข้องกับเดลไฟลอีกเลย

เดลไฟลเป็นออบเจ็กต์ปาสคาลประเภทที่สามารถสร้างโปรแกรมใช้งานได้อย่างรวดเร็วภาษาคอมพิวเตอร์ประเภทนี้มีชื่อย่อว่า *RAD (Rapid Application Development)* ซึ่งหมายความว่าเราจะสามารถเขียนโปรแกรมเพื่อรันในวินโดวส์ได้เร็วกว่าและง่ายกว่าการเขียนโปรแกรมออบเจ็กต์ปาสคาลแบบธรรมดา

อีเวนต์ (Event)

อีเวนต์ หมายถึง ทุกสิ่งทุกอย่างที่เกิดขึ้นในระบบคอมพิวเตอร์ เช่น การเลื่อนเมาส์ การคลิกเมาส์ การกดคีย์ การเลื่อนวินโดวส์ การเปลี่ยนขนาดของวินโดวส์ วินโดวส์จะเก็บอีเวนต์ทั้งหมดไว้ในคิว (Queue) ที่มีชื่อว่า โกลบอล อีเวนต์คิว (Global Even Queue – คิวของอีเวนต์ทั้งหมด ซึ่งบางครั้งที่เรียกว่า System – Wide Queue หรือ Hardware Even Queue) โกลบอลอีเวนต์คิวใช้สำหรับเก็บอีเวนต์ของทุดแอปพลิเคชันที่รันในวินโดวส์

เมสเสจ (Massage)

*เมสเสจ หมายถึง ชุดของข่าวสารที่เป็นรายละเอียดของแต่ละอีเวนต์ วินโดวส์จะสร้างเมส*เสจของอีเวนต์ โดยการส่งอีเวนต์ที่เกิดขึ้นให้แก่ ไดรเวอร์ (Driver - โปรแกรมที่คอมพิวเตอร์ใช้ติดต่อกับอุปกรณ์ชนิดนั้น)ที่สอดคล้องกับอีเวนต์นั้น เช่น เมื่อเราคลิกเมาส์จะเกิดอีเวนต์คลิกเมาส์

วินโดวส์จะส่งอีเวนต์นี้ให้แก่เมาส์ไดรเวอร์ และเมาส์ไดรเวอร์จะสร้างเมสเสจคลิกเมาส์ ให้แก่วินโดวส์ ไดรเวอร์จะเก็บเมสเสจจากนี้ไว้ในซิสเต็มคิว (System Queue - คิวของระบบ) ต่อจากนั้นวินโดวส์ จึงส่งเมสเสจจากซิสเต็มคิวไปที่แอปพลิเคชันคิว (Application Queue - คิวเก็บเมสเสจของแอปพลิเคชัน) ของแอปพลิเคชันต่าง ๆ ที่กำลังทำงานอยู่ในขณะนั้นซึ่งเป็นผลให้แอปพลิเคชันได้รับเมสเสจจากวินโดวส์ เนื่องจากวินโดวส์สามารถรันแอปพลิเคชันคิวสำหรับแอปพลิเคชันนั้นโดยเฉพาะความสัมพันธ์ระหว่างอีเวนต์กับเมสเสจ

หลักการทำงานของแอปพลิเคชันที่รันในวินโดวส์ก็คือ การดำเนินการกับเมสเสจ การรับส่งเมสเสจกับวินโดวส์หรือกับแอปพลิเคชั่นอื่น เมสเสจทั้งหมดของวินโดวส์กำหนดไว้เป็นคอนสแตนต์ (Constant - ค่าที่ไม่เปลี่ยนแปลง) เช่น เมื่อเราเลือกรายการหนึ่งจากเมนูที่ปรากฏบนจอภาพจะมีเมสเสจ WM_COMMAND เกิดขึ้น ในบรรดาเมสเสจเหล่านี้ มีหลายเมสเสจที่เกิดขึ้นพร้อมกับแวริเอเบิล (Variable - ค่าที่เปลี่ยนแปลงได้) จำนวนหนึ่งซึ่งนำมาใช้ในแอปพลิเคชั่นได้ เช่น เมื่อเรากดปุ่มซ้ายของเมาส์ วินโดวส์จะสร้างเมสเสจ WM_LBUTTONDOWN ขึ้นมา เมสเสจนี้ประกอบด้วยแวริเอเบิล ซึ่งเป็นโคออร์ดิเนต (Coordinate - ตำแหน่ง) ของเมาส์พอยน์เตอร์ เราสามารถนำค่าดังกล่าวนี้มาใช้ได้ เช่น ใช้สำหรับแสดงผลบนจอภาพ ใช้ในการตรวจสอบตำแหน่งของเมาส์ พอยน์เตอร์

2.12 โครงสร้างโปรแกรมเดลไฟล (The Structure of a Delphi Program)

การเขียนโปรแกรมเดลไฟลให้ได้ประโยชน์มากขึ้น ถ้านักพัฒนาโปรแกรมเข้าใจโครงสร้างของ Object Pascal ซึ่งเป็นรากฐานของเดลไฟล

IDE เป็นเครื่องมือที่ช่วยให้สามารถเขียนโปรแกรมได้เร็วขึ้น โดยที่ IDE ช่วยเขียนคำสั่งที่จำเป็นต้องใช้ และช่วยให้นักพัฒนาสามารถออกแบบจอภาพและเขียนคำสั่งที่ต้องการให้ทำงานเมื่อเหตุการณ์ (even)เกิดขึ้น

IDE จะช่วยเขียนโค้ดคำสั่งซึ่งจะอยู่ภายใต้กฎเกณฑ์ของ Object Pascal และช่วยสร้าง resource ที่ต้องใช้ตามกฎเกณฑ์การทำงานของระบบปฏิบัติการแบบ Windows

โพรซีเยอร์ (Procedure)

โพซีเยอร์ เป็นกลุ่มคำสั่งงานที่ใช้ในการทำงานอย่างใดอย่างหนึ่ง อาจจะเรียกอีกอย่างว่า "โปรแกรมย่อย" เช่น การคำนวณภาษี การแสดงจอภาพเพื่อรับค่าจากผู้ใช้งาน เป็นต้น

ตัวอย่างของโพรซีเยอร์

```
1. Procedure Button Click;
2. Begin
3. Edit1.Text:='Hello World';
4. End;
```

ในโพรซีเยอร์ตัวอย่างนี้จะแสดงข้อความ 'Hello World' ในคอนโทรล Edit1 เมื่อผู้งานกดปุ่ม Botton1 โดยที่ชื่อโพรซีเยอร์จอยู่บรรทัดที่ 1 และมีคำสั่ง begin และ end คลุมคำสั่งที่ใช้ในการทำงานของโพรซีเยอร์นี้

- คำสั่งอยู่ในโพรซีเยอร์อาจมีได้ตั้งแต่คำสั่งเดียวจนถึงหลาย ๆ หน้ากระดาษ
- เนื่องจากภาษา Object Pascal เป็นภาษาที่ Case-Insensitive ดังนั้น การตั้งชื่อต่าง ๆ ที่ใช้อักขระตัวใหญ่และอักขระตัวเล็กจะไม่แตกต่างกัน เช่น Edit1 หรือ edit1 จะเป็นการอ้างถึงชื่อเดียวกัน
- เพื่อความสะดวกในการอ่านโปรแกรมและความง่ายในการแก้ไขโปรแกรม ควรจะใช้อักขระตัวใหญ่ผสมกับตัวเล็ก

การเขียนคำอธิบายโปรแกรม (Comment)

เมื่อโปรแกรมมีมากขึ้น ซับซ้อนมากขึ้นเราต้องเขียนคำอธบายเพิ่มเติม ให้คนอื่น ๆ หรือแม้แต่ตัวเราเองเข้าใจได้ในภายหลัง โดยสามารถทำได้ 3 รูปแบบได้แก่

1. การใช้วงเล็บปีกกา {} ครอบข้อความที่เป็นคำอธิบายโปรแกรม จะครอบคลุมกี่บรรทัดก็ได้
2. ใช้ // นำหน้าข้อความที่เป็นคำอธิบายโปรแกรม มีผลแค่บรรทัดเดียวเท่านั้น
3. การใช้เครื่องหมาย (* กับ *) ครอบข้อความที่เป็นคำอธิบายโปรแกรม จะครอบคลุมกี่บรรทัดก็ได้

สำหรับนักพัฒนาแอพพลิเคชันแบบมืออาชีพมักจะเขียนคำอธิบายที่ค่อนข้างละเอียดและมีรูปแบบที่ถูกกำหนดโดยตกลงร่วมกันระหว่างทีมงานพัฒนาแอพพลิเคชัน ซึ่งทำให้ทุกคนสามารถเข้าใจการทำงานภายในได้โดยไม่ยากนัก

ตัวแปร (Variable)

ในการเขียนโปรแกรม เราจะใช้ตัวแปร (Variable) เป็นเสมือนที่พักข้อมูลชั่วคราว โดยจะนำข้อมูลที่เก็บในตัวแปรต่าง ๆ นำไปประมวลผลในโปรแกรม การใช้งานตัวแปรจึงเป็นเรื่องที่สำคัญยิ่งในการเขียนโปรแกรม

การประกาศตัวแปร (Variable Declaration)

ก่อนการใช้งานตัวแปรทุกครั้งเราจะต้องประกาศตัวแปร โดยเราจะทำการประกาศตัวแปรไว้ที่ตอนต้นของบล็อกของการทำงาน ด้วยรูปแบบต่อไปนี้

```
var ชื่อตัวแปร: ชนิดตัวแปร;
```

สำหรับการประกาศตัวแปรใน Delphi นั้นเราจะประกาศไว้ในตอนต้นของโปรแกรม (หรือตอนต้นของโปรแกรมย่อย) ซึ่งเราสามารถประกาศค่าตัวแปรชนิดเดียวกันพร้อม ๆ กันได้

การตั้งชื่อตัวแปร

เพื่อให้การใช้งานไม่สับสน เราจึงควรตั้งชื่อตัวแปรให้สื่อความหมาย คือสามารถเดาความหมายได้ไม่ยากนัก ไม่สั้นไม่ยาวเกินไป และควรตั้งชื่อให้สัมพันธ์กับการใช้งานในโปรแกรม สำหรับการตั้งชื่อตัวแปรใน Delphi 5.0 นั้น เรามีสิ่งที่ควรพิจารณา และถือว่าเป็นกฎการตั้งชื่อตัวแปรดังนี้

1. ชื่อตัวแปรความยาวไม่เกิน 63 ตัวอักษร (ถ้าเกินกว่านั้น Delphi จะตัดส่วนที่เกินออกไปอัตโนมัติ)
2. ต้องขึ้นต้นด้วยตัวอักษร หรือเส้นขีดล่าง (_) เท่านั้น
3. ตัวอักษรถัดไปสามารถเป็นได้ทั้งตัวอักษร ตัวเลขได้
4. ไม่อนุญาตให้ใช้สัญญาณบางตัวในชื่อตัวแปร เช่น $, %, *, @
5. ชื่อตัวแปรต้องไม่ตรงกับคำสงวน เช่น Begin, End, procedure
6. หลีกเลี่ยงการใช้ชื่อตัวแปรตรงกับชื่อชนิดข้อมูลที่มีใน Delphi เพราะจำทำให้เราไม่สามารถใช้ชนิดของข้อมูลนั้นได้

ชนิดข้อมูลพื้นฐาน

ตัวแปรจะทำหน้าที่เก็บข้อมูลต่าง ๆ ซึ่งข้อมูลที่จะนำมาเก็บนั้นมีอยู่หลายรูปแบบ แต่ละแบบมีขนาดที่ใช้เก็บแตกต่างกัน อีกทั้งวิธีการจัดการกับข้อมูลแบบต่าง ๆ ก็ต่างกันด้วย ทำให้เราต้องกำหนดชนิดของตัวแปรที่ใช้งาน ซึ่งใน Delphi มีตัวแปรชนิดต่าง ๆ ให้ใช้งานดังนี้

หมวด	ชนิดข้อมูล	จำนวนไบต์ที่ใช้เก็บ	ค่าที่เป็นไปได้
Integer	Shortint	1 ไบต์	จำนวนเต็มระหว่าง-128 ถึง 127
	Smallint	2 ไบต์	จำนวนเต็มระหว่าง-32,766 ถึง 32,767
	Integer	4 ไบต์	จำนวนเต็มระหว่าง-2,147,483,648 ถึง 2,147,483,647
	Longint	4 ไบต์	เท่ากับชนิด Integer
	Byte	1 ไบต์	จำนวนเต็มบวกตั้งแต่ 0 ถึง 255
	Word	2 ไบต์	จำนวนเต็มระหว่าง 0 ถึง 65,535
Real	Single	4ไบต์	จำนวนทศนิยมมีเศษได้ 8 หลัก มีค่าตั้งแต่ –3.4 x 1036 ถึง –1.5 x 10-45 และ 1.5 x 10-45 ถึง 3.4 x 1036
	Double	8 ไบต์	จำนวนทศนิยมมีเศษได้ 15 หลัก มีค่าตั้งแต่ –1.7 x 10308 ถึง –5.0 x 10-324 และ 5.0 x 10-324 ถึง 1.7 x 10308
	Comp	8 ไบต์	จำนวนทศนิยมมีเศษได้ 20 หลัก มีค่าตั้งแต่ –226+1ถึง 226-1
	Extended	10ไบต์	จำนวนทศนิยมมีเศษได้ 20 หลัก มีค่าตั้งแต่ –1.1x104932 ถึง –3.4x10-4932 และ 3.4x10-4932 ถึง 1.1x104932
	Curency	8ไบต์	จำนวนทศนิยมที่จัดเก็บแบบจำนวนเต็ม มีค่าตั้งแต่ 922,337,203,685,477.5808 ถึง 922,337,203,685,477.5807
	Real	6 ไบต์	จำนวนทศนิยมมีเศษได้ 12 หลัก มีค่าตั้งแต่ –1.7x1036 ถึง –2.9x10-39 และ 2.9x10-39 ถึง 1.7x1036

หมวด	ชนิดข้อมูล	จำนวนไบต์ที่ใช้เก็บ	ค่าที่เป็นไปได้
Currency	AnsiChar	1ไบต์	เป็นตัวอักษร 1 ตัวตามมาตรฐานของ ANSI
	WideChar	2ไบต์	เป็นตัวอักษร 1 ตัวตามมาตรฐานของ Unicode
	Char	1ไบต์	เท่ากับชนิด AnsiChar
String	ShortString	-	เก็บข้อความที่เก็บได้ 255 ตัวอักษร
	AnsiString	-	เก็บข้อความยาวได้ไม่จำกัด
	String	-	เก็บข้อความได้เทียบเท่า AnsiString
Pchar	Pchar	-	เป็นข้อความที่มีการปิดท้ายด้วย Null เพื่อบอการสิ้นสุดของข้อความ
Boolean	Boolean	1ไบต์	เก็บค่า True หรือ False เท่านั้น
Pointer	Pointer	4 ไบต์	เป็นตัวแปรที่เก็บแอดเดรสของหน่วยความจำที่เก็บข้อมูลที่เราต้องการ
Variant	Variant	-	เป็นตัวแปรทีกำหนดชนิดแน่นอนไม่ได้ แต่สามารถแก้ไขชนิดของข้อมูลได้ในภายหลัง

ตารางที่ 2.1 แสดงตัวแปรชนิดต่างๆใน Delphi

ชนิดข้อมูลที่สร้างจากข้อมูลพื้นฐาน

เราสามารถผสมผสาน หรือเพิ่มเติมความสามารถให้กับพื้นฐานให้เป็นข้อมูลที่มีรูปแบบต่าง ๆ เพื่อให้เหมาะแก่การใช้งานในรูปแบบต่าง ๆ ดังนี้

ข้อมูลชนิด Enumerated

Enumerated คือชนิดข้อมูลที่เก็บเป็นชุด ซึ่งมีจำนวนในชุดนั้นแน่นอน ทำให้สะดวกต่อการอ้างอิงสมาชิกในชุดนั้น ๆ ซึ่งมีรูปแบบการประกาศดังนี้

```
type ชนิดตัวแปร_Enumerated = (ค่าข้อมูลตัวที่1, ค่าข้อมูลตัวที่2,.....,ค่าข้อมูลตัวที่ n)
```

ข้อมูลชนิด SubRange

SubRange คือการกำหนดข้อมูลที่เป็นช่วง โดยจะต้องกำหนดว่าค่าต่ำสุด และค่าสูงสุดของช่วงนั้น ซึ่งชนิดข้อมูลจะกำหนดเป็นช่วงได้ต้องเป็น Boolean, Char, Integer หรือ Enumerate เท่านั้น ซึ่งมีรูปแบบการประกาศค่าดังนี้

```
type ชนิดตัวแปร_Sub_Range = (ค่าต่ำสุด....ค่าสูงสุด)
```

ข้อมูลชนิด Array

อาร์เรย์ (Array) เป็นรูปแบบการเก็บข้อมูลหลายตัวแปรที่มีชนิดเดียวกันไว้ด้วยกัน โดยการอ้างอิงถึงข้อมูลแต่ละตัวในอาร์เรย์เราจะใช้อินเด็กซ์เป็นตัวอ้างอิงถึง ซึ่งมีรูปแบบประกาศดังนี้

```
var ชื่ออาร์เรย์ : array [อินเด็กซ์เริ่มต้น....อินเด็กซ์สุดท้าย] of ชนิดข้อมูลที่เก็บอาร์เรย์;
```

หรือกำหนดชนิดข้อมูลก่อน แล้วค่อยประกาศตัวแปรอาร์เรย์ก็ได้

```
type ชนิดของอาร์เรย์ = array [อินเด็กซ์เริ่มต้น...อินเด็กซ์สุดท้าย] of ชนิดข้อมูลที่เก็บในอาร์เรย์;
```

Dynamic Array

ไดนามิคอาร์เรย์ (Dynamic Array) เป็นอาร์เรย์ที่มีคุณสมบัติเสริมเพิ่มเติม คือสามารถเปลี่ยนแปลงขนาดที่ใช้เก็บได้ในขณะที่แอพพลิเคชันทำงาน การที่เราสามารถเปลี่ยนแปลงขนาดของอาร์เรย์ได้ ทำให้เราใช้งานหน่วยความจำอย่างคุ้มค่า คือพื้นที่หน่วยความจำเท่าที่จำเป็นจะต้องเก็บจริง ๆ เพราะบางครั้งเราอาจจะไม่ทราบว่าจะต้องใช้อาร์เรย์ขนาดสักเท่าไหร่ในการเก็บ หรือต้องการเปลี่ยนขนาดของอาร์เรย์ตามสภาพการทำงานที่เปลี่ยนไป

ข้อมูลชนิด Set

Set คือกลุ่มของข้อมูลที่มีสมาชิกเท่าใดก็ได้ โดยไม่สนใจลำดับของสมาชิกที่อยู่ภายใน (สมาชิกข้างในอาจจะเป็นข้อมูลแบบ Enumerate หรือ SubRange ก็ได้) อีกทั้งถ้าสมาชิกมีซ้ำกัน ก็ถือว่าเป็นตัวเดียวกัน ซึ่งมีรูปแบบการประกาศดังนี้

```
Var ชื่อตัวแปร_Set: set of ชนิดข้อมูลที่เก็บใน_set;
```

