ระบบหาต้นแอคควิซิชั่นหลายโปรโตคอลโดยใช้เอฟพีจีเอและมีการเชื่อมต่อกับพอร์ตยูเอสบี

FPGA-BASED MULTI PROTOCOL DATA ACQUISITION WITH USB INTERFACE

ศิริศักดิ์ ถานี

SIRISAK THANEE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2553

KMITL-2010-EN-M-040-022

ระบบดาต้าแอคควิซิชั่นหลายโปรโตคอลโดยใช้เอฟพีจีเอและมีการเชื่อมต่อกับพอร์ตยูเอสบี

FPGA–BASED MULTI PROTOCOL DATA ACQUISITION WITH USB INTERFACE

ศิริศักดิ์ ธานี

SIRISAK THANEE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2553

KMITL 2010-EN-M-040-022

FPGA–BASED MULTI PROTOCOL DATA ACQUISITION WITH USB INTERFACE

SIRISAK THANEE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRONICS ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2010

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ ระบบดาต้าแอคควิซิชั่นหลายโปรโตคอลโดยใช้เอฟพีจีเอและมีการเชื่อมต่อกับพอร์ตยูเอสบี

Thesis Title FPGA-Based Multi Protocal Data Acquisition with USB Interface

นักศึกษา นายศิริศักดิ์ ธานี

รหัสประจำตัว 48060403

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์

อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.ดร.สุริภณ สมควรพาณิชย์

หมายเลขวิทยานิพนธ์ KMITL-2010-EN-M-040-022

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.สมศักดิ์	มิตะถา	
ผศ.ดร.อรฉัตร	จิตต์โสภักต์	
ผศ.ดร.พินิจ	กำหอม	
รศ.สุชาติ	คูณทวีเทพ	
รศ.ดร.สุริภณ	สมควรพาณิชย์	

วัน / เดือน/ ปี ที่สอบ วันจันทร์ที่ 8 มีนาคม พ.ศ. 2553 เวลา 10.00-12.00 น.

สถานที่สอบ ณ อาคาร A ชั้น 3 ห้องประชุม 2

คณะวิศวกรรมศาสตร์ รับรองแล้ว

(รองศาสตราจารย์ ดร.กอบชัย เดชหาญ)

คณบดี คณะวิศวกรรมศาสตร์

วันที่ 8 มีนาคม พ.ศ. 2553

หัวข้อวิทยานิพนธ์	ระบบดาต้าแอคควิซิชั่นหลายโปรโตคอลโดยใช้เอฟพีจีเอและมีการเชื่อมต่อกับพอร์ตยูเอสบี
นักศึกษา	นายศิริศักดิ์ ธานี
รหัสนักศึกษา	48060403
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมอิเล็กทรอนิกส์
พ.ศ.	2553
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร สุริภณ สมควรพาณิชย์
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	รศ. สุชาติ คุณทวีเทพ

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้อธิบายถึงการนำเอาชิปเอฟพีจีเอมาออกแบบเป็นระบบดาต้าแอคควิซิชั่นที่มีการเชื่อมต่อกับคอมพิวเตอร์ผ่านพอร์ตยูเอสบีโดยได้แสดงถึงระบบที่ประกอบด้วยการติดต่อกับไอซีแปลงสัญญาณอะนาลอกเป็นดิจิตอลในโปรโตคอลที่หลากหลายซึ่งในทางปฏิบัติได้ถูกนำมาใช้งานประกอบกันเป็นระบบใหญ่ 1 ระบบ ด้วยข้อดีของเอฟพีจีเอในแง่การทำงานของแต่ละโมดูลภายในชิปสามารถทำงานได้อย่างเป็นอิสระจึงทำให้มีประสิทธิภาพมากเมื่อนำเอาเอฟพีจีเอมาใช้ในระบบที่มีหลายช่องสัญญาณซึ่งแต่ละช่องสัญญาณนั้นมีโปรโตคอลเชื่อมต่อที่แตกต่างกัน ซึ่งในการออกแบบนี้ได้นำเสนอโปรโตคอลซึ่งประกอบด้วย โปรโตคอลแบบขนาน ,โปรโตคอลแบบ SPI, โปรโตคอลแบบ I2C และโปรโตคอลแบบ 1-Wire มาเป็นโปรโตคอลตัวอย่างของระบบนี้ นอกจากนั้นโปรแกรมใช้งานเพื่อแสดงผลและเก็บข้อมูลในคอมพิวเตอร์ซึ่งเขียนขึ้นด้วยภาษา Visual C++ ยังช่วยให้ระบบตอบสนองต่อการรับข้อมูลความเร็วสูงได้ดีอีกด้วย

Thesis Title	FPGA–Based Multi Protocol Data Acquisition with USB Interface
Student	Mr. Sirisak Thanee
Student ID.	48060403
Degree	Master of Engineering
Program	Electronics Engineering
Year	2010
Thesis Advisor	Assoc. Prof. Dr. Suripon Somkuarnpanit
Thesis Co-Advisor	Assoc. Prof. Suchat Khuntaweetep

ABSTRACT

This paper describes the implementation of the FPGA as a data acquisition system with USB interface. This can simplify the data interfacing to the PC by installing most data transfer protocols into one system. The FPGA has the advantage that it allows individual modules on a chip to work independently from each other. Therefore, we can utilize the FPGA as a performance solution for a system which is multi-channeled, with connections to four ADC signals, with the four different protocols of: Parallel, SPI, I^2C and One-Wire. In addition, the visual C^{++} programming for the user interface and data storage application, on a PC, helps this design to respond better to high rates of data acquisition.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้อย่างดี ด้วยคำแนะนำ และคำปรึกษาจาก รศ.ดร. สุริภณ สมควรพาณิชย์ และ รศ. สุชาติ คูณทวีเทพ ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ข้าพเจ้ารู้สึกทราบซึ้งในความอนุเคราะห์จากท่านอาจารย์ทั้งสองท่าน และขอขอบพระคุณเป็นอย่างสูง

ขอกราบขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุก ๆ ท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้า

ขอขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในภาควิชาวิศวกรรมอิเล็กทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกคนที่ให้คำแนะนำต่างๆ และคอยให้กำลังใจเสมอมา

ขอขอบคุณบัณฑิตศึกษาและบัณฑิตวิทยาลัย คณะวิศวกรรมศาสตร์ที่ให้ความช่วยเหลือในเรื่องต่างๆ

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจและให้การสนับสนุนในทุกเรื่องๆ ทำให้ข้าพเจ้าสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

ศิริศักดิ์ ธานี

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การนำเอาระบบดาต้าแอกควิซิชั่นมาประยุกต์ใช้ในงานวัดและทดสอบอัตโนมัติเพื่องานวิทยาศาสตร์และวิศวกรรมนั้น สามารถกระทำได้หลากหลายวิธีมาก เนื่องจากผู้พัฒนาระบบหรือผู้วิจัยสามารถเพิ่มเติมปรับแต่งและกำหนดค่าได้ตามแต่ลักษณะของซอฟแวร์ประยุกต์นั้นๆ ในหน้าที่หลักของซอฟแวร์คือ เก็บรวบรวมข้อมูลจากอุปกรณ์วัดหรือตัวตรวจจับมาเก็บไว้ในฐานข้อมูลแล้วทำการแสดงผลข้อมูลที่ได้รับตามเวลาจริงที่หน้าจอแสดงผลในลักษณะภาพกราฟิกและสามารถป้อนข้อมูลตั้งค่าการทดสอบได้ นำข้อมูลที่รวบรวมได้แสดงออกมาเป็นกราฟสรุปคุณลักษณะของข้อมูล ซึ่งระบบที่ทำงานดังกล่าวนี้ ข้อมูลจากภายนอกจะเป็นข้อมูลที่มาจากเซ็นเซอร์หรืออาจจะเป็นข้อมูลดิจิตอลที่มาจากระบบอื่นก็ได้ แต่หากเป็นข้อมูลที่ได้มาจากอุปกรณ์ที่เป็นอะนาลอก จะต้องแปลงให้เป็นข้อมูลทางดิจิตอลก่อน โดยใช้ตัวแปลงข้อมูลจากอะนาลอกเป็นดิจิตอล(ADC) ซึ่งโดยส่วนใหญ่จะเป็นไอซี แล้วจึงนำข้อมูลที่ได้ไปประมวลผลที่ส่วนประมวลผลสัญญาณดิจิตอลอีกที สำหรับส่วนประมวลผลสัญญาณดิจิตอลที่ว่านี้ อาจจะเป็นคอมพิวเตอร์หรือไมโครคอนโทรลเลอร์ก็ได้ ซึ่งหากเป็นระบบที่ใช้คอมพิวเตอร์จะมีข้อเสียคือราคาแพง,ขนาดใหญ่,สิ้นเปลืองพลังงาน แต่จะมีข้อดีคือง่ายต่อการใช้งานซอฟแวร์ที่อยู่บนคอมพิวเตอร์ หากเป็นระบบที่ใช้ไมโครคอนโทรลเลอร์จะมีข้อดีคือ ราคาถูก, ขนาดเล็ก, ประหยัดพลังงาน มีความยืดหยุ่นในการเขียนโปรแกรมจากคอมพิวเตอร์ หากรวมเอาข้อดีของทั้งสองระบบเข้าด้วยกัน ซึ่งสามารถแสดงผลข้อมูลและควบคุมการทำงานของอุปกรณ์ได้ด้วยแล้ว ก็จะได้เป็นระบบไมโครคอนโทรลเลอร์ที่เชื่อมต่อกับคอมพิวเตอร์ ซึ่งระบบนี้จะเป็นระบบที่มีประสิทธิภาพมากในด้านการประมวลผล และแสดงผลรวมทั้งการเก็บผลลัพธ์

จากบทความเรื่อง “A Novel Design of an Industrial Data Acquisition System”[1] และ”A Microcontroller Base Data Acquisition System With USB Interface”[4] ที่ได้เคยนำเสนอไปก่อนหน้านี้นั้น เป็นการนำเสนอการใช้ไมโครคอนโทรลเลอร์เป็นหน่วยประมวลผลกลาง เพื่อประมวลผลสัญญาณที่รับเข้ามาจากตัวตรวจจับและทำการเชื่อมต่อสัญญาณที่ได้ไปยังคอมพิวเตอร์ผ่านทางพอร์ตยูเอสบี

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ในปัจจุบันมีผู้ผลิตชิปไอซีหลายแห่งได้ผลิตไอซีเพื่อทำการแปลงสัญญาณอะนาลอกเป็นดิจิตอล(Analog to digital converter-ADC) ออกมาสู่ตลาดเป็นจำนวนมาก โดยผู้ผลิตแต่ละรายก็

จะมีโปรโตคอล (protocol) เพื่อให้หน่วยประมวลผลกลาง (Central Processing Unit-CPU) ใช้เป็นแบบแผนในการอ่านข้อมูลจากไอซีที่แตกต่างกันออกไปทำให้เกิดปัญหาความแตกต่างด้านเวลาการทำงานขึ้นเมื่อระบบดาต้าแอควิซิชั่น (Data acquisition) ที่กำลังออกแบบนั้นเป็นระบบหลายช่องสัญญาณที่แต่ละช่องสัญญาณใช้โปรโตคอลในการดึงข้อมูลที่แตกต่างกัน

ด้วยเหตุที่ระบบส่วนใหญ่นิยมนำเอาไมโครคอนโทรลเลอร์มาใช้เป็นหน่วยประมวลผลกลาง ซึ่งคุณสมบัติในการทำงานของไมโครคอนโทรลเลอร์นั้นมีลักษณะการทำงานตามลำดับของโค้ด หากนำมาใช้งานในระบบหลายช่องสัญญาณก็จะทำงานวนตามลำดับของช่องสัญญาณที่มีอยู่ในระบบ เช่น สมมุติว่าระบบเป็นระบบ 3 ช่องสัญญาณก็จะเริ่มงานในช่องที่ 1 ก่อน เมื่อเสร็จสิ้นการทำงานในช่องนี้ก็จะไปทำงานต่อในช่องที่ 2 และเมื่อสิ้นสุดการทำงานในช่องสัญญาณที่ 2 ก็จะข้ามไปทำงานต่อในช่องที่ 3 ซึ่งเป็นช่องสุดท้ายและเมื่อสิ้นสุดการทำงานในช่องนี้แล้วก็จะย้อนกลับไปเริ่มการทำงานในช่องที่ 1 ใหม่อีกครั้ง เป็นต้น

การทำงานในลักษณะเช่นนี้ก็ไม่ถือว่าเป็นเรื่องผิดปกติแต่อย่างใดเมื่อแต่ละช่องสัญญาณในระบบนั้นมีช่วงเวลาการดึงข้อมูลจาก ADC ใกล้เคียงกัน แต่เมื่อเป็นระบบที่แต่ละช่องสัญญาณใช้เวลาแตกต่างกันมากจะทำให้เกิดปัญหาขึ้นในทันที ยกตัวอย่างเช่น ในระบบหนึ่งมีช่องสัญญาณ ADC ที่ใช้โปรโตคอล 1-Wire ร่วมกับโปรโตคอล SPI ซึ่งในการทำงานนั้นโปรโตคอล 1-Wire ใช้เวลาในการดึงข้อมูล 500 มิลลิวินาที ในขณะที่โปรโตคอล SPI ใช้เวลาในการดึงข้อมูลไม่ถึง 1 มิลลิวินาที ซึ่งจะเห็นว่าแตกต่างกันมากกว่า 500 เท่า ดังนั้น ปัญหาที่ตามมาก็คือในขณะที่ไมโครคอนโทรลเลอร์กำลังทำงานเพื่อดึงข้อมูลจากช่องสัญญาณ 1-Wire อยู่นั้นจะทำให้ข้อมูลที่เกิดขึ้นที่ช่องสัญญาณ SPI ต้องสูญเสียไปโดยเปล่าประโยชน์เป็นเวลากว่า 500 มิลลิวินาทีเลยทีเดียว

ในปัจจุบันมีวิธีการหลากหลายวิธีที่ถูกออกแบบมาเพื่อแก้ปัญหานี้ เช่น การออกแบบให้ไมโครคอนโทรลเลอร์ทำงานบน Real Time Operation System โดยจะทำงานในลักษณะแบ่งเวลาการทำงานออกเป็นหลายๆเธรด โดยแต่ละเธรดรับผิดชอบทำงานให้แต่ละช่องสัญญาณเรียงลำดับกันไปแต่เป็นช่วงระยะเวลาสั้นๆในระดับไมโครวินาที ซึ่งจะทำให้ดูราวกับว่าแต่ละช่องสัญญาณทำงานไปพร้อมๆกัน ถึงแม้วิธีนี้จะช่วยแก้ปัญหาดังกล่าวข้างต้นได้ในระดับหนึ่งแต่เมื่อพิจารณาในเชิงลึกลงไปในแง่รูปแบบการทำงานแล้วจะเห็นว่ายังคงมีเวลาการทำงานที่ไม่แน่นอนเนื่องจากเวลาที่ได้รับการจัดสรรให้แต่ละเธรดนั้นมาจากระบบปฏิบัติการซึ่งต้องรับหน้าที่ในการทำ Context Switch หรือการเก็บสถานะของแต่ละเธรดก่อนการเปลี่ยนการทำงานไปยังเธรดต่อไป ซึ่งการทำ Context Switch นั้นใช้เวลาในการทำงานที่แตกต่างกันทำให้รอบเวลาการทำงานแต่ละรอบแตกต่างกันไปด้วย ดังนั้น ระบบนี้จึงไม่เหมาะกับระบบที่ต้องวัดสัญญาณในลักษณะเวลาจริง (Real Time)

ในการออกแบบนี้ได้นำเอาชิปเอฟพีจีเอมาใช้แก้ปัญหาแทนการใช้ไมโครคอนโทรลเลอร์เนื่องจากคุณสมบัติที่โดดเด่นของชิปเอฟพีจีเอในการทำงานแบบแข่งขนาน (Concurrent) ทำให้

แต่ละโมดูลภายในชิปทำงานอย่างอิสระพร้อมๆกันไปอย่างแท้จริง จึงมีประสิทธิภาพเป็นอย่างยิ่งเมื่อนำมาใช้กับระบบดาต้าแอคควิซิชั่นดังที่กล่าวมาข้างต้น

ปัจจุบันนี้ได้มีการพัฒนาหน่วยประมวลผลกลางที่ใช้ชิปเอฟพีจีเอมากขึ้น เนื่องด้วยประสิทธิภาพที่เหนือกว่าไมโครคอนโทรลเลอร์หลายประการ กล่าวคือ ในเอฟพีจีเอจะมีชุดอินพุต-เอาท์พุตพอร์ตมากกว่า ซึ่งจะทำให้การออกแบบสามารถที่สร้างเป็นระบบ multi-channel ได้ง่าย โดยไม่จำเป็นต้องมีไอซีเพื่อขยายพอร์ตเพิ่มเติม นอกจากนั้น ในการสร้างวงจรการทำงานบนเอฟพีจีเอนั้นจะเป็นในลักษณะสร้างลอจิกเกตไปยังตัวชิป และมีสัญญาณนาฬิกาควบคุมทั้งระบบ แต่การทำงานของไมโครคอนโทรลเลอร์จะทำงานในลักษณะบรรทัดต่อบรรทัดของโค้ด ถึงแม้จะใช้สัญญาณนาฬิกาเดียวกันแต่การทำงานในบางช่วงจะมีการทำงานที่มีเวลาไม่แน่นอน ดังนั้นจึงทำให้เอฟพีจีเอมีเสถียรภาพในการทำงานที่เหนือกว่า เมื่อมองในแง่ความเร็วของสัญญาณนาฬิกา ชิปเอฟพีจีเอก็ยังสามารถทำงานได้ที่สัญญาณนาฬิกาที่สูงกว่าไมโครคอนโทรลเลอร์ ซึ่งในปัจจุบันสามารถทำงานได้ที่สัญญาณนาฬิกามากกว่า 500 เมกะเฮิร์ทเลยทีเดียว

1.3 สมมติฐานของการศึกษา

ในการออกแบบหน่วยประมวลผลกลางเพื่อใช้ในระบบดาต้าแอคควิซิชั่นที่มีหลายช่องสัญญาณนั้นจำเป็นจะต้องทำการออกแบบโมดูลภายในชิปของแต่ละช่องสัญญาณเพื่อทำหน้าที่ดึงข้อมูลจากไอซี ADC (Analog to Digital Converter) แล้วนำมาเก็บไว้ในหน่วยความจำซึ่งอาจจะเป็นชนิดเข้าก่อนออกก่อน (FIFO) หรือชนิดใดก็ได้ แต่ทุกๆช่องสัญญาณจะต้องมี 2 องค์ประกอบนี้เป็นหลัก นอกจากนั้น จะต้องมีส่วนควบคุมเพื่อทำการจัดลำดับและลำเลียงข้อมูลออกไปที่คอมพิวเตอร์ผ่านทางพอร์ตยูเอสบี ซึ่งก็หมายความว่าในชิปเอฟพีจีเอจะต้องมีอีกหนึ่งโมดูลเพื่อทำการสร้างการเชื่อมต่อและสื่อสารกับไอซีที่ทำหน้าที่รับหรือส่งข้อมูลกับพอร์ตยูเอสบีอีกโมดูลหนึ่งด้วย

ในส่วนการแสดงผลและจัดเก็บข้อมูลในคอมพิวเตอร์นั้น จำเป็นจะต้องใช้ความสามารถในการประมวลผลพร้อมๆกันหลายเธรด(Multithreading) เพื่อให้โปรแกรมสามารถตอบสนองต่อข้อมูลความเร็วสูงได้อย่างมีประสิทธิภาพและใช้เทคนิคการ Synchronization เพื่อป้องกันการใช้ทรัพยากรร่วมกันของแต่ละเธรด

1.4 ทฤษฎีหรือแนวคิดที่ใช้ในการวิจัย

ในการออกแบบจะต้องทำการศึกษาเพื่อให้ทราบถึงกลไกหรือรูปแบบในการดึงข้อมูลจาก ADC (Analog to Digital Converter) ในแต่ละโปรโตคอลที่นำมาใช้ในการออกแบบ รวมทั้งรูปแบบของโปรโตคอลที่ใช้กับไอซีที่ทำหน้าที่จัดการการทำงานของพอร์ตยูเอสบีด้วย ซึ่งรูปแบบ

ดังกล่าวจะถูกนำเอามาอธิบายหรือเขียนเป็นโมดูลในชิปเอฟพีจีเอด้วยภาษา VHDL (Very High Speed Integrated Circuit, Hardware Description Language) นอกจากนั้น ในการเขียนแอพพลิเคชั่นซอฟแวร์บนคอมพิวเตอร์จำเป็นต้องศึกษาการเขียนโปรแกรมด้วยภาษา Visual C++ ในแง่ของ Multithreading, Synchronization และ System programming ประกอบกันเพื่อให้ซอฟแวร์ตอบสนองต่อระบบด้วยความเร็วสูงได้ตามต้องการ

1.5 การเปรียบเทียบระหว่างวิธีการที่นำเสนอกับวิธีการแบบพื้นฐาน

อ้างอิงบทความเรื่อง "FPGA-Based Acquisition of Sensor Data"[3] ซึ่งได้มีการประยุกต์ใช้เอฟพีจีเอในการประมวลผลสัญญาณดิจิตอลในระบบดาต้าแอควิซิชั่น แต่การออกแบบดังกล่าว ทำการแสดงผลข้อมูลที่ได้รับในบอร์ดเอฟพีจีเอโดยตรงโดยไม่มีการเชื่อมต่อไปยังคอมพิวเตอร์ ทำให้การเรียกดูข้อมูลมีรูปแบบที่ตายตัว ยากต่อการทำความเข้าใจกับผู้ใช้ และหน่วยความจำที่ใช้เก็บข้อมูลในบอร์ดก็มีจำกัด ทำให้การย้อนกลับมาดูข้อมูลในอดีตไม่สามารถทำได้ ดังนั้นในการออกแบบนี้จะนำเสนอการประยุกต์ใช้เอฟพีจีเอเป็นหน่วยประมวลผลกลางในระบบดาต้าแอควิซิชั่น พร้อมด้วยการเชื่อมต่อกับคอมพิวเตอร์เพื่อการแสดงผลที่ดูเข้าใจง่าย พร้อมทั้งมีการเก็บข้อมูลไว้ในระบบฐานข้อมูล สามารถเรียกข้อมูลในอดีตกลับมาดูได้อีกด้วย

1.6 ขอบเขตการวิจัย

ในวิทยานิพนธ์ฉบับนี้ได้จำลองระบบดาต้าแอคควิซิชั่นสายโปรโตคอล (Multi-Protocol Data Acquisition) โดยใช้หน่วยประมวลผลกลางเป็นชิปเอฟพีจีเอขึ้น ซึ่งแต่ละช่องสัญญาณนั้นประกอบด้วย 4 ช่องสัญญาณ คือ Parallel, SPI, I^2C และ 1-Wire โดยข้อมูลที่ได้รับทั้งหมดจะถูกส่งไปยังคอมพิวเตอร์ผ่านทางพอร์ตยูเอสบี

ทางด้านคอมพิวเตอร์จะมีซอฟแวร์ซึ่งถูกเขียนขึ้นจากภาษา Visual C++ เพื่อสนับสนุนกับระบบที่ออกแบบนี้โดยเฉพาะ โดยหน้าที่หลักของซอฟท์แวร์นั้นทำหน้าที่นำข้อมูลที่ได้รับมานั้นออกแสดงผลที่หน้าจอมอนิเตอร์ในลักษณะกราฟสรุปคุณลักษณะหรือเป็นค่าที่วัดได้ในขณะนั้น นอกจากนั้น ก็ยังทำหน้าที่เป็นส่วน User Interface ให้ผู้ใช้บันทึกค่าลงในฐานข้อมูลและนำเอาข้อมูลที่ได้นั้นไปตีความเพื่อนำเอาผลกลับมาควบคุมระบบที่กำลังวัดและทดสอบอยู่นั้นอีกหน้าที่หนึ่งด้วย ซึ่งระบบที่ได้จำลองขึ้นมามีคุณสมบัติดังนี้

- ใช้ความจุเกตของชิปเอฟพีจีเอทั้งระบบไม่เกิน 20,000 เกต
- ความถี่ใช้งานสูงสุดของระบบไม่ต่ำกว่า 30 MHz
- ความเร็วในการส่งถ่ายข้อมูลผ่านพอร์ตยูเอสบี 12Mbps
- Sampling Rate สูงสุด 200 KHz

- ใช้พอร์ตเชื่อมต่อของชิปเอพีจีเอไม่เกิน 50 พอร์ต
- กำลังไฟฟ้าโดยรวมของทั้ง 4 ช่องสัญญาณ 100 mW

1.7 ขั้นตอนของการศึกษา

วิทยานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 5 บทประกอบด้วย

บทที่ 1 กล่าวถึงความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ สมมติฐาน ทฤษฎีที่ใช้ ทฤษฎีหรือแนวคิดที่ใช้ในการวิจัย และขั้นตอนการศึกษา

บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานที่ใช้ในการวิจัย ประกอบด้วยพื้นฐานของระบบดาต้าแอคควิซิชั่น, เอฟพีจีเอ, ภาษาวีเอชดีแอล, การเขียนโปรแกรมภาษาวิชวลซีพลัสพลัส

บทที่ 3 กล่าวถึงการออกแบบโมดูลต่างๆที่ประกอบอยู่ในชิปเอฟพีจีเอและหน้าที่การทำงานของแต่ละโมดูล รวมไปถึงผลการจำลองการทำงานของแต่ละโมดูล

บทที่ 4 กล่าวถึงการออกแบบแอพพลิเคชั่นโปรแกรมเพื่อทำการแสดงผลและเก็บข้อมูลในคอมพิวเตอร์รวมไปถึงเทคนิคการเขียนโปรแกรมในแง่ Multithreading และ Synchronization

บทที่ 5 เป็นบทสรุปผลการวิจัยและข้อเสนอแนะ

บทที่ 2
ทฤษฎีพื้นฐานที่ใช้ในการวิจัย

ในหัวข้อนี้จะกล่าวถึงทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องในการวิจัย และพื้นฐานของระบบดาต้าแอคควิซิชั่น ซึ่งเนื้อหาในบทนี้จะกล่าวถึงองค์ประกอบของระบบดาต้าแอคควิซิชั่น, พื้นฐานเอฟพีจีเอ, ภาษาวีเอชดีแอล, ภาษาวิชวลซีพลัสพลัส ซึ่งเป็นความรู้พื้นฐานเพื่อเป็นแนวทางในการออกแบบระบบ

2.1 ระบบดาต้าแอคควิซิชั่น

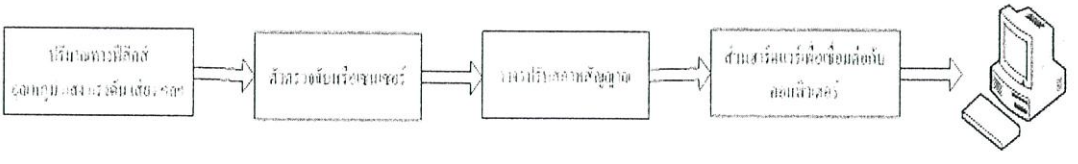
เป็นการเก็บรวบรวมวิเคราะห์ข้อมูลจริงในงานวิจัยทดลองวิทยาศาสตร์และทดสอบงานวิศวกรรมเชิงคุณภาพและประสิทธิผลผ่านคอมพิวเตอร์ โดยมีความแตกต่างจากงานระบบคอมพิวเตอร์ทั่วไปตรงที่มีฮาร์ดแวร์พิเศษเพื่อตรวจจับสัญญาณทางกายภาพทางวิทยาศาสตร์ อาทิ ความดันอากาศหรืออัตราการไหลแล้วแปลงเข้าสู่ระบบคอมพิวเตอร์ผ่านซอฟแวร์ประยุกต์ที่พัฒนาตามคุณลักษณะของงานวิจัยทดลองนั้นๆ ในลักษณะเวลาจริง (Real Time) ซึ่งในอดีตมักใช้เป็นระบบเฉพาะเจาะจงลงไปตามประเภทงานไม่สามารถใช้งานร่วมกับงานวิจัยอื่นได้ อีกทั้งยังมีราคาที่สูงมาก ทว่าด้วยความสามารถของคอมพิวเตอร์ในปัจจุบันทำให้การประยุกต์ใช้ระบบดาต้าแอคควิซิชั่นนี้มีความเป็นไปได้อย่างไม่ยุ่งยาก

การประยุกต์ใช้งานในระบบดาต้าแอคควิซิชั่น พอยกตัวอย่างได้ เช่น ระบบวัดทางไฟฟ้า, การวัดการสื่อสารข้อมูลดิจิตอล, การทดลองทฤษฎีงานควบคุม, งานประมวลผลสัญญาณ เป็นต้น

2.1.1 ส่วนประกอบสำคัญของระบบดาต้าแอคควิซิชั่น

โครงสร้างของระบบดาต้าแอคควิซิชั่นแสดงในรูปที่ 2.1 โดยประกอบด้วยส่วนสำคัญ 4 ส่วน คือ

i. ส่วนตรวจจับสัญญาณกายภาพ (Transducer/Sensor/Detector)
ii. ส่วนปรับสภาพสัญญาณ (Signal Conditioning)
iii. ส่วนฮาร์ดแวร์เพื่อเชื่อมต่อคอมพิวเตอร์ (Data Acquisition Board)
iv. ซอฟต์แวร์ใช้งาน (Application Software)



รูปที่ 2.1 โครงสร้างของระบบดาต้าแอคควิซิชั่น

2.1.1.1 ส่วนตรวจจับสัญญาณกายภาพ (Transducer/Sensor/Detector)

ชุดตรวจจับสัญญาณกายภาพหรือ ทรานสดิวเซอร์ (Transducer) คือตัวแปลงสัญญาณทางกายภาพเช่น อุณหภูมิ ความดัน ระดับของไหล ฯลฯ ให้อยู่ในรูปสัญญาณทางไฟฟ้า หรือบางครั้งเรียกอุปกรณ์เหล่านี้ว่า เซนเซอร์ (Sensor) ซึ่งมีด้วยกันหลากหลายรูปแบบ อาทิ เทอร์โมคัปเปิล (Thermocouple) เทอร์มิสเตอร์ (thermistor) และสเตรนเกจวัดแรงดัน (Strain gauges) เป็นต้น

2.1.1.2 ส่วนปรับสภาพสัญญาณ (Signal Conditioning)

หน้าที่หลักของส่วนปรับสภาพสัญญาณนี้คือ ปรับปรุงคุณภาพของสัญญาณไฟฟ้าที่ได้จากชุดตรวจจับก่อนส่งสัญญาณต่อไปยังฮาร์ดแวร์ที่เชื่อมต่อกับคอมพิวเตอร์ ซึ่งอาจมีความจำเป็นต้องปรับสเกลสัญญาณ, ขนาดสัญญาณ, แปลงรูปแบบสัญญาณให้เป็นเชิงเส้น (linearization), กรองคลื่นสัญญาณ, การแยกกราวด์ของสัญญาณ (Common-mode rejection), และทำการกระตุ้นสัญญาณ

หน้าที่เด่นของส่วนปรับสภาพสัญญาณคือ ขยายขนาดสัญญาณ (Amplify) เพราะโดยส่วนใหญ่สัญญาณที่ได้จากเซนเซอร์จะมีขนาดสัญญาณที่ต่ำมากในหน่วยมิลลิโวลต์ และมักมีสัญญาณรบกวนจากแหล่งจ่ายไฟปะปนมาซึ่งอาจรบกวนสัญญาณด้านอินพุตในขณะที่สัญญาณเข้าสู่ระบบทำให้ค่าสัญญาณที่วัดไม่ถูกต้องและเที่ยงตรงได้

นอกจากนั้นวงจรปรับสภาพสัญญาณยังใช้ในการแปลงสัญญาณไฟฟ้าที่ไม่ได้อยู่ในรูปของแรงดันไฟฟ้า เช่น กระแสไฟฟ้าหรือความต้านทานไฟฟ้ามาอยู่ในรูปของแรงดันไฟฟ้าให้เหมาะสมกับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลที่อยู่ในฮาร์ดแวร์พิเศษสำหรับวัดคุม เช่น แปลงค่ากระแสไฟฟ้า 4-20 mA จากตัวตรวจจับที่ให้ผลแบบกระแสไฟฟ้าเป็นแรงดันไฟตรง 0-5V หรือปรับแรงดันไฟตรงจากเซนเซอร์ที่ให้ผลเป็นแรงดัน -10V ถึง +10V เป็น 0 ถึง 5V เป็นต้น

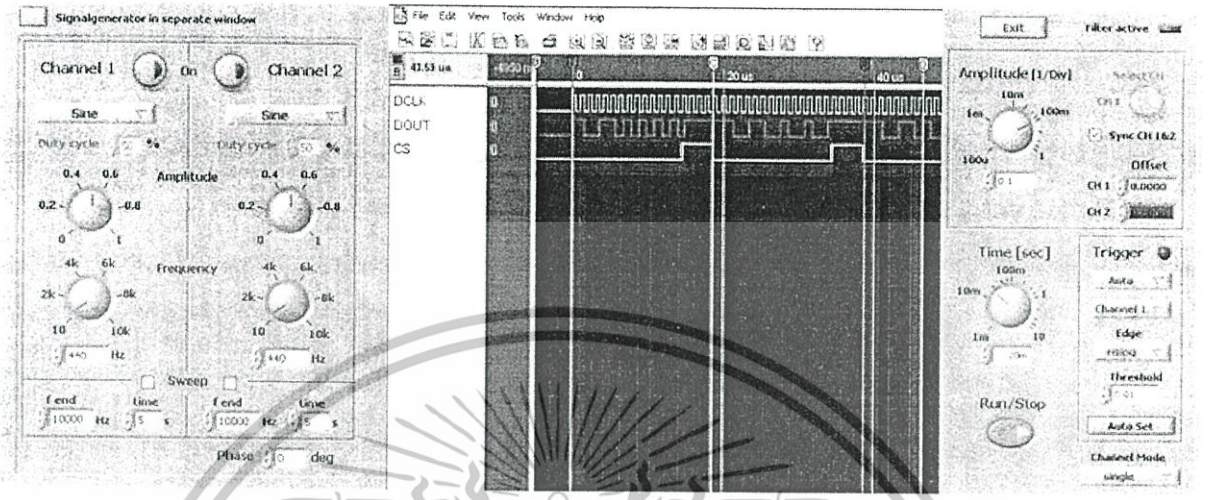
2.1.1.3 ส่วนฮาร์ดแวร์เพื่อเชื่อมต่อคอมพิวเตอร์ (Data Acquisition Board)

ฮาร์ดแวร์ที่ใช้วัดคุมหรือเชื่อมต่อเพื่อส่งผ่านสัญญาณเข้าสู่ระบบคอมพิวเตอร์นั้นมีอยู่ 2 แบบหลักๆคือ แบบการ์ดเสียบ (Plug-in DAQ card) และแบบเชื่อมต่อภายนอก (external DAQ System) ซึ่งการติดตั้งของการ์ดเสียบนี้จะกระทำผ่านบัส ISA สำหรับคอมพิวเตอร์รุ่นเก่าหรือผ่านบัส PCI สำหรับเครื่องคอมพิวเตอร์สมัยใหม่ ส่วนแบบเชื่อมต่อภายนอกจะใช้พอร์ตสื่อสารมาตรฐาน ซึ่งได้แก่พอร์ตอนุกรม RS-232, พอร์ขนาน IEEE1284 (พอร์ตเครื่องพิมพ์) พอร์ตขนาน IEEE1284 หรือบัส GPIB รวมถึงบัส USB ซึ่งเป็นบัสอนุกรมความเร็วสูงที่ถูกใช้ในการออแบบนี้

2.1.1.4 ซอฟต์แวร์ใช้งาน (Application Software)

ส่วนประกอบนี้จะขึ้นอยู่กับผู้พัฒนาระบบเป็นหลัก นั่นคือ ความถนัดในการใช้งานซอฟแวร์และการเขียนโปรแกรม โดยปัจจุบันผู้ผลิตฮาร์ดแวร์สำหรับงานดาต้าแอคควิซิชั่นมักรองรับทั้งการเลือกใช้ซอฟแวร์สำเร็จรูปและมีไดร์เวอร์สนับสนุนหรือทูลซอฟแวร์อย่าง OCX

สำหรับภาษาพัฒนาโปรแกรมต่างๆ เช่น Visual C++, Visual Basic หรือ Delphi เพื่อให้สามารถพัฒนาโปรแกรมได้อีกด้วย ในรูปที่ 2.2 แสดงตัวอย่างซอฟแวร์ในระบบดาต้าแอคควิซิชั่น



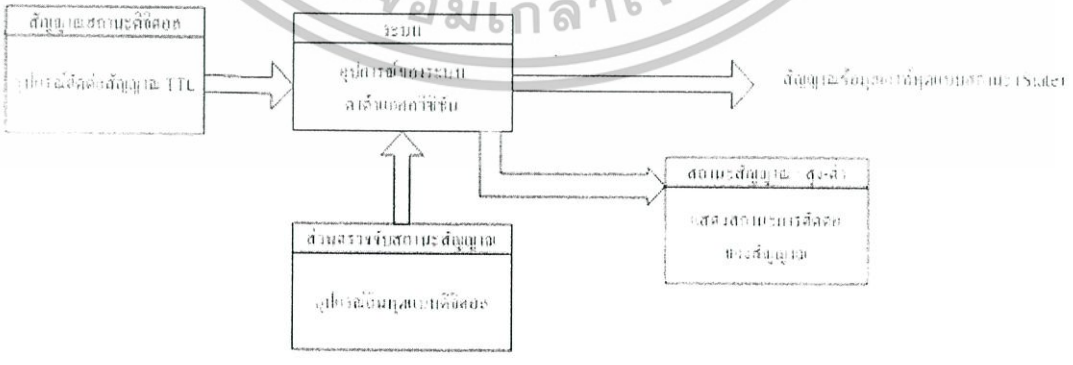
รูปที่ 2.2 ตัวอย่างซอฟแวร์ในระบบดาต้าแอคควิซิชั่น

2.1.2 ลักษณะสัญญาณในระบบดาต้าแอคควิซิชั่น

ในระบบดาต้าแอคควิซิชั่นได้กำหนดลักษณะสัญญาณทั้งด้านอินพุตและเอาท์พุตเป็นลักษณะใหญ่ๆคือ สัญญาณแบบดิจิตอลและสัญญาณแบบอะนาลอก

2.1.2.1 สัญญาณแบบดิจิตอล

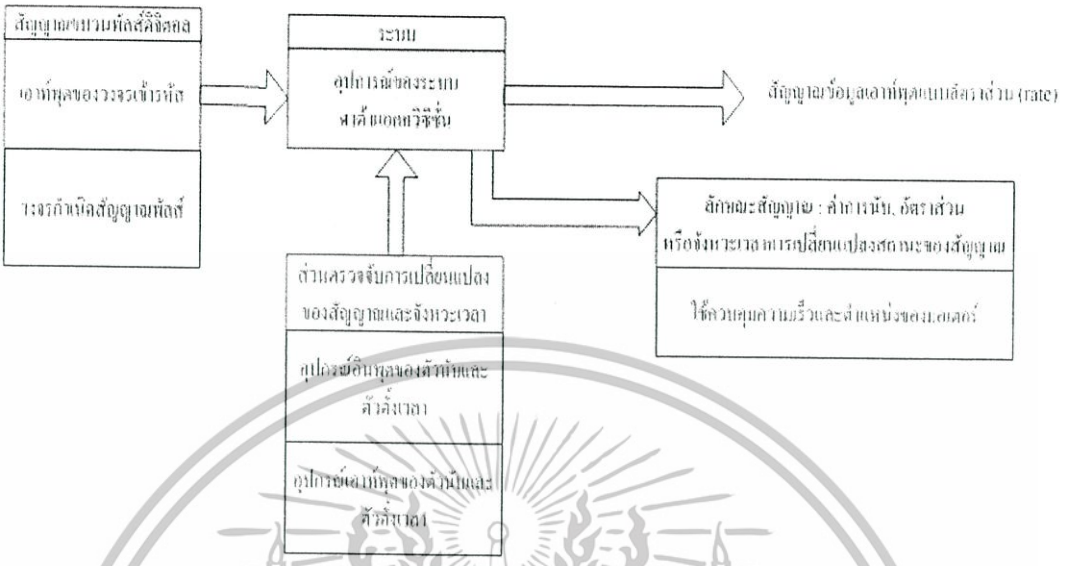
สัญญาณดิจิตอลในระบบดาต้าแอคควิซิชั่นมี 2 แบบ คือ ค่าสถานะ (State) และ อัตราการส่งสัญญาณ (rate) โดยค่าสถานะด้านอินพุตคือ การอ่านค่าสถานะเปิดหรือปิด (อาจพิจารณาเป็น 1 หรือ 0 ก็ได้) จากอุปกรณ์ประเภทสวิตช์และถ้าพิจารณาทางด้านเอาท์พุตก็จะเป็นการส่งค่าเปิด/ปิดไปยังอุปกรณ์อย่างรีเลย์หรือวาล์วดังแสดงในรูป 2.3



รูปที่ 2.3 กระบวนการทำงานกับสถานะดิจิตอลในระบบดาต้าแอคควิซิชั่น

สัญญาณอีกชนิดหนึ่งคือสัญญาณแบบอัตราการส่งสัญญาณ มักเป็นการอ่านค่าพัลส์จากอุปกรณ์จำพวกเอ็นโค้ดเดอร์ (Encoder) หรือตัวเข้ารหัสและอ่านค่าสัญญาณนาฬิกาจากอุปกรณ์

หรือวงจรกำเนิดสัญญาณภายนอก ถ้าพิจารณาทางด้านเอาท์พุตก็จะเป็นสัญญาณควบคุมความกว้างพัลส์และความถี่เพื่อใช้ในการควบคุมการเคลื่อนตำแหน่งของมอเตอร์ ดังแสดงในรูป 2.4

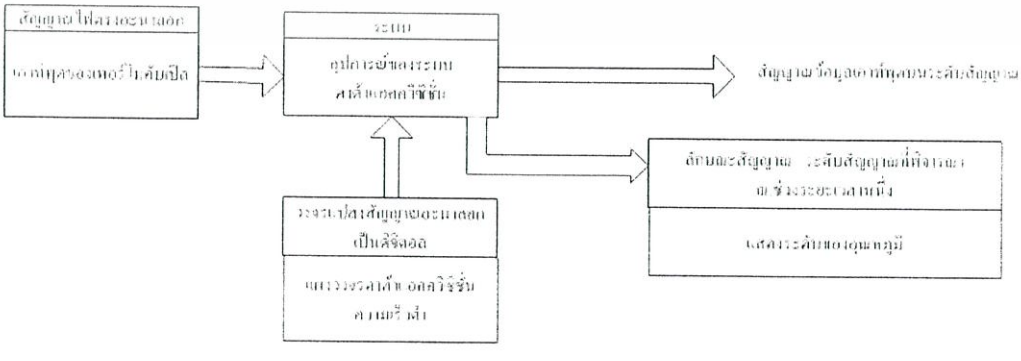


รูปที่ 2.4 กระบวนการทำงานกับสัญญาณดิจิตอลแบบขบวนพัลส์ในระบบดาต้าแอคควิซิชั่น

2.1.2.2 สัญญาณอะนาลอก

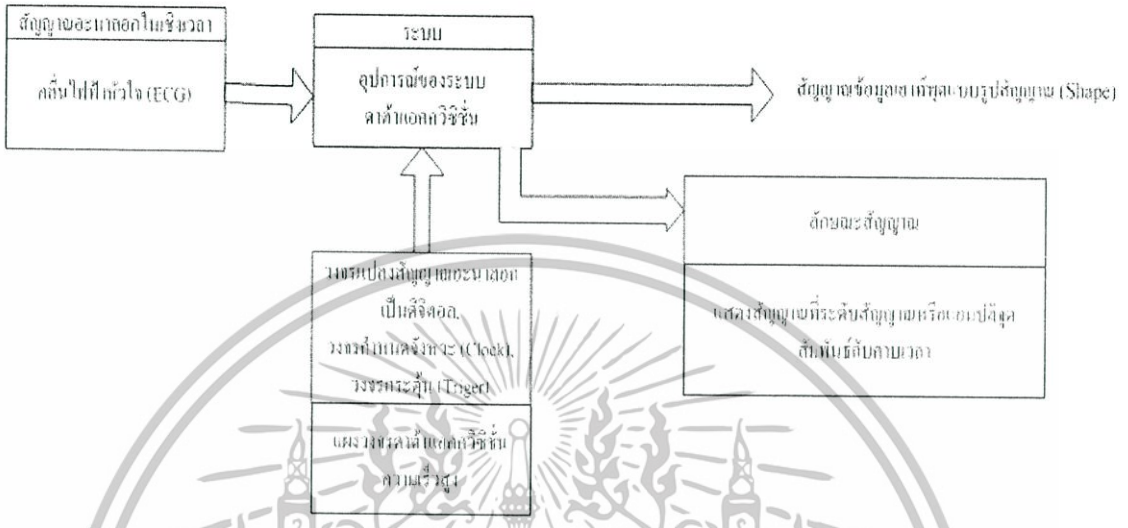
แบ่งได้เป็น 3 แบบคือ แบบสัญญาณไฟตรง (analog DC signals), แบบเปลี่ยนค่าตามเวลา (time domain) และ แบบเปลี่ยนค่าตามความถี่ (frequency domain) ซึ่งตรงกับลักษณะของข้อมูลที่ต้องการวิเคราะห์ในแต่ละลักษณะคือ ระดับสัญญาณ (level), รูปคลื่น (shape) และ ความถี่ (frequency content) ตามลำดับ

สัญญาณอะนาลอกไฟตรง มักเป็นค่าที่ได้จากการวัดขนาดหรือระดับของสัญญาณ ซึ่งมีการเปลี่ยนแปลงค่าสัญญาณในเวลาที่ไม่เร็วมากนัก เช่น อุณหภูมิ, ความดัน, อัตราการไหล เป็นต้น สามารถใช้วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC: Analog to digital converter) ที่มีอัตราการสุ่มสัญญาณไม่เร็วมากได้ ดังแสดงในรูปที่ 2.5



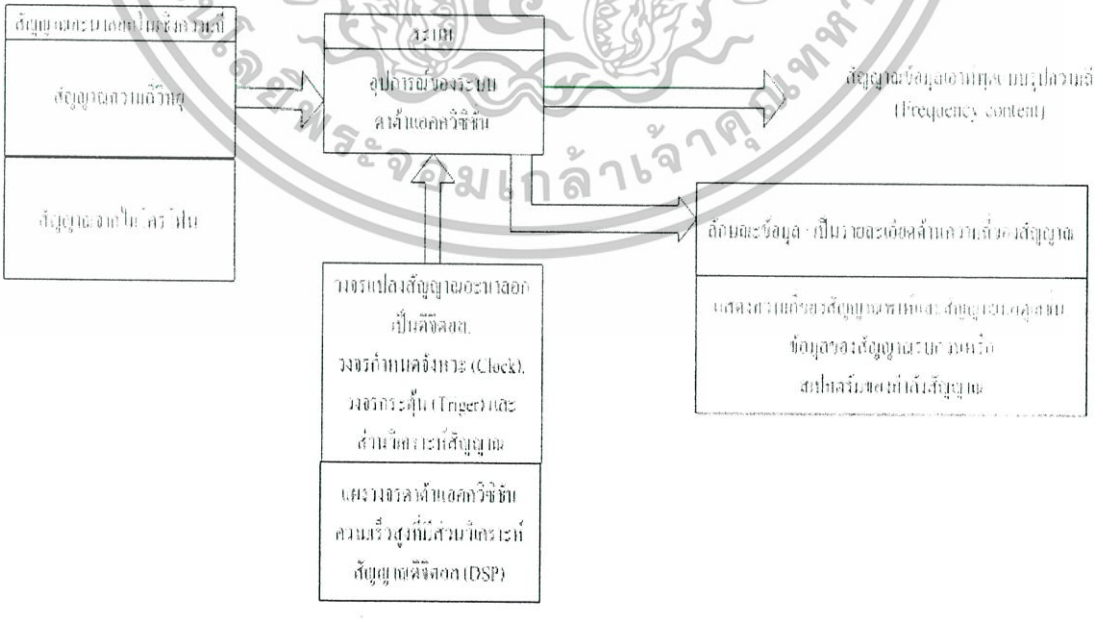
รูปที่ 2.5 กระบวนการทำงานกับสัญญาณไฟตรงอะนาลอกในระบบดาต้าแอคควิซิชั่น

สัญญาณอะนาลอกแบบที่เปลี่ยนค่าตามเวลา เป็นสัญญาณที่วัดเพื่อพิจารณาลักษณะรูปสัญญาณเป็นหลัก เช่น สัญญาณคลื่นหัวใจ (ECG) ซึ่งมีความจำเป็นต้องใช้วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลความเร็วสูง และอาจต้องมีฮาร์ดแวร์พิเศษเพื่อให้ค่าเวลาที่ได้จากสัญญาณมีความเที่ยงตรง ดังแสดงในรูปที่ 2.6



รูปที่ 2.6 กระบวนการทำงานกับสัญญาณอะนาลอกในเชิงเวลาของระบบคาต้าแอคควิซิชั่น

สัญญาณอะนาลอกแบบเปลี่ยนแปลงค่าตามความถี่ ได้แก่ สัญญาณความถี่วิทยุ (radio frequency : RF) และสัญญาณคลื่นเสียง เป็นต้น ในการวิเคราะห์จำเป็นต้องมีฮาร์แวร์พิเศษที่ช่วยวิเคราะห์อย่าง DSP (Digital Signal Processing) ทำงานร่วมกับวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ดังแสดงในรูปที่ 2.7



รูปที่ 2.7 กระบวนการทำงานกับสัญญาณอะนาลอกในเชิงความถี่ของระบบคาต้าแอคควิซิชั่น

2.2 ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลในระบบดาต้าแอคควิซิชั่น

จุดมุ่งหมายสำคัญในการแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอลนั้นคือ ข้อดีหลายประการของสัญญาณหรือข้อมูลแบบดิจิตอลรวมทั้งคุณสมบัติการทำงานของการประมวลผลสัญญาณดิจิตอลที่เหนือกว่าสัญญาณอะนาลอกซึ่งได้แก่ การมีเสถียรภาพในการทำงานที่สูงกว่า สามารถปรับปรุงการทำงานได้โดยวิธีการทางซอฟต์แวร์ ทำให้ประหยัดค่าใช้จ่ายในการพัฒนาหรือปรับปรุงระบบสามารถตรวจสอบความถูกต้องของข้อมูลได้ สามารถขจัดสัญญาณรบกวนต่างๆ ได้โดยไม่ทำให้สัญญาณหรือข้อมูลต้นฉบับเกิดการผิดเพี้ยน เป็นต้น คุณสมบัติต่างๆเหล่านี้ทำให้การประมวลผลตลอดจนการจัดเก็บข้อมูลด้วยวิธีการทางดิจิตอลได้รับการตอบรับอย่างสูง

ในปัจจุบันนี้ ADC ทุกชนิดทำงานโดยให้ค่าข้อมูลดิจิตอลเอาท์พุตที่แสดงถึงความสัมพันธ์ระหว่างแรงดันอินพุตกับค่าแรงดันอ้างอิงที่ป้อนให้ตัว ADC ซึ่งเราสามารถเขียนเป็นสมการได้ดังนี้

$$\text{Output} = 2^n \cdot Gain \cdot \frac{Vin}{Vref} \qquad (2.1)$$

โดยที่ Output คือ ข้อมูลดิจิตอลเอาท์พุตที่ได้จากการแปลง

n คือ จำนวนบิตของข้อมูลเอาท์พุตซึ่งเป็นตัวบ่งชี้ถึงความละเอียดของ ADC

Gain คือ ค่าอัตราการขยายของวงจรที่อาจเพิ่มเข้ามาเพื่อใช้ในการขยายแรงดัน (*Vin*) ให้มีขนาดใหญ่ขึ้นซึ่งจะทำให้การวัดสัญญาณขนาดเล็กมีความละเอียดมากยิ่งขึ้น ในกรณีที่เป็นการลดทอนสัญญาณอินพุตให้ต่ำลงเพื่อใช้ในการวัดสัญญาณที่มีค่าสูงกว่าค่าแรงดันอ้างอิงค่าอัตราการขยายนี้จะมีค่าต่ำกว่า 1 ในกรณีที่ไม่มีการขยายหรือลดทอนแรงดันอินพุตค่านี้จะมีค่าเท่ากับ 1

Vin คือ ค่าแรงดันอินพุตอะนาลอกที่เราต้องการหาค่า วัดค่าหรือแปลงเป็นข้อมูลดิจิตอล

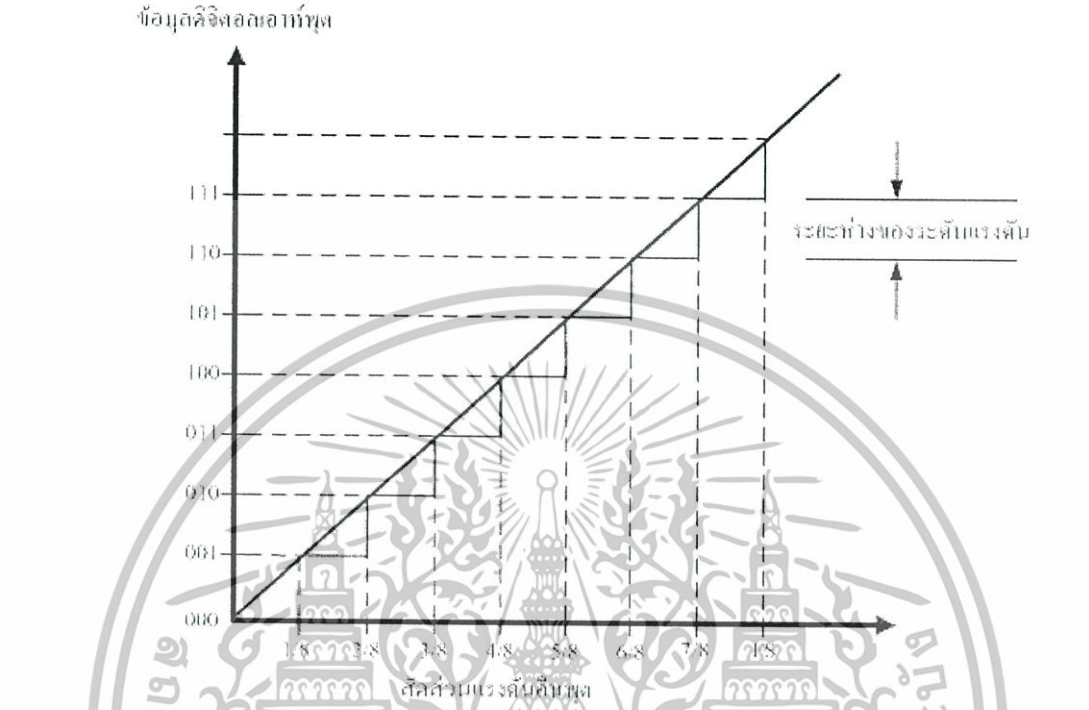
Vref คือ ค่าแรงดันอ้างอิงอะนาลอกที่เราใช้เป็นระดับอ้างอิงในการหาค่าแรงดันอินพุต

กระบวนการแปลงสัญญาณอะนาลอกเป็นดิจิตอลเป็นการแสดงความสัมพันธ์ระหว่างปริมาณของสัญญาณไฟฟ้าที่เป็นสัญญาณอะนาลอกกับข้อมูลตัวเลขที่ใช้แทนสัญญาณดิจิตอล ความแม่นยำของการแปลงจะขึ้นอยู่กับจำนวนบิตของข้อมูลดิจิตอล วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลขนาด *n* บิตจะเกิดข้อมูลดิจิตอลจำนวน 2^n ข้อมูล ยกตัวอย่างวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล 3 บิต จะเกิดข้อมูลดิจิตอลทางเอาท์พุตทั้งสิ้น 8 ข้อมูล ดังแสดงในรูป 2.8 ระยะห่างของแต่ละข้อมูลจะเป็นตัวกำหนดความแม่นยำของการแปลงสัญญาณ กระบวนการที่ทำหน้าที่ตีความระดับสัญญาณอะนาลอกว่าตรงกับข้อมูลดิจิตอลใดเรียกว่ากระบวนการควอนไตซิ่ง (quantizing)

ระยะห่างของระดับข้อมูลดิจิตอลในวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล สามารถคำนวนได้จากความสัมพันธ์ทางคณิตศาสตร์ดังนี้

$$\text{ระยะห่างของระดับแรงดัน} = V_{LSB} = \frac{V_{Fs}}{2^n}$$

โดยที่ V_{Fs} คือแรงดันเต็มสเกลหรือแรงดันสูงสุดที่สามารถเกิดขึ้นได้ในวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ปกติมีค่าเท่ากับไฟเลี้ยง



รูปที่ 2.8 กราฟแสดงความสัมพันธ์ระหว่างแรงดันอินพุตกับข้อมูลดิจิตอลเอาท์พุต

ถ้าหาก V_{Fs} ของวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล 3 บิต มีค่าเท่ากับ 5V ระยะห่างของระดับข้อมูลดิจิตอลเท่ากับ $\frac{5}{8} = 0.625$ V ข้อมูลดิจิตอลสูงสุดในรูปที่ 2.8 คือ 111_2 ซึ่งมีค่าเท่ากับ 7_{10} ดังนั้นที่ข้อมูลดิจิตอลสูงสุดของวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล 3 บิต จะมีค่าเทียบกับแรงดันอะนาลอกทางอินพุตเท่ากับ

$$\frac{7}{8} \times 5V = 4.375V$$

เมื่อเป็นเช่นนี้จึงสามารถที่จะกำหนดความสัมพันธ์ของแรงดันอะนาลอกอินพุตกับข้อมูลดิจิตอลสูงสุดในวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลไม่ว่าจะเป็นกี่บิตก็ตามได้ดังนี้

$$\textit{แรงดันอะนาลอกอินพุตที่ทำให้เกิดข้อมูลดิจิตอลสูงสุด} = V_{Fs} - V_{LSB}$$

โดยที่ V_{Fs} คือแรงดันเต็มสเกลหรือแรงดันสูงสุดที่สามารถเกิดขึ้นได้ในวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล ปกติมีค่าเท่ากับไฟเลี้ยง

V_{LSB} คือ ระยะห่างของระดับแรงดันที่ข้อมูล 1 บิตหรือค่าแรงดันที่ข้อมูลดิจิตอลเท่ากับ 1

ในวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล 3 บิตค่าแรงดัน V_{LSB} เท่ากับ 0.625 V ถ้าหากจำนวนบิตของวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลมีมากขึ้นค่าของแรงดัน V_{LSB} จะลดลงทำให้ความแม่นยำของการแปลงสัญญาณมีมากขึ้นและส่งผลให้ที่ข้อมูลดิจิตอลสูงสุดเมื่อ

เทียบกับแรงดันอะนาลอกทางอินพุตจะมีค่าใกล้เคียงกันมากขึ้น ยกตัวอย่าง วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล 8 บิตจะมีค่า V_{LSB} เท่ากับ

$$V_{LSB} = \frac{5}{2^8} \times 5 = \frac{5}{256} \times 5 = 0.0195V$$

ดังนั้นที่ข้อมูลดิจิตอลสูงสุดคือ 11111111_2 หรือ FFh จะมีค่าเทียบกับแรงดันอะนาลอกเป็น 5-0.0195 = 4.9805 V ถ้าเพิ่มเป็น 10 บิต แรงดันอะนาลอกที่ข้อมูลดิจิตอลสูงสุดจะเป็น 4.9951 V

2.2.1 หลักการแปลงสัญญาณอะนาลอกเป็นดิจิตอล

การแบ่งกลุ่มของ ADC ตามกลไกการทำงานของภาคคอนเวอร์เตอร์ซึ่งเป็นองค์ประกอบหลักของตัวแปลงสัญญาณและเป็นตัวกำหนดคุณสมบัติที่สำคัญของ ADC สามารถแบ่งออกได้หลายประเภทโดยหลักๆที่นิยมใช้งานกันอยู่ในปัจจุบันนี้มี 5 ชนิดด้วยกันคือ

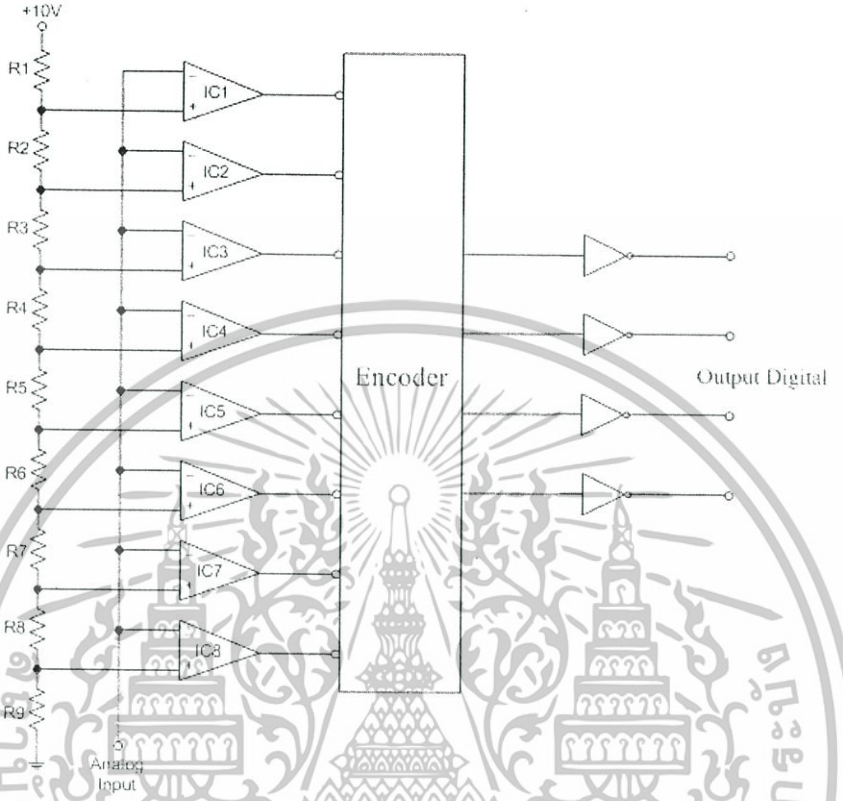
1. สโลปหรืออินทิเกรชันคอนเวอร์เตอร์ (Slope or Integration Converter)
2. ซัคเซสซีฟแอพพร็อกซิเมชันรีจีสเตอร์ (SAR: Successive Approximation Register)
3. แฟลชคอนเวอร์เตอร์ (Flash Converter)
4. ไปป์ไลน์คอนเวอร์เตอร์ (Pipeline Converter)
5. เดลต้า-ซิกม่าคอนเวอร์เตอร์ (Delta-Sigma Converter)

ซึ่ง ADC แต่ละชนิดก็จะมีหลักการทำงานที่แตกต่างกันออกไปแต่ที่จะกล่าวในที่นี้จะเป็นหลักการทำงานของ ADC ชนิด แฟลชคอนเวอร์เตอร์ (Flash Converter) ซึ่งเป็นแบบที่ถูกนำมาใช้งานอย่างยาวนานและยังคงเป็นที่นิยมใช้ในปัจจุบันอยู่มากเนื่องจากเป็น ADC ที่มีความเร็วในการทำงานสูงที่สุดใน 5 ชนิดที่กล่าวมา

การแปลงสัญญาณของ ADC ชนิดนี้จะใช้หลักการเปรียบเทียบแรงดัน (Voltage Comparator) โดยมีหลักการเบื้องต้นคือ ใช้การเปรียบเทียบแรงดันระหว่างอินพุตกับแรงดันอ้างอิง จากนั้นวงจรเปรียบเทียบแรงดันจะส่งสัญญาณสูงและต่ำซึ่งแทนผลการเปรียบเทียบเข้าสู่วงจรเข้ารหัสเพื่อกำหนดข้อมูลดิจิตอลทางเอาต์พุตต่อไป ดังแสดงในรูปที่ 2.9

วงจรในรูปที่ 2.9 นี้คือวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลขนาด 4 บิตโดยแรงดันอินพุตมีค่าสูงสุดไม่เกิน 10 V ตัวต้านทาน R1-R9 ใช้กำหนดค่าของแรงดันอ้างอิงให้แก่ออปแอมป์ IC1-IC8 ซึ่งทำหน้าที่เปรียบเทียบแรงดันอินพุตกับแรงดันอ้างอิง หากแรงดันอินพุตมีค่ามากกว่าแรงดันอ้างอิงที่จุดใด ออปแอมป์ตัวนั้นจะทำงานให้เอาท์พุตเป็นแรงดันต่ำส่งไปยังวงจรเข้ารหัสในทางตรงกันข้ามหากค่าของแรงดันอินพุตน้อยกว่าแรงดันอ้างอิง ออปแอมป์จะไม่ทำงาน

ยกตัวอย่าง ถ้าแรงดันอินพุตมีค่า 5V จะทำให้ IC4 ทำงานเกิดสัญญาณแรงดันต่ำป้อนไปยังวงจรเข้ารหัสซึ่งใช้ไอซีเบอร์ 74147 ได้ข้อมูลเอาท์พุตเป็น 1011 (เรียงลำดับจากเอาท์พุต DCBA) ซึ่งต้องกลับสถานะลอจิกด้วยนอตเกตเป็น 0100 ก็จะได้ค่าข้อมูลดิจิตอลตามต้องการ



รูปที่ 2.9 วงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอลแบบเปรียบเทียบแรงดัน

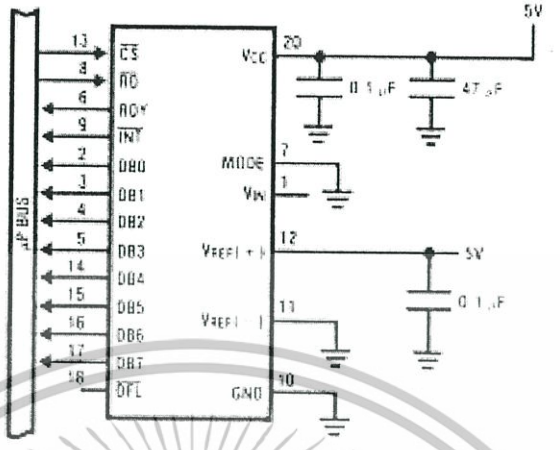
2.2.2 ประเภทของตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลตามการเชื่อมต่อ

