การออกแบบและสร้างระบบการส่งข้อมูลแบบขนาน สำหรับการควบคุมเครื่องมือวัดผ่านระบบบัสมาตรฐาน IEEE-488 (GPIB)

A DESIGN AND CONSTRUCTION OF PARALLEL DATA TRANSMISSION SYSTEM FOR CONTROL MEASURING EQUIPMENTS USING IEEE-488 STANDARD BUS (GPIB)

นายนพดล มณีรัตน์
MR. NOPPADOL MANEERAT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2540

ISBN 974-621-970-7

A DESIGN AND CONSTRUCTION OF PARALLEL DATA TRANSMISSION SYSTEM FOR CONTROL MEASURING EQUIPMENTS USING IEEE-488 STANDARD BUS (GPIB)

NOPPADOL MANEERAT

A THESIS SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
1997

ISBN 974-621-970-7

หัวข้อวิทยานิพนธ์	การออกแบบและสร้างระบบการส่งข้อมูลแบบขนานสำหรับการควบคุมเครื่องมือวัดผ่านระบบบัสมาตรฐาน IEEE-488 (GPIB)
นักศึกษา	นายนพดล มณีรัตน์
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ดร.กิติพล ชิตสกุล
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า
ภาควิชา	วิศวกรรมอิเลกทรอนิกส์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2540

บทคัดย่อ

การติดต่อสื่อสารผ่านคอมพิวเตอร์นั้นแบ่งได้เป็น 2 แบบคือ การสื่อสารแบบอนุกรม เช่น มาตรฐาน RS 232C เป็นต้น และการสื่อสารแบบขนาน เช่น มาตรฐาน IEEE-488 (GPIB) เป็นต้น ในการติดต่อสื่อสารระหว่างคอมพิวเตอร์กับเครื่องมือวัดต่างๆ ในทางอุตสาหกรรมนั้น ส่วนใหญ่จะนิยมใช้ระบบบัสมาตรฐาน IEEE-488 (GPIB) เป็นมาตรฐานในการติดต่อ เช่น การควบคุมการทำงานของ Digital Mutimeter, Printer, Oscilloscope เป็นต้น เนื่องจากการติดต่อสื่อสารผ่านระบบบัสดังกล่าวมีความเร็วสูง และสามารถที่จะเชื่อมต่ออุปกรณ์หลายๆ ตัวร่วมกันในระบบได้พร้อมๆ กัน การติดต่อสื่อสารระหว่างคอมพิวเตอร์ซึ่งทำหน้าที่เป็นตัวควบคุมระบบกับเครื่องมือวัดที่เชื่อมต่ออยู่ในระบบบัสมาตรฐาน IEEE-488 (GPIB) นั้น จำเป็นต้องมีการ์ดอินเตอร์เฟสเพื่อใช้ในการสื่อสารส่งข้อมูลระหว่างกัน สำหรับการ์ดอินเตอร์เฟสนั้นมีไอซีหลายเบอร์ที่ใช้ทำหน้าที่ในการประมวลผลคำสั่งการควบคุมตามมาตรฐาน IEEE-488 (GPIB) ได้ เช่น TMS 9914, uPD 7210 เป็นต้น ซึ่งในวิทยานิพนธ์นี้ได้ทำการออกแบบสร้างการ์ดอินเตอร์เฟสตามระบบบัสมาตรฐาน IEEE-488 (GPIB) ขึ้นโดยใช้ไอซีเบอร์ uPD 7210 เนื่องจากสาเหตุที่การ์ดอินเตอร์เฟสนั้นมีราคาแพงและการพัฒนาโปรแกรมควบคุมทำได้ยาก จึงได้ทำการพัฒนาออกแบบการ์ดนี้ขึ้นเอง ซึ่งสามารถที่จะนำไปใช้ได้ในการเรียนการสอน สำหรับระบบการควบคุมเครื่องมือวัดนี้ได้ทำการพัฒนาโปรแกรมควบคุมบนระบบปฏิบัติการ WINDOWS ทำให้สะดวก

ต่อผู้ใช้หรือผู้ที่จะศึกษาถึงรายละเอียดของระบบ และเข้าใจถึงการทำงานของระบบได้ชัดเจนมากขึ้น ซึ่งแต่เดิมการพัฒนาโปรแกรมควบคุมต้องพัฒนาบนระบบปฏิบัติการ DOS สำหรับการทดลองประยุกต์ใช้งานนั้นได้นำระบบการควบคุมเครื่องมือวัดนี้ไปใช้ในการวิเคราะห์สเปคตรัมของสัญญาณโดยวิธี FFT (Fast Fourier Transform)

Thesis Title	A Design and Construction of Parallel Data Transmission System for Control Measuring Equipments using IEEE-488 Standard Bus (GPIB)
Student	Mr.Noppadol Maneerat
Thesis Advisor	Dr.Kitipol Chitsakul
Level of Study	Master of Engineering in Electrical Engineering
Department	Electronics Engineering, King Mongkut's Institute of Technology Ladkrabang
Year	1997

Abstract

The computer communication interface can be devided into two categories. The first one is serial communication such as RS 232C standard etc. and the other one is parallel communication such as IEEE-488 (GPIB) standard etc.. Nowaday almost communications between computers and industrial equipments such as digital multimeter, printer or oscilloscope etc. use IEEE-488 (GPIB) standard bus becuase of its versatility with high speed and capability of connection with many kind of equipments into a system. The communication between computer system controller, and measuring equipments which connect to IEEE-488 (GPIB) standard bus system, however, need the interface cards to operate the connection together. Despite availability of the commercial IEEE-488 controller cards nowaday, they still have high cost and the complete documentations are not available for developing a complete system. The primary objective of this research is to develop a interface card based on available processors such as uPD 7210 including the softwares control developed on WINDOWS operating system for using with a microcomputer as system controller. Some applications of our system such as spectrum analyzer are also developed to verify the performances. The results show not only the high performances in the realworld applications but the know-how of development also provides for more understanding of the system IEEE-488

which a department can take benefit of its simplicity as a learning tool in a class of modern instrumentations.

กิตติกรรมประกาศ

การจัดทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงลงได้ด้วยดี เนื่องจากบุคคลและหน่วยงานต่อไปนี้ได้ช่วยสนับสนุนด้วยดีตลอดมา ผู้เขียนขอขอบพระคุณทุกๆ ท่านดังต่อไปนี้

คุณพ่อ คุณแม่ ผู้ซึ่งคอยเลี้ยงดู อบรมสั่งสอน ให้การสนับสนุนในการศึกษา เป็นกำลังใจและห่วงใยตลอดมา

ดร.กิติพล ซิตสกุล และ รศ.ดร.มนัส สังวรศิลป์ ผู้ซึ่งสละเวลาอันมีค่าคอยให้คำปรึกษา อบรมสั่งสอน ช่วยเหลือและชี้แนะแนวทางในการแก้ปัญหา ตลอดจนความรู้ความเข้าใจต่างๆ ที่เป็นประโยชน์เสมอมา

ผศ.ดร.บุญวัฒน์ อัตชู ที่ให้ความเมตตาดูแลเอาใจใส่และคำปรึกษาที่ดียิ่ง

ดร.สมศักดิ์ ชุมช่วย ที่คอยช่วยเหลือและให้คำแนะนำที่เป็นประโยชน์

ขอขอบพระคุณครูอาจารย์ทุกท่านที่ได้ให้ความรู้ตลอดมา

ขอขอบคุณพี่ๆ เพื่อนๆ และน้องๆ ทุกคน โดยเฉพาะ

ดร.ปราโมทย์ วาดเขียน ดร.สุรพันธ์ เอื้อไพบูลย์ คุณวิภา แสงพิสิทธิ์ คุณธีระณัฐ เริ่มคิดการ คุณมานิตย์ เกียรติกำจายขจร คุณบัณฑิต พัสยา คุณสุพจน์ จันทร์วิพัฒน์ คุณศิริชัย ปริตโตทกพร คุณวัฒนา โพธิ์เจริญ และคุณทัฬห์ รุ่งสว่าง ที่คอยช่วยให้กำลังใจและช่วยเหลือในการจัดเตรียมต้นฉบับในการทำวิทยานิพนธ์นี้

เจ้าหน้าที่ภาควิชาวิศวกรรมอิเลกทรอนิกส์ คณะวิศวกรรมศาสตร์ทุกท่าน ที่มีส่วนช่วยสนับสนุนอุปกรณ์และสถานที่ทำการวิจัย

เจ้าหน้าที่แผนกสารสนเทศ คณะวิศวกรรมศาสตร์ทุกท่าน ที่คอยช่วยให้กำลังใจและเอื้อเฟื้อสนับสนุนอุปกรณ์ในการนำเสนอผลงานวิจัยนี้

มูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร และบัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่สนับสนุนทุนในการศึกษาค้นคว้าและการทำวิทยานิพนธ์จนสำเร็จลุล่วงลงด้วยดี

รวมทั้งทุกท่านที่มีส่วนช่วยเหลือ แต่ไม่มีโอกาสกล่าวนามมาในที่นี้

นพดล มณีรัตน์

สารบัญ

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

หน้า

บทที่ 1
บทนำ

ความเป็นมาของงานวิจัย

ในปัจจุบันระบบเครื่องมือวัดและควบคุมทางอุตสาหกรรมหลายอย่างต้องทำงานเกี่ยวเนื่องและสัมพันธ์กันได้ภายใต้ตัวควบคุมระบบตัวเดียวหรือหลายตัว เพื่อให้ระบบสามารถทำงานได้อย่างมีประสิทธิภาพ การควบคุมและการรับส่งผ่านข้อมูลระหว่างตัวควบคุมและเครื่องมือวัดหลายๆ อย่าง จำเป็นต้องใช้ระบบบัสที่เป็นมาตรฐาน เพื่อที่จะทำให้การควบคุมและส่งผ่านข้อมูลในระบบเป็นไปอย่างถูกต้องแม่นยำ รวมถึงความง่ายและสะดวกในการเชื่อมต่อเครื่องมือเหล่านั้นเข้าด้วยกัน อีกทั้งยังทำให้สามารถที่จะนำอุปกรณ์ที่ต่างผู้ผลิตมาใช้งานร่วมกันบนระบบเดียวกันได้ ซึ่งระบบบัสที่ใช้กันในเครื่องมือวัดทางอุตสาหกรรมส่วนใหญ่จะใช้ระบบบัสมาตรฐาน IEEE-488 (GPIB) ซึ่งระบบบัสดังกล่าวมีการส่งผ่านข้อมูลแบบขนาน 8 บิท (ทำให้การส่งข้อมูลสามารถทำได้อย่างรวดเร็ว)

การอินเตอร์เฟสเครื่องมือวัดเข้ากับไมโครคอมพิวเตอร์ที่ใช้งานทั่วไป ซึ่งทำหน้าที่เป็นตัวควบคุมระบบเครื่องมือวัดนั้น จำเป็นต้องมีการ์ดอินเตอร์เฟสเพิ่มเติมเพื่อใช้สำหรับการอินเตอร์เฟสเครื่องมือวัดต่างๆ ในระบบเข้ากับตัวควบคุม แต่ปัญหาก็คือ เนื่องจากการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB) ที่ใช้ในการอินเตอร์เฟสระหว่างตัวควบคุมระบบกับอุปกรณ์ในระบบเครื่องมือวัดนั้น เมื่อต้องซื้อจากบริษัทผู้ผลิตจำเป็นต้องใช้ซอฟต์แวร์ไดร์ฟเวอร์ของการ์ดดังกล่าวควบคู่กันไปด้วย ทำให้เป็นอุปสรรคต่อการพัฒนาระบบให้ยืดหยุ่นต่อการใช้งานต่อไป นอกจากนี้การ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB) ที่มีขายอยู่ทั่วไป ในปัจจุบันยังคงมีราคาแพงมาก อีกทั้งการพัฒนาโปรแกรมควบคุมระบบยังทำได้ยาก เนื่องจากเป็นการยากที่จะทำความเข้าใจโครงสร้างโปรแกรมของซอฟท์แวร์ไดร์ฟเวอร์ของการ์ดดังกล่าวได้

ดังนั้นวิทยานิพนธ์ฉบับนี้ขอเสนอผลงานวิจัย เพื่อเป็นแนวทางในการแก้ปัญหาดังกล่าวมาข้างต้น (และนำเสนอการประยุกต์ใช้งานระบบเครื่องมือวัดดังกล่าว) ซึ่งจะช่วยประหยัดค่าใช้จ่ายลง และยังสามารถที่จะพัฒนาที่จะนำไปใช้งานด้านอื่นๆ ได้อีกต่อไป

จุดประสงค์และขอบเขตของงานวิจัย

งานวิจัยนี้เป็นการสร้างการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB) รวมถึงการประยุกต์ใช้งานในระบบเครื่องมือวัด โดยโปรแกรมควบคุมระบบเครื่องมือวัดจะทำการควบคุมบนระบบปฏิบัติการ WINDOWS ทำให้ผู้ใช้งานหรือผู้ที่จะศึกษาทราบรายละเอียดของระบบและสะดวกต่อการเข้าใจถึงการทำงานของระบบดังกล่าวได้ชัดเจนมากขึ้น ระเบียบวิธีการวิจัยแบ่งออกได้เป็น 3 ขั้นตอนหลัก คือ

ขั้นตอนที่ 1 : การออกแบบและสร้างการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB)

ขั้นตอนที 2 : การพัฒนาโปรแกรมควบคุมระบบ

ขั้นตอนที่ 3 : การทดสอบและการประยุกต์ใช้งาน

โครงร่างของวิทยานิพนธ์

วิทยานิพนธ์นี้ประกอบด้วยบทต่างๆ ดังนี้

บทที่ 1 ว่าด้วยเรื่องบทนำ

บทที่ 2 ว่าด้วยเรื่องความรู้พื้นฐานเกี่ยวกับระบบบัสมาตรฐาน IEEE-488 (GPIB)

บทที่ 3 ว่าด้วยเรื่องสถาปัตยกรรมของระบบ

บทที่ 4 ว่าด้วยเรื่องการออกแบบการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE -488 (GPIB)

บทที่ 5 ว่าด้วยเรื่องการพัฒนาโปรแกรมควบคุมระบบ

บทที่ 6 ว่าด้วยเรื่องการทดสอบและการประยุกต์ใช้งาน

บทที่ 7 ว่าด้วยเรื่องบทสรุปและแนวทางในการพัฒนา

ท้ายสุดในภาคผนวกได้แสดงโปรแกรมทั้งหมดที่ได้พัฒนาขึ้นสำหรับการใช้งานควบคุมระบบเครื่องมือวัดในงานวิจัยนี้

บทที่ 2
ความรู้พื้นฐานเกี่ยวกับระบบบัสมาตรฐาน IEEE-488 (GPIB)

กล่าวนำ

การใช้งานระบบเครื่องมือวัดบนระบบบัสมาตรฐาน IEEE-488 (GPIB) จำเป็นต้องทราบโครงสร้างและมาตรฐานของระบบบัสดังกล่าว เพื่อที่จะนำคุณสมบัติเหล่านี้ไปใช้ในการออกแบบการ์ดอินเตอร์เฟสและการพัฒนาโปรแกรมควบคุมระบบเครื่องมือวัด ในบทนี้จะกล่าวถึงประวัติความเป็นมาของระบบบัสแบบ IEEE-488 (GPIB) โครงสร้างและมาตรฐานของระบบบัส เพื่อให้เกิดความเข้าใจเกี่ยวกับการใช้งานและสัญญาณต่างๆ บนระบบบัสมาตรฐาน IEEE-488 (GPIB) ได้ดียิ่งขึ้น

ประวัติความเป็นมา [1]

ในปัจจุบันเครื่องมือวัดในทางอุตสาหกรรมที่ทันสมัยและมีสมรรถนะสูงนั้น จะมีการทำงานและแสดงผลในลักษณะเป็นดิจิตอล ในการใช้งานระบบเครื่องมือวัดจะมีเครื่องมือวัดหลายอย่างต่อร่วมกันในระบบ ซึ่งแต่เดิมบริษัทเครื่องมือวัดแต่ละบริษัทจะมีระบบบัสในการเชื่อมต่อเป็นของตนเอง ทำให้เกิดความยุ่งยากในการออกแบบการเชื่อมต่ออุปกรณ์เครื่องมือนี้เข้ากับระบบในกรณีที่ผู้ใช้มีเครื่องมือวัดหลายๆ บริษัท และยังเป็นอุปสรรคต่อการพัฒนาเครื่องมือวัดใหม่ๆ ของบริษัทอีกด้วย บริษัทต่างๆ ที่เป็นผู้ผลิตเครื่องมือวัดในประเทศสหรัฐอเมริกาจึงได้ร่วมกันพัฒนาระบบบัสอินเตอร์เฟสที่เป็นมาตรฐานสำหรับการใช้งานร่วมกันขึ้น และในประเทศเยอรมนีก็ได้มีการพิจารณาหาระบบบัสอินเตอร์เฟสที่จะใช้เป็นมาตรฐานร่วมกันขึ้นเช่นกันโดยความช่วยเหลือของ IEC (International Electrotechnical Commission) และในปี ค.ศ. 1972 สถาบันวิศวกรไฟฟ้าและอิเลคทรอนิคส์ของอเมริกา (Institute of Electrical and Electronics Engineering :IEEE) ได้จัดประชุมเพื่อวางแผนที่จะหามาตรฐานของระบบบัสข้อมูลดังกล่าวขึ้น ซึ่งบริษัท Hewlett Packard บริษัทผู้ผลิตเครื่องมือวัดรายใหญ่รายหนึ่งของประเทศสหรัฐอเมริกาได้ทำการพัฒนาระบบบัสมาตรฐานเครื่องมือวัดของตนอยู่ก่อนแล้ว มีชื่อว่า HPIB (Hewlett Packard Interface Bus) จึงได้เสนอระบบบัสดังกล่าวให้ IEEE พิจารณา และได้รับการยอมรับ

เป็นมาตรฐานลำดับที่ 488 ในปี ค.ศ. 1975 จึงเรียกมาตรฐานดังกล่าวว่า มาตรฐาน IEEE std 488-1975 และต่อมาได้มีการปรับปรุงขึ้นในปี ค.ศ. 1978 เรียกว่า มาตรฐาน IEEE std 488-1978 ซึ่งก็คือระบบบัสแบบ IEEE-488 (GPIB) นั่นเอง แต่ในขณะเดียวกัน IEC ก็ได้มีการกำหนดระบบบัสมาตรฐานสำหรับการเชื่อมต่อในระบบเครื่องมือวัดขึ้นมาอีกมาตรฐานหนึ่ง เรียกว่า IEC 625-1 โดยมีรายละเอียดทุกอย่างเหมือนกับมาตรฐาน IEEE std 488-1978 ทุกประการ เพียงแต่มีความแตกต่างกันในตำแหน่งของขาสัญญาณที่ขั้วต่อเพียงเล็กน้อยเท่านั้น ดังแสดงในตารางที่ 1 ดังนั้น ระบบบัสแบบ GPIB (General Purpose Interface Bus) นั้นจึงหมายความถึงบัสที่ใช้อินเตอร์เฟสทั่วไป ซึ่งจะเป็นการเรียกรวมทั้งระบบบัสมาตรฐาน IEEE std 488-1978 และมาตรฐาน IEC 625-1 รวมกัน โดยบัสมาตรฐานที่ใช้ในการรับส่งข้อมูลของอุปกรณ์ในระบบนี้สามารถที่จะต่อเครื่องมือเพิ่มเติมเข้ามาในระบบได้โดยไม่ต้องเพิ่มเติมวงจรหรืออุปกรณ์อื่นอีกเลย ทำให้มีความสะดวกและง่ายต่อผู้ใช้งานมาก