หรือกำหนดชนิดข้อมูลก่อน แล้วค่อยประกาศตัวแปรอาร์เรย์ก็ได้

```
type ชนิดตัวแปร_Set of ชนิดข้อมูลที่เกี่ยวใน_set;
```

ข้อมูลชนิด Record

เร็คคอร์ด (Record) คือชนิดข้อมูลที่สามารถเก็บข้อมูลชนิดอื่น ๆ ไว้ภายในได้ โดยเราเรียกข้อมูลแต่ละตัวที่อยู่ภายในว่า ฟิลด์ (Field) และก่อนจะใช้งานเร็คคอร์ดได้ เราต้องประกาศชนิดของเร็คคอร์ดก่อน จากนั้นจึงค่อยประกาศตัวแปรดังรูปแบบดังต่อไปนี้

```
type ชื่อตัวแปร_Record = record

    ฟิลด์ที่_1 : ชนิดข้อมูลของฟิลด์ที่_1;
    ฟิลด์ที่_2 : ชนิดข้อมูลของฟิลด์ที่_2;
                :
    ฟิลด์ที่_สุดท้าย : ชนิดข้อมูลของฟิลด์ที่_สุดท้าย;
```

หรือจะกำหนดชนิดข้อมูลก่อน แล้วค่อยประกาศตัวแปรอาร์เรย์ก็ได้

```
var ชื่อตัวแปร_Record : ชนิดข้อมูลแบบ_Record;
```

เมื่อเราใช้งานเร็คคอร์ด เราจะอ้างอิงข้อมูลที่เก็บอยู่ภายในเร็คคอร์ดได้โดยคั่นชื่อของเร็คคอร์ดกับข้อมูลที่อยู่ภายในด้วย

ข้อมูลชนิด Object

ออบเจ็กต์ (Object) คือ ชนิดข้อมูลที่ใช้สร้างออบเจ็กต์ หรือคอมโพเนนต์ของ Delphi ซึ่งในการประกาศเราจะต้องมีส่วนของพรอพเพอร์ตี้ เมธอด ของออบเจ็กต์ด้วย ซึ่งจะคล้ายคลึงกับการประกาศตัวแปรแบบ Record

การกำหนดค่าให้กับตัวแปร (Variable Assignment)

เมื่อเราประกาศตัวแปรเสร็จ ก็พร้อมใช้งานกับตัวแปรเหล่านั้น ซึ่งในการใช้งาน เราจะเก็บข้อมูลในตัวแปรได้โดยการกำหนดค่าให้ตัวแปร ซึ่งการกำหนดค่าให้กับตัวแปรจะต้องกำหนดให้มีชนิดข้อมูลของตัวแปรด้วย สำหรับรูปแบบการกำหนดค่าตัวแปรเป็นดังนี้

```
ชื่อตัวแปร:= ค่าที่กำหนด;
```

ค่าคงที่ (Constant)

สำหรับข้อมูลบางอย่างที่ต้องใช้ในการทำงานของโปรแกรม แต่เป็นข้อมูลที่ตายตัวไม่มีการเปลี่ยนแปลงแล้ว เราสามารถกำหนดให้เป็นค่าคงที่ได้ ซึ่งค่าคงที่สามารถถูกเรียกใช้ได้จากที่ใด ๆ ก็ได้ในโปรแกรม ทำให้สะดวกสบายในการอ้างอิง และก่อนการใช้งานค่าคงที่ทุกครั้ง เราก็ต้องประกาศค่าคงที่ด้วยรูแบบต่อไปนี้

```
Const ชื่อค่าคงที่ = ค่าที่กำหนด;
```

Expression และ Statement

ในการเขียนโปรแกรมเราก็มีการประกาศตัวแปร กำหนดค่าให้ตัวแปร และมีการสั่งให้ตัวแปรทำงานตามที่เราต้องการ ซึ่งเรียกเอารูปแบบของประโยคที่สั่งให้โปรแกรมทำงานว่า Statement ซึ่งจะใช้เครื่องหมาย : (Semi - Colon) แสดงการสิ้นสุดของ Statement

สำหรับส่วนของประโยคที่มีการประมวลผลเราเรียกกันว่า Expression นั่นคือ Expression จะต้องมีผลลัพธ์การทำงานออกมา

ตัวดำเนินการใน Delphi (Operator)

ในการทำงานของแอพพลิเคชันนั้น ที่จริงแล้วก็คือการประมวลผลข้อมูลต่าง ๆ ซึ่งรูปแบบของการประมวลผลเราจะใช้ Operator หรือตัวดำเนินการมาเป็นสัญลักษณ์แทนการประมวลผลแต่ละแบบ ซึ่งแบ่งตัวดำเนินการได้เป็น 4 กลุ่ม ได้แก่

1. ตัวดำเนินการทางคณิตศาสตร์
2. ตัวดำเนินการเพื่อเปรียบเทียบ
3. ตัวดำเนินการด้านตรรกะ
4. ตัวดำเนินการเกี่ยวกับเซต

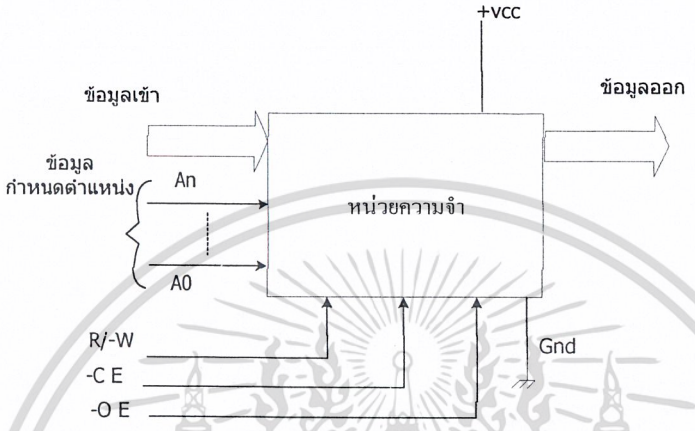
ตัวอย่างตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operator)

ตัวดำเนินการทางคณิตศาสตร์ ได้แก่ เครื่องหมายทางคณิตศาสตร์ต่าง ๆ ดังรายละเอียดในตาราง

ตัวดำเนินการ	คำอธิบาย
+	เป็นการบวกตัวเลข 2 จำนวนเข้าด้วยกัน หากใช้กับข้อมูลประเภท String ก็จะเป็นการนำ 2 ข้อความมาต่อกัน เช่น 'Inter]+]Networking]จะเท่ากัน'InterNetworking'เป็นต้น
-	เป็นการลบตัวเลข
*	เป็นการคูณตัวเลข
/	เป็นการหารตัวเลขทศนิยม (ในหมวด Real) ได้ผลลัพธ์เป็นเลขทศนิยม
Div	เป็นการหารตัวเลขทศนิยม (ในหมวด Integer) ได้ผลลัพธ์เป็นจำนวนเต็ม เช่น 7 div 2 ได้ผลัพธ์เป็น 3 เป็นต้น
Mod	เป็นการหาเศษจากการหารเลขจำนวนเต็ม เช่น 7 mod 5 ได้ผลลัพธ์เป็น 2

2.13 ลักษณะของหน่วยความจำแบบ Ram

หน่วยความจำแบบแรมคือหน่วยความจำที่มีการเข้าถึงได้ โดยไม่ต้องใส่ลำดับ (Sequential Access) ต้องการข้อมูล ที่ตำแหน่งใดก็ได้ โดยส่ง Address (ตัวเลขระบุตำแหน่ง) บล็อกไดอะแกรมของแรมแสดงได้ดังรูป 2.21 ซึ่งในที่นี้จะใช้แรมแบบสแตติก

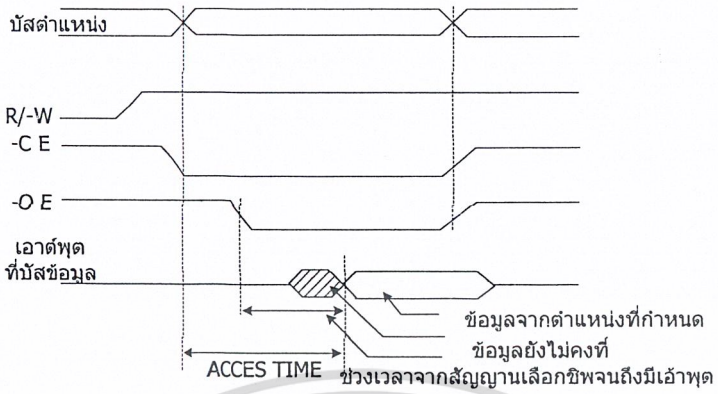


รูปที่2.21บล็อกไดอะแกรมของหน่วยความจำแบบแรม

จากบล็อกไดอะแกรม บัสสำหรับรับข้อมูลเข้าและส่งข้อมูลออกจะเป็นบัสเดียวกัน แต่เป็นบัสแบบ 2 ทิศทาง คือ เป็นแบบบัสขาเข้าเมื่ออยู่ในช่วงของการเขียน และเป็นแบบบัสขาออกเมื่ออยู่ในช่วงการอ่านหน่วยความจำแรมแบบสแตติกหรือ SRAM มีหลายขนาดให้เลือกเช่นเบอร์ 2114 เป็นแรมขนาด 1k x 4 เบอร์ 6116 เป็นแรมขนาด 2k x 8 เป็นต้นในโครงงานนี้ใช้ SRAM เบอร์ 62256 ซึ่งมีขนาด 32K x 8

ลำดับขั้นในการอ่านข้อมูลจากแรม

ในการอ่านข้อมูลจากแรมนั้น ต้องใช้สัญญาณควบคุมการอ่านและเขียนมาร่วมในการควบคุมการทำงานด้วย โดยทำการกำหนดให้สัญญาณควบคุมการอ่านและเขียนนี้แอคตีฟตามการลักษณะของการอ่านหรือเขียนข้อมูลดังต้องการ เช่น ถ้าให้สัญญาณนี้อยู่ในสภาวะแอคตีฟอ่าน (*สมมติ = 1*) การทำงานก็จะเป็นการอ่านข้อมูลจากหน่วยความจำตามตำแหน่งที่ระบุ ซึ่งลำดับขั้นของการอ่านข้อมูลนี้สามารถเขียนเป็นไดอะแกรมเวลาได้ดังรูปที่ 2.22



รูปที่ 2.22 ไดอะแกรมเวลาของการอ่านข้อมูลจากแรม

และมีลำดับขั้นเป็นดังนี้

1. ซีพียูส่งข้อมูลของตำแหน่งที่ต้องการอ่านออกไปบนบัสตำแหน่ง (กำหนดให้ในขณะนี้สัญญาณการอ่านแอคตีฟ)
2. ซีพียูส่งสัญญาณเลือกใช้ชิพและส่งไปยังหน่วยความจำ
3. ซีพียูรอเวลาสักระยะหนึ่ง นับตั้งแต่ส่งข้อมูลเพื่อกำหนดตำแหน่งเข้าไป ซึ่งเวลานี้ก็คือ Access time ซึ่งผ่านไปแล้ว ข้อมูลจากหน่วยความจำก็จะปรากฏบนบัสข้อมูล ซึ่งซีพียูจะอ่านข้อมูลในช่วงนี้เข้าไป
4. สัญญาณเลือกชิพเลือกแอคตีฟ บัสข้อมูลกลับสู่สภาวะอิมพีแดนซ์สูง

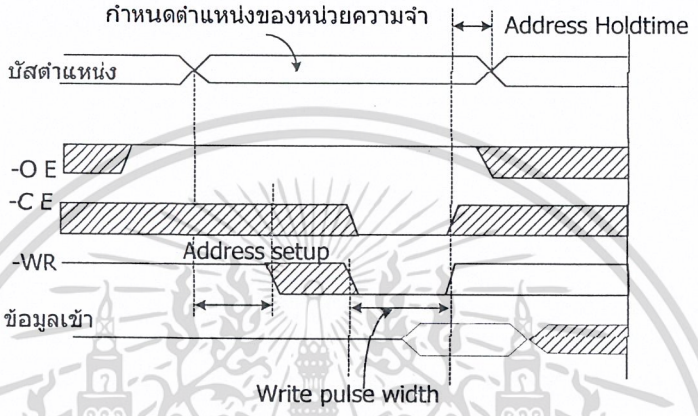
ลำดับขั้นในการเขียนข้อมูลลงหน่วยความจำแรม

ในการเขียนข้อมูลลงหน่วยความจำแบบแรม จะมีลำดับขั้นดังนี้

1. ซีพียูส่งข้อมูลของตำแหน่งที่ต้องการให้เขียนไปบนบัสตำแหน่ง และอาจพร้อมทั้งสัญญาณการเลือกใช้ชิพ ($\overline{CE}$)
2. ข้อมูลที่จะเป็นข้อมูลอินพุต ถูกส่งเข้ามายังอินพุตของแรม
3. หน่วยความจำต้องการเวลาสักระยะหนึ่ง เพื่อให้ข้อมูลในการกำหนดตำแหน่งอยู่ในสภาวะคงที่ ซึ่งช่วงเวลานี้คือ Address setup time ซึ่งจะมีค่าประมาณ 20 nS จากนั้นระบบจึงส่งสัญญาณควบคุมการเขียนให้อยู่ในสภาวะแอคตีฟ

4. หลังจากที่สัญญาณควบคุมการเขียนแอคตีฟสักระยะหนึ่ง หน่วยความจะทำการนำข้อมูลจากอินพุตเข้าเขียนเข้าไปยังหน่วยความจำ และเมื่อสัญญาณควบคุมการเขียนเลิกแอคตีฟ ก็เป็นอันสิ้นสุดขบวนการเขียน

ลำดับของการเขียนข้อมูลลงแรม สามารถแสดงได้โดยไดอะแกรมเวลาดังรูปที่ 2.23



รูปที่ 2.23 ไดอะแกรมเวลาของการเขียนหน่วยความจำ

จากรูปเห็นว่าได้ว่ากำหนดให้สัญญาณ $\overline{OE}$ อยู่ในสภาวะที่ไม่แอคตีฟ เพื่อไม่ให้มีสัญญาณเอาต์พุต ออกมาที่บัสข้อมูล แต่ถ้าสัญญาณ $\overline{OE}$ อยู่ในสภาวะแอคตีฟ และขณะนั้น สัญญาณ $\overline{CE}$ แอคตีฟด้วย ก็จะทำให้มีเอาต์พุตปรากฏออกมาที่บัสขาออก แต่เมื่อมีสัญญาณการเขียนเข้ามา บัสทางออกนี้ก็จะมีสภาวะเป็นอิมพีแดนซ์สูงเช่นกัน

บทที่ 3
การออกแบบและการสร้าง

ในโครงงานนี้ได้ใช้ FPGA ที่เป็นแบบ SRAM Base FPGA รุ่นที่ใช้ เป็น FLEX 10 K ซึ่งมีความจุของเกต ประมาณ 10,000 เกต ซึ่งที่เลือกใช้เนื่องจาก มีความจุของเกตมากพอที่จะนำไปใช้งาน การโปรแกรมจะใช้เวลาน้อย อยู่ในระดับนาโนวินาที การโปรแกรมทำได้ง่าย และสะดวกในการทดลอง ไอซีที่ทำหน้าที่แปลงสัญญาณอนาลอกเป็นดิจิตอล (A/D Converter) เลือกใช้ไอซีเบอร์ 0808 ซึ่งมีความละเอียดในการแปลงสัญญาณ 8 บิต หรือระดับ 256 ระดับ เป็นวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลแบบ ประมาณค่าต่อเนื่อง(Successive Approximation)ซึ่งสามารถหาซื้อได้ง่ายตามท้องตลาด ราคาไม่แพงมากนัก และสามรถนำมาใช้งานง่ายในการที่จะนำมาติดต่อกับ Processor ต่างๆ

3.1 ส่วนการแปลงสัญญาณอนาลอกเป็นดิจิตอล

จากที่ได้ทราบคุณสมบัติต่างของไอซี ADC 0808 ซึ่งได้กล่าวไว้ในบทที่ 2 แล้วนั้น ต่อจากนั้นจะทำการออกแบบเพื่อนำไปใช้งานโดยข้อมูลแรงดันต่างๆรวมถึงการเชื่อมต่อไปใช้งานจะแสดงไว้ในดาต้าชีตของปริญญานิพนธ์นี้

ไอซี ADC 0808 ซึ่งเป็นไอซีที่ทำหน้าที่รับสัญญาณอินพุทแล้วเปลี่ยนเป็นสัญญาณดิจิตอลเพื่อส่งให้ซีพียูประมวลผลซึ่งคุณสมบัติของไอซีจะเป็นแบบโมโนลิทิคซีมอส (Monolithic CMOS) ขนาด 8 บิต สามารถรับสัญญาณอินพุทได้ 8 แชนเนล ใช้หลักการประมาณค่าโดยลำดับ (Successive Approximation) ง่ายในการนำมาเชื่อมต่อกับ FPGA สำหรับสัญญาณนาฬิกา จากภายนอกป้อนให้กับ ไอซี ADC 0808 โดยความถี่ที่ใช้คือ 1 เมกกะเฮิร์ต ซึ่งไอซี ADC 0808 สามารถใช้งานช่วงความถี่ตั้งแต่ 10 ถึง 1200 กิโลเฮิร์ต

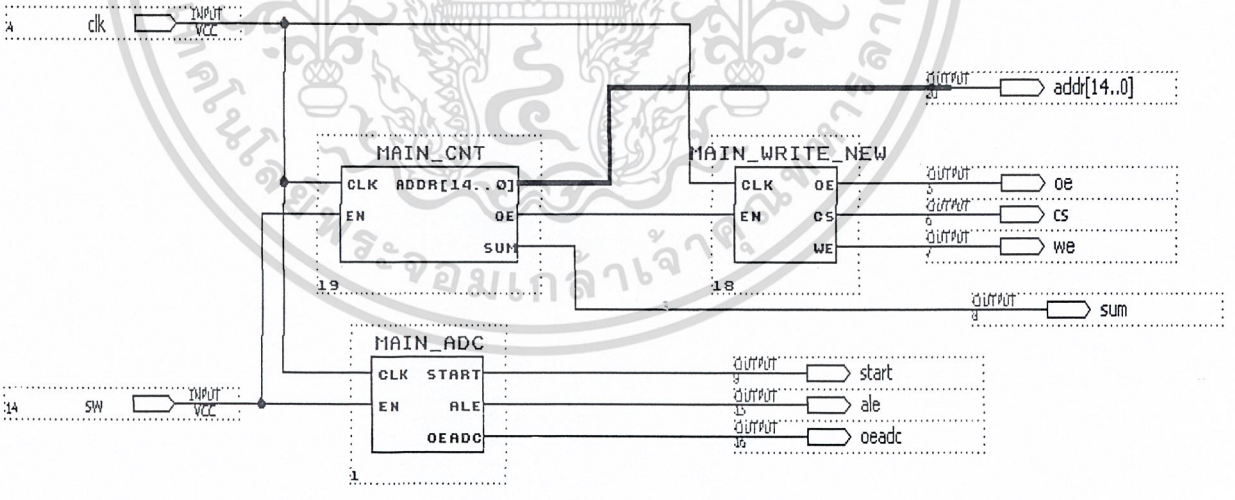
การทำงานของวงจรเริ่มต้นจาก FPGA ทำการส่งสถานะ “1” มาให้กับขา STRAT และ ALE เพื่อเริ่มกระบวนการแปลงสัญญาณ โดยในระหว่างนั้น FPGA จะทำการหน่วงเวลาไว้ประมาณ 100 ไมโครวินาที หลังจากนั้น FPGA จะส่งสถานะ “1” มายังขา OE ของไอซี ADC 0808 เพื่ออินาเบิล (Enable) ข้อมูลที่ทำการแปลงเสร็จเรียบร้อยแล้วโดยจะมีเอาท์พุทออกมาทางขา *21,20,19,18,8,15,14* และขา *17* ของไอซี ADC *0808* จากนั้นข้อมูลดิจิตอลที่ถูกแปลงแล้วจะส่งต่อไปยังแรม เพื่อทำการบันทึกข้อมูล ในส่วนของคอมพิวเตอร์จะทำการจัดการกับข้อมูลที่ได้นำไป