ในปัจจุบันมีผู้ผลิตชิปไอซีหลายแห่งได้ผลิตไอซีเพื่อทำการแปลงสัญญาณอะนาลอกเป็นดิจิตอล(Analog to digital converter: ADC) ออกมาสู่ตลาดเป็นจำนวนมาก โดยผู้ผลิตแต่ละรายก็จะมีโปรโตคอล (protocol) เพื่อให้หน่วยประมวลผลกลาง (Central Processing Unit: CPU) ใช้เป็นแบบแผนในการอ่านข้อมูลจากไอซีที่แตกต่างกันออกไป ดังนั้น นอกจากการแบ่งกลุ่มของ ADC ตามกลไกการทำงานของภาคคอนเวอร์เตอร์ดังที่ได้กล่าวไปแล้วนั้นยังสามารถแบ่งชนิดตามโปรโตคอลเชื่อมต่อได้อีกด้วยซึ่งโปรโตคอลที่นิยมนำมาใช้กับ ADC ในปัจจุบันประกอบด้วย 4 โปรโตคอลหลักๆ คือ โปรโตคอลแบบขนาน โปรโตคอลแบบ SPI โปรโตคอลแบบ I^2C และโปรโตคอลแบบ 1-Wire

2.2.2.1 ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลด้วยโปรโตคอลแบบขนาน

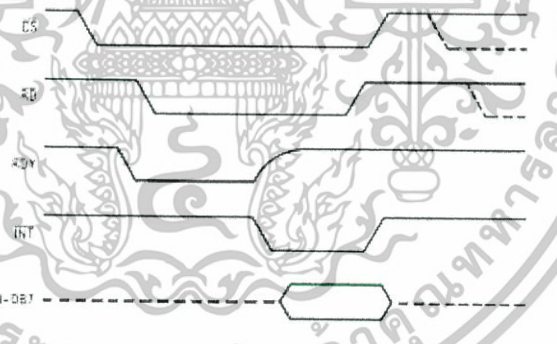
เป็นรูปแบบดั้งเดิมของการเชื่อมต่อตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลและในปัจจุบันก็ยังคงได้รับความนิยมใช้งานกันอย่างกว้างขวางเนื่องจากเป็นรูปแบบที่มีความเร็วในการทำงานสูงกว่าทุกโปรโตคอลแต่การเชื่อมต่อแบบนี้ไม่เหมาะกับงานที่ต้องการพื้นที่การใช้งานน้อยเนื่องจากโปรโตคอลชนิดนี้จำเป็นต้องใช้สายสัญญาณควบคุมจำนวนมากในการเชื่อมต่อ ดัง

แสดงตัวอย่างในรูปที่ 2.10 เป็นตัวอย่างการเชื่อมต่อกับไอซี ADC เบอร์ADC0820 ซึ่งใช้เป็นตัวอย่างในการออกแบบนี้



รูปที่ 2.10 การเชื่อมต่อ ADC ที่ใช้โปรโตคอลแบบขนาน

จะเห็นได้ว่าจำเป็นต้องใช้สายสัญญาณในการเชื่อมต่อมากถึง 12 เส้นเพื่อให้ ADC สามารถทำงานได้ โดยแบ่งเป็นสายสัญญาณข้อมูลหรือบัสข้อมูล 8 เส้นและสายสัญญาณควบคุม 4 เส้น รูปที่ 2.11 แสดงกราฟลำดับการทำงานในการอ่านข้อมูลจาก ADC



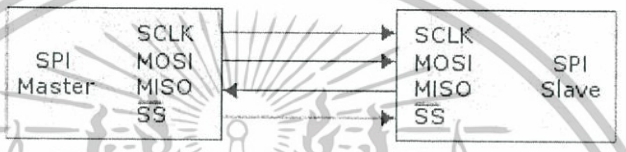
รูปที่ 2.11 ลำดับการทำงานในการอ่านข้อมูลจาก ADC ที่ใช้โปรโตคอลแบบขนาน

ลำดับในการทำงานเริ่มจากหน่วยประมวลผลกลางส่งสัญญาณ $\overline{CS}$ และ $\overline{RD}$ เพื่อสั่งให้ ADC เริ่มทำการแปลงข้อมูลซึ่งในขณะที่กำลังแปลงอยู่นั้น ADC จะทำให้ขา RDY เป็น “0” ซึ่งเป็นสถานะที่บอกหน่วยประมวลผลกลางว่าขณะนี้กำลังทำการแปลงอยู่ จนกระทั่งการแปลงเสร็จสิ้นลง ADC จะส่งสัญญาณมาบอกหน่วยประมวลผลกลางโดยทำให้ RDY เป็น “1” และขณะเดียวกันก็ทำให้ขา $\overline{INT}$ เป็น “0” ด้วย ซึ่งหน่วยประมวลผลกลางจะใช้การเปลี่ยนแปลงของขาใดขาหนึ่งเพื่อรับทราบและทำการอ่านข้อมูลออกไป

จากที่ได้กล่าวไปนี้จะเห็นว่าการอ่านข้อมูลจาก ADC ในแต่ละครั้งค่อนข้างจะเรียบง่าย เพราะใช้ขั้นตอนน้อยมาก ดังนั้น ADC ที่ใช้โปรโตคอลนี้จึงเหมาะสำหรับระบบที่ต้องการความเร็วในการทำงานสูง เช่น เครื่องมือวัดความถี่สูง เป็นต้น

2.2.2.2 ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลด้วยโปรโตคอลแบบ SPI

SPI (Serial Peripheral Interface Bus) เป็นการสื่อสารข้อมูลแบบอนุกรมซึ่งถูกคิดค้นขึ้นโดยบริษัทโมโตโรลาซึ่งสามารถทำงานในโหมดฟูลดูเพล็ก (full duplex) หรือทำงานได้ทั้งรับและส่งข้อมูลในเวลาเดียวกัน การสื่อสารจะเป็นลักษณะการแลกเปลี่ยนข้อมูลแบบมาสเตอร์และสเลฟ (master/slave mode) โดยอุปกรณ์มาสเตอร์จะทำหน้าที่เริ่มต้นเฟรมข้อมูลและระบุการเชื่อมต่อกับอุปกรณ์สเลฟที่ต้องการรับ-ส่งข้อมูลโดยใช้สัญญาณควบคุม Chip Select นั่นก็หมายความว่าอุปกรณ์มาสเตอร์หนึ่งตัวสามารถเชื่อมต่อกับอุปกรณ์สเลฟได้หลายตัวในวงจรเดียวกันนั่นเอง ในบางครั้งโปรโตคอลแบบ SPI สามารถเรียกได้อีกชื่อหนึ่งว่า "บัสอนุกรม 4 สาย" (four wire serial bus) เนื่องจากมีส่วนประกอบของสัญญาณควบคุมจำนวน 4 เส้นดังแสดงในรูป 2.12



รูปที่ 2.12 ระบบ SPI พื้นฐาน

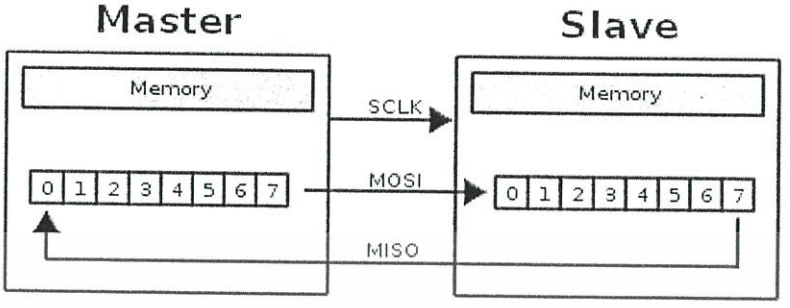
เมื่อเริ่มต้นการสื่อสารอุปกรณ์มาสเตอร์ต้องจัดเตรียมสัญญาณนาฬิกาเพื่อใช้กับอุปกรณ์สเลฟโดยความถี่ที่ใช้ต้องไม่มากกว่าหรือน้อยกว่าความถี่ที่อุปกรณ์สเลฟจะสามารถทำงานได้โดยปกติจะอยู่ในช่วง 1-70 เมกะเฮิร์ต (1-70 MHz) จากนั้นอุปกรณ์มาสเตอร์ส่งสัญญาณควบคุม $\overline{SS}$ เพื่อระบุอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วยและรอในช่วงระยะเวลาหนึ่งเพื่อให้อุปกรณ์สเลฟอยู่ในสถานะพร้อมทำงานหรือในกรณีของ ADC คือช่วงเวลาที่ ADC ใช้ในการแปลงข้อมูลจากนั้นจึงส่งสัญญาณนาฬิกาออกไป

สัญญาณนาฬิกาแต่ละลูกจะทำให้เกิดการรับ-ส่งข้อมูลระหว่างอุปกรณ์มาสเตอร์และอุปกรณ์สเลฟขึ้นโดยแบ่งเป็น 2 รูปแบบคือ

- อุปกรณ์มาสเตอร์ส่งบิตข้อมูลไปบนสายสัญญาณ MOSI และอุปกรณ์สเลฟอ่านค่านี้จากสายสัญญาณเดียวกัน
- อุปกรณ์สเลฟส่งบิตข้อมูลไปบนสาย MISO และอุปกรณ์มาสเตอร์อ่านค่านี้จากสายสัญญาณเดียวกัน

โดยปกติในการสื่อสารจะประกอบไปด้วยชิฟต์รีจีสเตอร์ 2 ตัวทำการแลกเปลี่ยนข้อมูลกันโดยตัวแรกอยู่ที่อุปกรณ์มาสเตอร์และอีกตัวอยู่ที่อุปกรณ์สเลฟทำการเชื่อมต่อกันเป็นวงรอบดังรูปที่ 2.13 โดยบิตข้อมูลที่มีนัยสำคัญสูงสุด (MSB) จะถูกเลื่อนออกไปก่อนขณะเดียวกันก็จะนำข้อมูลใหม่ใส่เข้าไปยังบิตที่มีนัยสำคัญต่ำสุด (LSB) แล้วเลื่อนบิต MSB ออกไปอีกทำอย่างนี้จนครบทุกบิตที่ต้องการส่ง จากนั้นอุปกรณ์แต่ละตัวจึงนำเอาข้อมูลที่ได้นี้ออกไปใช้งานและหากว่ามี

ข้อมูลที่ต้องการส่งอีกชิฟต์รีจีสเตอร์ก็จะทำการโหลดข้อมูลชุดใหม่และเริ่มกระบวนการดังกล่าวอีกครั้ง

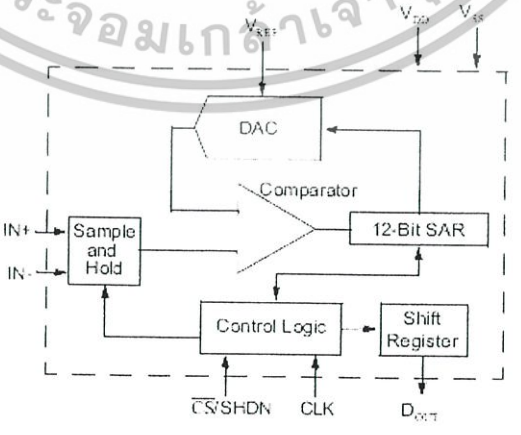


รูป 2.13 การรับส่งข้อมูลของโปรโตคอล SPI

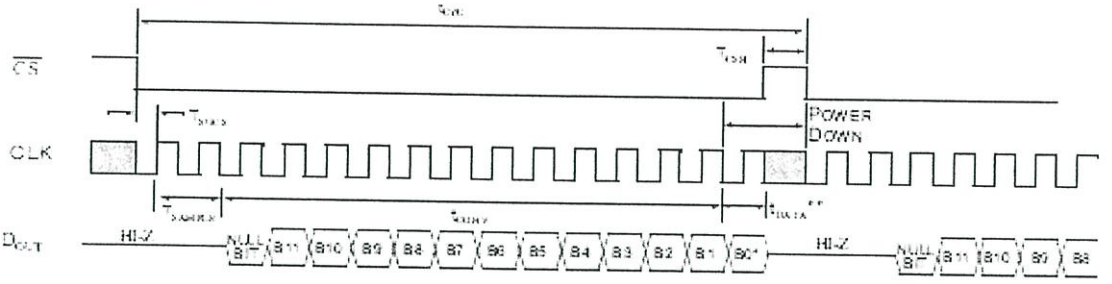
การรับ-ส่งข้อมูลแต่ละครั้งจะประกอบไปด้วยสัญญาณนาฬิกาหลายลูกซึ่งจะทำหน้าที่ให้จังหวะในการสื่อสารและเมื่อการรับ-ส่งข้อมูลสิ้นสุดลงอุปกรณ์มาสเตอร์จะหยุดสัญญาณนาฬิกาพร้อมๆกับทำให้สัญญาณ $\overline{SS}$ หยุดลง

สำหรับอุปกรณ์สเลฟที่ไม่อยู่ในสถานะทำงานหรือไม่มีสัญญาณ $\overline{SS}$ จะต้องไม่สนใจกับสัญญาณนาฬิกาและ MOSI ที่เกิดขึ้น ในขณะเดียวกันก็จะต้องไม่ส่งสัญญาณ MISO ออกมาด้วย สำหรับอุปกรณ์มาสเตอร์ต้องเลือกที่จะติดต่อกับอุปกรณ์สเลฟได้ทีละหนึ่งตัวเท่านั้น

ในการออกแบบนี้ได้นำไอซี ADC เบอร์ MCP3201 มาใช้ในการออกแบบวงจรเพื่อเป็นตัวอย่างสำหรับการสื่อสารด้วยโปรโตคอลแบบ SPI ซึ่ง ADC ตัวนี้เป็น ADC ที่ทำการแปลงข้อมูลด้วยเทคนิค SAR (Successive Approximation Register) ขนาด 12 บิต มีความเร็วในการแปลงข้อมูล 100 ksps ทำงานที่ความเร็วสัญญาณนาฬิกาสูงสุด 1.6 MHz มีโครงสร้างดังรูปที่ 2.14

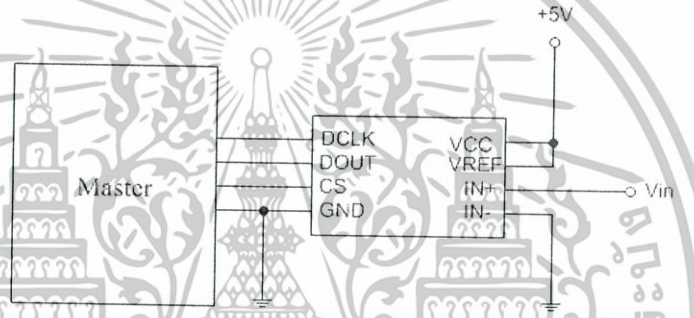


รูปที่ 2.14 ไดอะแกรมภายในของ MCP3201



รูปที่ 2.15 ลำดับการทำงานในการอ่านข้อมูลจาก ADC ที่ใช้โปรโตคอลแบบ SPI

จากรูปที่ 2.15 เป็นการแสดงลำดับการทำงานในระดับสัญญาณนาฬิกาและรูปที่ 2.16 แสดงวงจรการเชื่อมต่อระหว่างอุปกรณ์มาสเตอร์กับ MCP3201 ซึ่งจะประกอบด้วยขาสัญญาณที่ถูกใช้งานจำนวน 3 เส้นคือ $\overline{CS}$, CLK และ D_{out}



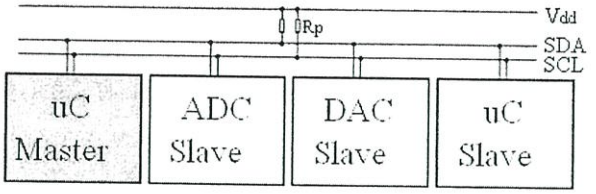
รูปที่ 2.16 วงจรเชื่อมต่ออุปกรณ์มาสเตอร์กับ MCP3201

โดยการทำงานเริ่มจากอุปกรณ์มาสเตอร์ทำให้สัญญาณ CS เป็น “0” เพื่อเป็นการบอกให้ไอซีเริ่มแปลงสัญญาณ จากนั้นอุปกรณ์มาสเตอร์จะรอตรวจสอบสัญญาณที่ขา DOUT เนื่องจากเมื่อไอซีแปลงสัญญาณเสร็จและพร้อมที่ส่งข้อมูลออกมาจะทำให้สัญญาณที่ขานี้เป็น “0” ดังนั้นเมื่ออุปกรณ์มาสเตอร์ได้รับสัญญาณนี้แล้ว จะเริ่มอ่านข้อมูลในรอบสัญญาณนาฬิกาถัดไป โดยเริ่มจาก MSB ไปจนกระทั่ง LSB และในช่วงเวลาของการอ่านบิตสุดท้ายหรือบิต LSB นั้น อุปกรณ์มาสเตอร์จะต้องทำให้ CS เป็น”1”อีกครั้งเพื่อบอกให้ไอซีหยุดการส่งข้อมูลซึ่งจะถือว่าสิ้นสุดการอ่านข้อมูลในหนึ่งรอบการทำงาน

2.2.2.3 ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลด้วยโปรโตคอลแบบ I^2C

โปรโตคอล I^2C (Inter-Integrated Circuit) หมายถึงการติดต่อสื่อสารระหว่างไอซีหลายตัว โดยบัส I^2C ได้รับการพัฒนาจากบริษัทฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือต้องการให้ไอซีหรือโมดูลสามารถติดต่อสั่งงานและควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นแรกคือสายข้อมูล อีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะในการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I^2C ทำได้ง่ายมากเพียงต่อสายสัญญาณข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไป ส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่อ

อุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว



รูปที่ 2.17 การเชื่อมต่อของอุปกรณ์ต่างๆบนระบบบัส I^2C

สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรมหรือ SDA (Serial Data Line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรมหรือ SCL (Serial Clock Line) ในการอธิบายต่อไปนี้จะเรียกสายสัญญาณทั้งสองว่า สาย SDA และสาย SCL

สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi-directional line) ต้องมีการต่อตัวต้านทานพูลอัพกับแรงดัน 5 โวลต์ไว้ตลอดเวลาเพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีปะปนเข้ามาในสายสัญญาณทั้งสอง วงจรเอาต์พุตที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (Open-Drain) หรือคอลเล็กเตอร์เปิด (Open Collector)

อัตราการถ่ายทอดข้อมูลบนบัส I^2C มีความเร็วประมาณ 100 kbps ในโหมดปกติ (Normal Mode), 400 kbps ในโหมดความเร็ว (Fast Mode), 1 Mbps ใน Fast Mode Plus และ3.4 Mbps ใน High Speed Mode อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400 pf การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่า คือ 7 บิต (7-bit addressing) หรือ 10 บิต (10-bit addressing)

ข้อเด่นอีกประการหนึ่งของบัส I^2C คือสามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้สามารถติดต่อสื่อสารกันได้ โดยอุปกรณ์บนบัส I^2C ตัวหนึ่งอาจใช้ไฟเลี้ยง 5 โวลต์ ในขณะที่อีกตัวหนึ่งใช้ไฟเลี้ยง 12 โวลต์ การต่อร่วมกันบนบัส I^2C สามารถกระทำได้ในลักษณะเดียวกับกรณีที่อุปกรณ์ทั้งสองใช้ไฟเลี้ยงเท่ากัน กล่าวคือ ให้ต่อสาย SDA และ SCL ของอุปกรณ์แต่ละตัวเข้าด้วยกันและต้องต่อตัวต้านทานพูลอัพ (Rp) เข้ากับแรงดัน 5 โวลต์ไว้ด้วยเสมอ ดังรูป 2.17

ดังที่กล่าวไปแล้วว่าบัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้นคือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัสหรือเรียกว่าโปรโตคอล (Protocol) เพื่อให้ผู้ใช้งานทราบว่าขณะนี้อุปกรณ์ใดติดต่อกันอยู่ และอุปกรณ์ใดเป็นตัวรับหรือตัวส่ง ต่อไปนี้จะขออธิบายลักษณะหน้าที่และนิยามของอุปกรณ์ที่ต่ออยู่บนบัส I^2C เพื่อเป็นข้อตกลงพื้นฐานก่อนที่จะอธิบายการทำงานของบัส I^2C ต่อไป

- อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูลเรียกว่า ตัวส่ง (Transmitter)
- อุปกรณ์ที่เป็นตัวรับข้อมูลเรียกว่า ตัวรับ (Receiver)
- อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการติดต่อบนบัส I^2C เรียกว่า มาสเตอร์ (Master)

- อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (Slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ ประการแรกการถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้นและประการที่สองในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามเมื่อสาย SCL มีสถานะเป็นลอจิกสูง สาย SDA ต้องรักษาข้อมูลไว้ไม่ให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

สภาวะที่เกิดขึ้นบนบัส I^2C มีด้วยกัน 5 สภาวะ ดังนี้

i. บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้งคู่ นั่นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้

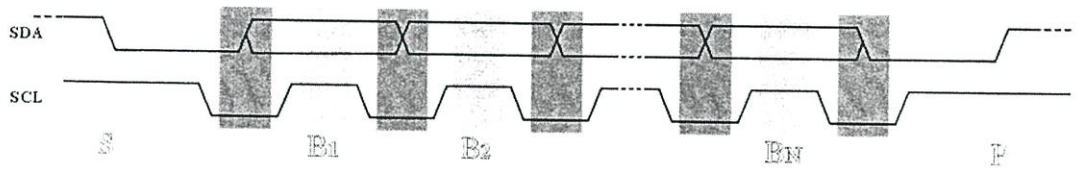
ii. เริ่มต้นการถ่ายทอดข้อมูล (Start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะเป็นลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่าสภาวะเริ่มต้น (Start)

iii. หยุดการถ่ายทอดข้อมูล (Stop data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูง ในขณะที่สาย SCL มีสถานะเป็นลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่าสภาวะหยุด (Stop)

iv. ข้อมูลดำรงอยู่บนบัส (data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้นโดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่าเป็น "0" หรือ "1" ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูงเพราะหากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้นอุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น

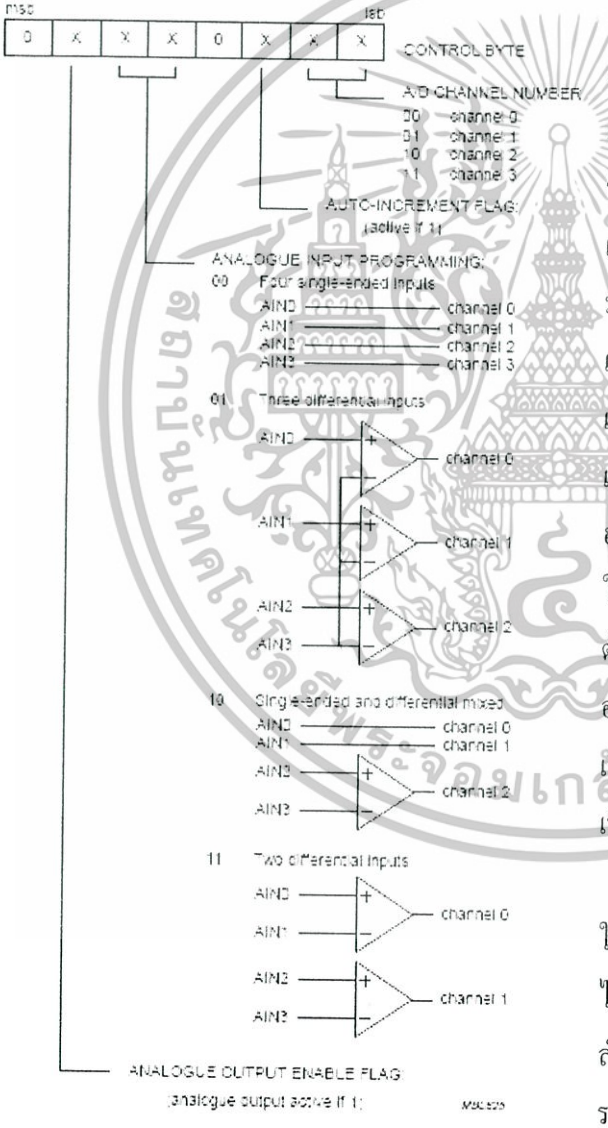
v. รับรู้ข้อมูล (Acknowledge) เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่าบิตรับรู้ (Acknowledge bit) มีสถานะเป็นลอจิกสูงหลังจากส่งข้อมูลมาครบแล้ว ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกาเพื่อตอบสนองบิตรับรู้ที่ส่งมาจากตัวส่งทางด้านตัวรับ จะส่งบิตรับรู้ที่มีสถานะลอจิกต่ำลงบนบัส อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้เพื่อตอบสนองให้ทราบว่าได้รับข้อมูลในแต่ละไบต์เรียบร้อยแล้ว

รูปที่ 2.18 แสดงไดอะแกรมเวลาที่แสดงถึงการเกิดสภาวะต่างๆบนบัส I^2C ไม่ว่าจะเป็นสภาวะบัสว่าง, เริ่มต้น, ถ่ายทอดข้อมูล และหยุดการถ่ายทอดข้อมูล



รูปที่ 2.18 ไดอะแกรมเวลาต่างๆในบัส I^2C

ในการออกแบบของวิทยานิพนธ์ฉบับนี้ได้ใช้ไอซี ADC เบอร์ PCF8591 เป็นตัวอย่างในการติดต่อสื่อสารด้วยโปรโตคอล I^2C ซึ่งเป็น ADC ที่ใช้เทคนิคการแปลงข้อมูลแบบ SAR (Successive Approximation) ขนาด 8 บิต 4 ช่องสัญญาณและสามารถต่อพ่วงกันได้สูงสุด 8 ตัวทำให้ได้วงจร ADC รวมสูงสุดถึง 32 ช่อง



รูปที่ 2.19 รายละเอียดข้อมูลควบคุมในไอซี PCF8591

ในระบบบัส I^2C การติดต่อกับอุปกรณ์แต่ละตัวต้องระบุแอดเดรสของอุปกรณ์เหล่านั้นอย่างชัดเจนถ้าเป็นการอ้างถึงแบบ 7 บิต ข้อมูลกำหนดแอดเดรส 4 บิตบนจะเป็นค่าแอดเดรสเฉพาะของอุปกรณ์ตัวนั้นๆที่กำหนดมาจากบริษัทผู้ผลิต ผู้ใช้งานไม่สามารถเปลี่ยนแปลงได้ สำหรับไอซี PCF8591 จะมีค่าเท่ากับ 1001_2 ข้อมูล 3 บิตถัดมาจะเป็นค่าของแอดเดรสที่ผู้ใช้งานสามารถกำหนดได้ทางฮาร์ดแวร์เพื่อเลือกไอซีตัวที่ต้องการติดต่อด้วยในกรณีที่มีการต่อใช้งาน PCF8591 มากกว่า 1 ตัว ส่วนบิต LSB ใช้ในการกำหนดว่าต้องการอ่านหรือเขียนข้อมูลกับไอซีตัวนั้นๆโดยหากเป็น “0” หมายถึงต้องการเขียนข้อมูลและหากเป็น “1” หมายถึง ต้องการอ่านข้อมูล

หลังจากส่งข้อมูลกำหนดแอดเดรสให้แก่ PCF8591 แล้วต้องส่งข้อมูลควบคุมตามไปด้วยเพื่อกำหนดคุณสมบัติของวงจรแปลงสัญญาณอะนาลอกเป็นดิจิตอล โดยมีรายละเอียดของข้อมูลในแต่ละบิตดังในรูปที่ 2.19

บิต 6 ของข้อมูลควบคุมใช้สำหรับการอีนาเบิลขาอะนาลอกเอาท์พุต เมื่อต้องการอีนาเบิลต้องกำหนดขานี้เป็น “1”

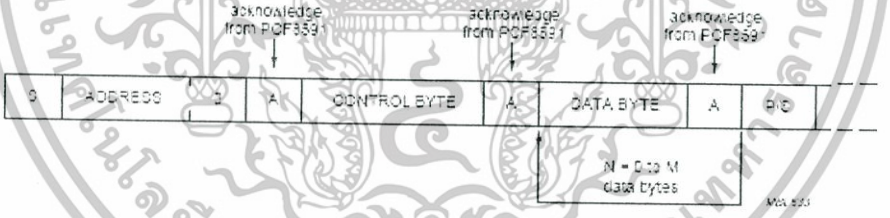
บิต 4 และ บิต 5 ของข้อมูลควบคุมใช้สำหรับกำหนดรูปแบบของสัญญาณอะนาลอกอินพุตที่ป้อนให้ PCF8591

บิต 2 ใช้สำหรับเลือกรูปแบบการอ่านข้อมูลจากขาอินพุตอะนาลอกว่าจะเป็นการอ่านจากเพียงอินพุตเดียวหรืออ่านแบบเรียงลำดับทุกอินพุต ถ้าต้องการเลือกให้อ่านแบบเรียงลำดับต้องกำหนดบิตนี้ให้เป็น 1

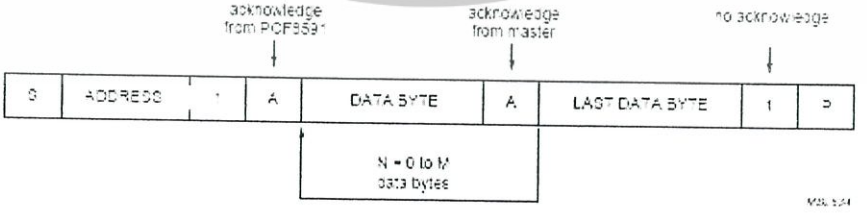
บิต 0 และ บิต 1 ใช้สำหรับกำหนดช่องของอินพุตอะนาลอกที่ต้องการอ่าน ถ้ากำหนดให้บิต 2 เป็น “1” หลังจากอ่านค่าของบิต 0 และบิต 1 แล้วในการอ่านค่าครั้งต่อไปจะเป็นการอ่านค่าอินพุตจากช่องที่ 1

ข้อมูลควบคุมทั้งหมดจะถูกเก็บไว้ในรีจีสเตอร์ควบคุมภายใน PCF8591 สำหรับการจ่ายไฟให้กับ PCF8591 ครั้งแรกบิตต่างๆของข้อมูลภายในรีจีสเตอร์ควบคุมจะเป็น “0”

รูปที่ 2.20 และ 2.21 แสดงการไดอะแกรมการเขียนและอ่านข้อมูลกับ PCF8591 ตามลำดับซึ่งเป็นที่น่าสังเกตว่าก่อนการอ่านหรือเขียนข้อมูลจะต้องเข้าสู่สภาวะเริ่มต้นและระบุแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อด้วยเสมอถึงแม้ว่าในบัสนั้นจะมีอุปกรณ์สเลฟหรือในที่นี้คือ PCF8591 เพียงตัวเดียวก็ตาม

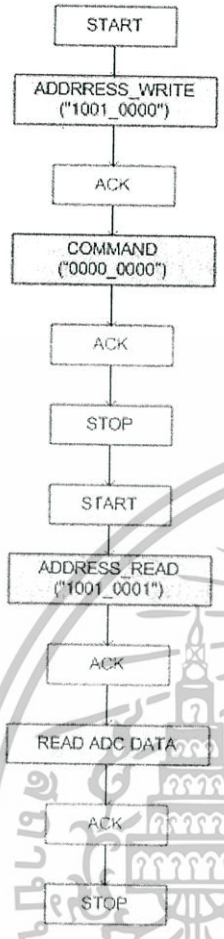


รูปที่ 2.20 โหมดการเขียนข้อมูลไปยัง PCF8591



รูปที่ 2.21 โหมดการอ่านข้อมูลจาก PCF8591

รูปที่ 2.23 แสดงขั้นตอนในการสื่อสารกับ PCF8591 เพื่อให้ได้มาซึ่งข้อมูลการแปลงสัญญาณอะนาลอกเป็นดิจิตอล 1 ค่าโดยเริ่มจากอุปกรณ์มาสเตอร์ส่งสัญญาณเพื่อเริ่มบัสข้อมูลและส่งแอดเดรสไปยังบัสเพื่อระบุแอดเดรสของอุปกรณ์ที่ต้องการติดต่อด้วย เมื่ออุปกรณ์สเลฟ

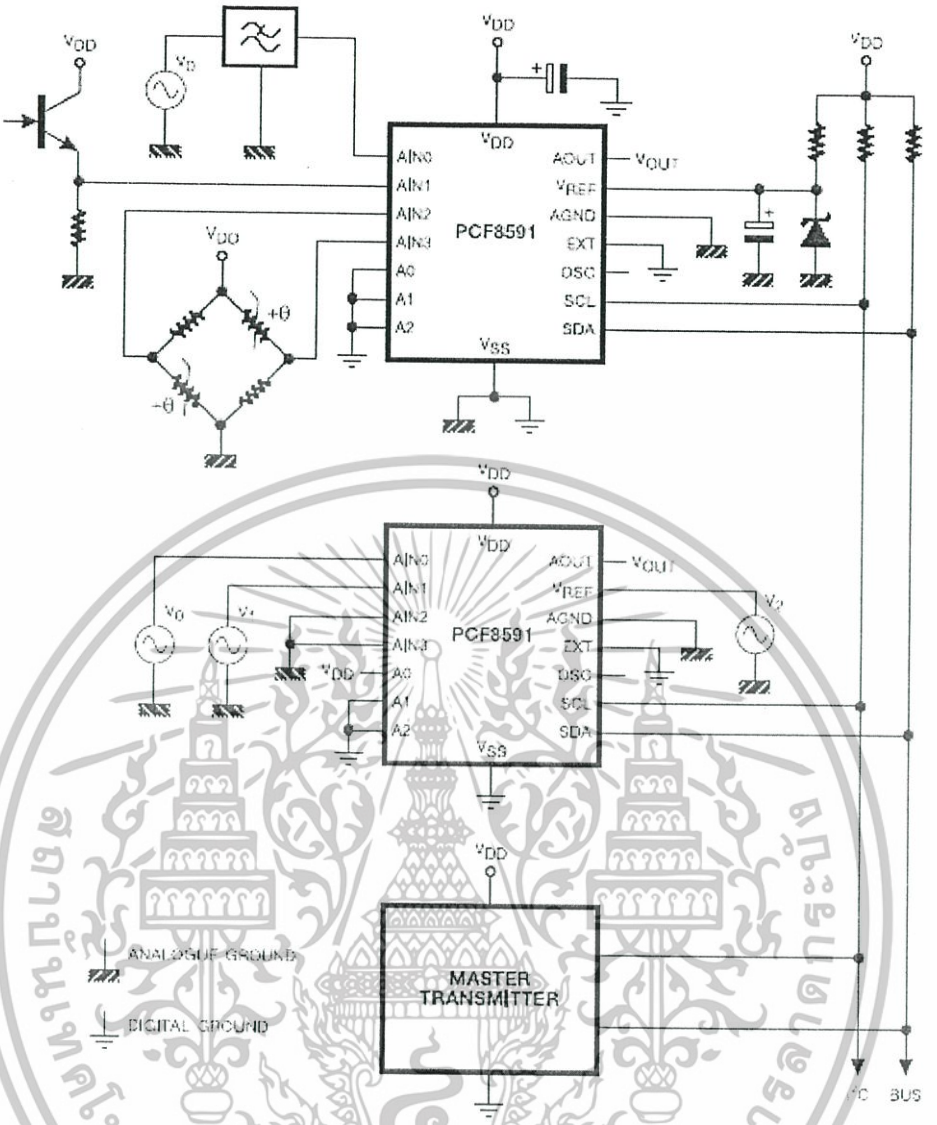


ได้รับข้อมูลแล้วและตรวจสอบได้ว่าเป็นแอดเดรสของตน อุปกรณ์สเลฟจะส่งสัญญาณ ACK กลับออกมา จากนั้นอุปกรณ์มาสเตอร์จะส่งคำสั่งเพื่อให้ PCF8591 นำไปตีความว่าต้องการให้ทำอะไรซึ่งในที่นี้คือ “0000 0000” หมายถึง ต้องการอ่านข้อมูลจาก PCF8591 ในช่องที่ 0 และมีอินพุตแบบ Single-end เมื่อ PCF8591 ได้รับข้อมูลถูกต้องจะส่งสัญญาณ ACK กลับมาถัดมาอุปกรณ์มาสเตอร์ส่งสภาวะหยุด

จากนั้นอุปกรณ์มาสเตอร์จะส่งสภาวะเริ่มต้นบัสออกมาอีกครั้งตามด้วยการระบุแอดเดรสและรอการตอบรับจาก PCF8591 เสร็จแล้วส่งคำสั่งว่าต้องการอ่านข้อมูลจาก PCF8591 และรอการตอบรับ เมื่อได้รับการตอบรับแล้วจะเริ่มอ่านข้อมูลจากบัสทีละบิตตั้งแต่ MSB จนกระทั่ง LSB เมื่ออ่านจนครบอุปกรณ์มาสเตอร์ต้องส่งสัญญาณตอบรับให้อุปกรณ์สเลฟรับรู้ว่าได้รับข้อมูลแล้วและสุดท้ายอุปกรณ์มาสเตอร์ส่งสภาวะหยุด ซึ่งถือเป็นการสิ้นสุดการแปลงข้อมูล 1 ค่า

รูปที่ 2.23 ขั้นตอนในการสื่อสารกับ PCF8591

ไดอะแกรมของการเชื่อมต่อเพื่อประยุกต์ใช้งานไอซี PCF8591 แสดงดังรูปที่ 2.24 ซึ่งโดยปกติแล้วไอซีเบอร์นี้ในอินพุตใดๆที่ไม่ได้ต่อใช้งานจะถูกต่อไว้กับ V_{SS} หรือ V_{DD} และถ้าเป็นอะนาลอกอินพุตแล้วขาที่ไม่ต่อใช้งานจะถูกต่อเข้ากับ AGND หรือ V_{REF} การที่ต้องต่อเช่นนี้ก็เพื่อเป็นการป้องกันสัญญาณรบกวนที่อาจแทรกซ้อนเข้ามาทางกราวด์หรือไฟเลี้ยงและเพื่อลดการรบกวนข้ามช่องของสัญญาณระหว่างส่วนของสัญญาณดิจิตอลไปยังส่วนของสัญญาณอะนาลอก นอกจากนั้นในส่วนของตัวเก็บประจุดีคับปลิ้งที่ต่ออยู่กับขาไฟเลี้ยงกับกราวด์และขารับแรงดันอ้างอิงกับกราวด์จะต้องมีค่าความจุไม่ต่ำกว่า 10 ไมโครฟารัดจึงจะเหมาะสม



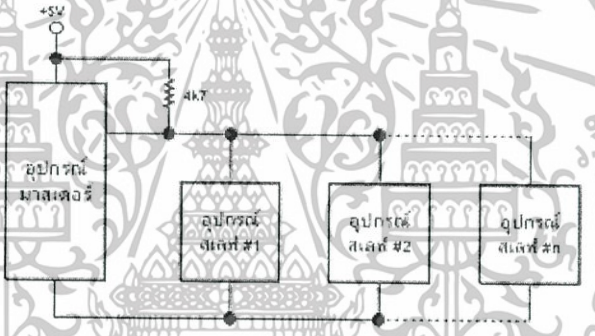
รูปที่ 2.24 วงจรการใช้งาน PCF8591

2.2.2.4 ตัวแปลงสัญญาณอะนาลอกเป็นดิจิตอลด้วยโปรโตคอลแบบ 1-Wire

ระบบสื่อสารข้อมูลอนุกรมแบบ 1 สาย (1-Wire Serial Bus) เป็นระบบการสื่อสารที่คิดค้นขึ้นมาโดยบริษัท ดัลลัส เซมิคอนดักเตอร์ ในบางครั้งจึงเรียกอีกชื่อหนึ่งว่า ระบบสื่อสาร 1 สายแบบดัลลัส (Dallas 1-Wire bus) ระบบสื่อสารข้อมูลแบบนี้เป็นระบบที่มีความชาญฉลาดเนื่องจากใช้สายสัญญาณเพียง 1 เส้นเท่านั้น โดยไม่ต้องมีสายสัญญาณนาฬิกามาควบคุมจังหวะการถ่ายทอดข้อมูลเหมือนกับระบบสื่อสารข้อมูลอนุกรมในแบบอื่นๆเนื่องจากสายข้อมูลจะทำหน้าที่เสมือนหนึ่งเป็นสายสัญญาณนาฬิกาในตัวค่าของข้อมูลจะพิจารณาจากลักษณะของรูปสัญญาณที่ปรากฏบนสายสัญญาณในแต่ละช่องของเวลาที่เรียกว่า Time Slot โดยคาบเวลาต่ำสุดหรือสูงสุดของสถานะต่างๆสำหรับการสื่อสารข้อมูลในแต่ละ Time Slot นั้นมีการกำหนดขอบเขตไว้ชัดเจนการถ่ายทอดข้อมูลจะเกิดขึ้นในแต่ละ Time Slot นั้นรูปแบบการถ่ายทอดข้อมูลจะเป็นแบบอะซิงโครนัสในระดับบิตโดยไม่มีการกำหนดความยาวของข้อมูลเป็นระดับไบต์

สายสัญญาณบนระบบบัสชนิดนี้จะเป็นสายสัญญาณแบบ 2 ทิศทางแต่ข้อมูลจะสามารถเดินทางได้ในทิศทางเดียวในช่วงเวลาหนึ่งๆ นั่นคือ มีลักษณะคล้ายกับระบบสื่อสารแบบครึ่งดูเพล็ก (half-duplex) อุปกรณ์บนระบบบัสต้องมีการระบุอย่างชัดเจนว่าตัวใดเป็นอุปกรณ์มาสเตอร์ ตัวใดเป็นอุปกรณ์สเลฟ โดยอุปกรณ์มาสเตอร์จะเป็นตัวจัดเตรียมความพร้อมของสายสัญญาณและควบคุมการถ่ายทอดข้อมูลบนสายสัญญาณนั้น

ข้อมูลทั้งหมดไม่ว่าจะเป็นข้อมูลควบคุมหรือข้อมูลใช้งานจะถูกส่งลงบนสายสัญญาณที่มีอยู่เพียงเส้นเดียวนี้ทั้งหมด ในระหว่างการทำงานอุปกรณ์มาสเตอร์และสเลฟสามารถเป็นได้ทั้งตัวส่งและตัวรับขึ้นอยู่กับเงื่อนไขของการทำงานในขณะนั้น ยกตัวอย่าง ถ้าหากมีการเขียนข้อมูลจากอุปกรณ์มาสเตอร์ไปยังอุปกรณ์สเลฟ ตัวส่งคืออุปกรณ์มาสเตอร์ ตัวรับคืออุปกรณ์สเลฟ ในทางตรงข้าม หากเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ ตัวส่งจะกลายเป็นอุปกรณ์สเลฟและตัวรับจะเป็นอุปกรณ์มาสเตอร์ อย่างไรก็ตาม ในระบบบัส 1 ระบบต้องมีอุปกรณ์มาสเตอร์เพียงตัวเดียวเท่านั้น



รูปที่ 2.25 การเชื่อมต่อบนระบบบัส 1 สาย

สายสัญญาณบนระบบบัสนี้ต้องกำหนดสภาวะปกติไว้ที่ลอจิกสูง โดยการต่อตัวต้านทานค่าประมาณ 4.7 กิโลโอห์ม พูลอัพกับไฟเลี้ยง 5 โวลต์ ดังนั้นอุปกรณ์ที่นำมาต่อบนระบบบัสนี้จึงต้องออกแบบให้ภาคเอาท์พุตที่ต้องต่อกับสายสัญญาณมีลักษณะเป็นคอลเลคเตอร์เปิดหรือเดรนเปิดในรูปที่ 2.25 แสดงไดอะแกรมการเชื่อมต่ออุปกรณ์ในระบบบัส 1 สายเบื้องต้น

อุปกรณ์มาสเตอร์จะเป็นอุปกรณ์เพียงตัวเดียวบนระบบบัส 1 สายที่สามารถกำหนดการเริ่มใช้สายสัญญาณเริ่มใช้สายสัญญาณได้ โดยอุปกรณ์มาสเตอร์จะกำหนดจุดเริ่มต้นของ Time Slot ด้วยการทำให้สายสัญญาณเป็นลอจิกต่ำในช่วงเวลาหนึ่ง จากนั้นก็จะทำให้สายสัญญาณกลับมาเป็นลอจิกสูง ถ้าอุปกรณ์สเลฟต้องการส่งข้อมูลมายังอุปกรณ์มาสเตอร์ อุปกรณ์สเลฟจะเปลี่ยนมาเป็นตัวควบคุมสภาวะของสายสัญญาณแทน แต่ถ้าอุปกรณ์มาสเตอร์ต้องการส่งข้อมูลก็จะเป็นตัวควบคุมสภาวะของสายสัญญาณต่อไปจนเสร็จสิ้นกระบวนการ ฟังก์ชันการทำงานของ Time Slot ที่กำหนดโดยอุปกรณ์มาสเตอร์มีอยู่ 4 ฟังก์ชันคือ

- รีเซต (Reset) สำหรับการเริ่มต้นติดต่อกับอุปกรณ์สเลฟ
- อ่านข้อมูล (Read Data) เพื่ออ่านข้อมูลจากอุปกรณ์สเลฟ

- เขียนข้อมูล "1" (Write One) ไปยังอุปกรณ์สเลฟ
- เขียนข้อมูล "0" (Write Zero) ไปยังอุปกรณ์สเลฟ

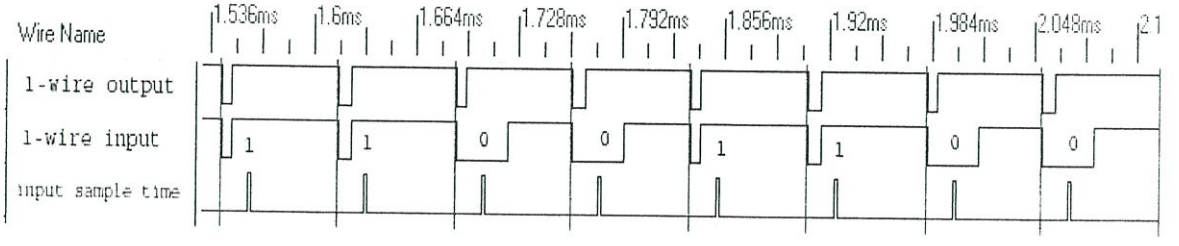
ทางด้านอุปกรณ์สเลฟมีฟังก์ชันในการทำงานของ Time Slot อยู่ 3 ฟังก์ชันประกอบด้วย

- ตอบสนอง (Presence) การติดต่อจากอุปกรณ์มาสเตอร์เพื่อแจ้งให้อุปกรณ์มาสเตอร์ทราบว่าขณะนี้สามารถติดต่อกันได้แล้ว
- เขียนข้อมูล "1" (Write One) ไปยังอุปกรณ์มาสเตอร์ซึ่งจะสัมพันธ์กับการอ่านข้อมูลของอุปกรณ์มาสเตอร์
- เขียนข้อมูล "0" (Write Zero) ไปยังอุปกรณ์มาสเตอร์ซึ่งจะสัมพันธ์กับการอ่านข้อมูลของอุปกรณ์มาสเตอร์

การแยกแยะฟังก์ชันของแต่ละ Time Slot จะใช้ความยาวของคาบเวลาและลักษณะของรูปสัญญาณเป็นตัวกำหนดและทุกครั้งที่มีการเปลี่ยนแปลงฟังก์ชันต้องทำให้สายสัญญาณอยู่ในสภาวะว่างเสมอซึ่งก็คือต้องทำให้สายสัญญาณเป็นลอจิกสูงเป็นเวลาอย่างน้อยหนึ่งไมโครวินาที

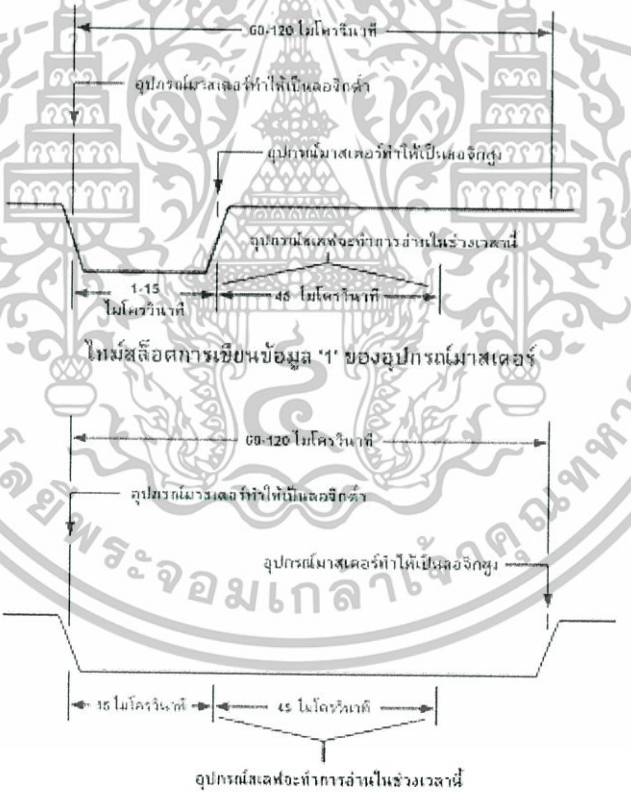
รูป 2.26 Time Slot การรีเซ็ทและการตอบรับของอุปกรณ์บนระบบบัส 1 สาย

รูปที่ 2.26 แสดง Time Slot ของการรีเซ็ทจากอุปกรณ์มาสเตอร์และการตอบสนองจากอุปกรณ์สเลฟ อุปกรณ์มาสเตอร์ทำให้เกิดการรีเซ็ทบนสายสัญญาณเพื่อแจ้งแก่อุปกรณ์สเลฟ โดยการทำให้สายสัญญาณเป็นลอจิกต่ำต่อเนื่องอย่างน้อย 480 ไมโครวินาที และจะต้องทำให้สายสัญญาณกลับมาเป็นลอจิกสูงภายใน 480 ไมโครวินาทีหลังจากนั้น ถ้าหากมีอุปกรณ์สเลฟต่ออยู่บนสายสัญญาณ จะมีการตอบสนองต่อสัญญาณรีเซตนั้นด้วยสัญญาณตอบสนอง โดยการทำให้สายสัญญาณเป็นลอจิกต่ำต่อเนื่องนานประมาณ 60-240 ไมโครวินาที หลังจากสิ้นสุดสัญญาณรีเซต (เปลี่ยนจากลอจิกต่ำเป็นลอจิกสูง)ประมาณ 15-60 ไมโครวินาที



รูปที่ 2.27 Time Slot การอ่านข้อมูลของอุปกรณ์มาสเตอร์หรือTime Slot การเขียนข้อมูลของอุปกรณ์สเลฟ ด้วยข้อมูล"0011 0011"

เมื่อต้องการอ่านข้อมูลจากอุปกรณ์สเลฟ อุปกรณ์มาสเตอร์จะทำให้สายสัญญาณเป็นลอจิกต่ำไม่น้อยกว่า 1 ไมโครวินาทีแต่น้อยกว่า 15 ไมโครวินาที จากนั้นอุปกรณ์มาสเตอร์ต้องทำให้สถานะของสายสัญญาณกลับมาเป็นลอจิกสูง แล้วทำการอ่านข้อมูลที่อุปกรณ์สเลฟส่งมาที่เวลาประมาณ 15 ไมโครวินาทีนับจากจุดเริ่มต้นของ Time Slot โดยถ้าข้อมูลที่อุปกรณ์สเลฟส่งมาให้อุปกรณ์มาสเตอร์เป็น “0” อุปกรณ์สเลฟจะทำให้สายสัญญาณเป็นลอจิกต่ำเป็นเวลาไม่เกิน 45 ไมโครวินาที แล้วทำให้สายสัญญาณกลับมาสู่สภาวะลอจิกสูงเหมือนเดิม แต่ถ้าข้อมูลเป็น “1” อุปกรณ์สเลฟจะทำให้สายสัญญาณเป็นลอจิกสูงต่อเนื่องไป โดยรวมแล้วใน Time Slot นี้ใช้เวลาทั้งหมดประมาณ 60-120 ไมโครวินาที นั่นคือ ใน Time Slot นี้จะใช้เวลารวมไม่เกิน 120 ไมโครวินาที ดังรูปที่ 2.27 แสดงรูปสัญญาณของ Time Slot การอ่านข้อมูลของอุปกรณ์มาสเตอร์ซึ่งมีลักษณะเหมือนกันกับการเขียนข้อมูลของอุปกรณ์สเลฟโดย Time Slot ทั้งสองจะเกิดขึ้นในช่วงเวลาเดียวกันคือเมื่ออุปกรณ์มาสเตอร์อ่านข้อมูลจากอุปกรณ์สเลฟตัวใด อุปกรณ์สเลฟตัวนั้นก็ต้องทำการเขียนหรือส่งข้อมูลมาให้อุปกรณ์มาสเตอร์นั่นเอง



รูปที่ 2.28 Time Slot การเขียนข้อมูลของอุปกรณ์มาสเตอร์ ซึ่งตรงกับ Time Slot การอ่านของอุปกรณ์สเลฟ

เมื่ออุปกรณ์มาสเตอร์ต้องการเขียนข้อมูล อุปกรณ์มาสเตอร์จะทำให้สายสัญญาณเป็นลอจิกต่ำประมาณ 1-15 ไมโครวินาที จากนั้นต้องทำให้สถานะของสายกลับมาเป็นลอจิกสูง แล้วทำการเขียนข้อมูลทันที ถ้าข้อมูลที่ต้องการเขียนไปยังอุปกรณ์สเลฟเป็น “0” อุปกรณ์มาสเตอร์จะ

ทำให้สายสัญญาณเป็นลอจิกต่ำนานประมาณ 45 ไมโครวินาที แล้วทำให้สายสัญญาณกลับมาสู่สภาวะลอจิกสูงเหมือนเดิม แต่ถ้าต้องการเขียนข้อมูล "1" อุปกรณ์มาสเตอร์จะทำให้สายสัญญาณเป็นลอจิกสูงต่อเนื่องไปอีก 45 ไมโครวินาที โดยรวมใน Time Slot นี้ใช้เวลาทั้งหมดประมาณ 60-120 ไมโครวินาที รูปที่ 2.28 แสดงรูปสัญญาณของ Time Slot การเขียนข้อมูลของอุปกรณ์มาสเตอร์ ซึ่งมีลักษณะเหมือนกับการอ่านข้อมูลของอุปกรณ์สเลฟ โดย Time Slot ทั้งสองจะเกิดขึ้นในช่วงเวลาเดียวกัน กล่าวคือ เมื่ออุปกรณ์มาสเตอร์เขียนข้อมูลไปยังอุปกรณ์สเลฟตัวใด อุปกรณ์สเลฟตัวนั้นก็ต้องทำการอ่านหรือรับข้อมูลจากอุปกรณ์มาสเตอร์นั่นเอง

ในการออกแบบของวิทยานิพนธ์ฉบับนี้ได้ใช้ไอซี DS1820 สำหรับสาธิตการทำงานในการสื่อสารด้วยโปรโตคอล 1-Wire สำหรับไอซีเบอร์นี้เป็นไอซีตรวจจับอุณหภูมิที่ใช้การติดต่อแบบระบบบัส 1 สาย มีขาสำหรับต่อใช้งานเพียง 3 ขา คือขาสัญญาณควบคุม, ขาไฟเลี้ยง และกราวด์ หัวใจสำคัญของ DS1820 อยู่ที่ตัวตรวจจับอุณหภูมิและหน่วยความจำความเร็วสูงที่เรียกว่าสแครดซ์แพด (scratchpad) ซึ่งมีขนาด 9 ไบต์ เมื่อวัดอุณหภูมิได้ก็จะนำค่าที่วัดได้มาเก็บไว้ในสแครดแพดที่ไบต์ 0 และ 1 ทั้งนี้เนื่องจาก DS1820 สามารถให้ข้อมูลของอุณหภูมิได้ละเอียดถึง 9 บิต สามารถแสดงค่าอุณหภูมิได้ละเอียดถึง 0.5 องศาเซลเซียส มีย่านวัดอุณหภูมิ -55 ถึง 125 องศาเซลเซียส ใช้เวลาในการแปลงข้อมูลในแต่ละครั้งประมาณ 200 มิลลิวินาที

กระบวนการในการสื่อสารข้อมูลกับ DS1820 ในแต่ละครั้งนั้นจะประกอบด้วยขั้นตอนหลักๆอยู่ 3 ขั้นตอนคือ

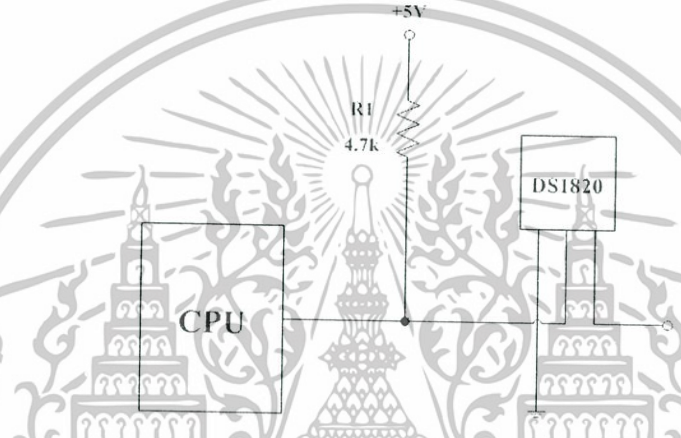
- เริ่มต้นสื่อสารข้อมูล (Initialization) โดยอุปกรณ์มาสเตอร์ส่งสัญญาณรีเซตไปที่อุปกรณ์สเลฟ (DS1820) จากนั้นอุปกรณ์สเลฟส่งสัญญาณตอบรับกลับมา
- คำสั่งที่ใช้ในการติดต่อข้อมูลในรอม (ROM Command) ได้แก่ คำสั่ง Search ROM, Read ROM, Skip ROM
- คำสั่งควบคุมการทำงาน (Function Command) เช่น Convert T, Read Scratchpad เป็นต้น

ปกติแล้วการที่อุปกรณ์มาสเตอร์จะติดต่อกับอุปกรณ์สเลฟในระบบบัส 1 สายนั้น จะต้องระบุตำแหน่งหรือหมายเลขเฉพาะตัวของอุปกรณ์ที่ต้องการติดต่อด้วยแต่ในกรณีของการออกแบบนี้ซึ่งมีอุปกรณ์อยู่บนบัสสัญญาณเพียงตัวเดียว อุปกรณ์มาสเตอร์สามารถติดต่ออุปกรณ์ได้โดยไม่จำเป็นต้องระบุตำแหน่งของอุปกรณ์นั้น ซึ่งรวมถึง DS1820 นี้ด้วย โดยการควบคุมการทำงานของ DS1820 สำหรับในกรณีนี้จะใช้คำสั่งเพียง 3 คำสั่งคือ

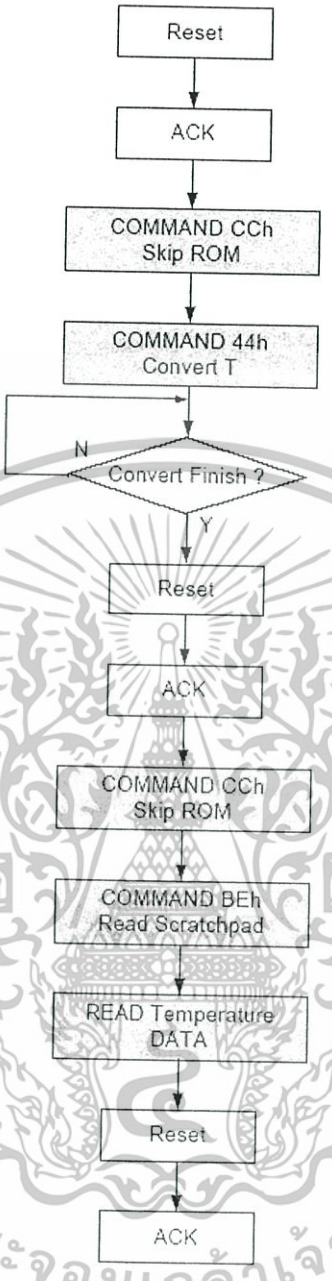
- คำสั่ง Skip ROM การใช้งาน DS1820 ในกรณีที่มี DS1820 อยู่บนบัสเพียงตัวเดียวนั้นไม่จำเป็นต้องอ้างอิงตำแหน่งของอุปกรณ์ จึงไม่ต้องติดต่อกับหน่วยความจำรอมที่เก็บข้อมูลเฉพาะตัวของอุปกรณ์เพื่ออ้างอิง โดยใช้คำสั่ง Skip ROM ซึ่งมีรหัสคือ CC_h

- คำสั่ง Convert T มีรหัสคือ 44_h เมื่อส่งคำสั่งนี้ให้ DS1820 จะต้องรออีกประมาณ 200 มิลลิวินาที เพื่อให้ DS1820 ได้ใช้เวลานี้ในการแปลงค่าอุณหภูมิเป็นข้อมูลดิจิตอลมาเก็บไว้ในหน่วยความจำสแครดซ์แพด
- คำสั่ง Read Scratchpad มีรหัสคือ BE_h เมื่อส่งคำสั่งนี้ DS1820 จะส่งข้อมูลค่าอุณหภูมิออกมาทั้งหมด

วงจรที่ใช้ทดลองการทำงานของ DS1820 ในที่นี้มีลักษณะดังรูปที่ 2.29 ซึ่งจะเห็นว่าหน่วยประมวลผลกลางจะใช้ขาพอร์ตเพียง 1 ขาสำหรับเชื่อมต่อกับ DS1820 โดยมีตัวต้านทานค่า 4.7 กิโลโอห์มต่อพูลอัพกับไฟเลี้ยง 5 โวลต์ และรูปที่ 2.30 แสดงขั้นตอนการติดต่อกับ DS1820



รูปที่ 2.29 วงจรใช้งาน DS1820



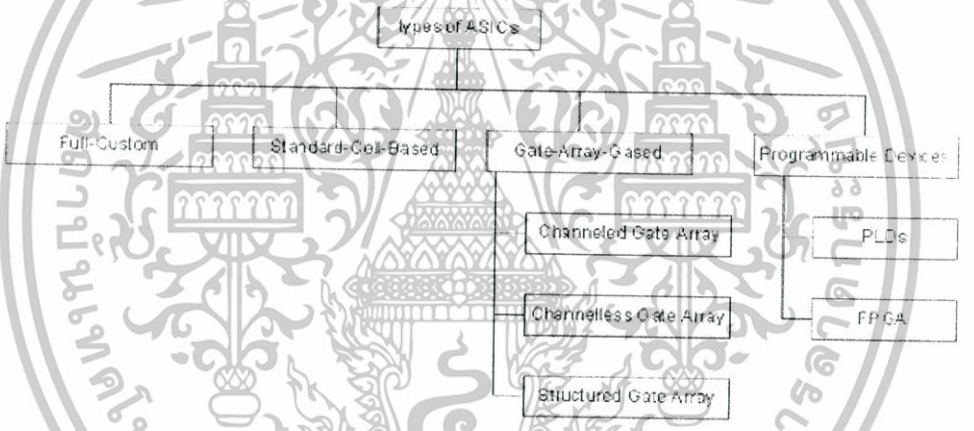
รูปที่ 2.30 ขั้นตอนการติดต่อกับ DS1820

2.3 พื้นฐาน FPGA และการออกแบบวงจรดิจิตอลด้วยภาษา VHDL

ชิพหรือไอซีที่สามารถโปรแกรมวงจรที่ผู้ออกแบบไว้ด้วยซอฟแวร์ได้เรามักจะเรียกชิพหรือไอซีประเภทนี้ว่าฟิลส์โปรแกรมเมเบิล (Field Programmable) ซึ่งมีอยู่หลายชนิดเช่น CPLD, FPLD และ FPGA เป็นต้น แต่จะมีลักษณะการสร้างหรือกำหนดการทำงานที่เหมือนกันคือผู้ใช้สามารถออกแบบและสร้างวงจรที่ต้องการใช้ลงในตัวอุปกรณ์ได้เองโดยไม่ต้องไปโรงงานผู้ผลิต การออกแบบวงจรดิจิตอลที่สามารถโปรแกรมได้นี้นับเป็นการก้าวหน้าอีกขั้นของการออกแบบวงจรดิจิตอลในรูปแบบของไอซีที่ใช้งานเฉพาะทาง (ASIC : Application Specific Integrated Circuit) และเป็นการเปลี่ยนลักษณะของการออกแบบบจากการนำเอาเกตแต่ละตัวมาต่อกันเป็น

แผนภาพวงจรที่เรียกว่า Schematic มาเป็นลักษณะของการออกแบบโดใช้ภาษาที่เรียกว่า HDL (Hardware Description Language) ที่ใช้อธิบายลักษณะการทำงานของวงจรแทน

จากการออกแบบวงจรรวมที่ต้องทำการเลย์เอาต์ (Layout) ทุกๆส่วนของวงจรด้วยตนเอง ไม่ว่าจะเป็นเกตต่างๆ การเชื่อมต่อกันระหว่างโหนดต่างๆ Contact, Pad และข้อกำหนดทางฟิสิกส์ เช่นนี้เป็นการออกแบบวงจรรวมที่ต้องมีทีมนักออกแบบที่มีความชำนาญและประสบการณ์สูง เพื่อที่จะได้รายละเอียดต่างๆสมบูรณ์พร้อมที่สุด การออกแบบวงจรรวมที่มีลักษณะเช่นนี้เรียกว่า Full-Custom Design ซึ่งวงจรที่ได้เราจะเรียกว่า Custom Chip ข้อดีของชิพแบบนี้คือ เราจะสามารถกำหนดค่าต่างๆได้ตามต้องการเช่น Delay Time ในโหนดต่างๆ, Power Consumption, ขนาดความจุของวงจร เป็นต้น ซึ่งจะทำให้ประสิทธิภาพของวงจรสูงเหมาะกับผลิตภัณฑ์จำนวนมาก (Mass Production) และ การใช้งานพิเศษเฉพาะทาง (Special application) แต่จะมีข้อเสียคือ ค่าใช้จ่ายสูงและใช้เวลาในการพัฒนานาน