ตารางที่ 1

แสดงการเปรียบเทียบการจัดตำแหน่งสัญญาณของบัสมาตรฐาน IEEE-488 กับ IEC 625-1

IEEE Standard		IEC Standard	
Pin	Designation	Pin	Designation
1	DIO 1	1	DIO 1
2	DIO 2	2	DIO 2
3	DIO3	3	DIO 3
4	DIO4	4	DIO 4
5	EOI	5	REN
6	DAV	6	EOI
7	NRFD	7	DAV
8	NDAC	8	NRFD
9	IFC	9	NDAC
10	SRQ	10	IFC
11	ATN	11	SRQ
12	SHIELD	12	ATN

ตารางที่ 1 (ต่อ)

แสดงการเปรียบเทียบการจัดตำแหน่งสัญญาณของบัสมาตรฐาน IEEE-488 กับ IEC 625-1

IEEE Standard		IEC Standard	
Pin	Designation	Pin	Designation
13	DIO 5	13	SHIELD
14	DIO 6	14	DIO 5
15	DIO 7	15	DIO 6
16	DIO 8	16	DIO 7
17	REN	17	DIO 8
18	GND 6	18	GND 5
19	GND 7	19	GND 6
20	GND 8	20	GND 7
21	GND 9	21	GND 8
22	GND 10	22	GND 9
23	GND 11	23	GND 10
24	LOGIC GND	24	GND 11
		25	GND 12

โครงสร้างของ GPIB [1]

ส่วนประกอบพื้นฐานของระบบ GPIB แบ่งออกได้เป็น 3 ส่วน คือ

1. ตัวส่ง (Talker) ทำหน้าที่ในการส่งข้อมูล ในระบบสามารถมีตัวส่งได้หลายๆ ตัว แต่จะมีตัวส่งเพียงตัวเดียวเท่านั้นที่กำลังทำงานอยู่ในระบบ เพราะไม่สามารถที่จะทำงานพร้อมกันได้

2. ตัวรับ (Listener) ทำหน้าที่ในการรับข้อมูล ตัวรับในระบบนั้นสามารถมีได้หลายตัวเช่นกันและสามารถที่จะทำงานพร้อมๆ กันได้ด้วยถ้าเป็นอุปกรณ์ที่เหมือนกัน

3. ตัวควบคุม (Controller) ทำหน้าที่ควบคุมสัญญาณต่างๆ บนบัส โดยตอบสนองความต้องการของอุปกรณ์ที่จะส่งข้อมูล โดยการกำหนดตัวส่งที่จะทำการส่งข้อมูล หรือกำหนดตัวรับให้ทำการรับข้อมูลที่ส่งลงบนบัส

การทำหน้าที่ของอุปกรณ์ในระบบ อุปกรณ์หรือเครื่องมือในระบบ GPIB นั้นสามารถแบ่งตามหน้าที่การทำงานได้ดังนี้

1. ทำหน้าที่เป็นตัวส่งเท่านั้น เช่น เครื่องมือวัด เป็นต้น

2. ทำหน้าที่เป็นตัวรับเท่านั้น เช่น เครื่องพิมพ์ เป็นต้น

3. ทำหน้าที่เป็นทั้งตัวส่งและตัวรับ เช่น ไมโครคอมพิวเตอร์, เครื่องมือวัดที่สามารถควบคุมได้จากภายนอก เป็นต้น

4. ทำหน้าที่เป็นตัวส่ง ตัวรับและตัวควบคุม เช่น ไมโครคอมพิวเตอร์ เป็นต้น

สำหรับแผนผังการแสดงอุปกรณ์และสายสัญญาณต่างๆ ในระบบ GPIB นั้น แสดงดังภาพที่ 1

ข้อจำกัดของระบบบัส IEEE-488

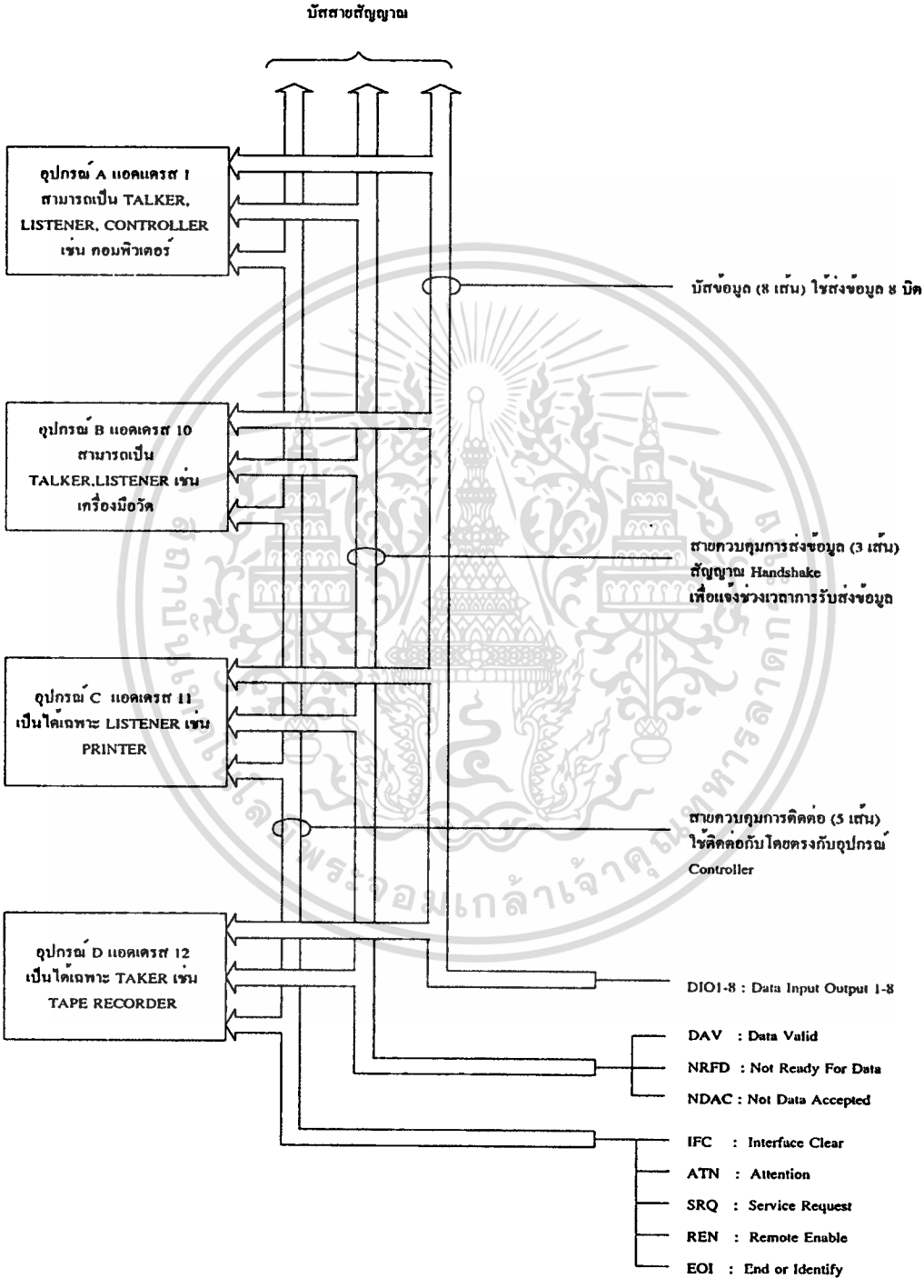
1.จำนวนอุปกรณ์ (ตัวรับ, ตัวส่ง, ตัวควบคุม) ที่ต่ออยู่กับบัสของระบบ 1 เส้น จะต้องไม่เกิน 15 เครื่อง

2. สายเคเบิ้ลที่ใช้ต่อระหว่างอุปกรณ์แต่ละตัวต้องยาวไม่เกิน 4 เมตร และความยาวรวมของสายเคเบิ้ลต้องไม่เกิน 20 เมตร

3. ความเร็วในการส่งข้อมูลต้องไม่เกิน 1 Mb/s

4. จำนวนของอุปกรณ์หรือเครื่องมือมากกว่าครึ่งหนึ่งต้องเปิดให้ทำงาน

ภาพที่ 1



แสดงแผนผังแสดงอุปกรณ์และสายสัญญาณต่างๆ ในระบบ GPIB

มาตรฐานระบบบัส IEEE-488 [3]

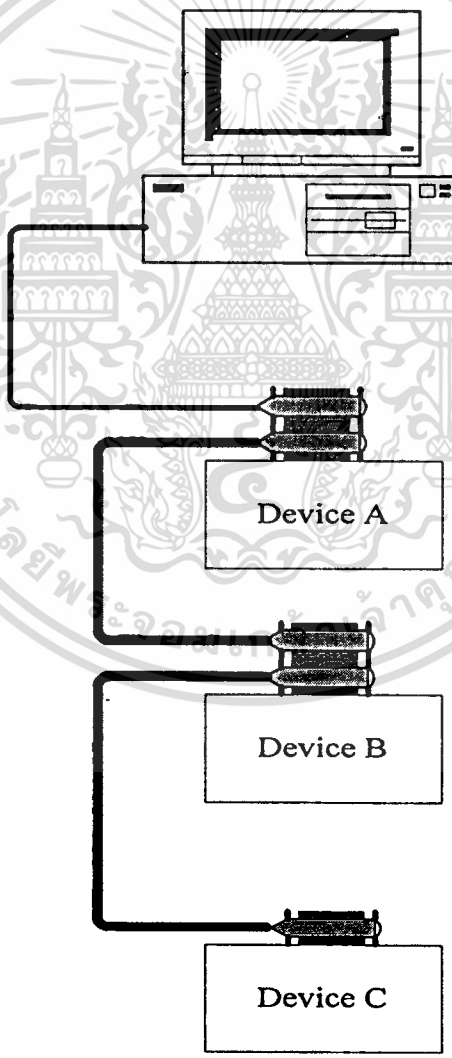
มาตรฐานของระบบบัส IEEE-488 สามารถแบ่งออกได้ ดังนี้

การเชื่อมต่ออุปกรณ์ต่างๆ ในระบบ

การเชื่อมต่ออุปกรณ์หรือเครื่องมือเข้าในระบบ GPIB นั้นมี 2 แบบ คือ

1 การเชื่อมต่อแบบเรียงต่อเนื่องกัน (Daisy chain configuration) ดังแสดงในภาพที่ 2

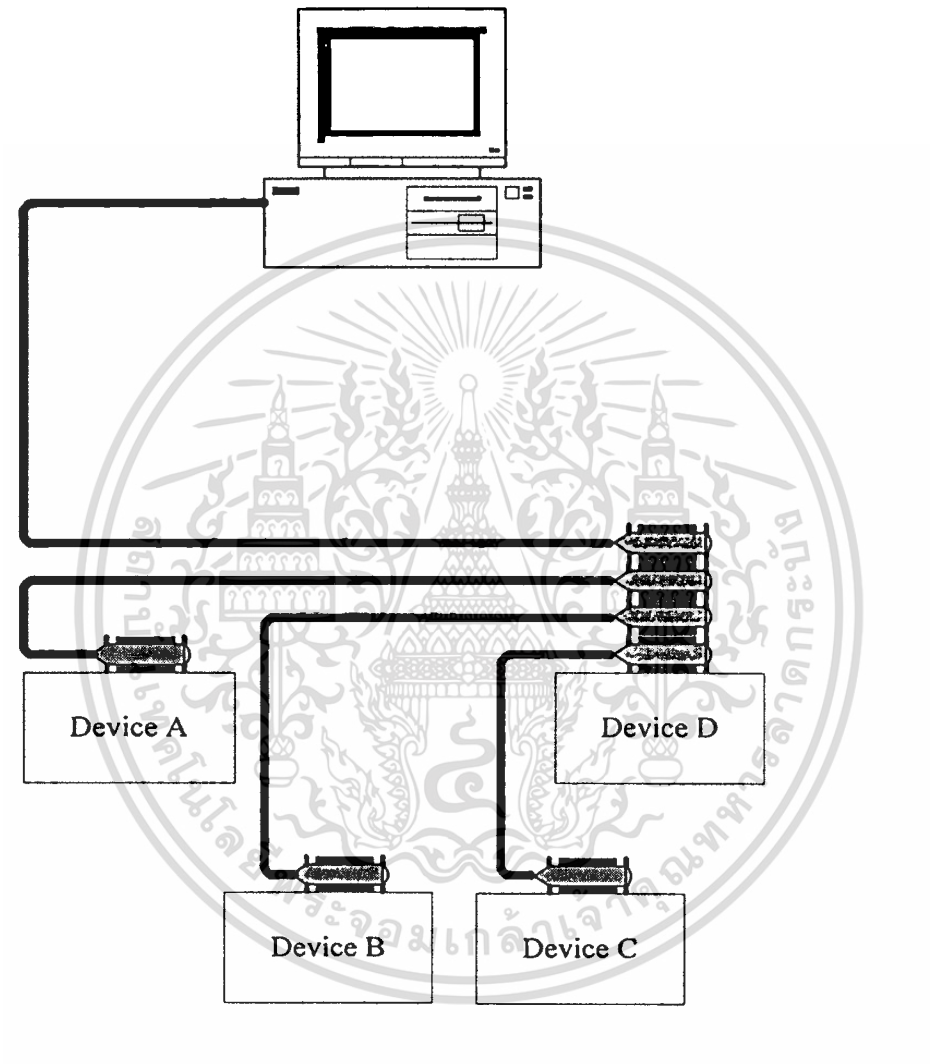
ภาพที่ 2



แสดงการเชื่อมต่อแบบเรียงต่อเนื่องกัน

2 การเชื่อมต่อแบบกระจาย (Star configuration) ดังแสดงในภาพที่ 3

ภาพที่ 3

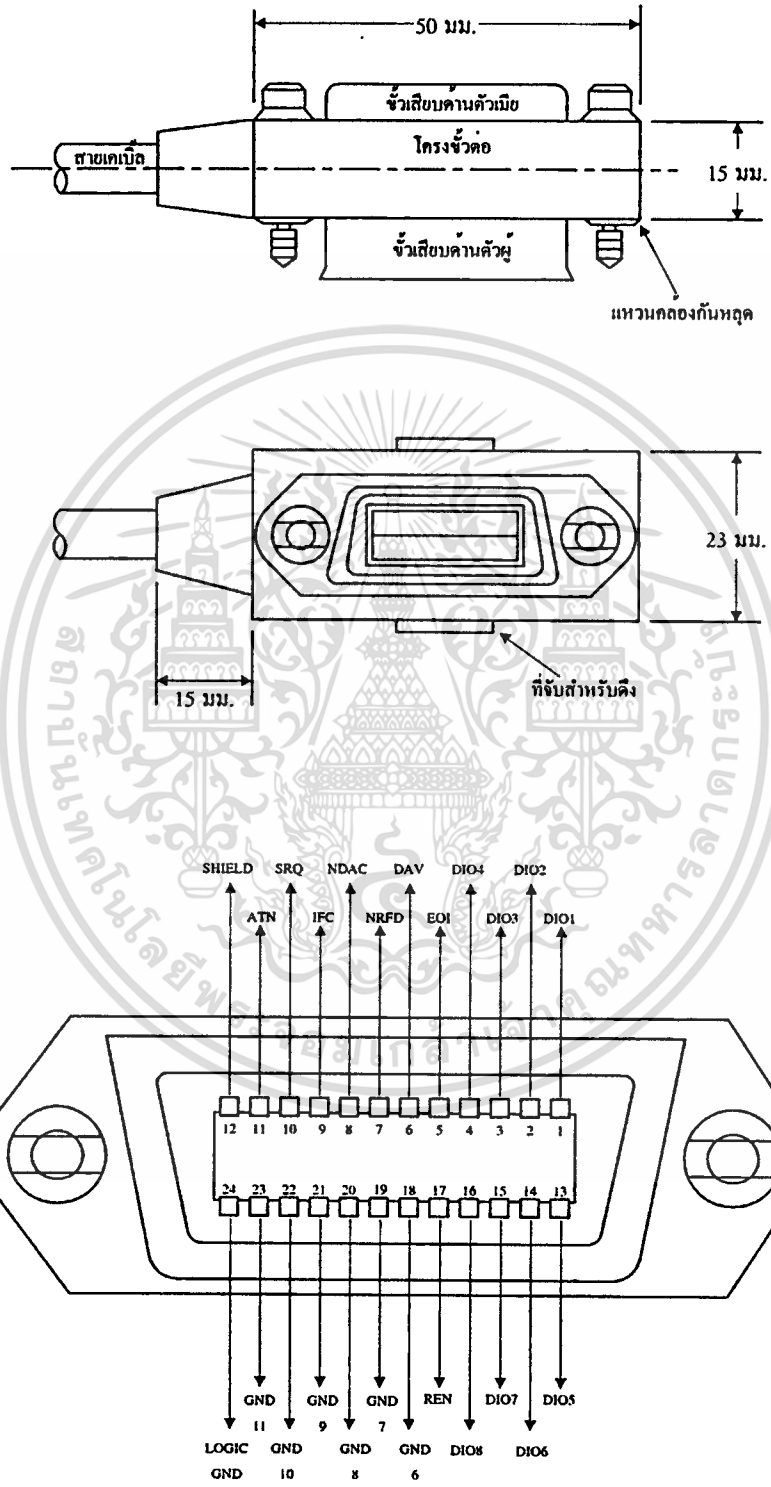


แสดงการเชื่อมต่อแบบกระจาย

ขั้วต่อและความหมายของสัญญาณต่างๆ ภายในบัส

ขั้วต่อตามมาตรฐานของ GPIB มีทั้งหมด 24 ขา มีรูปแบบและการจัดตำแหน่งของขาสัญญาณต่างๆ ดังแสดงในภาพที่ 4

ภาพที่ 4



แสดงขั้วของสัญญาณบัสมาตรฐาน IEEE-488

ความหมายของสัญญาณต่างๆ ในบัส ระบบบัส GPIB เป็นระบบบัสแบบขนาน ดังนั้น อุปกรณ์ทุกตัวที่ต่อร่วมบัสกันอยู่จึงต่อขนานกันหมด ดังนั้นสายสัญญาณทั้ง 24 เส้นในระบบจึงเชื่อมต่อถึงอุปกรณ์ในระบบทุกตัว สายสัญญาณทั้ง 24 เส้นสามารถแบ่งออกได้เป็น 3 กลุ่มสัญญาณ คือ

1. กลุ่มสัญญาณควบคุมการรับส่งข้อมูล มี 3 เส้น ใช้สำหรับการส่งข้อมูลแบบแฮนด์เช็ค ประกอบด้วย

DAV (Data Valid) เมื่อสัญญาณถูกดึงเป็นลอจิก Low โดยอุปกรณ์ตัวส่ง เป็นการแจ้งให้ทราบว่าขณะนี้ตัวส่งได้ทำการส่งข้อมูลลงไปที่บัสข้อมูลเรียบร้อยแล้ว

NRFD (Not Ready For Data) เมื่อสายสัญญาณนี้มีลอจิกเป็น Low เป็นการแจ้งให้ทราบว่า ขณะนี้ตัวรับในระบบยังไม่พร้อมที่จะรับข้อมูล เนื่องจากอุปกรณ์ในระบบยังไม่พร้อมที่จะทำงานหมดทุกตัว จนกว่าอุปกรณ์ทุกตัวจะส่งสัญญาณที่มีลอจิกเป็น High ครบทุกตัว เพื่อแสดงว่าพร้อมที่จะรับข้อมูล

NDAC (Not Data Accept) สัญญาณนี้ถูกควบคุมโดยอุปกรณ์ตัวรับ โดยจะให้ลอจิกเป็น Low ขณะที่ตัวรับกำลังเก็บข้อมูลจากบัสข้อมูลอยู่ และจะมีลอจิกเป็น High เมื่อทำการเก็บข้อมูลเรียบร้อยแล้ว

2. กลุ่มสัญญาณควบคุมการเชื่อมต่อ มี 5 เส้น ประกอบด้วย

ATN (Attention) เป็นสัญญาณที่ถูกส่งจากตัวควบคุม ใช้ในการสั่งให้อุปกรณ์ในระบบทุกตัวเตรียมพร้อมเพื่อรอรับคำสั่งต่อไป