แสดงผลซึ่งจะใช้โปรแกรม เดลไฟล์ ในการแสดงผลออกเป็นกราฟเพื่อวิเคราะห์ข้อมูลที่ได้ ในส่วนของแรงดันอ้างอิงที่ใช้ในกระบวนการแปลงสัญญาณจะใช้แรงดัน +5 โวลท์ ป้อนเข้าที่ขา + Vref และขา – Vref จะต่อลงกราวด์ โดยสัญญาณอินพุทที่ป้อนเข้ามาจะอยู่ในช่วง 0 ถึง 5 โวล์ท ความละเอียดในการแปลงสัญญาณอยู่ที่ 256 ระดับ หรือ อินพุทมีการเปลี่ยน 0.0195 โวล์ท อินพุทจะเปลี่ยนไป 1 บิต

3.2 หลักการทำงานของ FPGA

3.2.1 การบันทึกข้อมูล

จากรูปที่ 3.1 เป็นการแสดงบล็อกไดอะแกรมการบันทึกข้อมูล โดยข้อมูลจะถูกส่งมาจากไอซี ADC 0808 แล้วทำการบันทึกข้อมูลที่ได้ไว้ในแรมทั้ง 2 ตัว ซึ่ง FPGA จะทำหน้าที่ในการควบคุมการทำงานทั้งหมด เริ่มต้นจากทำการเลือกสวิตช์ Write Mode จะทำให้วงจรการบันทึกเริ่มทำงาน แต่วงจรส่งข้อมูลจะถูกระงับ จากนั้น FPGA จะทำการควบคุมให้ไอซี ADC 0808 แปลงสัญญาณอินพุท เป็นข้อมูลเลขฐาน 2 ขนาด 8 บิต และข้อมูลที่ได้นี้จะส่งไปยังแรม โดยข้อมูลจากไอซี ADC 0808 ของแชนเนล 1 จะส่งไปยังแรมตัวที่ 1 และข้อมูลจากไอซี ADC 0808 ของแชนเนล 2 จะส่งไปยังแรมตัวที่ 2 ต่อจากนั้น FPGA จะทำการส่งสัญญาณการควบคุม OE ,CS และWE ไปยังแรมทั้ง 2 ตัว เพื่อควบคุมให้อยู่ในสถานะของการ Write



รูปที่ 3.1 บล็อกไดอะแกรมการบันทึกข้อมูล

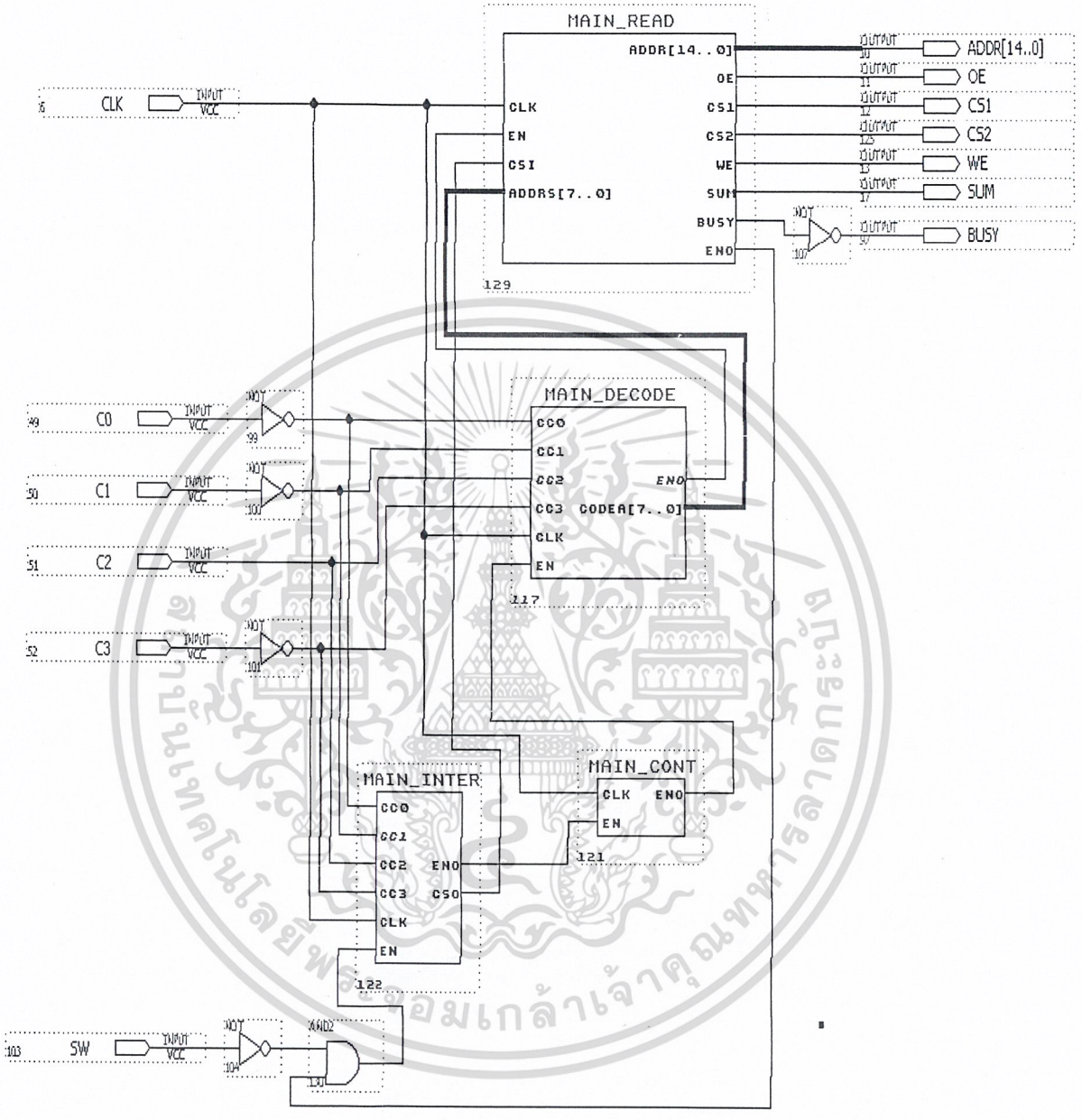
บล็อกไดอะแกรม MAIN_CNT ทำหน้าที่เป็นตัว Timer โดยจะควบคุมบล็อกไดอะแกรม MAIN_WRITE_NEW ให้ส่งสัญญาณการควบคุม OE ,CS และWE ไปยังแรมทั้ง 2 ตัว ทุกๆ 5 วินาที นอกจากนี้บล็อกไดอะแกรม MAIN_CNT ยังทำหน้าที่เป็นตัวกำหนดที่อยู่ (Address) ของข้อมูลอีกด้วย โดยจะเริ่มต้นจาก Address 0000H

3.2.2 การส่งข้อมูลไปยังคอมพิวเตอร์

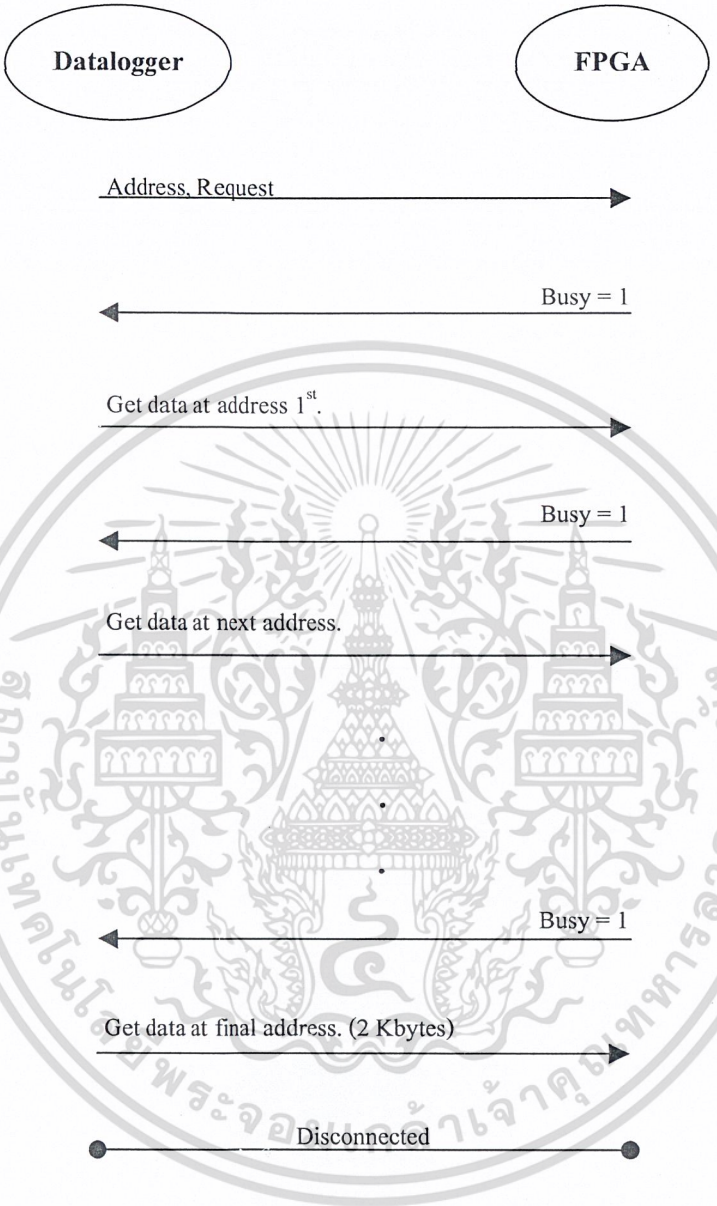
ในโครงงานนี้จะทำการติดต่อกับคอมพิวเตอร์โดยผ่านพอร์ตขนาน และใช้โปรแกรมเคลไฟล์ เป็นภาษาที่ใช้เขียนโปรแกรม DATALOGGER ในส่วนของโปรแกรม DATALOGGER จะทำการอธิบายในหัวข้อต่อไป

การส่งข้อมูลไปยังคอมพิวเตอร์เริ่มต้นจากทำการเลือกสวิตช์ Read Mode จะทำให้วงจรการส่งข้อมูลเริ่มทำงาน แต่วงจรบันทึกข้อมูลจะถูกระงับ จากนั้นโปรแกรม DATALOGGER จะทำการส่งสัญญาณร้องขอการส่งข้อมูลมายัง FPGA โดยในสัญญาณร้องขอนี้จะกำหนดแชนเนลมาด้วย จากนั้นก็จะส่งค่าที่อยู่ของข้อมูลตามมา โดยส่งผ่านสายสัญญาณควบคุมของพอร์ตขนาน เมื่อ FPGA ได้รับสัญญาณร้องขอก็จะทำการส่งสัญญาณควบคุม OE ,CS และWE ไปยังแรมตัวที่กำหนดมาในสัญญาณร้องขอ โดยสัญญาณควบคุมจะเป็นการกำหนดให้แรมอยู่ในสถานะ Read และ FPGA จะส่งค่าที่อยู่ของข้อมูลไปยังแรม ในการส่งข้อมูลไปยังคอมพิวเตอร์จะส่งครั้งละ 2 กิโลไบต์

จากรูปที่ 3.2 เป็นการแสดงบล็อกไดอะแกรมการส่งข้อมูลไปยังคอมพิวเตอร์ บล็อกไดอะแกรม MAIN_INTER ทำหน้าที่เป็นตัวตรวจจับสัญญาณการร้องขอเมื่อมีสัญญาณการร้องขอส่งเข้ามาก็จะทำการอนุญาตให้มีการส่งข้อมูลได้ และยังทำหน้าที่เป็นตัวกำหนดการนำส่งข้อมูลจากแชนเนลใดด้วย บล็อกไดอะแกรม MAIN_DECODE ทำหน้าที่รับสัญญาณที่อยู่ของข้อมูลจากคอมพิวเตอร์ บล็อกไดอะแกรม MAIN_READ ทำหน้าที่เป็นตัวส่งสัญญาณควบคุมการ Read ไปยังแรม กำหนดแชนเนล ส่งที่อยู่ของข้อมูลไปยังแรม และส่งสัญญาณ Busy ไปให้คอมพิวเตอร์



รูปที่ 3.2 บล็อกไดอะแกรมการส่งข้อมูลไปยังคอมพิวเตอร์



รูปที่ 3.3 การติดต่อระหว่างโปรแกรม DATALOGGER กับ FPGA

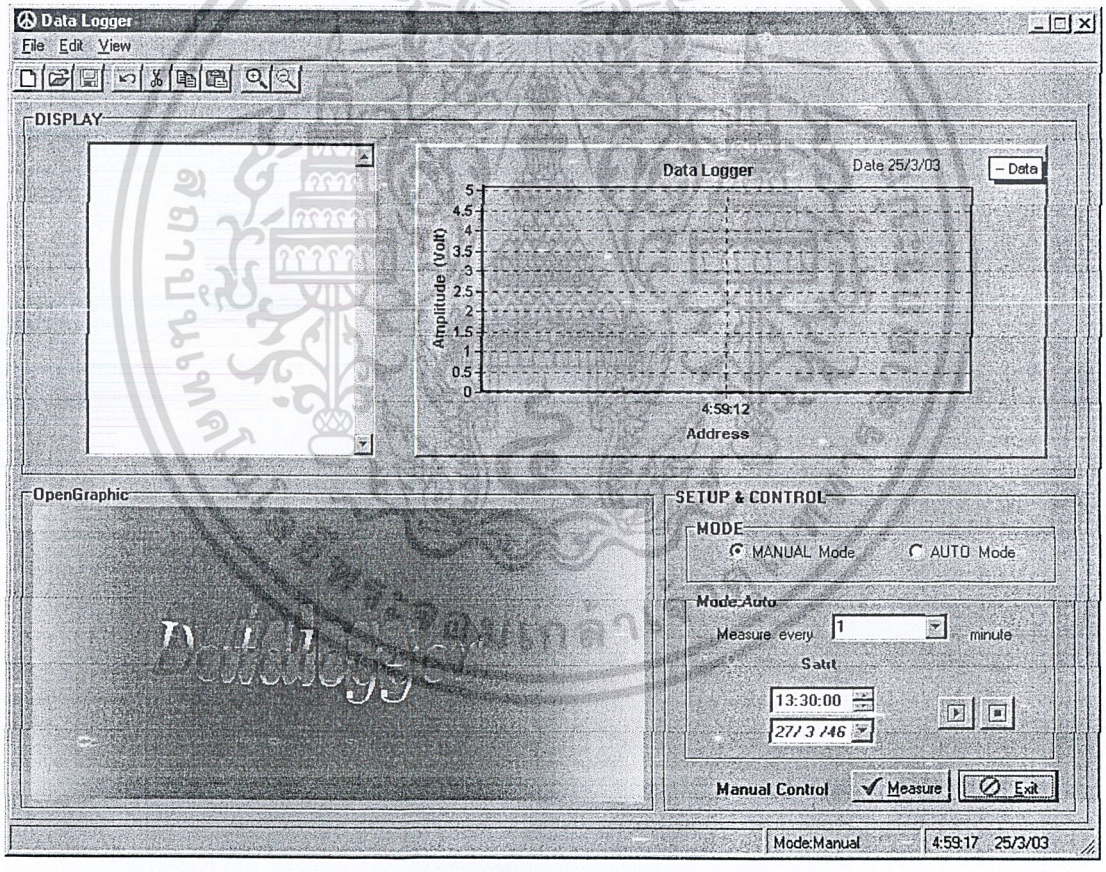
จากรูปที่ 3.3 เป็นการแสดงการติดต่อระหว่างโปรแกรม DATALOGGER กับ FPGA เริ่มต้นจากโปรแกรม DATALOGGER ส่งสัญญาณร้องขอ และที่อยู่มาให้ FPGA จากนั้น FPGA จะทำการดึงข้อมูลจากที่อยู่แรกในแรมออกมา แล้วก็ส่งสัญญาณ Busy = 1 ไปให้โปรแกรม DATALOGGER จากนั้นโปรแกรม DATALOGGER จะทำการรับข้อมูลและเก็บไว้ในฐานข้อมูล

ต่อมาเมื่อ FPGA จะทำการดึงข้อมูลจากที่อยู่ถัดไปในแรมออกมา แล้วก็ส่งสัญญาณ Busy = 1 ไปให้โปรแกรม DATALOGGER จากนั้นโปรแกรม DATALOGGER จะทำการรับข้อมูลและเก็บไว้ในฐานข้อมูล จะทำอย่างนี้ไปเรื่อยๆ จนกว่าข้อมูลที่ได้จะครบ 2 Kbytes