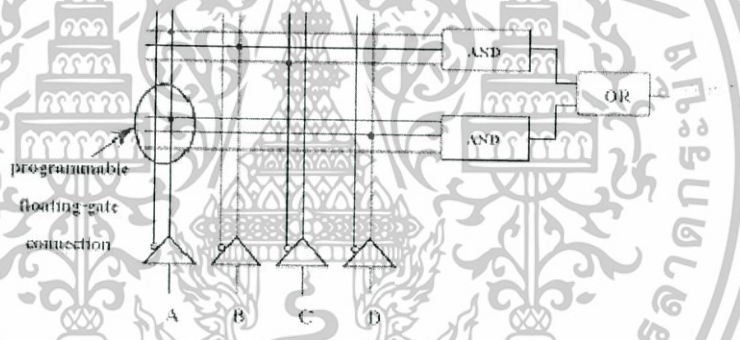


รูปที่ 2.31 ประเภทของ ASIC

ดังนั้นไอซีประเภท Field Programmable จึงถูกนำมาใช้ทดแทนข้อเสียของไอซีประเภทนี้ ซึ่งไอซี Field Programmable นี้จัดเป็นการออกแบบวงจรรวมประเภท Semi-Custom นั่นคือ เรามีชิปสำเร็จรูปที่มีวงจรบรรจุอยู่ภายในเป็นที่เรียบร้อยแล้วแต่เป็นวงจรที่ยืดหยุ่นสามารถโปรแกรมให้ทำหน้าที่ต่างๆ ได้ตามต้องการ การออกแบบก็จะใช้แผนภาพวงจรหรือภาษาประเภท HDL (Hardware Description Language) ในการระบุฟังก์ชันการทำงานของวงจรไม่ต้องใช้การเลเอาท์ (Layout) เหมือนแบบ Full-Custom ข้อดีของการออกแบบบนชิป Field Programmable นี้ก็คือใช้เวลาและประสบการณ์น้อยกว่า สามารถแก้ไขเปลี่ยนแปลงวงจรได้ง่าย ทำให้ลดเวลาในการออกแบบ เหมาะกับงาน ASIC Design ที่ต้องการผลิตน้อยชิ้นและใช้เวลาในการพัฒนาจำกัด

2.3.1 อุปกรณ์ FPGA (Field Programmable Gate Array)

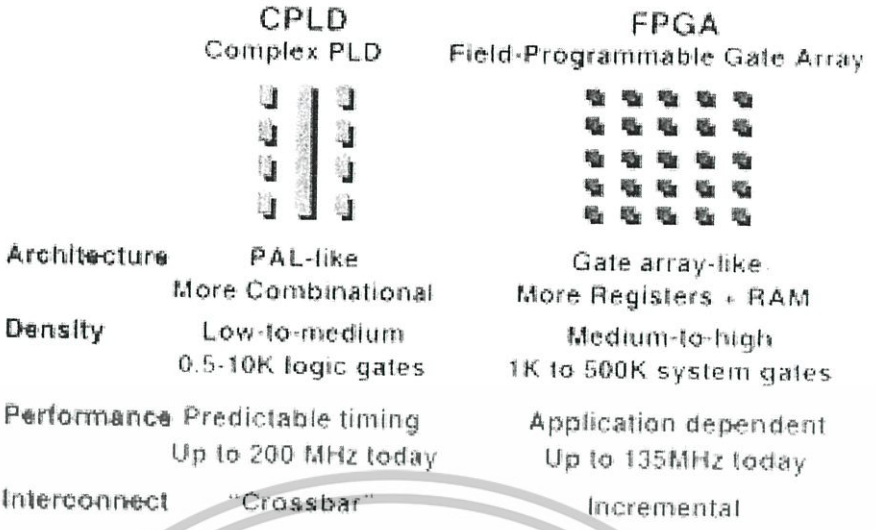
เป็นอุปกรณ์ประเภท Field Programmable หรืออยู่ในกลุ่มของ Programmable Logic Device: PLD ชนิดหนึ่งซึ่งภายในได้ถูกเตรียมไว้เป็นวงจรพื้นฐานทางด้านลอจิกต่อกันอยู่กันเป็นกลุ่มๆ มีทั้งวงจรที่เป็น Combination และ Sequential เป็นส่วนประกอบอยู่ภายใน เทคโนโลยีที่ใช้สร้างวงจรภายใน PLD มีทั้ง TTL, ECL และ CMOS ตามความเหมาะสม โดยพื้นฐานของวงจร Combination จะเป็นการทำงานแบบ Sum of Product ประกอบด้วยชุดของแอนด์เกตต่อรวมกับออร์เกต การโปรแกรมจะเป็นการเลือกว่าจะให้มีการต่ออินพุตภายในของแอนด์เกตกับสัญญาณอินพุตใดบ้าง ซึ่งมีทั้งจากภายนอกและสัญญาณป้อนกลับจากอินพุตภายในเอง การโปรแกรมนั้นถ้ามองทางกายภาพอุปกรณ์ทุกตัวจะถูกต่อผ่านฟิวส์เข้ากับแหล่งสัญญาณ ซึ่งถ้าโปรแกรมไม่ต้องการสัญญาณใดจะตัดฟิวส์นั้นออกทำให้สามารถโปรแกรมได้แค่เพียงครั้งเดียว อุปกรณ์ PLD บางชนิดใช้มอสเฟตทรานซิสเตอร์แทนฟิวส์ทำให้สามารถโปรแกรมโดยใช้กระแสไฟฟ้าได้และสามารถลบและโปรแกรมใหม่ได้อีก รูปที่ 2.32 แสดงลักษณะการทำงานของ PLD ในลักษณะ Sum of Product ของ Out = $AB\overline{C} + AD$



รูป 2.32 ตัวอย่างการเชื่อมต่อภายในเมื่อทำการโปรแกรมลงบนอุปกรณ์ PLD

ในปัจจุบันได้มีการพัฒนา PLD ออกมาและตั้งชื่อต่างๆกันออกไปเป็น SPLD (Simple Programmable Logic Device), CPLD (Complex Programmable Logic Device) และ FPGF (Field Programmable Gate Array) ตามมาเป็นลำดับ

เมื่อเปรียบเทียบความแตกต่างของ CPLD และ FPGA ดังรูปที่ 2.33 จะเห็นได้ว่าโครงสร้างภายในของ FPGA จะเป็นแบบ Gate array-link และ CPLD จะเป็นแบบ PAL-Link ซึ่งทำให้ความจุเกตของ CPLD น้อยกว่า FPGA แต่จะมี Timing ที่ดีกว่า



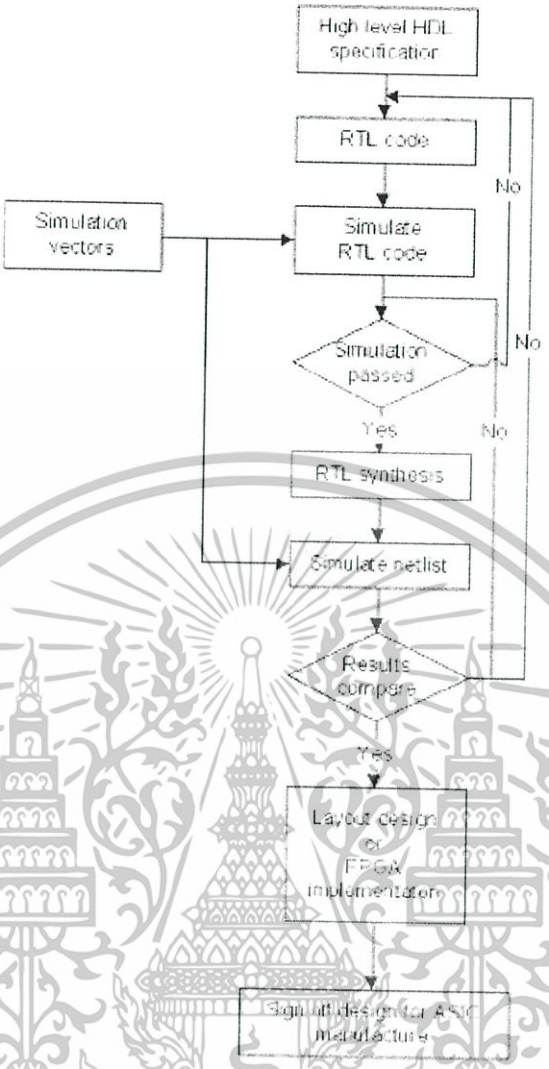
รูปที่ 2.33 เปรียบเทียบข้อแตกต่างระหว่าง CPLD กับ FPGA

2.3.2 ขั้นตอนการออกแบบวงจรรวม

ขั้นตอนการออกแบบไอซีวงจรรวมเฉพาะงาน (ASIC) โดยการใช้ภาษา HDL แสดงดังรูปที่ 2.34 การออกแบบจะเริ่มจากการออกแบบคุณสมบัติ (Specification) ของวงจรที่ต้องการออกแบบรวมทั้งวางบล็อคไดอะแกรม กำหนดฟังก์ชันการทำงานของแต่ละส่วนแล้วทำการเขียนอธิบายพฤติกรรมของแต่ละวงจรด้วย RTL Code ด้วย HDL จากนั้นทำการตรวจสอบการทำงานของวงจรที่สร้างด้วย RTL Code ด้วยโปรแกรมจำลองการทำงาน (Simulator) จากนั้นทำการสังเคราะห์วงจรด้วยโปรแกรมสังเคราะห์วงจร เมื่อผ่านขั้นตอนนี้จะได้โค้ดของวงจรที่บรรยายการเชื่อมต่อของวงจร Basic Element ของ Target Technology เช่น ลอจิกเกต, มัลติเพล็กเซอร์, ฟลิปฟลอป และบัฟเฟอร์ เป็นต้น จากนั้นนำเอาโค้ดของวงจรในระดับเกตนี้ไปจำลองการทำงานอีกครั้งโดยในครั้งนี้จะมีข้อมูลของค่าหน่วงเวลา (Delay) ของอุปกรณ์ (Element) ต่างๆร่วมด้วย

การจำลองการทำงานจะใกล้เคียงการทำงานจริงมากขึ้นเนื่องจากมีข้อมูลการหน่วงเวลาของแต่ละอุปกรณ์แต่ยังขาดข้อมูลของแต่ละเส้นทางการเชื่อมต่อระหว่างอุปกรณ์ต่างๆหรือที่เรียกว่า ค่าดีเลย์ของสาย (Wire Delay) แต่จะใช้วิธีประมาณจากข้อมูลของเจ้าของเทคโนโลยีที่เรียกใช้หรือที่เรียกว่า Wire-load model

เมื่อทดสอบการทำงานและแก้ไขวงจรจนทำงานได้ถูกต้องแล้วนำไปออกแบบวงจรในระดับเลย์เอาท์ (Layout) สำหรับการนำไปผลิตเป็นชิปบนซิลิกอนหรือจะนำไปออกแบบบน Programmable Device เช่น FPGA, Gate Array, CPLD ก็สามารถทำได้โดยใช้โปรแกรมที่ทำการวางบล็อกและกำหนดตำแหน่งลายวงจรบนชิปหรือที่เรียกว่าโปรแกรม Place & Route ซึ่งออกแบบมาสำหรับแต่ละประเภทของ Programmable Chip โดยเฉพาะ



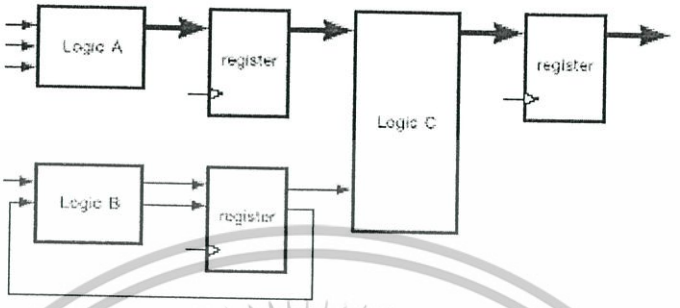
รูปที่ 2.34 ASIC Design Flow

2.3.3 การออกแบบวงจรดิจิตอลด้วยภาษา VHDL

ในช่วงปี 1980 กระทรวงกลาโหมของสหรัฐอเมริกา (The USA Department of Defense : DoD) ต้องการที่จะสร้างรูปแบบการออกแบบวงจรดิจิตอลที่สามารถอธิบายการทำงานของตัวมันเองได้ จึงได้จัดตั้งโครงการที่เรียกว่า "Very High Speed Integrated Circuit (VHSIC)" ขึ้นทำการออกแบบภาษาที่เป็นมาตรฐานในการบรรยายฟังก์ชันและโครงสร้างของวงจรดิจิตอลสำหรับการออกแบบวงจรรวม ภาษาที่ออกแบบขึ้นมานั้นจึงมีชื่อเต็มๆว่า "VHSIC Hardware Description Language (VHDL)" จากนั้นในปี 1987 ทาง IEEE (The Institute of Electrical and Electronic Engineering) ได้นำ VHDL มาศึกษากำหนดเป็นมารตฐานที่เรียกว่า IEEE 1976-1987 หรือเรียกสั้นๆว่า VHDL'87 และในปี 1993 IEEE ได้มีการปรับปรุงบางส่วนใหม่ และกำหนดเป็นมาตรฐานที่เรียกว่า IEEE 1976-1993 หรือที่เรียกว่า VHDL'1993 ใช้มาจนถึงปัจจุบัน

RTL (Register Transfer Level) คือการเขียนบรรยายพฤติกรรมของวงจรในระดับ Register และ Combination การออกแบบในลักษณะนี้นักออกแบบต้องนึกถึงรูปแบบการ

ประมวลผล เวลา (Timing) และการเชื่อมต่อสัญญาณระหว่างกลุ่ม Register และ Combination โดยที่ไม่จำเป็นต้องพิจารณาพฤติกรรมวงจรระดับลอจิกหรือระดับทรานซิสเตอร์ ดังรูปที่ 2.35 แสดงบล็อกไดอะแกรมของโครงสร้าง RTL



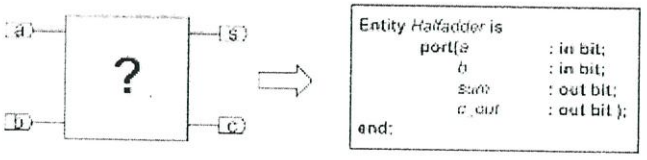
รูปที่ 2.35 ตัวอย่าง Register Transfer Level

ในการออกแบบวงจรดิจิตอลด้วย HDL ส่วนใหญ่จะทำการเขียนบรรยายพฤติกรรมวงจรในระดับ RTL เนื่องจากว่ามีความสะดวกในการออกแบบและให้วงจรที่มีประสิทธิภาพมากกว่าการเขียนบรรยายในระดับลอจิก ซึ่งใช้เวลาในการออกแบบนานและหาข้อผิดพลาดลำบากหรือในระดับ Behavioral ออกแบบง่ายแต่วงจรไม่สมบูรณ์ที่สุด (Optimize) การออกแบบในระดับ RTL จะมีความสะดวกในขั้นตอนของการสังเคราะห์วงจร, การทำ Timing analysis ซึ่งเป็นการวิเคราะห์แก้ปัญหาของ Timing ของวงจร, รวมทั้งการแบ่งวงจรแยกออกจากกันเป็นบล็อกย่อย (Partitioning) สำหรับการ Synthesis เพื่อให้ผลลัพท์ที่ดีที่สุดและยังสามารถทำความเข้าใจได้ไม่ยากนักสำหรับนักออกแบบ

องค์ประกอบหลักในการเขียนภาษา VHDL นั้นจำเป็นจะต้องเขียนให้มีรูปแบบตามข้อกำหนดที่มาตรฐานได้กำหนดไว้โดยมีองค์ประกอบหลักในการเขียนดังนี้

2.3.3.1 Entity

เป็นส่วนที่เขียนบรรยายโครงสร้างของพอร์ตของวงจรที่ต้องการออกแบบหรือหมายถึงส่วนที่ทำหน้าที่บอกว่าวงจรนั้นๆมีพอร์ตอะไรบ้างมีทิศทางเข้า (Input) หรือออก (Output) อย่างไรดังตัวอย่างรูปที่ 2.36



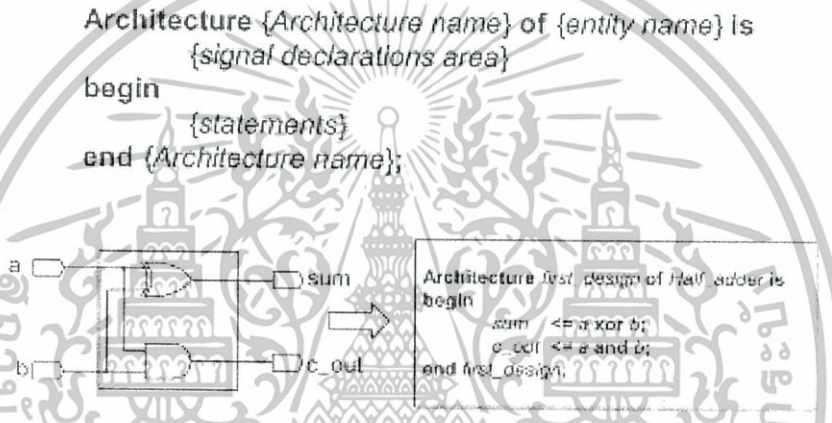
รูปที่ 2.36 Entity หรือส่วนที่เขียนบรรยายโครงสร้างของวงจรที่กำลังออกแบบ

อักษรตัวหนาคือ รูปแบบบังคับของภาษา ส่วนตัวเอียงคือคำที่ตั้งขึ้นเอง จากรูปที่ 2.36 เป็นการเขียน Entity เพื่อบรรยายพอร์ตของวงจรที่ชื่อ Half Adder ซึ่งมีพอร์ต a, b เป็นอินพุตมี

ประเภทสัญญาณเป็นแบบ “bit” คือมีขนาด 1 บิท มีสถานะเพียง “1” หรือ “0” เท่านั้น และมีเอาท์พุตพอร์ตเป็น sum และ c_out ซึ่งมีประเภทพอร์ตเป็น “bit” เช่นกัน จะเห็นว่า Entity จะไม่มีหน้าที่บรรยายการทำงานของวงจรแต่อย่างใดซึ่งจะเห็นเครื่องหมายคำถามในรูปนั้นหมายถึงยังไม่รู้ว่าวงจรภายในคืออะไร นั่นเอง

2.3.3.2 Architecture

เป็นส่วนของรายละเอียดวงจรภายในนั้นจะเขียนบรรยายอยู่ภายใต้คีย์เวอร์ด Architecture โดยมีรูปแบบดังในรูปที่ 2.37 โดยด้านบนเป็นรูปแบบบังคับของมาตรฐาน VHDL และด้านล่างเป็นตัวอย่างการเขียนสำหรับวงจร Half Adder ซึ่งมี Entity ที่ได้ออกแบบไว้ในรูปที่ 2.36 ที่ผ่านมา



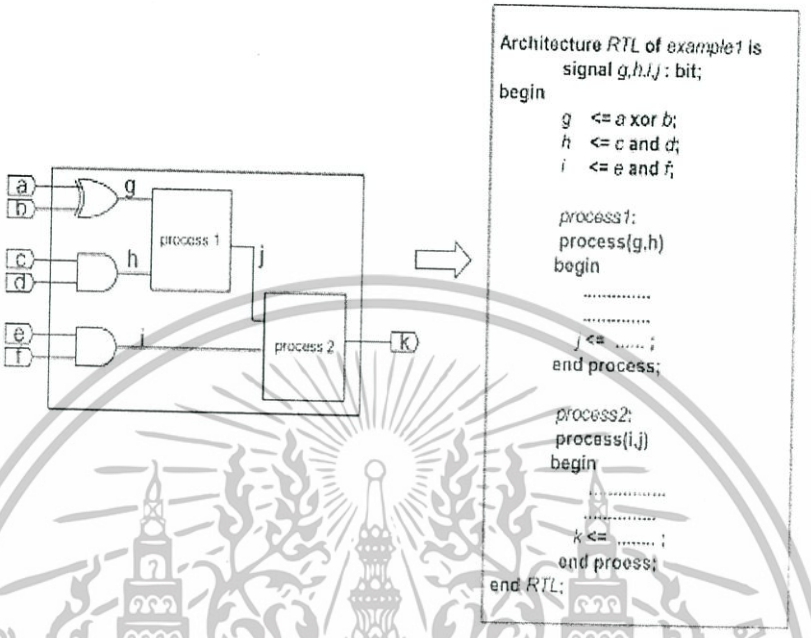
รูปที่ 2.37 แสดงรูปแบบของ Architecture และ ตัวอย่างการเขียน

จากโค้ดด้านขวามือของรูปที่ 2.37 first_design คือชื่อ architecture สำหรับแสดงเป็นรายละเอียดวงจรของ Entity ชื่อ Half_adder ที่ได้เขียนไว้แล้วตามรูปที่ 2.36 ส่วน xor และ and เป็นคำสั่งในภาษา VHDL ซึ่งเป็นการกระทำการทางลอจิก และเครื่องหมาย “<=” ในที่นี้ไม่ได้หมายถึง “น้อยกว่าเท่ากับ” แต่จะมองเป็นรูปหัวลูกศรมีความหมายเป็นการส่งค่าด้านขวามือของเครื่องหมายไปที่ด้านซ้ายให้มองเสมือนเป็นการเชื่อมต่อทางสัญญาณไฟฟ้าที่มีทิศทางตามหัวลูกศร ไม่ได้มองเป็นเครื่องหมายเท่ากับ เนื่องจากว่ามันมีความหมายทางวงจรคือทุกบรรทัดจะทำงานไปพร้อมๆกัน ค่าผลลัพท์ของ sum และ c_out จะเกิดขึ้นเวลาเดียวกัน จะไม่ทำทีละบรรทัด คำสั่งเหล่านี้เรียกว่า “Concurrent Statement”

2.3.3.3 Concurrent Statement

เหตุที่ภาษา HDL มีคำสั่ง Concurrent Statement เนื่องจากว่าต้องการให้สามารถเขียนบรรยายพฤติกรรมวงจรทางไฟฟ้าได้ ในการออกแบบวงจรในระดับฟังก์ชันเราจะมองว่าทุกส่วนยังไมมีค่าการหน่วงเวลาดังนั้นจากรูปที่ 2.37 ถ้าหากมองว่า and gate และ xor gate ไม่มีค่าหน่วงเวลาแล้ว ย่อมทำให้ sum และ c_out เกิดผลลัพท์ที่เวลาเดียวกัน

ส่วนที่อยู่ใต้ Architecture ทั้งหมดจะแบ่งกลุ่มคำสั่งได้ 2 ส่วนคือ statement ที่อยู่นอก Process และ statement ที่อยู่ใน Process โดยคำสั่งที่อยู่นอก Process นั่นจะเป็น Concurrent statement ทั้งหมด

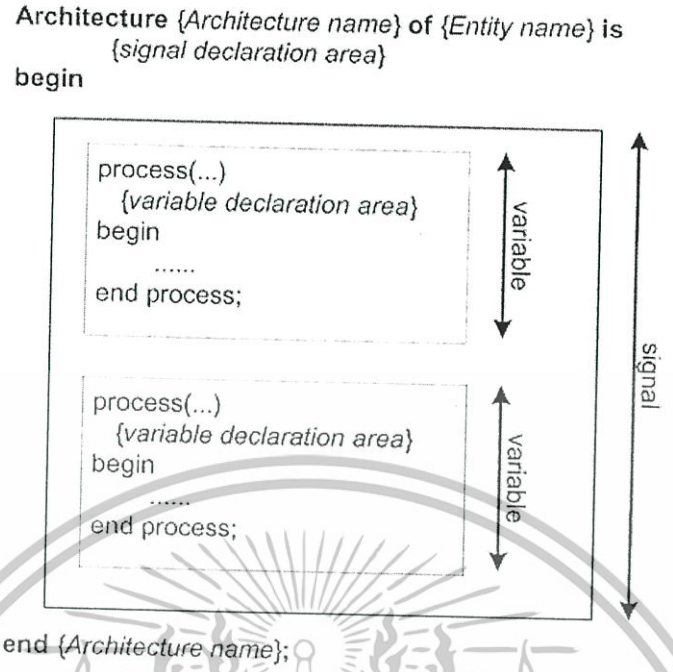


รูปที่ 2.38 ตัวอย่าง Concurrent Statement

จากรูปที่ 2.38 จะมีการกำหนด signal ที่ชื่อว่า g, h, I และ j เพิ่มขึ้นมาซึ่งเป็น signal ขนาด 1 บิต มี Process อยู่ 2 Process ชื่อ process1 และ process2 เชื่อมกันอยู่ด้วย signal j ตัวอย่างนี้ยังไม่ได้แสดงรายละเอียดวงจรภายใน process แต่จะแสดงให้เห็นว่าทุก signal (g, h, i, j) รวมทั้งเอาท์พุตพอร์ต k จะมีการเปลี่ยนแปลงค่าพร้อมๆกัน ดังนั้น ลำดับการวางตำแหน่งของ Concurrent Statement ในโค้ดจะไม่มีผลต่อการทำงานของวงจรแต่อย่างใด

2.3.3.4 Sequential Statement

จะอยู่ภายใต้คีย์เวอร์ด Process เท่านั้นลักษณะการใช้จะคล้ายกับการเขียนโปรแกรมซอฟแวร์ทั่วไปคือทำงานทีละบรรทัดตามลำดับตั้งแต่ต้นจนกระทั่งจบ process สิ่งสำคัญสิ่งหนึ่งสำหรับการใช้งาน process คือการใช้ signal และ variable ทั้ง 2 อย่างนี้ถ้าเทียบกับการเขียนโปรแกรมก็จะคล้ายๆกับตัวแปรต่างๆแต่จะเป็นในลักษณะแทนสัญญาณทางไฟฟ้า และมีความแตกต่างตรงการใช้งานของ variable จะอยู่ภายใต้ Process ที่ทำการประกาศ variable ไว้เท่านั้น ส่วน signal จะครอบคุมทั้ง Architecture จะอยู่ภายในหรือภายนอก Process ใดๆก็ได้ ดังรูป 2.39



รูปที่ 2.39 การใช้ Variable และ signal

ดังนั้นจากตัวอย่างในรูปที่ 2.39 ทั้งสอง Process สามารถประกาศ Variable เหมือนกันได้ เนื่องจากค่าของมันใช้ภายใน process ที่ประกาศไว้เท่านั้น ไม่มีผลอะไรกับสิ่งที่อยู่ภายนอก process

สำหรับในการใช้งาน Process สิ่งที่สำคัญอย่างหนึ่งคือ สิ่งต่างๆที่อยู่ภายใน Process จะเริ่มประมวลผลเมื่อสัญญาณที่ประกาศไว้ในวงเล็บหลังคำว่า "process (…, …, …)" มีการเปลี่ยนแปลงค่าเท่านั้นดังนั้นจะใส่ชื่อ signal ที่เป็นอินพุตของ process ไว้ภายในวงเล็บทุกๆ signal การทำงานของ process จึงจะถูกต้อง

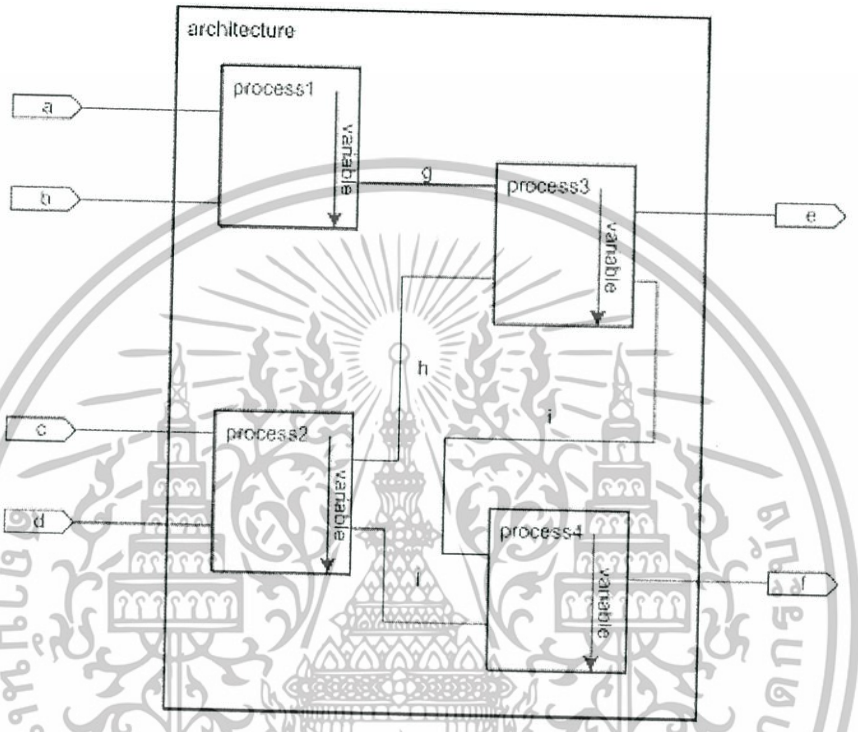
ความแตกต่างอีกอย่างระหว่างการใช้งาน signal และ variable ภายใน process คือการเปลี่ยนแปลงค่าของมัน สำหรับ variable การเปลี่ยนแปลงค่าจะเปลี่ยนบรรทัดต่อบรรทัดสามารถคิดแบบการเขียนโปรแกรมได้ แต่สำหรับ signal แล้วการเปลี่ยนแปลงค่าจะไม่เป็นบรรทัดต่อบรรทัด มันจะเปลี่ยนค่าเมื่อจบ process แล้วเท่านั้น

สรุปได้ว่า Sequential Statement คือคำสั่งที่อยู่ใน process มีการกระทำทีละบรรทัดเช่นเดียวกับการเขียนโปรแกรมสร้างซอฟท์แวร์ทั่วไปนั่นเอง เพียงแต่ค่าสัญญาณต่างๆจะเปลี่ยนแปลงอย่างไรก็ขึ้นอยู่กับว่าใช้ signal หรือ variable

ในการบรรยายพฤติกรรมของวงจรโดยใช้สิ่งที่กล่าวมาซึ่งได้แก่ Entity, Architecture, process, signal และ Variable จะเทียบได้กับ 2.40 คือ Entity เป็นส่วนที่บอกว่ามี input port คือ a, b, c, d และ มี e, f เป็น output port ภายใน Architecture ก็จะบอกว่ามี process อยู่ทั้งหมด 4 process และแต่ละ process จะเชื่อมต่อกันด้วย signal g, h, i และ j การบรรยายการทำงานแต่ละ

process สามารถสร้าง variable ที่รับค่าจาก signal แล้วนำมาประมวลผลเมื่อจบ process ก็ส่งค่าให้กับ signal ที่เชื่อมกับ process อื่นๆ

สิ่งสำคัญอย่างหนึ่งที่ลืมไม่ได้คือทุก Process ทำงาน concurrent กันทั้งหมดดังนั้นเอาต์พุทของทุก process นั่นคือ signal g, h, i, j และพอร์ต e, f จะเปลี่ยนแปลงค่าที่เวลาเดียวกันทั้งหมด

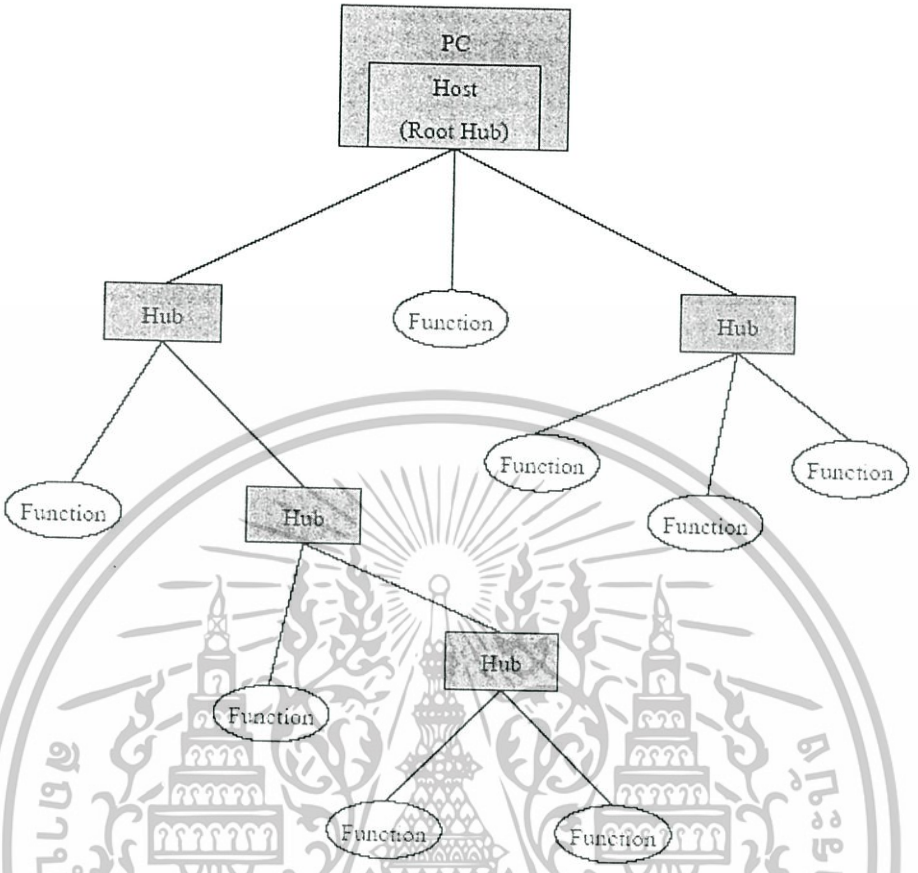


รูปที่ 2.40 รูปแบบการใช้งาน Variable กับ Signal ภายใต้ Architecture

2.4 พื้นฐานการสื่อสารด้วยพอร์ตยูเอสบี (USB Port)

USB คือบัสอนุกรมชนิดหนึ่ง ที่ออกแบบมาให้ง่ายต่อการเชื่อมต่อกับคอมพิวเตอร์ โดยจุดประสงค์หลักในการออกแบบในเบื้องต้นนั้น คือ เพื่อสร้างการเชื่อมต่อแบบพิเศษสำหรับอุปกรณ์ต่อพ่วงคนละชนิดกันแต่สามารถเชื่อมต่อกับคอนเน็คเตอร์ใดๆในคอมพิวเตอร์ก็ได้ โดยคุณสมบัติของแต่ละตัวสามารถกำหนดได้จากซอฟร์แวร์ USB แบ่งกลุ่มตามชนิดของการเชื่อมต่อออกเป็นสองคลาสคือ ฮับและฟังก์ชัน ฮับจะมีจุดเชื่อมต่อที่เรียกว่าพอร์ตซึ่งเป็นจุดที่เชื่อมต่อกับฟังก์ชัน สำหรับฟังก์ชันนั้นคืออุปกรณ์ทั่วไป เช่น คีย์บอร์ด ,เมาส์ ,เครื่องพิมพ์ เป็นต้น

USB จะมีการเชื่อมต่อแบบ multi-device ซึ่งอุปกรณ์ทุกตัวใช้สัญญาณการเชื่อมต่อเดียวกัน โดยจะส่งเป็นแพ็คเกตซึ่งทุกๆฟังก์ชันจะได้รับแพ็คเกตเหมือนกันแต่ในแพ็คเกตที่ได้รับนั้น จะมีแอดเดรสที่กำหนดว่าเป็นของอุปกรณ์ตัวใด ซึ่งอุปกรณ์แต่ละตัวก็จะเลือกรับเฉพาะแพ็คเกตของตนเองเท่านั้น ในการต่อฟังก์ชันนั้นจะสามารถต่อไปที่ฮับตัวใดก็ได้ในระบบแต่จะต้องไม่เกิน 5 ระดับ ดังแสดงในรูปที่ 2.41



รูปที่ 2.41 ระดับชั้นการเชื่อมต่อพอร์ต USB

แม้ว่าสเปคของ USB ในช่วงแรกจะกำหนดไว้ว่าสามารถเชื่อมต่อได้กับ host เพียงเครื่องเดียวในหนึ่งระบบแต่ในมาตรฐานล่าสุดคือ USB 2.0 ก็อนุญาตให้ใช้ host สองตัวในเวลาเดียวกันได้ ซึ่ง host จะถูกควบคุมจาก host คอนโทรลเลอร์ที่อยู่ในคอมพิวเตอร์อีกที

สำหรับสาย USB จะมีอยู่ 4 เส้นคือ V+, GND, Data+ และ Data- สองสายแรกจะเป็นไฟเลี้ยงและอีกสองสายจะเป็นสายสัญญาณ ที่มีเฟสกลับกัน ในส่วนของมาตรฐานของ USB นั้นจะมีความเร็วสามโหมดคือ 1 Mbits/s ในโหมด Low Speed ซึ่งกำหนดโดยมาตรฐาน USB 1.0, 12 Mbits/s ในโหมด Full Speed ของมาตรฐาน USB 1.1 และ 480 Mbits/s ในโหมด High Speed ซึ่งกำหนดโดยมาตรฐาน USB 2.0

เนื่องจาก USB ออกแบบไว้รองรับการทำงานของอุปกรณ์หลายประเภทซึ่งแต่ละประเภทจะมีความต้องการในการรับหรือส่งข้อมูลที่แตกต่างกันออกไปตามชนิด หน้าที่ของอุปกรณ์นั้นๆ ดังนั้นการรับส่งสัญญาณจึงต้องออกแบบไว้ให้ครอบคลุมการทำงานทุกๆประเภท ทำให้ USB กำหนดชนิดของการรับส่งข้อมูลเป็น 4 ชนิดคือ

2.4.1 การถ่ายทอดสัญญาณแบบไอโซโครนัส

การรับส่งสัญญาณแบบนี้จะมีการส่งหรือรับข้อมูลในทุกๆเฟรมข้อมูล (ทุกๆ 1 มิลลิวินาที) ทำให้เกิดการรับส่งข้อมูลด้วยอัตราเร็วที่คงที่ แต่นั่นก็หมายความว่า การรับส่งแบบนี้จะต้องจองการรับส่งข้อมูลในแต่ละเฟรม 1 มิลลิวินาทีไว้ตลอดด้วย

จังหวะการถ่ายทอดข้อมูลแบบไอโซโคนัสจะถูกกำหนดให้ส่งข้อมูลในทุกๆ 1 มิลลิวินาที เพื่อให้เกิดอัตราการส่งข้อมูลที่คงที่ขึ้น นั่นหมายความว่า จะต้องจองแบนด์วิดของสายไว้คงที่ตลอดเวลาขนาดข้อมูลที่ใหญ่ที่สุดที่สามารถส่งได้ในแต่ละเฟรมเท่ากับ 1023 ไบต์ โดยอุปกรณ์จะแจ้งขนาดของการรับส่งข้อมูลที่ต้องการใช้ในแต่ละเฟรมแก่โฮสต์ทางเอ็นพอยต์ดีสคริปเตอร์ ซึ่งโฮสต์จะนำไปใช้พิจารณาความสามารถระบบว่ารองรับการทำงานของอุปกรณ์ได้หรือไม่ หากพิจารณาแล้วเห็นว่าแบนด์วิดข้อมูลของระบบไม่เพียงพอต่อความต้องการของอุปกรณ์ ก็จะไม่ตั้งค่าที่จำเป็นในการทำงานให้อุปกรณ์ และปิดการทำงานของพอร์ตนั้นๆไป

เนื่องจากในการถ่ายทอดข้อมูลแบบนี้ต้องการความเร็วที่คงที่ตลอดเวลา เมื่อเกิดความผิดพลาดของข้อมูลจึงไม่สามารถส่งใหม่ได้ เพราะจะทำให้เกิดการซิงโครไนต์ของข้อมูลคลาดเคลื่อนไปส่งผลให้การรับข้อมูลเกิดความผิดพลาด ดังนั้นการถ่ายทอดข้อมูลแบบไอโซโครนัส จึงไม่มีการตรวจสอบความผิดพลาดของการส่งข้อมูลและไม่มีการส่งข้อมูลซ้ำ ทำให้วิธีการแบบนี้ไม่เหมาะกับงานที่ต้องการความถูกต้องของข้อมูลเป็นสำคัญ

2.4.2 การถ่ายทอดสัญญาณแบบอินเตอร์รัปต์

การรับส่งข้อมูลชนิดนี้สร้างขึ้นเพื่อเลียนแบบการสร้างสัญญาณอินเตอร์รัปต์ของอุปกรณ์ให้แก่ระบบ วิธีการเลียนแบบอาศัยการวนอ่านข้อมูลจากอุปกรณ์ต่างๆในระยะเวลาที่กำหนดอย่างสม่ำเสมอหรือที่เรียกว่า โพลลิ่ง (Polling) โดยถ้าหากอุปกรณ์ไม่มีข้อมูลที่ต้องการส่ง ก็จะส่งสัญญาณ NAK ตอบกลับมา

คาบเวลาในการวนอ่านข้อมูลของอุปกรณ์จะขึ้นอยู่กับความต้องการของอุปกรณ์แต่ละชนิด คาบเวลานี้สามารถเปลี่ยนแปลงได้ในช่วงกว้างตั้งแต่วนอ่านทุกๆหนึ่งเฟรม (1 มิลลิวินาที) ไปจนถึง 255 เฟรม (255 มิลลิวินาที) แต่ถ้าเป็นอุปกรณ์ความเร็วต่ำคาบเวลาที่สามารถวนอ่านได้เร็วที่สุดคือ 10 เฟรม (10 มิลลิวินาที) โฮสต์จะรู้คาบเวลาในการวนอ่านของเอนด์พอยต์ที่ใช้การส่งข้อมูลแบบนี้จากการอ่านเอนด์พอยต์ดีสคริปเตอร์

การส่งข้อมูลแต่ละครั้งสามารถกำหนดให้ส่งได้มากที่สุด 64 ไบต์ (ถ้าหากเป็นอุปกรณ์ความเร็วต่ำจะส่งได้สูงสุด 8 ไบต์) หากข้อมูลที่ต้องการส่งทั้งหมดมีมากกว่าก็จะต้องแบ่งเป็นส่วนๆโดยแต่ละส่วนจะต้องส่งให้เต็มจำนวนข้อมูลที่สามารถส่งได้สูงสุด (เพื่อให้โฮสต์รับทราบว่ายังส่งข้อมูลไม่ครบ) แล้วในการส่งครั้งสุดท้ายซึ่งจะส่งไม่เต็มจำนวนสูงสุดจะเป็นตัวบอกให้โฮสต์ทราบเองว่าจบการส่งข้อมูลแล้ว

การถ่ายทอดสัญญาณแบบนี้ถึงแม้จะต้องการอัตราการวนอ่านข้อมูลที่คงที่แต่ก็ไม่จำเป็นต้องซิงโครไนต์อัตราการส่งข้อมูลเข้ากับระบบ ดังนั้นเมื่อเกิดความผิดพลาดของการส่ง

ข้อมูลขึ้นก็สามารถส่งใหม่ได้ ทำให้การถ่ายทอดสัญญาณแบบนี้ต้องมีการตรวจสอบความถูกต้องของข้อมูลและมีการส่งข้อมูลใหม่ในรอบการวนอ่านถัดไป หากเกิดความผิดพลาดในการส่งข้อมูลเกิดขึ้น

2.4.3 การถ่ายทอดสัญญาณแบบบัลก์

เป็นการรับส่งข้อมูลผ่านพอร์ต USB สำหรับงานที่ไม่ต้องการอัตราเร็วในการถ่ายทอดสัญญาณที่คงที่ อุปกรณ์ที่เห็นได้ชัดที่สุดสำหรับการถ่ายทอดสัญญาณในลักษณะนี้คือ เครื่องพิมพ์ เพราะข้อมูลที่ผู้ใช้งานต้องการส่งไปพิมพ์นั้นไม่ต้องการความเร็วในการส่งที่สม่ำเสมอ การส่งช้าหรือเร็วก็จะส่งผลกระทบในด้านเวลาที่ต้องใช้มากขึ้นเท่านั้น แต่ข้อมูลที่ได้ก็ยังไม่เสียหายแต่อย่างใด

เนื่องจากการถ่ายทอดสัญญาณแบบนี้ไม่สามารถจองการใช้งานแบนด์วิดของบัสได้ ดังนั้นการรับส่งข้อมูลจะเกิดขึ้นก็ต่อเมื่อบัสว่างจากการส่งข้อมูลอื่นๆ ทั้งหมดแล้วเท่านั้น หรือกล่าวอย่างง่ายๆว่า การถ่ายทอดสัญญาณแบบนี้จะเกิดขึ้นได้ก็ต่อเมื่อเหลือที่ว่างของข้อมูลในแต่ละเฟรมเท่านั้น หากไม่มีที่ว่างเหลือก็จะต้องรอไปเรื่อยๆ จนกว่าจะสามารถส่งได้ แต่ในทางกลับกัน ถ้าในบัสไม่มีอุปกรณ์ตัวใดใช้งานอยู่เลยก็จะสามารถส่งข้อมูลได้เต็มความสามารถที่บัสจะส่งได้ทันที ทำให้ความเร็วในการส่งสูงมาก อาจสูงถึง 1 เมกะไบต์ต่อวินาทีเลยทีเดียว

ถึงแม้ว่าบัสจะว่างเพียงใด แต่การถ่ายทอดสัญญาณแบบบัลก์ก็ไม่สามารถส่งข้อมูลขนาดใหญ่ได้ในคราวเดียว เพราะเสี่ยงต่อการเกิดความผิดพลาดของข้อมูลระหว่างส่ง ข้อมูลจะต้องถูกแบ่งออกเป็นส่วนๆ แล้วทยอยส่งต่อกันไปในเฟรมเดียวกัน โดยขนาดของชิ้นข้อมูลที่ใหญ่ที่สุดหลังจากแบ่งแล้วจะเท่ากับ 8, 16, 32, หรือ 64 ไบต์ การส่งข้อมูลที่ถูกแบ่งจะต้องส่งในขนาดใหญ่ที่สุดที่ตกลงกันไว้กับโฮสต์เพื่อให้โฮสต์ทราบว่า ยังไม่จบการส่งข้อมูล และโฮสต์จะทราบว่าจบการส่ง เมื่อพบว่าขนาดของข้อมูลที่ส่งไม่เต็มขนาดใหญ่ที่สุดที่กำหนดไว้

เนื่องจากอุปกรณ์ที่ใช้การรับส่งสัญญาณแบบนี้ส่วนใหญ่ต้องการความถูกต้องของข้อมูลสูง เช่น ซีดีรอม เครื่องพิมพ์ หรือ เครื่องสแกน ดังนั้น การตรวจสอบข้อมูลจึงเป็นเรื่องจำเป็น ในการส่งข้อมูลแต่ละครั้งจึงมีการตรวจสอบความถูกต้องในทุกๆครั้ง และจะเกิดการส่งใหม่ถ้ามีความผิดพลาดเกิดขึ้น

2.4.4 การถ่ายทอดสัญญาณควบคุม

การรับส่งข้อมูลแบบนี้อาจถือได้ว่ามีความสำคัญที่สุดก็ว่าได้ เพราะใช้สำหรับส่งคำสั่งควบคุมการทำงานทั้งหมดของอุปกรณ์ทุกๆตัวตั้งแต่เริ่มแรกเมื่ออุปกรณ์ถูกเชื่อมต่อเข้ากับระบบ เกิดการอ่านข้อมูลดีสคริปเตอร์ต่างๆ เมื่อได้ข้อมูลมาโฮสต์ก็จะพิจารณาว่าสามารถรองรับการทำงานได้หรือไม่ เมื่อพิจารณาว่าได้ ก็จะตั้งค่าการทำงานต่างๆ และเริ่มการทำงานต่อไป ทั้งหมดที่กล่าวมานี้จะใช้การถ่ายทอดสัญญาณควบคุมเพื่อติดต่อกับเอ็นด์พอยต์คอนโทรลทั้งสิ้น

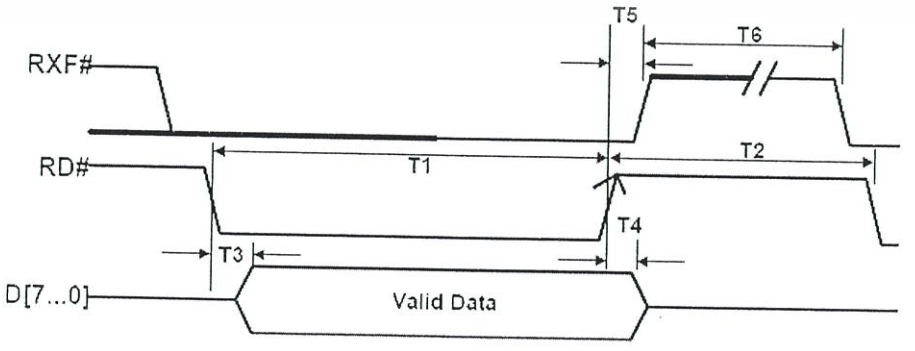
การถ่ายทอดสัญญาณทั้งสามแบบที่ผ่านมาล้วนแล้วแต่เป็นการรับส่งข้อมูลที่เกิดจากฟังก์ชันการทำงานของตัวอุปกรณ์ทั้งหมด แต่สำหรับการถ่ายทอดสัญญาณควบคุมนั้นเป็นการส่ง

ข้อมูลเพื่อควบคุมตัวอุปกรณ์ซึ่งความผิดพลาดในการส่งข้อมูลเป็นเรื่องที่ยอมรับไม่ได้ ดังนั้นการส่งข้อมูลชนิดนี้จะถูกออกแบบมาเป็นพิเศษกว่าการส่งชนิดอื่นๆ เพื่อป้องกันความผิดพลาดที่อาจจะเกิดขึ้น

การถ่ายทอดสัญญาณควบคุมประกอบด้วย การส่งข้อมูล 3 ช่วง โดยช่วงที่ 1 คือ Setup Stage ,ช่วงที่ 2 คือ Data Stage และ ช่วงที่ 3 คือ Handshake Stage จะเห็นได้ว่าคล้ายกับการแบ่งทรานแซคชั่นออกเป็น 3 ช่วงแต่จะต่างกันที่ แต่ละสเตจ คือ 1 ทรานแซคชั่น ซึ่งจะประกอบไปด้วยการส่งข้อมูล 3 ช่วง เรียบร้อยแล้ว จึงนำมาประกอบกันอีกชั้นหนึ่ง แต่สำหรับบางคำสั่งที่ไม่ต้องการการรับหรือส่งข้อมูลก็ไม่จำเป็นต้องมี Data Stage ซึ่งก็จะเหลือเพียงสองช่วงคือ Setup Stage และ Handshake Stage ใน Setup Stage จะบรรจุข้อมูลยาว 8 ไบต์ซึ่งภายในเป็นคำส่งต่างๆ เพื่อส่งไปยังตัวอุปกรณ์ ส่วน Data Stage จะบรรจุข้อมูลที่ต้องการส่งหรือรับขึ้นอยู่กับชนิดของคำสั่ง และส่วนสุดท้าย Handshake Stage เป็นการตรวจสอบการตอบรับคำสั่ง ซึ่งส่วนนี้เป็นส่วนที่พิเศษกว่าการส่งข้อมูลแบบปกติคือ จะใช้ทรานแซคชั่นขาออกที่มีความยาวข้อมูลเป็น 0 ในการตอบแฮนเช็กอีกครั้งหนึ่ง ซึ่งจะเห็นได้ว่าในการส่งคำสั่งแต่ละครั้งนั้นมีการตรวจความผิดพลาดกันอย่างเข้มงวดกว่าการส่งข้อมูลแบบอื่นๆอย่างเห็นได้ชัด

USB จะสำรองแบนด์วิดการส่งข้อมูลของบัสไว้ 10% เพื่อใช้สำหรับการถ่ายทอดสัญญาณควบคุมโดยเฉพาะ เพื่อไม่ให้เกิดปัญหาการควบคุมอุปกรณ์ล่าช้าเมื่อมีการรับส่งข้อมูลปริมาณมากๆเพราะอาจจะถูกอุปกรณ์บางตัวดึงแบนด์วิดไปใช้รับส่งข้อมูลจนไม่สามารถส่งสัญญาณควบคุมได้

ในส่วนการเชื่อมต่อและส่งถ่ายข้อมูลไปยังคอมพิวเตอร์จะอาศัยไอซี FT245 เพื่อช่วยในการสื่อสารด้วยยูเอสบี ถึงแม้ว่าโดยตัวของเอฟพีจีเอเองจะสามารถจัดการโปรโตคอลนี้ได้ด้วยตัวเอง แต่เนื่องจากการเชื่อมต่อด้วยยูเอสบีนั้นจำเป็นต้องอาศัยซอฟท์แวร์ Device Driver เพื่อจัดการในส่วนการรับส่งข้อมูลกับระบบปฏิบัติการซึ่งต้องใช้การเขียนโปรแกรมในระดับสูง รวมทั้งการเขียนซอฟแวร์ Device Driver นั้นจำเป็นต้องมีการลงทะเบียนกับองค์กรของยูเอสบีด้วย ซึ่งต้องเสียค่าใช้จ่ายสูง ด้วยเหตุนี้จึงจำเป็นต้องใช้ไอซีที่ทำหน้าที่จัดการโปรโตคอลยูเอสบี ซึ่งสามารถดาวท์โหลด ซอฟท์แวร์ Device Driver ได้จากเว็บไซท์ผู้ผลิต

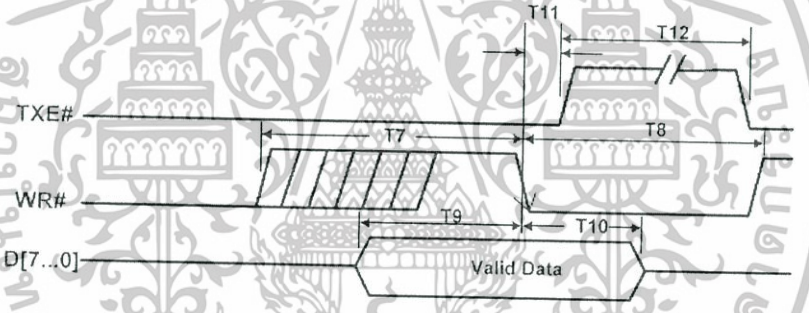


รูปที่ 2.42 ไดอะแกรมเวลาสำหรับการอ่านข้อมูลจาก FT2232

เมื่อโฮสต้องการส่งข้อมูลไปยังอุปกรณ์เชื่อมต่อโดยผ่านสาย USB ไอซี FT2232 จะทำให้ขา RXF# อยู่ในสภาวะลอจิกต่ำเพื่อเป็นการบอกหน่วยประมวลผลกลางที่เชื่อมต่ออยู่กับมันว่าขณะนั้นมีข้อมูลอยู่อย่างน้อย 1 ไบต์พร้อมให้หน่วยประมวลผลกลางอ่านออกไป ข้อมูลจะถูกอ่านโดยหน่วยประมวลผลกลางทำให้ขา RD# ของ FT2232 เป็น "0" ซึ่งจะทำให้ข้อมูล 1 ไบต์ปรากฏที่บัสเพื่อให้หน่วยประมวลผลกลางนำไปใช้ต่อไป และขา RXF# จะกลับสถานะมาเป็นลอจิกสูงอีกครั้ง

หากข้อมูลในบัฟเฟอร์ (FIFO) มีมากกว่า 1 ไบต์เมื่อการอ่านข้อมูลในไบต์แรกผ่านไปขา RXF# จะยังคงไม่เปลี่ยนสถานะเป็นลอจิกสูง จนกว่าบัฟเฟอร์จะว่าง ดังนั้น หน่วยประมวลผลกลางจะใช้ขานี้เพื่อดูสถานะในการวนอ่านข้อมูลจนหมดบัฟเฟอร์

สำหรับการส่งข้อมูลจากหน่วยประมวลผลกลางไปยังโฮสคอมพิวเตอร์ สามารถทำได้โดยการเขียนข้อมูลขนาด 1 ไบต์ไปยัง FT2232 ในขณะที่ขา TXE# มีสภาวะเป็นลอจิกต่ำ (FT2232 จะทำให้สภาวะของขา TXE# เป็นลอจิกสูงถ้าหากยังมีข้อมูลครั้งก่อนค้างหรือยังส่งไม่เสร็จ) โดยการทำให้ขา WR ของ FT2232 เป็น "1"



รูปที่ 2.43 ไดอะแกรมเวลาสำหรับการเขียนจาก FT2232 ไปยังโฮสคอมพิวเตอร์

2.5 พื้นฐานการเขียนโปรแกรมหลายเธรดและการซิงโครไนซ์ใน Visual C^{++}

ในการใช้งานโปรแกรมคอมพิวเตอร์ในปัจจุบันนิยมใช้เทคนิคที่เรียกว่า **multitasking** กันเป็นจำนวนมากซึ่งวิธีการนี้คือการทำงานหลายๆงานไปพร้อมๆกันโดยให้ process หรือแต่ละโปรแกรมแบ่งกันใช้งาน CPU โดยปกติแล้วภายในการทำงานของคอมพิวเตอร์ในช่วงเวลาใดเวลาหนึ่งจะทำงานได้เพียง 1 โปรแกรมเท่านั้นแต่สำหรับระบบ multitasking แล้วจะใช้การจัดสรรเวลา (scheduling) ให้กับแต่ละโปรแกรมโดยมีหลักการทำงานคือ เมื่อ CPU กำลังทำงานให้โปรแกรมแรกอยู่นั้น โปรแกรมที่เหลือในระบบจะรอจนกระทั่งถึงช่วงเวลาที่กำหนดแล้ว จึงเก็บสถานะต่างๆในโปรแกรมแรกไว้จากนั้นเปลี่ยนไปทำงานในโปรแกรมที่สองโดยทิ้งให้โปรแกรมแรกอยู่ในสถานะคอยการทำงานในรอบต่อไปและเมื่อทำงานในโปรแกรมที่สองจนถึงช่วงเวลาที่กำหนดก็จะเก็บสถานะต่างๆในโปรแกรมที่สองนี้ไว้พร้อมกับเปลี่ยนการทำงานไปที่โปรแกรมที่สามโดยให้โปรแกรมที่สองอยู่ในสถานะคอยการทำงานในรอบต่อไปเช่นเดียวกับ

โปรแกรมแรก และเมื่อถึงเวลาที่กำหนดอีกครั้งโปรแกรมที่สามก็จะทำงานในลักษณะเดียวกันนี้จนกระทั่งวนกลับมาที่โปรแกรมที่ 1 อีกครั้ง

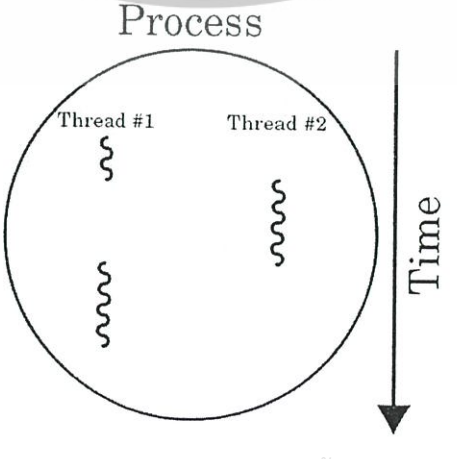
ลักษณะการสับเปลี่ยนการทำงานจากโปรแกรมหนึ่งไปอีกโปรแกรมหนึ่งเช่นนี้เรียกว่า "context switch" ซึ่งหากการสับเปลี่ยนการทำงานมีความเร็วสูงเพียงพอจะทำให้ดูเหมือนว่าทุกๆ โปรแกรมสามารถทำงานพร้อมๆกัน การทำงานในลักษณะนี้นั่นเองที่เรียกว่า "multitasking"

2.5.1 การเขียนโปรแกรมแบบหลายเธรด (Multithreading Programming)

เธรด (Thread) เป็นหน่วยของการทำงานหน่วยที่เล็กที่สุดในโปรแกรมคอมพิวเตอร์ซึ่งจะมองได้ว่าเป็นการทำงานตามคำสั่งตั้งแต่เริ่มต้นบรรทัดแรกจนกระทั่งสิ้นสุดบรรทัดสุดท้ายของแต่ละโปรแกรม จากตัวอย่างข้างต้นคำว่าโปรแกรมในที่นี้จะหมายถึง โปรเซส (Process) ซึ่งในโปรเซสหนึ่งๆจะมีอย่างน้อย 1 เธรด (Thread) อยู่ภายใน ซึ่งจริงๆแล้วถ้าหากมองว่าโปรเซสทั้ง 3 นั้นมีเธรดอยู่โปรเซสละ 1 เธรดก็จะหมายความว่าในการเปลี่ยนการทำงานของ Context Switch แต่ละครั้งนั้นก็คือการเปลี่ยนการทำงานของเธรดจากเธรดหนึ่งไปยังอีกเธรดหนึ่งนั่นเอง

ในทางปฏิบัติโปรเซสที่ทำงานอยู่ในช่วงเวลาใดเวลาหนึ่งในคอมพิวเตอร์นั้นมีอยู่มากมายหลายสิบโปรเซสซึ่งในแต่ละโปรเซสก็สามารถมีเธรดได้มากกว่า 1 เธรด สมมุติว่าในการทำงานบนคอมพิวเตอร์ในช่วงระยะเวลาหนึ่งมีเธรดอยู่ 100 เธรด ดังนั้น แต่ละเธรดจะได้รับการจัดสรรให้ทำงานจาก CPU ทุกๆ100 รอบ ซึ่งหากโปรเซสของเรามีเธรดอยู่เพียง 1 เธรดนั้นหมายความว่าโปรเซสของเราจะได้รับเวลาการทำงานเพียง 1 เปอร์เซ็นเท่านั้น ในความเป็นจริงหากโปรเซสของเราไม่มีอะไรต้องตอบสนองหรือคิดคำนวนอะไรมากเป็นพิเศษก็ถือว่าเพียงพอ แต่ถ้าหากมีความต้องการตอบสนองต่อข้อมูลจากอุปกรณ์อินพุต-เอาต์พุต พร้อมๆกับต้องคำนวนค่าที่ได้จากอุปกรณ์เหล่านั้นด้วยแล้ว จะเห็นว่ามีปริมาณงานที่ต้องทำเป็นจำนวนมากทำให้การทำงานด้วยเธรดเพียงเธรดเดียวจะไม่เพียงพอกับความต้องการ

ดังนั้น การเขียนโปรแกรมแบบหลายๆเธรด (Multithreading) จึงมีประโยชน์มากเมื่อโปรแกรมของเราต้องการการตอบสนองความเร็วสูงหรือมีปริมาณงานมากๆ รูปที่ 2.44 ลักษณะการทำงานแบบหลายเธรด



รูปที่ 2.44 ลักษณะการทำงาน Multithreading

สำหรับการเขียนโปรแกรม Visual C++ โดยใช้สถาปัตยกรรม MFC เพื่อให้ทำงานได้หลายๆเธรดนั้นจะสามารถทำได้โดย การเรียกฟังก์ชัน AfxBeginThread เพื่อสร้างเธรดใหม่ ฟังก์ชันนี้เมื่อทำงานเสร็จแล้วจะคืนค่าเป็นพอยเตอร์คลาส CWinThread สำหรับเก็บค่าแฮนเดิ้ล เพื่อใช้ในการจบการทำงานของเธรดจากนั้นก็จะไปเริ่มทำงานในฟังก์ชัน Thread ดังรูปที่ 2.45 ในตัวอย่างนี้ฟังก์ชันเธรดชื่อ ThreadFunction

```
void CDAQView::CreateNewThread()
{
    hThread = AfxBeginThread(ThreadFunction,this);
}
UINT ThreadFunction (LPVOID pParam)
{
    CDAQView *pView = (CDAQView*)pParam;
    ......
    ......
    ......
    return 0;
}
```

รูปที่ 2.45 ตัวอย่างการใช้งานเธรด

โดยปกติเมื่อเธรดทำงานในฟังก์ชันเธรด (ในที่นี้คือ ThreadFunction) จนกระทั่งพบคำสั่ง return 0; แล้วจะถือว่าการทำงานของเธรดสิ้นสุดลงและทรัพยากรหรือหน่วยความจำต่างๆที่ถูกใช้ไประหว่างที่เธรดทำงานก็จะถูกเคลียร์ออกไปด้วย

อย่างไรก็ตามในกรณีที่ต้องการจบการทำงานของเธรดในระหว่างที่เธรดกำลังทำงานอยู่นั้นจะเรียกฟังก์ชัน AfxEndThread ภายในฟังก์ชันเธรดหรือเรียกฟังก์ชัน TerminateThread เมื่อต้องการจบการทำงานจากเธรดหลัก

2.5.2 การซิงโครไนซ์เธรด (Thread Synchronization)

จากที่ได้ทราบแล้วว่าการสร้างเธรดขึ้นมาใหม่เพื่อทำงานอย่างหนึ่งอย่างใดนั้นจะสร้างขึ้นมาบนพื้นที่การทำงานของโปรเซส นั่นหมายความว่าหากมีตัวแปรโกลบอล (Global Variable) ใดที่ถูกประกาศไว้ในโปรเซสแล้วจะหมายความว่าทุกเธรดสามารถมองเห็นและเรียกใช้ตัวแปรนี้ได้ ดังนั้นจึงมีโอกาสที่ทุกเธรดจะเรียกใช้งานตัวแปรตัวนี้พร้อมกันซึ่งอาจก่อให้เกิดปัญหาความผิดพลาดในการคำนวนหรืออาจจะหนักถึงขั้นที่ทำให้โปรแกรมนั้นแฮงค์ได้ ดังนั้น จึงมีวิธีการที่จะทำให้แต่ละเธรดใช้งานตัวแปรโกลบอลไม่พร้อมกันหรือทำตามคิว เธรดใดเรียกใช้ก่อนก็ได้สิทธิ์ในการใช้งานก่อนพร้อมกับล็อคไม่ให้เธรดอื่นเข้ามาแก้ไขหรือเปลี่ยนแปลงค่าในตัวแปรนี้ได้ จนกระทั่งเธรดที่ถือครองอยู่ได้ปลดล็อกแล้วเท่านั้นเธรดที่เรียกใช้ถัดไปจึงจะสามารถแก้ไขเปลี่ยนแปลงค่าของตัวแปรนี้ได้ การทำงานลักษณะนี้เรียกว่า **"การซิงโครไนซ์เธรด"**