IFC (Interface Clear) เป็นสัญญาณที่ใช้ในการเคลียร์ระบบ ถูกควบคุมโดยตัวควบคุมเท่านั้น เมื่ออุปกรณ์ในระบบได้รับสัญญาณนี้ก็จะกลับคืนสู่สภาวะเริ่มต้นใหม่ เป็นสภาวะแรกเริ่มก่อนการกำหนดฟังก์ชันการทำงานเหมือนตอนเริ่มเปิดสวิทช์

REN (Remote Enable) เป็นสัญญาณที่ถูกควบคุมโดยตัวควบคุมอีกเช่นกัน เพื่อสั่งให้อุปกรณ์ในระบบเปลี่ยนสถานะการทำงานจากโหมดการใช้งานปกติมาเป็นการทำงานแบบควบคุมโดยตัวควบคุมแทนหรือโหมดรีโมทนั่นเอง

SRQ (Service Request) เป็นสัญญาณอินเทอร์รัพท์ เพื่อบอกแก่ระบบว่าขณะนี้อุปกรณ์ต้องการการติดต่อจากตัวควบคุม

EOI (End Or Identify) เป็นสัญญาณที่ถูกควบคุมโดยตัวควบคุมหรือตัวส่งก็ได้ ใช้แสดงว่าข้อมูลชุดที่ส่งนั้นสิ้นสุดลงแล้ว

3. กลุ่มสัญญาณข้อมูล ประกอบด้วยสายสัญญาณข้อมูลทั้ง 8 เส้น ได้แก่ DIO1 - DIO8 สำหรับเป็นทางผ่านของข้อมูลของระบบ ส่วนสายสัญญาณที่เหลืออีก 7 เส้นจะเป็นกราวด์ และมีเส้นหนึ่งเป็นสาย Shield สัญญาณควบคุมที่มีอยู่อาจไม่ได้ใช้พร้อมกันทั้งหมด บางเส้นอาจไม่ได้ใช้งานในอุปกรณ์บางเครื่องก็ได้ แล้วแต่ความจำเป็นในการติดต่อ

สำหรับลอจิกที่ใช้ในระบบบัสดังกล่าวนี้จะมีลักษณะเป็นคอมพลีเมนต์ทั้งหมด คือ ลอจิก Low จะมีค่าเป็น1 และลอจิก High จะมีค่าเป็น 0

ขบวนการแฮนด์เช็ค (Handshake Procedure)

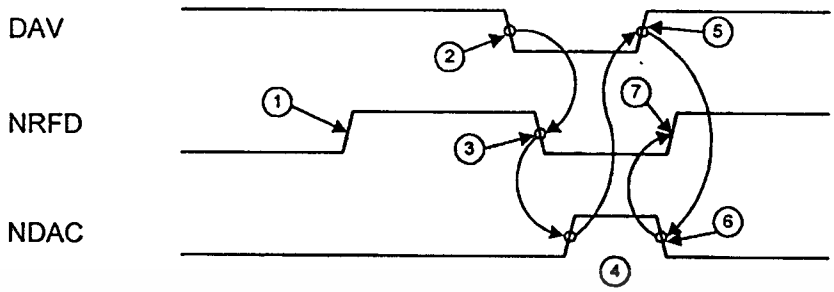
ในการสื่อสารระหว่างภายในระบบ GPIB นั้นจะเป็นการสื่อสารแบบอะซิงโครนัส คือ เมื่อมีการรับส่งข้อมูลระหว่างตัวส่งและตัวรับ ตัวส่งจะต้องแจ้งให้ตัวรับทราบว่าตัวส่งได้ส่งข้อมูลลงไปบนบัสแล้ว และให้ตัวรับทำการเก็บข้อมูลได้ เมื่อตัวรับทำการเก็บข้อมูลเสร็จแล้ว ก็จะต้องแจ้งแก่ตัวส่งให้ทราบว่า ได้รับข้อมูลที่ส่งมาเรียบร้อยแล้ว เพื่อที่ตัวส่งจะได้ทำการหยุดส่งข้อมูลหรือทำการส่งข้อมูลชุดใหม่ลงไปบนบัส กระบวนการเหล่านี้จะเกิดทุกครั้งที่มีการรับส่งข้อมูลในระบบ ซึ่งกระบวนการเหล่านี้ถูกเรียกว่า ขบวนการแฮนด์เช็ค (Handshake Procedure)

ในการพิจารณาถึงขบวนการแฮนด์เช็คนั้น จะทำการพิจารณาถึงระบบที่ไม่ซับซ้อนนักเพื่อที่จะทำความเข้าใจได้โดยง่าย โดยกำหนดให้ในระบบมีตัวส่งและตัวรับอย่างละหนึ่งตัว ดังแสดงในภาพที่ 5 ในการสื่อสารระหว่างตัวส่งและตัวรับนั้น จะมีสายสัญญาณควบคุมการรับส่งข้อมูลอยู่ 3 สัญญาณ คือ NRFD, NDAC, DAV โดยสัญญาณ DAV จะเป็นสัญญาณที่ถูกควบคุมโดยตัวส่ง ส่วนสัญญาณ NRFD, NDAC นั้นเป็นสัญญาณที่จะชี้ให้เห็นว่าตัวรับพร้อมที่จะรับข้อมูลที่ส่งลงมาบนบัสของระบบหรือไม่ สำหรับขั้นตอนของขบวนการแฮนด์เช็คสามารถแสดงได้ดังในภาพที่ 6

ภาพที่ 5

แสดงระบบการเชื่อมต่ออย่างง่าย

ภาพที่ 6



แสดงแผนผังเวลาของขบวนการแฮนด์เช็ค

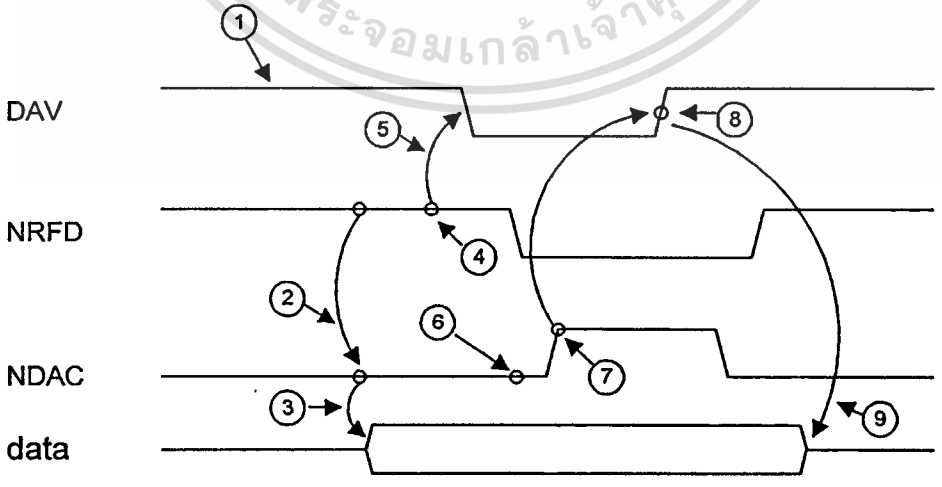
ขบวนการแฮนด์เช็คจะเริ่มขึ้นหลังจากที่ตัวควบคุมทำการบอกให้ระบบทราบว่าอุปกรณ์ตัวไหนทำหน้าที่เป็นตัวรับหรือตัวส่ง เมื่อตัวรับทราบแล้วก็จะส่งสัญญาณ NRFD ให้เป็น High (สถานะที่ 1) เพื่อบอกให้ตัวส่งทราบว่าตัวรับพร้อมที่จะรับข้อมูลแล้ว และตัวส่งก็จะทำการส่งข้อมูลลงไปบนบัสข้อมูล DIO1 - DIO8 และจะทำการรออยู่ชั่วขณะหนึ่งแล้วตัวส่งจะส่งสัญญาณ DAV ให้เป็น Low เพื่อแจ้งให้ทราบว่าขณะนี้ตัวส่งได้ส่งข้อมูลลงบนบัสข้อมูลแล้ว (สถานะที่ 2) เมื่อตัวรับทราบว่ามีข้อมูลอยู่บนบัสข้อมูลก็จะส่งสัญญาณ NRFD ให้มีค่าเป็น Low เมื่อตัวรับพร้อมที่จะรับข้อมูล (สถานะที่ 3) หลังจากตัวรับได้รับข้อมูลไปเก็บไว้ในบัฟเฟอร์เรียบร้อยแล้วก็จะส่งสัญญาณ NDAC ให้มีค่าเป็น High เพื่อแจ้งให้ทราบว่าตัวรับได้รับข้อมูลเรียบร้อยแล้ว (สถานะที่ 4) เมื่อตัวส่งได้รับสัญญาณ NDAC ที่เป็น High ตัวส่งก็จะทำการส่งสัญญาณ DAV ให้เป็น High เพื่อแจ้งให้ตัวรับไม่ต้องทำการเก็บข้อมูลนั้นอีก (สถานะที่ 5) เมื่อตัวรับได้รับสัญญาณ DAV ที่มีค่าเป็น High ก็จะส่งสัญญาณ NDAC ให้เป็น Low (สถานะที่ 6) ทำให้ข้อมูลในบัสถูกกำจัดออกไป หลังจากนั้นตัวรับก็จะส่งสัญญาณ NRFD ให้เป็น High (สถานะที่ 7) เพื่อบอกให้ทราบว่าตัวรับนั้นพร้อมที่จะรับข้อมูลชุดต่อไปที่จะถูกส่งเข้ามาในบัส เป็นอันเสร็จสิ้นขบวนการแฮนด์เช็ค

<u>ขบวนการแฮนด์เช็คเมื่อใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุมและตัวส่ง</u>

ขบวนการแฮนด์เช็คเมื่อใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุมและตัวส่งสามารถแสดงด้วยแผนผังเวลา ดังภาพที่ 7 ซึ่งจะทำให้ง่ายต่อการเข้าใจในขั้นตอนการส่งข้อมูล โดยขบวนการดังกล่าวจะเกิดขึ้นหลังจากการกำหนดอุปกรณ์ในระบบแล้ว

ขบวนการแฮนด์เช็คจะเริ่มขึ้น เมื่อตัวควบคุมส่งสัญญาณ DAV ให้เป็น High (สถานะที่ 1) ซึ่งตัวควบคุมได้เซ็ทให้สัญญาณ DAV ให้มีค่าเป็น High อยู่ก่อนแล้ว หลังจากนั้นตัวควบคุมจะทำการตรวจสอบสัญญาณ NRFD และ NDAC ว่ามีค่าเป็น High ทั้งคู่หรือไม่(สถานะที่ 2) ถ้าสัญญาณทั้งสองเป็น High ทั้งคู่ แสดงว่าอุปกรณ์ไม่พร้อมที่จะทำงาน ขบวนการแฮนด์เช็คก็จะถูกยกเลิกไป แต่ถ้าที่สถานะที่ 2 หากสัญญาณใดสัญญาณหนึ่งเป็น Low ตัวควบคุมจะทำการส่งข้อมูลลงในบัสข้อมูล (สถานะที่ 3) หลังจากนั้นตัวควบคุมจะทำการตรวจสอบสัญญาณ NRFD ว่าเป็น High หรือไม่ (สถานะที่ 4) ถ้าสัญญาณ NRFD เป็น High ตัวควบคุมก็จะส่งสัญญาณ DAV ให้มีค่าเป็น Low เพื่อบอกให้ตัวรับทราบว่ามีข้อมูลอยู่ในบัสข้อมูล (สถานะที่ 5) แต่ถ้าที่สถานะที่ 4 สัญญาณ NRFD มีค่าเป็น Low แสดงว่าขบวนการแฮนด์เช็คเกิดความผิดพลาดขึ้น จะต้องทำการเริ่มต้นใหม่ จากสถานะที่ 5 ตัวควบคุมจะรอเวลาให้ตัวรับทำการเก็บข้อมูล เมื่อถึงเวลาที่กำหนดตัวควบคุมจะทำการตรวจสอบสัญญาณว่าสัญญาณ NDAC ถูกเปลี่ยนให้เป็น High ในเวลาที่กำหนดหรือไม่ (สถานะที่ 6) ถ้าตัวรับไม่ได้รับข้อมูลในเวลาที่กำหนดขบวนการแฮนด์เช็คจะถูกยกเลิก แต่ถ้าตัวรับได้รับข้อมูลภายในเวลาที่กำหนดตัวรับจะทำการเปลี่ยนสัญญาณ NDAC ให้เป็น High (สถานะที่ 7) ตัวควบคุมก็จะทำการตอบสนองโดยการเปลี่ยนสัญญาณ DAV ให้เป็น High (สถานะที่ 8) และตัวควบคุมก็จะทำการลบข้อมูลที่อยู่ในบัสข้อมูลออกไป (สถานะที่ 9) เป็นการเสร็จสิ้นขบวนการแฮนด์เช็คดังกล่าว

ภาพที่ 7

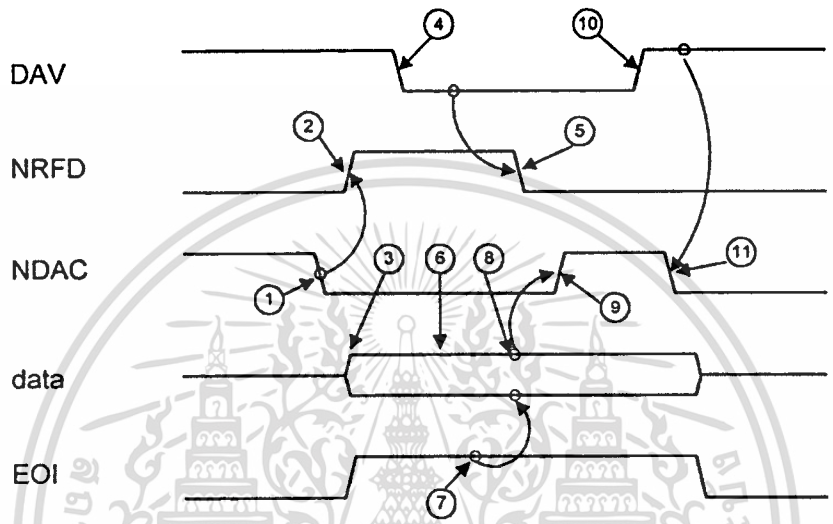


แสดงแผนผังเวลาของขบวนการแฮนด์เช็คเมื่อใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุมและตัวส่ง

<u>ขบวนการแฮนด์เช็คเมื่อใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุมและตัวรับ</u>

ขบวนการแฮนด์เช็คแบบนี้สามารถเขียนแทนด้วยแผนผังเวลา ดังภาพที่ 8

ภาพที่ 8



แสดงแผนผังเวลาของขบวนการแฮนด์เช็คเมื่อใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุมและตัวรับ

ขบวนการแฮนด์เช็คเริ่มขึ้นโดยตัวควบคุมรับรู้ว่าตัวส่งจะทำการส่งข้อมูล ตัวควบคุมจะส่งสัญญาณ NDAC ให้มีค่าเป็น Low เพื่อบอกให้ทราบว่าตัวควบคุมยังไม่ได้รับข้อมูล (สถานะที่ 1) ต่อจากนั้นตัวควบคุมจะส่งสัญญาณ NRFD ให้เป็น High เพื่อบอกให้ตัวส่งทราบว่าตัวควบคุมพร้อมที่จะรับข้อมูลแล้ว (สถานะที่ 2) ตัวส่งจะทราบได้ทันทีว่าขณะนี้สามารถที่จะส่งข้อมูลลงบนบัสข้อมูลได้แล้ว (สถานะที่ 3) จากการที่สัญญาณ NRFD มีลอจิกเป็น High และสัญญาณ NDAC มีลอจิกเป็น Low ขั้นตอนต่อไปตัวควบคุมจะทำการรอเวลาให้ตัวส่งทำการส่งข้อมูลลงบนบัสข้อมูลให้เสร็จ และจะทำการตรวจสอบด้วยว่าเกินเวลาที่กำหนดหรือไม่ หากเกินเวลาที่กำหนดก็จะทำการออกจากขบวนการแฮนด์เช็ค หากยังไม่เกินก็จะรอจนหมดเวลาหรือจนกว่าสัญญาณ DAV จะเป็น Low (สถานะที่ 4) เมื่อสัญญาณ DAV มีลอจิกเป็น Low ตัวควบคุมก็จะตอบรับโดยการทำให้สัญญาณ NRFD มีค่าเป็น Low(สถานะที่ 5) เพื่อบอกให้ตัวส่งทราบว่าตัวควบคุมพร้อมที่จะเริ่มทำการเก็บข้อมูลแล้ว (สถานะที่ 6) หลังจากนั้นตัวควบคุมจะทำการตรวจสอบสัญญาณ EOI ว่า ข้อมูลที่ตัวส่งทำการส่งมานั้นหมดหรือยัง โดยตัวส่งจะทำการเปลี่ยน

สัญญาณ EOI ให้มีลอจิกเป็น Low เมื่อข้อมูลไบต์สุดท้ายถูกส่งลงไปบนบัสข้อมูล (สถานะที่ 7) หากสัญญาณ EOI ยังคงเป็น High ตัวควบคุมจะทำการเก็บข้อมูลในบัสข้อมูลต่อไป (สถานะที่ 8) และเมื่อตัวควบคุมได้ทำการเก็บข้อมูลเสร็จเรียบร้อยแล้ว ตัวควบคุมจะเปลี่ยนลอจิกของสัญญาณ NDAC ให้เป็น High (สถานะที่ 9) เมื่อตัวควบคุมทำให้สัญญาณ NDAC มีลอจิกเป็น High แล้ว ตัวส่งจะทำการลบข้อมูลบนบัสข้อมูลออกโดยการเปลี่ยนสัญญาณ DAV ให้มีค่าเป็น High (สถานะที่ 10) ตัวควบคุมก็จะทำการนำข้อมูลที่ได้ไปใช้งานและเปลี่ยนสัญญาณ NDAC ให้มีค่าเป็น Low (สถานะที่ 11) เป็นการสิ้นสุดขบวนการแฮนด์เช็ค

คำสั่งใช้งานของ IEEE 488 (GPIB)

การควบคุมและกำหนดฟังก์ชันการทำงานให้แก่อุปกรณ์เครื่องมือวัดในระบบ GPIB นั้น ตัวควบคุมจะเป็นตัวกำหนด โดยการส่งรหัสคำสั่งไปยังตัวอุปกรณ์โดยผ่านทางบัสของระบบ คำสั่งสำหรับการกำหนดการทำงานต่างๆ ตามมาตรฐานของ IEEE 488 มีอยู่ 128 คำสั่ง แบ่งได้เป็น 2 กลุ่มคำสั่งใหญ่ๆ คือ กลุ่มคำสั่งหลัก (Primary Command Group) และกลุ่มคำสั่งรอง (Secondary Command Group) โดยกลุ่มคำสั่งหลักประกอบด้วย 4 กลุ่มคำสั่ง คือ กลุ่มคำสั่งเจาะจงจุดหมาย (Addressed Command Group), กลุ่มคำสั่งครอบคลุม (Universal Command Group), กลุ่มคำสั่งกำหนดอุปกรณ์ตัวรับ (Listen Address Group) และกลุ่มคำสั่งกำหนดอุปกรณ์ตัวส่ง (Talk Address Group) ดังแสดงในตารางที่ 2 รหัสที่ใช้ในระบบ IEEE 488 นั้นสามารถที่จะใช้ร่วมกันได้ทั้งรหัสข้อมูลและรหัสคำสั่ง นั่นคือ ข้อมูลที่เหมือนกันมีความหมายได้ 2 อย่าง คือ เมื่อสัญญาณ ATN เป็น Low ข้อมูลที่อยู่ในบัสข้อมูลจะหมายถึงรหัสคำสั่ง แต่ถ้าสัญญาณ ATN เป็น High ข้อมูลที่อยู่ในบัสข้อมูลจะหมายถึงข้อมูลที่เป็นรหัส ASCII ดังแสดงในตารางที่ 1 เช่นกัน