3.3 โปรแกรม DATALOGGER

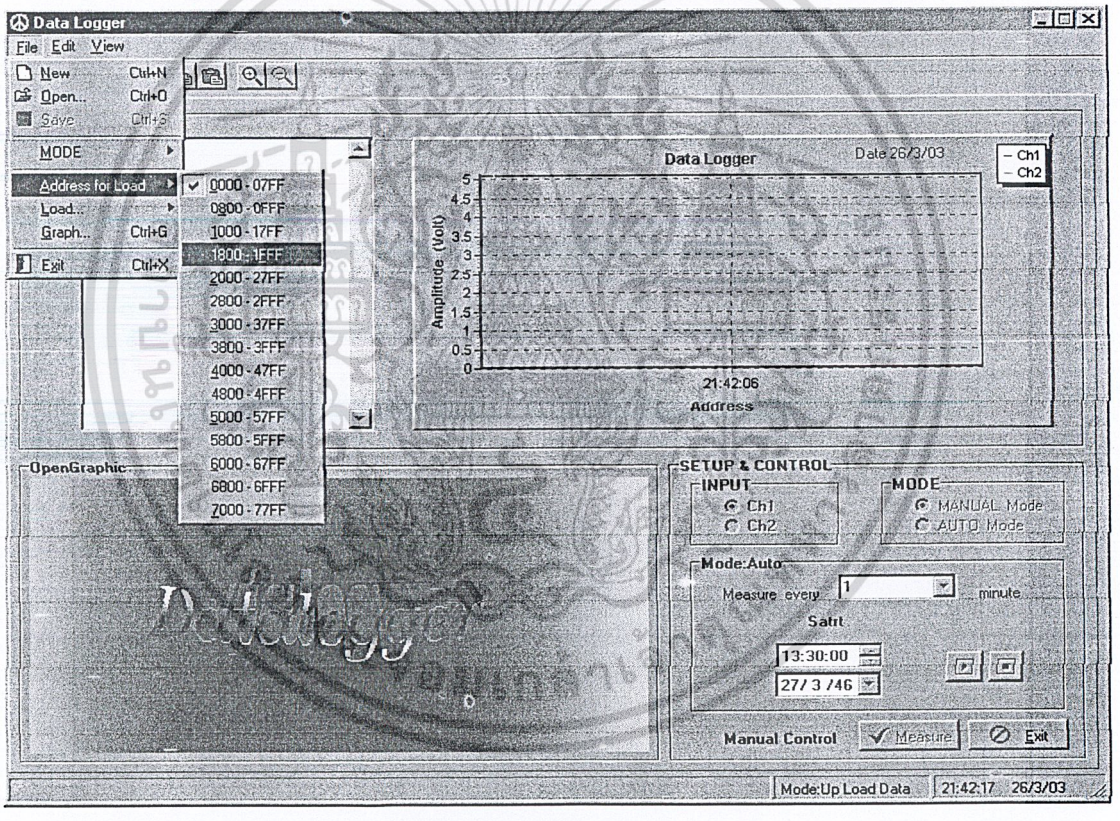
3.3.1 Measurement mode

เป็นโหมดการรับข้อมูลโดยตรงจาก ไอซี ADC 0808 ในการใช้งานขั้นแรกจะต้องแชนเนลในการรับข้อมูล จากนั้นเลือกโหมดในการรับข้อมูล



รูปที่ 3.4 การใช้งาน Measurement mode

- **Manual mode** ผู้ใช้สามารถรับข้อมูลโดยการคลิกที่ปุ่ม Measure จากนั้นข้อมูลที่ได้จะถูกแสดงออกมาในรูปของตัวอักษรและกราฟโดยแกน Y เป็นค่าแรงดัน และแกน X แสดงเวลาที่ทำการรับข้อมูล
- **Auto mode** ผู้ใช้จะต้องทำการเลือกรอบเวลาในการรับข้อมูล เวลาและวันที่เริ่มการรับข้อมูล จากนั้นคลิกปุ่ม *Start* เมื่อถึงเวลาและวันของการเริ่มรับข้อมูลโปรแกรม DATALOGGER จะทำการรับข้อมูลทุกๆ รอบเวลาของการรับข้อมูล และข้อมูลที่ได้จะถูกแสดงออกมาในรูปของตัวอักษรและกราฟ



รูปที่ 3.5 การใช้งาน Up load mode

3.3.2 Up load mode

เป็นโหมดการรับข้อมูลจากแรมโดยมี FPGA เป็นตัวควบคุม ในการใช้งานผู้ใช้จะต้องทำการเลือกที่อยู่ของข้อมูล จากนั้นเลือกแชนเนลที่จะรับข้อมูล เมื่อเลือกเสร็จข้อมูลจะถูกแสดงออกมา การนำข้อมูลที่ได้จากแรมมาทำการแสดงในรูปของกราฟ ทำได้คลิกที่เมนู Graph หลังจากนั้นข้อมูลก็จะแสดงออกมาในรูปของกราฟโดยแกน Y เป็นค่าแรงดัน และแกน X เป็นที่อยู่ของข้อมูล

บทที่ 4

การทดลองและผลการทดลอง

1 .ผลการทดลองการแปลงอนาลอกเป็นดิจิตอล(Analog To Digital : A/D)

ทดสอบการป้อนไฟ เข้าทางอินพุท เป็นไฟตรงตั้งระดับสูงสุด ที่ระดับ +5โวลท์ เละใช้ค่าแรงดันอ้างอิง +5โวล์ท จะได้ผลดังตารางที่ 4.1

Vin	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	0
2	0	1	1	0	0	1	1	1
3	1	0	0	1	1	0	0	1
4	1	1	0	0	1	1	0	1
5	1	1	1	1	1	1	1	1

ตารางที่ 4.1 ผลการทดลองวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล

จากตารางผลการทดลองเราสามารถหาค่า แรงดันไฟตรงจากสัญญาณ Digital ที่ทดลองได้ ซึ่งสามารถหาค่าความผิดพลาดของวงจรได้

จากสัญญาณ ดิจิตอล 8 บิต เราสามารถแยกระดับสัญญาณได้ $2^8 = 256$ ค่า

จากค่า Full scale resolution = 256 Divisions (0-255 digit)

Full scale resolution = 5 Volt

เราคำนวณหาค่า Analog อินพุท จากค่า Digital ที่วัดได้ดังนี้

1) เมื่อค่า ดิจิตอล ที่วัดได้ = 00110100

ค่า Division = 32+16+3=51

ดังนั้นค่าAnalog Input Voltage(51/255)x5= 1 Volt

2) เมื่อค่า ดิจิตอล ที่วัดได้ = 01100111

ค่า Division = 64+32+4+2+1=103

ดังนั้นค่าAnalog Input Voltage(103/255)x5=2.019Volt

3) เมื่อค่า ดิจิตอล ที่วัดได้ = 10011001

ค่า Division = 128+16+8+1=153

ดังนั้นค่าAnalog Input Voltage(153/255)x5=3.0Volt

4) เมื่อค่า ดิจิตอล ที่วัดได้ = 11001101

ค่า Division = 128+64+8+4+1=205

ดังนั้นค่าAnalog Input Voltage(205/255)x5=4.019Volt

5) เมื่อค่า ดิจิตอล ที่วัดได้ = 11111111

ค่า Division = 128+64+32+16+8+4+2+1=225

ดังนั้นค่าAnalog Input Voltage(225/255)x5= 5 Volt

จะเห็นว่าจากการทดลองป้อนค่าอินพุทโวล์ทเตจ และเอาท์พุทที่ได้จะมีค่าใกล้เคียงกันมาก ซึ่งมีผิดพลาดบ้างเล็กน้อยอาจเนื่องจาก วงจรA/D มีความผิดพลาดอยู่แล้วบ้าง และการทดลองที่ค่าแรงดันต่ำๆ ในส่วนของ LSB จะมีสัญญาณไม่หยุดนิ่ง แต่ไม่ค่อยมีผลเท่าใดนักเนื่องจากค่าที่ได้มีนัยสำคัญต่ำ

2 การทดลองการเก็บข้อมูลลงโปรแกรม DATA LOGGER โดยตรง

1.ทำการทดลองโดยใช้แหล่งจ่ายไฟตรงขนาด 2.51 V ต่อเข้าที่ Channel 1

2. เลือก Auto Mode

3. เลือกรับข้อมูลทุกๆ 1 นาที

4. ตั้งค่าเวลาเริ่มต้นที่ 16:09:00 น. ของวันที่ 26 มีนาคม 2546

ผลการทดลอง

======== Result:1 ========

Mode : Auto

16:09:00

26/3/03

CH1 = 2.51 V

======== Result:2 ========

Mode : Auto

16:10:00

26/3/03

CH1 = 2.51 V

======== Result:3 ========

Mode : Auto

16:11:00

26/3/03

CH1 = 2.51 V

======== Result:4 ========

Mode : Auto

16:12:00

26/3/03

CH1 = 2.51 V

======== Result:5 ========

Mode : Auto

16:13:00

26/3/03

CH1 = 2.51 V

======== Result:6 ========

Mode : Auto

16:14:00

26/3/03

CH1 = 2.51 V

======== Result:7 ========

Mode : Auto

16:15:00

26/3/03

CH1 = 2.51 V

======== Result:8 ========

Mode : Auto

16:16:00

26/3/03

CH1 = 2.51 V

======== Result:9 ========

Mode : Auto

16:17:00

26/3/03

CH1 = 2.51 V

======== Result:10 ========

Mode : Auto

16:18:00

26/3/03

CH1 = 2.51 V

======== Result:11 ========

Mode : Auto

16:19:00

26/3/03

CH1 = 2.51 V

======== Result:12 ========

Mode : Auto

16:20:00

26/3/03

CH1 = 2.51 V

======== Result:13 ========

Mode : Auto

16:21:00

26/3/03

CH1 = 2.51 V

======== Result:14 ========

Mode : Auto

16:22:00

26/3/03

CH1 = 2.51 V

======== Result:15 ========

Mode : Auto

16:23:00

26/3/03

CH1 = 2.51 V

======== Result:16 ========

Mode : Auto

16:24:00

26/3/03

CH1 = 2.51 V

======== Result:17 ========

Mode : Auto

16:25:00

26/3/03

CH1 = 2.51 V

======== Result:18 ========

Mode : Auto

16:26:00

26/3/03

CH1 = 2.51 V

======== Result:19 ========

Mode : Auto

16:27:00

26/3/03

CH1 = 2.51 V

======== Result:20 ========

Mode : Auto

16:28:00

26/3/03

CH1 = 2.51 V

======== Result:21 ========

Mode : Auto

16:29:00

26/3/03

CH1 = 2.51 V

======== Result:22 ========

Mode : Auto

16:30:00

26/3/03

CH1 = 2.51 V

======== Result:23 ========

Mode : Auto

16:31:00

26/3/03

CH1 = 2.51 V

======== Result:24 ========

Mode : Auto

16:32:00

26/3/03

CH1 = 2.51 V

======== Result:25 ========

Mode : Auto

16:33:00

26/3/03

CH1 = 2.51 V

======== Result:26 ========

Mode : Auto

16:34:00

26/3/03

CH1 = 2.51 V

======== Result:27 ========

Mode : Auto

16:35:00

26/3/03

CH1 = 2.51 V

======== Result:28 ========

Mode : Auto

16:36:00

26/3/03

CH1 = 2.51 V

======== Result:29 ========

Mode : Auto

16:37:00

26/3/03

CH1 = 2.51 V

======== Result:30 ========

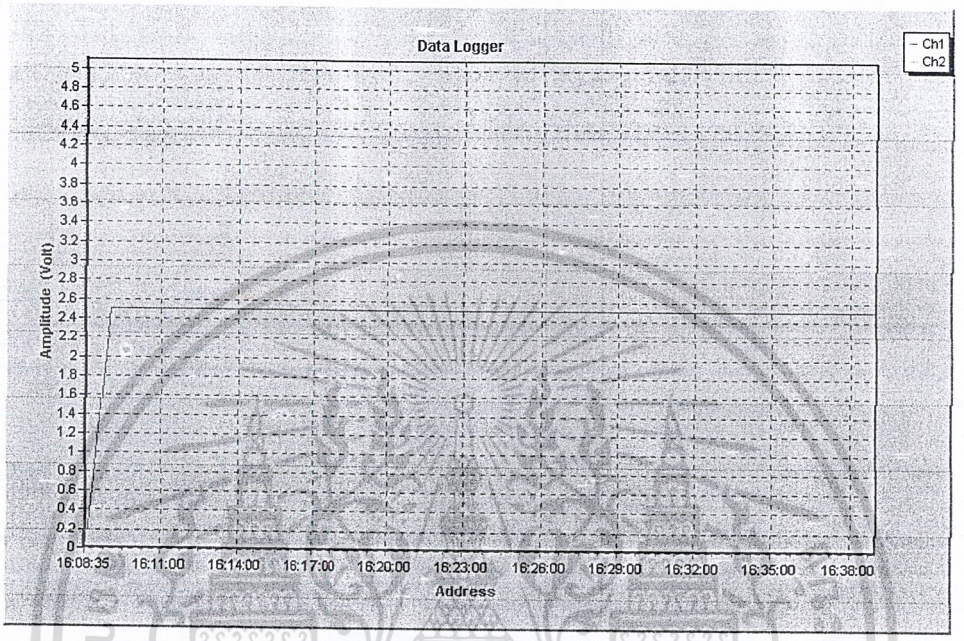
Mode : Auto

16:38:00

26/3/03

CH1 = 2.51 V

กราฟที่ได้จากการทดลองที่ 2



3. การทดลองบันทึกข้อมูลลงแรมและการเก็บลงบน PC (Up Load Data Mode)

1. ทำการทดลองโดยใช้แหล่งจ่ายไฟตรงขนาด 2.51 V ต่อเข้าที่ Channel 1 และ แหล่งจ่ายไฟตรงขนาด 5 V ต่อเข้าที่ Channel 2
2. เลือก Switch Write data
3. บันทึกข้อมูลลง RAM ทั้ง 2 ตัว เป็นเวลา 1 นาที
4. ใช้โปรแกรม Datalogger Load ข้อมูลจาก RAM ทั้ง 2 ตัว

ผลการทดลอง

ข้อมูล RAM1 (2kByte)

Addr : Value

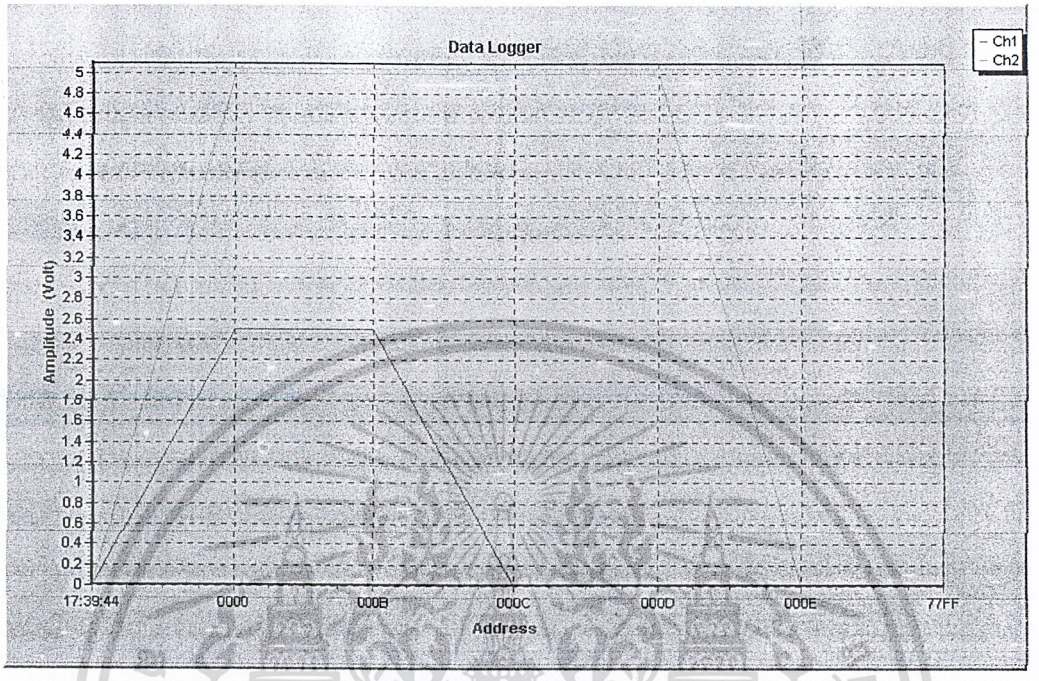
0000 : 2.51 V 0001 : 2.51 V 0002 : 2.51 V 0003 : 2.51 V 0004 : 2.51 V 0005 : 2.51 V
0006 : 2.51 V 0007 : 2.51 V 0008 : 2.51 V 0009 : 2.51 V 000A : 2.51 V 000B : 2.51 V
000C : 0 V 000D : 0 V 000E : 0 V 000F : 0 V 0010 : 0 V 0011 : 0 V
0012 : 0 V 0013 : 0 V 0014 : 0 V 0015 : 0 V

ข้อมูล RAM2 (2kByte)

Addr : Value

0000 : 5.00 V 0001 : 5.00 V 0002 : 5.00 V 0003 : 5.00 V 0004 : 5.00 V 0005 : 5.00 V
0006 : 5.00 V 0007 : 5.00 V 0008 : 5.00 V 0009 : 5.00 V 000A : 5.00 V 000B : 5.00 V
000C : 5.00 V 000D : 5.00 V 000E : 0 V 000F : 0 V 0010 : 0 V 0011 : 0 V
0012 : 0 V 0013 : 0 V 0014 : 0 V 0015 : 0 V

กราฟจากการทดลองที่ 3



ข้อมูลจาก RAM1

0000:2.51 V

000B:2.51 V

000C: 0 V

77FF: 0 V

ข้อมูลจาก RAM2

0000:5.00 V

000D:5.00 V

000E: 0 V

77FF: 0 V

สรุปและวิจารณ์

โครงงานนี้เป็นการเก็บข้อมูลจากตัวเซนต์เซอร์ที่ให้ผลออกมาเป็นค่าแรงดัน 0 ถึง 5 โวลท์ ทำให้การทำงานยังจำกัดขอบเขตของการเก็บข้อมูล สามารถเก็บข้อมูลลงในวงจรได้โดยตรงแต่ข้อมูลจะมีการจำกัดที่แอดเดรสของแรม ซึ่งความละเอียดของการเก็บข้อมูลมีความละเอียด 0.0196 โวลท์ ต่อการเปลี่ยนแปลงค่า ดิจิตอลเอ้าพุท 1 บิต

ทางด้านของภาษา VHDL และการใช้งาน FPGA ค่อนข้างมีรายละเอียดในการใช้งานทำให้ยุ่งยากบ้างในการศึกษาเบื้องต้น แต่หลังจากเริ่มเข้าใจระบบการออกแบบและโครงสร้างในส่วนต่างๆของภาษา ทำให้การออกแบบทำได้ง่ายขึ้น และหลังจากการสังเคราะห์โดยนำมาโปรแกรมลงบนชิพ FPGA แล้ว บางครั้งอาจจะมีการทำงานที่ผิดพลาดบ้าง เราจึงต้องทำการแก้ไขโปรแกรมภาษา VHDLใหม่อีก ส่วนการแปลงสัญญาณอนาลอกเป็นดิจิตอลนั้น ซึ่งใช้งานแบบ Successive Approximation บางครั้งการทำงานทำให้ค่าที่ออกทางด้านเอ้าพุทที่ได้มีความผิดพลาดเล็กน้อย แต่ก็ยังมีผลต่อประสิทธิภาพการทำงานของระบบไม่มากนัก ซึ่งการนำไปใช้งานจริงต้องพัฒนาความละเอียดให้มากกว่านี้ และตัวไอซี ADC 0808 ใช้งานง่ายสามารถนำมาควบคุมการทำงานร่วมกับ FPGA ได้สะดวก

ในส่วนของการโปรแกรม Delphi ที่ใช้สำหรับการเขียนเพื่อแสดงผล และบันทึกผลลงบนคอมพิวเตอร์ สามารถใช้งานสะดวก ซึ่งบางส่วนส่วนของโปรแกรมจะมีคอมโพเนนต์มาให้ แต่จำเป็นต้องสร้างเองหลายตัวด้วยกันซึ่งทำให้ช้าและเสียเวลาบ้าง และทำการดาวน์โหลดมาใช้งานในคอมโพเนนต์บางตัว

ประโยชน์ที่ได้รับจากโครงงานนี้คือ

1. มีความรู้ความเข้าใจเกี่ยวกับภาษา VHDL และ การใช้งาน FPGA
2. มีความรู้ความเข้าใจเกี่ยวกับ การเขียนโปรแกรม Delphi และการประยุกต์ใช้งานต่างๆ
3. ทำให้เกิดทักษะในการปฏิบัติงานจริง
4. รู้จักวางแผนงาน ในการปฏิบัติงานที่เป็นระบบมากยิ่งขึ้น
5. รู้จักแก้ไขปัญหาต่าง ได้ด้วยตัวเองหรือขอคำปรึกษาจากอาจารย์และพี่ๆได้ดี
6. ส่งเสริมความคิดที่เป็นระบบ ระเบียบมีแบบแผนมากยิ่งขึ้น
7. เป็นแนวทางเพื่อนำไปสู่การศึกษาค้นคว้าวิจัยในระดับสูงต่อไป

ภาคผนวก

HM62256B Series

32,768-word × 8-bit High Speed CMOS Static RAM

HITACHI

ADE-203-135D (Z)
Rev. 4.0
Nov. 29, 1995

Description

The Hitachi HM62256B is a CMOS static RAM organized 32-kword × 8-bit. It realizes higher performance and low power consumption by employing 0.8 µm Hi-CMOS process technology. The device, packaged in 8 × 14 mm TSOP, 8× 13.4 mm TSOP with thickness of 1.2 mm, 450-mil SOP (foot print pitch width), 600-mil plastic DIP, or 300-mil plastic DIP, is available for high density mounting. It offers low power standby power dissipation; therefore, it is suitable for battery back-up systems.