การเขียนโปรแกรมเพื่อทำซิงโครไนเธรดโดยใช้ Visual C^{++} นั้นมีอยู่หลายวิธีแต่สำหรับวิธีที่จะแนะนำและใช้ในการออกแบบโครงงานในวิทยานิพนธ์ฉบับนี้ คือ "Critical Section" ซึ่งการจัดการด้วยวิธีนี้จะเป็นแบบที่ง่ายที่สุดและเร็วที่สุดในการใช้ทรัพยากรร่วมกันของแต่ละเธรด

ในการใช้งานนั้นเริ่มต้นด้วยการสร้างออบเจ็ค CCriticalSection ใหม่ซึ่งเป็นคลาสที่ MFC ได้จัดเตรียมมาไว้โดยเฉพาะในการทำ Critical Section จากนั้นเรียกฟังก์ชัน Lock ในจุดก่อนที่จะมีการเรียกใช้ตัวแปรโกลบอลเป้าหมาย เพื่อป้องกันไม่ให้เธรดอื่นๆเรียกใช้ตัวแปรโกลบอลนี้ได้ ในระหว่างนี้หากเธรดอื่นต้องการเรียกใช้ตัวแปรตัวนี้ก็จะหยุดรอจนกระทั่งเธรดที่กำลังใช้อยู่นี้เรียกฟังก์ชัน UnLock เธรดต่อไปจึงจะเริ่มทำงานต่อไปได้ รูปที่ 2.46 แสดงการเขียนโปรแกรมเพื่อใช้งาน Critical Section

```
CCriticalSection g_cr;
int iSample[100];

////ในเธรดแรก////
g_cr.Lock();
...
////เปลี่ยนแปลงหรือแก้ไขข้อมูลใน iSample
...
g_cr.Unlock();
-------------------------------
////ในเธรดที่สอง////
g_cr.Lock();
...
////เปลี่ยนแปลงหรือแก้ไขข้อมูลใน iSample
...
g_cr.Unlock();
```

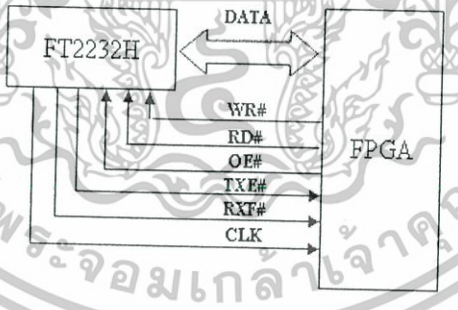
รูปที่ 2.46 การเขียนโปรแกรมเพื่อใช้งาน Critical Section ในการทำซิงโครไนเธรด

จากรูปที่ 2.46 จะเห็นว่ามีพื้นที่การทำงานแบ่งออกเป็น 2 บล็อคภายใต้ฟังก์ชัน Lock และ Unlock ของแต่ละเธรด (ในทางปฏิบัติบล็อคทั้งสองจะอยู่ในไฟล์หรือฟังก์ชันที่แตกต่างกันการเขียนให้ติดกันดังรูปเป็นเพียงเพื่อการอธิบายเท่านั้น) เมื่อเธรดแรกเข้ามาทำงานในบล็อคซึ่งเป็นพื้นที่ที่ถูกล็อคจะส่งผลทำให้เธรดที่สองไม่สามารถเข้าไปใช้งานพื้นที่ภายในบล็อคของตนได้โดยจะหยุดรอจนกว่าเธรดแรกทำงานจนกระทั่งเรียกฟังก์ชัน Unlock แล้วเท่านั้นเธรดที่สองจึงจะสามารถเข้าไปทำงานต่อได้ และในทางกลับกันเมื่อเธรดที่สองกำลังทำงานอยู่ภายในบล็อคจะส่งผลให้เธรดแรกไม่สามารถเข้าไปทำงานในบล็อกของตนได้เช่นกันและหยุดรอจนกว่าเธรดที่สองจะเรียกฟังก์ชัน Unlock แล้วเท่านั้นจึงจะสามารถเข้าไปทำงานในบล็อคต่อได้ ดังนั้น การแก้ไขเปลี่ยนแปลงค่าตัวแปรโกลบอลซึ่งในที่นี้คือ iSample จึงไม่มีโอกาสที่แต่ละเธรดจะทำพร้อมกันได้

บทที่ 3
การเชื่อมต่อกับคอมพิวเตอร์และโปรแกรมใช้งาน

การเชื่อมต่อกับคอมพิวเตอร์ที่ใช้ในการออกแบบนี้จะใช้การเชื่อมต่อด้วยพอร์ตยูเอสบีที่ทำงานด้วยความเร็วเต็มขั้น (Full Speed) โดยรูปแบบการส่งถ่ายข้อมูลของยูเอสบีนั้นจะส่งถ่ายไปทีละเฟรมข้อมูลโดยในแต่ละเฟรมจะประกอบด้วยหลายทรานแซคชั่นและในแต่ละทรานแซคชั่นจะประกอบด้วยแพคเกจ ในแต่ละแพคเกจที่รวมกันเป็นทรานแซคชั่นนี้จะประกอบไปด้วยข้อมูลที่หลากหลาย เช่น ชนิดของทรานแซคชั่น แอดเดรสของอุปกรณ์ยูเอสบี หมายเลขเอนพอยต์ และข้อมูลตรวจสอบความผิดพลาด เป็นต้น

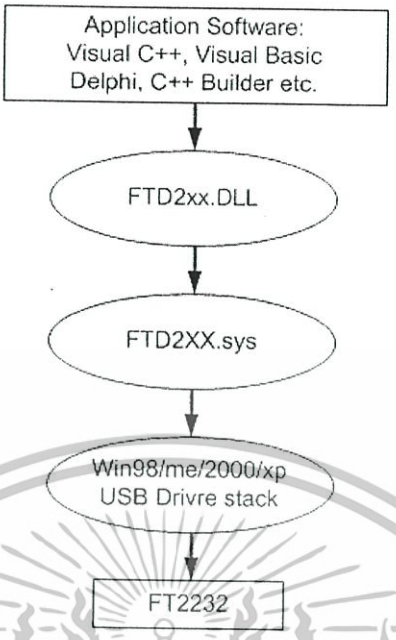
การสร้างโปรโตคอลเหล่านี้ในการออกแบบจะใช้ชิป FT2232H ของบริษัท FTDI ซึ่งทำหน้าที่จัดการโครงสร้างข้อมูลในระดับล่างของการสื่อสารผ่านพอร์ตยูเอสบี รวมถึงซอฟแวร์ของการส่งถ่ายข้อมูลหรือดีไวซ์ไดร์เวอร์ของชิปนี้จะใช้ไดร์เวอร์ชนิด D2XX.DLL ซึ่งเป็นการเข้าถึงข้อมูลผ่านฟังชั่นในไฟล์ DLL ที่ให้ความเร็วได้เต็มประสิทธิภาพของชิป ในส่วนต่อไปจะได้อธิบายโครงสร้างทางซอฟท์แวร์ของไดรเวอร์ชนิดนี้ สำหรับการเชื่อมต่อระหว่างเอฟพีจีเอกับไอซีตัวนี้จะใช้การเชื่อมต่อที่มีขาในการต่อใช้งานดังรูปที่ 4.1



รูปที่ 3.1 แสดงการเชื่อมต่อเอฟพีจีเอกับ FT2232H

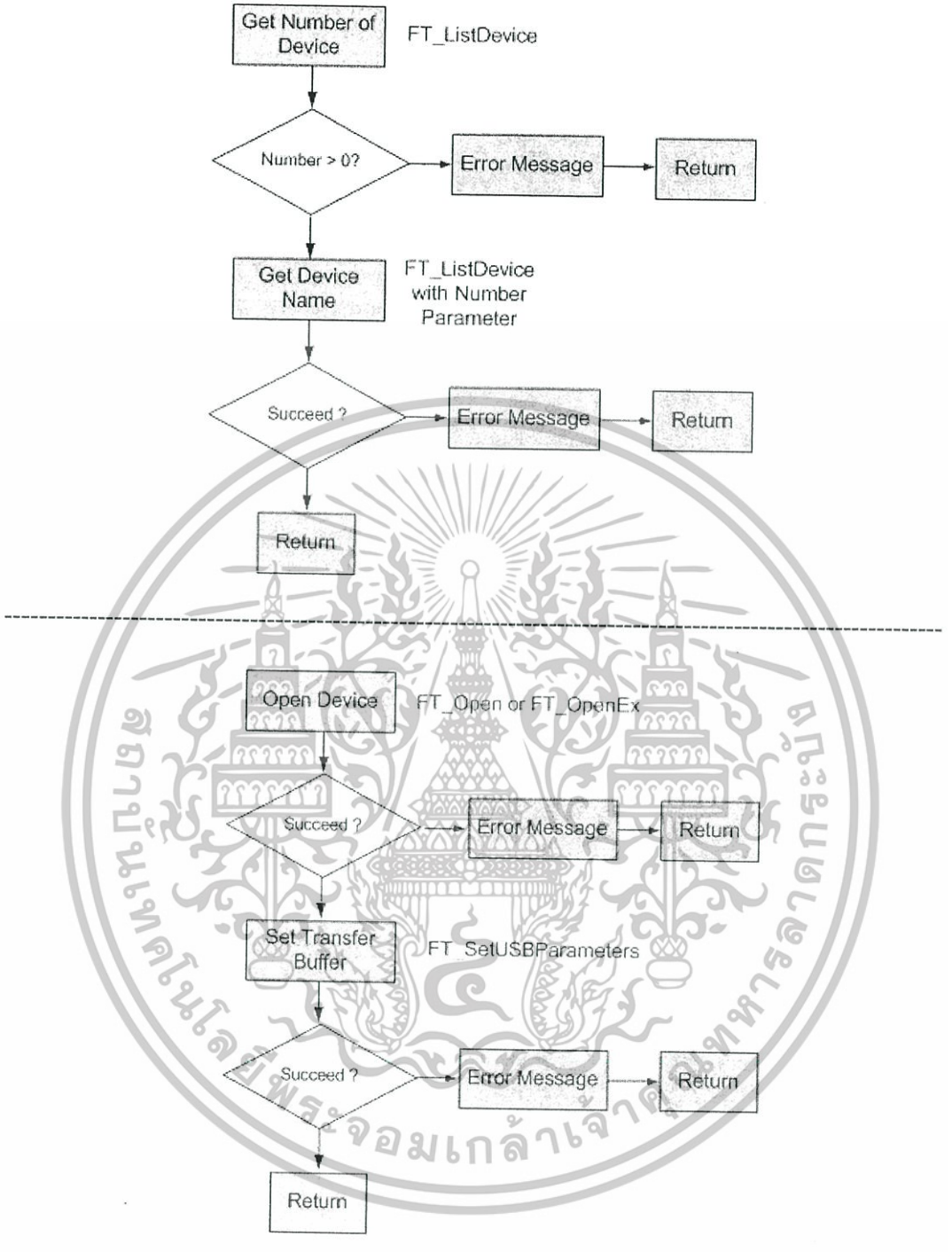
3.1 โครงสร้างของ D2XX ไดเร็คไดร์ฟเวอร์ (D2XX Device Driver Structure)

D2XX ไดเร็คไดร์ฟเวอร์เป็นไดร์ฟเวอร์อีกตัวหนึ่งนอกเหนือจากไดร์ฟเวอร์ Virtual COM Port ซึ่งช่วยให้แอพพลิเคชั่นซอฟร์แวร์สามารถเชื่อมต่อกับ FT2232 โดยผ่านทาง DLL (Dynamic Link Library) โครงสร้างของ D2XX ประกอบด้วยแบบจำลองไดร์ฟเวอร์ของระบบปฏิบัติการวินโดว์ (Window Driver Model) ซึ่งใช้สื่อสารข้อมูลกับพอร์ตยูเอสบีผ่านทางชั้น DLL ซึ่งเชื่อมต่อระหว่างแอพพลิเคชั่นซอฟท์แวร์กับ WDM ไดร์เวอร์ ดังรูปที่ 3.2



รูปที่ 3.2 แบบจำลองของ D2XX ไดเร็คไดร์ฟเวอร์

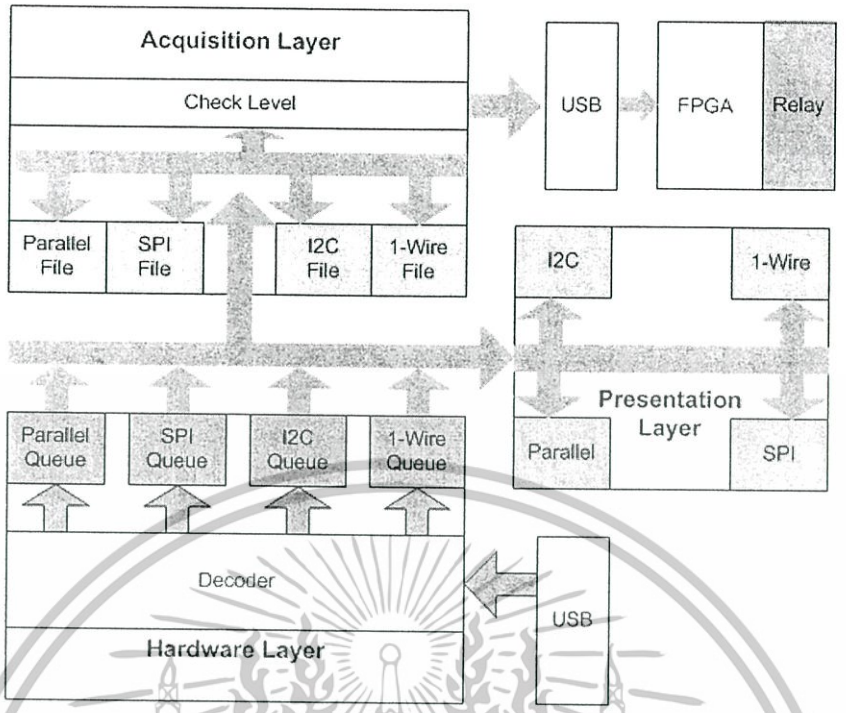
ภาพรวมการทำงานของไดร์ฟเวอร์ตัวนี้เริ่มจากการใช้คำสั่ง FT_ListDevices เพื่ออ่านข้อมูลเกี่ยวกับ FT2232 ที่ถูกต่ออยู่ภายในระบบ ซึ่งข้อมูลที่อ่านมาได้นี้จะช่วยให้แอพพลิเคชั่นซอฟร์แวร์สามารถตัดสินใจได้ว่า FT2232 ตัวใดที่มันต้องการทำการติดต่อสื่อสารด้วย แต่ก่อนที่จะสามารถแลกเปลี่ยนข้อมูลกับ FT2232 ได้นั้นเราต้องทำการเปิดการเชื่อมต่อออกมาก่อนด้วยการใช้คำสั่ง FT_Open หรือ FT_OpenEx ซึ่งฟังก์ชันทั้งสองนี้จะส่งค่าแฮนเดิ้ลของ FT2232 ที่ถูกเปิดกลับมาเพื่อให้ฟังก์ชั่นทุกๆฟังก์ชั่นใช้อ้างอิงเพื่อให้สามารถสื่อสารกับ FT2232 ที่ต้องการได้อย่างถูกต้อง จากนั้นการแลกเปลี่ยนข้อมูลโดยผ่านทาง I/O จะสามารถทำได้โดยการใช้คำสั่ง FT_Read และ FT_Write ซึ่งเป็นคำสั่งในการอ่านและเขียนข้อมูล ตามลำดับ จากนั้น หลังจากการทำงานทุกอย่างเสร็จสิ้นลง FT2232 จะถูกปิดโดยการเรียกใช้ฟังก์ชั่น FT_Close รูปที่3.3-3.5 แสดง Flow Chart สำหรับใช้ฟังก์ชัน FT_ListDevices, FT_Open, FT_OpenEx และ FT_Close



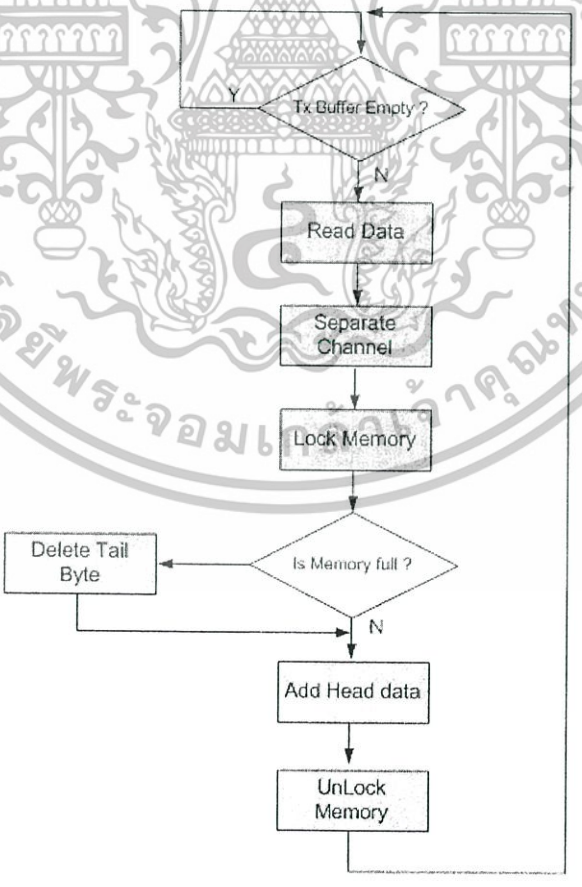
รูปที่ 3.3 Flow Chart การใช้ฟังก์ชัน FT_ListDevices, FT_Open, FT_OpenEx

3.2 โปรแกรมใช้งาน (Application Software)

โปรแกรมใช้งานของระบบนี้เขียนขึ้นจากภาษา Visual C++ โดยมีโครงสร้างของโปรแกรมแบ่งออกเป็น 3 เลเยอร์ประกอบด้วย เลเยอร์ฮาร์ดแวร์ เลเยอร์แอคควิซิชั่นและเลเยอร์แสดงผล ดังแสดงในรูปที่ 3.3



รูปที่ 3.4 โครงสร้างหลักภายในโปรแกรม



รูปที่ 3.5 Flow Chart การอ่านข้อมูลของฮาร์ดแวร์เลเยอร์

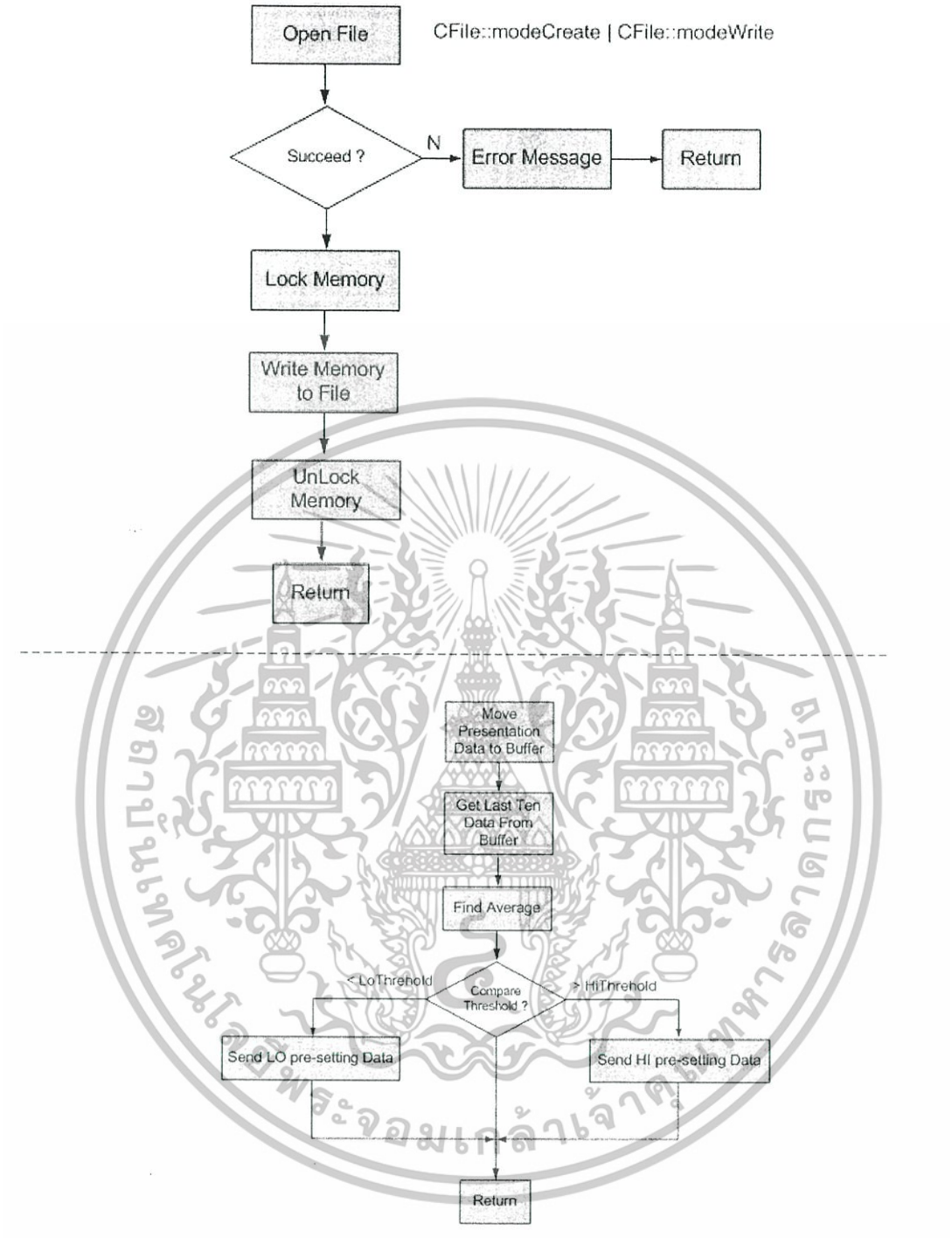
เลเยอร์ฮาร์ดแวร์ (Hardware Layer) ทำหน้าที่อ่านข้อมูลจากยูเอสบีพอร์ตซึ่งจะมีเธรดในการอ่านข้อมูลอยู่หนึ่งเธรดทำการอ่านข้อมูลทั้งหมดจากพอร์ตยูเอสบี จากนั้นทำการถอดรหัสเพื่อแยกข้อมูลออกเป็น 4 ช่องสัญญาณและเก็บไว้ใน Share Memory ที่มีโครงสร้างข้อมูลแบบคิวหรือ FIFO ซึ่งข้อมูลในหน่วยความจำนี้จะถูกอ่านออกไปโดยเลเยอร์แอคควิซิชั่น รูปที่ 3.5 แสดง Flow Chart การอ่านข้อมูลของฮาร์ดแวร์เลเยอร์โดยนอกจากการอ่านข้อมูลออกมาแล้วยังจำเป็นต้องถอดรหัสสัญญาณออกมาเป็นข้อมูลของแต่ละช่องสัญญาณด้วย

เลเยอร์แอคควิซิชั่น (Acquisition Layer) ทำหน้าที่อ่านข้อมูลจาก Memory ของเลเยอร์ฮาร์ดแวร์แล้วบันทึกผลลงคอมพิวเตอร์ตามเวลาที่ผู้ใช้กำหนดโดยจะมีเธรดรับผิดชอบในการบันทึกข้อมูลของแต่ละช่องสัญญาณแยกจากกันเป็นอิสระ นอกจากนั้นเลเยอร์นี้ยังมีหน้าที่ในการตัดสินใจว่าข้อมูลที่อ่านได้ในขณะนั้นมากกว่าหรือน้อยกว่าค่าเทรสโฮลด์ (Threshold) ที่กำหนดหรือไม่ โดยหากเกินกว่าขีดที่กำหนดไว้จะทำหน้าที่ส่งข้อมูลที่ได้กำหนดไว้ออกไปยังรีเลย์ ดังรูปที่ 3.6 แสดง Flow Chart การเขียนข้อมูลลงไฟล์และเปรียบเทียบข้อมูลกับค่าเทรชโฮลด์เพื่อควบคุมรีเลย์ตามค่าที่ได้กำหนดไว้

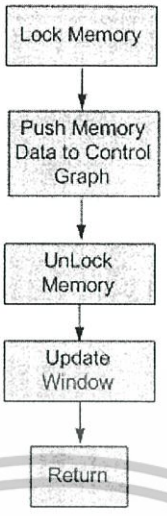
เลเยอร์แสดงผล (Presentation Layer) จะเป็นเธรดหลักของซอฟท์แวร์ซึ่งรับผิดชอบในการนำข้อมูลจาก Share Memory ของทุกช่องสัญญาณออกมาแสดงผลต่อผู้ใช้ ดังรูปที่ 3.7 เป็นตัวอย่างการนำข้อมูลจากหน่วยความจำมาวาดกราฟ

จากรูป 3.5 ถึง 3.7 ที่ผ่านมาโปรดสังเกตว่ามีบล็อกไดอะแกรมที่ชื่อ Lock Memory และ UnLock Memory เป็นส่วนประกอบอยู่ใน Flow Chart ด้วย ทั้งสองบล็อกนี้เป็นส่วนที่ทำหน้าที่ในการซิงโครไนซ์เธรด (Syncronization) เพื่อป้องกันไม่ให้แต่ละเธรดใช้ข้อมูลชนกัน

ท่านจะสังเกตได้ว่าทั้งสามเลเยอร์ไม่ว่าจะเป็นการอ่านข้อมูลจากพอร์ตยูเอสบี การเขียนข้อมูลลงไฟล์หรือการแสดงผลจะใช้ข้อมูลที่อยู่ใน Memory เดียวกันซึ่งหากไม่ทำการซิงโครไนซ์ก็อาจทำให้เกิดเหตุไม่พึงประสงค์ขึ้น เช่น ในขณะที่เธรดหนึ่งกำลังอ่านข้อมูลจากพอร์ตยูเอสบียังไม่เสร็จแต่อีกเธรดหนึ่งนำเอาข้อมูลที่ยังไม่สมบูรณ์นั้นไปวาดกราฟ ทำให้ได้ข้อมูลที่แสดงในกราฟผิดเพี้ยนไปได้ ดังนั้น เพื่อทำให้เกิดการซิงโครไนซ์หรือกำหนดจังหวะการใช้งาน Memory ที่แต่ละเธรดใช้ร่วมกัน ระหว่างบล็อก Lock Memory และ UnLock Memory นั้นจึงเป็นส่วนที่ทำให้เธรดที่กำลังใช้งาน Memory อยู่นั้นถือครองสิทธิ์ในการใช้งานแต่เพียงเธรดเดียวและแน่ใจได้ว่าข้อมูลใน Memory จะไม่ถูกเธรดอื่นเปลี่ยนแปลงระหว่างการใช้งาน

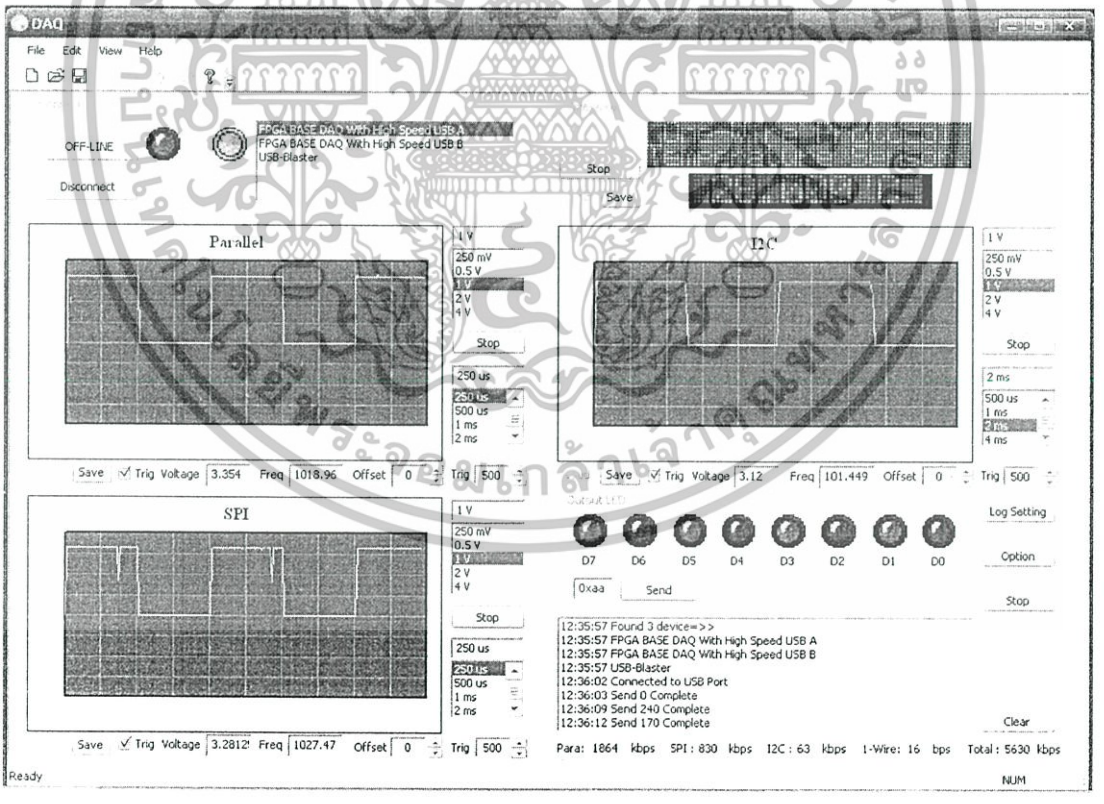


รูปที่ 3.6 Flow Chart การเขียนข้อมูลลงไฟล์และการส่งข้อมูลไปยังรีเลย์ของแอควิซิชั่นเลเยอร์



รูปที่ 3.7 Flow Chart การนำข้อมูลมาวาดกราฟของ Presentation Layer

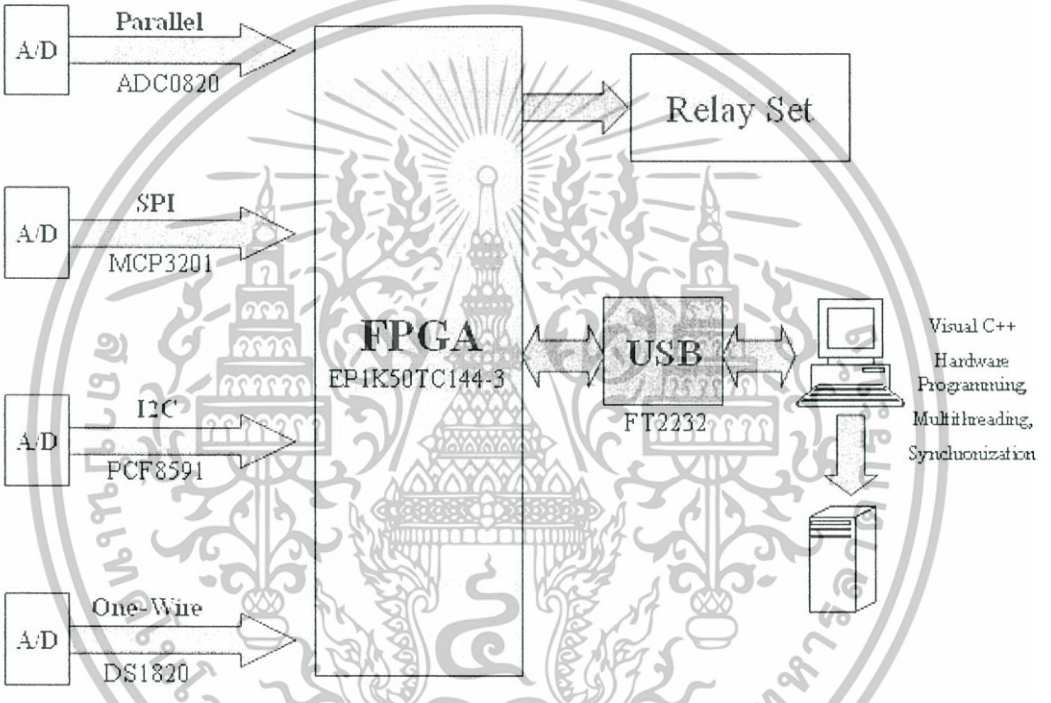
เมื่อนำเอาส่วนประกอบต่างๆของทั้งสามเลเยอร์ที่ได้กล่าวถึงไปแล้วนี้มาใช้งานร่วมกันก็จะได้เป็นโปรแกรมใช้งานที่สมบูรณ์ ดังแสดงในรูปที่ 3.8



รูปที่ 3.8 โปรแกรมใช้งานโดยสมบูรณ์ของระบบ

บทที่ 4
โครงสร้างของระบบและวงจรการทดลอง

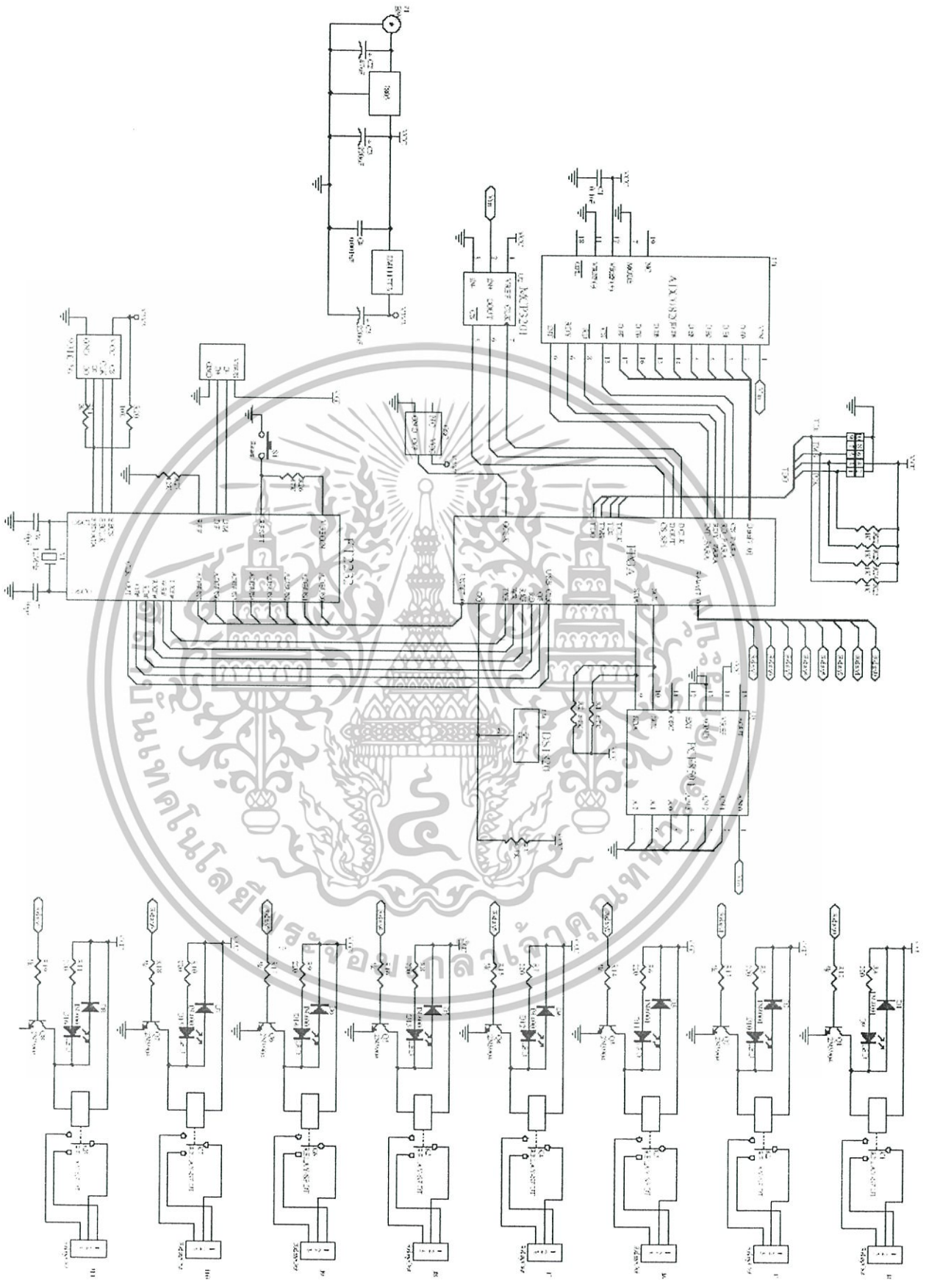
จากที่ได้กล่าวถึงทฤษฎีพื้นฐานซึ่งมีความจำเป็นต่อการออกแบบระบบไปแล้วในส่วนต่อไปจะเป็นการกล่าวถึงโครงสร้างและส่วนประกอบต่างๆของระบบรวมทั้งอธิบายแต่ละส่วนประกอบในส่วนฮาร์ดแวร์หรือวงจรการทดลองในเชิงการนำไปใช้งานจริง ดังรูปที่ 4.1แสดงโครงสร้างโดยรวมของการออกแบบนี้ทั้งระบบ



รูปที่ 4.1 โครงสร้างระบบโดยรวม

4.1 โครงสร้างของระบบ

จากรูปที่ 4.1 ระบบที่ทำการออกแบบประกอบด้วยช่องสัญญาณที่มีความแตกต่างทางด้านโปรโตคอลอยู่ 4 ช่องสัญญาณอันประกอบไปด้วย Parallel, SPI, I^2C และ 1-Wire ซึ่งใช้ไอซี ที่ทำหน้าที่ในการแปลงสัญญาณอะนาลอกเป็นดิจิตอลเบอร์ ADC0820, MCP3201, PCF8591 และ DS1820 ตามลำดับ (สำหรับ DS1820 เป็นเซนเซอร์วัดอุณหภูมิ) โดยช่องสัญญาณทั้งหมดเชื่อมต่อไปยังชิปเอฟพีจีเอเบอร์ EPF1K50TC144-3 ซึ่งทำหน้าที่เป็นหน่วยประมวลผลกลางของระบบและทำหน้าที่ดึงข้อมูลจาก ADC ของแต่ละช่องสัญญาณมา จากนั้นข้อมูลจะถูกส่งผ่านชิป FT2232 ซึ่งเป็นยูเอสบีโฮสคอนโทรลเลอร์ไปยังคอมพิวเตอร์เพื่อแสดงผลและเก็บข้อ



รูปที่ 4.2 วงจรการทดลอง

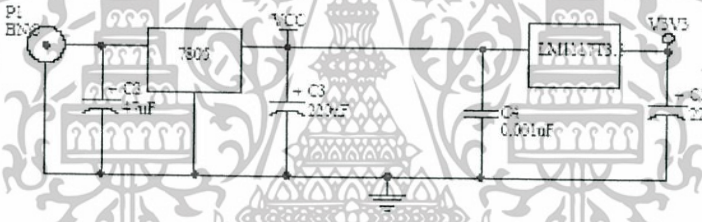
มูลต่อไป นอกจากนั้น เมื่อผู้ใช้ต้องการควบคุมระบบกลับมาก็สามารถสั่งผ่านซอฟท์แวร์เพื่อมาควบคุมชุดรีเลย์ได้อีกด้วย

4.2 วงจรการทดลอง

จากรูปที่ 4.2 แสดงรูปวงจรการทดลองซึ่งสามารถแบ่งการทำงานออกเป็น 5 ส่วนประกอบด้วย ภาคจ่ายไฟ, ภาคแปลงสัญญาณอะนาลอกเป็นดิจิตอล, ภาคยูเอสบีโฮสคอนโทรลเลอร์, ภาคหน่วยประมวลผลกลาง และภาคขับรีเลย์

4.2.1 ภาคจ่ายไฟ

ภาคจ่ายเป็นส่วนที่ทำหน้าที่จ่ายไฟเลี้ยงวงจรให้กับภาคอื่นๆ ซึ่งจะมีแรงดันไฟเลี้ยง 2 ระดับคือ 3.3 โวลต์จ่ายให้กับชิป FPGA และ 5 โวลต์ จ่ายให้กับภาคที่เหลือทั้งหมด รูปที่ 3.3 แสดงวงจรภาคจ่ายไฟของระบบ โดยแรงดันที่ป้อนให้กับภาคนี้นำมาจากหม้อแปลงอะแดปเตอร์ขนาด 12 โวลต์ 800 มิลลิแอมป์ จากภายนอกซึ่งเชื่อมต่อกับไฟบ้านโดยตรง



รูปที่ 4.3 วงจรภาคจ่ายไฟ

จากรูปที่ 4.3 จะเห็นว่ามีส่วนประกอบหลัก 2 ส่วนคือไอซีเบอร์ 7805 ซึ่งเป็นไอซีรักษาระดับแรงดันให้คงที่ 5 โวลต์โดยมี C2 และ C3 ทำหน้าที่กรองแรงดันให้เรียบก่อนส่งให้ไอซีเบอร์ LM1117T3.3 ซึ่งเป็นไอซีลดระดับแรงดันลงให้เหลือ 3.3 โวลต์ โดยคุณสมบัติเด่นของไอซีเบอร์นี้คือแรงดันตกคร่อมขณะทำงานจะต่ำกว่าไอซีเรกูเลเตอร์เบอร์อื่นๆ สำหรับ C4 จะทำหน้าที่กรองความถี่สูงที่ติดมาจากแหล่งจ่ายไฟทิ้งไปและ C5 ทำหน้าที่ในการกรองแรงดันให้เรียบอีกครั้งก่อนส่งไปให้ภาคต่อไป

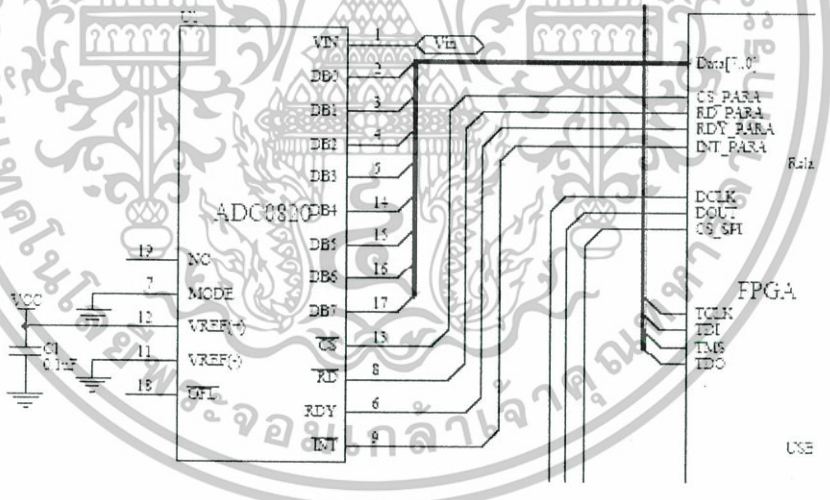
4.2.2 ภาคแปลงสัญญาณอะนาลอกเป็นดิจิตอล

ประกอบด้วย ADC จำนวน 4 ตัวคือ ADC0820 MCP3201 PCF8591 และ DS1820 ทำหน้าที่ในการแปลงสัญญาณอะนาลอกเป็นดิจิตอลแล้วสื่อสารกับชิปเอฟพีจีเอเพื่อส่งข้อมูลที่ได้จากการแปลงไปให้เอฟพีจีเอทำการประมวลผลและส่งไปยังคอมพิวเตอร์ต่อไป

ADC0820 เป็นไอซีแปลงอะนาลอกเป็นดิจิตอลชนิดแฟลชคอนเวอเตอร์ ใช้เวลาในการแปลงสัญญาณ (Conversion Time) 1.5 ไมโครวินาที ใช้โปรโตคอลในการดึงข้อมูลแบบขนานมีขาที่ใช้ต่อเป็นสัญญาณควบคุม 4 ขาประกอบด้วย

- $\overline{CS}$ ขานี้เป็นชนิดอินพุตทำงานเมื่อเป็นลอจิก “0” ทำหน้าที่อีนาเบิลชิปให้ทำงาน ซึ่งไอซีจะเริ่มแปลงข้อมูลเมื่อได้รับสัญญาณที่ขานี้
- $\overline{RD}$ ขานี้เป็นชนิดอินพุตทำงานเมื่อเป็นลอจิก “0” ทำหน้าที่รับคำสั่งจากชิปเอฟพีจีเอเมื่อต้องการอ่านข้อมูล
- RDY เป็นขาชนิดเอาท์พุตทำหน้าที่แจ้งสถานะขณะไอซีกำลังทำการแปลงข้อมูล โดยจะเป็น “0” เมื่อกำลังแปลงข้อมูลและเป็น “1” เมื่อแปลงเสร็จแล้ว
- INT เป็นขาชนิดเอาท์พุตทำหน้าที่แจ้งสถานะเมื่อไอซีแปลงสัญญาณเสร็จคล้ายขา RDY แต่ต่างกันที่ INT จะเป็น “0” เมื่อแปลงข้อมูลเสร็จแล้ว

นอกจากขาควบคุมที่กล่าวไปแล้วนี้ยังมีขาอื่นๆซึ่งต่อร่วมในวงจรด้วยเช่น Vin, VREF(+), VREF(-), Mode เป็นต้น รูปที่ 4.4 แสดงวงจรการเชื่อมต่อวงจรระหว่าง ADC0820 และชิปเอฟพีจีเอ



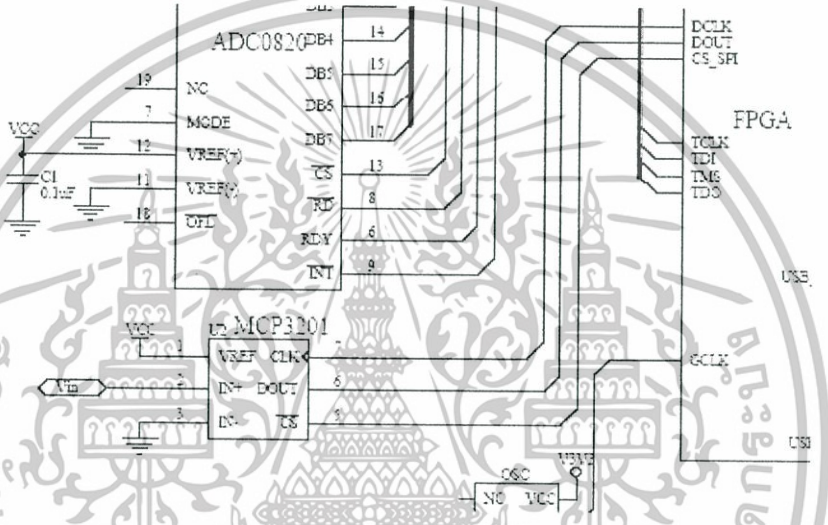
รูปที่ 4.4 วงจรเชื่อมต่อระหว่าง ADC0820 กับ เอฟพีจีเอ

MCP3201 เป็นไอซีแปลงอะนาลอกเป็นดิจิตอลชนิด Successive Approximation ใช้เวลาในการแปลงสัญญาณสูงสุด 12 สัญญาณนาฬิกาโดยสัญญาณนาฬิกาที่สามารถตอบสนองได้สูงสุดคือ 1.6 เมกะเฮิร์ตซ์ ใช้โปรโตคอลในการดึงข้อมูลแบบ SPI มีขาที่ใช้ต่อเป็นสัญญาณควบคุม 3 ขาประกอบด้วย

- DCLK เป็นขาชนิดอินพุตทำหน้าที่รับสัญญาณนาฬิกาเพื่อควบคุมการทำงานของชิปโดยสัญญาณนาฬิกาสูงสุดต้องไม่เกิน 1.6 เมกะเฮริตซ์

- $\overline{CS}$ ขานี้เป็นชนิดอินพุตทำงานเมื่อเป็นลอจิก “0” ทำหน้าที่อีนาเบิลชิปให้ทำงาน ซึ่งไอซีจะเริ่มแปลงข้อมูลเมื่อได้รับสัญญาณที่ขานี้
- DOUT เป็นขาชนิดเอาท์พุตทำหน้าที่ในการส่งข้อมูลแบบอนุกรม โดยก่อนส่งจะทำให้ขานี้เป็น “0” ก่อนแล้วจึงเริ่มปล่อยข้อมูลออกมาทีละบิตตามจังหวะของสัญญาณนาฬิกาที่ขา DCLK จาก MSB ถึง LSB

นอกจากขาควบคุมที่กล่าวไปแล้วนี้ยังมีขาอื่นๆซึ่งต่อร่วมในวงจรด้วยกล่าวคือ VREF เป็นขาสำหรับต่อกับแรงดันอ้างอิงและ IN+ กับ IN- ซึ่งเป็นขาสำหรับต่อกับสัญญาณขาเข้า รูปที่ 4.5 แสดงวงจรการเชื่อมต่อวงจรระหว่าง MCP3201 และ ชิปเอฟพีจีเอ

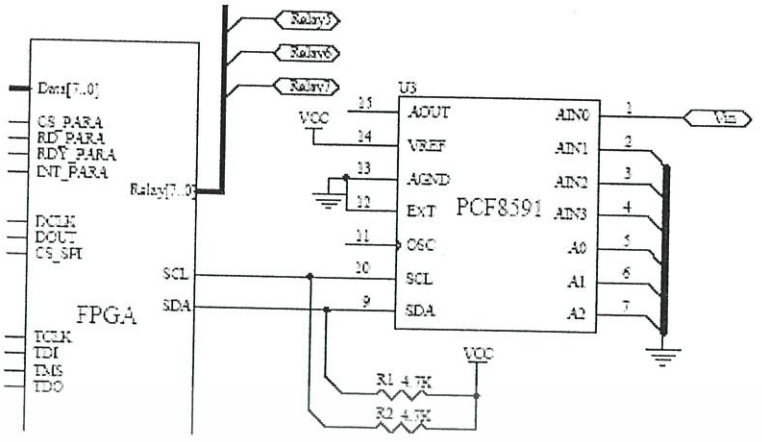


รูปที่ 4.5 วงจรเชื่อมต่อระหว่าง MCP3201 กับ เอฟพีจีเอ

PCF8591 เป็นไอซีแปลงอะนาลอกเป็นดิจิตอลชนิด Successive Approximation ขนาดสี่ช่องสัญญาณสามารถทำหน้าที่ในการแปลงสัญญาณดิจิตอลเป็นอะนาลอก (DAC) ได้ด้วยใช้เวลาในการแปลงสัญญาณสูงสุด 90 ไมโครวินาที และใช้โปรโตคอลในการดึงข้อมูลแบบ I^2C มีขาที่ใช้ต่อเป็นสัญญาณควบคุม 2 ขาซึ่งต่อร่วมกับชิปเอฟพีจีประกอบด้วย

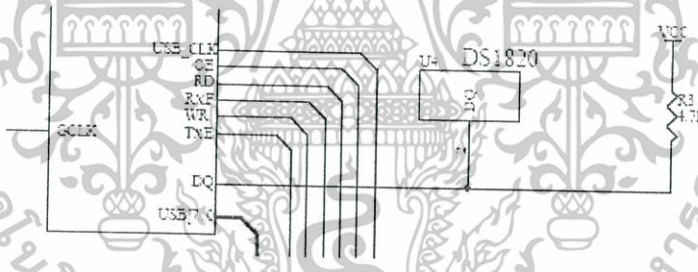
- SCL (Serial Clock Line) เป็นขาชนิดอินพุตทำหน้าที่กำหนดจังหวะการทำงานในสถานะต่างๆตามมาตรฐาน I^2C
- SDA (Serial Data Line) เป็นขาชนิดสองทิศทางทำหน้าที่ในการรับ-ส่งข้อมูลในระดับบิตหรือสร้างสถานะเริ่ม-หยุดการทำงานในมาตรฐาน I^2C

นอกจากขาควบคุมที่กล่าวไปแล้วนี้ยังมีขาอื่นๆซึ่งต่อร่วมในวงจรด้วยเช่น Vref, AIN0-AIN3 ซึ่งเป็นขาสำหรับต่อกับแรงดันอ้างอิงและแรงดันอินพุตในแต่ละช่องสัญญาณ ตามลำดับ และ A0-A2 ทำหน้าที่ในการกำหนดแอดเดรสของชิป (ในกรณีมีชิปหลายตัวต่อในบัสเดียวกัน) รูปที่ 4.6 แสดงวงจรการเชื่อมต่อวงจรระหว่าง PCF8591 และ ชิปเอฟพีจีเอ สำหรับ R1 และ R2 ทำหน้าที่เป็นตัวต้านทานพูลอัพเพื่อให้สัญญาณมีลอจิกสูงขณะไม่ได้ใช้งาน



รูปที่ 4.6 วงจรเชื่อมต่อระหว่าง PCF8591 กับเอฟพีจีเอ

DS1820 เป็นไอซีที่มีชุดเซนเซอร์แปลงอุณหภูมิอยู่ภายในหรือโดยภาพรวมคือชิปไอซีที่ทำหน้าที่ในการแปลงอุณหภูมิเป็นสัญญาณดิจิตอลนั่นเอง การดึงข้อมูลจากไอซีตัวนี้กระทำผ่านทางโปรโตคอล 1-Wire ดังนั้น ในการต่อร่วมกับชิปเอฟพีจีเอจึงใช้พอร์ตเพียงพอร์ตเดียวในการเชื่อมต่อ ดังรูปที่ 4.7 แสดงวงจรเชื่อมต่อระหว่าง DS1820 กับ เอฟพีจีเอ สำหรับ R1 และ R2 ทำหน้าที่เป็นตัวต้านทานพูลอัพเพื่อให้สัญญาณมีลอจิกสูงขณะไม่ได้ใช้งาน



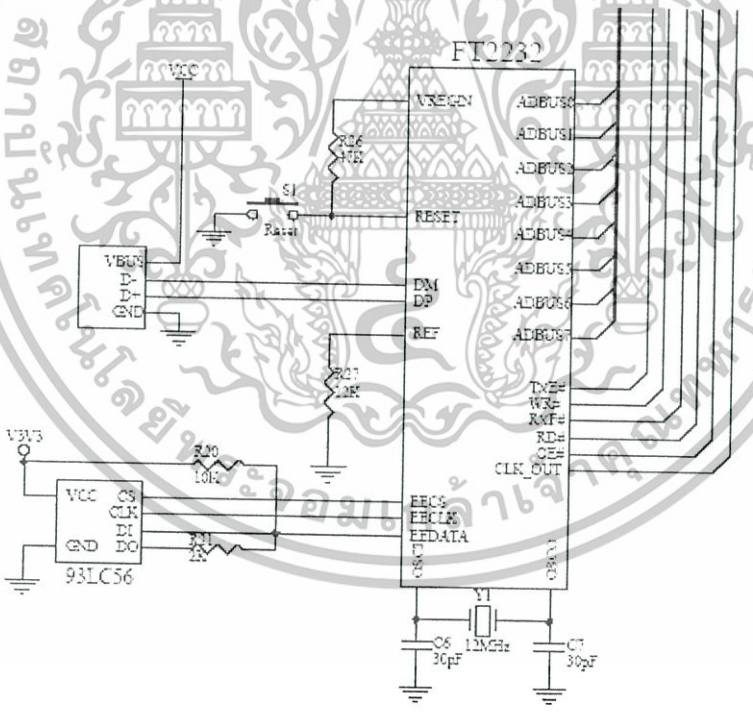
รูปที่ 4.7 วงจรเชื่อมต่อระหว่าง DS1820 กับเอฟพีจีเอ

4.2.3 ภาคยูเอสบีโฮสคอนโทรลเลอร์

หัวใจสำคัญของภาคนี้คือชิปไอซี FT2232 ซึ่งทำหน้าที่จัดการโครงสร้างข้อมูลในระดับล่างของการสื่อสารผ่านพอร์ตยูเอสบี การเชื่อมต่อระหว่างเอฟพีจีเอกับชิปตัวนี้จะใช้การเชื่อมต่อในโหมด FIFO Synchronous ซึ่งเป็นการเชื่อมต่อในโหมดความเร็วเต็มขั้น (Full Speed) ที่มีขาเชื่อมต่อในการควบคุม 6 ขาและเป็นบัสขนาด 8 บิต อีกหนึ่งบัสซึ่งมีรายละเอียดดังนี้

- ADBUS0-ADBUS7 เป็นบัสข้อมูลชนิดสองทิศทางทำหน้าที่ในการรับ-ส่งข้อมูลกับชิปเอฟพีจีเอ
- TXE# เป็นขาชนิดเอาท์พุตทำหน้าที่แจ้งสถานะการส่งข้อมูลจากเอฟพีจีเอมายัง FT2232 โดยหากขานี้เป็น “1” แสดงว่าการส่งข้อมูลยังไม่แล้วเสร็จ
- WR เป็นขาชนิดอินพุตทำงานร่วมกับขา TXE# โดยเมื่อขานี้เป็น “1” FT2232 จะอ่านข้อมูลจากบัสเพื่อส่งต่อให้กับดีไวซไดร์เวอร์

- RXF# เป็นขาชนิดเอาท์พุตทำหน้าที่แจ้งสถานะให้ชิปเอฟพีจีเออ่านข้อมูลโดยขานี้จะเป็น “0” เมื่อมีข้อมูลในบัฟเฟอร์อย่างน้อย 1 ไบต์
- RD เป็นขาชนิดอินพุตทำงานร่วมกับขา RXF# เมื่อขานี้เป็น “0” FT2232 จะถ่ายข้อมูลจากบัฟเฟอร์มายังบัสเพื่อให้ชิปเอฟพีจีเออ่านออกไป
- OE# เป็นขาเพื่อเปลี่ยนสถานะของบัสให้เป็นชนิดอินพุตหรือเอาท์พุต โดยเมื่อขานี้เป็น “0” บัสจะมีสถานะเป็นเอาท์พุตสำหรับขานี้จะถูกใช้ในโหมดความเร็วสูง (High Speed) เท่านั้น
- DM เป็นขาชนิด 2 ทิศทาง ทำหน้าที่เชื่อมต่อกับสายสัญญาณ D- ของมาตรฐาน USB
- DP เป็นขาชนิด 2 ทิศทาง ทำหน้าที่เชื่อมต่อกับสายสัญญาณ D+ ของมาตรฐาน USB
- CLK_OUT เป็นขาชนิดเอาท์พุตทำหน้าที่ส่งสัญญาณนาฬิกาความถี่ 60 MHz เพื่อกำหนดจังหวะการอ่านหรือเขียนข้อมูลให้กับชิปเอฟพีจีเอ สำหรับขานี้จะถูกใช้ในโหมดความเร็วสูง (High Speed) เท่านั้น



รูปที่ 4.8 วงจรภาคยูเอสบีโฮสคอนโทรลเลอร์

เนื่องจากชิป FT2232 จำเป็นที่ต้องใช้ข้อมูลบางอย่างในการสร้างการเชื่อมต่อกับพอร์ตยูเอสบี เช่น Product description, Max Bus Power, Connection mode เป็นต้น และเพื่อให้ข้อมูลเหล่านี้มีความยืดหยุ่นสามารถเปลี่ยนแปลงได้ ดังนั้น จึงต้องเชื่อมต่อ FT2232 กับ EEPROM เพื่อให้ EEPROM เก็บข้อมูลเหล่านี้ สำหรับ EEPROM ที่ใช้ในการออกแบบนี้คือเบอร์ 93LC56

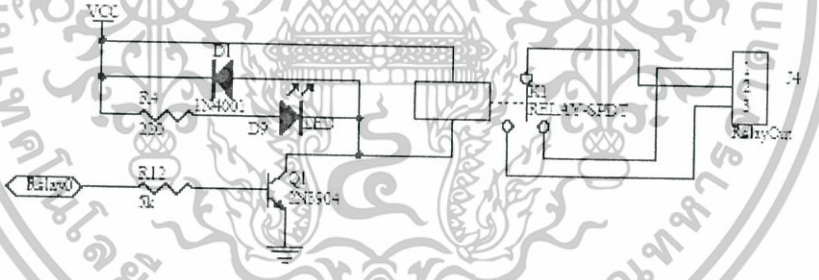
ซึ่งเป็น EEPROM ขนาด 256 × 8 บิท ทำงานแบบอนุกรม ซึ่ง FT2232 ได้ออกแบบให้รองรับกับ EEPROM ชนิดนี้แล้วโดยมีขาที่ต่อใช้งาน 3 ขาประกอบด้วย

- EECS เป็นขาชนิด 2 ทิศทาง ใช้เชื่อมต่อกับขา CS ของ EEPROM
- EECLK เป็นขาชนิดเอาท์พุตทำหน้าที่ส่งสัญญาณนาฬิกา เพื่อกำหนดจังหวะการอ่านหรือเขียนข้อมูลให้กับ EEPROM
- EEDATA เป็นขาชนิด 2 ทิศทางทำหน้าที่รับ-ส่งข้อมูลอนุกรมกับ EEPROM

สำหรับส่วนประกอบอื่นๆในวงจรประกอบด้วยชุดรีเซ็ทไอซีประกอบด้วย S1 และ R26 นอกจากนั้นก็จะเป็นชุดสร้างสัญญาณนาฬิกาซึ่งประกอบด้วย Y1, C6 และ C7 รูปที่ 4.8 แสดงวงจรของภาคยูเอสบีโฮสคอนโทรลเลอร์

4.2.4 ภาคควบคุมรีเลย์

ภาคควบคุมรีเลย์เป็นส่วนที่ทำหน้าที่ตอบสนองต่อคำสั่งจากผู้ใช้เมื่อต้องการควบคุมระบบซึ่งคำสั่งที่ส่งมาเป็นข้อมูลขนาด 1 ไบต์หรือ 8 บิท ดังนั้น ภาคควบคุมรีเลย์จึงมีทั้งหมด 8 ชุด ซึ่งก็คือควบคุมบิตละชุดนั่นเอง สำหรับการอธิบายหลักการทำงานนั้นเนื่องจากว่าทั้ง 8 ชุดมีวงจรการทำงานเหมือนกัน ดังนั้น จึงจะอธิบายการทำงานเพียง 1 ชุดเท่านั้น ดังรูปที่ 4.9 แสดงวงจรภาคควบคุมรีเลย์ 1 ชุด



รูปที่ 4.9 วงจรภาคควบคุมรีเลย์ 1 ชุด

จากรูปที่ 4.9 โพรบ Relay0 (อยู่มุมทางซ้ายมือ) นั้นเป็นโพรบที่เชื่อมต่อกับพอร์ตเอฟพีจีเอซึ่งทำหน้าที่เป็นสัญญาณเอาท์พุตบิทแรก เมื่อบิทนี้เป็น “0” จะทำให้ไม่มีแรงดันตกคร่อมที่ขา B-E ของ Q1 ดังนั้น Q1 จึงไม่นำกระแสทำให้รีเลย์ไม่ทำงาน

เมื่อบิทนี้เป็น “1” จะทำให้มีแรงดันไบอัสให้ Q1 ผ่าน R12 ทำให้ Q1 ทำงาน ซึ่งจะทำหน้าที่คล้ายสวิทช์ดึงให้คอยล์ด้านหนึ่งของรีเลย์ต่อลงกราวด์ทำให้รีเลย์ทำงาน นอกจากนั้น ผลของ Q1 ยังส่งผลให้ขาคาโทดของ LED ต่อลงกราวด์ด้วย ทำให้เกิดกระแสไหลผ่าน R4 ซึ่งทำหน้าที่จำกัดกระแสไม่ให้ไหลผ่านมากเกินไป แล้วผ่าน LED ลงกราวด์ทำให้ LED ติดสว่างด้วย สำหรับ D1 ทำหน้าที่ป้องกันกระแสไหลเกินอันเนื่องมากจากการเริ่มทำงานของคอยล์รีเลย์

4.2.5 ภาคหน่วยประมวลผลกลาง

หน่วยประมวลผลกลางที่ใช้ในการออกแบบนี้คือชิปเอฟพีจีเอเบอร์ EP1K10TC144-3 ของบริษัท Altera ซึ่งเป็นชิปเอพพีจีเอที่มีความจุ 2,880 LE หรือประมาณ 50,000 เกต มีหน่วยความจำภายใน 40,960 บิท และมีขาอินพุต-เอาท์พุตให้สามารถต่อใช้งานได้ถึง 102 ขา สามารถใช้งานกับพอร์ตอินพุต-เอาท์พุตแบบแรงดันไฟฟ้าสามระดับคือ 5โวลต์ 3.3โวลต์และ 2.5 โวลต์ ทำงานที่ความถี่นาฬิกาสูงสุดที่ 180 เมกะเฮิร์ต

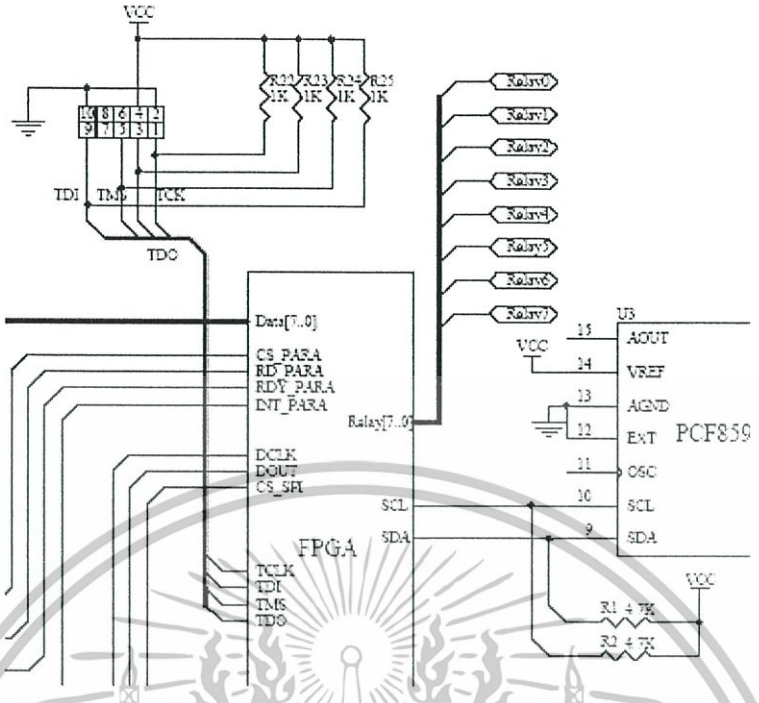
หน้าที่หลักของเอฟพีจีเอคือทำหน้าที่ในการสื่อสารเพื่อดึงข้อมูลจาก ADC แต่ละช่องสัญญาณ จากนั้นจัดลำดับข้อมูล นำข้อมูลมาเข้ารหัสและส่งข้อมูลที่ได้ไปยังคอมพิวเตอร์ผ่านพอร์ตยูเอสบี นอกจากนั้นยังทำหน้าที่ในการขับชุดรีเลย์ตามคำสั่งที่ได้รับมาจากผู้ใช้อีกด้วย สำหรับรายละเอียดของการเชื่อมต่อสัญญาณกับ ADC และชุดรีเลย์นั้นได้กล่าวไปแล้วในตอนต้น บทดังนั้นจะกล่าวเฉพาะส่วนที่ยังไม่กล่าวถึงเท่านั้นคือส่วนที่ทำหน้าที่โปรแกรมลงบนตัวชิป