1. กลุ่มคำสั่งเจาะจงจุดมุ่งหมาย (Address Command Group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ที่เป็นตัวส่งหรือตัวรับที่กำหนดไว้ล่วงหน้าแล้ว ประกอบด้วย

GTL (Go To Local) สั่งให้อุปกรณ์กลับไปสู่สถานะควบคุมด้วยปุ่มปรับที่หน้าปัทม์ตามปกติ

SDC (Selected Device Clear) สั่งให้อุปกรณ์กลับไปสู่สถานะเริ่มต้น

PPC (Parallel Poll Configure) เป็นคำสั่งสำหรับทำการจัดสรรสายสัญญาณของการกระทำกระบวนการตรวจสอบสภาพอุปกรณ์โดยวิธีขนาน โดยใช้งานร่วมกับกลุ่มคำสั่งรอง

GET (Group Execute Trigger) ใช้สั่งเริ่มต้นการทำงานของอุปกรณ์ทีละหลาย ๆ ตัว

TCT (Take Control) กำหนดให้อุปกรณ์ตัวส่งทำหน้าที่เป็นตัวควบคุม

2. กลุ่มคำสั่งครอบคลุม (Universal Command Group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ทุกตัวที่ต่ออยู่ในระบบ ประกอบด้วย

LLO (Local Lockout) สั่งให้อุปกรณ์ล็อคอยู่ในสถานะควบคุมด้วยปุ่มหน้าปัทม์ตามปกติ

DCL (Device Clear) สั่งให้อุปกรณ์ทุกตัวกลับไปสู่สถานะเริ่มต้น

PPU (Parallel Poll Configure) ใช้ยกเลิกกระบวนการตรวจสอบสภาพแบบขนานทั้งหมด

SPE (Serail Poll Enable) เป็นการเปลี่ยนโหมดการตรวจสอบสภาพเป็นแบบอนุกรม โดยในโหมดนี้จะเป็นการส่งสถานะของเครื่องแทนการส่งข้อมูล

SPD (Serail Poll Disable) ยกเลิกกระบวนการตรวจสอบสภาพแบบอนุกรม

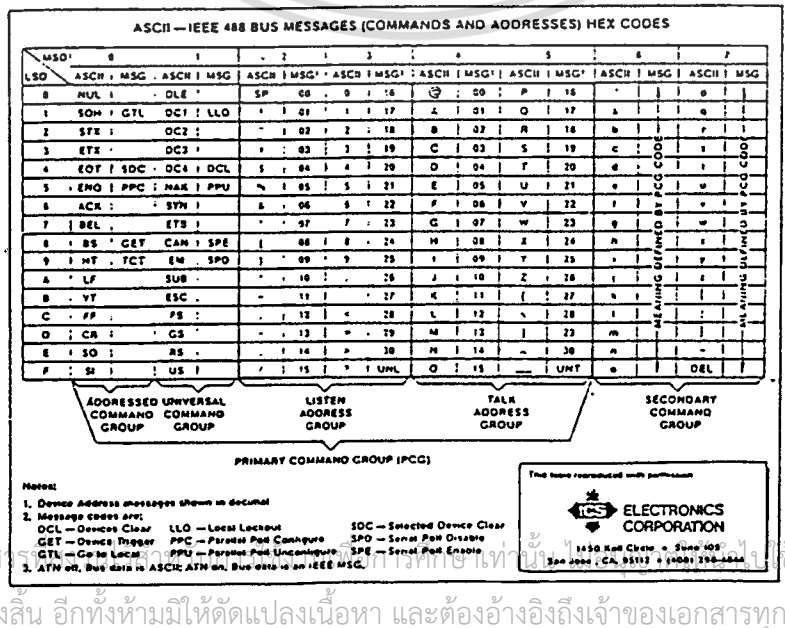
3. กลุ่มคำสั่งกำหนดอุปกรณ์ตัวรับ (Listen Address Group) เป็นคำสั่งสำหรับกำหนดอุปกรณ์เป็นตัวรับตามรหัสหมายเลข 0 ถึง 30 และมีคำสั่ง UNL (Unlistener) สำหรับใช้ยกเลิก

4. กลุ่มคำสั่งกำหนดอุปกรณ์ตัวส่ง (Talk Address Group) ใช้สำหรับกำหนดอุปกรณ์เป็นตัวส่งตามรหัสหมายเลข 0 ถึง 30 และมีคำสั่ง UNT (Untalker) ใช้สำหรับยกเลิกเช่นกัน

5. กลุ่มคำสั่งรอง (Secondary Command Group) เป็นคำสั่งที่กำหนดรายละเอียดย่อยของอุปกรณ์แต่ละตัวที่ต่ออยู่ในระบบ ให้มีการทำงานตามจุดประสงค์การใช้งานของอุปกรณ์นั้น คำสั่งรองนี้จะใช้ตามหลังคำสั่งหลัก คือ จะใช้หลังจากที่อุปกรณ์ต่างๆ ถูกกำหนดไว้ในระบบเรียบร้อยแล้ว

ตารางที่ 2

แสดงคำสั่งมาตรฐาน IEEE-488 และรหัส ASCII



ASCII—IEEE 488 BUS MESSAGES (COMMANDS AND ADDRESSES) HEX CODES

MSD → / LSD ↓	0		1		2		3		4		5		6		7	
	ASCII	MSG	ASCII	MSG	ASCII	MSG¹	ASCII	MSG¹	ASCII	MSG¹	ASCII	MSG¹	ASCII	MSG	ASCII	MSG
0	NUL		DLE		SP	GC	0	16	@	00	P	16	`		p	
1	SOH	GTL	DC1	LLO	!	01	1	17	A	01	Q	17	a		q	
2	STX		DC2		"	02	2	18	B	02	R	18	b		r	
3	ETX		DC3		#	03	3	19	C	03	S	19	c		s	
4	EOT	SDC	DC4	DCL	$	04	4	20	D	04	T	20	d		t	
5	ENQ	PPC	NAK	PPU	%	05	5	21	E	05	U	21	e		u	
6	ACK		SYN		&	06	6	22	F	06	V	22	f		v	
7	BEL		ETB		'	07	7	23	G	07	W	23	g		w	
8	BS	GET	CAN	SPE	(	08	8	24	H	08	X	24	h		x	
9	HT	TCT	EM	SPD	)	09	9	25	I	09	Y	25	i		y	
A	LF		SUB		*	10	:	26	J	10	Z	26	j		z	
B	VT		ESC		+	11	;	27	K	11	[	27	k		{	
C	FF		FS		,	12	<	28	L	12	\	28	l		|	
D	CR		GS		-	13	=	29	M	12	]	29	m		}	
E	SO		RS		.	14	>	30	N	14	^	30	n		~	
F	SI		US		/	15	?	UNL	O	15	_	UNT	o		DEL	

MEANING DEFINED BY PCG CODE

ADDRESSED COMMAND GROUP | UNIVERSAL COMMAND GROUP | LISTEN ADDRESS GROUP | TALK ADDRESS GROUP | SECONDARY COMMAND GROUP

PRIMARY COMMAND GROUP (PCG)

Notes:
1. Device Address messages shown in decimal
2. Message codes are:
GCL — Device Clear
GET — Device Trigger
GTL — Go to Local
LLO — Local Lockout
PPC — Parallel Poll Configure
PPU — Parallel Poll Unconfigure
SDC — Selected Device Clear
SPD — Serial Poll Disable
SPE — Serial Poll Enable
3. ATN on, Bus data is ASCII; ATN on, Bus data is an IEEE MSG.

This table reproduced with permission
ICS ELECTRONICS CORPORATION
1450 Koll Circle • Suite 103
San Jose, CA 95112 • (408) 294-4844

คำสั่งต่างๆ ซึ่งใช้ในการกำหนดสภาวะการทำงานของอุปกรณ์แต่ละสภาวะจะถูกกำหนดและมีจุดประสงค์ดังนี้ คือ

Device Clear ทำให้อุปกรณ์กลับคืนสู่สภาวะเริ่มต้น ซึ่งเป็นสภาวะที่ยังไม่มีการกำหนดฟังก์ชันใดๆ สภาวะเริ่มต้นนี้จะแตกต่างกันไปแล้วแต่ว่าอุปกรณ์นั้นได้ออกแบบมาไว้อย่างไร Device Clear แบ่งออกได้เป็น 2 ลักษณะ คือ

DCL (Device Clear) ทำการเคลียร์อุปกรณ์ทุกตัวที่ต่ออยู่

SDC (Select Device Clear) ทำการเคลียร์เฉพาะเจาะจงอุปกรณ์ตัวใดตัวหนึ่ง

แต่ว่าในการเคลียร์อุปกรณ์ให้อยู่ในสภาวะเริ่มต้นไม่ได้หมายความว่าอินเตอร์เฟสฟังก์ชันของระบบ GPIB จะถูกเคลียร์ให้กลับไปสู่สถานะเริ่มต้นด้วย เป็นเพียงแค่การเคลียร์ตัวอุปกรณ์เท่านั้น อินเตอร์เฟสฟังก์ชันก็คือ สภาพการอินเตอร์เฟสที่ได้กำหนดเอาไว้ในระบบประกอบด้วยฟังก์ชันต่างๆ ดังตารางที่ 3

ตารางที่ 3

แสดงฟังก์ชันการอินเตอร์เฟสในระบบ GPIB

ฟังก์ชัน	สัญลักษณ์	กลับสู่สภาวะเริ่มต้นด้วยคำสั่ง IFC
Source Handshake	SH	ได้
Acceptor Handshake	AH	ได้
Talker หรือ Enlarge Talker	T หรือ TE	ได้
Listener หรือ Enlarge Listener	L หรือ LE	ได้
Service Request	SR	ไม่ได้
Remot / Local	RL	ไม่ได้
Parallel Poll	PP	ไม่ได้
Device Clear	DC	ได้
Device Trigger	DT	ได้
Controller	C	ได้

Interface Clear จะใช้ในการเคลียร์สภาพการอินเตอร์เฟสให้อยู่ในสภาวะเริ่มต้น ซึ่งจะทำให้ทุกฟังก์ชันถูกยกเลิกไป ยกเว้น SR (Service Request), RL (Remote/Local) และ PP (Parallel Poll)

Remote เป็นการกำหนดให้อุปกรณ์ที่ต่ออยู่ในระบบถูกควบคุมโดยอุปกรณ์ตัวอื่นหรือตัวควบคุมระบบ ทำให้ไม่สามารถที่จะควบคุมอุปกรณ์จากปุ่มหน้าปัทม์ของเครื่องได้

Local เป็นการกำหนดให้อุปกรณ์ที่ต่ออยู่ในระบบสามารถควบคุมได้จากปุ่มหน้าปัทม์ของอุปกรณ์ตามปกติ

การทำงานของ GPIB ในสถานะ Remote และ Local มี 4 ลักษณะ คือ

1. LOCS เป็นโหมด local ที่อยู่ในสภาพการควบคุมจากหน้าปัทม์ตามปกติ ซึ่งอุปกรณ์จะอยู่ในสภาวะนี้เมื่อเริ่มเปิดสวิทช์ของอุปกรณ์ หรือสัญญาณ REN มีลอจิกเป็น High หรือเมื่ออุปกรณ์ได้รับคำสั่ง GTL (Go To Local)

2. REMS เป็นโหมด Remote หมายถึงการตัดการควบคุมอุปกรณ์ออกจากการควบคุมด้วยปุ่มบนหน้าปัทม์ โดยถูกควบคุมจากอุปกรณ์ตัวอื่นที่ทำหน้าที่เป็นตัวควบคุม สภาวะการ Remote จะเกิดขึ้นเมื่อสัญญาณ REN (Remote Enable) มีลอจิกเป็น Low และจะถูกล็อคไว้ที่สถานะนี้จนกว่าสวิทช์ Local ที่ตัวอุปกรณ์จะถูกกด เพื่อที่จะทำให้อุปกรณ์กลับสู่สถานะ Local

3. RWLS เป็นสภาวะการ Remote ที่ถูกล็อคไว้เช่นกัน แต่จะตัดการควบคุมของสวิทช์ Local ที่ตัวอุปกรณ์ออกไป สภาวะการ Remote แบบ RWLS นี้มีความสำคัญสูงกว่าสภาวะการ Remote แบบ REMS และสามารถที่จะยกเลิกสภาวะดังกล่าวนี้ด้วยคำสั่ง LLO (Local Lock Out)

4. LWLS มีสภาวะเช่นเดียวกับ Local แต่ต่างกันที่สภาวะLWLS นี้ เมื่อได้รับคำสั่งกำหนดให้เป็นอุปกรณ์ตัวรับ ก็จะทำการเปลี่ยนไปใสภาวะการ Remote แบบที่ถูกล็อคทันที การจะเข้าสู่สภาวะแบบ LWLS มีอยู่ 2 วิธี คือ

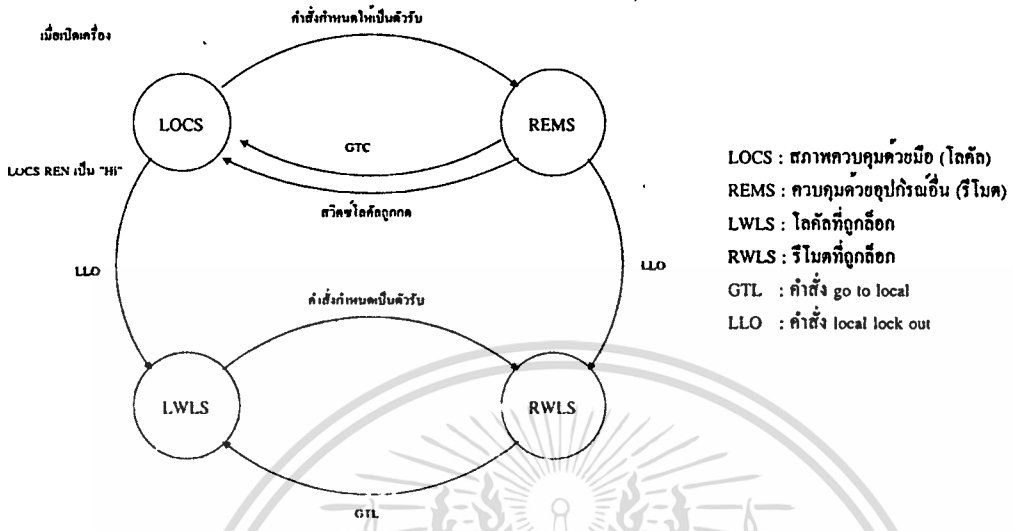
1. เมื่ออยู่ในสภาวะ Local แบบธรรมดา (LOCS) แล้วได้รับคำสั่ง LLO (Local Lock Out)

2. เมื่ออยู่ในสภาวะ REMS แล้วได้รับคำสั่ง GTL

การเปลี่ยนแปลงสภาวะของการ Remote หรือ Local ทั้ง 4 ลักษณะแสดงดังภาพที่ 9 เมื่อสัญญาณ REN มีลอจิกเป็น High อุปกรณ์ก็จะอยู่ในสภาวะ Local ทันทีไม่ว่าสถานะเดิมจะเป็นอะไรก็ตาม แต่เมื่อสัญญาณ REN เป็น Low แล้ว หากไม่มีคำสั่งกำหนดอุปกรณ์ตัวรับ หรือคำสั่ง LLO เข้ามาอุปกรณ์ก็ยังคงสถานะ Local อยู่เช่นเดิม

ในการรับส่งข้อมูลในระบบ GPIB จะมีอัลกอริธึมดังแสดงในแผนภูมิที่ 10

ภาพที่ 9



แสดงการเปลี่ยนแปลงสภาวะของโหมด Remote และ Local ทั้ง 4 ลักษณะ

การขอบริการและการตรวจสอบ (Service Request and Polling)

เมื่อตัวควบคุมได้รับสัญญาณ SRQ ที่ไลอจิกเป็น Low ก็จะส่งสัญญาณให้อุปกรณ์ทำการส่งข้อมูลแสดงสถานะการทำงาน ซึ่งมีอยู่ 2 แบบ คือ

1. การตรวจสอบแบบอนุกรม (Serail Poll) ซึ่งมีขั้นตอน คือ

1.1 สัญญาณ ATN ถูกเปลี่ยนให้เป็น Low หลังจากที่สัญญาณ SRQ ถูกเปลี่ยนให้เป็น Low

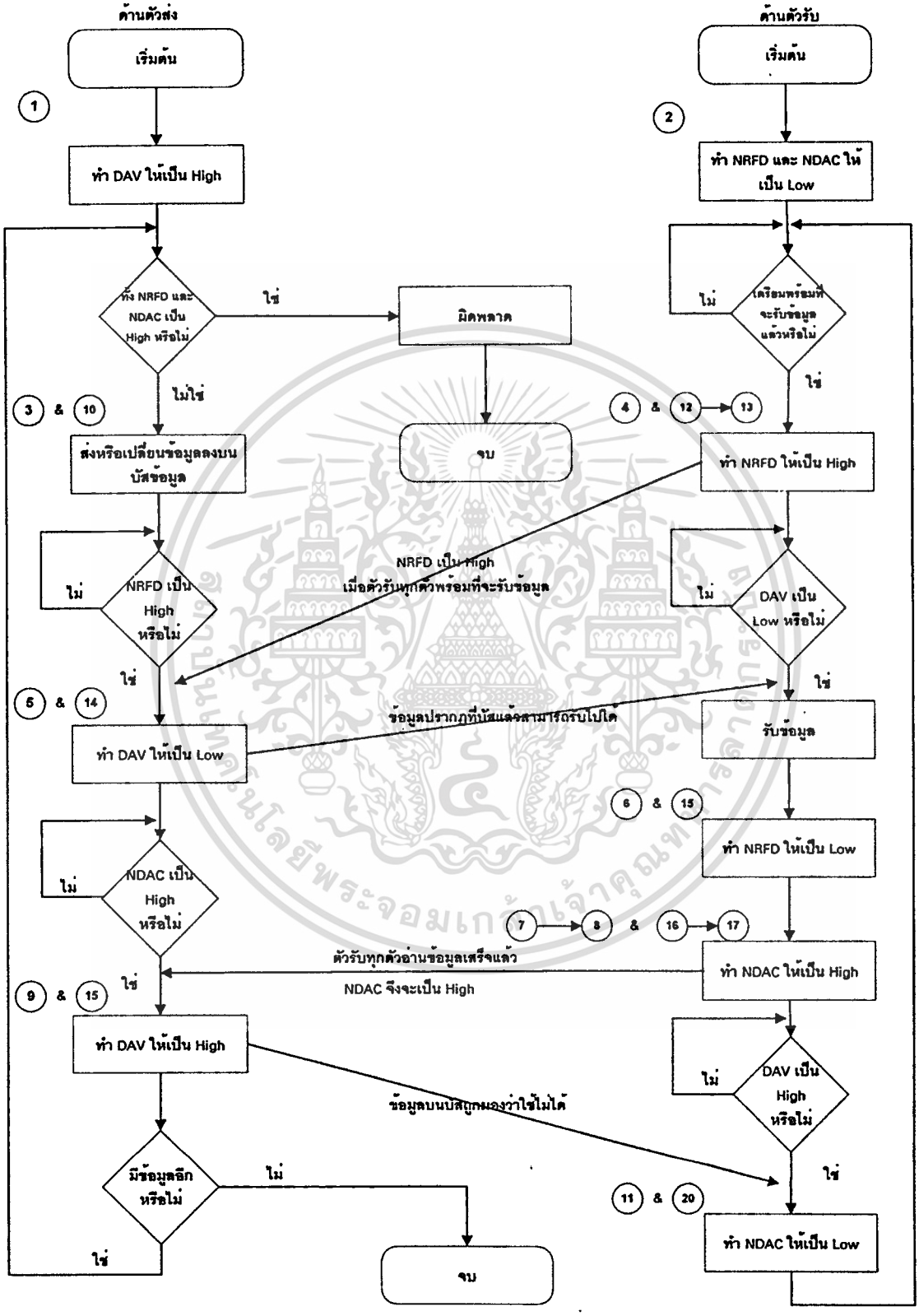
1.2 คำสั่ง UNL (Unlistener) ถูกส่งไปยังอุปกรณ์ในระบบ