Features

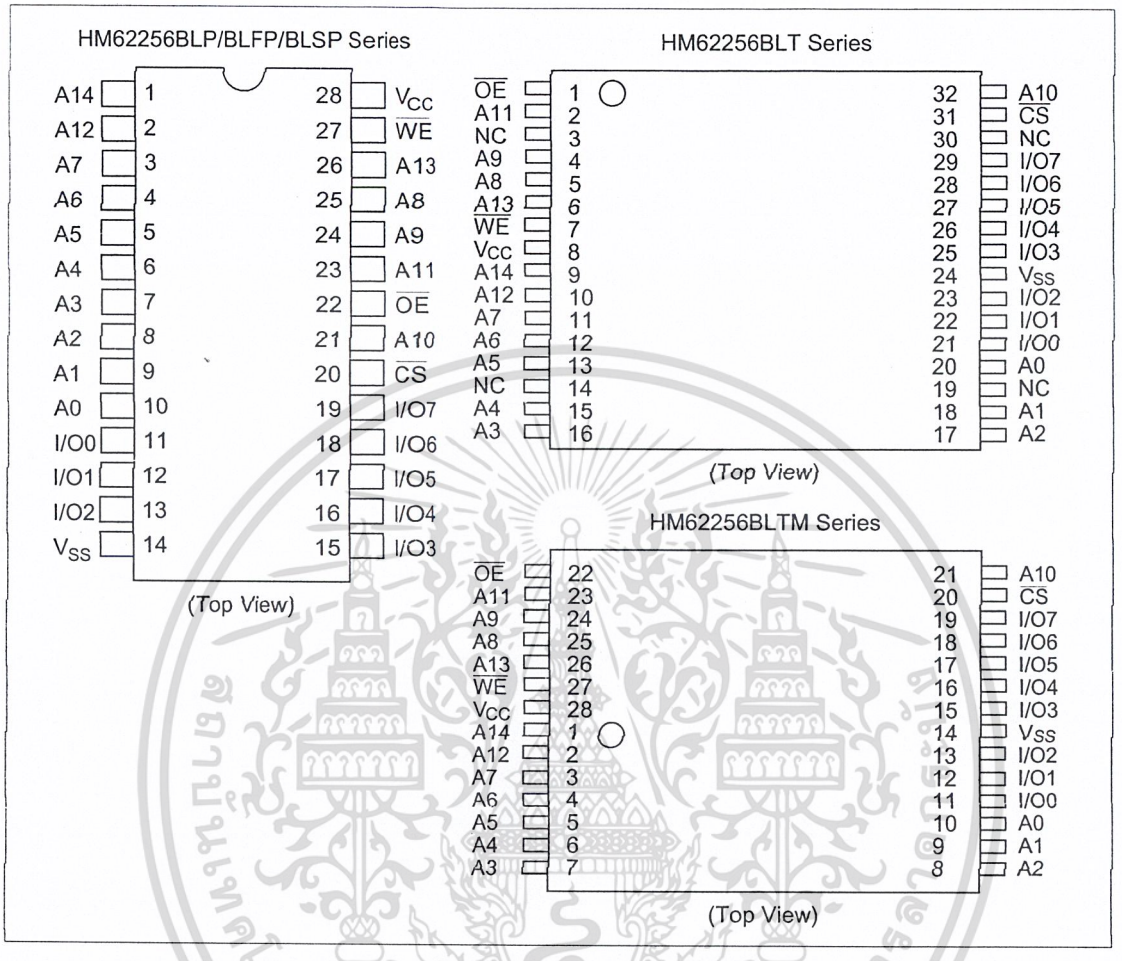
- High speed
  Fast access time: 45/55/70/85 ns (max)
- Low power
  Standby: 1.0 µW (typ)
  Operation: 25 mW (typ) (f = 1 MHz)
- Single 5 V supply
- Completely static memory
  No clock or timing strobe required
- Equal access and cycle times
- Common data input and output
  Three state output
- Directly TTL compatible
  All inputs and outputs
- Capability of battery back up operation

Ordering Information

Type No.	Access Time	Package
HM62256BLP-7	70 ns	600-mil 28-pin plastic DIP (DP-28)
HM62256BLP-7SL	70 ns	
HM62256BLSP-7	70 ns	300-mil 28-pin plastic DIP (DP-28NA)
HM62256BLSP-7SL	70 ns	
HM62256BLFP-7T	70 ns	450-mil 28-pin plastic SOP (FP-28DA)
HM62256BLFP-4SLT[*1]	45 ns	
HM62256BLFP-5SLT	*55 ns*	
HM62256BLFP-7SLT	70 ns	
HM62256BLFP-7ULT	70 ns	
HM62256BLT-8	85 ns	8 mm × 14 mm 32-pin TSOP (TFP-32DA)
HM62256BLT-7SL	70 ns	
HM62256BLTM-8	85 ns	8 mm × 13.4 mm 28-pin TSOP (TFP-28DA)
HM62256BLTM-4SL[*1]	45 ns	
HM62256BLTM-5SL	55 ns	
HM62256BLTM-7SL	70 ns	
HM62256BLTM-7UL	70 ns	

Note: 1. Under development

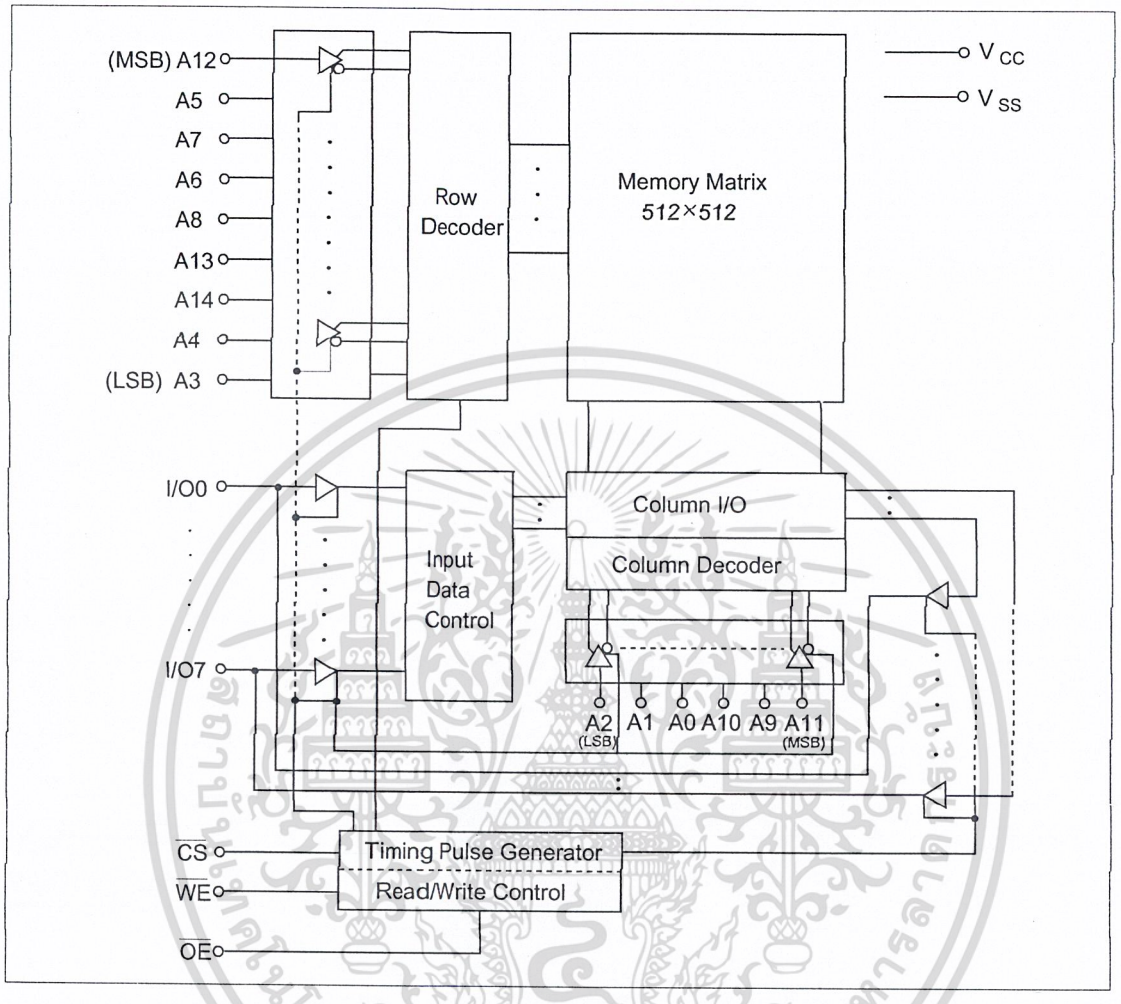
Pin Arrangement



Pin Description

Symbol	Function
A0 – A14	Address
I/O0 – I/O7	Input/output
$\overline{CS}$	Chip select
$\overline{WE}$	Write enable
$\overline{OE}$	Output enable
NC	No connection
V_{cc}	Power supply
V_{ss}	Ground

Block Diagram



Function Table

$\overline{WE}$	$\overline{CS}$	$\overline{OE}$	Mode	V_{CC} Current	I/O Pin	Ref. Cycle
X	H	X	Not selected	I_{SB}, I_{SB1}	High-Z	—
H	L	H	Output disable	I_{CC}	High-Z	—
H	L	L	Read	I_{CC}	Dout	Read cycle (1)–(3)
L	L	H	Write	I_{CC}	Din	Write cycle (1)
L	L	L	Write	I_{CC}	Din	Write cycle (2)

Note: X: H or L

Absolute Maximum Ratings

Parameter	Symbol	Value	Unit
Power supply voltage*1	V_{CC}	−0.5 to +7.0	V
Terminal voltage*1	V_T	−0.5*2 to V_{CC} + 0.3*3	V
Power dissipation	P_T	1.0	W
Operating temperature	Topr	0 to + 70	°C
Storage temperature	Tstg	−55 to +125	°C
Storage temperature under bias	Tbias	−10 to +85	°C

Notes: 1. Relative to V_{SS}
2. V_T min: −3.0 V for pulse half-width ≤ 50 ns
3. Maximum voltage is 7.0 V

Recommended DC Operating Conditions (Ta = 0 to +70°C)

Parameter	Symbol	Min	Typ	Max	Unit
Supply voltage	V_{CC}	4.5	5.0	5.5	V
	V_{SS}	0	0	0	V
Input high (logic 1) voltage	V_{IH}	2.2	—	V_{CC}+0.3	V
Input low (logic 0) voltage	V_{IL}	−0.5 *1	—	0.8	V

Note: 1. V_{IL} min: −3.0 V for pulse half-width ≤ 50 ns

DC Characteristics (Ta = 0 to +70°C, V_{CC} = 5 V ±10%, V_{SS} = 0 V)

Parameter		Symbol	Min	Typ*1	Max	Unit	Test Conditions
Input leakage current		$\|I_{LI}\|$	—	—	1	µA	Vin = V_{SS} to V_{CC}
Output leakage current		$\|I_{LO}\|$	—	—	1	µA	$\overline{CS}$ = V_{IH} or $\overline{OE}$ = V_{IH} or $\overline{WE}$ = V_{IL}, $V_{SS} \le V_{I/O} \le V_{CC}$
Operating power supply current		I_{CC}	—	6	15	mA	$\overline{CS}$ = V_{IL}, others = V_{IH}/V_{IL} $I_{I/O}$ = 0 mA
Average operating power supply current	HM62256B-4	I_{CC1}	—	—	70	mA	min cycle, duty = 100 %, $I_{I/O}$ = 0 mA $\overline{CS}$ = V_{IL}, others = V_{IH}/V_{IL}
	HM62256B-5	I_{CC1}	—	—	60		
	HM62256B-7	I_{CC1}	—	33	60		
	HM62256B-8	*I_{CC1}*	—	*29*	*50*		
		I_{CC2}	—	5	15	mA	Cycle time = 1 µs, $I_{I/O}$ = 0 mA $\overline{CS}$ = V_{IL}, V_{IH} = V_{CC}, V_{IL} = 0
Standby power supply current		I_{SB}	—	0.3	2	mA	$\overline{CS}$ = V_{IH}
		I_{SB1}	—	0.2	100	µA	Vin ≥ 0 V, $\overline{CS}$ ≥ V_{CC} – 0.2 V,
			—	0.2*2	50*2		
				0.2*3	10*3		
Output low voltage		V_{OL}	—	—	0.4	V	I_{OL} = 2.1 mA
Output high voltage		V_{OH}	2.4	—	—	V	I_{OH} = –1.0 mA

Notes: 1. Typical values are at V_{CC} = 5.0 V, Ta = +25°C and not guaranteed.
2. This characteristics is guaranteed only for L-SL version.
3. This characteristics is guaranteed only for L-UL version.

Capacitance (Ta = 25°C, f = 1.0 MHz)*1

Parameter	Symbol	Min	Typ	Max	Unit	Test Conditions
Input capacitance*1	Cin	—	—	8	pF	Vin = 0 V
Input/output capacitance*1	$C_{I/O}$	—	—	10	pF	$V_{I/O}$ = 0 V

Note: 1. This parameter is sampled and not 100% tested.

AC Characteristics (Ta = 0 to +70°C, V_{CC} = 5 V ± 10%, unless otherwise noted.)

Test Conditions

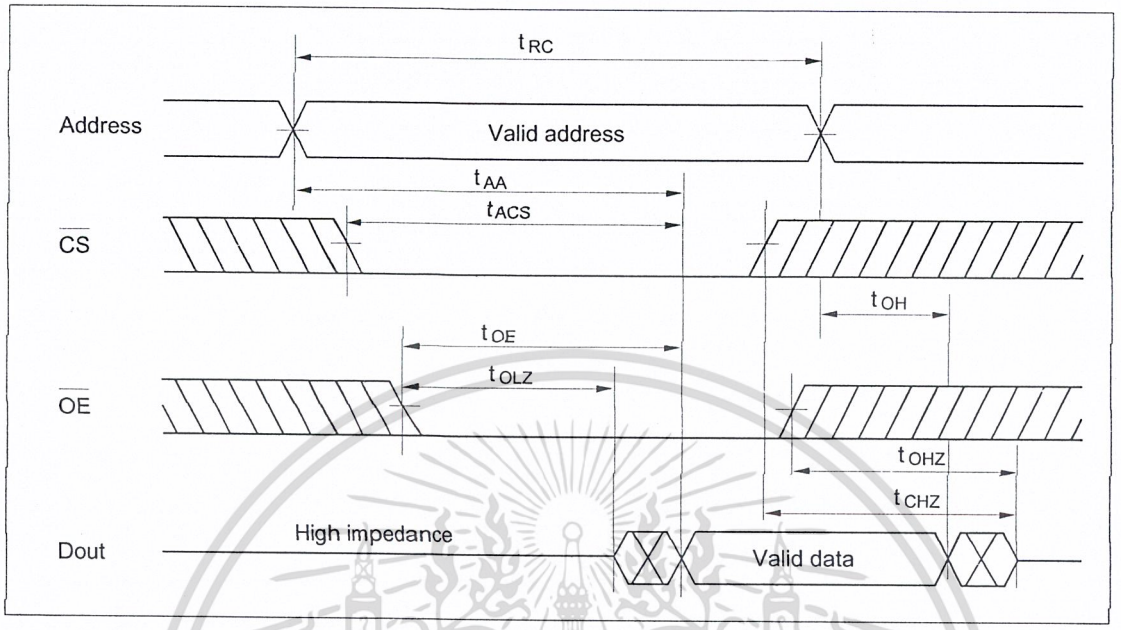
- Input pulse levels: 0.8 V to 2.4 V
- Input rise and fall times: 5 ns
- Input and output timing reference level: 1.5 V
- Output load: HM62256B-4: 1 TTL Gate + C_L (30 pF)(Including scope & jig)
  HM62256B-5: 1 TTL Gate + C_L (50 pF)(Including scope & jig)
  HM62256B-7/8: 1 TTL Gate + C_L (100 pF)(Including scope & jig)

Read Cycle

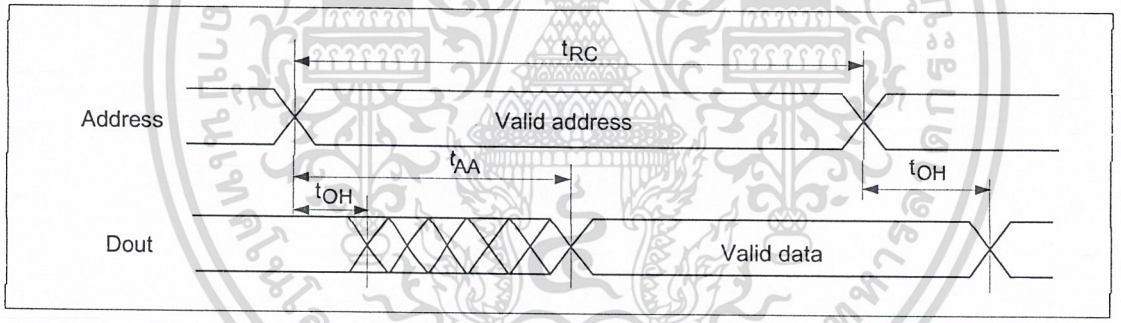
		HM62256B									
		-4		-5		-7		-8			
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Min	Max	Unit	Notes
Read cycle time	t_{RC}	45	—	55	—	70	—	85	—	ns	
Address access time	t_{AA}	—	45	—	55	—	70	—	85	ns	
Chip select access time	t_{ACS}	—	45	—	55	—	70	—	85	ns	
Output enable to output valid	t_{OE}		30	—	35	—	40	—	45	ns	
Chip selection to output in low-Z	t_{CLZ}	5	—	5	—	10	—	10	—	ns	2
Output enable to output in low-Z	t_{OLZ}	5	—	5	—	5	—	5	—	ns	2
Chip deselection in to output in high-Z	t_{CHZ}	0	20	0	20	0	25	0	30	ns	1, 2
Output disable to output in high-Z	t_{OHZ}	0	20	0	20	0	25	0	30	ns	1, 2
Output hold from address change	t_{OH}	5	—	5	—	5	—	10	—	ns	

Notes: 1. t_{CHZ} and t_{OHZ} defined as the time at which the outputs achieve the open circuit conditions and are not referred to output voltage levels.
2. This parameter is sampled and not 100% tested.