เมื่อการออกแบบโมดูลต่างๆในชิปเอฟพีจีเอเสร็จสิ้นลงและทำการจำลองการทำงานจนได้ผลเป็นที่น่าพอใจแล้ว ขั้นตอนต่อไปเราจะต้องเอาโมดูลเหล่านี้โปรแกรมไปยังชิปเอฟพีจีเอซึ่งส่วนที่ทำหน้าที่โปรแกรมนั้นจะประกอบด้วยทั้งซอฟท์แวร์และฮาร์ดแวร์ โดยในส่วนซอฟท์แวร์นั้นโปรแกรม Quartus II ซึ่งเป็นซอฟท์แวร์ที่ใช้ในการออกแบบโมดูลเป็นผู้จัดการให้ ด้านฮาร์ดแวร์การโปรแกรมจะส่งถ่ายข้อมูลตามมาตรฐาน JTAG (Joint Test Action Group) โดยมีขาเชื่อมต่อจำนวน 4 ขา ประกอบด้วย

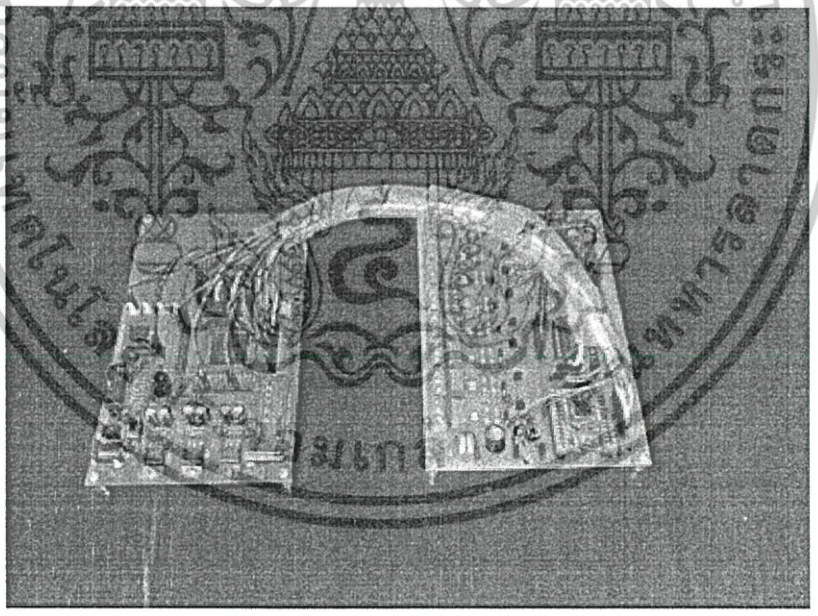
- **TDI** (Test Data In)
- **TDO** (Test Data Out)
- **TCK** (Test Clock)
- **TMS** (Test Mode Select)

รูปที่ 4.10 แสดงวงจรการเชื่อมต่อในส่วนโปรแกรมลงบนตัวชิปตามมาตรฐาน JTAG โดยมีขาสำคัญ 4 ขาดังที่กล่าวข้างบนและมี R22-R24 ทำหน้าที่เป็นตัวต้านทานพูลอัพเพื่อให้ขาทั้งหมดมีสภาวะเป็นลอจิกสูงเมื่อไม่ใช้งาน

เมื่อนำเอาส่วนประกอบของวงจรทั้งหมดมาประกอบรวมกันจะได้เป็นโครงงานโดยสมบูรณ์ดังรูปที่ 4.11



รูปที่ 4.10 วงจรส่วน JTAG



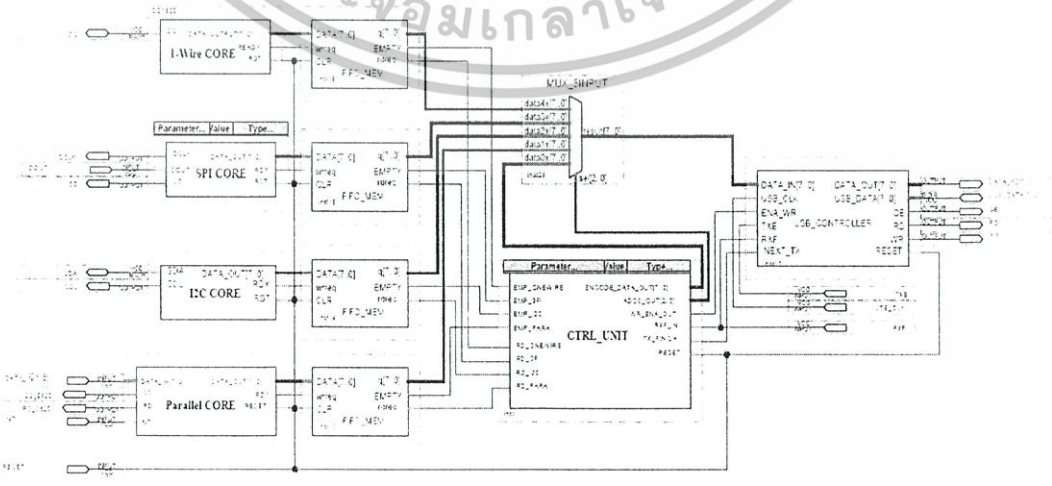
รูปที่ 4.11 รูปถ่ายโครงงานที่สมบูรณ์

บทที่ 5
การออกแบบโมดูลภายในชิปเอฟพีจีเอ

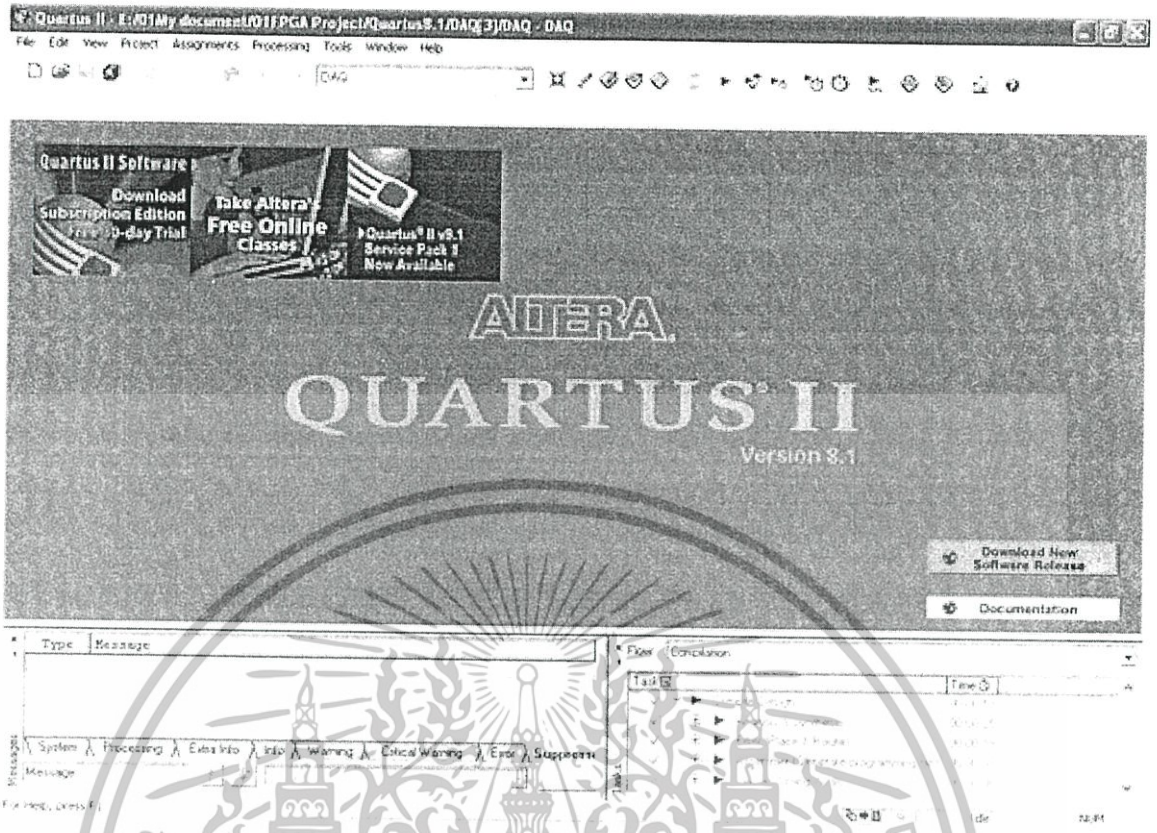
ในการออกแบบระบบจะต้องทำการศึกษาเพื่อให้ทราบถึงกลไกหรือรูปแบบในการดึงข้อมูลจาก ADC (Analog to Digital Converter) ในแต่ละโปรโตคอลที่นำมาใช้ในการออกแบบ รวมทั้งรูปแบบของโปรโตคอลที่ใช้กับไอซีที่ทำหน้าที่จัดการการทำงานของพอร์ตยูเอสบีด้วย ซึ่งรูปแบบดังกล่าวจะถูกนำเอามาอธิบายหรือเขียนเป็นโมดูลในชิปเอฟพีจีเอด้วยภาษา VHDL (Very High Speed Integrated Circuit, Hardware Description Language)

สำหรับหลักการออกแบบชิปเอฟพีจีเอเพื่อทำงานกับระบบนี้นั้นจำเป็นต้องมีโมดูลภายในชิปของแต่ละช่องสัญญาณเพื่อทำหน้าที่ในการดึงข้อมูลจากไอซี ADC มาเก็บไว้ในหน่วยความจำของแต่ละช่องสัญญาณนั้น จากนั้นจึงมีอีกโมดูลซึ่งทำหน้าที่ในการอ่าน, เข้ารหัส, จัดลำดับ และส่งข้อมูลไปยังคอมพิวเตอร์ผ่านทางพอร์ตยูเอสบีและเนื่องจากในการส่งข้อมูลไปยังพอร์ตยูเอสบีจำเป็นต้องสร้างการเชื่อมต่อและสื่อสารกับไอซียูเอสบีโฮสคอนโทรลเลอร์ (FT2232) ดังนั้นจึงจำเป็นต้องมีอีกหนึ่งโมดูลซึ่งทำหน้าที่ดังกล่าวนี้ด้วย และโมดูลนี้นอกจากทำการสื่อสารกับยูเอสบีโฮสคอนโทรลเลอร์แล้วยังต้องออกแบบให้รับคำสั่งจากผู้ใช้เพื่อควบคุมชุดรีเลย์อีกหน้าที่หนึ่งด้วย สำหรับส่วนประกอบทั้งหมดของแต่ละโมดูลภายในชิปเอฟพีจีเอแสดงดังรูปที่ 5.1

ในการทดลองได้ทำการจำลองการทำงานของโมดูลต่างๆที่เกี่ยวข้องเพื่อให้เป็นไปตามรูปแบบที่กำหนดและให้แต่ละโมดูลสามารถทำงานเข้าจังหวะกันได้ โดยซอฟท์แวร์ที่ใช้ในการเขียนโค้ดภาษา VHDL และจำลองการทำงานของทุกโมดูลนั้นใช้โปรแกรม QuartusII เวอร์ชั่น 8.1 ของบริษัท Altera ซึ่งเป็นบริษัทผู้ผลิตชิปเอฟพีจีเอเบอร์ EP1K50TC144-3 ที่ใช้ในการออกแบบระบบนี้ รูปที่ 5.2 แสดงลักษณะโปรแกรม Quartus II 8.1



รูปที่ 5.1 ส่วนประกอบโมดูลภายในชิปเอฟพีจีเอ



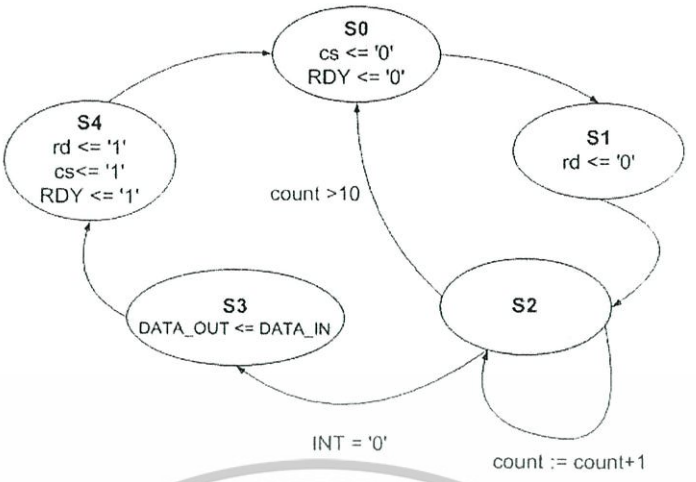
รูปที่ 5.2 โปรแกรม Quartus II 8.1

5.1 ผลการจำลองการทำงานภาคแปลงสัญญาณอะนาลอกเป็นดิจิตอล

จากที่ได้กล่าวไปแล้วว่าพอร์ตอินพุต-เอาท์พุตของเอฟพีจีเอจะเชื่อมต่ออยู่กับบัสข้อมูลและบัสควบคุมของ ADC แต่ละตัวซึ่งจะมีจำนวนพอร์ตที่ใช้ไม่เท่ากันขึ้นอยู่กับไอซีนั้นใช้โปรโตคอลในการเรียกดูข้อมูลแบบใด เช่น ไอซีเบอร์ ADC0820 ซึ่งทำการเชื่อมต่อด้วยโปรโตคอลแบบขนานจะใช้พอร์ตในการใช้งานถึง 12 พอร์ต แต่ในขณะที่ไอซี DS1820 ซึ่งทำการเชื่อมต่อแบบ 1-Wire จะใช้พอร์ตเพียงแค่หนึ่งพอร์ตเท่านั้น สำหรับผลการจำลองการทำงานของโมดูลแปลงอะนาลอกเป็นดิจิตอลทั้งสี่โปรโตคอลสามารถแยกรายละเอียดการเชื่อมต่อได้ดังนี้

5.1.1 โปรโตคอลแบบขนาน (Parallel Protocol)

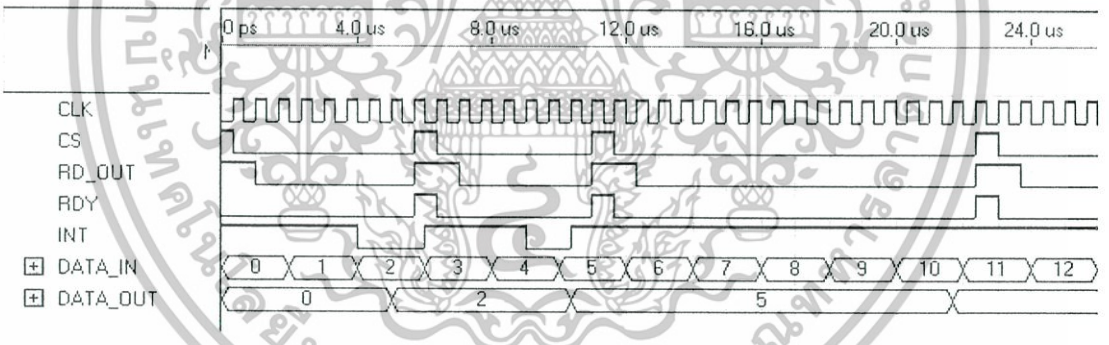
เป็นรูปแบบดั้งเดิมของการใช้งานไอซีแปลงสัญญาณอะนาลอกเป็นสัญญาณดิจิตอลซึ่งมีข้อดีของการทำงานคือให้สัญญาณเอาท์พุตเร็วและง่ายต่อการควบคุม สำหรับการออกแบบนี้ใช้ไอซีเบอร์ ADC0820 ซึ่งสามารถเขียนเป็น State Diagram ได้ดังรูปที่ 5.3



รูปที่ 5.3 State Diagram การทำงานเพื่อสื่อสารกับ ADC0820

โดยขั้นตอนการอ่านจะมีเพียง 2 ขั้นตอนหลักคือ

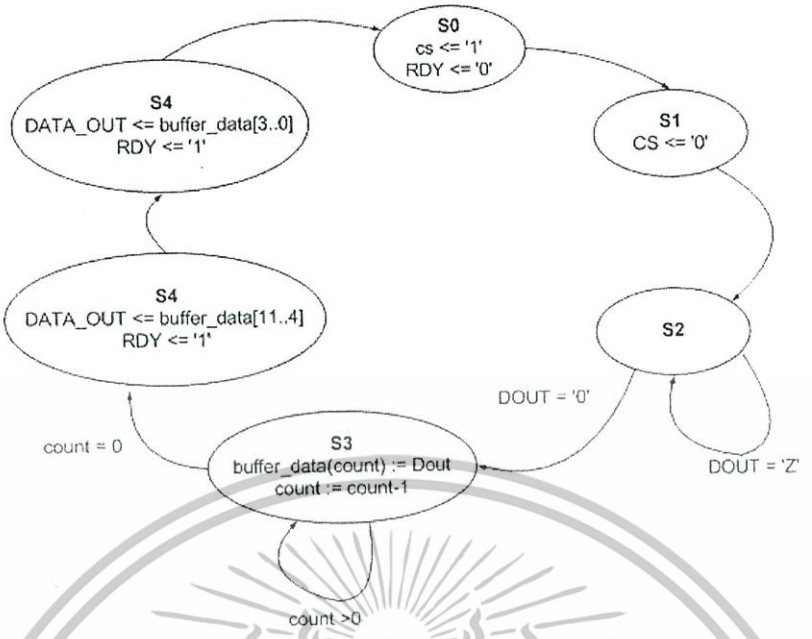
- เอฟพีจีเอส่งสัญญาณเริ่มต้นการทำงานให้กับไอซีจากนั้นรอรับสัญญาณตอบกลับ
- เมื่อไอซีแปลงสัญญาณเสร็จไอซีจะส่งสัญญาณตอบกลับมาแล้วเอฟพีจีเอจึงทำการอ่านข้อมูลเข้ามาทั้ง 8 บิตพร้อมกันและเริ่มรอบการทำงานในรอบต่อไป



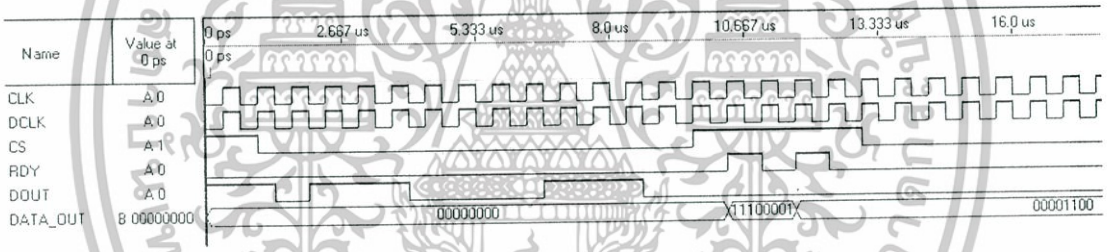
รูปที่ 5.4 ผลการจำลองการทำงานการสื่อสารกับ ADC0820

5.1.2 โปรโตคอลแบบ SPI (Serial Peripheral Interface Protocol)

สำหรับการออกแบบนี้จะใช้ไอซีเบอร์ MCP3201 ในการเชื่อมต่อโดยมีการ State Diagram การทำงานดังรูปที่ 5.5 และมีผลการจำลองการทำงานดังรูปที่ 5.6 โดยขั้นตอนในการได้ข้อมูลมาของโปรโตคอลชนิดนี้จะคล้ายกับแบบขนานคือส่งสัญญาณให้ไอซีเริ่มทำงานและเมื่อไอซีทำงานเสร็จแล้วจะส่งสัญญาณตอบกลับมาเพื่อให้เอฟพีจีเออ่านข้อมูลออกไปแต่วิธีการอ่านของโปรโตคอลนี้จะอ่านข้อมูลแบบอนุกรมผ่านสายสัญญาณเส้นเดียวเรียงลำดับจาก MSB จนกระทั่ง LSB เมื่ออ่านครบก็จะถือว่าเสร็จสิ้นหนึ่งรอบการทำงาน



รูปที่ 5.5 State Diagram การทำงานเพื่อสื่อสารกับ MCP3201



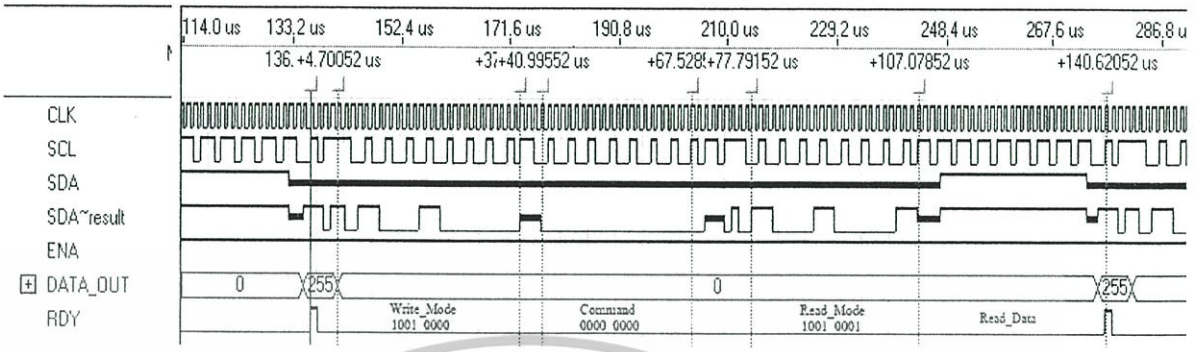
รูปที่ 5.6 ผลการจำลองการทำงานการสื่อสารกับ MCP3201

5.1.3 โปรโตคอลแบบ I^2C (Inter-integrated circuit Protocol)

การสื่อสารแบบ I^2C เป็นอีกโปรโตคอลหนึ่งที่ถูกออกแบบมาให้ง่ายต่อการเชื่อมต่อซึ่งจะใช้สายสัญญาณเพียงสองสายเท่านั้นคือ SCL และ SDA สำหรับในการออกแบบนี้ใช้ไอซีเบอร์ PCF8591 โดยมีรอบการทำงานเป็นขั้นตอนค่อนข้างซับซ้อนดังแสดง State Diagram ในรูปที่ 5.8 และ 5.9 ซึ่งมีรายละเอียดคร่าวๆดังนี้

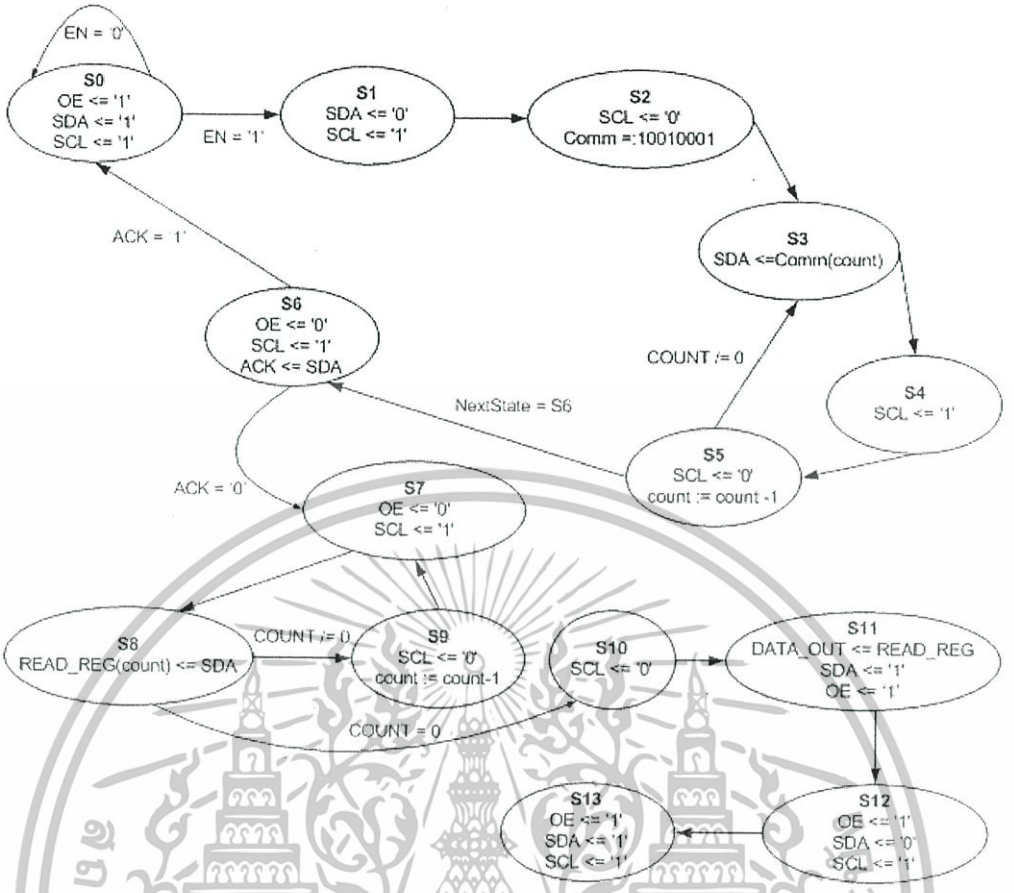
- เอฟพีจีเอส่งสัญญาณเพื่อเริ่มการทำงานของบัสและระบุแอดเดรสของอุปกรณ์ที่ต้องการเชื่อมต่อ
- ระบุโหมดการเขียนและช่องที่ต้องการ
- เอฟพีจีเอส่งสัญญาณเพื่อเริ่มการทำงานของบัสและระบุแอดเดรสของอุปกรณ์ที่ต้องการเชื่อมต่ออีกครั้ง
- ระบุโหมดการอ่านและช่องที่ต้องการ
- อ่านข้อมูลจากบัส I^2C

เมื่อสิ้นสุดการอ่านข้อมูลจากบัสก็ถือเป็นการจบการทำงานในหนึ่งรอบการทำงาน



รูปที่ 5.7 ผลการจำลองการทำงานการสื่อสารกับ PCF8591

รูปที่ 5.8 State Diagram การทำงานเพื่อสื่อสารกับ PCF8591 (ช่วงที่ 1)



รูปที่ 5.9 State Diagram การทำงานเพื่อสื่อสารกับ PCF8591 (ช่วงที่ 2)

5.1.4 โปรโตคอลแบบ 1-Wire (1-Wire Protocol)

การเชื่อมต่อแบบนี้เป็นการเชื่อมต่อด้วยสายสัญญาณน้อยที่สุดคือใช้สายเพียงเส้นเดียวในการสื่อสาร ซึ่งในการออกแบบได้ใช้ไอซี DS1820 ซึ่งเป็นไอซีวัดอุณหภูมิที่ทำการอ่านข้อมูลโดยการใช้โปรโตคอล 1-Wire

การสื่อสารแบบ 1-Wire แบ่งการสื่อสารในแต่ละบิตออกเป็นไทม์สล็อตโดยแต่ละไทม์สล็อตมีความยาวอยู่ในช่วง 60-960 µs ขึ้นอยู่กับชนิดของไทม์สล็อตโดยไทม์สล็อตสำหรับอุปกรณ์มาสเตอร์มีอยู่สี่ฟังก์ชันคือ

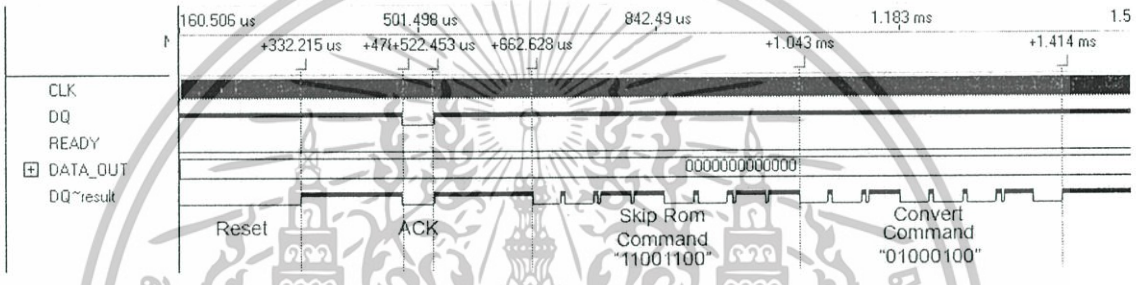
- รีเซ็ต ใช้สำหรับการเริ่มต้นติดต่อกับอุปกรณ์สเลฟ
- การเขียนข้อมูล “1” ไปยังอุปกรณ์สเลฟ
- การเขียนข้อมูล “0” ไปยังอุปกรณ์สเลฟ
- การอ่านข้อมูล

สำหรับขั้นตอนในการสื่อสารนั้นเริ่มจากเอฟพีจีเอส่งสัญญาณรีเซ็ตออกไปและรอการตอบรับจากไอซี เมื่อได้รับการตอบรับแล้วเอฟพีจีเอจะส่งคำสั่งเพื่อระบุแอดเดรสของอุปกรณ์ จากนั้นจะตามด้วยคำสั่งเพื่อสั่งให้ไอซีเริ่มแปลงสัญญาณ

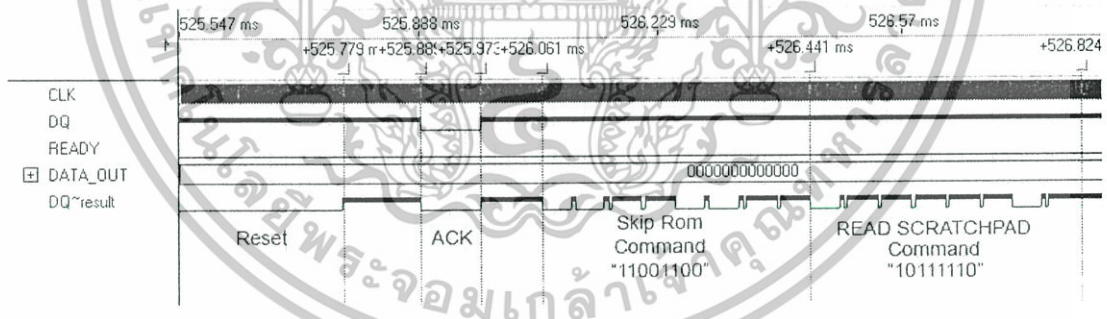
เอฟพีจีเอจะหยุดรอชั่วขณะ เพื่อรอให้ไอซีประมวลผลเสร็จ จากนั้นจึงส่งสัญญาณรีเซ็ต และระบุแอดเดรสไอซีอีกครั้งและตามด้วยคำสั่งเพื่ออ่านข้อมูล จากนั้นเริ่มอ่านข้อมูลจากบัสทีละบิตจนครบแล้วส่งสัญญาณรีเซ็ตออกไป และรอการตอบรับจากอุปกรณ์สเลฟอีกครั้ง

เมื่อสิ้นสุดขั้นตอนนี้ก็ถือเป็นการสิ้นสุดการอ่านข้อมูลจากไอซีในหนึ่งรอบ ถ้าหากต้องการอ่านข้อมูลอีกครั้งก็จะวนรอบการทำงานดังกล่าวอีกครั้ง จากรูปที่ 5.10 ถึง 5.12 เป็นผลการจำลองการทำงานในสภาวะต่างๆ

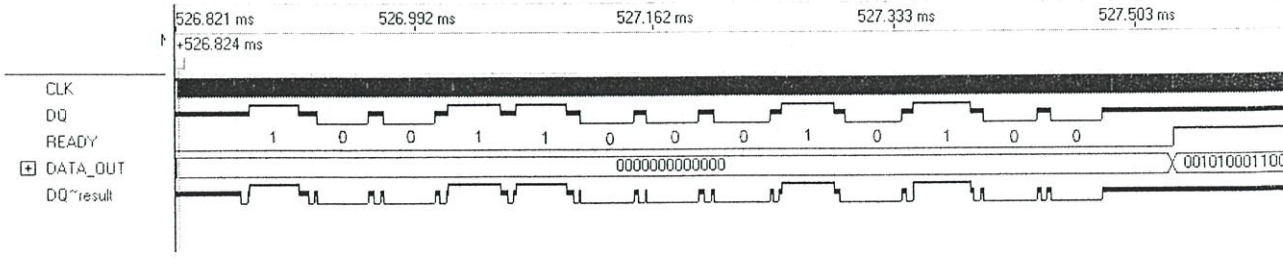
เนื่องจากขั้นตอนการทำงานระหว่างเอฟพีจีเอและ DS1820 มีค่อนข้างมากจึงทำให้ State Diagram มีมากตามไปด้วยดังนั้นจึงไม่ขอกล่าวถึงในที่นี้ โดยจะอ้างอิงในส่วนลำดับการทำงานตามรูปที่ 2.30 ของบทที่ 2 เป็นหลักแทน



รูปที่ 5.10 ผลการจำลองการทำงานการสื่อสารกับ DS1820 ในสภาวะรีเซ็ต สภาวะตอบรับและสภาวะของการเขียนคำสั่ง Skip Rom และ Convert



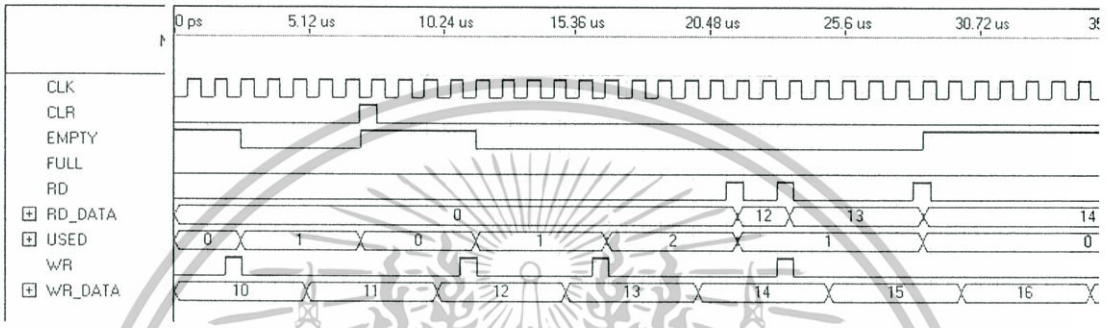
รูปที่ 5.11 ผลการจำลองการทำงานการสื่อสารกับ DS1820 ในสภาวะรีเซ็ต สภาวะตอบรับและสภาวะของการเขียนคำสั่ง Skip Rom และ Read Scratchpad



รูปที่ 5.12 ผลการจำลองการทำงานการสื่อสารกับ DS1820 ในสภาวะอ่านค่าจาก Scratchpad

5.2 หน่วยความจำชนิดเข้าก่อนออกก่อน (First in First out Memory Module)

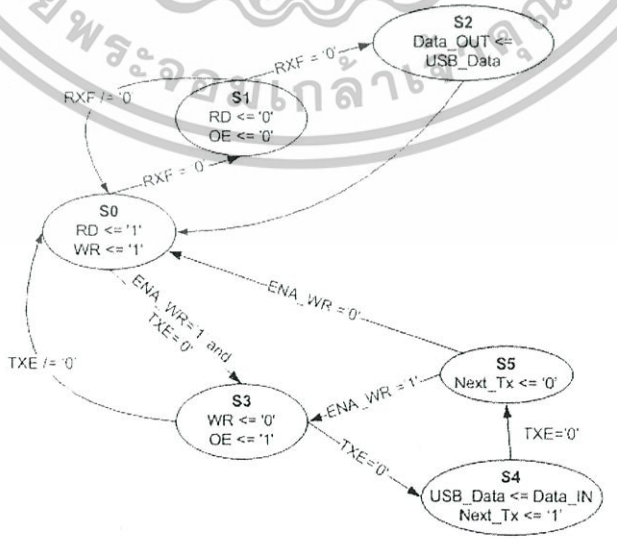
หน่วยความจำ FIFO เป็นหน่วยความจำที่นิยมนำมาใช้เป็นที่พักข้อมูลในการส่งข้อมูลสื่อสารจากที่หนึ่งไปยังอีกทีหนึ่งเนื่องจากมีคุณสมบัติการเก็บข้อมูลเป็นไปตามลำดับคือข้อมูลใดเข้ามาก่อนก็จะถูกอ่านออกไปก่อนทำให้ง่ายในการดึงข้อมูลและไม่เกิดความสับสน สำหรับการออกแบบนี้ FIFO ถูกนำมาใช้พักข้อมูลของแต่ละช่องสัญญาณเพื่อรอการส่งต่อไปยังคอมพิวเตอร์ รูปที่ 5.13 แสดงการจำลองการทำงานการอ่านและเขียนข้อมูลใน FIFO



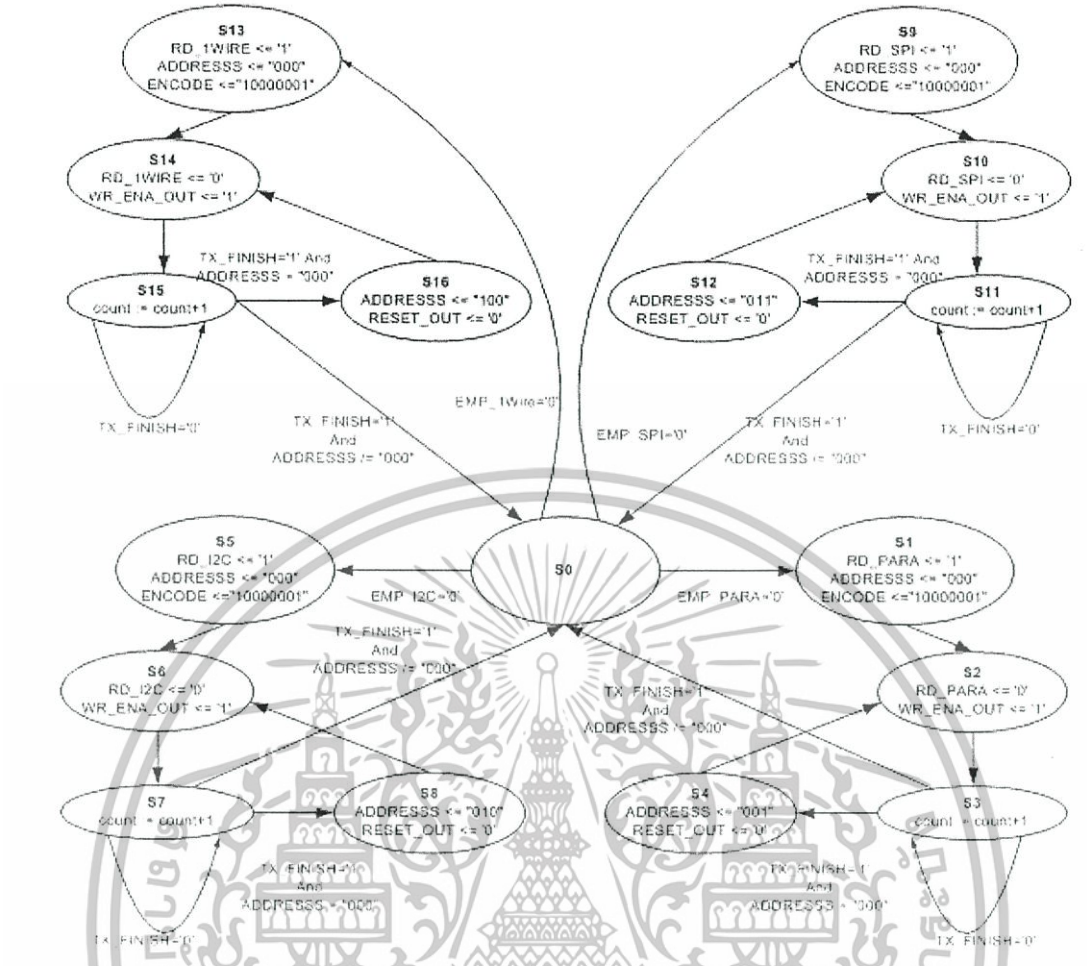
รูปที่ 5.13 จำลองการทำงานการอ่านและเขียนข้อมูล FIFO

5.3 หน่วยควบคุมหลักและหน่วยควบคุมยูเอสบี (Control Unit and USB Controller)

เนื่องจากหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบีเป็นโมดูลที่ทำงานร่วมกันในการลำเลียงข้อมูลไปยังคอมพิวเตอร์ดังนั้นเพื่อความสะดวกในการอธิบายจึงได้นำโมดูลทั้งสองนำมากล่าวไว้ในหัวข้อเดียวกัน รูปที่ 5.14 แสดง State Diagram การทำงานของเพื่ออ่านและเขียนข้อมูลระหว่างเอฟพีจีเอและ FT2232 และรูปที่ 5.15 แสดง State Diagram การทำงานระหว่างหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบี

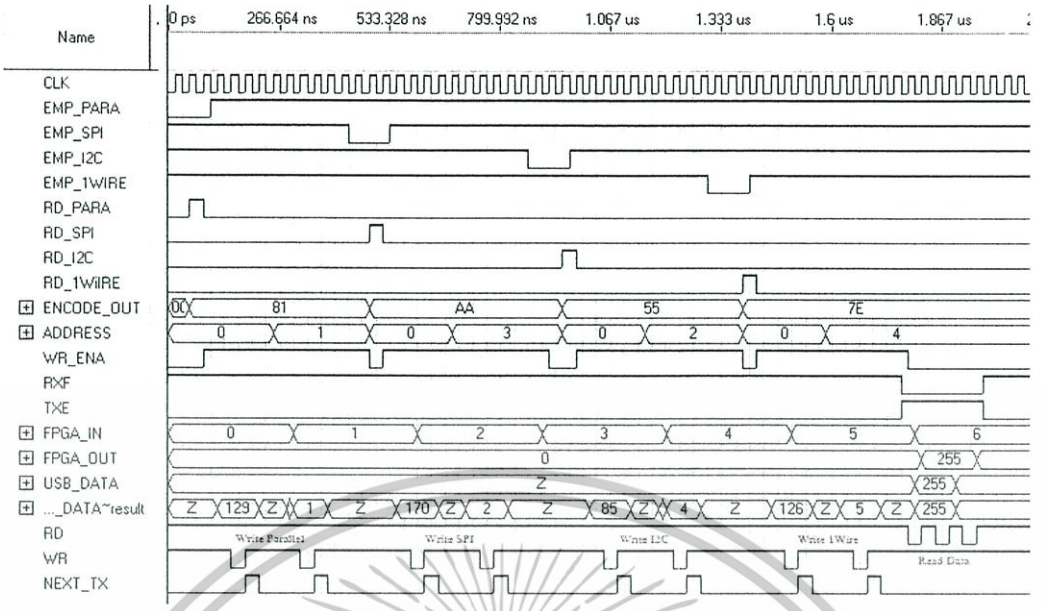


รูปที่ 5.14 State Diagram การทำงานเพื่ออ่านและเขียนข้อมูลกับ FT2232



รูปที่ 5.15 State Diagram การทำงานระหว่างหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบี

เมื่อมีข้อมูลปรากฏขึ้นในหน่วยความจำ(FIFO)ไม่ว่าจะเป็นของโมดูลโปรโตคอลใดก็ตาม หน่วยควบคุมหลักจะทำการตีความว่าเป็นของช่องสัญญาณใด จากนั้นจะส่งสัญญาณไปยังส่วนควบคุมยูเอสบีเพื่อส่งข้อมูลรหัสออกไปยังพอร์ตยูเอสบีหนึ่งไบต์ โดยข้อมูลไบต์แรกที่ส่งไปนี้เพื่อให้ภาครับทราบได้ว่าข้อมูลที่จะส่งตามมานี้เป็นข้อมูลของช่องสัญญาณใด จากนั้น ก็จะส่งสัญญาณไปยังส่วนควบคุมยูเอสบีอีกครั้งเพื่อส่งข้อมูลที่อยู่ในหน่วยความจำ(FIFO) ออกไปยังพอร์ตยูเอสบีทีละไบต์จนกระทั่งหมด ซึ่งถือเป็นการสิ้นสุดกระบวนการทำงานของหนึ่งช่องสัญญาณ และหากมีข้อมูลจากหน่วยความจำตัวอื่นๆรอส่งอยู่ก็จะไปทำการส่งให้กับหน่วยความจำตัวนั้นต่อไป รูปที่ 5.16 การจำลองการทำงานของหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบี



รูปที่ 5.16 จำลองการทำงานของหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบี

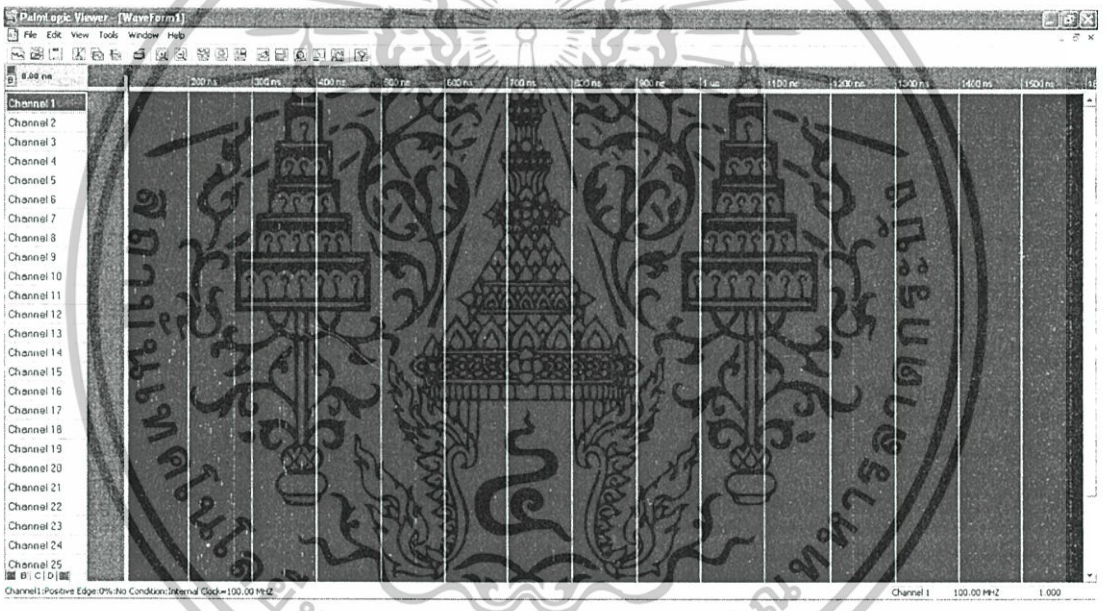
จากรูปที่ 5.16 เป็นผลการจำลองการทำงานการส่งข้อมูลจาก FIFO ของแต่ละช่องสัญญาณและการอ่านข้อมูลของชิปเอฟทีดีไอ ในที่นี้จะอธิบายเฉพาะการทำงานของช่องสัญญาณแบบขนานเท่านั้น โดยเริ่มจากเมื่อมีข้อมูลปรากฏขึ้นที่ FIFO บัฟเฟอร์ จะทำให้พอร์ต EMP_PARA มีลอจิกต่ำซึ่งทำให้โมดูลนี้เริ่มทำงานโดยส่งสัญญาณไปอ่านข้อมูลจาก FIFO พร้อมกันนั้นจะกำหนดแอดเดรสให้กับมัลติเพล็กเซอร์เป็นช่องที่ 0 ซึ่งในขณะนี้ ENCODE_DATA เก็บข้อมูล 81H อยู่ โดยข้อมูลนี้จะเป็นรหัสเริ่มต้นของช่องสัญญาณแบบขนาน พร้อมกันนี้ก็จะส่งสัญญาณ WR_ENA ไปให้กับหน่วยควบคุมยูเอสบีเริ่มส่งข้อมูล

หน่วยควบคุมยูเอสบีจะทำการเซ็ตพอร์ต WR เป็นลอจิกต่ำซึ่งพอร์ตนี้จะถูกต่ออยู่กับขาชื่อเดียวกันของไอซี FT2232 เพื่อส่งข้อมูลรหัส 81H นี้ไปยังคอมพิวเตอร์ผ่านพอร์ตยูเอสบี และเมื่อการส่งเสร็จสิ้นโมดูลนี้ก็จะส่งสัญญาณ NEXT_TX กลับมายังโมดูลหน่วยควบคุมหลักเพื่อบอกให้โมดูลเริ่มส่งข้อมูลต่อไป

โมดูลหน่วยควบคุมหลักจะกำหนดแอดเดรสใหม่ให้กับมัลติเพล็กเซอร์เป็นช่องที่ 1 ซึ่งเป็นบัสข้อมูลเอาท์พุตของ FIFO ในช่องสัญญาณแบบขนานและได้ถูกอ่านออกมารอไว้ก่อนหน้านี้แล้วในที่นี้ข้อมูลบนบัสคือ 51_{10} ทำให้ข้อมูลนี้มาปรากฏที่พอร์ต FPGA_IN จากนั้นหน่วยควบคุมยูเอสบีจะทำการเซ็ตพอร์ต WR เป็นลอจิกต่ำอีกครั้งเพื่อทำการส่งข้อมูลไปยังคอมพิวเตอร์ และเช่นกันเมื่อการส่งเสร็จสิ้นโมดูลนี้ก็จะส่งสัญญาณ NEXT_TX กลับมายังโมดูลหน่วยควบคุมหลักเพื่อบอกให้โมดูลเริ่มส่งข้อมูลต่อไป (ถ้ามี) กลับมาอีกครั้ง

บทที่ 6
ผลการทดลอง

ในการทดลองได้ทำการทดลองการทำงานของโมดูลต่างๆที่เกี่ยวข้องเพื่อให้เป็นไปตามรูปแบบที่กำหนดและให้แต่ละโมดูลสามารถทำงานเข้าจังหวะกันได้ โดยลอจิกอะนาไลเซอร์ที่ใช้วัดเป็นรุ่น Palm Logic ของบริษัท Design Gateway ซึ่งเป็นลอจิกอะนาไลเซอร์ขนาด 32 ช่อง ความถี่ในการสุ่มตัวอย่างสูงสุด 100 MHz มีการแสดงผลผ่านทางซอฟท์แวร์ที่หน้าจอคอมพิวเตอร์ รูปที่ 6.1 แสดงลักษณะโปรแกรมแสดงผลของลอจิกอะนาไลเซอร์รุ่นนี้



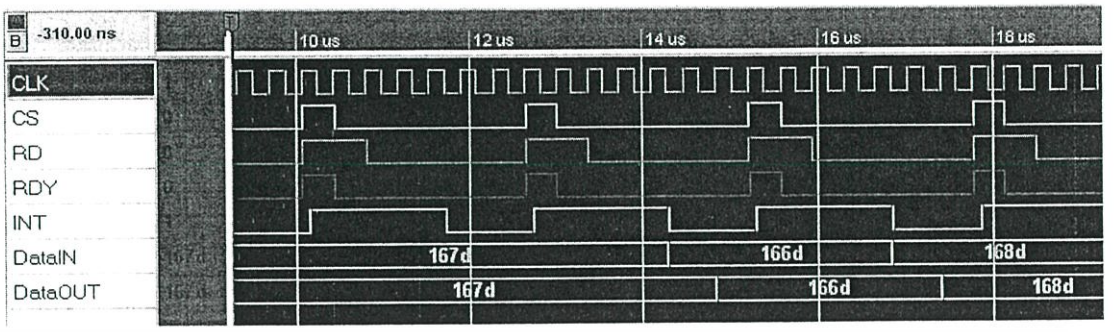
รูปที่ 6.1 โปรแกรม Palm Logic Analyzer

6.1 ผลการทดลองภาคแปลงสัญญาณอะนาลอกเป็นดิจิตอล

สำหรับผลการทดลองของไอซีแปลงอะนาลอกเป็นดิจิตอลทั้งสี่โปรโตคอลสามารถแยกรายละเอียดการเชื่อมต่อได้ดังนี้

6.1.1 โปรโตคอลแบบขนาน (Parallel Protocol)

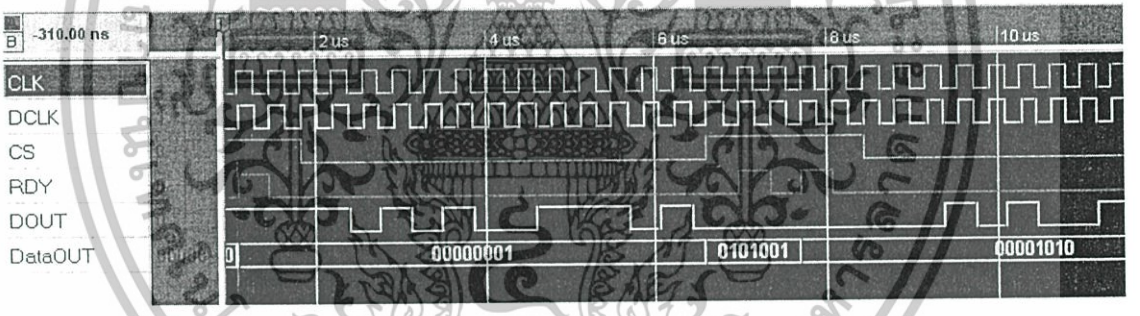
ในการทดลองนี้ทำการทดลองที่สัญญาณนาฬิกา 1.5 MHz สัญญาณอินพุตเป็นข้อมูลเลขฐานสิบส่งออกจากชิปเอฟพีจีเอซึ่งมีผลการทดลอง ดังรูปที่ 6.2 จะเห็นว่าในหนึ่งรอบการทำงานใช้สัญญาณนาฬิกา 7 ลูกหรือใช้เวลาการทำงานประมาณ 4.67 ไมโครวินาที สำหรับในโปรเซสนี้ใช้ 521 ลอจิกเกทในการสร้างวงจร



รูปที่ 6.2 ผลการวัดสัญญาณการทำงานการสื่อสารกับ ADC0820

6.1.2 โปรโตคอลแบบ SPI (Serial Peripheral Interface Protocol)

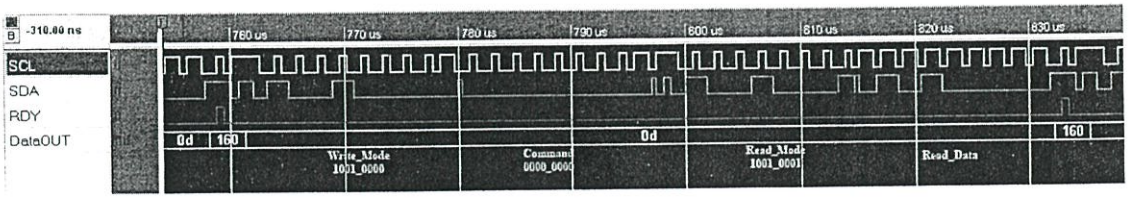
การออกแบบนี้จะใช้ไอซีเบอร์ MCP3201 ในการเชื่อมต่อโดยทำการทดลองที่สัญญาณนาฬิกา 1.5 MHz กำหนดมีผลการทดลองดังรูปที่ 6.3 โดยจะเห็นว่าในหนึ่งรอบการทำงานใช้สัญญาณนาฬิกา 18 ลูกหรือใช้เวลาการทำงานประมาณ 12 ไมโครวินาที สำหรับในโปรเซสนี้ใช้ 746 ลอจิกเกทในการสร้างวงจร



รูปที่ 6.3 ผลวัดสัญญาณการทำงานการสื่อสารกับ MCP3201

6.1.3 โปรโตคอลแบบ I²C (Inter-integrated circuit Protocol)

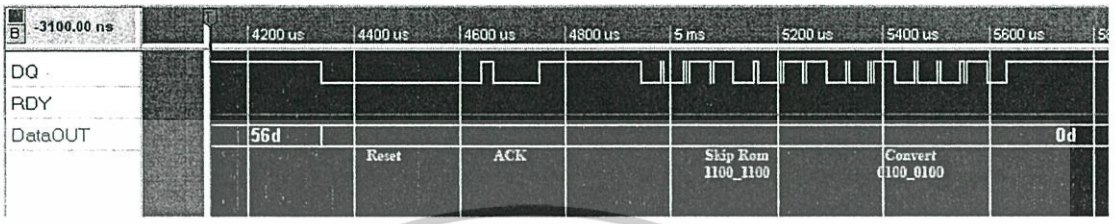
ในการทดลองนี้ทำการทดลองที่สัญญาณนาฬิกา 0.83 MHz มีผลการทดลอง ดังรูปที่ 6.4 จะเห็นว่าในหนึ่งรอบการทำงานใช้สัญญาณนาฬิกา 116 ลูกหรือใช้เวลาการทำงานประมาณ 139 ไมโครวินาที สำหรับในโปรเซสนี้ใช้ 2187 ลอจิกเกทในการสร้างวงจร



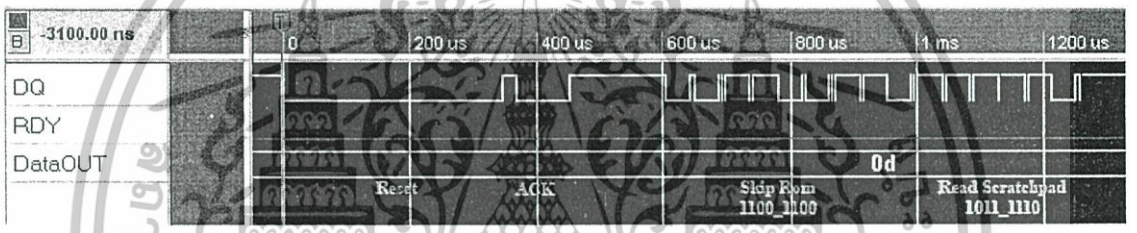
รูปที่ 6.4 ผลวัดสัญญาณการทำงานการสื่อสารกับ PCF8591

6.1.4 โปรโตคอลแบบ 1-Wire (1-Wire Protocol)

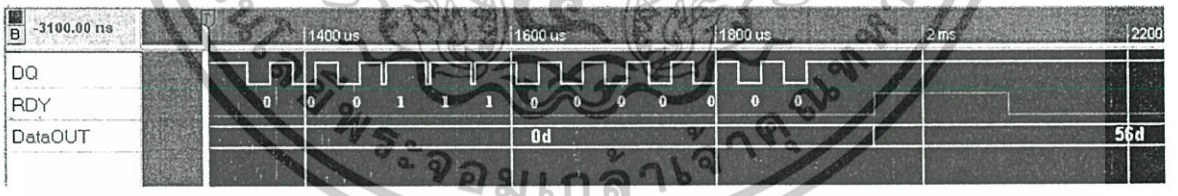
การทดลองนี้ทำการทดลองที่สัญญาณนาฬิกา 1.5 MHz ได้ผลการทดลองดังรูปที่ 6.5 ถึง 6.7 เป็นผลการจำลองการทำงานในสภาวะต่างๆ สำหรับในโปรเซสนี้ใช้ 3,889 ลอจิกเกทในการสร้างวงจร



รูปที่ 6.5ผลวัดสัญญาณการทำงานการสื่อสารกับ DS1820 ในสภาวะรีเซ็ต สภาวะตอบรับและสภาวะของการเขียนคำสั่ง Skip Rom และ Convert



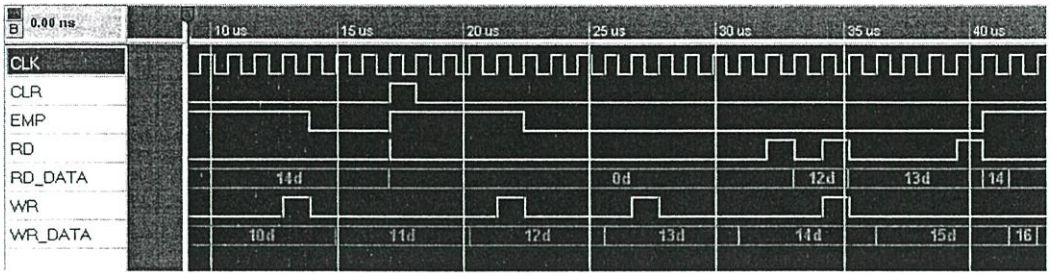
รูปที่ 6.6 ผลวัดสัญญาณการทำงานการสื่อสารกับ DS1820 ในสภาวะรีเซ็ต สภาวะตอบรับและสภาวะของการเขียนคำสั่ง Skip Rom และ Read Scratchpad



รูปที่ 6.7 ผลวัดสัญญาณการทำงานการสื่อสารกับ DS1820 ในสภาวะอ่านค่าจาก Scratchpad

6.2 หน่วยความจำชนิดเข้าก่อนออกก่อน (First in First out Memory Module)

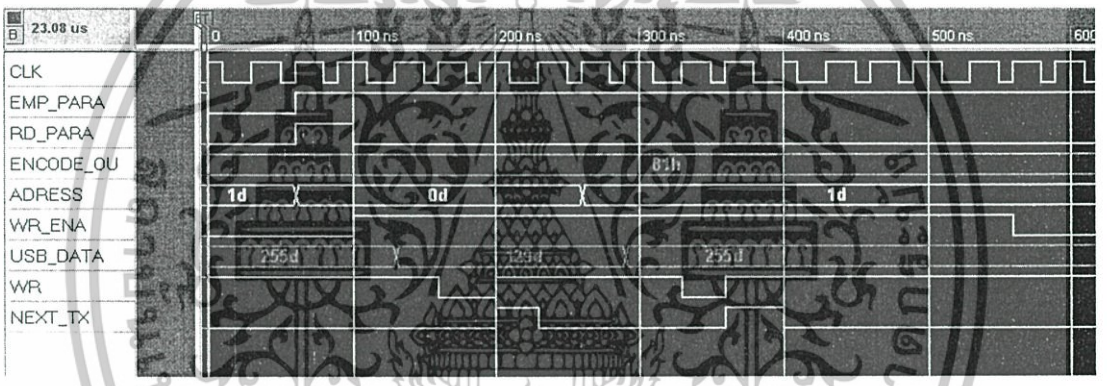
ในการทดลองนี้ทำการทดลองที่สัญญาณนาฬิกา 1 MHz การอ่านหรือเขียนข้อมูลกับโมดูลนี้สามารถทำเสร็จได้ภายใน 2 สัญญาณนาฬิกาคือ กำหนดสัญญาณควบคุมที่ขา RD และ WR เป็นลอจิกสูงตามแต่ว่าต้องการอ่านหรือเขียนจากนั้นรอสัญญาณนาฬิกาทริกที่ขอบขาขึ้นเพื่อทำการอ่านหรือเขียนตามต้องการ สำหรับในโปรเซสนี้ใช้ 712 ลอจิกเกทในการสร้างวงจร



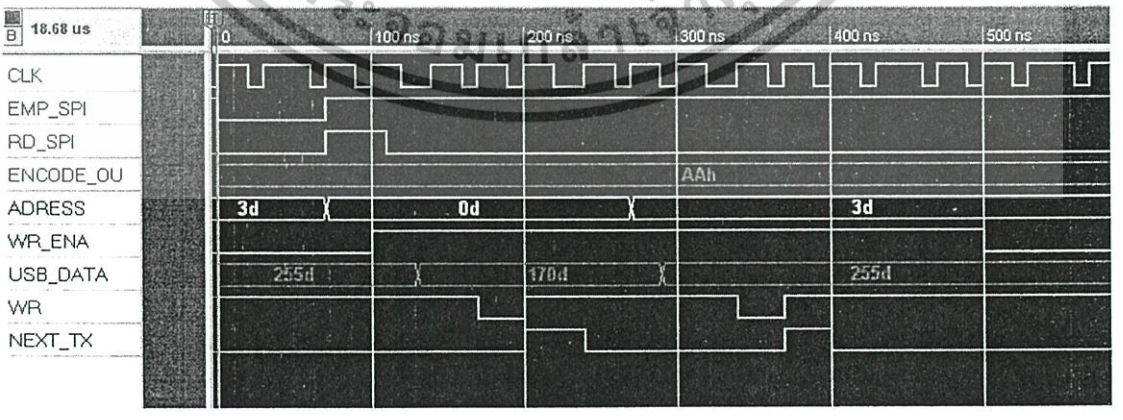
รูปที่ 6.8 ผลวัดสัญญาณการทำงานการอ่านและเขียนข้อมูล FIFO

6.3 หน่วยควบคุมหลักและหน่วยควบคุมยูเอสบี (Control Unit and USB Controller)

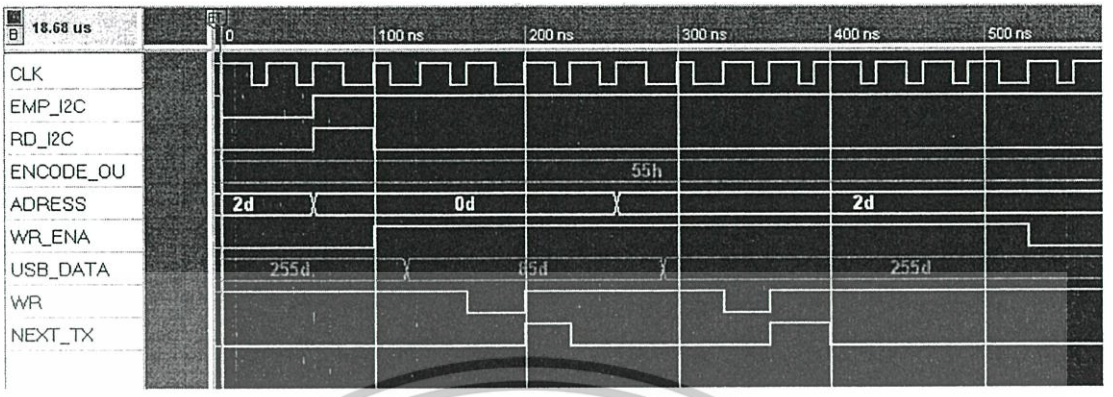
รูปที่ 6.9-6.12 แสดงผลการวัดสัญญาณการทำงานเพื่อเขียนข้อมูลในแต่ละช่องสัญญาณ และรูปที่ 6.13 แสดงผลการวัดสัญญาณการทำงานเพื่ออ่านข้อมูล