1.3 ตัวควบคุมจะแจ้งแอดเดรสของตัวรับ และกำหนดแอดเดรสตัวส่งของอุปกรณ์ที่จะตรวจสอบไปที่บัส

1.4 ทำการส่งคำสั่ง SPE (Serail Poll Enable) และสายสัญญาณ ATN ถูกเปลี่ยนให้มีลอจิกเป็น High ซึ่งอุปกรณ์ที่ถูกเรียกจะทำการส่งข้อมูลแสดงสถานะออกมา 1 ไบต์ โดยที่บิทที่ 7 จะเป็นตัวชี้ว่า อุปกรณ์นั้นเป็นตัวขอบริการหรือไม่ โดยจะมีค่าเป็น Low เมื่อขอบริการ ส่วนบิทอื่นๆ จะใช้บอกข้อมูลอื่นๆ ซึ่งไม่ได้กำหนดเฉพาะ

1.5 สายสัญญาณ ATN จะถูกเปลี่ยนให้มีลอจิกเป็น Low อีกครั้งหนึ่ง เพื่อส่งคำสั่งยกเลิกการตรวจสอบ คือ SPD (Serial Poll Disable)

แผนภูมิที่ 10



แสดงขั้นตอนการรับส่งข้อมูลในระบบ GPIB

1.6 จากนั้นคำสั่ง UNT (Untalker) จะถูกส่งไปยังอุปกรณ์เพื่อยกเลิกการเป็นตัวส่ง ซึ่งหากสายสัญญาณ SRQ ยังคงเป็นลอจิก Low อยู่ ก็จะมีการตรวจสอบไปยังอุปกรณ์ตัวอื่นต่อไปตามขั้นตอนเดิม

2. การตรวจสอบแบบขนาน (Parallel Poll) สามารถทำได้เร็วกว่าการตรวจสอบแบบอนุกรม ทั้งนี้เพราะว่าสามารถอ่านข้อมูลเพียงไบต์เดียวก็สามารถรู้ได้ว่า อุปกรณ์ตัวใดเป็นผู้ขอบริการ

รูปแบบของข้อมูล

การส่งข้อมูลของ GPIB จำเป็นต้องมีรูปแบบที่แน่นอน เพื่อให้การส่งและรับข้อมูลเป็นไปอย่างถูกต้อง โดยทั่วไปข้อมูลจากอุปกรณ์แบ่งออกได้เป็น 3 ส่วน คือ ส่วนหัว (HR), ส่วนข้อมูลตัวเลข (NR) และส่วนสัญลักษณ์ (SR) ดังแสดงในภาพที่ 11

ภาพที่ 11

แสดงส่วนประกอบของข้อมูลจากอุปกรณ์

ส่วนหัว (HR) ประกอบด้วยตัวอักษรภาษาอังกฤษตัวพิมพ์ใหญ่กับช่องว่างที่เป็นตัวเว้นวรรค (Space, แทนด้วยเครื่องหมาย Δ) เป็นตัวบอกชนิดข้อมูล ปกติจะมีตัวอักษร 1 - 3 ตัว แสดงดังภาพที่ 12

ส่วนข้อมูลตัวเลข (NR) เป็นเนื้อหาของข้อมูล ซึ่งใช้แทนค่าตัวเลข มีอยู่ 3 แบบ คือ NR1, NR2 และ NR3 ดังแสดงในภาพที่ 13 และในส่วนของข้อมูลตัวเลขยังอาจจะมีตัวอักษรแสดงหน่วยตามมาอีกด้วย

แบบ NR1 ตัวเลขจำนวนเต็ม

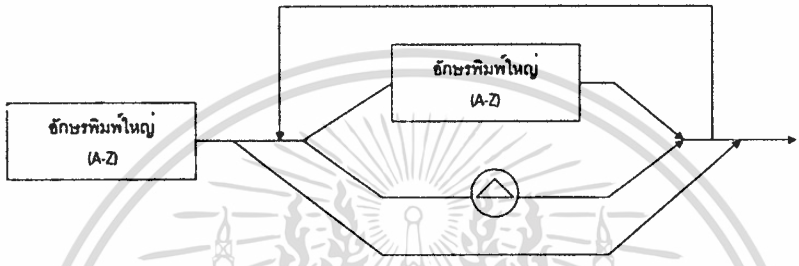
แบบ NR2 ตัวเลขจำนวนจริง

แบบ NR3 ตัวเลขยกกำลัง

หมายเหตุ 1 จุดทศนิยมนั้นตัวส่งต้องส่งออกไปด้วยทุกครั้ง และตัวรับก็ต้องถือว่ามีตำแหน่งนี้อยู่

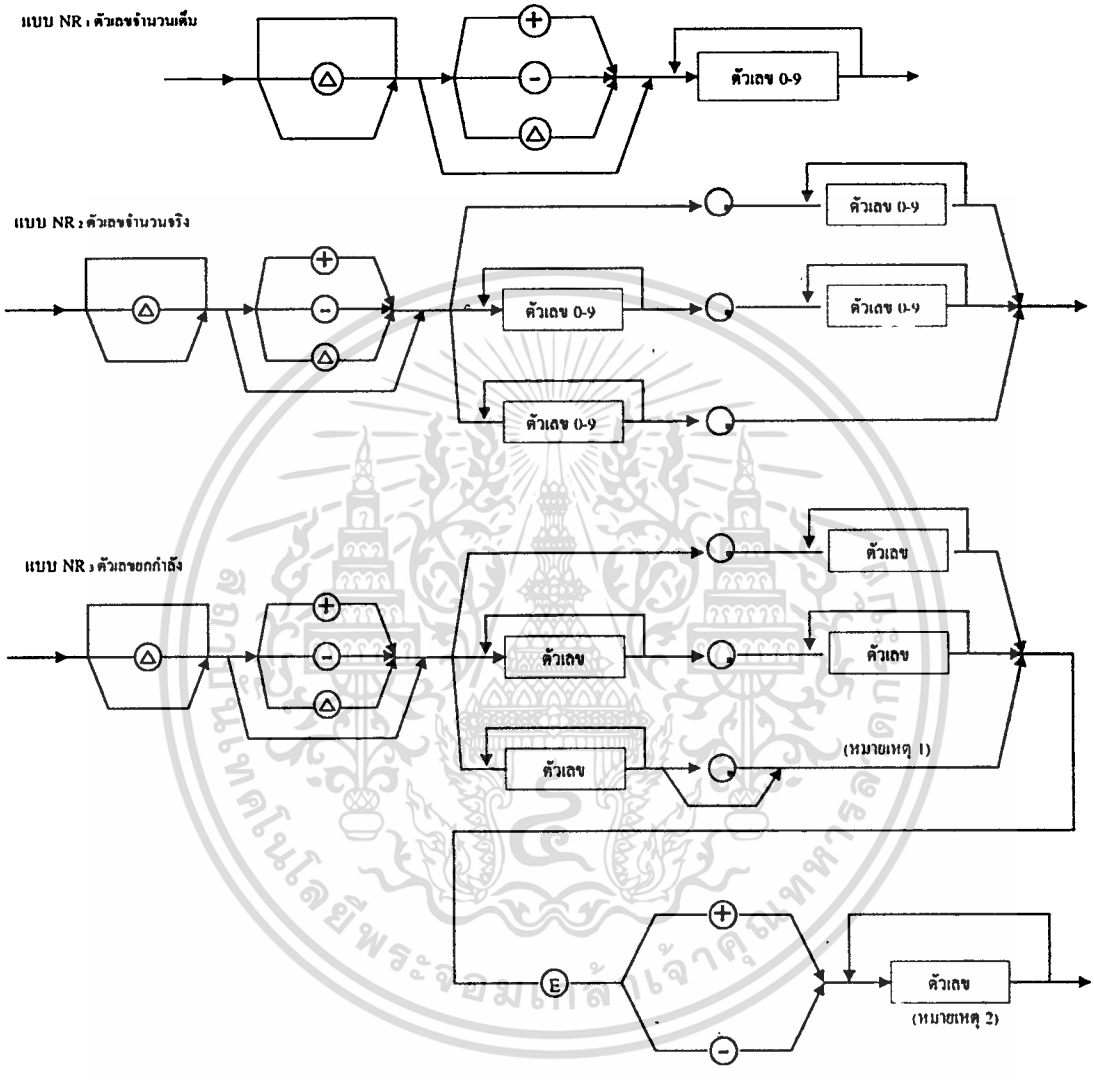
หมายเหตุ 2 เลขชี้กำลังขนาด 2 หลักเป็นมาตรฐาน แต่สามารถเลือกใช้เพียง 1 หลัก หรือทั้ง 3 หลักก็ได้

ภาพที่ 12



แสดงรายละเอียดของข้อมูลส่วนหัว (HR)

ภาพที่ 13

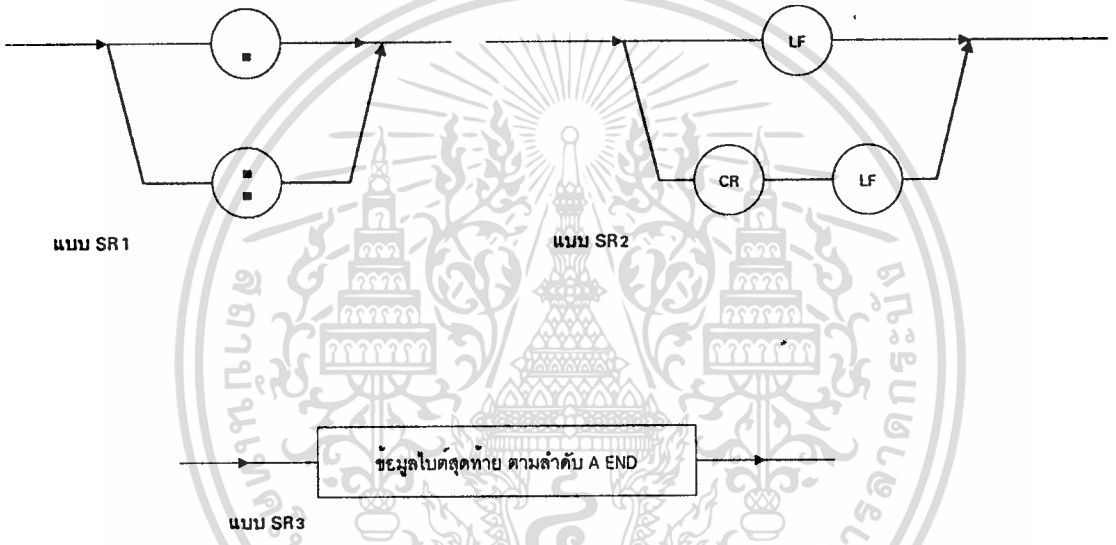


แสดงรายละเอียดของส่วนข้อมูลตัวเลข (NR)

<u>ส่วนสัญลักษณ์</u> (SR) เป็นสัญลักษณ์ที่ใช้ในการแบ่งข้อมูลแต่ละชุด ใช้แสดงการสิ้นสุดของข้อมูล แบ่งออกเป็น 3 แบบ คือ SR1, SR2 และ SR3 ดังแสดงในภาพที่ 14 โดย SR1 แสดง

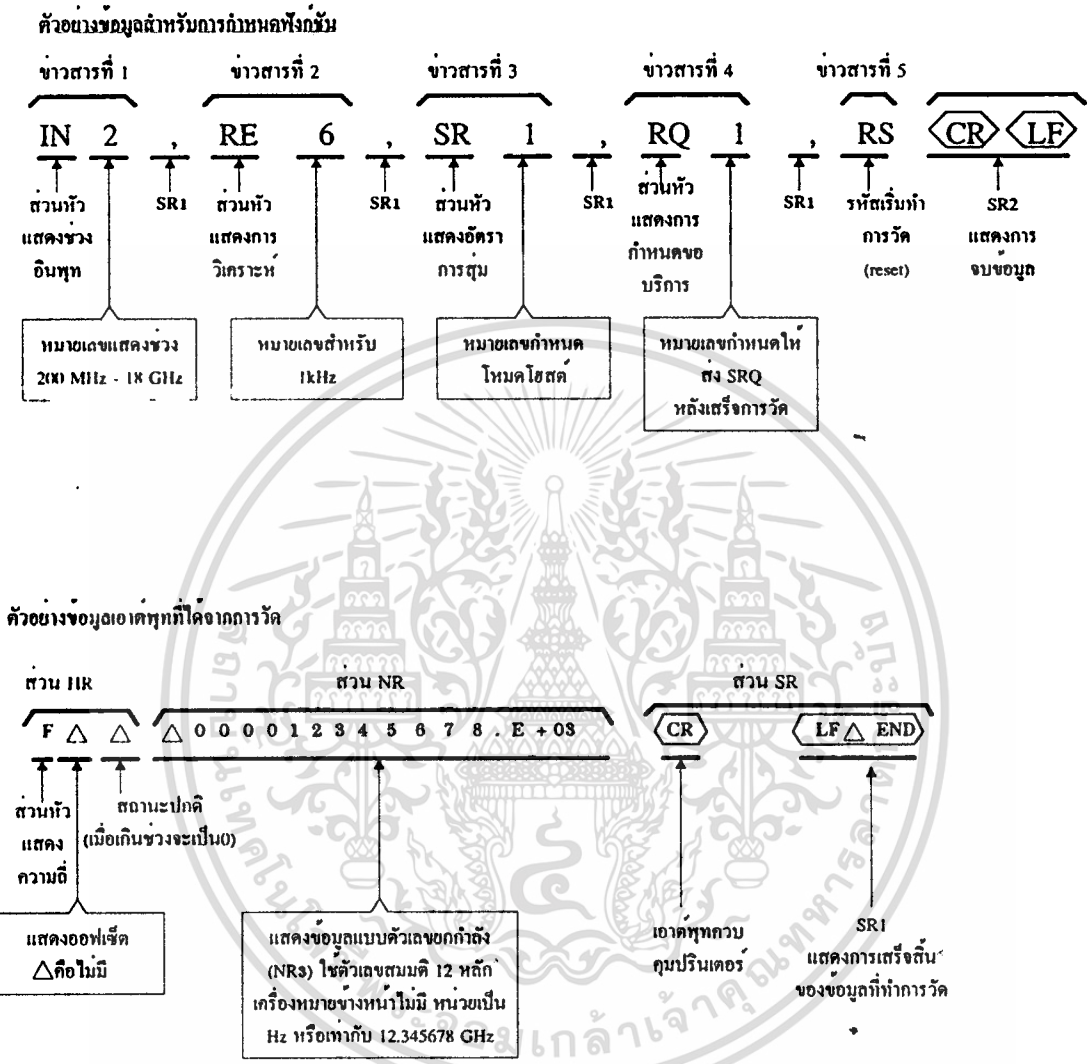
การต่อเนื่องของข้อมูล (ข้อมูลยังมีต่อ) SR2 และ SR3 แสดงการสิ้นสุดของข้อมูล แต่ SR3 แสดงถึงการบอกการเสร็จสิ้นของข้อมูลจากการวัด และภาพที่ 15 แสดงตัวอย่างของข้อมูลสำหรับการกำหนดฟังก์ชันให้เครื่องวัดความถี่และข้อมูลความถี่ที่วัดได้

ภาพที่ 14



แสดงรายละเอียดของส่วนสัญลักษณ์ (SR)

ภาพที่ 15



แสดงตัวอย่างข้อมูลสำหรับการกำหนดฟังก์ชันให้เครื่องวัดความถี่และข้อมูลความถี่ที่วัดได้

บทที่ 3
สถาปัตยกรรมของระบบ

กล่าวนำ

สถาปัตยกรรมของระบบเป็นการกล่าวถึงโครงสร้างของระบบเครื่องมือวัดที่ทำการศึกษาและฟังก์ชันคำสั่งในการทำงานของอุปกรณ์ที่ต่ออยู่บนระบบ โดยระบบเครื่องมือวัดนี้ประกอบด้วยตัวควบคุมระบบซึ่งใช้ไมโครคอมพิวเตอร์ทำหน้าที่ดังกล่าว และอุปกรณ์เครื่องมือพื้นฐานทางด้านอิเลกทรอนิกส์ 4 อุปกรณ์ คือ

1. Power Supply ทำหน้าที่เป็นแหล่งจ่ายไฟให้กับวงจร
2. Function Generator ทำหน้าที่สร้างสัญญาณรูปแบบต่างๆ ป้อนให้แก่วงจรที่ทดสอบ
3. Digital Multimeter ทำหน้าที่วัดค่าพารามิเตอร์ต่างๆ ทางอิเลกทรอนิกส์
4. Storage Oscilloscope ทำหน้าที่วัดสัญญาณต่างๆ ในวงจร และทำการบันทึกสัญญาณเก็บไว้ เพื่อที่จะได้นำสัญญาณที่บันทึกไว้มาวิเคราะห์ต่อไป

สำหรับการเชื่อมต่ออุปกรณ์ในระบบนั้น เป็นส่วนประกอบที่สำคัญอย่างหนึ่งที่จะช่วยให้ระบบมีประสิทธิภาพในการทำงานที่ดี

อุปกรณ์ภายในระบบ

1 ตัวควบคุมระบบ

ตัวควบคุมระบบตามมาตรฐาน IEEE-488 เป็นหัวใจสำคัญของระบบ เนื่องจากจะเป็นตัวควบคุมการติดต่อสื่อสารระหว่างตัวรับ ตัวส่งและตัวควบคุมได้อย่างถูกต้องและรวดเร็ว ซึ่งในระบบที่ทำการศึกษานั้นได้ใช้ไมโครคอมพิวเตอร์เป็นตัวควบคุม ข้อดีคือ สะดวกในการอินเตอร์เฟสเนื่องจากมีสล็อตไว้ให้เลือกใช้อยู่แล้ว การพัฒนาโปรแกรมทำได้สะดวกรวดเร็วและยังสามารถบันทึกค่าข้อมูลที่ได้จากการวัดในระบบเก็บเอาไว้ได้ เพื่อที่จะนำค่าที่บันทึกไว้ไปใช้งานต่อไป

2 อุปกรณ์ที่ต่ออยู่ในระบบ

ระบบเครื่องมือวัดตามมาตรฐาน IEEE-488 ที่ทำการศึกษานั้น นอกจากจะมีตัวควบคุมระบบแล้ว ยังมีอุปกรณ์ที่ต่ออยู่ในระบบซึ่งได้แก่อุปกรณ์ที่เป็นเครื่องมือพื้นฐานทางอิเลกทรอนิกส์

ที่ใช้สำหรับการทดลองทางอิเลกทรอนิกส์ โดยอุปกรณ์เหล่านี้นอกจากจะสามารถควบคุมได้ผ่านหน้าปัทม์แล้ว ยังสามารถควบคุมได้โดยผ่านระบบบัสตามมาตรฐาน IEEE-488 หรือ GPIB ได้ด้วย ซึ่งในระบบจะประกอบด้วย

1. Programmable Power Supply ของบริษัท HAMEG รุ่น HM 8142 ทำหน้าที่เป็นแหล่งจ่ายไฟฟ้ากระแสตรง เพื่อใช้เป็นไฟเลี้ยงให้แก่วงจรที่จะทำการทดลอง โดยมีแหล่งจ่ายไฟอยู่ 2 ตัว และมีคำสั่งควบคุมฟังก์ชันในการทำงานผ่านระบบบัสแบบ IEEE-488 คือ