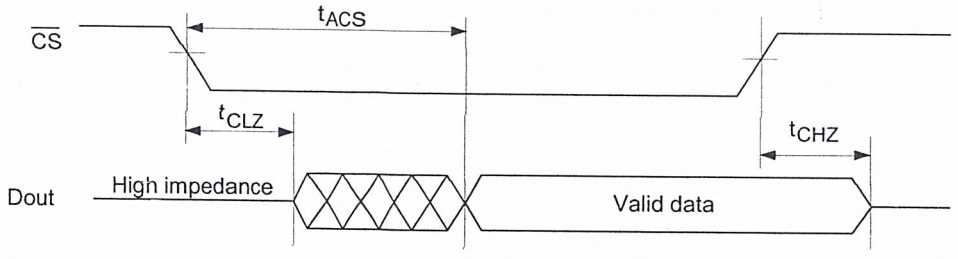
Read Timing Waveform (1) ($\overline{WE}$=V_{IH})



Read Timing Waveform (2) ($\overline{WE}$=V_{IH}, $\overline{CS}$=V_{IL}, $\overline{OE}$=V_{IL})



Read Timing Waveform (3) ($\overline{WE}$=V_{IH}, $\overline{OE}$=V_{IL})*1



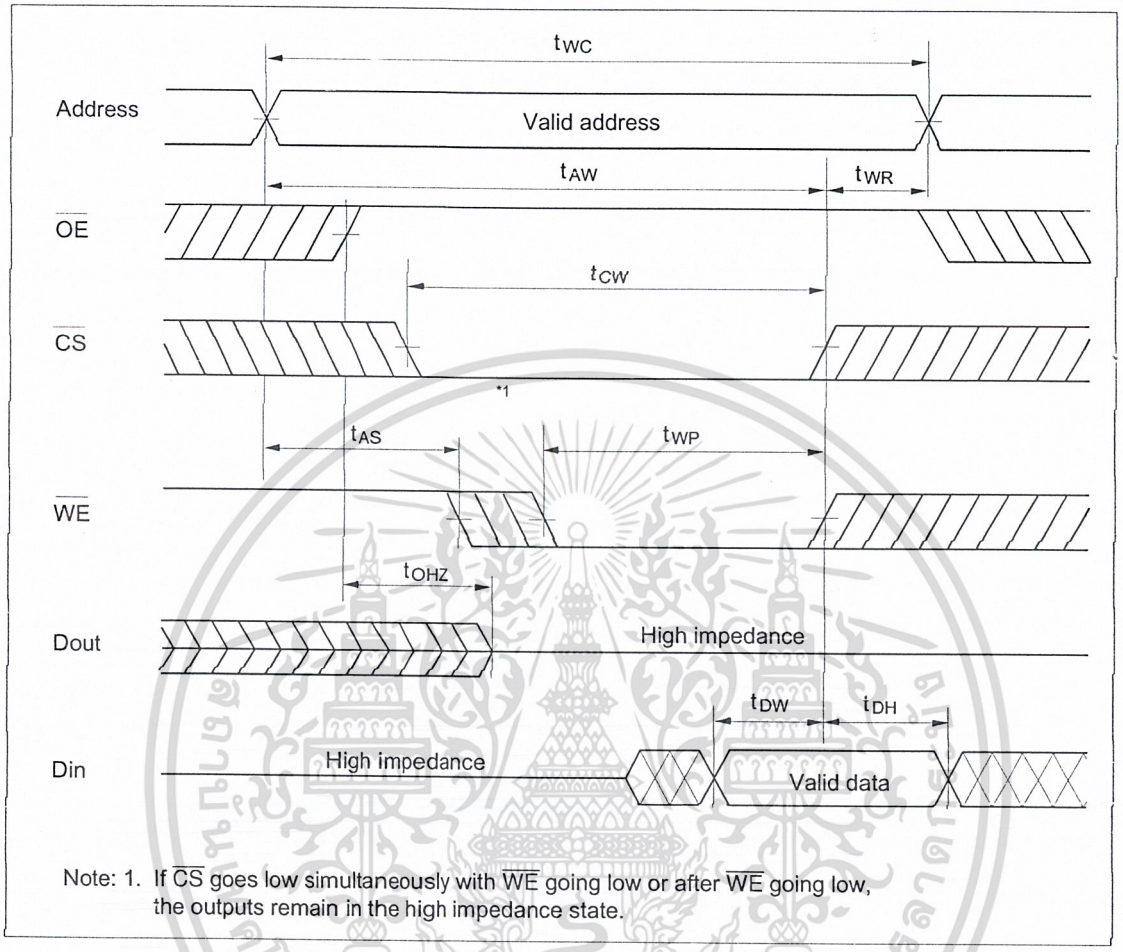
Note: 1. Address must be valid prior to or simultaneously with $\overline{CS}$ going low.

Write Cycle

		HM62256B									
		-4		-5		-7		-8			
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Min	Max	Unit	Notes
Write cycle time	t_{WC}	45	—	55	—	70	—	85	—	ns	
Chip selection to end of write	t_{CW}	35	—	40	—	60	—	75	—	ns	4
Address setup time	t_{AS}	0	—	0	—	0	—	0	—	ns	5
Address valid to end of write	t_{AW}	35	—	40	—	60	—	75	—	ns	
Write pulse width	t_{WP}	30	—	35	—	50	—	55	—	ns	3, 8
Write recovery time	t_{WR}	0	—	0	—	0	—	0	—	ns	6
$\overline{WE}$ to output in high-Z	t_{WHZ}	0	20	0	20	0	25	0	40	ns	1, 2, 7
Data to write time overlap	t_{DW}	20	—	25	—	30	—	35	—	ns	
Data hold from write time	t_{DH}	0	—	0	—	0	—	0	—	ns	
Output active from end of write	t_{OW}	5	—	5	—	5	—	5	—	ns	2
Output disable to output in high-Z	t_{OHZ}	0	20	0	20	0	25	0	40	ns	1, 2, 7

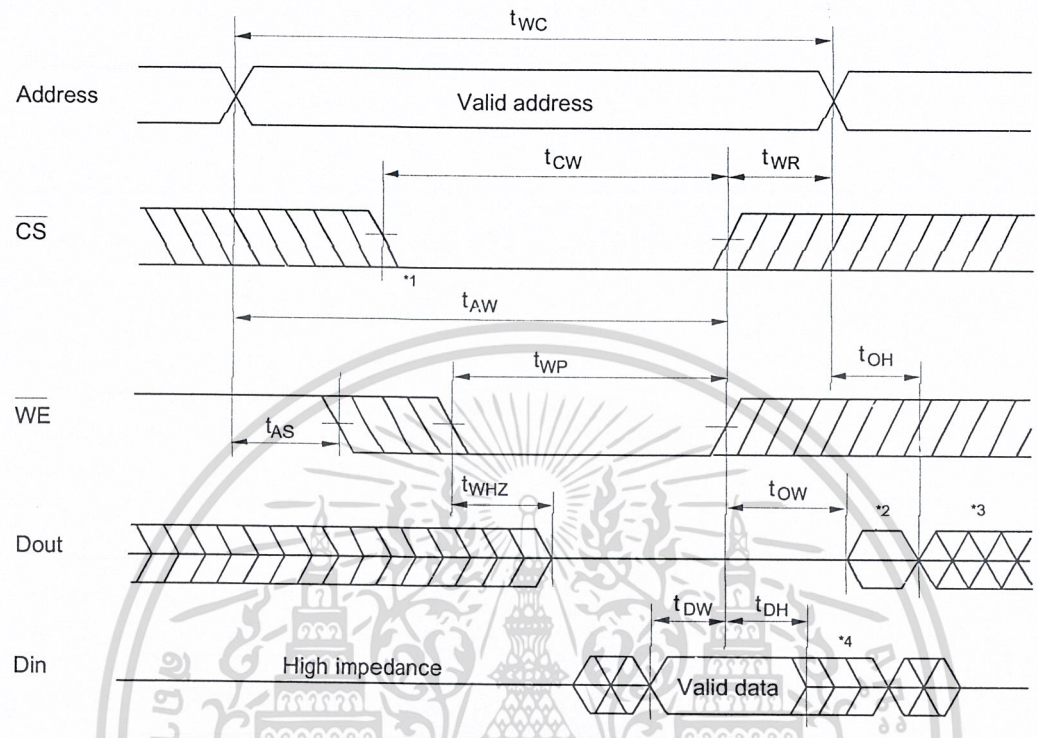
Notes:
1. t_{OHZ} and t_{WHZ} are defined as the time at which the outputs achieve the open circuit conditions and are not referred to output voltage levels.
2. This parameter is sampled and not 100% tested.
3. A write occurs during the overlap (t_{WP}) of a low $\overline{CS}$ and a low $\overline{WE}$. A write begins at the later transition of $\overline{CS}$ going low or $\overline{WE}$ going low. A write ends at the earlier transition of $\overline{CS}$ going high or $\overline{WE}$ going high. t_{WP} is measured from the beginning of write to the end of write.
4. t_{CW} is measured from $\overline{CS}$ going low to the end of write.
5. t_{AS} is measured from the address valid to the beginning of write.
6. t_{WR} is measured from the earlier of $\overline{WE}$ or $\overline{CS}$ going high to the end of write cycle.
7. Durng this period, I/O pins are in the output state so that the input signals of the opposite phase to the outputs must not be applied.
8. In the write cycle with $\overline{OE}$ low fixed, t_{WP} must satisfy the following equation to avoid a problem of data bus contention, $t_{WP} \geq t_{WHZ}$ max + t_{DW} min.

Write Timing Waveform (1) ($\overline{OE}$ Clock)



Note: 1. If $\overline{CS}$ goes low simultaneously with $\overline{WE}$ going low or after $\overline{WE}$ going low, the outputs remain in the high impedance state.

Write Timing Waveform (2) ($\overline{OE}$ Low Fixed) ($\overline{OE} = V_{IL}$)



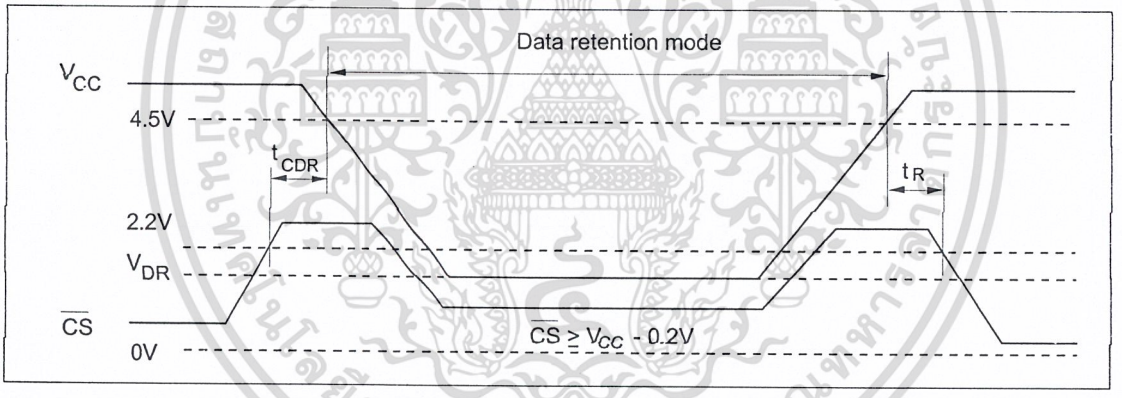
Notes: 1. If $\overline{CS}$ goes low simultaneously with $\overline{WE}$ going low or after $\overline{WE}$ going low, the outputs remain in the high impedance state.
2. Dout is the same phase of the write data of this write cycle.
3. Dout is the read data of next address.
4. If $\overline{CS}$ is low during this period, I/O pins are in the output state. Therefore, the input signals of the opposite phase to the output must not be applied to them.

Low V_{CC} Data Retention Characteristics (Ta = 0 to +70°C)

Parameter	Symbol	Min	Typ*1	Max	Unit	Test Conditions*6
V_{CC} for data retention	V_{DR}	2.0	—	5.5	V	$\overline{CS} \geq V_{CC} - 0.2$ V, Vin ≥ 0 V
Data retention current	I_{CCDR}	—	0.05	30*2	µA	V_{CC} = 3.0 V, Vin ≥ 0 V
		—	0.05	10*3		$\overline{CS} \geq V_{CC} - 0.2$ V,
		—	0.05	3*4		
Chip deselect to data retention time	t_{CDR}	0	—	—	ns	See retention waveform
Operation recovery time	t_R	t_{RC}*5	—	—	ns	

Notes: 1. Typical values are at V_{CC} = 3.0 V, Ta = 25°C and not guaranteed.
2. 10 µA max at Ta = 0 to + 40°C.
3. This characteristics guaranteed for only L-SL version. 3 µA max at Ta = 0 to +40°C.
4. This characteristics guaranteed for only L-UL version. 0.6 µA max at Ta = 0 to +40°C.
5. t_{RC} = read cycle time.
6. $\overline{CS}$ controls address buffer, $\overline{WE}$ buffer, $\overline{OE}$ buffer, and Din buffer. If $\overline{CS}$ controls data retention mode, other input levels (address, $\overline{WE}$, $\overline{OE}$, I/O) can be in the high impedance state.

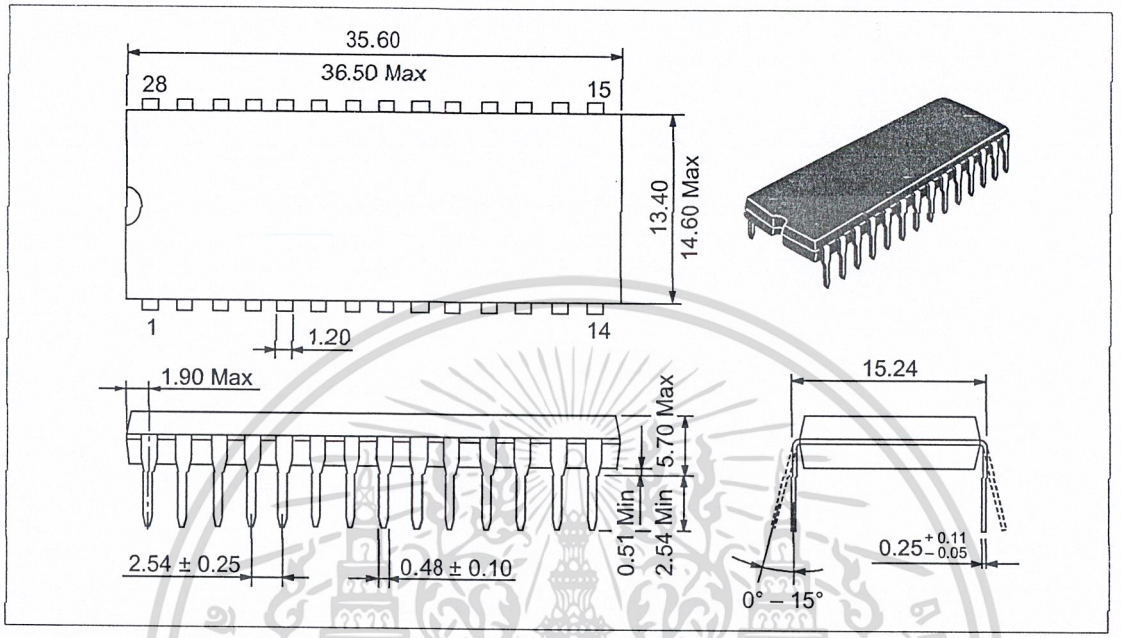
Low V_{CC} Data Retention Timing Waveform



Package Dimensions

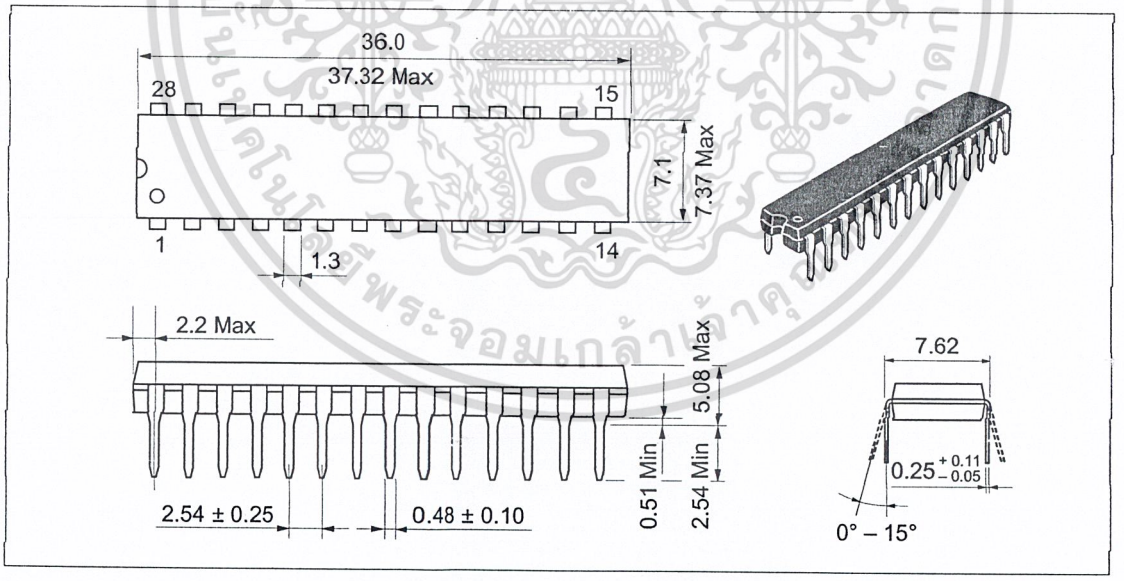
HM62256BLP Series (DP-28)