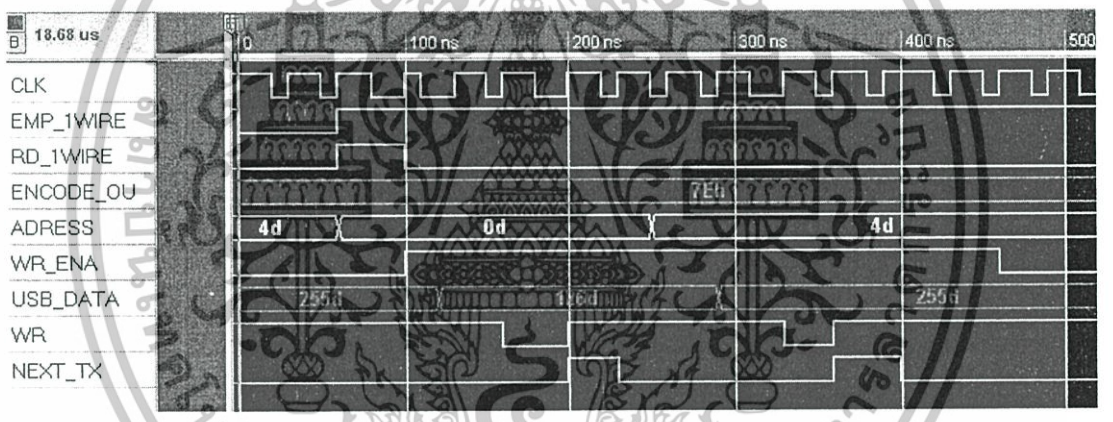
รูปที่ 6.9 ผลการวัดสัญญาณหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบีเพื่อเขียนข้อมูลในช่องสัญญาณ Parallel



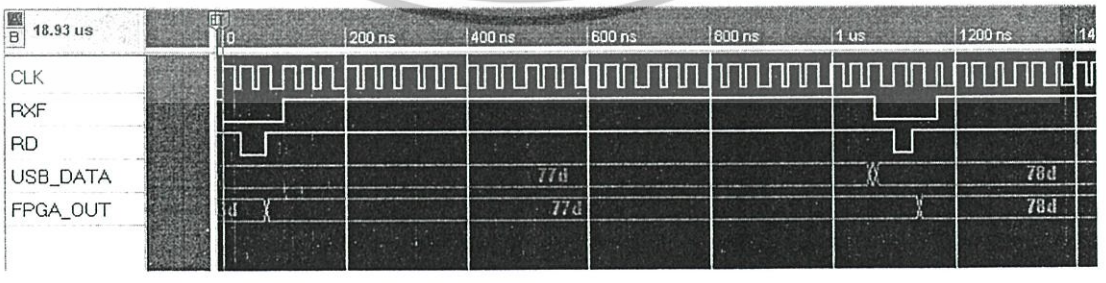
รูปที่ 6.10 ผลการวัดสัญญาณหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบีเพื่อเขียนข้อมูลในช่องสัญญาณ SPI



รูปที่ 6.11 ผลการวัดสัญญาณหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบีเพื่อเขียนข้อมูลในช่องสัญญาณ I2C



รูปที่ 6.12 ผลการวัดสัญญาณหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบีเพื่อเขียนข้อมูลในช่องสัญญาณ 1-WIRE



รูปที่ 6.13 ผลการวัดสัญญาณหน่วยควบคุมหลักและหน่วยควบคุมยูเอสบีเพื่ออ่านข้อมูล

สำหรับโมดูลทั้งสองนี้ทำการทดลองที่สัญญาณนาฬิกา 30 MHz สัญญาณ จะเห็นว่าในหนึ่งรอบการทำงานใช้สัญญาณนาฬิกา 11 ลูกหรือใช้เวลาการทำงานประมาณ 360 นาโนวินาที สำหรับในโปรเซสนี้ใช้ 2,621 ลอจิกเกทในการสร้างวงจรของโมดูลหน่วยควบคุมหลัก (Control Unit) และ 694 ลอจิกเกทในการสร้างวงจรของโมดูลหน่วยควบคุมยูเอสบี (USB Controller)

จากที่ได้กล่าวถึงรายละเอียดของแต่ละโมดูลที่อยู่ภายในชิปแล้ว จะเห็นว่ารายละเอียดการใช้ทรัพยากรในแต่ละโมดูลจะมีความแตกต่างกันไปขึ้นอยู่กับหน้าที่และความซับซ้อนในการทำงานซึ่งพอจะสรุปได้ดังตารางที่ 6.1

Name	Logic Element	Gate	Used (%)	Pin
1-Wire	224	3889	8	1
SPI	43	746	1	3
I2C	126	2187	4	2
Parallel	30	521	1	11
USB Controller	40	694	1	22
Control Unit	151	2621	5	-
FIFO (x4)	164	2847	6	-
Other	60	1042	2	1(Clk)
Total	838	14548	29	40

ตาราง 6.1 สรุปจำนวนทรัพยากรของชิปเอฟพีจีเอที่ใช้ในการออกแบบ

จากตารางที่ 6.1 จะเห็นว่าโมดูลที่ใช้ลอจิกเกตในการสร้างวงจรมากที่สุดคือโมดูลที่ใช้สร้างการสื่อสารในโปรโตคอล 1- Wire เนื่องจากเป็นโปรโตคอลที่ใช้สายสัญญาณควบคุมเพียงเส้นเดียวทำให้การได้มาซึ่งข้อมูลหนึ่งค่าจะใช้ขั้นตอนหลายขั้นตอนและในแต่ละขั้นตอนก็มีการเก็บสถานะของรีจีสเตอร์ของสเตทปัจจุบันเพื่อใช้เป็นค่าอินพุตของสเตทต่อไป ซึ่งเป็นหลักการของ RTL (Register Transfer Level) และ รูปแบบการเขียนโปรแกรม FSM แบบ Mealy Model และเมื่อมองในทางตรงกันข้ามกับโมดูลที่ใช้การสื่อสารในโปรโตคอลแบบ Parallel ซึ่งใช้ลอจิกเกตในการสร้างวงจรน้อยที่สุดแต่กลับใช้จำนวนขาเชื่อมต่อเพื่อการควบคุมมากถึง 12 ขาในการส่งถ่ายข้อมูล ดังนั้นจึงสรุปได้ว่าในการสร้างการสื่อสารให้กับแต่ละโปรโตคอลนั้น ยิ่งจำนวนขาควบคุมน้อยจะยิ่งทำให้ความซับซ้อนในการสร้างการสื่อสารมากขึ้น

จากที่ได้ทำการเชื่อมต่อ ADC ไอซีจากหลากหลายโปรโตคอลมาใช้งานร่วมระบบเดียวกันแล้วทำให้สามารถเปรียบเทียบข้อดีและข้อเสียของแต่ละโปรโตคอลได้ดังตารางที่ 6.2 ซึ่ง

จะเห็นว่าโปรโตคอลแบบขนานมีความเร็วในการทำงานมากที่สุดแต่จะมีข้อเสียคือใช้จำนวนพอร์ตเชื่อมต่อมากทำให้วงจรมีขนาดใหญ่

Item	Parallel	SPI	I2C	1-Wire
Sample Time(us)	4.67	12	139	530ms
Sample Rate(Ksps)	214	83.3	7	0.002
Bit-Rate(kbps)	1,710	1,000	56	0.016
Connection Port	11	3	2	1
Acquire Solution	ง่าย	ง่าย	ยุ่งยาก	ยุ่งยาก
Power (mW)	82.5	2.8	6.3	7.8

ตาราง 6.2 เปรียบเทียบคุณสมบัติของแต่ละโปรโตคอล

ในทางกลับกันเมื่อเปรียบเทียบกับการเชื่อมต่อด้วยโปรโตคอล I^2C หรือ 1-Wire ซึ่งใช้พอร์ตเชื่อมต่อเพียงแค่ 2 พอร์ตและ 1 พอร์ต ตามลำดับเท่านั้น แต่ความเร็วในการทำงานจะช้ากว่าไอซี ADC โดยทั่วไปมาก ดังนั้น ในการนำไปใช้งานจริงจึงขึ้นอยู่กับผู้ใช้จะตัดสินใจเลือกว่าต้องการนำไปใช้กับงานประเภทใด

การสรุปผลการทดลองที่ได้นี้จะแยกออกเป็น 2 ประการ คือ Verification และ Performance โดยการ Verification หรือการเปรียบเทียบกับผลการจำลองการทำงานนั้นเมื่อเปรียบเทียบแล้วพบว่ามีการทำงานที่ใกล้เคียงกันมากจะแตกต่างกันบ้างในจุดที่มีการเปลี่ยนแปลงข้อมูลของบัสจากค่าหนึ่งไปยังอีกค่าหนึ่ง เช่น ในรูปที่ 6.13 โปรดสังเกตสัญญาณที่ปรากฏบน USB_BUS ในขณะที่เปลี่ยนข้อมูลจาก 77 เป็น 78 นั้นจะเห็นว่าในช่วงแรกจะเกิดการกระเพื่อมของสัญญาณทำให้ไม่สามารถตีความได้ว่าข้อมูลที่ได้นั้นคือค่าอะไรซึ่งจะเป็นลักษณะการทำงานปกติในการทำงานความถี่สูงของการส่งข้อมูลแบบขนาน ดังนั้น เพื่อไม่ให้การอ่านข้อมูลผิดพลาดจึงต้องรออ่านข้อมูลในช่วงที่ไม่เกิดการเปลี่ยนแปลงหรือข้อมูลในบัสคงที่แล้วเท่านั้น

สำหรับการพิจารณาในด้าน Performance หรือประสิทธิภาพของแต่ละโมดูลนั้นจะพิจารณาในองค์ประกอบต่างๆเช่น จำนวนเกต, จำนวนพอร์ตที่ใช้, ความเร็ว และ กำลังงานที่ใช้ เป็นต้น ซึ่งสามารถสรุปได้ดังแสดงในตารางที่ 6.1 และ 6.2

6.4 ทดลองการเพิ่มวงจรให้มีความจุเกตเป็นสองเท่าแล้วดูผลด้านความถี่

การทดลองนี้เป็นการทดลองเพื่อให้ทราบว่าหากมีโมดูลในชิปเอฟพีจีเอมากขึ้นอีกเท่าตัวจะมีผลต่อความถี่สูงสุดที่ชิปรองรับได้หรือไม่ จากการทดลองพบว่าการตอบสนองความถี่สูงสุด

ของชิปเอฟพีจีเอจากเดิม 34.72 MHz เหลือ 22.78 MHz ซึ่งทำให้ทราบว่าการเพิ่มขึ้นของจำนวนเกตมีผลทำให้การตอบสนองความถี่ของชิปลดลงแต่ไม่มากนัก หรือไม่ได้ลดลงในแบบเชิงเส้น และเมื่อทดลองนำเอาโมดูลของแต่ละช่องสัญญาณออกทีละโมดูลพบว่าโมดูลช่องสัญญาณ Parallel, SPI, I^2C มีผลต่อการตอบสนองความถี่น้อยมาก ในขณะที่เมื่อนำเอาโมดูลช่องสัญญาณ 1-Wire ออกพบว่าทำให้การตอบสนองความถี่ของชิปเพิ่มขึ้นอย่างเห็นได้ชัดคือประมาณ 72 MHz ทั้งนี้เนื่องจากในโมดูลนี้มี สเตทในการทำงานเป็นจำนวนมากอันเนื่องมาจากขั้นตอนในการดึงข้อมูลด้วยโปรโตคอลชนิดนี้มีขั้นตอนที่ซับซ้อนนั่นเอง

ดังนั้นการพัฒนาระบบในแง่การลดสเตทไดอะแกรมให้น้อยลงของช่องสัญญาณที่มีความซับซ้อนนั้นจึงเป็นอีกด้านหนึ่งในการพัฒนาระบบ โดยออกแบบให้เป็นโมดูลที่ทำงานในลักษณะหน่วยประมวลผลกลางหรือ CPU ซึ่งหมายถึงเป็นการทำงานในแบบบรรทัดคำสั่งคล้ายกับไมโครคอนโทรลเลอร์ ที่มีการทำงานตามคำสั่งหรือโปรแกรมที่ผู้ใช้เขียนขึ้น และเนื่องจากว่าโมดูลประเภทนี้ในขณะทำงานจะใช้สเตทในการทำงานแต่ละคำสั่งไม่มาก ดังนั้นโปรแกรมที่ทำงานอยู่แม้จะมีมากหรือซับซ้อนเพียงใดก็จะมีผลในด้านความเร็วในการทำงานของโปรแกรมเท่านั้นแต่จะไม่มีผลทำให้สเตทไดอะแกรมในการทำงานเพิ่มขึ้นหรือมีผลต่อการตอบสนองความถี่ของชิปเอฟพีจีเอแต่อย่างใด

บทที่ 7
สรุปผลการวิจัย และข้อเสนอแนะ

การออกแบบระบบดาต้าแอคควิซิชั่นของวิทยานิพนธ์ฉบับนี้ได้จำลองระบบหลายช่องสัญญาณที่แต่ละช่องสัญญาณมีโปรโตคอลการเชื่อมต่อกับ ADC แตกต่างกันซึ่งประกอบด้วยโปรโตคอลแบบ Parallel, SPI, I^2C และ 1-Wire จากนั้นใช้ชิปเอฟพีจีเอทำการดึงข้อมูลจาก ADC เหล่านี้แล้วรวบรวมก่อนจะส่งข้อมูลที่ได้ไปยังคอมพิวเตอร์เพื่อแสดงผลและเก็บข้อมูลตามเวลาจริง (Real Time) ดังได้แสดงรายละเอียดของส่วนประกอบต่างๆในบทที่ 3-5 ไปแล้วนั้น สำหรับในบทนี้จะเป็นการสรุปเปรียบเทียบว่าระบบที่ได้ทำการออกแบบนี้ได้ช่วยแก้ปัญหาให้กับระบบเดิมอย่างไร

7.1 เปรียบเทียบกับระบบที่ใช้ไมโครคอนโทรลเลอร์

อ้างอิงจากระบบดาต้าแอคควิซิชั่นที่ทำการออกแบบไว้ก่อนหน้านี้คือ "A Novel Design of an Industrial Data Acquisition System" [1] ซึ่งเป็นระบบดาต้าแอคควิซิชั่นที่ใช้ไมโครคอนโทรลเลอร์ตระกูล AVR เป็นหน่วยประมวลผลกลางด้วยข้อดีคือราคาถูกและใช้งานง่ายของไมโครคอนโทรลเลอร์ตระกูลนี้จึงเป็นที่นิยมนำมาใช้ในระบบดาต้าแอคิวซิชั่นโดยทั่วไปอย่างกว้างขวาง แต่อย่างไรก็ตามหากนำมาประยุกต์ใช้กับระบบดาต้าแอคควิซิชั่นเดียวกันกับที่ออกแบบในวิทยานิพนธ์ฉบับนี้จะให้ผลการทำงานที่แตกต่างกันออกไป

จากตารางที่ 5.2 ในบทที่ 5 ได้ทำการเปรียบเทียบเวลาที่ใช้ในการดึงข้อมูล (Sample Time) จากแต่ละโปรโตคอล โดยเมื่อนำเวลาทั้งหมดมารวมกันจะมีค่าประมาณ 530 มิลลิวินาที หรือมีค่าใกล้เคียงกับเวลาที่ใช้ในการดึงข้อมูลจากโปรโตคอล 1-Wire นั่นเอง ผลรวมเวลาที่ได้มานี้เมื่อพิจารณาในแง่การทำงานของไมโครคอนโทรลเลอร์ซึ่งมีรูปแบบการทำงานแบบเรียงลำดับของโค้ดกล่าวคือ ทำงานในช่องที่ 1 เสร็จแล้วไปทำต่อในช่องที่ 2 3 และ 4 เป็นทอดๆไปทีละช่องเมื่อครบทุกช่องแล้วก็วนกลับมาทำงานในช่องที่ 1 อีกครั้ง

การทำงานในลักษณะนี้เปรียบเสมือนว่าในหนึ่งรอบการทำงานของทุกๆช่องซึ่งได้ข้อมูลมาเพียงหนึ่งตัวอย่าง (Sample) แต่งานที่ทำทั้งหมดของไมโครคอนโทรลเลอร์จะเกิดขึ้นที่ช่องสัญญาณ 1-Wire เกือบทั้งหมด จึงทำให้เกิดการสูญเสียข้อมูลในช่องสัญญาณอื่นๆเป็นจำนวนมาก ดังนั้น จึงถือว่าการออกแบบระบบเช่นนี้ไม่เหมาะสม

เพื่อแก้ไขปัญหาข้างต้นในปัจจุบันได้มีวิธีการซึ่งเป็นการแก้ปัญหาทางซอฟท์แวร์ที่ใช้การแบ่งเวลาการทำงานให้กับแต่ละงานเป็นคาบ เพื่อให้ดูคล้ายว่าทุกๆงานได้ทำไปพร้อมกันซึ่งระบบนี้เรียกว่า RTOS (Real Time Operating System) ระบบดังกล่าวนี้ จะแบ่งเวลาให้กับแต่ละช่องสัญญาณเท่าๆกัน โดยเมื่อทำงานให้กับช่องสัญญาณใดจนครบคาบเวลาแล้วจะเก็บสถานะของช่องสัญญาณนั้นและย้ายการทำงานไปทำให้กับช่องสัญญาณต่อไป ตามลำดับจนครบทุกช่อง

แล้วจึงวนกลับมาทำงานในช่องสัญญาณแรกอีกครั้ง ซึ่งคาบเวลาในการทำงานแต่ละช่องนั้นมีความเร็วสูงในการทำงานระดับมิลลิวินาทีจึงทำให้ดูราวกับว่าทุกๆช่องสัญญาณทำงานไปพร้อมกันในเวลาเดียวกัน แต่อย่างไรก็ตามถึงแม้ว่าวิธีนี้จะให้ผลการทำงานที่น่าพอใจแต่เมื่อพิจารณาในเชิงลึกจะพบว่ายังมีโอกาสก่อให้เกิดการสูญเสียข้อมูลขึ้นได้เมื่อ Sample Time มีค่าต่ำกว่า Switch Time ของ RTOS หรือเมื่อเพิ่มช่องสัญญาณให้มากขึ้นเพราะยิ่งช่องสัญญาณมากขึ้นเท่าใดจะยิ่งทำให้คาบเวลาในการวนกลับมาเริ่มทำงานในช่องที่ 1 ใหม่ย่อมมากขึ้นเท่านั้น

เพื่อแก้ปัญหานี้จึงควรใช้การอินเตอร์รัพท์มาทำงานในลักษณะนี้แทน กล่าวคือ กำหนดให้ระดับความสำคัญการอินเตอร์รัพท์ (Interrupt Priority) ของช่องสัญญาณที่มีความเร็วมากให้มีระดับความสำคัญสูงสุดและกำหนดให้ช่องสัญญาณที่มีความเร็วต่ำกว่าให้มีระดับความสำคัญต่ำลงลดหลั่นกันลงไปตามความเร็วของช่องสัญญาณแต่ละช่องนั้น เมื่อมีสัญญาณเกิดขึ้นที่ช่องสัญญาณใดและไมโครคอนโทรลเลอร์พิจารณาแล้วเห็นว่ามีระดับความสำคัญการอินเตอร์รัพสูงกว่าช่องสัญญาณที่กำลังทำอยู่ในปัจจุบันก็จะไปทำงานตอบสนองต่อช่องสัญญาณนั้นในทันทีเมื่อเสร็จแล้วจึงไปทำงานตอบสนองต่องานที่ค้างอยู่นั้นต่อไป วิธีการนี้ถือว่าเป็นการจัดการที่ให้ผลดีมากไม่เกิดการสูญเสียข้อมูลเหมือนอย่างวิธีที่กล่าวมาข้างต้นแต่มีข้อจำกัดทางด้านจำนวนอินเตอร์รัพท์ของไมโครคอนโทรลเลอร์ที่มีจำกัดทำให้ไม่สามารถเพิ่มจำนวนช่องสัญญาณให้ระบบได้มากนัก

ดังนั้น การแก้ไขปัญหาโดยใช้ชิปเอฟพีจีเอมาทำการออกแบบหน่วยประมวลผลกลางของระบบคาต้าแอคควิซิชั่นในวิทยานิพนธ์นี้นับว่าเป็นการแก้ปัญหาได้อย่างมีประสิทธิภาพ เนื่องจาก ชิปประเภทนี้สามารถทำงานในลักษณะ *"แข่งขนาน"* (Concurrent) หรือทุกโมดูลภายในชิปสามารถทำงานไปพร้อมๆกันได้อย่างเป็นอิสระทำให้แต่ละช่องสัญญาณแยกกันทำงานไปโดยสิ้นเชิงผลที่ตามมาคือสามารถขยายระบบได้มากขึ้นตราบเท่าที่ทรัพยากรในชิปเอฟพีจีเอที่ใช้ยังเหลืออยู่

7.2 ข้อเสนอแนะและการพัฒนาระบบ

ในการขยายหรือพัฒนาระบบสามารถแยกออกได้เป็นหลายประการดังนี้

7.2.1 ความจุเกตของชิปเอฟพีจีเอ (FPGA Capacity)

ชิปที่นำมาใช้ในการออกแบบนี้คือ EP1K50TC144-3 ซึ่งเป็นชิปเอพพีจีเอที่มีความจุ 2,880 Logic Element หรือประมาณ 50,000 เกต ในการออกแบบนี้ได้ใช้งานไปแล้ว 838 Logic Element หรือประมาณ 14,548 เกต โดยมีรายละเอียดดังตารางที่ 5.1 ดังนั้นจึงเหลือจำนวนเกตที่สามารถพัฒนาระบบได้อีกประมาณ 2,042 Logic Element หรือประมาณ 35,452 เกต

7.2.2 จำนวนพอร์ตอินพุต-เอาท์พุต (FPGA Port)

ชิปเอฟพีจีเอ EP1K50TC144-3 มีจำนวนพอร์ต อินพุต-เอาท์พุตทั้งหมด 102 ขา จากข้อมูลในตารางที่ 5.1 จะเห็นว่าได้ใช้ไปแล้ว 40 ขา ดังนั้น จึงเหลือจำนวนพอร์ตที่สามารถพัฒนาระบบได้อีก 62 พอร์ต

7.2.3 แบนด์วิดของระบบการเชื่อมต่อ (PC Interface Bandwidth)

เนื่องจากไอซี FT2232 เมื่อทำงานในโหมดความเร็วเต็มขั้น (Full Speed) ซึ่งอัตราเร็วในการรับส่งข้อมูลในโหมดนี้ถูกจำกัดที่ 12 Mbps และจากผลการทดลองทำให้ทราบว่าอัตราเร็วในการรับ-ส่งข้อมูลทั้ง 4 ช่องสัญญาณรวมกับอัตราเร็วของข้อมูลส่วนหัว (Over Head) มีค่าประมาณ 5.5 Mbps ดังนั้น จึงเหลือแบนด์วิดที่สามารถพัฒนาระบบได้อีกประมาณ 6.5 Mbps

เมื่อพิจารณาข้อจำกัดในการขยายหรือพัฒนาระบบดังกล่าวข้างต้นแล้วจะเห็นว่ามีความเป็นไปได้ในการพัฒนาแยกออกเป็นสองส่วนคือการพัฒนาด้านชิปเอฟพีจีเอและการพัฒนาด้านการเชื่อมต่อกับคอมพิวเตอร์ (PC Interface)

การพัฒนาระบบด้านชิปเอฟพีจีเอนั้น เนื่องจากในปัจจุบันได้มีชิปเอฟพีจีเอผลิตออกมาสู่ท้องตลาดเป็นจำนวนมากซึ่งมีความจุเกตตั้งแต่ขนาดเล็กต่ำกว่า 1,000 เกตมีพอร์ตอินพุต-เอาท์พุตไม่ถึง 20 พอร์ต จนกระทั่งมีขนาดความจุมากกว่าหนึ่งล้านเกตและมีพอร์ตอินพุต-เอาท์พุตมากกว่า 200 พอร์ต ซึ่งจะมีราคาที่แตกต่างกันออกไป ดังนั้น ในการพัฒนาระบบจึงต้องใช้ดุลพินิจทั้งด้านความจุ จำนวนพอร์ตและราคาประกอบกันเพื่อให้ได้ความเหมาะสมกับระบบที่กำลังออกแบบอยู่นั้น

การพัฒนาระบบด้านการเชื่อมต่อกับคอมพิวเตอร์นั้น สามารถที่จะพัฒนาไปสู่ระบบยูเอสบีความเร็วสูงได้อีกซึ่งมีความเร็วในการส่งถ่ายข้อมูลสูงสุดถึง 480 Mbps และเนื่องจากชิป FT2232 สนับสนุนการทำงานในโหมดนี้อยู่แล้วจึงสามารถเปลี่ยนไปใช้โหมดความเร็วสูงได้โดยการแก้ไขฮาร์ดแวร์และซอฟท์แวร์บางส่วนเท่านั้น นอกจากการเชื่อมต่อทางพอร์ตยูเอสบีแล้วหากระบบต้องการพัฒนาระบบให้ส่งถ่ายข้อมูลความเร็วสูงในระดับหลายกิกะบิตต่อวินาทีก็ควรพัฒนาให้ระบบเชื่อมต่อกับคอมพิวเตอร์ผ่านทางการ์ด PCI

7.2.4 การพัฒนาโมดูลแต่ละช่องสัญญาณเป็น IP Core

IP Core (intellectual property core) เป็น Library โมดูลซึ่งสามารถที่จะนำไปใช้ในโปรเจคอื่นๆได้ทำให้โปรเจคนั้นไม่จำเป็นต้องเขียนโมดูลประเภทเดียวกันนี้ซ้ำอีกช่วยให้การพัฒนาระบบใหม่ๆเป็นไปได้ด้วยความรวดเร็วยิ่งขึ้น สำหรับในการออกแบบนี้สามารถนำเอาโมดูลที่ทำหน้าที่อ่านข้อมูลจาก ADC ในแต่ละช่องสัญญาณนั้นไปใช้ได้แต่จะต้องเป็นไอซีที่มีเบอร์ตรงกันกับในการออกแบบนี้เท่านั้น

อย่างไรก็ตามการพัฒนาให้โมดูลมีความยืดหยุ่นสามารถนำไปใช้กับ ADC เบอร์อื่นๆที่มีโปรโตคอลการสื่อสารชนิดเดียวกันได้นั้นเป็นเรื่องที่ควรนำไปพัฒนาระบบให้ดีขึ้นโดยมี

องค์ประกอบที่จำเป็นต้องใช้พิจารณาในการพัฒนาโมดูลอยู่ 3 ประการคือจำนวนบิตข้อมูล, แอดเดรสของ ADC และรหัสคำสั่ง

เนื่องจากจำนวนบิตข้อมูลที่ใช้ใน ADC แต่ละตัวนั้นมีความแตกต่างกันไม่ว่าจะเป็น 8 บิต , 10 บิต, 12 บิต ฯลฯ ซึ่งการออกแบบเพื่อให้โมดูลสามารถตอบสนองต่อ ADC ที่มีความแตกต่างกันด้านบิตข้อมูลเหล่านี้จะต้องเพิ่มช่องทางให้โมดูลนั้นสามารถเลือกหรือกำหนดพารามิเตอร์ได้ว่าต้องการทำงานกับ ADC ที่มีจำนวนบิตเท่าใด

ในแง่ขององค์ประกอบทางด้านการระบุแอดเดรสนั้นจะเห็นได้จากการทำงานกับโปรโตคอลแบบ I^2C และ 1-Wire นั้นจำเป็นต้องระบุแอดเดรสของอุปกรณ์เสมอเพื่อให้โมดูลสามารถแยกแยะได้ว่าการอ่านหรือเขียนข้อมูลแต่ครั้งนั้นทำกับอุปกรณ์สเลฟตัวใด ดังนั้น การพัฒนาโมดูลให้มีความสามารถทางด้านนี้ก็คือการเพิ่มช่องทางให้โมดูลสามารถเลือกหรือกำหนดแอดเดรสได้นั่นเอง

ประการสุดท้ายคือการพัฒนาเพื่อให้โมดูลสามารถตอบสนองต่อคำสั่งได้มากยิ่งขึ้น คำสั่งในที่นี้คือคำสั่งที่ระบุว่าต้องการให้โมดูลทำงานใดหรือทำอะไรกับ ADC เช่น อ่านข้อมูล, เขียนข้อมูล, สั่งให้แปลงข้อมูล, กำหนดโหมดการทำงาน เป็นต้น ซึ่งคำสั่งเหล่านี้เป็นคำสั่งที่ถูกกำหนดขึ้นจากบริษัทผู้ผลิต ADC ซึ่งแต่ละรายก็จะมีคำสั่งที่แตกต่างกันออกไป ดังนั้น การพัฒนาโมดูลเพื่อให้สามารถทำงานได้กับชิปทุกผู้ผลิตจึงต้องกำหนดช่องทางให้โมดูลสามารถระบุคำสั่งเหล่านี้ลงไปได้ นอกจากนั้น ยังต้องจัดลำดับการทำงานของคำสั่งเหล่านี้ได้อีกด้วย

บรรณานุกรม

[1] Ziad Salem, Ismail Al Kamal, Alaa Al Bashar "**A Novel Design of an Industrial Data Acquisition System**", Proceeding of International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA 2006. pp. 2589-2594.

[2] Jorge Yáñez, David Quintana, Camilo Quintáns, José Fariña, Juan J. Rodríguez-Andina," **FPGA-based system for the education in data acquisition and signal generation**", Proceeding of Industrial Electronics Society Conference, IECON 2005. pp. 2168-2173.

[3] A. Sagahyroon ,T. Al-khudairi, "**FPGA Based Acquisition of Sensor Data**" Proceeding of International Conference on Industrial Technology, ICIT 2004. pp. 1398 – 1401.

[4] M. Popa, M. Marcu, A. S. Popa, "**A Microcontroller Based Data Acquisition System with USB Interface**", Proceeding of International Conference on Electrical, Electronic and Computer Engineering, ICEEC 2004 pp. 206-209

[5] FTDI Corporation., **www.ftdichip.com**, Glasgow City, Scotland,

[6] Hamblen J., Furman M., "**Rapid Prototyping of Digital Systems**", Quartus II Edition, Springer Science+Business Media Inc., New York, USA, 2006

[7] Dluglas L. Perry "**VHDL Programming By Example**", Forth Edition, McGraw-Hill Company, USA, 2002.

[8] Pong P. Chu "**RTL Hardware Design Using VHDL**", First Edition, John Wiley & Sons, Inc., Hoboken, Canada, 2006.

[9] George Shepherd, David Kruglinski "**Programming with Microsoft Visual C++ .NET**", Sixth Edition, Microsoft Press, Washington, USA, 2003.

[10] Axelson, J. , "**USB Complete: Everything You Need to Develop Custom USB Peripherals**", Lakeview Research, 2001, ISBN 09655081958

[11] Altera Corporation. **www.altera.com**

[12] Michael Barr, "**Programming Embedded System in C and C++**"

[13] Craig Peacock, "**USB in a nutshell**" www.beyondlogic.com

ภาคผนวก

ภาคผนวก ก.

โปรแกรมภาษาวีเอชดีแอล

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY CIRCUIT_MAIN_FIFO IS
        PORT
        (
                TXE :  IN  STD_LOGIC;
                RXF :  IN  STD_LOGIC;
                CLK :  IN  STD_LOGIC;
                DOUT :  IN  STD_LOGIC;
                INT :  IN  STD_LOGIC;
                RESET :  IN  STD_LOGIC;
                SDA :  INOUT  STD_LOGIC;
                DQ :  INOUT  STD_LOGIC;
                DATA_IN :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
                USB_DATA :  INOUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
                WR :  OUT  STD_LOGIC;
                RD :  OUT  STD_LOGIC;
                OE :  OUT  STD_LOGIC;
                SCL :  OUT  STD_LOGIC;
                DCLK :  OUT  STD_LOGIC;
                CS_0820 : OUT  STD_LOGIC;
                RD_0820 : OUT  STD_LOGIC;
                WR_CTRL :  OUT  STD_LOGIC;
                READ_CTRL :  OUT  STD_LOGIC;
                TEST2 : OUT  STD_LOGIC;
                TEST1 : OUT  STD_LOGIC;
                TEST3 : OUT  STD_LOGIC;
                TEST5 : OUT  STD_LOGIC;
                CS_SPI :  OUT  STD_LOGIC;
                TEST4 : OUT  STD_LOGIC;
                TEST_TICK :  OUT  STD_LOGIC;
                SIGNAL_PARA :  OUT  STD_LOGIC;
                SIGNAL_SPI :  OUT  STD_LOGIC;
                SIGNAL_I2C :  OUT  STD_LOGIC;
                DATA_RD :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
                STATE :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0);
                TEST_OUT :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END CIRCUIT_MAIN_FIFO;

ARCHITECTURE bdf_type OF CIRCUIT_MAIN_FIFO IS

COMPONENT data_usb_controller
        PORT(CLK : IN STD_LOGIC;
                TXE : IN STD_LOGIC;
                ENA_WR : IN STD_LOGIC;
                RXF : IN STD_LOGIC;
                RESET : IN STD_LOGIC;
                CLK_RD : IN STD_LOGIC;
                FPGA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                USB_DATA : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                WR_CTRL : OUT STD_LOGIC;
                READ_CTRL : OUT STD_LOGIC;
                OE : OUT STD_LOGIC;
                NEXT_TX : OUT STD_LOGIC;
                NEXT_RX : OUT STD_LOGIC;
                RD : OUT STD_LOGIC;
                WR : OUT STD_LOGIC;
                FPGA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                STATE : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;

COMPONENT one_clk
        PORT(CLK_IN : IN STD_LOGIC;
                ENA0 : IN STD_LOGIC;
                CLK_OUT : OUT STD_LOGIC
        );
END COMPONENT;

COMPONENT circuit_pcf8591
        PORT(CLK : IN STD_LOGIC;
```

```
                RST_N : IN STD_LOGIC;
                ENA : IN STD_LOGIC;
                SDA : INOUT STD_LOGIC;
                SCL : OUT STD_LOGIC;
                RDY : OUT STD_LOGIC;
                DATA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;

COMPONENT adc0820
        PORT(RESET : IN STD_LOGIC;
                CLK : IN STD_LOGIC;
                INT : IN STD_LOGIC;
                DATA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                CS : OUT STD_LOGIC;
                RD : OUT STD_LOGIC;
                RDY : OUT STD_LOGIC;
                DATA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                STATE_OUT : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;

COMPONENT lpm_fifo0
        PORT(wrreq : IN STD_LOGIC;
                rdreq : IN STD_LOGIC;
                clock : IN STD_LOGIC;
                aclr : IN STD_LOGIC;
                data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                full : OUT STD_LOGIC;
                empty : OUT STD_LOGIC;
                q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                usedw : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;

COMPONENT control_unit
GENERIC (IDLE : STD_LOGIC_VECTOR(3 DOWNTO 0);
                READ_DATA : STD_LOGIC_VECTOR(3 DOWNTO 0);
                READ_I2C : STD_LOGIC_VECTOR(3 DOWNTO 0);
                READ_ONEWIRE : STD_LOGIC_VECTOR(3 DOWNTO 0);
                READ_PARA : STD_LOGIC_VECTOR(3 DOWNTO 0);
                READ_SPI : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_I2C : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_I2C_START : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_ONEWIRE : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_ONEWIRE_START : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_PARA : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_PARALLEL_START : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_SPI : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_SPI_START : STD_LOGIC_VECTOR(3 DOWNTO 0);
                WRITE_STATE : STD_LOGIC_VECTOR(3 DOWNTO 0)
                );
        PORT(CLK : IN STD_LOGIC;
                RESET : IN STD_LOGIC;
                EMP_PARA : IN STD_LOGIC;
                EMP_I2C : IN STD_LOGIC;
                EMP_SPI : IN STD_LOGIC;
                EMP_ONEWIRE : IN STD_LOGIC;
                TX_FINISH : IN STD_LOGIC;
                RXF_IN : IN STD_LOGIC;
                SYN_100ms : IN STD_LOGIC;
                RESET_OUT : OUT STD_LOGIC;
                WR_ENA_OUT : OUT STD_LOGIC;
                RD_PARA : OUT STD_LOGIC;
                RD_I2C : OUT STD_LOGIC;
                RD_SPI : OUT STD_LOGIC;
                RD_ONEWIRE : OUT STD_LOGIC;
                ADDS_OUT : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                ENCODE_DATA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                STATE_OUT : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;
```

```
COMPONENT mux_5input
        PORT(data0x : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                data1x : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                data2x : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                data3x : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                data4x : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                sel : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                result : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;

COMPONENT div10
        PORT(clk_in : IN STD_LOGIC;
                clk_out : OUT STD_LOGIC
        );
END COMPONENT;

COMPONENT div20
        PORT(clk_in : IN STD_LOGIC;
                clk_out : OUT STD_LOGIC
        );
END COMPONENT;

COMPONENT mcp3201
GENERIC (IDLE : INTEGER;
                        NEXT_LOOP : INTEGER;
                        SEND_DATA : INTEGER;
                        START : INTEGER;
                        TRANSFER : INTEGER;
                        WAIT_DOUT : INTEGER
                        );
        PORT(CLK_IN : IN STD_LOGIC;
                DOUT : IN STD_LOGIC;
                DCLK : OUT STD_LOGIC;
                CS : OUT STD_LOGIC;
                RDY : OUT STD_LOGIC;
                DATA_8BIT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                STATE_OUT : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;

COMPONENT div30000
        PORT(clk_in : IN STD_LOGIC;
                clk_out : OUT STD_LOGIC
        );
END COMPONENT;

COMPONENT div36
        PORT(clk_in : IN STD_LOGIC;
                clk_out : OUT STD_LOGIC
        );
END COMPONENT;

COMPONENT ds1820
        PORT(CLK : IN STD_LOGIC;
                DQ : INOUT STD_LOGIC;
                READY : OUT STD_LOGIC;
                SIGN : OUT STD_LOGIC;
                DATA_OUTPUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;

SIGNAL SYNTHESIZED_WIRE_0: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_1: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_2: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_3: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_4: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_5: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_6: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_7: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_8: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_9: STD_LOGIC;
```

```
SIGNAL SYNTHESIZED_WIRE_10: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_11: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_12: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_13: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_14: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_15: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_16: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_17: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_18: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_19: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_20: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_21: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_22: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_23: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_24: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_25: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_26: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_27: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_28: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_29: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_30: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_31: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_32: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_33: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_34: STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL SYNTHESIZED_WIRE_35: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_36: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_37: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_38: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_39: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_40: STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_41: STD_LOGIC;


BEGIN
TEST1 <= CLK;
TEST_TICK <= CLK;
TEST_OUT <= USB_DATA;
WR <= SYNTHESIZED_WIRE_41;
TEST5 <= SYNTHESIZED_WIRE_41;
SIGNAL_PARA <= SYNTHESIZED_WIRE_43;
SIGNAL_SPI <= SYNTHESIZED_WIRE_43;
SYNTHESIZED_WIRE_7 <= '1';
SYNTHESIZED_WIRE_16 <= '1';


b2v_inst : data_usb_controller
PORT MAP(CLK => CLK,
         TXE => TXE,
         ENA_WR => SYNTHESIZED_WIRE_0,
         RXF => RXF,
         RESET => SYNTHESIZED_WIRE_1,
         CLK_RD => CLK,
         FPGA_IN => SYNTHESIZED_WIRE_2,
         USB_DATA => USB_DATA,
         WR_CTRL => WR_CTRL,
         READ_CTRL => READ_CTRL,
         OE => OE,
         NEXT_TX => SYNTHESIZED_WIRE_21,
         RD => RD,
         WR => SYNTHESIZED_WIRE_41,
         FPGA_OUT => DATA_RD);


b2v_inst1:one_clk
PORT MAP(CLK_IN => CLK,
         ENA0=> SYNTHESIZED_WIRE_3,
         CLK_OUT => SYNTHESIZED_WIRE_26);
```

```
b2v_inst10:one_clk
PORT MAP(CLK_IN => CLK,
         ENA0=> SYNTHESIZED_WIRE_4,
         CLK_OUT => SYNTHESIZED_WIRE_23);


b2v_inst14:one_clk
PORT MAP(CLK_IN => CLK,
         ENA0=> SYNTHESIZED_WIRE_5,
         CLK_OUT => SYNTHESIZED_WIRE_13);


b2v_inst15:circuit_pcf8591
PORT MAP(CLK => SYNTHESIZED_WIRE_6,
         ENA => SYNTHESIZED_WIRE_7,
         SDA => SDA,
         SCL => SCL,
         RDY => SYNTHESIZED_WIRE_5,
         DATA_OUT => SYNTHESIZED_WIRE_15);


b2v_inst16:adc0820
PORT MAP(CLK => SYNTHESIZED_WIRE_8,
         INT => INT,
         DATA_IN => DATA_IN,
         CS => CS_0820,
         RD => RD_0820,
         RDY => SYNTHESIZED_WIRE_38,
         DATA_OUT => SYNTHESIZED_WIRE_11);


b2v_inst17:lpm_fifo0
PORT MAP(wrreq => SYNTHESIZED_WIRE_9,
         rdreq => SYNTHESIZED_WIRE_10,
         clock => CLK,
         data => SYNTHESIZED_WIRE_11,
         empty => SYNTHESIZED_WIRE_17,
         q => SYNTHESIZED_WIRE_30);


b2v_inst18:one_clk
PORT MAP(CLK_IN => CLK,
         ENA0=> SYNTHESIZED_WIRE_12,
         CLK_OUT => SYNTHESIZED_WIRE_10);


b2v_inst19:lpm_fifo0
PORT MAP(wrreq => SYNTHESIZED_WIRE_13,
         rdreq => SYNTHESIZED_WIRE_14,
         clock => CLK,
         data => SYNTHESIZED_WIRE_15,
         empty => SYNTHESIZED_WIRE_18,
         q => SYNTHESIZED_WIRE_31);


b2v_inst2:control_unit
GENERIC MAP(IDLE => "0000",
            READ_DATA => "1110",
            READ_I2C => "0010",
            READ_ONEWIRE => "0100",
            READ_PARA => "0001",
            READ_SPI => "0011",
            WRITE_I2C => "1011",
            WRITE_I2C_START => "0110",
            WRITE_ONEWIRE => "1101",
            WRITE_ONEWIRE_START => "1000",
            WRITE_PARA => "1010",
            WRITE_PARALLEL_START => "0101",
```

```
                        WRITE_SPI => "1100",
                        WRITE_SPI_START => "0111",
                        WRITE_STATE => "1001"
                        )
PORT MAP(CLK => CLK,
                 RESET => SYNTHESIZED_WIRE_16,
                 EMP_PARA => SYNTHESIZED_WIRE_17,
                 EMP_I2C => SYNTHESIZED_WIRE_18,
                 EMP_SPI => SYNTHESIZED_WIRE_19,
                 EMP_ONEWIRE => SYNTHESIZED_WIRE_20,
                 TX_FINISH => SYNTHESIZED_WIRE_21,
                 RXF_IN => RXF,
                 RESET_OUT => SYNTHESIZED_WIRE_1,
                 WR_ENA_OUT => SYNTHESIZED_WIRE_0,
                 RD_PARA => SYNTHESIZED_WIRE_12,
                 RD_I2C => SYNTHESIZED_WIRE_22,
                 RD_SPI => SYNTHESIZED_WIRE_35,
                 RD_ONEWIRE => SYNTHESIZED_WIRE_36,
                 ADDS_OUT => SYNTHESIZED_WIRE_34,
                 ENCODE_DATA_OUT => SYNTHESIZED_WIRE_29,
                 STATE_OUT => STATE);


b2v_inst20:one_clk
PORT MAP(CLK_IN => CLK,
                 ENA0=> SYNTHESIZED_WIRE_22,
                 CLK_OUT => SYNTHESIZED_WIRE_14);


b2v_inst21:lpm_fifo0
PORT MAP(wrreq => SYNTHESIZED_WIRE_23,
                 rdreq => SYNTHESIZED_WIRE_24,
                 clock => CLK,
                 data => SYNTHESIZED_WIRE_25,
                 empty => SYNTHESIZED_WIRE_19,
                 q => SYNTHESIZED_WIRE_32);


b2v_inst22:lpm_fifo0
PORT MAP(wrreq => SYNTHESIZED_WIRE_26,
                 rdreq => SYNTHESIZED_WIRE_27,
                 clock => CLK,
                 data => SYNTHESIZED_WIRE_28,
                 empty => SYNTHESIZED_WIRE_20,
                 q => SYNTHESIZED_WIRE_33);


b2v_inst23:mux_5input
PORT MAP(data0x => SYNTHESIZED_WIRE_29,
                 data1x => SYNTHESIZED_WIRE_30,
                 data2x => SYNTHESIZED_WIRE_31,
                 data3x => SYNTHESIZED_WIRE_32,
                 data4x => SYNTHESIZED_WIRE_33,
                 sel => SYNTHESIZED_WIRE_34,
                 result => SYNTHESIZED_WIRE_2);


b2v_inst24:one_clk
PORT MAP(CLK_IN => CLK,
                 ENA0=> SYNTHESIZED_WIRE_35,
                 CLK_OUT => SYNTHESIZED_WIRE_24);


b2v_inst25:one_clk
PORT MAP(CLK_IN => CLK,
                 ENA0=> SYNTHESIZED_WIRE_36,
                 CLK_OUT => SYNTHESIZED_WIRE_27);


b2v_inst27:div10
```

```
PORT MAP(clk_in => SYNTHESIZED_WIRE_43,
                clk_out => SIGNAL_I2C);


b2v_inst3:one_clk
PORT MAP(CLK_IN => CLK,
                ENA0=> SYNTHESIZED_WIRE_38,
                CLK_OUT => SYNTHESIZED_WIRE_9);


b2v_inst34:div20
PORT MAP(clk_in => CLK,
                clk_out => SYNTHESIZED_WIRE_39);


b2v_inst4:mcp3201
GENERIC MAP(IDLE => 0,
                    NEXT_LOOP => 5,
                    SEND_DATA => 4,
                    START => 1,
                    TRANSFER => 3,
                    WAIT_DOUT => 2
                    )
PORT MAP(CLK_IN => SYNTHESIZED_WIRE_39,
                DOUT => DOUT,
                DCLK => DCLK,
                CS => CS_SPI,
                RDY => SYNTHESIZED_WIRE_4,
                DATA_8BIT => SYNTHESIZED_WIRE_25);


b2v_inst57:div20
PORT MAP(clk_in => CLK,
                clk_out => SYNTHESIZED_WIRE_40);


b2v_inst59:div30000
PORT MAP(clk_in => CLK,
                clk_out => SYNTHESIZED_WIRE_43);


b2v_inst6:div20
PORT MAP(clk_in => CLK,
                clk_out => SYNTHESIZED_WIRE_8);


b2v_inst8:div36
PORT MAP(clk_in => CLK,
                clk_out => SYNTHESIZED_WIRE_6);


b2v_inst9:ds1820
PORT MAP(CLK => SYNTHESIZED_WIRE_40,
                DQ => DQ,
                READY => SYNTHESIZED_WIRE_3,
                DATA_OUTPUT => SYNTHESIZED_WIRE_28);


END bdf_type;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;


ENTITY DS1820 IS
 PORT
 (   CLK:IN STD_LOGIC;

  DQ:INOUT std_logic;
  NUM_SIGNAL:OUT std_logic_vector(13 downto 0);
  DATA_OUT :OUT std_logic_vector(12 downto 0);
  DATA_OUTPUT:OUT std_logic_vector(7 downto 0);
  READY :OUT std_logic:='0';
  SIGN :OUT std_logic
 );
END DS1820;


ARCHITECTURE block_name_architecture OF DS1820 IS
SIGNAL count_int : std_logic_vector(6 downto 0);
SIGNAL num       : std_logic_vector(13 downto 0):="00000000000000";
SIGNAL SIGN0      :   std_logic;
SIGNAL SIG_READY :std_logic:='0';
BEGIN
PROCESS(CLK)
variable DATA : std_logic_vector(12 downto 0):="0000000000000";
BEGIN
IF rising_edge(CLK) THEN
       NUM_SIGNAL <= num;
       READY<= SIG_READY;
  count_int <= count_int+1;
   IF (num>=11200 OR num<=0) THEN
    num<="00000000000000";
   END IF;
   IF (count_int>=70 OR count_int<0) THEN
   count_int<="0000000";
   num<=num+1;
   END IF;
       IF (num>=0 AND num<=6) THEN
       DQ<='0';
       SIG_READY <='0';
       END IF;
   IF (num>=7 AND num<=13) THEN
       DQ<='Z';
       END IF;
   IF (num=16 OR num=17 OR num=20 OR num=21) THEN
              IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=7 AND count_int<=70) THEN
                   DQ<='Z';
              END IF;
   END IF;
   IF (num=14 OR num=15 OR num=18 OR num=19) THEN
            IF (count_int>=0 AND count_int<=60) THEN
                 DQ<='0';
            END IF;
                      IF (count_int>=61 AND count_int<=70) THEN
                 DQ<='Z';
            END IF;
   END IF;
   IF (num=22 OR num=23 OR num=25 OR num=26 OR num=27 OR num=29) THEN
                      IF (count_int>=0 AND count_int<=60) THEN
                              DQ<='0';
                      END IF;
                      IF (count_int>=61 AND count_int<=70) THEN
                              DQ<='Z';
                      END IF;
       END IF;
   IF (num=24 OR num=28) THEN
                      IF (count_int>=0 AND count_int<=6) THEN
                              DQ<='0';
                      END IF;
                      IF (count_int>=7 AND count_int<=70) THEN
                                                      DQ<='Z';
                      END IF;
       END IF;
```

```
    IF (num>=11100 AND num<=11106) THEN
          DQ<='0';
     END IF;
    IF (num>=11108 AND num<=11113) THEN
          DQ<='Z';
     END IF;
    IF (num=11116 OR num=11117 OR num=11120 OR num=11121) THEN
              IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
              END IF;
     END IF;
    IF (num=11114 OR num=11115 OR num=11118 OR num=11119) THEN
              IF (count_int>=0 AND count_int<=60) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=61 AND count_int<=70) THEN
                    DQ<='Z';
              END IF;
     END IF;
    IF (num=11122 OR num=11128) THEN
              IF (count_int>=0 AND count_int<=60) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=61 AND count_int<=70) THEN
                    DQ<='Z';
              END IF;
     END IF;
   IF (num=11123 OR num=11124 OR num=11125 OR num=11126 OR num=11127 OR num=11129)
THEN
              IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
              END IF;
     END IF;
    IF (num=11131) THEN
      IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
              END IF;
              IF (count_int=12) THEN
                    IF (DQ='0') THEN
                          DATA(0):='0';
                    END IF;
                    IF (DQ='1') THEN
                          DATA(0):='1';
                    END IF;
              END IF;
    END IF;
    IF (num=11132) THEN
        IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=7 AND count_int<=70) THEN
                          DQ<='Z';
                    END IF;
              IF (count_int=12) THEN
                    IF (DQ='0') THEN
                     DATA(1):='0';
                     END IF;
                     IF (DQ='1') THEN
                     DATA(1):='1';
                     END IF;
              END IF;
    END IF;
    IF (num=11133) THEN
       IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
              END IF;
              IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
              END IF;
              IF (count_int=12) THEN
```

```
                        IF (DQ='0') THEN
                                DATA(2):='0';
                         END IF;
                         IF (DQ='1') THEN
                                DATA(2):='1';
                         END IF;
                END IF;
END IF;
IF (num=11134) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                        DQ<='0';
                END IF;
                IF (count_int>=7 AND count_int<=70) THEN
                        DQ<='Z';
                END IF;
                IF (count_int=12) THEN
                        IF (DQ='0') THEN
                                DATA(3):='0';
                         END IF;
                         IF (DQ='1') THEN
                                DATA(3):='1';
                         END IF;
                END IF;
END IF;
IF (num=11135) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                        DQ<='0';
                END IF;
                IF (count_int>=7 AND count_int<=70) THEN
                        DQ<='Z';
                END IF;
                IF (count_int=12) THEN
                        IF (DQ='0') THEN
                                DATA(4):='0';
                         END IF;
                         IF (DQ='1') THEN
                                DATA(4):='1';
                         END IF;
                END IF;
END IF;
IF (num=11136) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                        DQ<='0';
                END IF;
                IF (count_int>=7 AND count_int<=70) THEN
                        DQ<='Z';
                END IF;
                IF (count_int=12) THEN
                        IF (DQ='0') THEN
                                DATA(5):='0';
                         END IF;
                         IF (DQ='1') THEN
                                DATA(5):='1';
                         END IF;
                END IF;
END IF;
IF (num=11137) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                        DQ<='0';
                END IF;
                IF (count_int>=7 AND count_int<=70) THEN
                        DQ<='Z';
                END IF;
                IF (count_int=12) THEN
                        IF (DQ='0') THEN
                                DATA(6):='0';
                         END IF;
                         IF (DQ='1') THEN
                                DATA(6):='1';
                         END IF;
 END IF;
END IF;
IF (num=11138) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                        DQ<='0';
                END IF;
                IF (count_int>=7 AND count_int<=70) THEN
                        DQ<='Z';
                END IF;
```

```
            IF (count_int=12) THEN
                    IF (DQ='0') THEN
                            DATA(7):='0';
                     END IF;
                     IF (DQ='1') THEN
                            DATA(7):='1';
                     END IF;
            END IF;
END IF;
IF (num=11139) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
            END IF;
            IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
            END IF;
            IF (count_int=12) THEN
                    IF (DQ='0') THEN
                            DATA(8):='0';
                     END IF;
                     IF (DQ='1') THEN
                            DATA(8):='1';
                     END IF;
            END IF;
END IF;
IF (num=11140) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
            END IF;
            IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
            END IF;
            IF (count_int=12) THEN
                    IF (DQ='0') THEN
                            DATA(9):='0';
                     END IF;
                     IF (DQ='1') THEN
                            DATA(9):='1';
                     END IF;
            END IF;
END IF;
IF (num=11141) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
            END IF;
            IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
            END IF;
            IF (count_int=12) THEN
                    IF (DQ='0') THEN
                            DATA(10):='0';
                     END IF;
                     IF (DQ='1') THEN
                            DATA(10):='1';
                     END IF;
            END IF;
END IF;
IF (num=11142) THEN
     IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
            END IF;
            IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
            END IF;
            IF (count_int=12) THEN
                    IF (DQ='0') THEN
                            DATA(11):='0';
                     END IF;
                     IF (DQ='1') THEN
                            DATA(11):='1';
                     END IF;
            END IF;
END IF;
IF (num=11143) THEN
    IF (count_int>=0 AND count_int<=6) THEN
                    DQ<='0';
            END IF;
            IF (count_int>=7 AND count_int<=70) THEN
                    DQ<='Z';
```

```
                END IF;
                IF (count_int=12) THEN
                        IF (DQ='0') THEN
                                SIGN0<='0';
                                DATA(12):='0';
                         END IF;
                         IF (DQ='1') THEN
                                SIGN0<='1';
                                DATA(12):='1';
                         END IF;
                END IF;
    END IF;
            IF (num=11145 ) THEN
                   SIGN<=SIGN0;
                   DATA_OUT <= DATA;
                   DATA_OUTPUT<=DATA(7 downto 0);
                   SIG_READY <='1';
            END IF;
            IF (num=11148 ) THEN
                  SIG_READY<='0';
            END IF;
END IF;
END PROCESS;


END block_name_architecture;
```

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

Entity MCP3201 is
generic (
                        constant IDLE : integer range 0 to 15:=0;
                        constant START: integer range 0 to 15:=1;
                        constant WAIT_DOUT: integer range 0 to 15:=2;
                        constant TRANSFER: integer range 0 to 15:=4;
                        constant SEND_DATA: integer range 0 to 15:=8
                );
port(
        CLK_IN : IN std_logic;
        DOUT : IN std_logic;
        DCLK : OUT std_logic;
        CS : OUT std_logic:='1';
        RDY : OUT std_logic:='0';
        DATA_8BIT : OUT std_logic_vector(7 downto 0):="00000000";
        STATE_OUT : OUT std_logic_vector(3 downto 0)
);
end MCP3201;
Architecture rtl of MCP3201 is
begin
        DCLK <= CLK_IN;
        process(CLK_IN)
        variable mode : integer range 0 To 7:=IDLE;
        variable buffer_data : std_logic_vector(11 downto 0);
        variable buffer_8bit : std_logic_vector(7 downto 0);
        variable buffer_4bit : std_logic_vector(3 downto 0);
        variable count : integer range 0 To 15:=11;
        variable delay : integer range 0 To 31:=0;
        Begin
        STATE_OUT <= CONV_STD_LOGIC_VECTOR(mode,4);
                if CLK_IN'event and CLK_IN = '1' then
                        case mode is
                        when IDLE =>
                                CS <= '1';
                                mode := START;
                                count := 11;
                                RDY <= '0';
                        when START =>
                                CS <= '0';
                                mode := WAIT_DOUT;
                                RDY <= '0';
                        when WAIT_DOUT =>
                                if DOUT = '0' then
                                        mode := TRANSFER;
                                elsif DOUT = 'Z' then
                                        mode := WAIT_DOUT;
                                else
                                        mode := TRANSFER;
                                end if;
                        when TRANSFER =>
                                buffer_data(count) := Dout;
                                if (count>=4) then
                                        buffer_8bit(count-4) := Dout;
                                end if;
                                if count = 0 then
                                        count := 11;
                                        mode := SEND_DATA;
                                        CS <= '1';
                                else
                                        count := count-1;
                                end if;
                        when SEND_DATA =>
                                DATA_8BIT <= buffer_8bit;
                                mode := IDLE;
                                RDY <= '1';
                        when others =>
                                mode :=IDLE;
                        end case;
                end if;
        End Process;
end rtl;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PCF8591_Interface is

        generic (
        constant IDEL : std_logic_vector(5 downto 0):= "000000";
        constant START : std_logic_vector(5 downto 0):= "000001";
        constant START_NEXT : std_logic_vector(5 downto 0):= "000010";
        constant ADDR_WRITE : std_logic_vector(5 downto 0):= "000011";
        constant SEND_WRITE: std_logic_vector(5 downto 0):= "000100";
        constant ADDR_WRITE_NEXT: std_logic_vector(5 downto 0):= "000101";
        constant ADDR_WRITE_CLR : std_logic_vector(5 downto 0):= "000110";

        constant ACK1 : std_logic_vector(5 downto 0):= "000111";
        constant WAIT_ACK1 : std_logic_vector(5 downto 0):= "001000";
        constant READ_ACK1 : std_logic_vector (5 downto 0):="001001";

        constant COMD_DEV : std_logic_vector(5 downto 0):= "001010";
        constant SEND_COMD: std_logic_vector(5 downto 0):= "001011";
        constant SEND_COMD_NEXT: std_logic_vector(5 downto 0):= "001100";
        constant COMD_DEV_CLR: std_logic_vector(5 downto 0):= "001101";
        constant ACK2 : std_logic_vector(5 downto 0):= "001110";
        constant WAIT_ACK2 : std_logic_vector(5 downto 0):= "001111";
        constant READ_ACK2 : std_logic_vector (5 downto 0):="010000";
        constant STOP1 : std_logic_vector(5 downto 0):= "010001";
        constant SEND_STOP1 : std_logic_vector(5 downto 0):= "010010";
        constant START1 : std_logic_vector(5 downto 0):= "010011";
        constant START_NEXT1 : std_logic_vector(5 downto 0):= "010100";
        constant ADDR_READ : std_logic_vector(5 downto 0):= "010101";
        constant SEND_READ: std_logic_vector(5 downto 0):= "010110";
        constant ADDR_READ_NEXT: std_logic_vector(5 downto 0):= "010111";
        constant ADDR_READ_CLR: std_logic_vector(5 downto 0):= "011000";
        constant ACK3 : std_logic_vector(5 downto 0):= "011001";
        constant WAIT_ACK3 : std_logic_vector(5 downto 0):= "011010";
        constant READ_ACK3 : std_logic_vector (5 downto 0):="011011";


        constant READ_DATA : std_logic_vector(5 downto 0):= "011100";
        constant RECEIVE_DATA: std_logic_vector(5 downto 0):= "011101";
        constant READ_NEXT: std_logic_vector(5 downto 0):= "011110";
        constant READ_CLR: std_logic_vector(5 downto 0):= "011111";
        constant NACK_ST : std_logic_vector(5 downto 0):= "100000";
        constant WAIT_NACK : std_logic_vector(5 downto 0) := "100001";
        constant READ_NACK : std_logic_vector(5 downto 0):= "100010";
        constant STOP2 : std_logic_vector(5 downto 0):= "100011";
        constant SEND_STOP2 : std_logic_vector(5 downto 0):= "100100";
        constant COMD_DEV1 : std_logic_vector(5 downto 0):= "100101";
        constant SEND_COMD1: std_logic_vector(5 downto 0):= "100110";
        constant SEND_COMD_NEXT1: std_logic_vector(5 downto 0):= "100111";
        constant COMD_DEV_CLR1: std_logic_vector(5 downto 0):= "101000";
        constant ACK4 : std_logic_vector(5 downto 0):= "101001";
        constant WAIT_ACK4 : std_logic_vector(5 downto 0):= "101010";
        constant READ_ACK4 : std_logic_vector (5 downto 0):="101011"

                        );

                port    (
                        CLK_I2C : in std_logic;
                        RESET_N : in std_logic;
                        SDA : inout std_logic;
                        SCL : out std_logic;
                        EN : in std_logic;
                        ACK : out std_logic;
                        ADC_DATA_OUT : out std_logic_vector(7 downto 0);
                        ADC_READY:     out std_logic;
                        STATE_STATUS : out std_logic_vector(5 downto 0);
                        DATA_OUT : out std_logic_vector(7 downto 0)
                        );

end PCF8591_Interface;

architecture Behavioral of PCF8591_Interface is
        signal COUNT : std_logic_vector(2 downto 0);
```

```
        signal STATE : std_logic_vector(5 downto 0);
        signal ACK_NEXT : std_logic_vector(1 downto 0);
        signal WRITE_COMM  : std_logic_vector(7 downto 0);
        signal READ_COMM  : std_logic_vector(7 downto 0);
        signal READ_REG : std_logic_vector(7 downto 0);
        signal COMMAND : std_logic_vector(7 downto 0);
        signal COMMAND1 : std_logic_vector(7 downto 0);
        signal SDA_IN : std_logic;
        signal SDA_OUT : std_logic;
        signal OE : std_logic;
        signal stop_reg : std_logic;

begin

        process(RESET_N,CLK_I2C)
        begin
                STATE_STATUS <= STATE;
                DATA_OUT <= READ_REG;
                if(RESET_N = '0')then
                        COUNT <= (others=>'1');
                        READ_REG <= (others=>'0');
                        ADC_DATA_OUT <= (others=>'0');
                        SDA_OUT <= '1';
                        SCL <= '1';
                        ADC_READY <= '0';
                        OE <= '0';
                        ACK <= '0';
                        ACK_NEXT <= (others=>'0');
                        stop_reg <= '0';
                        WRITE_COMM <= b"1001_0000";
                        READ_COMM <= b"1001_0001";
                        COMMAND <= b"0000_0000";
                        COMMAND1 <= b"0100_0100";
                        STATE <= IDEL;

                elsif(rising_edge(CLK_I2C))then

                        case(STATE)is

                                when IDEL =>
                                        OE <= '1';
                                        SDA_OUT <= '1';
                                        SCL <= '1';
                                        ADC_READY <= '0';
                                        ACK_NEXT <= (others=>'0');
                                        COUNT <= (others=>'1');
                                        READ_REG <= (others=>'0');
                                        ADC_DATA_OUT <= (others=>'0');
                                        if(EN = '1') then
                                                STATE <= START;
                                        else
                                                STATE <= IDEL;
                                        end if;
                                when START =>
                                        OE <= '1';
                                        SDA_OUT <= '0';
                                        SCL <= '1';
                                        COUNT <= (others=>'1');
                                        STATE <= START_NEXT;

                                when START_NEXT =>
                                        SCL <= '0';
                                        STATE <= ADDR_WRITE;
                                when ADDR_WRITE =>
                                        OE <= '1';
                                        SDA_OUT <= WRITE_COMM(conv_integer(COUNT));
                                        STATE <= SEND_WRITE;


                                when SEND_WRITE =>
                                        SCL <= '1';
                                        if(COUNT /= 0) then
                                                STATE <= ADDR_WRITE_NEXT;
                                        else
                                                COUNT <= (others=>'1');
                                                STATE <= ADDR_WRITE_CLR;
                                        end if;
                                when ADDR_WRITE_NEXT =>
```

```
                SCL <= '0';
                COUNT <= COUNT - '1';
                STATE <= ADDR_WRITE;
        when ADDR_WRITE_CLR =>
                SCL <= '0';
                STATE <= ACK1;
        when ACK1 =>
                OE <= '0';
                SCL <= '1';
                STATE <= WAIT_ACK1;
        when WAIT_ACK1 =>
                ACK <= SDA_IN;
                STATE <= READ_ACK1;

        when READ_ACK1 =>
                SCL <= '0';
                STATE <= COMD_DEV;
        when COMD_DEV1 =>
                OE <= '1';
                SDA_OUT <=      COMMAND1(conv_integer(COUNT));
                STATE <= SEND_COMD1;

        when SEND_COMD1 =>
                SCL <= '1';

                if(COUNT /= 0) then
                        STATE <= SEND_COMD_NEXT1;
                else
                        COUNT <= (others=>'1');
                        STATE <= COMD_DEV_CLR1;
                end if;
        when SEND_COMD_NEXT1 =>
                SCL <= '0';
                COUNT <= COUNT - '1';
                STATE <= COMD_DEV1;

        when COMD_DEV_CLR1 =>
                SCL <= '0';
                STATE <= ACK4;
        when ACK4 =>
                OE <= '0';
                SCL <= '1';
                STATE <= WAIT_ACK4;
        when WAIT_ACK4 =>
                ACK <= SDA_IN;
                STATE <= READ_ACK4;

        when READ_ACK4 =>
                SCL <= '0';
                STATE <= COMD_DEV;
        when COMD_DEV =>
                OE <= '1';
                SDA_OUT <=      COMMAND(conv_integer(COUNT));
                STATE <= SEND_COMD;
        when SEND_COMD =>
                SCL <= '1';
                if(COUNT /= 0) then
                        STATE <= SEND_COMD_NEXT;
                else
                        COUNT <= (others=>'1');
                        STATE <= COMD_DEV_CLR;
                end if;
        when SEND_COMD_NEXT =>
                SCL <= '0';
                COUNT <= COUNT - '1';
                STATE <= COMD_DEV;
        when COMD_DEV_CLR =>
                SCL <= '0';
                STATE <= ACK2;
        when ACK2 =>
                OE <= '0';
                SCL <= '1';
                STATE <= WAIT_ACK2;
        when WAIT_ACK2 =>
                ACK <= SDA_IN;
                STATE <= READ_ACK2;
        when READ_ACK2 =>
                SCL <= '0';
                STATE <= STOP1;
```

```
when STOP1 =>
        OE <= '1';
        SDA_OUT <= '0';
        SCL <= '1';
        STATE <= SEND_STOP1;

when SEND_STOP1 =>
        SDA_OUT <= '1';
        SCL <= '1';
        STATE <= START1;
when START1 =>
        OE <= '1';
        SDA_OUT <= '0';
        SCL <= '1';
        COUNT <= (others=>'1');
        STATE <= START_NEXT1;
when START_NEXT1 =>
        SCL <= '0';
        STATE <= ADDR_READ;
when ADDR_READ =>
        OE <= '1';
        SDA_OUT <=      READ_COMM(conv_integer(COUNT));
        STATE <= SEND_READ;
when SEND_READ =>
        SCL <= '1';
        if(COUNT /= 0) then
                STATE <= ADDR_READ_NEXT;
        else
                COUNT <= (others=>'1');
                STATE <= ADDR_READ_CLR;
        end if;
when ADDR_READ_NEXT =>
        SCL <= '0';
        COUNT <= COUNT - '1';
        STATE <= ADDR_READ;

when ADDR_READ_CLR =>
        SCL <= '0';
        STATE <= ACK3;
when ACK3 =>
        OE <= '0';
        SCL <= '1';
        STATE <= WAIT_ACK3;
when WAIT_ACK3 =>
        ACK <= SDA_IN;
        STATE <= READ_ACK3;

when READ_ACK3 =>
        SCL <= '0';
        STATE <= READ_DATA;
when READ_DATA =>
        OE <= '0';
        SCL <= '1';
        STATE <= RECEIVE_DATA;

when RECEIVE_DATA=>
        READ_REG <= (READ_REG(6 downto 0) & SDA_IN);
        if(COUNT /= 0) then
                STATE <= READ_NEXT;
        else
                COUNT <= (others=>'1');
                STATE <= READ_CLR;
                end if;
when READ_NEXT =>
        SCL <= '0';
        COUNT <= COUNT - '1';
        STATE <= READ_DATA;

when READ_CLR =>
        SCL <= '0';
        STATE <= NACK_ST;
when NACK_ST =>
        if(READ_REG="01010101") then
                ADC_DATA_OUT <= "01010110";
        else
                ADC_DATA_OUT <= READ_REG;
        end if;
        ADC_READY <= '0';
```

```
                OE <= '1';
                SDA_OUT <= '1';
                STATE <= WAIT_NACK;

            when WAIT_NACK =>
                ADC_READY <= '1';
                SCL <= '1';
                STATE <= READ_NACK;
            when READ_NACK =>
                ADC_READY <= '0';
                SCL <= '0';
                STATE <= STOP2;
            when STOP2 =>
                OE <= '1';
                SDA_OUT <= '0';
                SCL <= '1';
                STATE <= SEND_STOP2;
            when SEND_STOP2 =>
                SDA_OUT <= '1';
                SCL <= '1';
                if(EN = '0')then
                        STATE <= IDEL;
                else
                        STATE <= START1;
                end if;
                STATE <= IDEL;
            when others =>
                STATE <= IDEL;
        end case;
    end if;
end process;
SDA <= SDA_OUT when (OE = '1') else 'Z';
SDA_IN <= SDA;
end Behavioral;
```