คำสั่ง : RM1 / RM0

รูปแบบ : RM1

ฟังก์ชัน : สั่งให้ Power Supply ทำงานในโหมดรีโมท โดยไม่สามารถควบคุมผ่านหน้าปัทม์ได้ แต่สามารถควบคุมการทำงานได้โดยผ่านระบบบัสแบบ IEEE-488 เท่านั้น และสามารถออกจากโหมดรีโมทได้โดยการส่งคำสั่ง RM0 หรือการกดปุ่ม LOCAL ที่หน้าปัทม์ทม์ของเครื่อง

รูปแบบ : RM0

ฟังก์ชัน : ทำการยกเลิกโหมดรีโมทให้เครื่องกลับสู่สถานะ LOCAL (สามารถควบคุมการทำงานของเครื่องได้โดยผ่านปุ่มหน้าปัทม์)

ข้อสังเกต : คำสั่ง RM0 จะมีผลต่อคำสั่ง LK1

คำสั่ง : MX1 / MX0

รูปแบบ : MX1

ฟังก์ชัน : เปลี่ยนโหมดรีโมทไปเป็นโหมดแบบผสม ในโหมดนี้สามารถที่จะควบคุมการทำงานของเครื่องได้ทั้งโดยการส่งคำสั่งผ่านระบบบัสแบบ IEEE-488 และการควบคุมผ่านทางหน้าปัทม์ของเครื่อง

รูปแบบ : MX0

ฟังก์ชัน : ยกเลิกโหมดแบบผสมและกลับเข้าสู่สถานะปกติที่ควบคุม โดยผ่านระบบบัสแบบ IEEE-488

คำสั่ง : LK1 / LK0

รูปแบบ : LK1

ฟังก์ชัน : สั่งให้เครื่องเข้าสู่โหมดรีโมท โดยไม่สามารถกดปุ่ม LOCAL เพื่อที่จะให้เครื่องกลับเข้าสู่โหมด LOCAL ได้

รูปแบบ : LK0

ฟังก์ชัน : สั่งให้เครื่องออกจากโหมดรีโมทที่ไม่สามารถกดปุ่ม LOCAL ทำให้สามารถกดปุ่ม LOCAL เพื่อที่จะสามารถควบคุมการทำงานผ่านหน้าปัทม์ของเครื่องได้

ข้อสังเกต : โหมดรีโมทที่ไม่สามารถกดปุ่ม LOCAL นี้สามารถถูกยกเลิกได้โดยคำสั่ง RM0

คำสั่ง : SU1 และ SU2

รูปแบบ : SU1 VV.mV.mV หรือ SU2 01.34

ฟังก์ชัน : ตั้งค่าศักดาไฟฟ้าของแหล่งจ่ายที่ 1 หรือ 2

คำสั่ง : SI1 และ SI2

รูปแบบ : SI1 A.mAmAmA หรือ SI 0.123

ฟังก์ชัน : ตั้งค่ากระแสไฟฟ้าของแหล่งจ่ายตัวที่ 1 หรือ 2 เพื่อจำกัดค่ากระแสไฟฟ้า

คำสั่ง : RU1 และ RU2

รูปแบบ : RU1 หรือ RU2

ฟังก์ชัน : ค่าศักดาไฟฟ้าที่ส่งกลับโดยเครื่อง จะเป็นค่าศักดาไฟฟ้าที่ถูกกำหนดค่าโดยคำสั่งกำหนดค่า (SET)

ข้อสังเกต : ใช้คำสั่ง MUX เพื่อที่จะถามค่ากระแสไฟฟ้าจริงที่วัดได้จากเอาท์พุท

คำสั่ง : RI1 และ RI2

รูปแบบ : RI1 หรือ RI2

ค่าที่ส่งกลับ : I1_1.000A หรือ I2_0.012A

ฟังก์ชัน : ค่ากระแสไฟฟ้าที่ส่งกลับโดยเครื่อง จะเป็นค่ากระแสไฟฟ้าที่ถูกกำหนดค่าโดยคำสั่งจำกัดค่ากระแสไฟฟ้า (SI1 หรือ SI2)

ข้อสังเกต : ใช้คำสั่ง MIX เพื่อถามค่ากระแสไฟฟ้าที่วัดได้จากเอาท์พุทจริง

คำสั่ง : MU1 และ MU2

รูปแบบ : MU1 หรือ MU2

ค่าที่ส่งกลับ : U1:12.34V หรือ U2:12.24V

ฟังก์ชัน : ค่าศักดาไฟฟ้าที่ส่งกลับโดยเครื่อง ซึ่งเป็นค่าที่วัดได้จากเอาท์พุทจริง

ข้อสังเกต : คำสั่ง RUX ใช้เพื่อถามถึงค่าศักดาไฟฟ้าที่ทำการกำหนด

คำสั่ง : MI1 และ MI2

รูปแบบ : MI1 หรือ MI2

ค่าที่ส่งกลับ : I1=+1.000A หรือ I2=-0.123A

ฟังก์ชัน : ค่ากระแสไฟฟ้าที่ส่งกลับโดยเครื่องเป็นค่ากระแสไฟฟ้าที่วัดได้

ข้อสังเกต : คำสั่ง RIX ใช้เพื่อถามถึงค่ากระแสไฟฟ้าที่จำกัดเอาไว้ ถ้าเอาท์พุทถูกปิดไว้ ค่าที่ส่งกลับจะเป็น I1=_1.000A

คำสั่ง : TRU

รูปแบบ : TRU

ฟังก์ชัน : กำนดค่าศักดาไฟฟ้าของแหล่งจ่ายตัวที่ 1 และ 2 ให้มีค่าเท่ากันตามต้องการ

คำสั่ง : TRI

รูปแบบ : TRI

ฟังก์ชัน : กำนดค่ากระแสไฟฟ้าของแหล่งจ่ายตัวที่ 1 และ 2 ให้มีค่าเท่ากันตามต้องการ

คำสั่ง : SR1 และ SR0

รูปแบบ : SR1

ฟังก์ชัน : ทำการ enable โหมด service request โดยไม่มีการเปลี่ยนแปลงใดๆ ในสถานะของอุปกรณ์ที่จะมีผลต่อสัญญาณ SRQ (service request)

รูปแบบ : SR0

ฟังก์ชัน : ทำการยกเลิกโหมด service request

คำสั่ง : STA

รูปแบบ : STA

ค่าที่ส่งกลับ : OP1/0SR1/0ER0/1CV1/CC1/CV2/CC2/RM0/1

OP0 สวิทช์เอาท์พุทปิด

OP1 สวิทช์เอาท์พุทเปิด

SQ1 แสดงสถานะของอุปกรณ์ที่เปลี่ยนแปลง (CV ไปเป็น CC หรือ OP1 ไปเป็น OP0 เป็นต้น) (จะทำงานเฉพาะเมื่อ SRQ นั้น enable หรือมีค่าเป็น 1, ดูคำสั่ง SR1)

SQ0 เมื่อสัญญาณ service request (SRQ) ถูกทำให้ enable แสดงว่าไม่มีการเปลี่ยนแปลงสถานะของเครื่องมือ

ER0 ไม่มีความผิดพลาด

ER1 ความร้อนสูงเกิน

CV1 แหล่งจ่ายตัวที่ 1 ทำการจ่ายศักดาไฟฟ้าคงที่

CC1 แหล่งจ่ายตัวที่ 1 ทำการจ่ายกระแสไฟฟ้าคงที่

CV2 แหล่งจ่ายตัวที่ 2 ทำการจ่ายศักดาไฟฟ้าคงที่

CC2 แหล่งจ่ายตัวที่ 2 ทำการจ่ายกระแสไฟฟ้าคงที่

RM1 อุปกรณ์อยู่ในโหมดรีโมท

RM0 อุปกรณ์ไม่อยู่ในโหมดรีโมท

ฟังก์ชัน : สั่งให้เครื่องส่งค่าสตริงที่เก็บสถานะปัจจุบันของเครื่องมาให้

คำสั่ง : OP1 และ OP0

รูปแบบ : OP1

ฟังก์ชัน : สวิทช์เอาท์พุทเปิด

รูปแบบ : OP0

ฟังก์ชัน : สวิทช์เอาท์พุทปิด

คำสั่ง : CLR (clear)

รูปแบบ : CLR

ฟังก์ชัน : ยกเลิกฟังก์ชันทั้งหมดของเครื่อง และทำการเริ่มต้นใหม่ที่สถานะศูนย์ (zero status) ในโหมดรีโมท คีย์บอร์ดไม่สามารถใช้งานได้และสวิทช์เอาท์พุทปิด รวมทั้งศักดาไฟฟ้าและกระแสไฟฟ้าจะถูกตั้งค่าให้เป็นศูนย์

คำสั่ง : VER

รูปแบบ : VER

ค่าที่ส่งกลับ : sw Vx.xhw Vx.xxxxxxx HAMEG/Paris KRP&VM

ฟังก์ชัน : แสดงเวอร์ชันของซอฟต์แวร์และฮาร์ดแวร์ของเครื่อง

คำสั่ง : ID?

รูปแบบ : ID?

ค่าที่ส่งกลับ : HM8142-1

ฟังก์ชัน : แสดงถึงชื่อรุ่นของเครื่อง

2. Programmable Function Genertor ของบริษัท HAMEG รุ่น HM 8130 ทำหน้าที่สร้างสัญญาณรูปแบบต่างๆ ป้อนให้แก่วงจรที่ทำการทดลอง และมีคำสั่งควบคุมฟังก์ชันในการทำงานผ่านระบบบัสแบบ IEEE-488 โดยสามารถแบ่งได้เป็นกลุ่มคำสั่ง คือ

คำสั่งที่ไม่ต้องมีข้อมูลประกอบ

คำสั่ง : SIN

รูปแบบ : SIN

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นรูปซายน์

คำสั่ง : TRI

รูปแบบ : TRI

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นรูปสามเหลี่ยม

คำสั่ง : SQR

รูปแบบ : SQR

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นรูปสี่เหลี่ยม

คำสั่ง : PLS

รูปแบบ : PLS

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นเป็นพัลซ์

คำสั่ง : RMS

รูปแบบ : RMS

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นรูปฟันเลื่อย โดยมีสัญญาณขาขึ้นเป็นบวก

คำสั่ง : RMN

รูปแบบ : RMN

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นรูปฟันเลื่อย โดยมีสัญญาณขาขึ้นเป็นลบ

คำสั่ง : ARB

รูปแบบ : ARB

ฟังก์ชัน : ฟังก์ชันการสร้างสัญญาณคลื่นรูปแบบ arbitary

คำสั่ง : SW1 / 0

รูปแบบ : SW1 / 0

ฟังก์ชัน : การเลือกการกวาดสัญญาณเปิด / ปิด

คำสั่ง : CTM

รูปแบบ : CTM

ฟังก์ชัน : การเลือกโหมดการสร้างสัญญาณแบบต่อเนื่อง

คำสั่ง : GTM

รูปแบบ : GTM

ฟังก์ชัน : การเลือกโหมดการสร้างสัญญาณแบบ GATED

คำสั่ง : TRM

รูปแบบ : TRM

ฟังก์ชัน : การเลือกโหมดการสร้างสัญญาณแบบ TRIGGER

คำสั่ง : OT1 / 0

รูปแบบ : OT1 / 0

ฟังก์ชัน : การเลือกให้เอาท์พุทเปิด / ปิด

คำสั่ง : OF1 / 0

รูปแบบ : OF1 / 0

ฟังก์ชัน : การเลือกให้ค่าออฟเซ็ทเปิด / ปิด

คำสั่ง : DFR

รูปแบบ : DFR

ฟังก์ชัน : ให้แสดงความถี่ของสัญญาณ

คำสั่ง : DST

รูปแบบ : DST

ฟังก์ชัน : ให้แสดงความถี่เริ่มต้นของสัญญาณ

คำสั่ง : DSP

รูปแบบ : DSP

ฟังก์ชัน : ให้แสดงความถี่สุดท้ายของสัญญาณ

คำสั่ง : DWT

รูปแบบ : DWT

ฟังก์ชัน : ให้แสดงความกว้างของสัญญาณพัลซ์

คำสั่ง : DSW

รูปแบบ : DSW

ฟังก์ชัน : ให้แสดงเวลาในการกวาดสัญญาณ

คำสั่ง : DAM

รูปแบบ : DAM

ฟังก์ชัน : ให้แสดงค่าแอมปลิจูดของเอาท์พุท

คำสั่ง : DOF

รูปแบบ : DOF

ฟังก์ชัน : ให้แสดงค่าออฟเซ็ท

คำสั่ง : RM0

รูปแบบ : RM0

ฟังก์ชัน : ยกเลิกโหมดรีโมท แล้วกลับเข้าสู่การควบคุมการทำงานผ่านหน้าปัทม์ของเครื่อง เงื่อนไขนี้สามารถทำได้โดยการกดปุ่ม LOCAL เช่นเดียวกัน

ข้อสังเกต : คำสั่ง RM0 จะยกเลิกคำสั่ง LK1 ด้วย

คำสั่ง : LK1

รูปแบบ : LK1

ฟังก์ชัน : ยกเลิกปุ่ม LOCAL ชั่วคราว โดยที่สถานะนี้การควบคุมการทำงานของเครื่องทำได้โดยผ่านทางระบบบัสเท่านั้น และการกลับสู่สถานะ LOCAL ไม่สามารถทำได้โดยการกดปุ่ม LOCAL

คำสั่ง : LK0

รูปแบบ : LK0

ฟังก์ชัน : ยกเลิกสถานะการยกเลิกปุ่ม LOCAL ชั่วคราว ทำให้สามารถกดปุ่ม LOCAL เพื่อที่จะกลับสู่สถานะ LOCAL ได้ ทำให้สามารถควบคุมการทำงานของเครื่องผ่านปุ่มหน้าปัทม์ได้อีกครั้ง

คำสั่ง : TRG

รูปแบบ : TRG

ฟังก์ชัน : TRIG คาบของสัญญาณ (ปิดการกวาดสัญญาณ) หรือการกวาดสัญญาณที่สมบูรณ์ (การกวาดสัญญาณเปิด)

คำสั่ง : CLR

รูปแบบ : CLR

ฟังก์ชัน : รีเซ็ทเครื่องและกลับสู่สถานะเริ่มต้น คำสั่ง CLR เหมือนกับคำสั่ง SDC ตามมาตรฐานของ IEEE-488

คำสั่ง : ARC

รูปแบบ : ARC

ฟังก์ชัน : ลบข้อมูลที่เป็นการสร้างสัญญาณแบบใดๆ และรีเซ็ทตัวนับภายในให้มีค่าเป็นศูนย์

คำสั่ง : ARE

รูปแบบ : ARE

ฟังก์ชัน : ยกเลิกการเขียนข้อมูลที่ใช้สร้างสัญญาณแบบ arbitary

คำสั่งที่ประกอบด้วยข้อมูลที่เป็นเลขทศนิยม

คำสั่งทั้งหมดที่ประกอบด้วยข้อมูลแบบทศนิยมจะประกอบด้วยตัวอักษร 3 ตัวแล้วตามด้วย " : " ความยาวของข้อมูลที่มากที่สุดจะเป็นเลขจำนวน 5 หลักรวมกับจุดทศนิยมด้วย โดยรูปแบบของคำสั่งนั้นจะมีค่าเลขยกกำลังหรือไม่มีก็ได้ จะมีจุดทศนิยมหรือไม่มีก็ได้ แต่จะมีหน่วยเป็นโวลท์, เฮิร์ทซ์ และวินาที แต่จะไม่มีการส่งค่าหน่วยลงไปบนบัส ข้อมูลสามารถมีเครื่องหมายนำหน้าได้ถ้าจำเป็น แต่สัญญาณที่เป็นบวกไม่ต้องมีเครื่องหมายนำหน้าก็ได้ ระหว่างเครื่องหมายกับค่าที่กำหนดจะต้องติดกันห้ามเว้นวรรค

ตัวอย่าง : FRQ:1000 มีค่าเท่ากับ FRQ:1000.0

FRQ:1E3 มีค่าเท่ากับ FRQ:1E+3

FRQ:1.0000E+3 มีค่าเท่ากับ FRQ:10E+2

FRQ:0.0001E7 มีค่าเท่ากับ FRQ:10000E-1

คำสั่ง : FRQ

รูปแบบ : FRQ:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าความถี่ที่ต้องการ < > Hz

คำสั่ง : STT

รูปแบบ : STT:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าความถี่เริ่มต้น < > Hz

คำสั่ง : STP

รูปแบบ : STP:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าความถี่สุดท้าย < > Hz

คำสั่ง : SWT

รูปแบบ : SWT:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าเวลาในการกวาดสัญญาณ < > s

คำสั่ง : WDT

รูปแบบ : WDT:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าความกว้างของสัญญาณพัลซ์ < > s

คำสั่ง : AMP

รูปแบบ : AMP:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าแอมปลิจูดของสัญญาณ < > V

คำสั่ง : OFS

รูปแบบ : OFS:< ข้อมูล >

ฟังก์ชัน : กำหนดค่าออฟเซ็ทของสัญญาณ < > V

หมายเหตุ : ค่าความถี่และเวลากำหนดได้มากที่สุด 5 หลัก และค่าศักดาไฟฟ้ากำหนดได้มากที่สุด 3 หลัก

ค่าแอมปลิจูดสามารถกำหนดได้ 2 วิธี ถ้าเป็นค่าที่ไม่มีเครื่องหมายจะถือว่าเป็นค่าศักดาไฟฟ้าแบบ peak to peak (ในกรณีของสัญญาณรูปพัลซ์นั้น ศักดาไฟฟ้าสูงสุดจะมีค่าเท่ากับครึ่งหนึ่งของค่านี้) ถ้าเป็นค่าที่มีเครื่องหมายนำหน้าจะถือว่าเป็นค่าศักดาไฟฟ้าสูงสุด

คำสั่งร้องขอ

คำสั่งเหล่านี้จะทำการเก็บค่าสตริงของข้อมูลที่สามารถอ่านออกมาได้ทันทีที่เครื่องถูกกำหนดให้เป็นตัวส่ง โดยมีคำสั่งต่างๆ และรูปแบบ คือ

คำสั่ง : FRQ?

รูปแบบ : FRQ?

ฟังก์ชัน : แสดงความถี่ของสัญญาณ

คำสั่ง : STT?

รูปแบบ : STT?

ฟังก์ชัน : แสดงความถี่เริ่มต้นของสัญญาณ

คำสั่ง : STP?

รูปแบบ : STP?

ฟังก์ชัน : แสดงความถี่สุดท้ายของสัญญาณ

คำสั่ง : SWT?

รูปแบบ : SWT?

ฟังก์ชัน : แสดงเวลาในการกวาดสัญญาณ

คำสั่ง : WDT?

รูปแบบ : WDT?

ฟังก์ชัน : แสดงความกว้างของสัญญาณพัลซ์

คำสั่ง : AMP?

รูปแบบ : AMP?

ฟังก์ชัน : แสดงค่าแอมปลิจูดของศักดาไฟฟ้าเอาท์พุท

คำสั่ง : OFS?

รูปแบบ : OFS?

ฟังก์ชัน : แสดงค่าออฟเซ็ท

คำสั่ง : ARD?

รูปแบบ : ARD?

ฟังก์ชัน : แสดงข้อมูลของสัญญาณแบบ arbitary

คำสั่ง : ID?

รูปแบบ : ID?

ฟังก์ชัน : แสดงหมายเลขรุ่นของเครื่อง

คำสั่ง : VER

รูปแบบ : VER

ฟังก์ชัน : แสดงเวอร์ชันของซอฟต์แวร์และฮาร์ดแวร์ของเครื่อง

คำสั่ง : STA?

รูปแบบ : STA?