Unit: mm



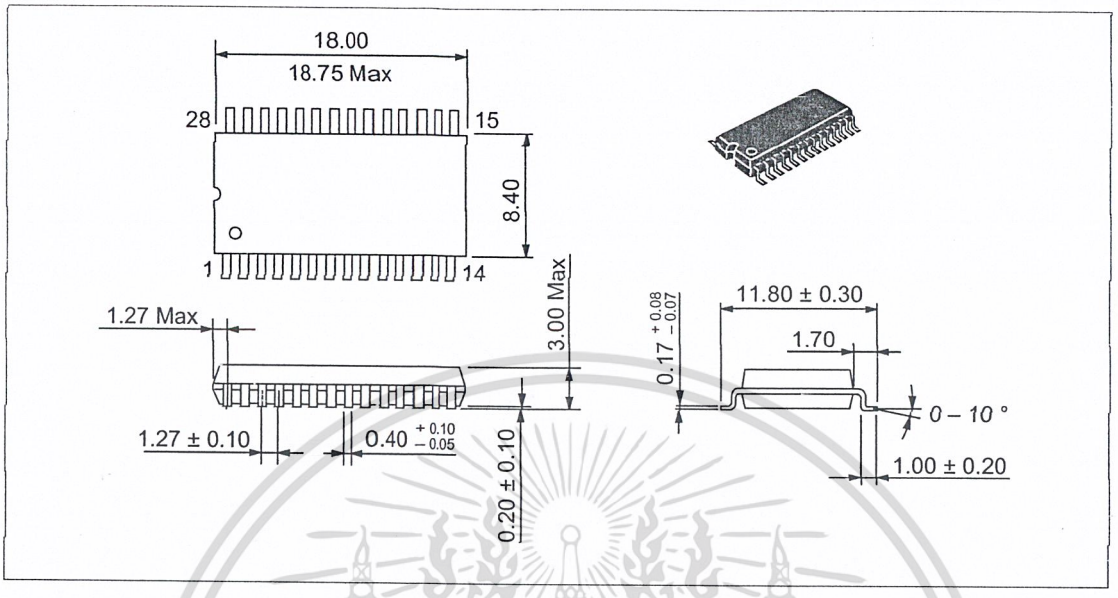
HM62256BLSP Series (DP-28NA)

Unit: mm



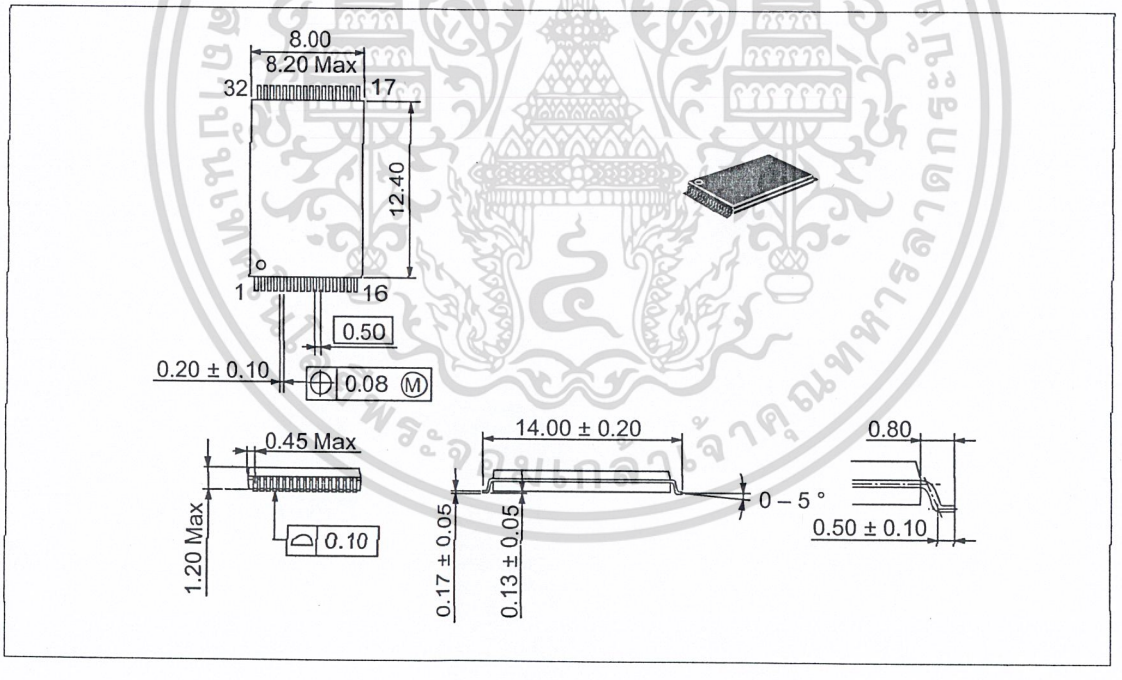
HM62256BLFP Series (FP-28DA)

Unit: mm



HM62256BLT Series (TFP-32DA)

Unit: mm



National Semiconductor

October 1999

ADC0808/ADC0809
8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE® outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 data sheet. (See AN-247 for more information.)

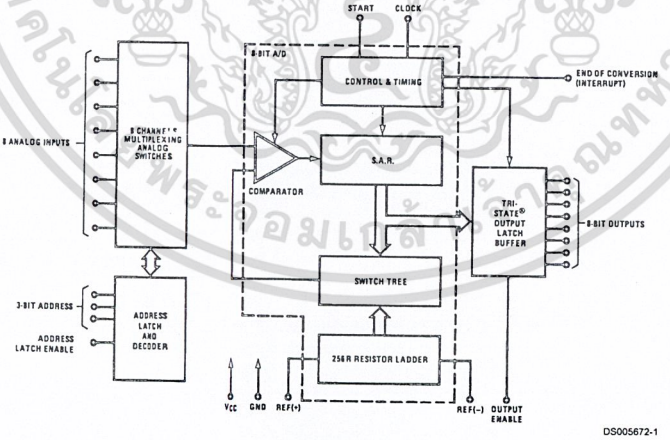
Features

- Easy interface to all microprocessors
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- No zero or full-scale adjust required
- 8-channel multiplexer with address logic
- 0V to 5V input range with single 5V power supply
- Outputs meet TTL voltage level specifications
- Standard hermetic or molded 28-pin DIP package
- 28-pin molded chip carrier package
- ADC0808 equivalent to MM74C949
- ADC0809 equivalent to MM74C949-1

Key Specifications

■ Resolution	8 Bits
■ Total Unadjusted Error	±½ LSB and ±1 LSB
■ Single Supply	5 V_{DC}
■ Low Power	15 mW
■ Conversion Time	100 µs

Block Diagram



See Ordering Information

TRI-STATE® is a registered trademark of National Semiconductor Corp.

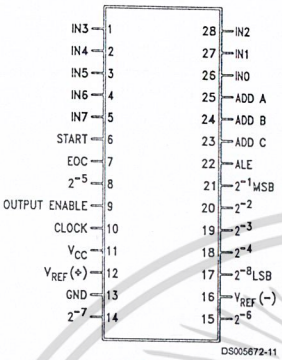
 DS005672

www.national.com

ADC0808/ADC0809 8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer

Connection Diagrams

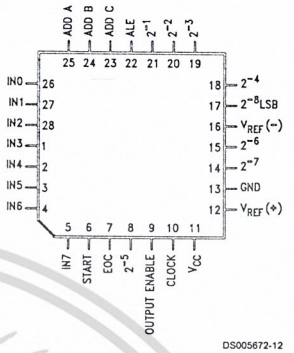
Dual-In-Line Package



DS005672-11

Order Number ADC0808CCN or ADC0809CCN
See NS Package J28A or N28A

Molded Chip Carrier Package



DS005672-12

Order Number ADC0808CCV or ADC0809CCV
See NS Package V28A

Ordering Information

TEMPERATURE RANGE		−40°C to +85°C			−55°C to +125°C
Error	±½ LSB Unadjusted	ADC0808CCN	ADC0808CCV	ADC0808CCJ	ADC0808CJ
	±1 LSB Unadjusted	ADC0809CCN	ADC0809CCV		
Package Outline		N28A Molded DIP	V28A Molded Chip Carrier	J28A Ceramic DIP	J28A Ceramic DIP

Absolute Maximum Ratings (Notes 2, 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage (V_{CC}) (Note 3)	6.5V
Voltage at Any Pin Except Control Inputs	−0.3V to (V_{CC}+0.3V)
Voltage at Control Inputs (START, OE, CLOCK, ALE, ADD A, ADD B, ADD C)	−0.3V to +15V
Storage Temperature Range	−65°C to +150°C
Package Dissipation at T_A=25°C	875 mW
Lead Temp. (Soldering, 10 seconds)	
Dual-In-Line Package (plastic)	260°C
Dual-In-Line Package (ceramic)	300°C
Molded Chip Carrier Package	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C
ESD Susceptibility (Note 8)	400V

Operating Conditions (Notes 1, 2)

Temperature Range (Note 1)	$T_{MIN} \le T_A \le T_{MAX}$
ADC0808CCN,ADC0809CCN	−40°C≤T_A≤+85°C
ADC0808CCV, ADC0809CCV	−40°C ≤ T_A ≤ +85°C
Range of V_{CC} (Note 1)	4.5 V_{DC} to 6.0 V_{DC}

Electrical Characteristics

Converter Specifications: V_{CC}=5 V_{DC}=V_{REF+}, $V_{REF(-)}$=GND, $T_{MIN} \le T_A \le T_{MAX}$ and f_{CLK}=640 kHz unless otherwise stated.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
	ADC0808					
	Total Unadjusted Error	25°C			±½	LSB
	(Note 5)	T_{MIN} to T_{MAX}			±¾	LSB
	ADC0809					
	Total Unadjusted Error	0°C to 70°C			±1	LSB
	(Note 5)	T_{MIN} to T_{MAX}			±1¼	LSB
	Input Resistance	From Ref(+) to Ref(−)	1.0	2.5		kΩ
	Analog Input Voltage Range	(Note 4) V(+) or V(−)	GND−0.10		V_{CC}+0.10	V_{DC}
$V_{REF(+)}$	Voltage, Top of Ladder	Measured at Ref(+)		V_{CC}	V_{CC}+0.1	V
$\frac{V_{REF(+)}+V_{REF(-)}}{2}$	Voltage, Center of Ladder		V_{CC}/2-0.1	V_{CC}/2	V_{CC}/2+0.1	V
$V_{REF(-)}$	Voltage, Bottom of Ladder	Measured at Ref(−)	−0.1	0		V
I_{IN}	Comparator Input Current	f_c=640 kHz, (Note 6)	−2	±0.5	2	μA

Electrical Characteristics

Digital Levels and DC Specifications: ADC0808CCN, ADC0808CCV, ADC0809CCN and ADC0809CCV, 4.75≤V_{CC}≤5.25V, −40°C≤T_A≤+85°C unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
ANALOG MULTIPLEXER						
$I_{OFF(+)}$	OFF Channel Leakage Current	V_{CC}=5V, V_{IN}=5V,				
		T_A=25°C		10	200	nA
		T_{MIN} to T_{MAX}			1.0	μA
$I_{OFF(-)}$	OFF Channel Leakage Current	V_{CC}=5V, V_{IN}=0,				
		T_A=25°C	−200	−10		nA
		T_{MIN} to T_{MAX}	−1.0			μA
CONTROL INPUTS						
$V_{IN(1)}$	Logical "1" Input Voltage		V_{CC}−1.5			V
$V_{IN(0)}$	Logical "0" Input Voltage				1.5	V
$I_{IN(1)}$	Logical "1" Input Current (The Control Inputs)	V_{IN}=15V			1.0	μA
$I_{IN(0)}$	Logical "0" Input Current (The Control Inputs)	V_{IN}=0	−1.0			μA
I_{CC}	Supply Current	f_{CLK}=640 kHz		0.3	3.0	mA

Electrical Characteristics (Continued)

Digital Levels and DC Specifications: ADC0808CCN, ADC0808CCV, ADC0809CCN and ADC0809CCV, $4.75 \le V_{CC} \le 5.25V$, $-40^{\circ}C \le T_A \le +85^{\circ}C$ unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
DATA OUTPUTS AND EOC (INTERRUPT)						
$V_{OUT(1)}$	Logical "1" Output Voltage	V_{CC} = 4.75V I_{OUT} = −360µA I_{OUT} = −10µA		**2.4** **4.5**		V(min) V(min)
$V_{OUT(0)}$	Logical "0" Output Voltage	I_O=1.6 mA			0.45	V
$V_{OUT(0)}$	Logical "0" Output Voltage EOC	I_O=1.2 mA			0.45	V
I_{OUT}	TRI-STATE Output Current	V_O=5V V_O=0	 −3		3	µA µA

Electrical Characteristics

Timing Specifications $V_{CC}=V_{REF(+)}=5V$, $V_{REF(-)}=GND$, $t_r=t_f=20$ ns and $T_A=25^{\circ}C$ unless otherwise noted.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{WS}	Minimum Start Pulse Width	(*Figure 5*)		100	200	ns
t_{WALE}	Minimum ALE Pulse Width	(*Figure 5*)		100	200	ns
t_s	Minimum Address Set-Up Time	(*Figure 5*)		25	50	ns
t_H	Minimum Address Hold Time	(*Figure 5*)		25	50	ns
t_D	Analog MUX Delay Time From ALE	R_S=0Ω (*Figure 5*)		1	2.5	µs
t_{H1}, t_{H0}	OE Control to Q Logic State	C_L=50 pF, R_L=10k (*Figure 8*)		125	250	ns
t_{1H}, t_{0H}	OE Control to Hi-Z	C_L=10 pF, R_L=10k (*Figure 8*)		125	250	ns
t_c	Conversion Time	f_c=640 kHz, (*Figure 5*) (Note 7)	90	100	116	µs
f_c	Clock Frequency		10	640	1280	kHz
t_{EOC}	EOC Delay Time	(*Figure 5*)	0		8+2 µS	Clock Periods
C_{IN}	Input Capacitance	At Control Inputs		10	15	pF
C_{OUT}	TRI-STATE Output Capacitance	At TRI-STATE Outputs		10	15	pF

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 2: All voltages are measured with respect to GND, unless otherwise specified.

Note 3: A zener diode exists, internally, from V_{CC} to GND and has a typical breakdown voltage of 7 V_{DC}.

Note 4: Two on-chip diodes are tied to each analog input which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the V_{CC}n supply. The spec allows 100 mV forward bias of either diode. This means that as long as the analog V_{IN} does not exceed the supply voltage by more than 100 mV, the output code will be correct. To achieve an absolute $0V_{DC}$ to $5V_{DC}$ input voltage range will therefore require a minimum supply voltage of 4.900 V_{DC} over temperature variations, initial tolerance and loading.

Note 5: Total unadjusted error includes offset, full-scale, linearity, and multiplexer errors. See *Figure 3*. None of these A/Ds requires a zero or full-scale adjust. However, if an all zero code is desired for an analog input other than 0.0V, or if a narrow full-scale span exists (for example: 0.5V to 4.5V full-scale) the reference voltages can be adjusted to achieve this. See *Figure 13*.

Note 6: Comparator input current is a bias current into or out of the chopper stabilized comparator. The bias current varies directly with clock frequency and has little temperature dependence (*Figure 6*). See paragraph 4.0.

Note 7: The outputs of the data register are updated one clock cycle before the rising edge of EOC.

Note 8: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Functional Description

Multiplexer. The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. *Table 1* shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

TABLE 1.

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

The Converter

The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach (*Figure 1*) was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in *Figure 1* are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached +½ LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. *Figure 2* shows a typical example of a 3-bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bits using the 256R network.

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed through a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors.

Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

Functional Description (Continued)

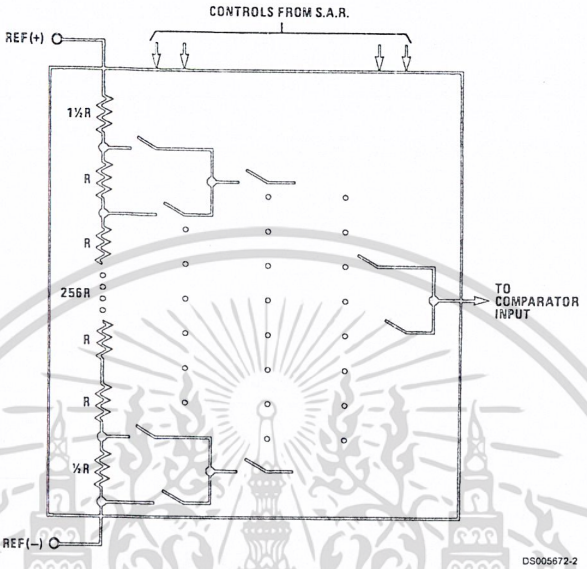


FIGURE 1. Resistor Ladder and Switch Tree

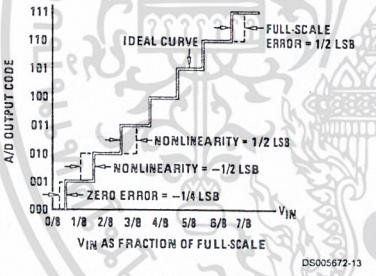


FIGURE 2. 3-Bit A/D Transfer Curve

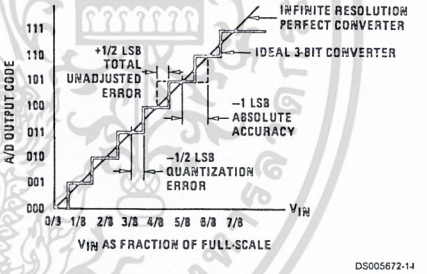


FIGURE 3. 3-Bit A/D Absolute Accuracy Curve

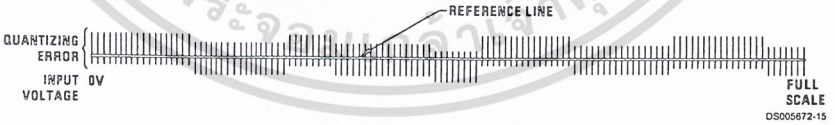


FIGURE 4. Typical Error Curve

Timing Diagram

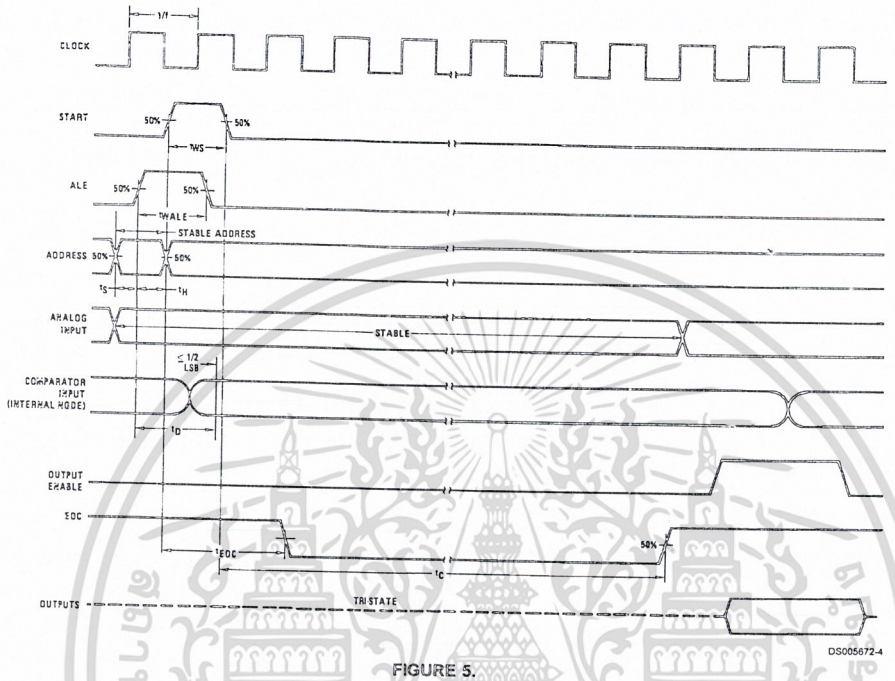


FIGURE 5.