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

Entity ADC0820 is
Port(
        RESET : IN std_logic := '0';
        CS : OUT std_logic := '1';
        RD : OUT std_logic := '1';
        CLK : IN std_logic;
        INT: IN std_logic;
        DATA_IN: IN std_logic_vector(7 downto 0);
        DATA_OUT : OUT std_logic_vector(7 downto 0);
        RDY: OUT std_logic;
        STATE_OUT:OUT std_logic_vector(3 downto 0)
);
end ADC0820;
Architecture rtl of ADC0820 is
BEGIN
        process(CLK,RESET)
        variable count : integer range 0 to 15 := 0 ;
        variable mode  : integer range 0 to 7 := 0;
        begin
        STATE_OUT <= CONV_STD_LOGIC_VECTOR(mode,4);
        if RESET='1' then
                cs <= '1';
                rd <= '1';
                DATA_OUT <="00000000";
                mode := 0;
                count := 0;
        elsif CLK'Event and CLK = '1' then
                        case mode is
                        when 0 =>
                                cs <= '0';
                                RDY <= '0';
                                mode := 1;
                        when 1 =>
                                rd <= '0';
                                mode := 2;
                        when 2 =>
                                mode := 3;
                        when 3 =>
                                cs <= '0';
                                rd <= '0';
                                mode := 4;
                        when 4 =>
                                count := count+1;
                                if (INT = '0' or count >10) then
                                        mode := 5;
                                        count := 0;
                                else
                                        mode := 4;
                                end if;
                        when 5 =>
                                if(DATA_IN="10000001") then
                                        DATA_OUT <= "10000010";
                                        mode := 6;
                                else
                                        DATA_OUT <= DATA_IN;
                                        mode := 6;
                                end if;
                        when 6 =>
                                rd <= '1';
                                cs<= '1';
                                RDY <= '1';
                                mode := 0;
                        when others =>
                                cs <= '1';
                                rd <= '1';
                                mode := 0;
                end case;
        end if;
        End process;
End rtl;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY DATA_USB_CONTROLLER IS
	PORT
	(
		RXF :  IN  STD_LOGIC;
		CLK :  IN  STD_LOGIC;
		TXE :  IN  STD_LOGIC;
		ENA_WR :  IN  STD_LOGIC;
		RESET :  IN  STD_LOGIC;
		CLK_RD :  IN  STD_LOGIC;
		FPGA_IN :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
		USB_DATA :  INOUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
		RD :  OUT  STD_LOGIC;
		WR :  OUT  STD_LOGIC;
		OE :  OUT  STD_LOGIC;
		NEXT_TX :  OUT  STD_LOGIC;
		WR_CTRL :  OUT  STD_LOGIC;
		READ_CTRL :  OUT  STD_LOGIC;
		NEXT_RX :  OUT  STD_LOGIC;
		FPGA_OUT :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
		STATE :  OUT  STD_LOGIC_VECTOR(3 DOWNTO 0)
	);
END DATA_USB_CONTROLLER;

ARCHITECTURE bdf_type OF DATA_USB_CONTROLLER IS

COMPONENT fifo_usb_control
GENERIC (IDLE : STD_LOGIC_VECTOR(3 DOWNTO 0);
			PREPARE_READ : STD_LOGIC_VECTOR(3 DOWNTO 0);
			READ_NEXT : STD_LOGIC_VECTOR(3 DOWNTO 0);
			READ_STATE : STD_LOGIC_VECTOR(3 DOWNTO 0);
			WRITE_LOOP : STD_LOGIC_VECTOR(3 DOWNTO 0);
			WRITE_STATE : STD_LOGIC_VECTOR(3 DOWNTO 0);
			WRITE_VERIFY : STD_LOGIC_VECTOR(3 DOWNTO 0)
			);
	PORT(CLK_IN : IN STD_LOGIC;
		RESET : IN STD_LOGIC;
		TXE0 : IN STD_LOGIC;
		ENA_WR : IN STD_LOGIC;
		RXF0 : IN STD_LOGIC;
		CLK_RD : IN STD_LOGIC;
		DATA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
		WR : OUT STD_LOGIC;
		RD : OUT STD_LOGIC;
		WR_OUT : OUT STD_LOGIC;
		RD_OUT : OUT STD_LOGIC;
		OE : OUT STD_LOGIC;
		NEXT_TX : OUT STD_LOGIC;
		NEXT_RX : OUT STD_LOGIC;
		STATE_OUT : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
	);
END COMPONENT;

COMPONENT bi_direct
	PORT(ENA_READ : IN STD_LOGIC;
		ENA_WRITE : IN STD_LOGIC;
		FPGA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
		USB_DATA : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
		FPGA_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
	);
END COMPONENT;

COMPONENT reg8bit_p
	PORT(clk : IN STD_LOGIC;
		data_i : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
		data_o : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
	);
END COMPONENT;

SIGNAL	SYNTHESIZED_WIRE_6 :  STD_LOGIC;
SIGNAL	SYNTHESIZED_WIRE_1 :  STD_LOGIC;
SIGNAL	SYNTHESIZED_WIRE_7 :  STD_LOGIC;
SIGNAL	JKFF_inst3 :  STD_LOGIC;
```

```
SIGNAL  SYNTHESIZED_WIRE_5 :  STD_LOGIC_VECTOR(7 DOWNTO 0);


BEGIN
RD <= JKFF_inst3;
WR_CTRL <= SYNTHESIZED_WIRE_1;
READ_CTRL <= SYNTHESIZED_WIRE_6;
SYNTHESIZED_WIRE_7 <= '1';



b2v_inst : fifo_usb_control
GENERIC MAP(IDLE => "0000",
                        PREPARE_READ => "0100",
                        READ_NEXT => "0110",
                        READ_STATE => "0101",
                        WRITE_LOOP => "0011",
                        WRITE_STATE => "0001",
                        WRITE_VERIFY => "0010"
                        )
PORT MAP(CLK_IN => CLK,
                RESET => RESET,
                TXE0 => TXE,
                ENA_WR => ENA_WR,
                RXF0 => RXF,
                CLK_RD => CLK_RD,
                DATA_IN => FPGA_IN,
                WR => WR,
                WR_OUT => SYNTHESIZED_WIRE_1,
                OE => OE,
                NEXT_TX => NEXT_TX,
                NEXT_RX => NEXT_RX,
                STATE_OUT => STATE);


SYNTHESIZED_WIRE_6 <= NOT(RXF);



b2v_inst2 : bi_direct
PORT MAP(ENA_READ => SYNTHESIZED_WIRE_6,
                ENA_WRITE => SYNTHESIZED_WIRE_1,
                FPGA_IN => FPGA_IN,
                USB_DATA => USB_DATA,
                FPGA_OUT => SYNTHESIZED_WIRE_5);


PROCESS(CLK_RD,SYNTHESIZED_WIRE_6)
VARIABLE synthesized_var_for_JKFF_inst3 : STD_LOGIC;
BEGIN
IF (SYNTHESIZED_WIRE_6 <= '0') THEN
       synthesized_var_for_JKFF_inst3 := '1';
ELSIF (RISING_EDGE(CLK_RD)) THEN
       synthesized_var_for_JKFF_inst3  :=  (NOT(synthesized_var_for_JKFF_inst3)  AND
SYNTHESIZED_WIRE_7)           OR          (synthesized_var_for_JKFF_inst3          AND
(NOT(SYNTHESIZED_WIRE_7)));
END IF;
       JKFF_inst3 <= synthesized_var_for_JKFF_inst3;
END PROCESS;


b2v_inst4 : reg8bit_p
PORT MAP(clk => JKFF_inst3,
                data_i => SYNTHESIZED_WIRE_5,
                data_o => FPGA_OUT);



END bdf_type;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity Reg8Bit_P is
port(
        clk : in std_logic;
        data_i : in std_logic_vector(7 downto 0);
        data_o : out std_logic_vector(7 downto 0)
        );
end Reg8Bit_P;

architecture RTL of Reg8Bit_P is
begin
        process(clk)
        begin
                if clk'event and clk = '1' then
                        data_o <= data_i;
                end if;
        end process;
end RTL;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY BI_DIRECT IS
        PORT
        (
                ENA_READ :  IN  STD_LOGIC;
                ENA_WRITE :  IN  STD_LOGIC;
                FPGA_IN :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
                USB_DATA :  INOUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
                FPGA_OUT :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END BI_DIRECT;

ARCHITECTURE bdf_type OF BI_DIRECT IS


BEGIN


PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
        USB_DATA(7) <= FPGA_IN(7);
ELSE
        USB_DATA(7) <= 'Z';
END IF;
END PROCESS;

PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
        USB_DATA(6) <= FPGA_IN(6);
ELSE
        USB_DATA(6) <= 'Z';
END IF;
END PROCESS;

PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
        USB_DATA(5) <= FPGA_IN(5);
ELSE
        USB_DATA(5) <= 'Z';
END IF;
END PROCESS;
```

```
PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
       USB_DATA(4) <= FPGA_IN(4);
ELSE
       USB_DATA(4) <= 'Z';
END IF;
END PROCESS;

PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
       USB_DATA(3) <= FPGA_IN(3);
ELSE
       USB_DATA(3) <= 'Z';
END IF;
END PROCESS;

PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
       USB_DATA(2) <= FPGA_IN(2);
ELSE
       USB_DATA(2) <= 'Z';
END IF;
END PROCESS;

PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
       USB_DATA(1) <= FPGA_IN(1);
ELSE
       USB_DATA(1) <= 'Z';
END IF;
END PROCESS;

PROCESS(FPGA_IN,ENA_WRITE)
BEGIN
if (ENA_WRITE = '1') THEN
       USB_DATA(0) <= FPGA_IN(0);
ELSE
       USB_DATA(0) <= 'Z';
END IF;
END PROCESS;


PROCESS(ENA_READ,USB_DATA)
BEGIN
IF (ENA_READ <= '1') THEN
       FPGA_OUT <= USB_DATA;
END IF;
END PROCESS;


END bdf_type;
```

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

Entity FIFO_USB_CONTROL is
generic(constant IDLE: std_logic_vector(3 downto 0):= "0000";
                constant WRITE_STATE: std_logic_vector(3 downto 0):= "0001";
                constant WRITE_VERIFY: std_logic_vector(3 downto 0):= "0010";
                constant WRITE_LOOP: std_logic_vector(3 downto 0):= "0011";
                constant PREPARE_READ: std_logic_vector(3 downto 0):= "0100";
                constant READ_STATE: std_logic_vector(3 downto 0):= "0101";
                constant READ_NEXT: std_logic_vector(3 downto 0):= "0110"
                        );
port(
        CLK_IN          : IN std_logic;
        RESET           : IN std_logic;
        DATA_IN         : IN std_logic_vector(7 downto 0);
        TXE0            : IN std_logic;
        ENA_WR          : IN std_logic;
        RXF0            : IN std_logic;
        CLK_RD          : IN std_logic;
        WR                      : OUT std_logic;
        RD                      : OUT std_logic;
        WR_OUT          : OUT std_logic;
        RD_OUT          : OUT std_logic;
        OE                      : OUT std_logic;
        NEXT_TX         : OUT std_logic;
        NEXT_RX         : OUT std_logic;
        STATE_OUT:OUT std_logic_vector(3 downto 0)
        );
end FIFO_USB_CONTROL;
Architecture rtl of FIFO_USB_CONTROL is
signal STATE :std_logic_vector(3 downto 0) := IDLE;
signal SIG_DATA_CHECK :std_logic_vector(7 downto 0) ;
signal SIG_RD:std_logic;
signal SIG_WR:std_logic;
signal SIG_RD_OUT : std_logic;
signal SIG_WR_OUT:std_logic;
signal SIG_OE:std_logic;
begin
        process(CLK_IN,RESET,ENA_WR)
        variable count :integer range 0 to 10;
        Begin
                STATE_OUT <= STATE;
                RD <= SIG_RD;
                WR <= SIG_WR;
                RD_OUT <= SIG_RD_OUT;
                WR_OUT <= SIG_WR_OUT;
                OE <= SIG_OE;
        if (RESET='0') then
                SIG_OE <='1';
                SIG_WR <= '1';
                SIG_RD <= '1';
                SIG_WR_OUT<='0';
                SIG_RD_OUT<='0';
                NEXT_TX<='0';
                STATE <= IDLE;
        elsif CLK_IN'event and CLK_IN = '1'   then
                if(not(ENA_WR='1' and RXF0='0') ) then
                        case STATE is
                        when IDLE =>
                                SIG_OE <='1';
                                SIG_WR <= '1';
                                SIG_RD <= '1';
                                SIG_WR_OUT<='0';
                                SIG_RD_OUT<='0';
                                NEXT_TX<='0';
                                SIG_DATA_CHECK <= DATA_IN;
                                if(RXF0='0') then
                                        STATE <= PREPARE_READ;
                                        count := 0;
                                        elsif (ENA_WR='1' and TXE0='0' )  then
                                                STATE <= WRITE_STATE;
                                                SIG_WR_OUT<='1';
                                                else
                                                        STATE <= IDLE;
                                end if;
```

```
                when WRITE_STATE =>
                        if(TXE0='0' ) then
                                SIG_OE <='1';
                                SIG_WR <= '0';
                                SIG_RD <= '1';
                                SIG_WR_OUT<='1';
                                SIG_RD_OUT<='0';
                                NEXT_TX<='0';
                                STATE <= WRITE_VERIFY;
                        else
                                STATE <= IDLE;
                        end if;
                when WRITE_VERIFY =>
                        if(TXE0='0' ) then
                                SIG_OE <='1';
                                SIG_WR <= '1';
                                SIG_RD<= '1';
                                SIG_WR_OUT<='1';
                                SIG_RD_OUT<='0';
                                NEXT_TX<='1';
                                SIG_DATA_CHECK <= DATA_IN;
                                if(ENA_WR='1' and TXE0='0') then
                                        STATE <= WRITE_LOOP;
                                else
                                        STATE <= IDLE;
                                end if;
                        else
                                STATE <= WRITE_VERIFY;
                        end if;
                when WRITE_LOOP=>
                        if(TXE0='0' ) then
                                SIG_OE <='1';
                                SIG_WR <= '0';
                                SIG_RD<= '1';
                                SIG_WR_OUT<='1';
                                SIG_RD_OUT<='0';
                                NEXT_TX<='0';
                                STATE <= WRITE_VERIFY;
                        else
                                STATE <= IDLE;
                                SIG_WR <= '1';
                                SIG_WR_OUT<='1';
                        end if;
                when PREPARE_READ=>
                        SIG_OE <='0';
                        STATE<=READ_STATE;
                when READ_STATE =>
                        if(RXF0='0') then
                                SIG_RD_OUT<='1';
                                STATE<=READ_NEXT;
                        else
                                SIG_RD_OUT<='0';
                                STATE <= IDLE;
                        end if;
                when READ_NEXT =>
                        if(RXF0='0') then
                                SIG_RD_OUT<='0';
                                STATE<=IDLE;
                        else
                                SIG_RD_OUT<='0';
                                STATE <= IDLE;
                        end if;
                when others =>
                        STATE <= IDLE;
                end case;
                else
                        SIG_OE <='1';
                        SIG_WR <= '1';
                        SIG_RD <= '1';
                        SIG_WR_OUT<='0';
                        SIG_RD_OUT<='0';
                        NEXT_TX<='0';
                        STATE <= IDLE;
                        SIG_DATA_CHECK <= DATA_IN;
                end if;
        end if;
End Process;
end rtl;
```

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

entity CONTROL_UNIT is
generic(constant IDLE: std_logic_vector(3 downto 0) := "0000";
                constant READ_PARA: std_logic_vector(3 downto 0) := "0001";
                constant READ_I2C: std_logic_vector(3 downto 0) := "0010";
                constant READ_SPI: std_logic_vector(3 downto 0) := "0011";
                constant READ_ONEWIRE: std_logic_vector(3 downto 0) := "0100";
                constant WRITE_PARALLEL_START: std_logic_vector(3 downto 0) := "0101";
                constant WRITE_I2C_START: std_logic_vector(3 downto 0) := "0110";
                constant WRITE_SPI_START: std_logic_vector(3 downto 0) := "0111";
                constant WRITE_ONEWIRE_START: std_logic_vector(3 downto 0) := "1000";
                constant WRITE_STATE: std_logic_vector(3 downto 0) := "1001";
                constant WRITE_PARA: std_logic_vector(3 downto 0) := "1010";
                constant WRITE_I2C: std_logic_vector(3 downto 0) := "1011";
                constant WRITE_SPI: std_logic_vector(3 downto 0) := "1100";
                constant WRITE_ONEWIRE: std_logic_vector(3 downto 0) := "1101";
                constant READ_DATA: std_logic_vector(3 downto 0) := "1110"
                        );
port(
                CLK : in std_logic;
                RESET : in std_logic:='1';
                EMP_PARA : in std_logic:='1';
                EMP_I2C : in std_logic:='1';
                EMP_SPI : in std_logic:='1';
                EMP_ONEWIRE : in std_logic:='1';
                TX_FINISH : in std_logic;
                RXF_IN : in std_logic:='1';
                SYN_100ms : in std_logic:='1';
                ENCODE_DATA_OUT : out std_logic_vector(7 downto 0);
                ADDS_OUT : out std_logic_vector(2 downto 0);
                STATE_OUT : out std_logic_vector(3 downto 0);
                RESET_OUT : OUT std_logic;
                WR_ENA_OUT : OUT std_logic;
                RD_PARA : OUT std_logic;
                RD_I2C : OUT std_logic;
                RD_SPI : OUT std_logic;
                RD_ONEWIRE : OUT std_logic
                );
end CONTROL_UNIT;

architecture rtl of CONTROL_UNIT is
signal STATE :std_logic_vector(3 downto 0) := IDLE;
signal SIG_RESET_OUT : std_logic;
signal SIG_RDY : std_logic:='0';
signal SIG_ADDS_OUT : std_logic_vector(2 downto 0);
begin
        RESET_OUT <= SIG_RESET_OUT;
        STATE_OUT <= STATE;
        ADDS_OUT<= SIG_ADDS_OUT;
        process(CLK,RESET,EMP_PARA,EMP_I2C,EMP_SPI,EMP_ONEWIRE,RXF_IN)
        variable count : integer range 0 to 20:=0;
        variable next_state :std_logic_vector(3 downto 0) := IDLE;
        begin
                if (RESET='0')then
                        SIG_RESET_OUT <= '0';
                        STATE <= IDLE;
                        elsif CLK'event and CLK = '1' then
                                case STATE is
                                        when IDLE =>
                                                SIG_RESET_OUT <= '0';
                                                RD_PARA <= '0';
                                                RD_I2C <= '0';
                                                WR_ENA_OUT <= '0';
                                                if(EMP_PARA='0') then
                                                STATE <= READ_PARA;
                                                elsif(EMP_I2C='0') then
                                                STATE <= READ_I2C;
                                                elsif(EMP_SPI='0') then
                                                        STATE <= READ_SPI;
                                                elsif(EMP_ONEWIRE='0') then
                                                        STATE <= READ_ONEWIRE;
                                                elsif(RXF_IN='0') then
                                                        count := 0;
                                                        STATE <= READ_DATA;
                                                else
                                                        STATE <= IDLE;
```

```
                end if;
        when READ_PARA=>
                if(EMP_PARA='0') then
                        SIG_RESET_OUT <= '0';
                        RD_PARA <= '1';
                        WR_ENA_OUT <= '0';
                        STATE   <=  WRITE_PARALLEL_START;

                        SIG_ADDS_OUT <= "000";
                        ENCODE_DATA_OUT <="10000001";

                else
                        STATE <= READ_I2C;
                end if;
        when READ_I2C=>
                if(EMP_I2C='0') then
                        SIG_RESET_OUT <= '0';
                        RD_I2C<= '1';
                        WR_ENA_OUT <= '0';
                        STATE <= WRITE_I2C_START;

                        SIG_ADDS_OUT <= "000";
                        ENCODE_DATA_OUT <="01010101";

                else
                        STATE <= READ_SPI;
                end if;
        when READ_SPI=>
                if(EMP_SPI='0') then
                        SIG_RESET_OUT <= '0';
                        RD_SPI<= '1';
                        WR_ENA_OUT <= '0';
                        STATE <= WRITE_SPI_START;

                        SIG_ADDS_OUT <= "000";
                        ENCODE_DATA_OUT <="10101010";
                else
                        STATE <= READ_ONEWIRE;
                end if;
        when READ_ONEWIRE=>
                if(EMP_ONEWIRE='0') then
                        SIG_RESET_OUT <= '0';
                        RD_ONEWIRE<= '1';
                        WR_ENA_OUT <= '0';
                        STATE <= WRITE_ONEWIRE_START;
                        SIG_ADDS_OUT <= "000";
                        ENCODE_DATA_OUT <="01111110";
                else
                        STATE <= IDLE;
                end if;
        when READ_DATA=>
                count := count+1;
                SIG_RESET_OUT <= '1';
                if(count >= 6 or RXF_IN='1') then
                        SIG_RESET_OUT <= '0';
                        STATE <= IDLE;
                        count := 0;
                else
                        STATE <= READ_DATA;
                End if;
        when WRITE_PARALLEL_START=>
                SIG_RESET_OUT <= '1';
                RD_PARA <= '0';
                WR_ENA_OUT <= '1';
                STATE <= WRITE_STATE;
                next_state := WRITE_PARA;
                SIG_ADDS_OUT <= "000";
                count := 0;
        when WRITE_I2C_START=>
                SIG_RESET_OUT <= '1';
                RD_I2C <= '0';
                WR_ENA_OUT <= '1';
                STATE <= WRITE_STATE;
                next_state := WRITE_I2C;
                SIG_ADDS_OUT <= "000";
                count := 0;
        when WRITE_SPI_START=>
                SIG_RESET_OUT <= '1';
                RD_SPI <= '0';
```

```
                WR_ENA_OUT <= '1';
                STATE <= WRITE_STATE;
                next_state := WRITE_SPI;
                SIG_ADDS_OUT <= "000";
                count := 0;
            when WRITE_ONEWIRE_START=>
                SIG_RESET_OUT <= '1';
                RD_ONEWIRE <= '0';
                WR_ENA_OUT <= '1';
                STATE <= WRITE_STATE;
                next_state := WRITE_ONEWIRE;
                SIG_ADDS_OUT <= "000";
                count := 0;
            when WRITE_STATE =>
                count := count+1;
                SIG_RESET_OUT <= '1';
                if(count >= 20) then
                        SIG_RESET_OUT <= '0';
                        STATE <= IDLE;
                        count := 0;
                        elsif(TX_FINISH='1') then
                                SIG_RESET_OUT <= '0';
                                STATE <= next_state;
                                count := 0;
                        end if;
            when WRITE_PARA =>
                        SIG_RESET_OUT <= '1';
                        RD_PARA <= '0';
                        WR_ENA_OUT <= '1';
                        STATE <= WRITE_STATE;
                        SIG_ADDS_OUT <= "001";
                        count := 0;
                        next_state := READ_PARA;

            when WRITE_I2C =>
                        SIG_RESET_OUT <= '1';
                        RD_I2C<= '0';
                        WR_ENA_OUT <= '1';
                        STATE <= WRITE_STATE;
                        SIG_ADDS_OUT <= "010";
                        count := 0;
                        next_state := READ_I2C;

             when WRITE_SPI =>
                        SIG_RESET_OUT <= '1';
                        RD_SPI <= '0';
                        WR_ENA_OUT <= '1';
                        STATE <= WRITE_STATE;
                        SIG_ADDS_OUT <= "011";
                        count := 0;
                        next_state := READ_SPI;
            when WRITE_ONEWIRE =>
                        SIG_RESET_OUT <= '1';
                        RD_ONEWIRE<= '0';
                        WR_ENA_OUT <= '1';
                        STATE <= WRITE_STATE;
                        SIG_ADDS_OUT <= "100";
                        count := 0;
                        next_state := READ_ONEWIRE;

            when others =>
                        STATE <= IDLE;
        end case;
    end if;
  end process;
end rtl;
```

ภาคผนวก ข.

โปรแกรมภาษาวิชวลซีพลัสพลัส

```
#pragma once
#include "afxwin.h"
#include "source/ntgraph.h"
#include "afxcmn.h"

class CDAQView : public CFormView
{
public:
        HICON hRedLedOn ;
        HICON hRedLedOff ;
        HICON hGreenLedOn ;
        HICON hGreenLedOff ;
public:
        BOOL SendData(unsigned char  chSend);
        void UpdateLEDData(unsigned char chData);
        void I2CPlot(void);
        void SPIPlot(void);
        void ParaPlot(void);
        void ParaPlotOffline(void);
        void ONEWIREPlot(void);
        void DisConnect();
        void InitGraph(CNTGraph &m_graph);
        void SettingParaGraph(double &Xlower, double &Xupper, double &Ylower, double
&Yupper);
        void SettingSPIGraph(double &Xlower, double &Xupper, double &Ylower, double
&Yupper);
        void SettingI2CGraph(double &Xlower, double &Xupper, double &Ylower, double
&Yupper);
protected:
        CDAQView();
        DECLARE_DYNCREATE(CDAQView)

public:
        enum{ IDD = IDD_DAQ_FORM };
public:
        CDAQDoc* GetDocument() const;
public:
        public:
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
        virtual void DoDataExchange(CDataExchange* pDX);
        virtual void OnInitialUpdate();

public:
        virtual ~CDAQView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:
protected:
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnBnClickedConnect();
public:
        afx_msg void OnBnClickedSearch();
public:
        CListBox m_clist_device;
public:
        CStatic m_cLedSearch;
public:
        CStatic m_cLedConnect;
public:
        void SendMsgToList(CString str);
public:
        CListBox m_list_msg;
public:
        afx_msg void OnClose();
public:
        afx_msg void OnDestroy();
public:
        CNTGraph m_ParaGraph;
public:
        BOOL m_bParaStop;
public:
        afx_msg void OnBnClickedParastop();
```

```
        afx_msg void OnBnClickedAllstop();
public:
        afx_msg void OnTimer(UINT_PTR nIDEvent);
public:
        CComboBox m_para_voltdiv;
public:
        CComboBox m_para_timediv;
public:
        CNTGraph m_spiGraph;
public:
        CComboBox m_spi_voltdiv;
public:
        CComboBox m_spi_timediv;
public:
        CNTGraph m_I2Cgraph;
public:
        CComboBox m_i2c_voltdiv;
public:
        CComboBox m_i2c_timediv;
public:
        BOOL m_bSPIstop;
public:
        afx_msg void OnBnClickedSpistop();
public:
        BOOL m_bi2cStop;
public:
        afx_msg void OnBnClickedI2cstop();
public:
        CEdit m_cEditSend;
public:
        CString m_strEditSend;
public:
        afx_msg void OnBnClickedSend();
public:
        CStatic m_D7;
public:
        CStatic m_D6;
public:
        CStatic m_D5;
public:
        CStatic m_D4;
public:
        CStatic m_D3;
public:
        CStatic m_D2;
public:
        CStatic m_D1;
public:
        CStatic m_D0;
public:
        virtual BOOL PreTranslateMessage(MSG* pMsg);
public:
        CMatrixStatic m_LCDTempdata;
public:
        CMatrixStatic m_LCDtemp;
public:
        BOOL m_b1wireStop;
public:
        afx_msg void OnBnClicked1wirestop();
public:
        afx_msg void OnBnClickedLogsetting();
public:
        void OnLogSettingOk(void);
public:
        BOOL m_bOffLine;
public:
        afx_msg void OnBnClickedChkOffline();
public:
        afx_msg void OnBnClickedLoadPara();
public:
        afx_msg void OnBnClickedLoadSpi();
public:
        afx_msg void OnBnClickedLoadI2c();
public:
        afx_msg void OnBnClickedLoadOnewire();
public:
        afx_msg void OnBnClickedSaveParallel();
public:
        afx_msg void OnBnClickedSaveSpi();
```

```
public:
        afx_msg void OnBnClickedSaveI2c();
public:
        afx_msg void OnBnClickedSaveOnewire();
public:
        afx_msg void OnBnClickedClearListCtrl();
public:
        CStatic m_staBitRatePara;
public:
        CStatic m_staAverage;
public:
        CEdit m_edit_offset_para;
public:
        int m_iEditOffsetPara;
public:
        float CalculateVoltPara(void);
        float CalculateFreqPara(void);
        float CalculateVoltSPI(void);
        float CalculateFreqSPI(void);
        float CalculateVoltI2C(void);
        float CalculateFreqI2C(void);
        int FindTrigListPara(POSITION PosStart);
        int FindTrigListSPI(POSITION PosStart);
        int FindTrigListI2C(POSITION PosStart);
        void CheckLevelPara(void);
        void CheckLevelSPI(void);
        void CheckLevelI2C(void);
        void CheckLevel1WIRE(float fData);
public:
        CButton m_chk_trig_para;
public:
        BOOL m_bChkTrigPara;
public:
        CSpinButtonCtrl m_spin_offset_para;
public:
        CButton m_chk_trig_spi;
public:
        BOOL m_bChkTrigSPI;
public:
        CButton m_chk_trig_i2c;
public:
        BOOL m_bChkTrigI2c;
public:
        CStatic m_staBitRateSpi;
public:
        CStatic m_staBitRateI2c;
public:
        CEdit m_editOffSetSPI;
public:
        int m_iEditOffSetSPI;
public:
        CSpinButtonCtrl m_spin_offset_spi;
public:
        CSpinButtonCtrl m_spin_offset_i2c;
public:
        CEdit m_EditOffsetI2c;
public:
        int m_iEditOffsetI2c;
public:
        CStatic m_staBitrateMain;
public:
        CStatic m_staBitRate1Wire;
public:
        BOOL m_bAllStop;
public:
        CEdit m_edit_trig_para;
public:
        int m_iedit_trig_para;
public:
        CSpinButtonCtrl m_spin_trig_para;
public:
        CEdit m_edit_trig_spi;
public:
        int m_iedit_trig_spi;
public:
        CEdit m_edit_trig_i2c;
public:
        int m_iedit_trig_i2c;
public:
```

```
        CSpinButtonCtrl m_spin_trig_spi;
public:
        CSpinButtonCtrl m_spin_trig_i2c;
public:
        float m_fvolt_para;
public:
        CEdit m_edit_volt_para;
public:
        CEdit m_edit_volt_spi;
public:
        float m_fvolt_spi;
public:
        CEdit m_edit_volt_i2c;
public:
        float m_fvolt_i2c;
public:
        CEdit m_edit_freq_para;
public:
        float m_fFreq_para;
public:
        CEdit m_edit_freq_spi;
public:
        float m_fFreqspi;
public:
        CEdit m_edit_freq_i2c;
public:
        float m_fFreqi2c;
public:
        afx_msg void OnBnClickedOption();
};

#ifndef _DEBUG
inline CDAQDoc* CDAQView::GetDocument() const
   { return reinterpret_cast<CDAQDoc*>(m_pDocument); }
#endif
```

```
// DAQView.cpp : implementation of the CDAQView class
//

#include "stdafx.h"
#include "DAQ.h"
#include "DAQDoc.h"
#include "DAQView.h"
#include "LogSettingDlg.h"
#include "DAQGlobal.h"
#include "OptionDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
IMPLEMENT_DYNCREATE(CDAQView, CFormView)
BEGIN_MESSAGE_MAP(CDAQView, CFormView)
	ON_BN_CLICKED(IDC_CONNECT, &CDAQView::OnBnClickedConnect)
	ON_BN_CLICKED(IDC_SEARCH, &CDAQView::OnBnClickedSearch)
	ON_WM_CLOSE()
	ON_WM_DESTROY()
	ON_BN_CLICKED(IDC_PARASTOP, &CDAQView::OnBnClickedParastop)
	ON_WM_TIMER()
	ON_BN_CLICKED(IDC_ALLSTOP,&CDAQView::OnBnClickedAllstop)
	ON_BN_CLICKED(IDC_SPISTOP, &CDAQView::OnBnClickedSpistop)
	ON_BN_CLICKED(IDC_I2CSTOP, &CDAQView::OnBnClickedI2cstop)
	ON_BN_CLICKED(IDC_SEND, &CDAQView::OnBnClickedSend)
	ON_BN_CLICKED(IDC_1WIRESTOP, &CDAQView::OnBnClicked1wirestop)
	ON_BN_CLICKED(IDC_LOGSETTING, &CDAQView::OnBnClickedLogsetting)
	ON_BN_CLICKED(IDC_CHK_OFFLINE, &CDAQView::OnBnClickedChkOffline)
	ON_BN_CLICKED(IDC_LOAD_PARA, &CDAQView::OnBnClickedLoadPara)
	ON_BN_CLICKED(IDC_LOAD_SPI, &CDAQView::OnBnClickedLoadSpi)
	ON_BN_CLICKED(IDC_LOAD_I2C, &CDAQView::OnBnClickedLoadI2c)
	ON_BN_CLICKED(IDC_LOAD_ONEWIRE, &CDAQView::OnBnClickedLoadOnewire)
	ON_BN_CLICKED(IDC_SAVE_PARALLEL, &CDAQView::OnBnClickedSaveParallel)
	ON_BN_CLICKED(IDC_SAVE_SPI, &CDAQView::OnBnClickedSaveSpi)
	ON_BN_CLICKED(IDC_SAVE_I2C, &CDAQView::OnBnClickedSaveI2c)
	ON_BN_CLICKED(IDC_SAVE_ONEWIRE, &CDAQView::OnBnClickedSaveOnewire)
	ON_BN_CLICKED(IDC_CLEARLISTCTRL,&CDAQView::OnBnClickedClearListCtrl)
	ON_BN_CLICKED(IDC_OPTION, &CDAQView::OnBnClickedOption)
END_MESSAGE_MAP()
CDAQView::CDAQView()
	: CFormView(CDAQView::IDD)
	, m_bParaStop(FALSE)
	, m_bSPIstop(FALSE)
	, m_bi2cStop(FALSE)
	, m_strEditSend(_T(""))
	, m_b1wireStop(FALSE)
	, m_bOffLine(FALSE)
	, m_iEditOffsetPara(0)
	, m_bChkTrigPara(FALSE)
	, m_bChkTrigSPI(FALSE)
	, m_bChkTrigI2c(FALSE)
	, m_iEditOffSetSPI(0)
	, m_iEditOffsetI2c(0)
	, m_bAllStop(FALSE)
	, m_iedit_trig_para(0)
	, m_iedit_trig_spi(0)
	, m_iedit_trig_i2c(0)
	, m_fvolt_para(0)
	, m_fvolt_spi(0)
	, m_fvolt_i2c(0)
	, m_fFreq_para(0)
	, m_fFreqspi(0)
	, m_fFreqi2c(0)
{

}

CDAQView::~CDAQView()
{
	bContinue = FALSE;
	if(ftHandle)
	{
		FT_Close(ftHandle);
	}
	if(hEvent)
	{
		hEvent.SetEvent();
	}
	if (pReadThread)
```

```
        {
                WaitForSingleObject(pReadThread->m_hThread,INFINITE);
        }
}

void CDAQView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        DDX_Control(pDX, IDC_LIST_DEVICE, m_clist_device);
        DDX_Control(pDX, IDC_PIC_SEARCH, m_cLedSearch);
        DDX_Control(pDX, IDC_PIC_CONNECT, m_cLedConnect);
        DDX_Control(pDX, IDC_LIST_MESSAGE, m_list_msg);
        DDX_Control(pDX, IDC_PARAGRAPH, m_ParaGraph);
        DDX_Check(pDX, IDC_PARASTOP, m_bParaStop);
        DDX_Control(pDX, IDC_COMBO_PARA_VOLTDIV, m_para_voltdiv);
        DDX_Control(pDX, IDC_COMBO_PARA_TIMEDIV, m_para_timediv);
        DDX_Control(pDX, IDC_SPIGRAPH, m_spiGraph);
        DDX_Control(pDX, IDC_COMBO_SPI_VOLTDIV, m_spi_voltdiv);
        DDX_Control(pDX, IDC_COMBO_SPI_TIMEDIV, m_spi_timediv);
        DDX_Control(pDX, IDC_I2CGRAPH, m_I2Cgraph);
        DDX_Control(pDX, IDC_COMBO_I2CVOLTDIV, m_i2c_voltdiv);
        DDX_Control(pDX, IDC_COMBO_I2CTIMEDIV, m_i2c_timediv);
        DDX_Check(pDX, IDC_SPISTOP, m_bSPIstop);
        DDX_Check(pDX, IDC_I2CSTOP, m_bi2cStop);
        DDX_Control(pDX, IDC_EDITSEND, m_cEditSend);
        DDX_Text(pDX, IDC_EDITSEND, m_strEditSend);
        DDX_Control(pDX, IDC_LED_D7, m_D7);
        DDX_Control(pDX, IDC_LED_D6, m_D6);
        DDX_Control(pDX, IDC_LED_D5, m_D5);
        DDX_Control(pDX, IDC_LED_D4, m_D4);
        DDX_Control(pDX, IDC_LED_D3, m_D3);
        DDX_Control(pDX, IDC_LED_D2, m_D2);
        DDX_Control(pDX, IDC_LED_D1, m_D1);
        DDX_Control(pDX, IDC_LED_D0, m_D0);
        DDX_Control(pDX, IDC_LCDTEMPDATA, m_LCDTempdata);
        DDX_Control(pDX, IDC_LCDTEMP, m_LCDtemp);
        DDX_Check(pDX, IDC_1WIRESTOP, m_b1wireStop);
        DDX_Check(pDX, IDC_CHK_OFFLINE, m_bOffLine);
        DDX_Control(pDX, IDC_STA_BITRATE_PARA, m_staBitRatePara);
        DDX_Control(pDX, IDC_STA_AVERAGE, m_staAverage);
        DDX_Control(pDX, IDC_EDIT_OFFSET_PARA, m_edit_offset_para);
        DDX_Text(pDX, IDC_EDIT_OFFSET_PARA, m_iEditOffsetPara);
        DDX_Control(pDX, IDC_CHK_TRIG, m_chk_trig_para);
        DDX_Check(pDX, IDC_CHK_TRIG, m_bChkTrigPara);
        DDX_Control(pDX, IDC_SPIN_OFFSET_PARA, m_spin_offset_para);
        DDX_Control(pDX, IDC_CHK_TRIG_SPI, m_chk_trig_spi);
        DDX_Check(pDX, IDC_CHK_TRIG_SPI, m_bChkTrigSPI);
        DDX_Control(pDX, IDC_CHK_TRIG_I2C, m_chk_trig_i2c);
        DDX_Check(pDX, IDC_CHK_TRIG_I2C, m_bChkTrigI2c);
        DDX_Control(pDX, IDC_STA_BITRATE_SPI, m_staBitRateSpi);
        DDX_Control(pDX, IDC_STA_BITRATE_I2C, m_staBitRateI2c);
        DDX_Control(pDX, IDC_EDIT_OFFSET_SPI, m_editOffSetSPI);
        DDX_Text(pDX, IDC_EDIT_OFFSET_SPI, m_iEditOffSetSPI);
        DDX_Control(pDX, IDC_SPIN_OFFSET_SPI, m_spin_offset_spi);
        DDX_Control(pDX, IDC_SPIN_OFFSET_I2C, m_spin_offset_i2c);
        DDX_Control(pDX, IDC_EDIT_OFFSET_I2C, m_EditOffsetI2c);
        DDX_Text(pDX, IDC_EDIT_OFFSET_I2C, m_iEditOffsetI2c);
        DDX_Control(pDX, IDC_STA_BITRATE_MAIN, m_staBitrateMain);
        DDX_Control(pDX, IDC_STA_BITRATE_1WIRE, m_staBitRate1Wire);
        DDX_Check(pDX, IDC_ALLSTOP, m_bAllStop);
        DDX_Control(pDX, IDC_EDIT_TRIG_PARA, m_edit_trig_para);
        DDX_Text(pDX, IDC_EDIT_TRIG_PARA, m_iedit_trig_para);
        DDX_Control(pDX, IDC_SPIN_TRIG_PARA, m_spin_trig_para);
        DDX_Control(pDX, IDC_EDIT_TRIG_SPI, m_edit_trig_spi);
        DDX_Text(pDX, IDC_EDIT_TRIG_SPI, m_iedit_trig_spi);
        DDX_Control(pDX, IDC_EDIT_TRIG_I2C, m_edit_trig_i2c);
        DDX_Text(pDX, IDC_EDIT_TRIG_I2C, m_iedit_trig_i2c);
        DDX_Control(pDX, IDC_SPIN_TRIG_SPI, m_spin_trig_spi);
        DDX_Control(pDX, IDC_SPIN_TRIG_I2C, m_spin_trig_i2c);
        DDX_Text(pDX, IDC_EDIT_VOLT, m_fvolt_para);
        DDX_Control(pDX, IDC_EDIT_VOLT, m_edit_volt_para);
        DDX_Control(pDX, IDC_EDIT_VOLT_SPI, m_edit_volt_spi);
        DDX_Text(pDX, IDC_EDIT_VOLT_SPI, m_fvolt_spi);
        DDX_Control(pDX, IDC_EDIT_VOLT_I2C, m_edit_volt_i2c);
        DDX_Text(pDX, IDC_EDIT_VOLT_I2C, m_fvolt_i2c);
        DDX_Control(pDX, IDC_EDIT_FREQ_PARA, m_edit_freq_para);
        DDX_Text(pDX, IDC_EDIT_FREQ_PARA, m_fFreq_para);
        DDX_Control(pDX, IDC_EDIT_FREQ_SPI, m_edit_freq_spi);
        DDX_Text(pDX, IDC_EDIT_FREQ_SPI, m_fFreqspi);
```

```
	DDX_Control(pDX, IDC_EDIT_FREQ_I2C, m_edit_freq_i2c);
	DDX_Text(pDX, IDC_EDIT_FREQ_I2C, m_fFreqi2c);
}

BOOL CDAQView::PreCreateWindow(CREATESTRUCT& cs)
{

	return CFormView::PreCreateWindow(cs);
}

void CDAQView::OnInitialUpdate()
{
	CFormView::OnInitialUpdate();
	GetParentFrame()->RecalcLayout();
	ResizeParentToFit();
	hRedLedOn = AfxGetApp()->LoadIcon(IDI_RED_LED_ON);
	hRedLedOff = AfxGetApp()->LoadIcon(IDI_RED_LED_OFF);
	hGreenLedOn = AfxGetApp()->LoadIcon(IDI_GREEN_LED_ON);
	hGreenLedOff = AfxGetApp()->LoadIcon(IDI_GREEN_LED_OFF);
	OnBnClickedSearch();
	InitGraph(m_ParaGraph);
	InitGraph(m_spiGraph);
	InitGraph(m_I2Cgraph);
	m_para_voltdiv.SetCurSel(2);
	m_para_timediv.SetCurSel(1);
	m_spi_voltdiv.SetCurSel(2);
	m_spi_timediv.SetCurSel(1);
	m_i2c_voltdiv.SetCurSel(2);
	m_i2c_timediv.SetCurSel(3);
	m_LCDtemp.SetNumberOfLines(2);
	m_LCDtemp.SetXCharsPerLine(19);
	m_LCDtemp.SetSize(CMatrixStatic::SMALL);
	m_LCDtemp.SetDisplayColors(RGB(0, 0, 0), RGB(0, 255, 0), RGB(0, 70, 30));
	m_LCDtemp.AdjustClientXToSize(18);
	m_LCDtemp.AdjustClientYToSize(1);
	m_LCDtemp.SetText(_T(" One-Wire Protocol Temperature Sensor.."));
	m_LCDtemp.SetAutoPadding(true);
	m_LCDTempdata.SetNumberOfLines(1);
	m_LCDTempdata.SetXCharsPerLine(10);
	m_LCDTempdata.SetSize(CMatrixStatic::LARGE);
	m_LCDTempdata.SetDisplayColors(RGB(0, 0, 0), RGB(255, 0, 0), RGB(70, 0, 30));
	m_LCDTempdata.AdjustClientXToSize(10);
	m_LCDTempdata.AdjustClientYToSize(1);
	m_LCDTempdata.SetText(_T("  Data  "));
	m_LCDTempdata.SetAutoPadding(true);
	LogDlg = new CLogSettingDlg;
	LogDlg->Create(IDD_LOGSETTINGDLG,this);
	LogDlg->SetWindowPos(NULL,500,500,0,0,SWP_NOSIZE);
	m_bChkTrigPara = TRUE;
	m_spin_offset_para.SetBuddy(&m_edit_offset_para);
	m_spin_offset_para.SetPos(0);
	m_spin_offset_para.SetRange(0,1000);
	m_spin_trig_para.SetBuddy(&m_edit_trig_para);
	m_spin_trig_para.SetPos(0);
	m_spin_trig_para.SetRange(1,1000);
	m_iedit_trig_para = 500;
	m_bChkTrigSPI = TRUE;
	m_spin_offset_spi.SetBuddy(&m_editOffSetSPI);
	m_spin_offset_spi.SetPos(0);
	m_spin_offset_spi.SetRange(0,1000);
	m_spin_trig_spi.SetBuddy(&m_edit_trig_spi);
	m_spin_trig_spi.SetPos(0);
	m_spin_trig_spi.SetRange(1,1000);
	m_iedit_trig_spi = 500;
	m_bChkTrigI2c = TRUE;
	m_spin_offset_i2c.SetBuddy(&m_EditOffsetI2c);
	m_spin_offset_i2c.SetPos(0);
	m_spin_offset_i2c.SetRange(0,1000);
	m_spin_trig_i2c.SetBuddy(&m_edit_trig_i2c);
	m_spin_trig_i2c.SetPos(0);
	m_spin_trig_i2c.SetRange(1,1000);
	m_iedit_trig_i2c = 500;
	UpdateData(FALSE);
}
#ifdef _DEBUG
```

```
#endif
void CDAQView::OnBnClickedSearch()
{
	CWaitCursor wait;
	CString str="";

	SetDlgItemText(IDC_STATIC_NUMBER_OF_DEVICE,str);
	m_clist_device.ResetContent();
	UpdateData(FALSE);
	FT_STATUS ftStatus;
	DWORD numDevs;
	char *BufPtrs[64];

	ftStatus = FT_ListDevices(&numDevs, NULL, FT_LIST_NUMBER_ONLY);
	if(ftStatus == FT_OK)
	{
		if(numDevs>0)
		{
			ftStatus = FT_ListDevices(&numDevs, NULL,
FT_LIST_NUMBER_ONLY);
			if(ftStatus == FT_OK)
			{
				str.Format("Found %d device=>>", (int)numDevs);
				SendMsgToList(str);
				DWORD d;
				for(d=0; d<numDevs; d++)
					BufPtrs[d] = new char[64];
				BufPtrs[d] = NULL;

				ftStatus = FT_ListDevices(BufPtrs, &numDevs,
FT_LIST_ALL|FT_OPEN_BY_DESCRIPTION);
				if (FT_SUCCESS(ftStatus))
				{
					m_cLedSearch.SetIcon(hRedLedOn);
					for(DWORD u=0; u<numDevs; u++)
					{
						str.Format("%s", BufPtrs[u]);
						m_clist_device.AddString(str);
						m_clist_device.SetCurSel(0);
						SendMsgToList(str);
					}
				}
				else
				{
					str.Format("ListDevices failed");
					m_cLedSearch.SetIcon(hRedLedOff);
					UpdateData(FALSE);
					MessageBox(str,NULL,MB_OK|MB_ICONERROR);
					SendMsgToList(str);
				}
				for(d=0; d<numDevs; d++)
				{
					delete BufPtrs[d];
				}
			}
		}
		else
		{
			CString str = "Can not found any device!!";
			MessageBox(str,NULL,MB_OK|MB_ICONERROR);
			SendMsgToList(str);
			UpdateData(FALSE);
		}
	}
}
void CDAQView::OnBnClickedConnect()
{
	CString DeviceName;
```

```
	FT_STATUS ftStatus;

	CWaitCursor wait;
	if (m_clist_device.GetCurSel()==LB_ERR)
	{
		CString str="No Device Select in List Box!!";
		SendMsgToList(str);
		return;
	}
	m_clist_device.GetText(m_clist_device.GetCurSel(),DeviceName);
	if (bConnect==FALSE)
	{
		ftStatus = FT_OpenEx((PVOID)(LPCTSTR)DeviceName,
			FT_OPEN_BY_DESCRIPTION,&ftHandle);
		if(ftStatus>0)
		{
			CString str = "Can not open device!!";
			MessageBox(str,NULL,MB_OK|MB_ICONERROR);
			SendMsgToList(str);
		}
		else
		{

			FT_ResetDevice(ftHandle);
			FT_Purge(ftHandle,FT_PURGE_RX || FT_PURGE_TX);
			FT_SetTimeouts(ftHandle, 100, 100);
			DWORD BufferTransferSize=63488;
			if (FT_SetUSBParameters(ftHandle,BufferTransferSize,0)!=FT_OK)
			{
				CString str = "FT_SetUSBParameters ERROR!!!";
				MessageBox(str,NULL,MB_ICONERROR);
				SendMsgToList(str);
				return;
			}
			FT_ResetDevice(ftHandle);
			ZeroMemory(ONEWIREData,sizeof(float)*10);
			ParaDataList.RemoveAll();
			SpiDataList.RemoveAll();
			I2CDataList.RemoveAll();
			pReadThread = AfxBeginThread(ReadProcThread,this);
			FT_SetEventNotification(ftHandle, FT_EVENT_RXCHAR, hEvent);
			GetDlgItem(IDC_SEARCH)->EnableWindow(FALSE);
			GetDlgItem(IDC_CONNECT)->EnableWindow(TRUE);
			SetDlgItemText(IDC_CONNECT,"Disconnect");
			CString str = "Connected to USB Port";
			SendMsgToList(str);
			m_cLedConnect.SetIcon(hGreenLedOn);
			m_cLedSearch.SetIcon(hRedLedOn);
			bConnect = TRUE;
			SendData(0);
			Sleep(500);
			KillTimer(TimerDrawData);
			TimerDrawData = SetTimer(10,TimeLoop,0);
		}
	}else
	{
		DisConnect();
	}
}

void CDAQView::DisConnect(void)
{
	if (!bConnect)
	{
		return;
	}
	bConnect = FALSE;
	bContinue = FALSE;
	KillTimer(TimerDrawData);
	KillTimer(TimerParallel);
	if(hEvent)
	{
		hEvent.SetEvent();
	}
	Sleep(100);
	if(ftHandle)
	{
		FT_Close(ftHandle);
	}
```

```
        SetDlgItemText(IDC_CONNECT,"Connect");
        CString str = "Disconnected From USB Port";
        SendMsgToList(str);
        GetDlgItem(IDC_SEARCH)->EnableWindow(TRUE);
        m_cLedConnect.SetIcon(hGreenLedOff);
        m_LCDTempdata.SetText(_T("   Data   "));
        m_ParaGraph.ClearGraph();
        m_spiGraph.ClearGraph();
        m_I2Cgraph.ClearGraph();
        ParaDataList.RemoveAll();
        SpiDataList.RemoveAll();
        I2CDataList.RemoveAll();
        ZeroMemory(ONEWIREData,sizeof(float)*10);
        iStatusLevelPara = -1;
        iStatusLevelSPI = -1;
        iStatusLevelI2C = -1;
}


void CDAQView::SendMsgToList(CString str)
{
        CTime t1;
        t1 = CTime::GetCurrentTime();
        CString strTosend = t1.Format("%H:%M:%S ");
        strTosend += str;
        m_list_msg.InsertString(m_list_msg.GetCount(),strTosend);
        m_list_msg.SetTopIndex(m_list_msg.GetCount()-1);
}
void CDAQView::OnBnClickedClearListCtrl()
{
        m_list_msg.ResetContent();
}
void CDAQView::OnClose()
{

        CFormView::OnClose();
}
void CDAQView::OnDestroy()
{
        CFormView::OnDestroy();

        DisConnect();
        if (LogDlg)
        {
                delete LogDlg;
        }
        if (OptionDlg)
        {
                delete OptionDlg;
        }
}
void CDAQView::OnBnClickedParastop()
{
        UpdateData();
}
void CDAQView::OnBnClickedSpistop()
{
        UpdateData();
}
void CDAQView::OnBnClickedI2cstop()
{
        UpdateData();
}
void CDAQView::OnBnClicked1wirestop()
{
        UpdateData();
}
void CDAQView::OnBnClickedAllstop()
{
        UpdateData();
}

void CDAQView::InitGraph(CNTGraph &m_graph)
{
        m_graph.ClearGraph();
        m_graph.SetElementLineColor(RGB(0,255,0));
        m_graph.SetGridColor(RGB(0,150,0));
        m_graph.SetPlotAreaColor(RGB(0,100,0));
        m_graph.SetXGridNumber(10);
```

```
        m_graph.SetYGridNumber(8);
        m_graph.SetXTime(TRUE);
        m_graph.SetYTime(TRUE);

}
void CDAQView::OnLogSettingOk(void)
{
        if (LogDlg->m_iParallelTime>0)
        {
                KillTimer(TimerParallel);
                TimerParallel = SetTimer(20,1000*LogDlg->m_iParallelTime,NULL);
        }else
        {
                KillTimer(TimerParallel);
        }

        if (LogDlg->m_iSpiTime>0)
        {
                KillTimer(TimerSpi);
                TimerSpi = SetTimer(21,1000*LogDlg->m_iSpiTime,NULL);
        }else
        {
                KillTimer(TimerSpi);
        }

        if (LogDlg->m_iI2cTime>0)
        {
                KillTimer(TimerI2c);
                TimerI2c = SetTimer(22,1000*LogDlg->m_iI2cTime,NULL);
        }else
        {
                KillTimer(TimerI2c);
        }

        if (LogDlg->m_iOneWireTime>0)
        {
                KillTimer(TimerOneWire);
                TimerOneWire = SetTimer(23,1000*LogDlg->m_iOneWireTime,NULL);
        }else
        {
                KillTimer(TimerOneWire);
        }
}
void CDAQView::OnTimer(UINT_PTR nIDEvent)
{
        switch (nIDEvent)
        {
                case 10 :
                {
                        if (!m_bParaStop && !m_bAllStop)
                        {
                                if (!m_bOffLine)
                                {
                                        ParaPlot();
                                }
                                else
                                {
                                        ParaPlotOffline();
                                }
                        }
                        if (!m_bSPIstop && !m_bAllStop)
                        {
                                SPIPlot();
                        }
                        if (!m_bi2cStop && !m_bAllStop)
                        {
                                I2CPlot();
                        }
                        if (!m_b1wireStop && !m_bAllStop)
                        {
                                ONEWIREPlot();
                        }
                        static int iCountTime=0;
                        iCountTime++;
                        if (iCountTime>=10)
                        {
                                CString strBitRate;
                                strBitRate.Format("%d",(int)iCountReceived/1000);
                                m_staBitrateMain.SetWindowText(strBitRate);
```

```
                        iCountTime = 0;
                        iCountReceived = 0;
                }
                break;
        }
        case 20 :
        {
                if (bConnect && bContinue)
                {
                        DWORD dwCode;
                        if(pWriteParaThread)
                        {
                                GetExitCodeThread(pWriteParaThread-
>m_hThread,&dwCode);
                                if (dwCode==STILL_ACTIVE)
                                {
                                        SendMsgToList("Parallel Log Error, The
last save still running!!");
                                        return ;
                                }
                        }
                        pWriteParaThread =
AfxBeginThread(WriteDataParallel,this);
                }
                break;
        }
        case 21 :
        {
                if (bConneet && bContinue)
                {
                        DWORD dwCode;
                        if(pWriteSpiThread)
                        {
                                GetExitCodeThread(pWriteSpiThread-
>m_hThread,&dwCode);
                                if (dwCode==STILL_ACTIVE)
                                {
                                        SendMsgToList("SPI Log Error, The last
save still running!!");
                                        return ;
                                }
                        }
                        pWriteSpiThread = AfxBeginThread(WriteDataSpi,this);
                }
                break;
        }
        case 22 :
        {
                if (bConnect && bContinue)
                {
                        DWORD dwCode;
                        if(pWriteI2cThread)
                        {
                                GetExitCodeThread(pWriteI2cThread-
>m_hThread,&dwCode);
                                if (dwCode==STILL_ACTIVE)
                                {
                                        SendMsgToList("I2C Log Error,
The last save still running!!");
                                        return ;
                                }
                        }
                        pWriteI2cThread =
AfxBeginThread(WriteDataI2c,this);
                }
                break;
        }
        case 23 :
        {
                if (bConnect && bContinue)
                {
                        DWORD dwCode;
                        if(pWriteOneWireThread)
                        {
                                GetExitCodeThread(pWriteOneWireThread-
>m_hThread,&dwCode);
                                if (dwCode==STILL_ACTIVE)
```

```
                                                        SendMsgToList("One-Wire Log
Error, The last save still running!!");
                                                        return ;
                                                }
                                        }
                                        pWriteOneWireThread =
AfxBeginThread(WriteDataOneWire,this);
                                }
                                break;
                        }
                case 30 :
                {
                        static BOOL bInterval=FALSE;
                        bInterval = !bInterval;
                        bInterval ? m_cLedSearch.SetIcon(hRedLedOn) :
m_cLedSearch.SetIcon(hRedLedOff);
                }
        }
        CFormView::OnTimer(nIDEvent);
}

int CDAQView::FindTrigListPara(POSITION PosStart)
{
        double data[10000];
        POSITION posBuff[10000],pos1,pos2,pos3,pos4;
        int data1,data2,data3, data4;
        int fMinimum=5000;
        if (ParaDataList.IsEmpty())
        {
                return -1;
        }
        POSITION pos = PosStart;
        for (int k=0;k<1000;k++)
        {
                int fBuff = static_cast<int>(1000*ParaDataList.GetNext(pos));
                if (fBuff<fMinimum)
                {
                        fMinimum = fBuff;
                }
        }
        int iTrigOffset = 1;
        iTrigOffset = m_iedit_trig_para;
        for (int i=0;i < 10000;i++)
        {
                posBuff[i] = PosStart;
                data[i] = ParaDataList.GetNext(PosStart);
                if (i>3)
                {
                        data1 = static_cast<int>(1000*data[i-3]);pos1=posBuff[i-3];
                        data2 = static_cast<int>(1000*data[i-2]);pos2=posBuff[i-2];
                        data3 = static_cast<int>(1000*data[i-1]);pos3=posBuff[i-1];
                        data4 = static_cast<int>(1000*data[i]);pos4=posBuff[i];
                        if (data1==fMinimum && data4-data1>iTrigOffset)
                        {
                                return i;
                        }
                }
        }
        return -2;
}
int CDAQView::FindTrigListSPI(POSITION PosStart)
{
        double data[10000];
        POSITION posBuff[10000],pos1,pos2,pos3,pos4;
        int data1,data2,data3,data4,fMinimum=5000;
        if (SpiDataList.IsEmpty())
        {
                return -1;
        }
        POSITION pos = PosStart;
        for (int k=0;k<1000;k++)
        {
                int fBuff = static_cast<int>(1000*SpiDataList.GetNext(pos));
                if (fBuff<fMinimum)
                {
                        fMinimum = fBuff;
                }
        }
        int iTrigOffset = 1;
```

```
	iTrigOffset = m_iedit_trig_spi;
	for (int i=0;i < 10000;i++)
	{
		posBuff[i] = PosStart;
		data[i] = SpiDataList.GetNext(PosStart);
		if (i>3)
		{
			data1 = static_cast<int>(1000*data[i-3]);pos1=posBuff[i-3];
			data2 = static_cast<int>(1000*data[i-2]);pos2=posBuff[i-2];
			data3 = static_cast<int>(1000*data[i-1]);pos3=posBuff[i-1];
			data4 = static_cast<int>(1000*data[i]);pos4=posBuff[i];
			if (data1==fMinimum && data4-data1>iTrigOffset)
			{
				return i;
			}
		}
	}
	return -2;
}
int CDAQView::FindTrigListI2C(POSITION PosStart)
{
	double data[2000];
	POSITION posBuff[2000],pos1,pos2,pos3,pos4;
	int data1,data2,data3, data4,fMinimum=5000;
	int iReturn=0;
	if (I2CDataList.IsEmpty())
	{
		return iReturn;
	}
	POSITION pos = PosStart;
	for (int k=0;k<100;k++)
	{
		if (k<I2CDataList.GetCount())
		{
			int fBuff = static_cast<int>(1000*I2CDataList.GetNext(pos));
			if (fBuff<fMinimum)
			{
				fMinimum = fBuff;
			}
		}
	}
	int iTrigOffset = 1;
	iTrigOffset = m_iedit_trig_i2c;
	for (int i=0;i < 2000;i++)
	{
		if (i>=I2CDataList.GetCount())
		{
			return iReturn;
		}
		posBuff[i] = PosStart;
		data[i] = I2CDataList.GetNext(PosStart);
		if (i>3)
		{
			data1 = static_cast<int>(1000*data[i-3]);pos1=posBuff[i-3];
			data2 = static_cast<int>(1000*data[i-2]);pos2=posBuff[i-2];
			data3 = static_cast<int>(1000*data[i-1]);pos3=posBuff[i-1];
			data4 = static_cast<int>(1000*data[i]);pos4=posBuff[i];
			if (data1==fMinimum && data4-data1>iTrigOffset)
			{
				return i;
			}
		}
	}
	return iReturn;
}

void CDAQView::ParaPlot(void)
{
	static int iCountPlot=0;
	iCountPlot++;
	SettingParaGraph(ParaXlower,ParaXupper,ParaYlower,ParaYupper);
	m_ParaGraph.SetRange(ParaXlower,ParaXupper,ParaYlower,ParaYupper);
	m_ParaGraph.ClearGraph();
	m_ParaGraph.SetElementLineColor(RGB(0,255,0));
	m_ParaGraph.SetRedraw(FALSE);
	ParaPlotDataList.RemoveAll();
	CSingleLock singleLock(&m_csQueuePara);
	singleLock.Lock();
	static int iCountTime=0;
```