ฟังก์ชัน : แสดงสถานะของเครื่อง

สตริงข้อมูลที่อ่านได้จะมีทั้งหมด 21 ตัวอักษร ซึ่งจะบอกถึงสถานะของเครื่อง โดยมีความหมาย คือ

OT0OF0SW0SINCTMDFRDAM

1. OT0 สวิทช์เอาท์พุทเปิด (ปิด)
2. OF0 การเซ็ทค่าออฟเซ็ทปิด (เปิด)
3. SW0 การกวาดสัญญาณปิด (เปิด)
4. SIN รูปแบบของสัญญาณ (ซายน์)
5. CTM โหมดของการทำงาน (โหมดการสร้างสัญญาณแบบต่อเนื่อง)
6. DFR แสดงรายละเอียด (ความถี่)
7. DAM แสดงรายละเอียด (แอมปลิจูด)

ค่าเอาท์พุทจะอยู่ในรูปแบบเลขทศนิยมกับเลขยกกำลัง สตริงข้อมูลแต่ละค่าจะขึ้นต้นด้วยคำสั่งของตัวมันเอง เช่น

"FRQ:1.2345E+3"

"OFS:-3.0E+0"

"WDT:45.6E-6"

สำหรับการส่งคำสั่งควบคุมเครื่องสามารถที่จะส่งคำสั่งไปควบคุมพร้อมกันหลายฟังก์ชัน โดยสามารถส่งคำสั่งเหล่านั้นไปพร้อมกันเป็นข้อมูลชุดเดียวกันได้ เช่น

"FRQ:12.34E+3 TRI OT1 AMP:10"

Frequency 12.3 K Hz; Triangle; Output on; Voltage amplitude 10 V

3. Programmable Multimeter ของบริษัท HAMEG รุ่น HM 8112-2 ใช้สำหรับวัดค่าพารามิเตอร์ต่างๆ ทางไฟฟ้า โดยมีคำสั่งควบคุมฟังก์ชันการทำงานผ่านระบบบัสแบบ IEEE-488 คือ

คำสั่ง : VD

รูปแบบ : VD

ฟังก์ชัน : การวัดศักดาไฟฟ้ากระแสตรง

คำสั่ง : VA

รูปแบบ : VA

ฟังก์ชัน : การวัดศักดาไฟฟ้ากระแสสลับ

คำสั่ง : O2

รูปแบบ : O2

ฟังก์ชัน : การวัดค่าความต้านทาน

คำสั่ง : ID

รูปแบบ : ID

ฟังก์ชัน : การวัดไฟฟ้ากระแสตรง

คำสั่ง : IA

รูปแบบ : IA

ฟังก์ชัน : การวัดไฟฟ้ากระแสสลับ

คำสั่ง : TC

รูปแบบ : TC

ฟังก์ชัน : การวัดอุณหภูมิแบบเซลเซียส

คำสั่ง : TK

รูปแบบ : TK

ฟังก์ชัน : การวัดอุณหภูมิแบบเคลวิน

คำสั่ง : TF

รูปแบบ : TF

ฟังก์ชัน : การวัดอุณหภูมิแบบฟาเรนไฮต์

คำสั่ง : RX

รูปแบบ : RX

ฟังก์ชัน : เลือกช่วงการวัดช่วงต่างๆ โดย " X " คือช่วงการวัดที่ต้องการ คือ

R1 ช่วงเวลา	0.2 Vdc,	Vac,	KΩ	...	...
R2 ช่วงเวลา	2 Vdc,	Vac,	KΩ	2 mAdc	2 mAac
R3 ช่วงเวลา	20 Vdc,	Vac,	KΩ	...	...
R4 ช่วงเวลา	200 Vdc,	Vac,	KΩ	...	...
R5 ช่วงเวลา	1000 Vdc,	Vac,	2000 KΩ	2000 mAdc	2000 mAac
R6 ช่วงเวลา	10000 Vdc,	Vac,	12000 KΩ	...	...

คำสั่ง : A0 / A1

รูปแบบ : A0 / A1

ฟังก์ชัน : ไม่เลือกฟังก์ชันการปรับช่วงการวัดอัตโนมัติ, เลือกฟังก์ชันการปรับช่วงการวัดอัตโนมัติ

คำสั่ง : TX

รูปแบบ : TX

ฟังก์ชัน : เลือกช่วงเวลาและจำนวนหลักในการแสดงผลที่ได้จากการวัด

T1 ช่วงเวลา	100 ms	การแสดงผล	5 1/2 หลัก
T2 ช่วงเวลา	1 s	การแสดงผล	5 1/2 หลัก
T3 ช่วงเวลา	1 s	การแสดงผล	6 1/2 หลัก
T4 ช่วงเวลา	10 s	การแสดงผล	6 1/2 หลัก

คำสั่ง : ZO

รูปแบบ : ZO

ฟังก์ชัน : เลือกการปรับค่าออฟเซ็ท

คำสั่ง : S0

รูปแบบ : S0

ฟังก์ชัน : เริ่มการวัดแบบต่อเนื่อง

คำสั่ง : S1

รูปแบบ : S1

ฟังก์ชัน : หยุดการวัดแบบต่อเนื่อง

คำสั่ง : MO

รูปแบบ : MO

ฟังก์ชัน : เลือกช่องมัลติเพลกเซอร์

คำสั่ง : M0 - M9

รูปแบบ : MO - M9

ฟังก์ชัน : เลือกช่อง 0 - 9

คำสั่ง : L0

รูปแบบ : L0

ฟังก์ชัน : ส่งข้อมูลเฉพาะข้อมูลบล็อกแรก

คำสั่ง : L1

รูปแบบ : L1

ฟังก์ชัน : ส่งข้อมูลทั้งสองบล็อกข้อมูล (ข้อมูลที่วัดและตัวอักษรจะอยู่ในข้อมูลบล็อกแรก ส่วนข้อมูลที่เป็นการโปรแกรมจะอยู่ในข้อมูลบล็อกที่ 2)

รูปแบบข้อมูลที่ส่งนั้น จะประกอบด้วยข้อมูล 2 บล็อกข้อมูล โดยข้อมูลบล็อกแรกจะประกอบด้วยตัวอักษร 12 ตัว และข้อมูลบล็อกที่ 2 จะประกอบด้วย ตัวอักษร 20 ตัวอักษรบวกกับตัวอักษรที่ปิดท้ายข้อมูล โดยรูปแบบข้อมูลจะมีลักษณะดังนี้ คือ

+X.XXXXXXXE+XVDR1A0T1S0Q0M0X0P0B0

โดยข้อมูลบล็อกแรกจะเป็นข้อมูลที่ได้จากการวัด (+X.XXXXXXXE+X) และข้อมูลบล็อกที่ 2 จะเป็นฟังก์ชันในการวัดของเครื่องหรือสถานะของเครื่องนั่นเอง

คำสั่ง : Q0

รูปแบบ : Q0

ฟังก์ชัน : ไม่ส่งสัญญาณ SRQ (service request)

คำสั่ง : Q1

รูปแบบ : Q1

ฟังก์ชัน : ส่งสัญญาณ SRQ (service request)

คำสั่ง : NVXXXXXXXX

รูปแบบ : NVXXXXXXXX

ฟังก์ชัน : ทำการปรับแต่งเครื่องผ่านระบบบัสแบบ IEEE 488 โดยหลังคำสั่ง NV นั้น เครื่องจะถือว่าเลขจำนวนเต็ม 6 หลักเป็นค่าที่ใช้สำหรับการปรับแต่ง

คำสั่ง : P1

รูปแบบ : P1

ฟังก์ชัน : แสดงค่าออฟเซ็ท คำนวนจาก R = X - C

คำสั่ง : P2

รูปแบบ : P2

ฟังก์ชัน : แสดงเปอร์เซ็นต์ความเบี่ยงเบน คำนวนจาก R = 100 (X - C) / C

คำสั่ง : P3

รูปแบบ : P3

ฟังก์ชัน : แสดง dB คำนวนจาก R = 20 LOG (X / C)

คำสั่ง : P4

รูปแบบ : P4

ฟังก์ชัน : แสดง dBm คำนวนจาก R = 20 LOG (X - C)

เมื่อ C = 0.775 V สำหรับค่าศักดาไฟฟ้า และ C = 1.29 mA สำหรับค่ากระแสไฟฟ้า

คำสั่ง : PXEN

รูปแบบ : PXEN

ฟังก์ชัน : เลือกค่าการวัดที่ X เป็นค่าคงที่ ถ้า P2-4; X มีค่าเท่ากับ 1, 2, 3 ตามลำดับ

คำสั่ง : ID?

รูปแบบ : ID?

ฟังก์ชัน : ทำการส่งหมายเลขรุ่นของเครื่องลงบนหน่วยความจำของเครื่อง

คำสั่ง : STA?

รูปแบบ : STA?

ฟังก์ชัน : ทำการส่งสถานะของเครื่อง (ข้อมูลบล็อกที่ 2)

คำสั่ง : D0 / D1

รูปแบบ : D0 / D1

ฟังก์ชัน : ปิด / เปิดหน้าจอ

คำสั่ง : BX

รูปแบบ : BX

ฟังก์ชัน : ส่งค่าของการสวิทช์ที่เลือกครั้งสุดท้าย โดยที่ x = 1, ..., 9, A, ..., F

คำสั่ง : EOI

รูปแบบ : EOI

ฟังก์ชัน : เลือกให้มีการส่งสัญญาณ EOI

คำสั่ง : EOS1 / EOS2

รูปแบบ : EOS1 / EOS2

ฟังก์ชัน : เลือกให้ข้อมูลที่ส่งมีการส่งค่า EOS1 หรือ EOS2 ปิดท้ายข้อมูล

คำสั่ง : END

รูปแบบ : END

ฟังก์ชัน : ยกเลิกตัวอักษรที่ปิดท้ายสตริงข้อมูล

4. Storage Analog / Digital Oscilloscope ของบริษัท HAMEG รุ่น HM 1007 โดยสามารถวัดสัญญาณที่มีค่าความถี่ได้ถึง 100 เมกะเฮิร์ทซ์ โดยOscilloscope นี้สามารถที่จะทำการบันทึกสัญญาณที่ได้จากการวัดมาเก็บไว้ในหน่วยความจำของเครื่อง และสามารถที่จะอ่านค่าของสัญญาณที่บันทึกเก็บไว้ผ่านระบบบัสแบบ IEEE 488 ได้ แต่ไม่สามารถที่จะควบคุมช่วงของเวลาต่อช่องและจำนวนช่องต่อโวล์ทได้ สำหรับคำสั่งที่ใช้ควบคุมการทำงานผ่านระบบบัส IEEE 488 มีดังนี้ คือ

คำสั่ง : ID?

รูปแบบ : ID?

ฟังก์ชัน : ทำการส่งหมายเลขรุ่นของเครื่องลงบนหน่วยความจำของเครื่อง

คำสั่ง : DIG

รูปแบบ : DIG [channel code]

ฟังก์ชัน : 1 ส่งข้อมูลเก็บไว้ในหน่วยความจำของเครื่องเฉพาะช่องการวัดที่ 1

2 ส่งข้อมูลเก็บไว้ในหน่วยความจำของเครื่องเฉพาะช่องการวัดที่ 2

3 ส่งข้อมูลเก็บไว้ในหน่วยความจำของเครื่องทั้งสองช่องการวัด

คำสั่ง : GET [channel code]

รูปแบบ : GET

ฟังก์ชัน : 1 ส่งสัญญาณ TRIG โดยไม่กำหนดช่องการวัด

2 ส่งสัญญาณ TRIG โดยกำหนดช่องการวัด

คำสั่ง : STA

รูปแบบ : STA

ฟังก์ชัน : ตรวจสอบสถานะการเซ็ทช่องการวัดของ Oscilloscope

คำสั่ง : TXT

รูปแบบ : TXT < สตริง@ >

ฟังก์ชัน : ส่งสตริงไปยังการอินเตอร์เฟส

คำสั่ง : OFS

รูปแบบ : OFS

ฟังก์ชัน : ตรวจสอบการเลื่อนตำแหน่งของ Y จากหน่วยความจำอ้างอิง หลังจากสัญญาณอ้างอิงถูกเก็บไว้ในหน่วยความจำ

คำสั่ง : FRM

รูปแบบ : FRM [format]

ฟังก์ชัน : กำหนดรูปแบบข้อมูลที่ทำการส่ง (0 ข้อมูลเป็นแบบไบนารี)

คำสั่ง : V24

รูปแบบ : V24

ฟังก์ชัน : ส่งข้อมูลผ่านระบบบัสแบบ IEEE-488 ไปยังอุปกรณ์ภายนอกที่ต่อผ่านระบบบัสแบบ RS232C

คำสั่ง : PRN

รูปแบบ : PRN

ฟังก์ชัน : เริ่มส่งข้อมูลผ่านพอร์ตใดพอร์ตหนึ่งในจำนวน 3 พอร์ตของเครื่อง

คำสั่ง : XYZ

รูปแบบ : XYZ

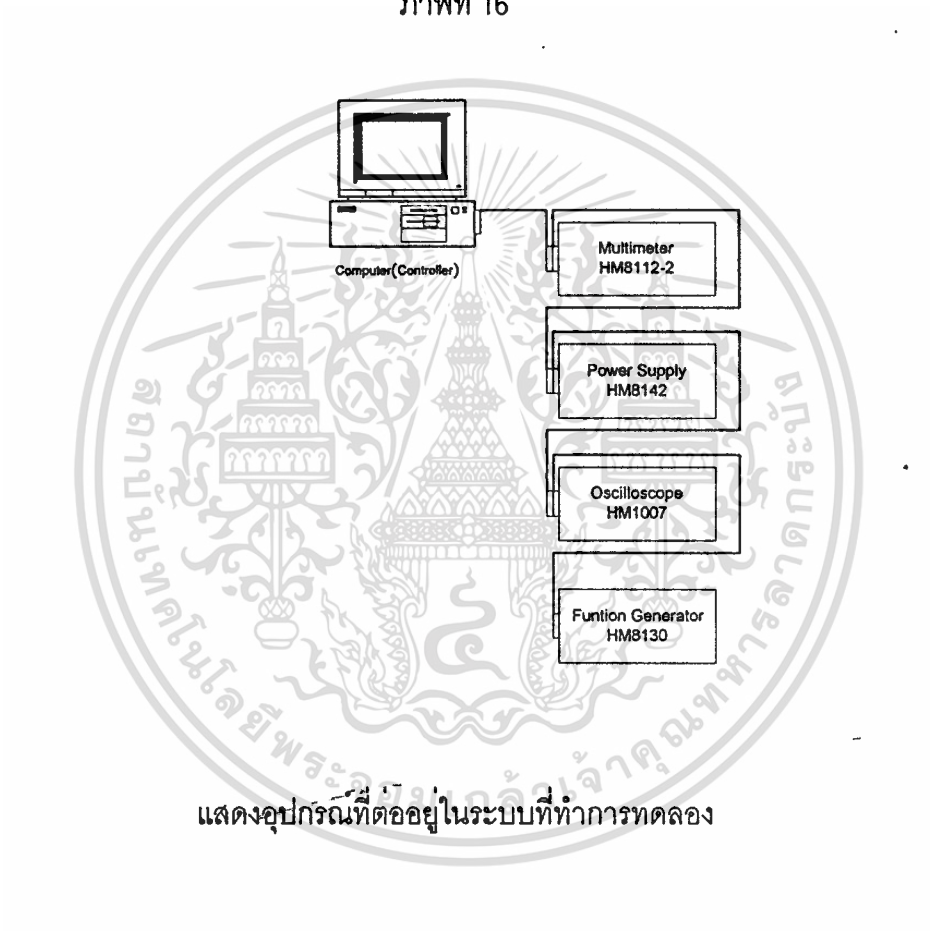
ฟังก์ชัน : เปลี่ยนโหมดการอ่านค่าสัญญาณจากโหมดปกติเป็นการอ่านค่าสัญญาณแบบ XY

การเชื่อมต่ออุปกรณ์ในระบบ

สำหรับการเชื่อมต่ออุปกรณ์ในระบบเครื่องมือวัดที่ทำการศึกษานั้น ได้ทำการต่ออุปกรณ์ตามรูปแบบมาตรฐาน IEEE-488 โดยทำการต่อแบบการต่อต่อเนื่องกัน (Daisy Chain Configuration) ดังแสดงในภาพที่ 2 เนื่องจากมีข้อดีคือ ขั้วต่อของสายสัญญาณที่ต่อเข้ากับ

อุปกรณ์ในระบบไม่ได้ต่อรวมกันไว้ที่จุดจุดเดียว แต่จะเรียงต่อเนื่องกันไป ทำให้น้ำหนักของขั้วต่อสัญญาณซึ่งมีขนาดใหญ่ และมีน้ำหนักมากจะกระจายไปอยู่ทุกๆ ตัวอุปกรณ์ ในขณะที่การต่อแบบกระจาย (Star Configuration) เป็นการต่อที่ขั้วต่อของสายสัญญาณจะต่อรวมเข้าไว้ที่จุดใดจุดหนึ่ง ทำให้น้ำหนักของขั้วสายสัญญาณไปรวมอยู่ที่จุดใดจุดหนึ่ง ซึ่งจะทำให้ขั้วต่อสายสัญญาณที่อุปกรณ์ตัวนั้นอาจเกิดการชำรุดเสียหายได้ง่าย ดังแสดงในภาพที่ 16

ภาพที่ 16



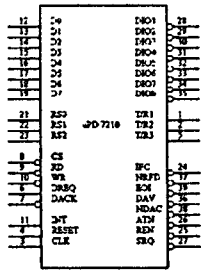
แสดงอุปกรณ์ที่ต่ออยู่ในระบบที่ทำการทดลอง

บทที่ 4
การออกแบบการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB)

กล่าวนำ

บทนี้จะกล่าวถึงการศึกษาเพื่อออกแบบสร้างการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 โดยใช้ไอซี uPD 7210 สำหรับการควบคุมอุปกรณ์ในระบบที่ทำการทดลอง ซึ่งประกอบด้วย Digital Multimeter, Power Supply, Storage Oscilloscope และ Function Gnerator โดยการ์ดนี้จะทำหน้าที่เปลี่ยนระบบบัสแบบ PCAT ให้เป็นระบบบัสแบบ IEEE-488 ภายในการ์ดประกอบด้วยวงจรหลัก 2 ส่วน คือ วงจรถอดรหัสแอดเดรสจากไมโครคอมพิวเตอร์ และวงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบ ซึ่งส่วนที่มีความสำคัญมากที่สุดได้แก่ วงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบ เพราะในวงจรส่วนนี้ประกอบด้วยไอซีที่ทำหน้าที่เป็นส่วนประมวลผลและควบคุมสัญญาณต่างๆ ของระบบ และไอซีที่ทำหน้าที่เป็นบัฟเฟอร์ของสัญญาณควบคุมและบัสข้อมูล สำหรับไอซีที่ทำหน้าที่ในการควบคุมสัญญาณต่างๆ ในระบบและทำการประมวลผลในการส่งคำสั่งไปควบคุมอุปกรณ์ในระบบนั้น ได้ทำการเลือกใช้ไอซี เบอร์ uPD 7210 ของบริษัท NEC เนื่องจากหาซื้อได้ง่ายตามท้องตลาดทั่วไปและมีราคาไม่แพงมากนัก โดยไอซีนี้ทางบริษัทผู้ผลิตได้ออกแบบมาสำหรับการใช้งานในการควบคุมการอินเตอร์เฟสตามมาตรฐาน IEEE-488 โดยเฉพาะจึงทำให้เกิดความสะดวกในการออกแบบและใช้งานเป็นอย่างมาก

ภาพที่ 17



แสดงไอซี uPD 7210 และขาสัญญาณต่างๆ

ไอซี uPD 7210 มีขาสัญญาณทั้งหมด 40 ขา ดังแสดงในภาพที่ 17 และรายละเอียดของความหมายสัญญาณต่างๆ แสดงไว้ดังตารางที่ 4 คุณสมบัติที่สำคัญของไอซีเบอร์ดังกล่าว คือ ภายในจะประกอบด้วยรีจิสเตอร์ภายในทั้งหมด 16 ตัว โดยแบ่งเป็นรีจิสเตอร์สำหรับการอ่านข้อมูล 8 ตัว และรีจิสเตอร์สำหรับการเขียนข้อมูลอีก 8 ตัว สำหรับการควบคุมอุปกรณ์ต่างๆ ภายในระบบควบคุมนั้น สามารถทำได้โดยการเขียนและอ่านข้อมูลผ่านรีจิสเตอร์ภายในเหล่านี้