Typical Performance Characteristics

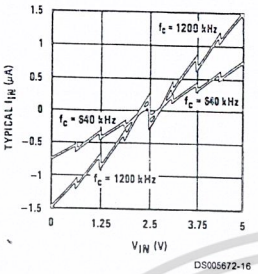


FIGURE 6. Comparator I_{IN} vs V_{IN} (V_{CC}=V_{REF}=5V)

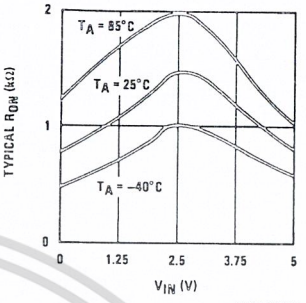


FIGURE 7. Multiplexer R_{ON} vs V_{IN} (V_{CC}=V_{REF}=5V)

TRI-STATE Test Circuits and Timing Diagrams

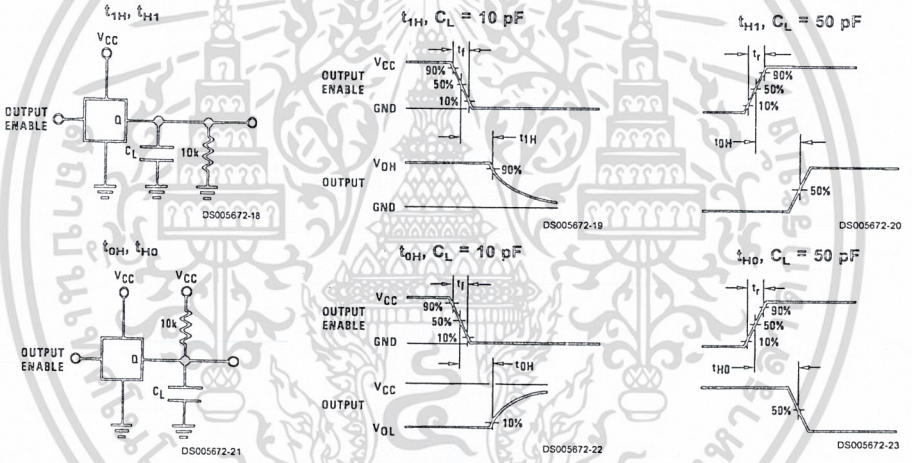


FIGURE 8.

Applications Information

OPERATION

1.0 RATIOMETRIC CONVERSION

The ADC0808, ADC0809 is designed as a complete Data Acquisition System (DAS) for ratiometric conversion systems. In ratiometric systems, the physical variable being measured is expressed as a percentage of full-scale which is not necessarily related to an absolute standard. The voltage input to the ADC0808 is expressed by the equation

$$\frac{V_{IN}}{V_{fs}-V_Z}=\frac{D_X}{D_{MAX}-D_{MIN}} \qquad (1)$$

V_{IN}=Input voltage into the ADC0808

V_{fs}=Full-scale voltage

V_Z=Zero voltage

D_X=Data point being measured

D_{MAX}=Maximum data limit

D_{MIN}=Minimum data limit

A good example of a ratiometric transducer is a potentiometer used as a position sensor. The position of the wiper is directly proportional to the output voltage which is a ratio of the full-scale voltage across it. Since the data is represented as a proportion of full-scale, reference requirements are greatly reduced, eliminating a large source of error and cost for many applications. A major advantage of the ADC0808, ADC0809 is that the input voltage range is equal to the supply range so the transducers can be connected directly across the supply and their outputs connected directly into the multiplexer inputs, (*Figure 9*).

Ratiometric transducers such as potentiometers, strain gauges, thermistor bridges, pressure transducers, etc., are suitable for measuring proportional relationships; however, many types of measurements must be referred to an absolute standard such as voltage or current. This means a sys-

Applications Information (Continued)

tem reference must be used which relates the full-scale voltage to the standard volt. For example, if $V_{CC}=V_{REF}=5.12V$, then the full-scale range is divided into 256 standard steps. The smallest standard step is 1 LSB which is then 20 mV.

2.0 RESISTOR LADDER LIMITATIONS

The voltages from the resistor ladder are compared to the selected into 8 times in a conversion. These voltages are coupled to the comparator via an analog switch tree which is referenced to the supply. The voltages at the top, center and bottom of the ladder must be controlled to maintain proper operation.

The top of the ladder, Ref(+), should not be more positive than the supply, and the bottom of the ladder, Ref(−), should not be more negative than ground. The center of the ladder voltage must also be near the center of the supply because the analog switch tree changes from N-channel switches to P-channel switches. These limitations are automatically satisfied in ratiometric systems and can be easily met in ground referenced systems.

Figure 10 shows a ground referenced system with a separate supply and reference. In this system, the supply must be trimmed to match the reference voltage. For instance, if a 5.12V is used, the supply should be adjusted to the same voltage within 0.1V.

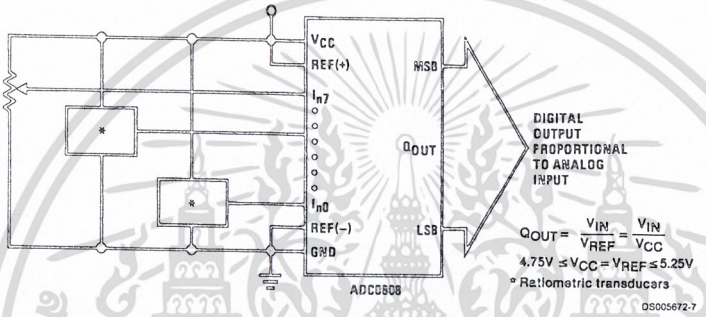
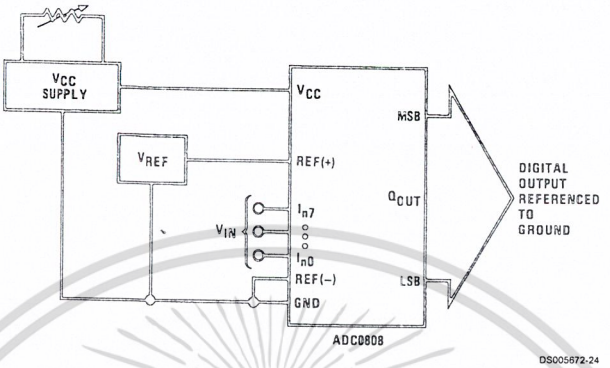


FIGURE 9. Ratiometric Conversion System

The ADC0808 needs less than a milliamp of supply current so developing the supply from the reference is readily accomplished. In *Figure 11* a ground referenced system is shown which generates the supply from the reference. The buffer shown can be an op amp of sufficient drive to supply the milliamp of supply current and the desired bus drive, or if a capacitive bus is driven by the outputs a large capacitor will supply the transient supply current as seen in *Figure 12*. The LM301 is overcompensated to insure stability when loaded by the 10 µF output capacitor.

The top and bottom ladder voltages cannot exceed V_{CC} and ground, respectively, but they can be symmetrically less than V_{CC} and greater than ground. The center of the ladder voltage should always be near the center of the supply. The sensitivity of the converter can be increased, (i.e., size of the LSB steps decreased) by using a symmetrical reference system. In *Figure 13*, a 2.5V reference is symmetrically centered about $V_{CC}/2$ since the same current flows in identical resistors. This system with a 2.5V reference allows the LSB bit to be half the size of a 5V reference system.

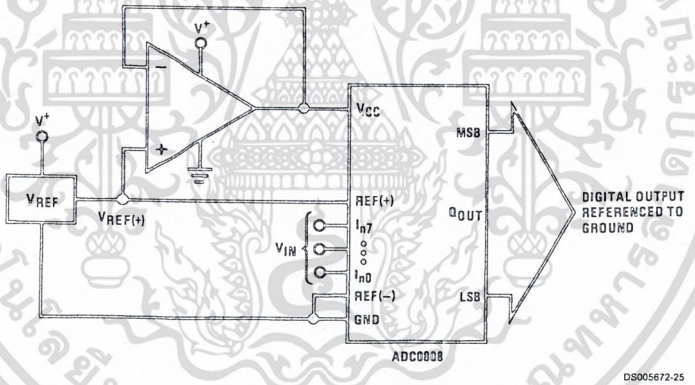
Applications Information (Continued)



$$Q_{OUT} = \frac{V_{IN}}{V_{REF}}$$

$4.75V \leq V_{CC} = V_{REF} \leq 5.25V$

FIGURE 10. Ground Referenced Conversion System Using Trimmed Supply



$$Q_{OUT} = \frac{V_{IN}}{V_{REF}}$$

$4.75V \leq V_{CC} = V_{REF} \leq 5.25V$

FIGURE 11. Ground Referenced Conversion System with Reference Generating V_{CC} Supply

Applications Information (Continued)

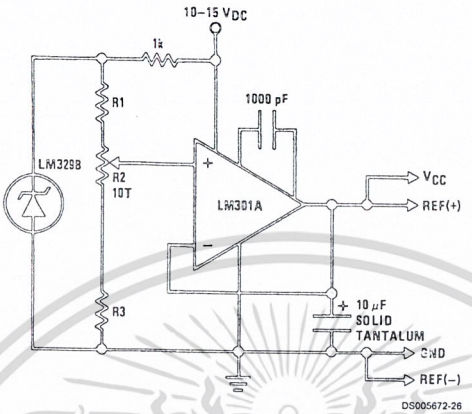
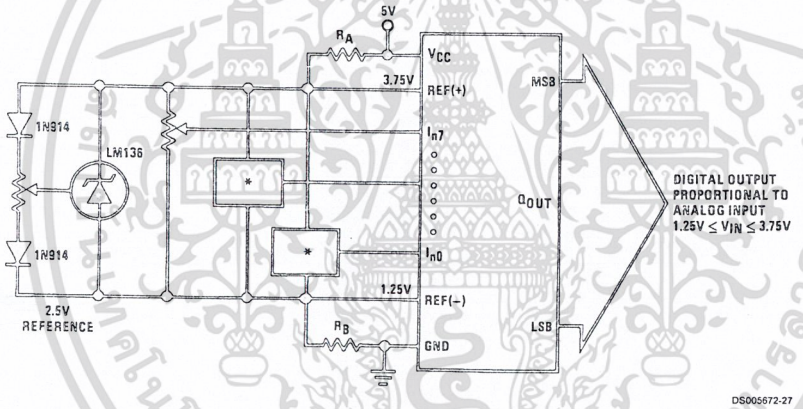


FIGURE 12. Typical Reference and Supply Circuit



$R_A = R_B$

*Ratiometric transducers

FIGURE 13. Symmetrically Centered Reference

3.0 CONVERTER EQUATIONS

The transition between adjacent codes N and N+1 is given by:

$$V_{IN} = \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} + \frac{1}{512} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (2)$$

The center of an output code N is given by:

$$V_{IN} \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (3)$$

The output code N for an arbitrary input are the integers within the range:

$$N = \frac{V_{IN} - V_{REF(-)}}{V_{REF(+)} - V_{REF(-)}} \times 256 \pm \text{Absolute Accuracy} \quad (4)$$

Where: V_{IN}=Voltage at comparator input

$V_{REF(+)}$=Voltage at Ref(+)

$V_{REF(-)}$=Voltage at Ref(−)

V_{TUE}=Total unadjusted error voltage (typically $V_{REF(+)} \div 512$)

Applications Information (Continued)

4.0 ANALOG COMPARATOR INPUTS

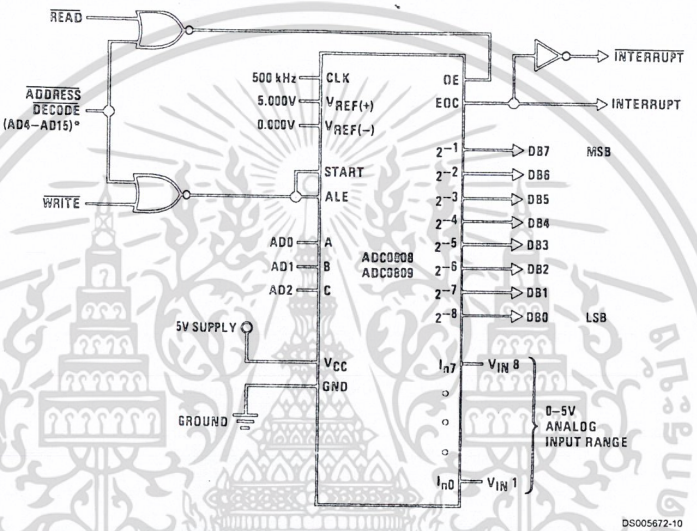
The dynamic comparator input current is caused by the periodic switching of on-chip stray capacitances. These are connected alternately to the output of the resistor ladder/switch tree network and to the comparator input as part of the operation of the chopper stabilized comparator.

The average value of the comparator input current varies directly with clock frequency and with V_{IN} as shown in *Figure 6.*

If no filter capacitors are used at the analog inputs and the signal source impedances are low, the comparator input current should not introduce converter errors, as the transient created by the capacitance discharge will die out before the comparator output is strobed.

If input filter capacitors are desired for noise reduction and signal conditioning they will tend to average out the dynamic comparator input current. It will then take on the characteristics of a DC bias current whose effect can be predicted conventionally.

Typical Application

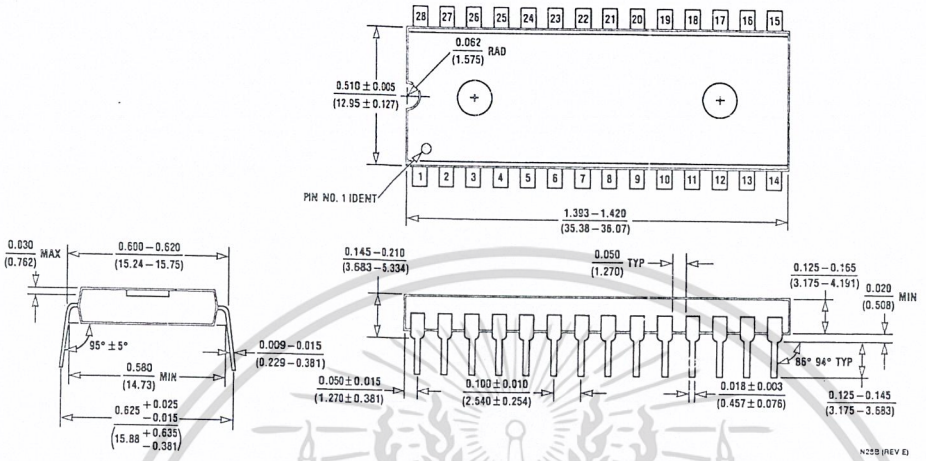


*Address latches needed for 8085 and SC/MP interfacing the ADC0808 to a microprocessor.

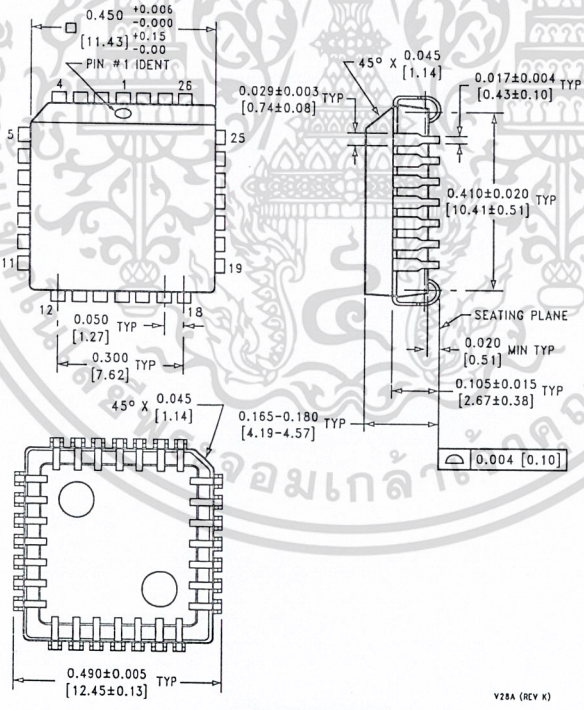
TABLE 2. Microprocessor Interface Table

PROCESSOR	READ	WRITE	INTERRUPT (COMMENT)
8080	MEMR	MEMW	INTR (Thru RST Circuit)
8085	RD	WR	INTR (Thru RST Circuit)
Z-80	RD	WR	INT (Thru RST Circuit, Mode 0)
SC/MP	NRDS	NWDS	SA (Thru Sense A)
6800	VMA•φ2•R/W	VMA•φ•R/W	IRQA or IRQB (Thru PIA)

Physical Dimensions inches (millimeters) unless otherwise noted



Molded Dual-In-Line Package (N)
Order Number ADC0808CCN or ADC0809CCN
NS Package Number N28B



Molded Chip Carrier (V)
Order Number ADC0808CCV or ADC0809CCV
NS Package Number V28A

กิตติกรรมประกาศ

ในการจัดทำโครงงานในครั้งนี้ที่สำเร็จลุล่วงได้ด้วยดีนั้น ต้องขอขอบคุณ รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา ที่คอยให้คำแนะนำแนวความคิดในการทำงานแต่ละขั้นตอน รวมทั้งตรวจแก้ไขข้อบกพร่อง และให้ข้อเสนอแนะที่เป็นประโยชน์ต่างๆ ทำให้โครงงานนี้สำเร็จลุล่วงมาได้ด้วยดี และขอขอบคุณ คุณบุญอนันต์ เกียงเอีย (พี่อั๋น), พี่แม็ก ที่ได้แนะนำวิธีการศึกษาค้นคว้า และให้การช่วยเหลือเป็นอย่างดีมาตลอด รวมถึงเพื่อนๆและพี่ๆทุกคนที่ให้กำลังใจให้คำปรึกษาและคอยช่อยเหลือจนกระทั่งปริญญานิพนธ์นี้เสร็จสมบูรณ์ได้

ผู้จัดทำ

เอกสารอ้างอิง

1. กฤษฎา วิศวธีรานนท์ , เรียน เล่นใช้ ไอซีดิจิตอล ,บริษัทซีเอ็ดยูเคชั่น จำกัด 2537
2. กฤษฎา ใจเย็น ,โครงงาน, วารสารเซมิคอนดัคเตอร์ อิเล็กทรอนิกส์ ฉบับที่ 153,2538
3. Electronics Laboratory IV , "EL 327 Data Acquisition and Conversion" ภาควิชาอิเล็กทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
4. Stefan Sjholm ,Lennart Lindh, "VHDL for Designers" A division of Simon & Schuster International Group, Prentice Hall Europe, 1997