```
        iCountTime++;
        if (iCountTime>=10)
        {
                CString strBitRate;
                strBitRate.Format("%d",(int)iCountParaSample/1000);
                m_staBitRatePara.SetWindowText(strBitRate);
                iCountTime = 0;
                iCountParaSample = 0;
        }
        UpdateData();
        POSITION pos = ParaDataList.GetHeadPosition();
        POSITION PosStart = pos;
        int iOffset=0;
        UpdateData(TRUE);
        if (m_bChkTrigPara)
        {
                iOffset = FindTrigListPara(pos);
                if (iOffset==-1)
                {
                        SendMsgToList("Error from FindTrigListPara function");
                }else if(iOffset==-2)
                {
                        SendMsgToList("Can't find start trig point--Error from
FindTrigListPara function");
                }
                for (int k=0;k<(iOffset-m_iEditOffsetPara-3);k++)
                {
                        ParaDataList.GetNext(pos);
                }
        }
        double fdata;
        for (int i=0;i < 10000;i++)
        {
                if (!ParaDataList.IsEmpty())
                {
                        fdata = ParaDataList.GetNext(pos);
                }else
                {
                        return;
                }
                if (fdata==0)
                {
                        fdata = fdata * 0.01;
                }
                if (fdata>-4 && fdata<4)
                {
                        if (i < ParaXupper)
                        {
                                m_ParaGraph.PlotXY(i,fdata,0);
                                ParaPlotDataList.AddTail((float)fdata);
                        }else
                        {
                                break;
                        }
                }
        }
        singleLock.Unlock();
        m_ParaGraph.SetRedraw(TRUE);
        m_ParaGraph.Invalidate();
        m_ParaGraph.UpdateWindow();
        m_fvolt_para = CalculateVoltPara();
        m_fFreq_para = CalculateFreqPara();
        UpdateData(FALSE);
        CheckLevelPara();
}
float CDAQView::CalculateVoltPara(void)
{
        float fMin=5.0,fMax=0.0,fDataBuff,fVoltMax;
        POSITION pos = ParaPlotDataList.GetHeadPosition();
        for (int i=0;i<ParaPlotDataList.GetCount();i++)
        {
                fDataBuff = ParaPlotDataList.GetNext(pos);
                if (fDataBuff>fMax)
                {
                        fMax = fDataBuff;
                }
                if (fDataBuff<fMin)
                {
                        fMin = fDataBuff;
                }
```

```
        }
        fVoltMax = fMax-fMin;
        if (fVoltMax>=5.0)
        {
                return -1;
        }
        return fVoltMax;
}
float CDAQView::CalculateFreqPara()
{
        float fMinimum=5.0;
        int data[10000],data1,data2,data3,data4;
        POSITION posBuff[10000],pos1,pos2,pos3,pos4;
        POSITION pos = ParaPlotDataList.GetHeadPosition();
        for (int k=0;k<ParaPlotDataList.GetCount();k++)
        {
                int fBuff = static_cast<int>(1000*ParaDataList.GetNext(pos));
                if (fBuff<fMinimum)
                {
                        fMinimum = (float)fBuff;
                }
        }
        int iTrigOffset = 1;
        iTrigOffset = m_iedit_trig_para;
        float flGridPoint = (float)ParaXupper/m_ParaGraph.GetXGridNumber();
        int iCount;
        pos = ParaPlotDataList.GetHeadPosition();
        for (iCount=0;iCount < ParaPlotDataList.GetCount();iCount++)
        {
                posBuff[iCount] = pos;
                data[iCount] = static_cast<int>(ParaPlotDataList.GetNext(pos));
                if (iCount>10)
                {
                        data1 = 1000*data[iCount-3];pos1=posBuff[iCount-3];
                        data2 = 1000*data[iCount-2];pos2=posBuff[iCount-2];
                        data3 = 1000*data[iCount-1];pos3=posBuff[iCount-1];
                        data4 = 1000*data[iCount];pos4=posBuff[iCount];
                        if (data1==fMinimum && data4-data1>iTrigOffset)
                        {
                                iCount -= 6;
                                break;
                        }
                }
        }
        double flGridTime=0.0;
        switch(m_para_timediv.GetCurSel())
        {
        case 0 :
                {
                        flGridTime = 125e-6;
                        break;
                }
        case 1 :
                {
                        flGridTime = 250e-6;
                        break;
                }
        case 2 :
                {
                        flGridTime = 500e-6;
                        break;
                }
        case 3 :                {
                        flGridTime = 1e-3;
                        break;
                }
        case 4 :
                {
                        flGridTime = 2e-3;
                        break;
                }
        default:
                {
                        flGridTime = 1e-3;
                        break;
                }
        }
        float fTime = (float)(flGridTime * iCount)/flGridPoint;        float fFreq =
1/fTime;
```

```
        return fFreq;
void CDAQView::SPIPlot(void)
{
        SettingSPIGraph(SpiXlower,SpiXupper,SpiYlower,SpiYupper);
        m_spiGraph.SetRange(SpiXlower,SpiXupper,SpiYlower,SpiYupper);
        m_spiGraph.ClearGraph();
        m_spiGraph.SetElementLineColor(RGB(0,255,0));
        m_spiGraph.SetRedraw(FALSE);
        SpiPlotDataList.RemoveAll();
        CSingleLock singleLock(&m_csQueueSPI);
        singleLock.Lock();
        static int iCountTime=0;
        static int iCountAverage=0;
        iCountTime++;
        if (iCountTime>=10)
        {
                CString strBitRate;
                strBitRate.Format("%d",(int)iCountSpiSample/1000);
                m_staBitRateSpi.SetWindowText(strBitRate);
                iCountTime = 0;
                iCountSpiSample = 0;
        }
        POSITION pos = SpiDataList.GetHeadPosition();
        POSITION PosStart = pos;
        int iOffset=0;
        UpdateData(TRUE);
        if (m_bChkTrigSPI)
        {
                iOffset = FindTrigListSPI(pos);
                if (iOffset==-1)
                {
                        SendMsgToList("Error from FindTrigListSPI function");
                }else if(iOffset==-2)
                {
                        SendMsgToList("Can't find start trig point--Error from
FindTrigListSPI function");
                }
                for (int k=0;k<(iOffset-m_iEditOffSetSPI-1);k++)
                {
                        SpiDataList.GetNext(pos);
                }
        }
        for (int i=0;i < 10000;i++)
        {
                double fdata;
                if (!SpiDataList.IsEmpty())
                {
                        fdata = SpiDataList.GetNext(pos);
                }else
                {
                        return;
                }
                if (fdata==0)
                {
                        fdata = fdata * 0.01;
                }
                if (fdata>-4 && fdata<4)
                {
                        if (i < SpiXupper)
                        {
                                m_spiGraph.PlotXY(i,fdata,0);
                                SpiPlotDataList.AddTail((float)fdata);
                        }else
                        {
                                break;
                        }
                }
        }
        singleLock.Unlock();
        m_spiGraph.SetRedraw(TRUE);
        m_spiGraph.Invalidate();
        m_spiGraph.UpdateWindow();
        m_fvolt_spi = CalculateVoltSPI();
        m_fFreqspi = CalculateFreqSPI();
        UpdateData(FALSE);
        CheckLevelSPI();
}
float CDAQView::CalculateVoltSPI(void)
{
```

```
	float fMin=5.0,fMax=0.0,fDataBuff,fVoltMax;
	POSITION pos = SpiPlotDataList.GetHeadPosition();
	for (int i=0;i<SpiPlotDataList.GetCount();i++)
	{
		fDataBuff = SpiPlotDataList.GetNext(pos);
		if (fDataBuff>fMax)
		{
			fMax = fDataBuff;
		}
		if (fDataBuff<fMin)
		{
			fMin = fDataBuff;
		}
	}
	fVoltMax = fMax-fMin;
	if (fVoltMax>=5.0)
	{
		return -1;
	}
	return fVoltMax;
}
float CDAQView::CalculateFreqSPI()
{
	float fMinimum=5.0;
	int data[10000],data1,data2,data3,data4;
	POSITION posBuff[10000],pos1,pos2,pos3,pos4;
	POSITION pos = SpiPlotDataList.GetHeadPosition();
	for (int k=0;k<SpiPlotDataList.GetCount();k++)
	{
		int fBuff = static_cast<int>(1000*SpiPlotDataList.GetNext(pos));
		if (fBuff<fMinimum)
		{
			fMinimum = (float)fBuff;
		}
	}
	int iTrigOffset = 1;
	iTrigOffset = m_iedit_trig_spi;
	float flGridPoint = (float)SpiXupper/m_spiGraph.GetXGridNumber();
	int iCount;
	pos = SpiPlotDataList.GetHeadPosition();
	for (iCount=0;iCount < SpiPlotDataList.GetCount();iCount++)
	{
		posBuff[iCount] = pos;
		data[iCount] = static_cast<int>(SpiPlotDataList.GetNext(pos));
		if (iCount>10)
		{
			data1 = 1000*data[iCount-3];pos1=posBuff[iCount-3];
			data2 = 1000*data[iCount-2];pos2=posBuff[iCount-2];
			data3 = 1000*data[iCount-1];pos3=posBuff[iCount-1];
			data4 = 1000*data[iCount];pos4=posBuff[iCount];
			if (data1==fMinimum && data4-data1>iTrigOffset)
			{
				iCount -= 4;
				break;
			}
		}
	}
	double flGridTime=0.0;
	switch(m_spi_timediv.GetCurSel())
	{
	case 0 :
		{
			flGridTime = 125e-6;
			break;
		}
	case 1 :
		{
			flGridTime = 250e-6;
			break;
		}
	case 2 :
		{
			flGridTime = 500e-6;
			break;
		}
	case 3 :
		{
			flGridTime = 1e-3;
			break;
```

```
		}
	case 4 :
		{
			flGridTime = 2e-3;
			break;
		}
	default:
		{
			flGridTime = 1e-3;
			break;
		}
	}
	float fTime = (float)(flGridTime * iCount)/flGridPoint;
	float fFreq = 1/fTime;
	return fFreq;
}
void CDAQView::I2CPlot(void)
{
	SettingI2CGraph(I2CXlower,I2CXupper,I2CYlower,I2CYupper);
	m_I2Cgraph.SetRange(I2CXlower,I2CXupper,I2CYlower,I2CYupper);
	m_I2Cgraph.ClearGraph();
	m_I2Cgraph.SetElementLineColor(RGB(0,255,0));
	m_I2Cgraph.SetRedraw(FALSE);
	I2CPlotDataList.RemoveAll();
	CSingleLock singleLock(&m_csQueueI2C);
	singleLock.Lock();
	static int iCountTime=0;
	static int iCountAverage=0;
	iCountTime++;
	if (iCountTime>=10)
	{
		CString strBitRate;
		strBitRate.Format("%d",(int)iCountI2cSample/1000);
		m_staBitRateI2c.SetWindowText(strBitRate);
		iCountTime = 0;
		iCountI2cSample = 0;
	}
	POSITION pos = I2CDataList.GetHeadPosition();
	POSITION PosStart = pos;
	int iOffset=0;
	UpdateData(TRUE);
	if (m_bChkTrigI2c)
	{
		int k=0;
		iOffset = FindTrigListI2C(pos);
		for (k=0;k<(iOffset-m_iEditOffsetI2c-1);k++)
		{
			I2CDataList.GetNext(pos);
		}

	}
	for (int i=0;i < 2000;i++)
	{
		double fdata;
		if (!I2CDataList.IsEmpty())
		{
			fdata = I2CDataList.GetNext(pos);
		}else
		{
			return;
		}
		if (fdata==0)
		{
			fdata = fdata * 0.01;
		}
		if (fdata>-4 && fdata<4)
		{
			if (i < I2CXupper)
			{
				m_I2Cgraph.PlotXY(i,fdata,0);
				I2CPlotDataList.AddTail((float)fdata);
			}else
			{
				break;
			}
		}
	}
	singleLock.Unlock();
	m_I2Cgraph.SetRedraw(TRUE);
```

```
        m_I2Cgraph.Invalidate();
        m_I2Cgraph.UpdateWindow();
        m_fvolt_i2c = CalculateVoltI2C();
        m_fFreqi2c = CalculateFreqI2C();
        UpdateData(FALSE);
        CheckLevelI2C();
}
float CDAQView::CalculateVoltI2C(void)
{
        float fMin=5.0,fMax=0.0,fDataBuff,fVoltMax;
        POSITION pos = I2CPlotDataList.GetHeadPosition();
        for (int i=0;i<I2CPlotDataList.GetCount();i++)
        {
                fDataBuff = I2CPlotDataList.GetNext(pos);
                if (fDataBuff>fMax)
                {
                        fMax = fDataBuff;
                }
                if (fDataBuff<fMin)
                {
                        fMin = fDataBuff;
                }
        }
        fVoltMax = fMax-fMin;
        if (fVoltMax>=5.0)
        {
                return -1;
        }
        return fVoltMax;
}
float CDAQView::CalculateFreqI2C()
{
        float fMinimum=5.0;
        int data[10000],data1,data2,data3,data4;
        POSITION posBuff[10000],pos1,pos2,pos3,pos4;
        POSITION pos = I2CPlotDataList.GetHeadPosition();
        for (int k=0;k<I2CPlotDataList.GetCount();k++)
        {
                int fBuff = static_cast<int>(1000*I2CDataList.GetNext(pos));
                if (fBuff<fMinimum)
                {
                        fMinimum = (float)fBuff;
                }
        }
        int iTrigOffset = 1;
        iTrigOffset = m_iedit_trig_i2c;
        float f1GridPoint = (float)I2CXupper/m_I2Cgraph.GetXGridNumber();
        int iCount;
        pos = I2CPlotDataList.GetHeadPosition();
        for (iCount=0;iCount < I2CPlotDataList.GetCount();iCount++)
        {
                posBuff[iCount] = pos;
                data[iCount] = static_cast<int>(I2CPlotDataList.GetNext(pos));
                if (iCount>10)
                {
                        data1 = 1000*data[iCount-3];pos1=posBuff[iCount-3];
                        data2 = 1000*data[iCount-2];pos2=posBuff[iCount-2];
                        data3 = 1000*data[iCount-1];pos3=posBuff[iCount-1];
                        data4 = 1000*data[iCount];pos4=posBuff[iCount];
                        if (data1==fMinimum && data4-data1>iTrigOffset)
                        {
                                iCount -= 4;
                                break;
                        }
                }
        }
        double f1GridTime=0.0;
        switch(m_i2c_timediv.GetCurSel())
        {
        case 0 :
                {
                        f1GridTime = 250e-6;
                        break;
                }
        case 1 :
                {
                        f1GridTime = 500e-6;
                        break;
                }
```

```
        case 2 :
                {
                        f1GridTime = 1e-3;
                        break;
                }
        case 3 :
                {
                        f1GridTime = 2e-3;
                        break;
                }
        case 4 :
                {
                        f1GridTime = 4e-3;
                        break;
                }
        default:
                {
                        f1GridTime = 1e-3;
                        break;
                }
        }
        float fTime = (float)(f1GridTime * iCount)/f1GridPoint;
        float fFreq = 1/fTime;
        return fFreq;
}
void CDAQView::ONEWIREPlot(void)
{
        CSingleLock singleLock(&m_csQueueONEWIRE);
        singleLock.Lock();
        double fdata = 0;
        for (int i=0;i<10;i++)
        {
                fdata += ONEWIREData[i];
        }
        fdata /= 10;
        CString str;
        str.Format(" %.3f'C ",fdata);
        static int iCountTime=0;
        static int iCountAverage=0;
        iCountTime++;
        if (iCountTime>=10)
        {
                CString strBitRate;
                strBitRate.Format("%d",iCount1WireSample);
                m_staBitRate1Wire.SetWindowText(strBitRate);
                iCountTime = 0;
                iCount1WireSample = 0;
        }
        CheckLevel1WIRE((float)fdata);
        m_LCDTempdata.SetText(str);
        singleLock.Unlock();
}
void CDAQView::SettingParaGraph(double &Xlower, double &Xupper, double &Ylower,
double &Yupper)
{
        switch(m_para_timediv.GetCurSel())
        {
        case 0 :
                {
                        Xlower = 0;Xupper = i1msParaVal*0.5*0.5*0.5;
                        break;
                }
        case 1 :
                {
                        Xlower = 0;Xupper = i1msParaVal*0.5*0.5;
                        break;
                }
        case 2 :
                {
                        Xlower = 0;Xupper = i1msParaVal*0.5;
                        break;
                }
        case 3 :
                {
                        Xlower = 0;Xupper = i1msParaVal;
                        break;
                }
        case 4 :
                {
```

```
                        Xlower = 0;Xupper = ilmsParaVal*2;
                        break;
                }
        default:
                {
                        Xlower = 0;Xupper = ilmsParaVal*0.5;
                        break;
                }
        }
        switch(m_para_voltdiv.GetCurSel())
        {
        case 0 :
                {
                        Ylower = -1;Yupper = 1;
                        break;
                }
        case 1 :
                {
                        Ylower = -2;Yupper = 2;
                        break;
                }
        case 2 :
                {
                        Ylower = -4;Yupper = 4;
                        break;
                }
        case 3 :
                {
                        Ylower = -8;Yupper = 8;
                        break;
                }
        case 4 :
                {
                        Ylower = -16;Yupper = 16;
                        break;
                }
        default:
                {
                        Ylower = -4;Yupper = 4;
                        break;
                }
        }
}

void CDAQView::SettingSPIGraph(double &Xlower, double &Xupper, double &Ylower, double
&Yupper)
{
        switch(m_spi_timediv.GetCurSel())
        {
        case 0 :
                {
                        Xlower = 0;Xupper = ilmsSpiVal*0.5*0.5*0.5;
                        break;
                }
        case 1 :
                {
                        Xlower = 0;Xupper = ilmsSpiVal*0.5*0.5;
                        break;
                }
        case 2 :
                {
                        Xlower = 0;Xupper = ilmsSpiVal*0.5;
                        break;
                }
        case 3 :
                {
                        Xlower = 0;Xupper = ilmsSpiVal;
                        break;
                }
        case 4 :
                {
                        Xlower = 0;Xupper = ilmsSpiVal*2;
                        break;
                }
        default:
                {
                        Xlower = 0;Xupper = ilmsSpiVal*0.5;
                        break;
                }
```

```
	}
	switch(m_spi_voltdiv.GetCurSel())
	{
	case 0 :
		{
			Ylower = -1;Yupper = 1;
			break;
		}
	case 1 :
		{
			Ylower = -2;Yupper = 2;
			break;
		}
	case 2 :
		{
			Ylower = -4;Yupper = 4;
			break;
		}
	case 3 :
		{
			Ylower = -8;Yupper = 8;
			break;
		}
	case 4 :
		{
			Ylower = -16;Yupper = 16;
			break;
		}
	default:
		{
			Ylower = -4;Yupper = 4;
			break;
		}
	}
}
void CDAQView::SettingI2CGraph(double &Xlower, double &Xupper, double &Ylower, double
&Yupper)
{
	switch(m_i2c_timediv.GetCurSel())
	{
	case 0 :
		{
			Xlower = 0;Xupper = i1msI2cVal*0.5*0.5*0.5;
			break;
		}
	case 1 :
		{
			Xlower = 0;Xupper = i1msI2cVal*0.5*0.5;
			break;
		}
	case 2 :
		{
			Xlower = 0;Xupper = i1msI2cVal*0.5;
			break;
		}
	case 3 :
		{
			Xlower = 0;Xupper = i1msI2cVal;
			break;
		}
	case 4 :
		{
			Xlower = 0;Xupper = i1msI2cVal*2;
			break;
		}
	case 5 :
		{
			Xlower = 0;Xupper = 400;
			break;
		}
	default:
		{
			Xlower = 0;Xupper = i1msI2cVal*0.5;
			break;
		}
	}
	switch(m_i2c_voltdiv.GetCurSel())
	{
	case 0 :
```

```
		{
			Ylower = -1;Yupper = 1;
			break;
		}
	case 1 :
		{
			Ylower = -2;Yupper = 2;
			break;
		}
	case 2 :
		{
			Ylower = -4;Yupper = 4;
			break;
		}
	case 3 :
		{
			Ylower = -8;Yupper = 8;
			break;
		}
	case 4 :
		{
			Ylower = -16;Yupper = 16;
			break;
		}
	default:
		{
			Ylower = -4;Yupper = 4;
			break;
		}
	}
}

void CDAQView::OnBnClickedSend()
{
	int chSend;
	UpdateData(true);
	if (m_strEditSend!="")
	{
		chSend = atoi(m_strEditSend);
		if (atoi(m_strEditSend)>255)
		{
			CString str = "You need to specific data between \"0-255\" for
get the correct data result";
			MessageBox(str,"Value not correct",MB_ICONINFORMATION|MB_OK);
			SendMsgToList(str);
		}
		if (chSend!=0)
		{
			if(SendData(chSend))
			{
				UpdateLEDData(chSend);
			}
		}
		else if (m_strEditSend[0]=='0' && (m_strEditSend[1]=='x' ||
m_strEditSend[1]=='X'))
		{
			sscanf_s(m_strEditSend, "%04x", &chSend);
			if(SendData(chSend))
			{
				UpdateLEDData(chSend);
			}
		}
	}
}
void CDAQView::UpdateLEDData(unsigned char chData)
{
	chData&0x01 ? m_D0.SetIcon(hRedLedOn) : m_D0.SetIcon(hRedLedOff);
	chData&0x02 ? m_D1.SetIcon(hRedLedOn) : m_D1.SetIcon(hRedLedOff);
	chData&0x04 ? m_D2.SetIcon(hRedLedOn) : m_D2.SetIcon(hRedLedOff);
	chData&0x08 ? m_D3.SetIcon(hRedLedOn) : m_D3.SetIcon(hRedLedOff);
	chData&0x10 ? m_D4.SetIcon(hRedLedOn) : m_D4.SetIcon(hRedLedOff);
	chData&0x20 ? m_D5.SetIcon(hRedLedOn) : m_D5.SetIcon(hRedLedOff);
	chData&0x40 ? m_D6.SetIcon(hRedLedOn) : m_D6.SetIcon(hRedLedOff);
	chData&0x80 ? m_D7.SetIcon(hRedLedOn) : m_D7.SetIcon(hRedLedOff);
}
BOOL CDAQView::SendData(unsigned char  chSend)
{
	FT_STATUS ftStatus;
```

```
        DWORD dwRetByte;
        FT_ResetDevice(ftHandle);
        FT_Purge(ftHandle,FT_PURGE_TX);
        ftStatus = FT_Write(ftHandle,&chSend,sizeof(chSend),&dwRetByte);
        CString strSend;
        if(dwRetByte==1 && ftStatus==FT_OK)
        {
                strSend.Format("Send %d Complete",chSend);
                SendMsgToList(strSend);
        }else
        {
                strSend = "Send Data Error";
                SendMsgToList(strSend);
                return FALSE;
        }
        FT_SetEventNotification(ftHandle, FT_EVENT_RXCHAR, hEvent);
        return TRUE;
}
BOOL CDAQView::PreTranslateMessage(MSG* pMsg)
{
        if (pMsg->message==WM_KEYDOWN)
        {
                if (pMsg->wParam==VK_RETURN)
                {
                        if (GetDlgItem(IDC_EDITSEND)==AfxGetMainWnd()->GetFocus())
                        {
                                OnBnClickedSend();
                        }
                }
        }
        GetDlgItem(IDC_LOAD_PARA)->EnableWindow(m_bOffLine);
        GetDlgItem(IDC_LOAD_SPI)->EnableWindow(m_bOffLine);
        GetDlgItem(IDC_LOAD_I2C)->EnableWindow(m_bOffLine);
        GetDlgItem(IDC_LOAD_ONEWIRE)->EnableWindow(m_bOffLine);
        GetDlgItem(IDC_SEND)->EnableWindow(bConnect);
        GetDlgItem(IDC_SAVE_PARALLEL)->EnableWindow(bConnect);
        GetDlgItem(IDC_SAVE_SPI)->EnableWindow(bConnect);
        GetDlgItem(IDC_SAVE_I2C)->EnableWindow(bConnect);
        GetDlgItem(IDC_SAVE_ONEWIRE)->EnableWindow(bConnect);
        m_cEditSend.EnableWindow(bConnect);
        CString str;
        m_edit_trig_para.GetWindowText(str);
        if(str=="")
        {
                m_edit_trig_para.SetWindowText("1");
        }
        m_edit_offset_para.GetWindowText(str);
        if(str=="")
        {
                m_edit_offset_para.SetWindowText("0");
        }
        m_edit_trig_spi.GetWindowText(str);
        if(str=="")
        {
                m_edit_trig_spi.SetWindowText("1");
        }
        m_editOffSetSPI.GetWindowText(str);
        if(str=="")
        {
                m_editOffSetSPI.SetWindowText("0");
        }
        m_edit_trig_i2c.GetWindowText(str);
        if(str=="")
        {
                m_edit_trig_i2c.SetWindowText("1");
        }
        m_EditOffsetI2c.GetWindowText(str);
        if(str=="")
        {
                m_EditOffsetI2c.SetWindowText("0");
        }
        return CFormView::PreTranslateMessage(pMsg);
}

void CDAQView::OnBnClickedLogsetting()
{
        LogDlg->ShowWindow(SW_SHOW);
}
void CDAQView::OnBnClickedOption()
```

```
{
        COptionDlg opDlg;
        opDlg.m_bChkPara = OptionDlg.m_bChkPara;
        opDlg.m_strOuthiPara = OptionDlg.m_strOuthiPara;
        opDlg.m_strOutlowPara= OptionDlg.m_strOutlowPara;
        opDlg.m_fHithPara = OptionDlg.m_fHithPara;
        opDlg.m_fLowthPara = OptionDlg.m_fLowthPara;
        opDlg.m_bChkSPI = OptionDlg.m_bChkSPI;
        opDlg.m_strOuthiSPI = OptionDlg.m_strOuthiSPI;
        opDlg.m_strOutlowSPI= OptionDlg.m_strOutlowSPI;
        opDlg.m_fHiThSPI = OptionDlg.m_fHiThSPI;
        opDlg.m_fLowThSPI = OptionDlg.m_fLowThSPI;
        opDlg.m_bChkI2C = OptionDlg.m_bChkI2C;
        opDlg.m_strOuthiI2C = OptionDlg.m_strOuthiI2C;
        opDlg.m_strOutlowi2c= OptionDlg.m_strOutlowi2c;
        opDlg.m_fHiThI2C = OptionDlg.m_fHiThI2C;
        opDlg.m_fLowThI2C = OptionDlg.m_fLowThI2C;
        opDlg.m_bChk1Wire = OptionDlg.m_bChk1Wire;
        opDlg.m_strOutHi1Wire = OptionDlg.m_strOutHi1Wire;
        opDlg.m_strOutLow1Wire= OptionDlg.m_strOutLow1Wire;
        opDlg.m_fhiTh1Wire = OptionDlg.m_fhiTh1Wire;
        opDlg.m_fLowTh1Wire = OptionDlg.m_fLowTh1Wire;
        if (opDlg.DoModal()==IDOK)
        {
                OptionDlg.m_bChkPara = opDlg.m_bChkPara;
                OptionDlg.m_strOuthiPara = opDlg.m_strOuthiPara;
                OptionDlg.m_strOutlowPara= opDlg.m_strOutlowPara;
                OptionDlg.m_fHithPara = opDlg.m_fHithPara;
                OptionDlg.m_fLowthPara = opDlg.m_fLowthPara;
                OptionDlg.m_bChkSPI = opDlg.m_bChkSPI;
                OptionDlg.m_strOuthiSPI = opDlg.m_strOuthiSPI;
                OptionDlg.m_strOutlowSPI= opDlg.m_strOutlowSPI;
                OptionDlg.m_fHiThSPI = opDlg.m_fHiThSPI;
                OptionDlg.m_fLowThSPI = opDlg.m_fLowThSPI;
                OptionDlg.m_bChkI2C = opDlg.m_bChkI2C;
                OptionDlg.m_strOuthiI2C = opDlg.m_strOuthiI2C;
                OptionDlg.m_strOutlowi2c= opDlg.m_strOutlowi2c;
                OptionDlg.m_fHiThI2C = opDlg.m_fHiThI2C;
                OptionDlg.m_fLowThI2C = opDlg.m_fLowThI2C;
                OptionDlg.m_bChk1Wire = opDlg.m_bChk1Wire;
                OptionDlg.m_strOutHi1Wire = opDlg.m_strOutHi1Wire;
                OptionDlg.m_strOutLow1Wire= opDlg.m_strOutLow1Wire;
                OptionDlg.m_fhiTh1Wire = opDlg.m_fhiTh1Wire;
                OptionDlg.m_fLowTh1Wire = opDlg.m_fLowTh1Wire;
        }
}

UINT WriteDataSpi(void *pParam)
{
        CDAQView *pDaqView = (CDAQView*)pParam;
        CTime st = CTime::GetCurrentTime();
        CString strPath = LogDlg->m_strEditPath;
        CString strFileName = st.Format("\\DAQ_%Y-%m-%d");
        strPath += strFileName;
        CreateDirectory(strPath,NULL);
        strFileName = st.Format("\\SPI");
        strPath += strFileName;
        CreateDirectory(strPath,NULL);
        strFileName = st.Format("\\%H-%M-%S.txt");
        strPath += strFileName;
        CStdioFile file;
        if (file.Open(strPath,
                CFile::modeCreate | CFile::modeWrite ))
        {
                CSingleLock singleLock(&m_csQueueSPI);
                CString strData;
                POSITION pos = SpiDataList.GetHeadPosition();
                singleLock.Lock();
                for (int i=0;i<SpiDataList.GetCount();i++)
                {
                        strData.Format("SPI
Data[%d]=%f\n",i+1,SpiDataList.GetNext(pos));
                        file.WriteString(strData);
                }
                singleLock.Unlock();
                pDaqView->SendMsgToList("Save SPI Data Finish");
        }else
                pDaqView->SendMsgToList("Can't Open File for SPI Log!!");
```

```
	}

	return 0;
}
UINT WriteDataI2c(void *pParam)
{
	CDAQView *pDaqView = (CDAQView*)pParam;
	CTime st = CTime::GetCurrentTime();
	CString strPath = LogDlg->m_strEditPath;
	CString strFileName = st.Format("\\DAQ_%Y-%m-%d");
	strPath += strFileName;
	CreateDirectory(strPath,NULL);
	strFileName = st.Format("\\I2C");
	strPath += strFileName;
	CreateDirectory(strPath,NULL);
	strFileName = st.Format("\\%H-%M-%S.txt");
	strPath += strFileName;
	CStdioFile file;
	if (file.Open(strPath,
		CFile::modeCreate | CFile::modeWrite ))
	{
		CSingleLock singleLock(&m_csQueueI2C);
		CString strData;
		POSITION pos = I2CDataList.GetHeadPosition();
		singleLock.Lock();
		for (int i=0;i<I2CDataList.GetCount();i++)
		{
			strData.Format("I2C
Data[%d]=%f\n",i+1,I2CDataList.GetNext(pos));
			file.WriteString(strData);
		}
		singleLock.Unlock();
		pDaqView->SendMsgToList("Save I2C Data Finish");
	}else
	{
		pDaqView->SendMsgToList("Can't Open File for I2C Save!!");
	}

	return 0;
}
UINT WriteDataOneWire(void *pParam)
{
	CDAQView *pDaqView = (CDAQView*)pParam;
	CTime st = CTime::GetCurrentTime();
	CString strPath = LogDlg->m_strEditPath;
	CString strFileName = st.Format("\\DAQ_%Y-%m-%d");
	strPath += strFileName;
	CreateDirectory(strPath,NULL);
	strFileName = st.Format("\\One-Wire");
	strPath += strFileName;
	CreateDirectory(strPath,NULL);
	strFileName = st.Format("\\%H-%M-%S.txt");
	strPath += strFileName;
	CStdioFile file;
	if (file.Open(strPath,
		CFile::modeCreate | CFile::modeWrite ))
	{
		CSingleLock singleLock(&m_csQueueONEWIRE);
		CString strData;
		singleLock.Lock();
		memcpy(OneWireDataBk,ONEWIREData,sizeof(float)*(10));
		singleLock.Unlock();
		for (int i=0;i<10;i++)
		{
			strData.Format("One-Wire Data[%d]=%f\n",i,OneWireDataBk[i]);
			file.WriteString(strData);
		}
		pDaqView->SendMsgToList("Log One-Wire Data");
	}else
	{
		pDaqView->SendMsgToList("Can't Open File for One-Wire Log!!");
	}

	return 0;
}
void CDAQView::OnBnClickedChkOffline()
{
	UpdateData();
	if (m_bOffLine)
```

```
	{
		bConnect = FALSE;
		bContinue = FALSE;
		if(hEvent)
		{
			hEvent.SetEvent();
		}
		Sleep(100);
		if(ftHandle)
		{
			FT_Close(ftHandle);
		}
		SetDlgItemText(IDC_CONNECT,"Connect");
		GetDlgItem(IDC_CONNECT)->EnableWindow(FALSE);
		SendMsgToList("Disconnected From USB Port");
		SendMsgToList("Change to Off-line mode");
		GetDlgItem(IDC_SEARCH)->EnableWindow(FALSE);
		m_cLedConnect.SetIcon(hGreenLedOff);
		m_cLedSearch.SetIcon(hRedLedOff);
		TimerOffLine = SetTimer(30,500,NULL);
		KillTimer(TimerDrawData);
		TimerDrawData = SetTimer(10,TimeLoop,0);
	}else
	{
		SendMsgToList("Change to On-line mode");
		SetDlgItemText(IDC_CONNECT,"Connect");
		GetDlgItem(IDC_CONNECT)->EnableWindow(TRUE);
		GetDlgItem(IDC_SEARCH)->EnableWindow(TRUE);
		ParaDataList.RemoveAll();
		SpiDataList.RemoveAll();
		I2CDataList.RemoveAll();
		ZeroMemory(ONEWIREData,sizeof(float)*10);
		m_LCDTempdata.SetText(_T("   Data   "));
		m_ParaGraph.ClearGraph();
		m_spiGraph.ClearGraph();
		m_I2Cgraph.ClearGraph();
		KillTimer(TimerOffLine);
		KillTimer(TimerDrawData);
	}
}

void CDAQView::OnBnClickedLoadPara()
{
	CFileDialog fd(TRUE,NULL,NULL,OFN_HIDEREADONLY,"Text File|*.txt||");
	if (fd.DoModal()==IDOK)
	{
		CWaitCursor wait;
		CStdioFile file(fd.GetPathName(),CFile::modeRead);
		CString strBuffer;
		file.ReadString(strBuffer);
		if (strBuffer.Find("Parallel")==-1)
		{
			MessageBox("Parallel File format error!",NULL,MB_ICONERROR);
			SendMsgToList("Parallel File format error!!");
			return;
		}
		int i;
		float dataBuff;
		CSingleLock singlelock(&m_csQueuePara);
		singlelock.Lock();
		ParaDataList.RemoveAll();
		sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		while(!feof(file.m_pStream))
		{
			if (i>0 && i<=100000)
			{
				ParaDataList.AddTail(dataBuff);
			}
			file.ReadString(strBuffer);
			sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		}
		singlelock.Unlock();
	}
}
void CDAQView::OnBnClickedLoadSpi()
{
	CFileDialog fd(TRUE,NULL,NULL,OFN_HIDEREADONLY,"Text File|*.txt||");
	if (fd.DoModal()==IDOK)
```

```
	{
		CWaitCursor wait;
		CStdioFile file(fd.GetPathName(),CFile::modeRead);
		CString strBuffer;
		file.ReadString(strBuffer);
		if (strBuffer.Find("SPI")==-1)
		{
			MessageBox("SPI File format error!!",NULL,MB_ICONERROR);
			SendMsgToList("SPI File format error!!");
			return;
		}
		int i;
		float dataBuff;
		sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		while(!feof(file.m_pStream))
		{
			file.ReadString(strBuffer);
			sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		}
	}
}

void CDAQView::OnBnClickedLoadI2c()
{
	CFileDialog fd(TRUE,NULL,NULL,OFN_HIDEREADONLY,"Text File|*.txt||");
	if (fd.DoModal()==IDOK)
	{
		CWaitCursor wait;
		CStdioFile file(fd.GetPathName(),CFile::modeRead);
		CString strBuffer;
		file.ReadString(strBuffer);
		if (strBuffer.Find("I2C")==-1)
		{
			MessageBox("I2C File format error!!",NULL,MB_ICONERROR);
			SendMsgToList("I2c File format error!!");
			return;
		}
		int i;
		float dataBuff;
		sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		while(!feof(file.m_pStream))
		{
			file.ReadString(strBuffer);
			sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		}
		CSingleLock singlelock(&m_csQueueI2C);
	}
}

void CDAQView::OnBnClickedLoadOnewire()
{
	CFileDialog fd(TRUE,NULL,NULL,OFN_HIDEREADONLY,"Text File|*.txt||");
	if (fd.DoModal()==IDOK)
	{
		CWaitCursor wait;
		CStdioFile file(fd.GetPathName(),CFile::modeRead);
		CString strBuffer;
		file.ReadString(strBuffer);
		if (strBuffer.Find("One-Wire")==-1)
		{
			MessageBox("One-Wire File format error!!",NULL,MB_ICONERROR);
			SendMsgToList("One-Wire File format error!!");
			return;
		}
		int i;
		float dataBuff;
		sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		while(!feof(file.m_pStream))
		{
			if (i>0 && i<10)
			{
				OneWireDataBk[i]=dataBuff;
			}
			file.ReadString(strBuffer);
			sscanf_s(strBuffer,"%*s Data[%d]=%f",&i,&dataBuff);
		}
		CSingleLock singlelock(&m_csQueueONEWIRE);
		singlelock.Lock();
		memcpy(ONEWIREData,OneWireDataBk,sizeof(float)*10);
```

```
                singlelock.Unlock();
        }
}


void CDAQView::OnBnClickedSaveParallel()
{
        if (bConnect && bContinue)
        {
                DWORD dwCode;
                if(pWriteParaThread)
                {
                        GetExitCodeThread(pWriteParaThread->m_hThread,&dwCode);
                        if (dwCode==STILL_ACTIVE)
                        {
                                SendMsgToList("Parallel Save Error, The last save still
running!!");
                                return ;
                        }
                }
                SendMsgToList("Parallel are Saving...");
                pWriteParaThread = AfxBeginThread(WriteDataParallel,this);
        }
}

void CDAQView::OnBnClickedSaveSpi()
{
        if (bConnect && bContinue)
        {
                DWORD dwCode;
                if(pWriteSpiThread)
                {
                        GetExitCodeThread(pWriteSpiThread->m_hThread,&dwCode);
                        if (dwCode==STILL_ACTIVE)
                        {
                                SendMsgToList("SPI Save Error, The last save still
running!!");
                                return ;
                        }
                }
                SendMsgToList("SPI are Saving...");
                pWriteSpiThread = AfxBeginThread(WriteDataSpi,this);
        }
}

void CDAQView::OnBnClickedSaveI2c()
{
        if (bConnect && bContinue)
        {
                DWORD dwCode;
                if(pWriteI2cThread)
                {
                        GetExitCodeThread(pWriteI2cThread->m_hThread,&dwCode);
                        if (dwCode==STILL_ACTIVE)
                        {
                                SendMsgToList("I2C Save Error, The last save still
running!!");
                                return ;
                        }
                }
                SendMsgToList("I2C are Saving...");
                pWriteI2cThread = AfxBeginThread(WriteDataI2c,this);
        }
}

void CDAQView::OnBnClickedSaveOnewire()
{
        if (bConnect && bContinue)
        {
                DWORD dwCode;
                if(pWriteOneWireThread)
                {
                        GetExitCodeThread(pWriteOneWireThread->m_hThread,&dwCode);
                        if (dwCode==STILL_ACTIVE)
                        {
                                SendMsgToList("One-Wire Log Error, The last save still
running!!");
                                return ;
                        }
```

```
                }
                SendMsgToList("One-Wire are Saving...");
                pWriteOneWireThread = AfxBeginThread(WriteDataOneWire,this);
        }
}
void CDAQView::CheckLevelPara(void)
{
        float fAverage=0;
        if (ParalistLevel.GetCount()<10)
        {
                ParalistLevel.AddTail(m_fvolt_para);
        }else
        {
                ParalistLevel.RemoveHead();
                ParalistLevel.AddTail(m_fvolt_para);
        }
        POSITION pos = ParalistLevel.GetHeadPosition();
        for (int i=0;i<ParalistLevel.GetCount();i++)
        {
                fAverage += ParalistLevel.GetNext(pos);
        }
        fAverage = fAverage/ParalistLevel.GetCount();
        if (iStatusLevelPara!=1 && (fAverage > OptionDlg.m_fHithPara))
        {
                UpdateData(true);
                if (OptionDlg.m_bChkPara==TRUE && OptionDlg.m_strOuthiPara!="")
                {
                        int chSend = atoi(OptionDlg.m_strOuthiPara);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelPara = 1;
                                }
                        }
                        else if (OptionDlg.m_strOuthiPara[0]=='0' &&
(OptionDlg.m_strOuthiPara[1]=='x' || OptionDlg.m_strOuthiPara[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOuthiPara, "%04x", &chSend);
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelPara = 1;
                                }
                        }
                }
        }else if (iStatusLevelPara!=0 && (fAverage < OptionDlg.m_fLowthPara))
        {
                UpdateData(true);
                if (OptionDlg.m_bChkPara==TRUE && OptionDlg.m_strOutlowPara!="")
                {
                        int chSend = atoi(OptionDlg.m_strOutlowPara);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelPara = 0;
                                }
                        }
                        else if (OptionDlg.m_strOutlowPara[0]=='0' &&
(OptionDlg.m_strOutlowPara[1]=='x' || OptionDlg.m_strOuthiPara[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOutlowPara, "%04x", &chSend);
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelPara = 0;
                                }
                        }
                }
        }
}
void CDAQView::CheckLevelSPI(void)
{
        float fAverage=0;
        if (SpilistLevel.GetCount()<10)
        {
```

```
                SpilistLevel.AddTail(m_fvolt_spi);
        }else
        {
                SpilistLevel.RemoveHead();
                SpilistLevel.AddTail(m_fvolt_spi);
        }
        POSITION pos = SpilistLevel.GetHeadPosition();
        for (int i=0;i<SpilistLevel.GetCount();i++)
        {
                fAverage += SpilistLevel.GetNext(pos);
        }
        fAverage = fAverage/SpilistLevel.GetCount();
        if (iStatusLevelSPI!=1 && (fAverage > OptionDlg.m_fHiThSPI))
        {
                UpdateData(true);
                if (OptionDlg.m_bChkSPI==TRUE && OptionDlg.m_strOuthiSPI!="")
                {
                        int chSend = atoi(OptionDlg.m_strOuthiSPI);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelSPI = 1;
                                }
                        }
                        else if (OptionDlg.m_strOuthiSPI[0]=='0' &&
(OptionDlg.m_strOuthiSPI[1]=='x' || OptionDlg.m_strOuthiSPI[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOuthiSPI, "%04x", &chSend);

                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelSPI = 1;
                                }
                        }
                }
        }else if (iStatusLevelSPI!=0 && (fAverage < OptionDlg.m_fLowThSPI))
        {
                UpdateData(true);
                if (OptionDlg.m_bChkSPI==TRUE && OptionDlg.m_strOutlowSPI!="")
                {
                        int chSend = atoi(OptionDlg.m_strOutlowSPI);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelSPI = 0;
                                }
                        }
                        else if (OptionDlg.m_strOutlowSPI[0]=='0' &&
(OptionDlg.m_strOutlowSPI[1]=='x' || OptionDlg.m_strOuthiSPI[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOutlowSPI, "%04x", &chSend);

                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelSPI = 0;
                                }
                        }
                }
        }
}
void CDAQView::CheckLevelI2C(void)
{
        float fAverage=0;
        if (I2clistLevel.GetCount()<10)
        {
                I2clistLevel.AddTail(m_fvolt_i2c);
        }else
        {
                I2clistLevel.RemoveHead();
                I2clistLevel.AddTail(m_fvolt_i2c);
        }
        POSITION pos = I2clistLevel.GetHeadPosition();
        for (int i=0;i<I2clistLevel.GetCount();i++)
```

```
        {
                fAverage += I2clistLevel.GetNext(pos);
        }
        fAverage = fAverage/I2clistLevel.GetCount();
        if (iStatusLevelI2C!=1 && (fAverage > OptionDlg.m_fHiThI2C))
        {
                UpdateData(true);
                if (OptionDlg.m_bChkI2C==TRUE && OptionDlg.m_strOuthiI2C!="")
                {
                        int chSend = atoi(OptionDlg.m_strOuthiI2C);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelI2C = 1;
                                }
                        }
                        else if (OptionDlg.m_strOuthiI2C[0]=='0' &&
(OptionDlg.m_strOuthiI2C[1]=='x' || OptionDlg.m_strOuthiI2C[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOuthiI2C, "%04x", &chSend);
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelI2C = 1;
                                }
                        }
                }
        }else if (iStatusLevelI2C!=0 && (fAverage < OptionDlg.m_fLowThI2C))
        {
                UpdateData(true);
                if (OptionDlg.m_bChkI2C==TRUE && OptionDlg.m_strOutlowi2c!="")
                {
                        int chSend = atoi(OptionDlg.m_strOutlowi2c);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelI2C = 0;
                                }
                        }
                        else if (OptionDlg.m_strOutlowi2c[0]=='0' &&
(OptionDlg.m_strOutlowi2c[1]=='x' || OptionDlg.m_strOuthiI2C[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOutlowi2c, "%04x", &chSend);
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevelI2C = 0;
                                }
                        }
                }
        }
}
void CDAQView::CheckLevel1WIRE(float fData)
{
        float fAverage=0;
        if (ONEwirelistLevel.GetCount()<10)
        {
                ONEwirelistLevel.AddTail(fData);
        }else
        {
                ONEwirelistLevel.RemoveHead();
                ONEwirelistLevel.AddTail(fData);
        }
        POSITION pos = ONEwirelistLevel.GetHeadPosition();
        for (int i=0;i<ONEwirelistLevel.GetCount();i++)
        {
                fAverage += ONEwirelistLevel.GetNext(pos);
        }
        fAverage = fAverage/ONEwirelistLevel.GetCount();
        if (iStatusLevel1Wire!=1 && (fAverage > OptionDlg.m_fhiTh1Wire))
        {
                UpdateData(true);
                if (OptionDlg.m_bChk1Wire==TRUE && OptionDlg.m_strOutHi1Wire!="")
                {
                        int chSend = atoi(OptionDlg.m_strOutHi1Wire);
```

```
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevel1Wire = 1;
                                }
                        }
                        else if (OptionDlg.m_strOutHi1Wire[0]=='0' &&
(OptionDlg.m_strOutHi1Wire[1]=='x' || OptionDlg.m_strOutHi1Wire[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOutHi1Wire, "%04x", &chSend);
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevel1Wire = 1;
                                }
                        }
                }
        }else if (iStatusLevel1Wire!=0 && (fAverage < OptionDlg.m_fLowTh1Wire))
        {
                UpdateData(true);
                if (OptionDlg.m_bChk1Wire==TRUE && OptionDlg.m_strOutLow1Wire!="")
                {
                        int chSend = atoi(OptionDlg.m_strOutLow1Wire);
                        if (chSend!=0)
                        {
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevel1Wire = 0;
                                }
                        }
                        else if (OptionDlg.m_strOutLow1Wire[0]=='0' &&
(OptionDlg.m_strOutLow1Wire[1]=='x' || OptionDlg.m_strOutHi1Wire[1]=='X'))
                        {
                                sscanf_s(OptionDlg.m_strOutLow1Wire, "%04x", &chSend);
                                if(SendData(chSend))
                                {
                                        UpdateLEDData(chSend);
                                        iStatusLevel1Wire = 0;
                                }
                        }
                }
        }
}
```

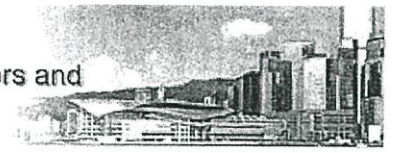
ภาคผนวก ค.

ผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่

1. S. Thanee, S. Somkuarnpanit and S. Khuntaweetep,"**Implementation of Multi-Protocol, Data Acquisition with High Speed USB Interface, Using FPGA**," International Conference on Computer Science (ICCS'10), Hong Kong, 17-19 March, 2010.

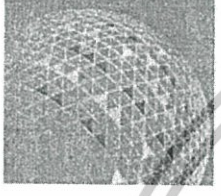
International Association of Engineers

International MultiConference of Engineers and Computer Scientists 2010



ICCS 2010 Home

- Home Page
- Conference Topics
- Publications
- Paper Submission
- Registration
- Important Dates
- Conference Committee
- Travel & Accommodation
- Conference Program
- Past IAENG
- Contact IMECS'2010



IAENG International Conference on Computer Science (ICCS'10)

Hong Kong, 17-19 March, 2010

Conferences | Publications | Membership | About IAENG | FAQ | Contact Us

The IAENG International Conference on Computer Science (ICCS'10) will take place in Hong Kong, 17-19 March, 2010

The conference ICCS'10 is held under the International MultiConference of Engineers and Computer Scientists 2010. The IMECS 2010 is organized by the International Association of Engineers (IAENG), and serves as good platforms for the engineering community members to meet with each other and to exchange ideas. The last IMECS 2009 has attracted more than one thousand participants from over 50 countries. All submitted papers will be under peer review and accepted papers will be published in the conference proceeding (ISBN: 978-988-17012-8-2). The abstracts will be indexed and available at major academic databases. The accepted papers will also be considered for publication in the special issues of the journal Engineering Letters, in IAENG journals and in edited books.

Calls for Manuscript Submissions

Important Dates:

Draft Manuscript submission deadline (extended): 15 January, 2010 (past)

Camera Ready Papers Due & Registration Deadline (extended): 30 January, 2010

ICCS 2010: 17 - 19 March, 2010

Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol I,
IMECS 2010, March 17 - 19, 2010, Hong Kong

Implementation of Multi-Protocol, Data Acquisition With High Speed USB Interface, Using FPGA

S. Thanee S. Somkuarnpanit S. Khuntaweetep

***Abstract*—This paper describes the implementation of the FPGA as a data acquisition system with high-speed USB interface. This can simplify the data interfacing to the PC by installing most data transfer protocols into one system. The FPGA has the advantage that it allows individual modules on a chip to work independently from each other. Therefore, we can utilize the FPGA as a performance solution for a system which is multi-channeled, with connections to four ADC signals, with the four different protocols of: Parallel, SPI, I²C and One-Wire. In addition, the visual C⁺⁺ programming for the user interface and data storage application, on a PC, helps this design to respond better to high rates of data acquisition.**

Keywords: Data acquisition, FPGA, high-speed USB

I. Introduction

The data acquisition system has been broadly applied in the automatic test and measurement systems for engineering and science, as we are able to either adjust or define values via controlling software [1]. The main function of the software is to collect the data from any peripheral, input devices, such as meters or sensors and to transfer it into the main database and show the results, in real time, on the screen, or to display them in a summary graph.

This paper proposed the design of the data acquisition system using FPGA [3] interfaced to a PC [2]. The system has the capability of receiving digital signals from multi-channel sensors with four, different, ADC protocols.

II. Overall System

The overall system is shown in Figure 1. It represents connections to four different ADC (analog to digital converter) sensors with four different protocols: Parallel, SPI, I²C and One-Wire. The FPGA would collect individual data from all ADC sensors.

The system would process, separately, in individual protocols, and consequently produce a stream of data through the output USB port [4], which sends the ADC data to the PC.

Manuscript received December 7, 2009.

S. Thanee is a graduate student, with a Master's Degree, in the Faculty of Engineering at King Mongkut's Institute of Technology, Ladkrabang, Bangkok Thailand (e-mail: sirisak_max@yahoo.com).

S. Somkuarnpanit is now with the Department of Electronics, Faculty of Engineering at King Mongkut's Institute of Technology, Ladkrabang, Bangkok Thailand (e-mail: kssumpo@kmitl.ac.th).

S. Khuntaweetep is with the Department of Electronics, Faculty of Engineering at King Mongkut's Institute of Technology, Ladkrabang, Bangkok Thailand (e-mail: kksuchar@kmitl.ac.th).

There would be a specific application program to prepare the PC to wait for the data from a USB port. It would then interpret the data into separate data bytes for individual channels. Consequently, the data can be shown to the user, and saved to the main database at the same time. Alternatively, the user could control the system by sending instructions via the USB port.

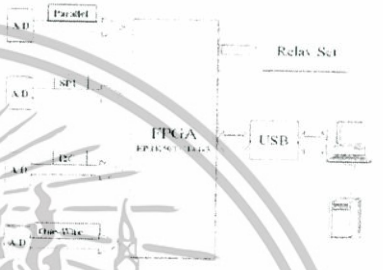


Figure 1 The overall system diagram.

III. Processing Unit

The processing unit employed for this design is EP1K10TC144-3 from the Altera Company. It has a capacity of 2,880 logic elements, or about 50,000 gates, consisting of 40,960 internal ram bits and 102 input/output ports for connection to external hardware. It supports 3 level power supply of 5V, 3.3V and 2.5V. The maximum working frequency is 180 MHz. This design has used VHDL as the language [7][8] for program writing code.

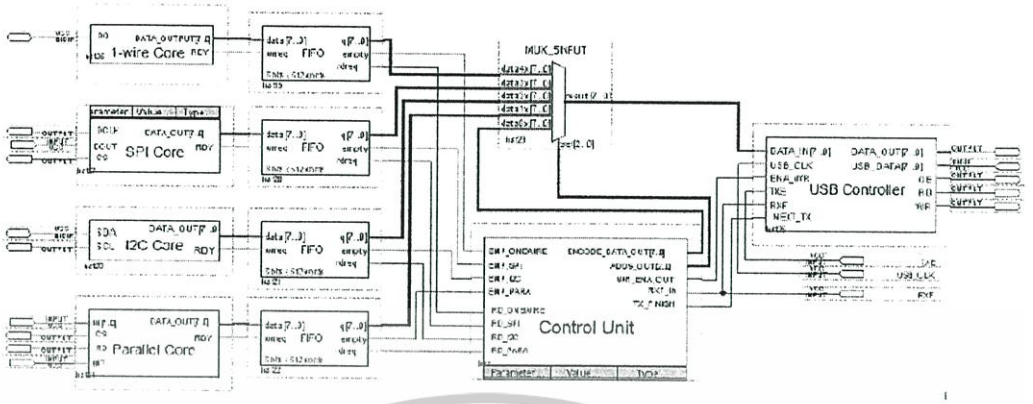
The chip is the central point of acquisition for the all sensors. Its responsibility is to bridge between the ADC inputs and the USB connection i.e. sending/receiving data to, and from, the PC. Figure 2 illustrates the internal modules within the FPGA chip [6]. These modules are described as below:

Parallel Protocol: This protocol is the traditional type for most ADCs. It has the advantage of high speed data throughput. This design uses ADC0820 as a template example. Figure 3 represents a simulation for acquiring data from this ADC. There are two main steps in the conversion process:

- FPGA sends a 'start' signal to activate the ADC and then waits for an 'acknowledgement' signal.
- After the ADC finishes converting the data, it will send an acknowledgement signal to the FPGA, then the FPGA will read all the data from the bus. After that it will send another 'start' signal to activate the ADC again for data acquisition on the next read cycle.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol I,
IMECS 2010, March 17 - 19, 2010, Hong Kong



It can be seen that, this data acquisition is very simple and fast. Thus, this protocol should properly work within a high speed system.

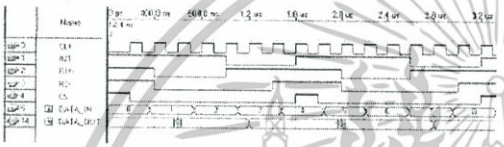


Figure 3 Simulated parallel communication protocol.

Serial Peripheral Interface (SPI) Protocol: This protocol was developed by Motorola to accomplish easy communication, and to reduce I/O ports. This design utilizes MCP3201 for building connections to FPGA. Figure 4 shows a simulation of how the FPGA obtains data from an ADC. The steps of this ADC data acquisition look like the previous protocol. The FPGA would send the control signal to activate ADC and the ADC would send the acknowledgement signal back, after it finishes converting data, to tell the FPGA to read the data from its output. The difference from the previous protocol is that the FPGA would read the stream of data in a serial pattern instead, from MSB to LSB.

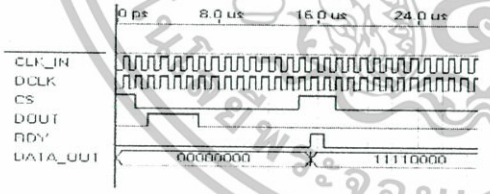


Figure 4 Simulated SPI communication protocol.

Inter-Integrated Circuit Protocol (I^2C): I^2C communication is a protocol which is designed to reduce the I/O ports. It requires only two signal connection wires called SCL and SDA. This design uses PCF8591 ADC as a template example. The procedure for this ADC data acquisition is quite complicated, as in the following steps:

- The FPGA sends a signal to start the bus and to identify the address of the device.
- The FPGA defines the write mode and sends a command to the ADC it wants to convert.
- The FPGA defines the 'read' mode and identifies which channel it wants to read from.
- The FPGA reads the data from the I^2C bus.

At this point, the data reading has completed 1 time cycle. Figure 5 illustrates the example of data reading by this protocol. We can see that one reading cycle of this protocol takes more time than the previous protocols whereas it has the good point of using only two wires in the data communication.

One-Wire Protocol: This protocol employs only one signal line in communications. The bus is not active unless all data has been transferred. We used the temperature sensor DS1820 as the input of this One-Wire acquisition data protocol.

The protocol separates the data bit by a *time slot*. It has a length of between 60-960 µs, depending on the user definition and the status of communications between master and slave devices. There are four status types:

- Reset: used to start the communication
- Write data "1" to slave device
- Write data "0" to slave device
- Read data from slave

The first step for this protocol is the 'reset' from the FPGA (master device). The FPGA would send a reset signal to the bus, and wait for an acknowledgement signal from sensor (slave device). After receiving the acknowledgement signal, the FPGA would send the address command to identify the sensor and let it start data conversion.

The FPGA would wait for a sensor processing the command. Then it would send the reset signal and identify the device address again, followed by the 'read' command to the sensor, to read data from the sensor memory one by one from LSB to MSB. Finally, it would send the reset signal again to the bus and wait for an acknowledgement signal.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol I,
IMECS 2010, March 17 - 19, 2010, Hong Kong

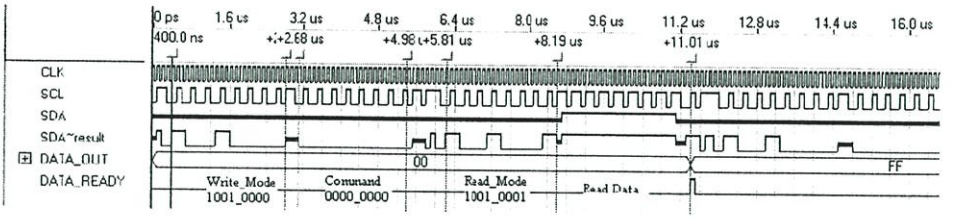


Figure 5 Simulated I²C communication protocol

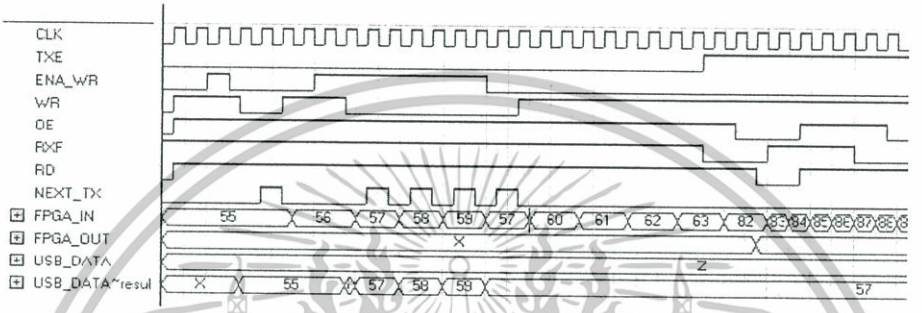


Figure 6 Simulated Control Unit and USB Controller communication

After these procedures, the data read procedure has completed one time cycle. To obtain more data, the FPGA would process the same steps again. Figure 7 displays the simulated communications for 'reset', 'acknowledgement', and 'write' "1100 1100" data.

Figure 7 The One-wire communications protocol.

Control Unit and USB Controller: If there is any data appearing on the FIFO buffer, the control unit would determine which channel the data is from. Subsequently, it would send a one-byte data code to the USB port. This first data byte informs the application program on the PC which channel the following data is from. Then it will send a signal to the USB controller again to send the data, in the FIFO, to the USB port byte by byte, until the FIFO is empty. This will mean the end of the data sending procedure in that individual channel. If there is data waiting to be sent out from another FIFO channel, the FPGA will repeat the same procedures. Figure 6 shows a simulation for interconnection between the control unit and the USB controller.

IV. The PC Interfacing

The PC interfacing used in this design is the high-speed USB which can transfer data at the rate of 480 Mbps. The transfer operation is achieved via *'frames'*, with a period of 125 µs. Each frame consists of a number of *'Transactions'*, each which consists of a number of *Packets*. These packets include information about the type of transaction, the address of the USB device, and the number of the end points required, in addition to data, and CRC, packets, and a synchronization packet.

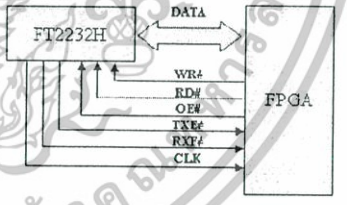


Figure 8 FT2232H/FPGA interfacing

The FT2232H chip from FTDI Corp. [5] has been used to implement this protocol in the design. It is a dual USB to parallel FIFO, bi-directional, data transfer chip with 4 kilobytes FIFO Tx and Rx buffers, which handles the entire USB protocol on the silicon level. The manufacturer provides the D2XX.DLL driver, which allows full accessibility to all the chip features through a DLL function

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

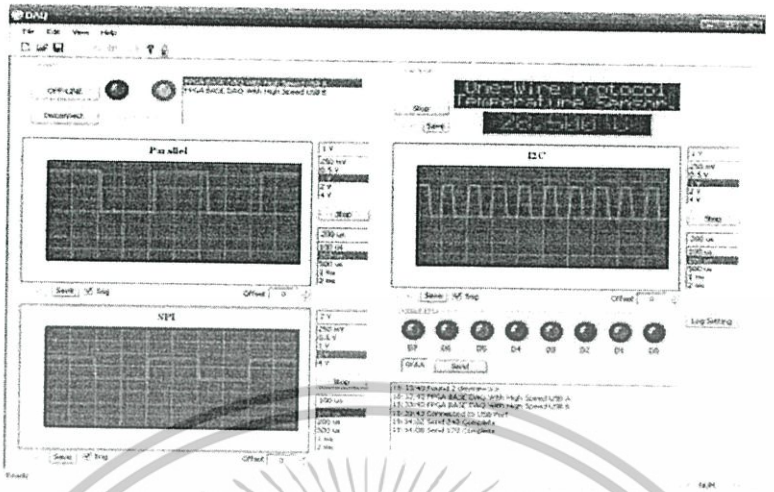


Figure 9 The sample displays in the application program

This chip provides an interface between the FPGA and the USB port with an 8 bit, bi-directional data bus, plus five control signals and one clock signal. Figure 8 illustrates the FT2232H/FPGA interfacing.

V. The Application

The application program of this design was written in the visual C++ language [9] in the following 3 layer structure :

Presentation layer: for plotting all the data graphs to be monitored by the user.

Hardware layer: continuously issuing 'read request' in a "reading worker thread" that will return one or more data items and put the data in a queue which the data acquisition layer can read from.

Data acquisition layer: When the data requires some kind of interpretation, each sensor will have a thread that collects the data, and saves it in the PC. When this thread runs, it moves the data from the queue, shared with the hardware layer, into its own container, and updates the current value shared with the presentation layer.

This application is responsible for reading data from each protocol and presenting it to the user in a graphic oscilloscope-like form. Since the one-wire protocol is used for temperature sensors, it will show the temperature level number instead. Figure 9 displays an example of the application program in operation.

VI. Conclusions

From our simulation results, we can see that the parallel protocol is the fastest, whereas, it uses the most connective wires. On the other hand, the I^2C or one-wire protocol requires only two wires and one wire, respectively, for the connection, but the bit transfer rate is much slower than general the ADC. We can use this knowledge when considering the use of a protocol in most applications.

Finally, we may now claim that our data acquisition system with channels for multiple protocols is a useful solution. Consisting of input channels with individual ADC protocols, our system allows the channels to work independently from each other. In addition, with a large number of I/O ports in the FPGA, it is feasible to add more channels in the future. Eventually, we may be able to utilize the maximum of 102 I/O ports, being the maximum number, for this FPGA.

References

[1] Ziad Salem, Ismail Al Kamal, Alaa Al Bashar "A Novel Design of an Industrial Data Acquisition System", Proceeding of International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA 2006, pp. 2589-2594.

[2] Jorge Yáñez, David Quintana, Camilo Quintáns, José Farña, Juan J. Rodríguez-Andina," FPGA-based system for the education in data acquisition and signal generation", Proceeding of Industrial Electronics Society Conference, IECON 2005, pp. 2168-2173.

[3] A. Sagahyroon ,T. Al-khudairi, "FPGA Based Acquisition of Sensor Data" Proceeding of International Conference on Industrial Technology, ICIT 2004, pp. 1398 – 1401.

[4] M. Popa, M. Marcu, A. S. Popa," A Microcontroller Based Data Acquisition System with USB Interface", Proceeding of International Conference on Electrical, Electronic and Computer Engineering, ICEEC 2004 pp. 206-209.

[5] FTDI Corporation, www.ftdichip.com, Glasgow City, Scotland,

[6] Hamblen J., Furman M., "Rapid Prototyping of Digital Systems", Quartus II Edition, Springer Science+Business Media Inc., New York, USA, 2006

[7] Douglas L. Perry "VHDL Programming By Example", Forth Edition, McGraw-Hill Company, USA, 2002.

[8] Pong P. Chu "RTL Hardware Design Using VHDL", First Edition, John Wiley & Sons, Inc., Hoboken, Canada, 2006.

[9] George Shepherd, David Kruglinski "Programming with Microsoft Visual C++ .NET", Sixth Edition, Microsoft Press, Washington, USA, 2003.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

ประวัติผู้เขียน

นายศิริศักดิ์ ธานี เกิดเมื่อวันที่ 1 กันยายน พ.ศ.2521 ที่จังหวัดอุบลราชธานี สำเร็จการศึกษาปริญญาตรีอุตสาหกรรมศาสตรบัณฑิต สาขาเทคโนโลยีโทรคมนาคม จากภาควิชาเทคนิคอุตสาหกรรม (วิศวกรรมสารสนเทศในปัจจุบัน) คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ในปีการศึกษา 2545 และเข้าศึกษาต่อในระดับปริญญาโท หลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมอิเล็กทรอนิกส์ ภาควิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2548