ตารางที่ 4

แสดงรายละเอียดของความหมายขาสัญญาณต่างๆ ของไอซี uPD 7210

Pin	Name	I / O	Description
1	T/R1	O	Transmit / Receive Control - Input / Output Control Signal for the GPIB Bus Transcievers
2	T/R2	O	Transmit / Receive Control - The functions of T/R2, T/R3 are determined by the values of TRM1 of the address mode register
3	CLOCK	I	Clock - (1-8 Mhz) Reference Clock for generating the state change prohibit times T1, T6, T7, T9 specified in IEEE Standard 488 - 1978
4	RST	I	Reset - Resets 7210 to an idle state when high (active high)
5	T/R3	O	Transmit / Receive Control - Function determined by TRM1 and TRM0 of address mode register
6	DRQ	O	DMA Request - 7210 requests data transfer to the computer system, becomes low on input of DMA acknowledge signal DACK
7	DACK	I	DMA Acknowledge - (active low) Signal connects the computer system data on bus to the data register of 7210
8	CS	I	Chip Select - (active low) Enables access to the register selected by RS0 - 2 on D0 - 7 (read or write operation)
9	RD	I	Read - (active low) Places contents of read register specified by RS0 - 2 on D0 - 7 (Computer Bus)
10	WR	I	Write - (active low) writes data on D0 - 7 into the write register specified by RS0 - 2

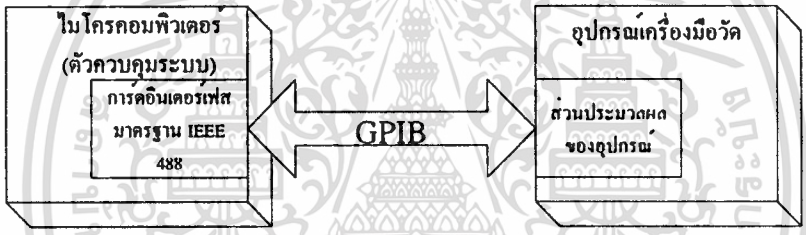
ตารางที่ 4 (ต่อ)

แสดงรายละเอียดของความหมายขาสัญญาณต่างๆ ของไอซี uPD 7210

Pin	Name	I / O	Description
11	INT	O	Interrupt Request - (active high / low) Becomes active due to any 1 of 13 internal interrupt factors (unmarked) active state software configurable, active high on chip reset
12-19	D0 - 7	I / O	Data Bus - 8 bit bidirectional data bus, for interface to computer system
20	GND		Ground
21-23	RS0 - 2	I	Register Select - These lones select one of eight read registers during a read (write) operation
24	IFC	I / O	Interface Clear - Control line used for clearing the interface functions
25	REN	I / O	Remote Enable - Control line used to select remote or local control of the devices
26	ATN	I / O	Attention - Control lone which indicates whether data on DIO lines is an interface message or device dependent message
27	SRQ	I / O	Service Request - Control line used to request the controller for service
28-35	DIO1-8	I / O	Data Input / Output - 8 bit bidirectional bus for transfer of message on the GPIB
36	DAV	I / O	Data Valid - Handshake line indicating that data on DIO lines is valid
37	NRFD	I / O	Ready for Data-Handshake line indicating that device is ready for data
38	NDAC	I / O	Data Accepted - Handshake line indicating completion of message reception
39	EOI	I / O	End or Identify - Control line used to indicate the end of multiple byte transfer sequence or to execute a parallel polling in conjunction with ATN
40	VCC	I / O	+ 5V DC - Technical Specifications : +5V; NMOS; 500mW; 40 Pins; TTL Compatible; 1-8 MHz

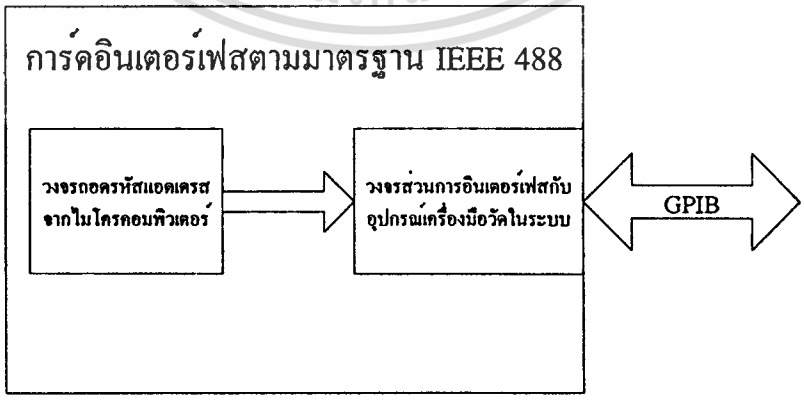
ระบบการควบคุมเครื่องมือวัดผ่านระบบบัสมาตรฐาน IEEE-488 สามารถแสดงให้เข้าใจได้โดยง่ายดังภาพที่ 18 ในไมโครคอมพิวเตอร์จะประกอบด้วย การ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 ซึ่งเป็นสิ่งสำคัญ เพราะทำหน้าที่ในการอินเตอร์เฟสอุปกรณ์หรือเครื่องมือวัดในระบบเข้ากับไมโครคอมพิวเตอร์หรือตัวควบคุมผ่านระบบบัสมาตรฐานดังกล่าว การ์ดอินเตอร์เฟสนี้สามารถแบ่งออกได้เป็น 2 ส่วนหลัก คือ วงจรถอดรหัสแอดเดรสจากไมโครคอมพิวเตอร์ และวงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบ ซึ่งสามารถแสดงโครงสร้างภายในของการ์ดอินเตอร์เฟสได้ดังภาพที่ 19 และวงจรของการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 ที่ทำการออกแบบสามารถแสดงได้ดังภาพที่ 20 ส่วนรายละเอียดของวงจรแต่ละส่วนนั้นจะได้กล่าวต่อไป

ภาพที่ 18



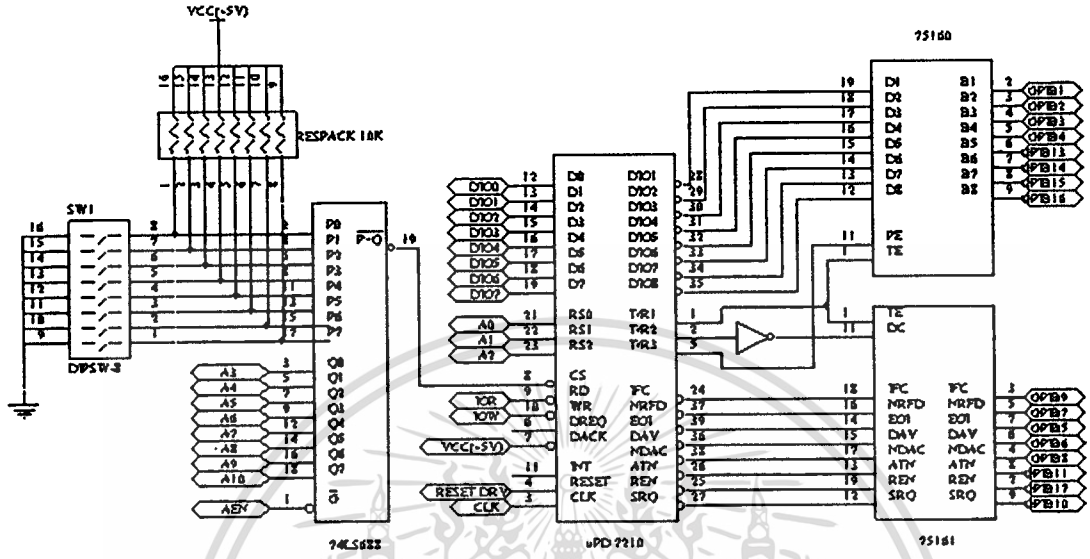
แสดงระบบการควบคุมเครื่องมือวัดผ่านระบบบัสมาตรฐาน IEEE-488 (GPIB)

ภาพที่ 19



แสดงโครงสร้างภายในของการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB)

ภาพที่ 20



แสดงวงจรการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 ที่ทำการออกแบบ

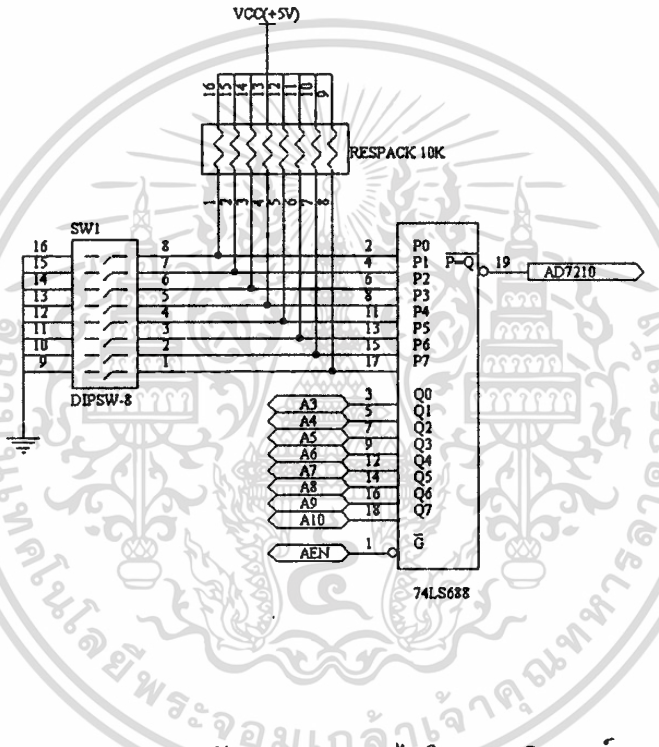
วงจรถอดรหัสแอดเดรสจากไมโครคอมพิวเตอร์ [2]

วงจรประกอบด้วย ไอซี เบอร์ 74LS688, DIP SWITCH แบบ 8 bit และตัวต้านทานขนาด 10 กิโลโอห์มจำนวน 8 ตัว วงจรส่วนนี้จะทำหน้าที่ถอดรหัสแอดเดรสตามที่ต้องการ รายละเอียดของวงจรถอดรหัสแอดเดรสจากไมโครคอมพิวเตอร์แสดงดังภาพที่ 21

หลักการทำงานของวงจร คือ การเลือกกลุ่มแอดเดรสที่ทำการถอดรหัสทำได้โดยการเซ็ท DIP SWITCH ที่ขา P0 - P7 ของไอซี 74LS688 โดยไอซี 74LS688 จะทำการเปรียบเทียบค่าอินพุททั้ง 2 ชุดที่ถูกส่งเข้ามาทางขา P0 - P7 และขา Q0 - Q7 ถ้าอินพุททั้ง 2 ชุดมีค่าเท่ากันเอาท์พุทที่ขา P = Q จะมีลอจิกเป็น 0 (low) จากวงจรนั้นขา Q0 - Q7 จะต่อกับแอดเดรสบิต A3 - A10 ของสล็อตไมโครคอมพิวเตอร์ ส่วนขา P0 - P7 จะต่อกับตัวต้านทานขนาด 10 กิโลโอห์มที่ทำหน้าที่รักษาระดับแรงดัน เพื่อที่จะทำให้ลอจิกเป็น 1 (high) เมื่อขา DIP SWITCH ถูก on (pull up) และขา P0 - P7 ยังต่อเข้ากับขาของ DIP SWITCH ด้วย ส่วนขาอีกด้านหนึ่งของ DIP SWITCH จะต่อลงกราวด์ไว้ ถ้า DIP SWITCH ขาใดถูกทำให้ on ขานั้นก็จะมีลอจิกเป็น 0 แต่ถ้าขาใดถูกทำ

ให้ off ขานั้นก็จะมีลอจิกเป็น 1 ดังนั้นเมื่อมีการเปลี่ยนแปลงการเซ็ท DIP SWITCH ก็จะทำให้แอดเดรสบิท A3 - A10 ซึ่งต่อกับขา Q0 - Q7 ต้องเปลี่ยนแปลงตามไปด้วย จึงจะทำให้เอาท์พุท P = Q ของไอซี 74LS688 แอคทีฟ (0) ได้ ส่วนขา G นั้นต่อเข้ากับขา AEN จากไมโครคอมพิวเตอร์

ภาพที่ 21



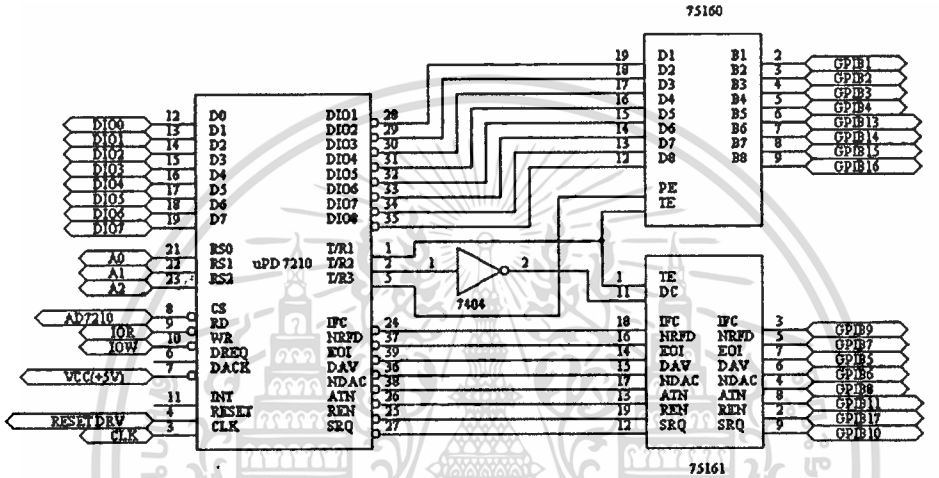
แสดงวงจรถอดรหัสแอดเดรสจากไมโครคอมพิวเตอร์

วงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบ

วงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบนี้ ประกอบด้วยไอซีทั้งหมด 3 ตัว คือ uPD 7210, 75LS160 และ 75LS161 โดยไอซี uPD 7210 เป็นไอซีที่ออกแบบมาสำหรับการควบคุมการอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB) โดยตรง โครงสร้างภายในจะประกอบด้วยรีจิสเตอร์ทั้งหมด 16 ตัว แบ่งเป็นรีจิสเตอร์สำหรับการอ่านและการเขียนอย่างละ 8 ตัว การควบคุมการทำงานของอุปกรณ์ที่ต่ออยู่บนระบบบัสแบบ IEEE-488 (GPIB) ทำได้โดยการส่งคำสั่งหรือข้อมูลผ่านรีจิสเตอร์เหล่านี้ ส่วนไอซี 75LS160 และ 75LS161 เป็นไอซีที่ทำหน้าที่เป็น

ตัวผ่านข้อมูลแบบ 2 ทิศทางหรือบัฟเฟอร์ เพื่อทำการส่งผ่านข้อมูลจากตัวควบคุมไปยังอุปกรณ์ที่ต่ออยู่ในระบบหรือในทางกลับกัน วงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบแสดงดังภาพที่ 22

ภาพที่ 22



แสดงวงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบ

การเข้าถึงรีจิสเตอร์ของ uPD 7210 สามารถอ้างอิงได้จากขา RS0 - RS2 ของไอซี uPD 7210 ที่ต่อไปยังขา A0 - A2 ของสล็อตไมโครคอมพิวเตอร์ โดยแอดเดรสของพอร์ตที่ทำการติดต่อจะเป็นค่าแอดเดรส (AD7210) ที่เกิดจากการเปรียบเทียบแอดเดรสของวงจรถอดรหัสแอดเดรสที่ต่อไปยังขา CS (Chip Select) รวมกับค่าแอดเดรสของ RS0-RS2 ที่สามารถเลือกได้จากการโปรแกรม ทำให้สามารถเข้าถึงรีจิสเตอร์สำหรับการอ่านและเขียนข้อมูลทั้ง 16 ตัวได้ สำหรับการควบคุมการอ่านและเขียนรีจิสเตอร์นั้นควบคุมด้วยขาสัญญาณ RD และ WR ที่ต่อกับขา IOR และ IOW ของสล็อตไมโครคอมพิวเตอร์ตามลำดับ ขา CLOCK เป็นขาสัญญาณนาฬิกาของ uPD 7210 สามารถใช้งานได้ที่ความถี่ 1 - 8 เมกกะเฮิร์ทซ์ โดยต่อกับขา CLK (System Clock) ของสล็อตไมโครคอมพิวเตอร์ ขา RESET ต่อกับขา RESET DRV ของสล็อต ซึ่งระบบควบคุมเครื่องมือวัดนั้นจะถูกทำการรีเซ็ทไปพร้อมกันกับการรีเซ็ทระบบของไมโครคอมพิวเตอร์ ขา

DMAACK (DMA Acknowledge) ต่อเข้ากับไฟบวก VCC เนื่องจากในระบบควบคุมนั้นไม่ต้องการทำขบวนการ DMA

ส่วนอีกด้านหนึ่งของไอซี uPD 7210 จะต่อกับไอซีอีก 2 ตัว คือ 75LS160 และ 75LS161 ขา DIO1 - DIO8 ต่อกับขา D1 - D8 ของไอซี 75LS160 ตามลำดับ เพื่อทำหน้าที่เป็นบัฟเฟอร์ให้กับข้อมูลที่ถูกส่งลงมาบนบัส โดยมีขา TE (Talk Enable) คอยควบคุมสถานะของเอาท์พุทของขา B1 - B8 ที่ต่อไปยังบัสข้อมูลของ GPIB ให้มีสถานะแบบ Free - State เมื่อขา TE มีลอจิกเป็น Low โดยขา TE นี้จะต่อกับขา T/R1 ของไอซี uPD 7210 ส่วนขา PE (Pull Up Enable) ของไอซี 75LS160 จะต่อกับขา T/R3 ของไอซี uPD 7210 ซึ่งสามารถเลือกให้เอาท์พุทของขา B1 -B8 ให้มีสถานะแบบ Totem Pole หรือแบบ Open Collector สำหรับขา T/R2 และ T/R3 ของไอซี uPD 7210 นั้นถูกกำหนดโดยค่า TRM1 และ TRM0 ของรีจิสเตอร์ address mode ส่วนขาสัญญาณต่างๆ ที่เป็นสัญญาณควบคุมและจัดการระบบบัส GPIB ของไอซี uPD 7210 นั้น จะต่อเข้ากับไอซี 75LS161 ที่ทำหน้าที่เป็นบัฟเฟอร์ให้กับสัญญาณควบคุมและจัดการระบบของ GPIB ทั้ง 8 เส้นก่อนที่จะต่อไปยังบัสควบคุมของ GPIB ที่ต่อไปยังอุปกรณ์ในระบบ ขา TE จะต่อเข้ากับขา T/R1 ของไอซี uPD 7210 เช่นเดียวกับขา TE ของไอซี 75LS160 โดยขา TE และขา DC (Direction Control) เป็นฟังก์ชันควบคุมสถานะของสายสัญญาณควบคุมและจัดการระบบของ GPIB

บทที่ 5
การพัฒนาโปรแกรมควบคุมระบบ

กล่าวนำ

บทนี้จะกล่าวถึง การเขียนโปรแกรมควบคุมอุปกรณ์บนระบบบัสมาตรฐาน IEEE-488 (GPIB) การติดต่อระหว่างตัวควบคุมกับตัวส่งหรือตัวรับ ขั้นตอนต่างๆ ในการส่งข้อมูลหรือคำสั่งลงไปบนบัสข้อมูล รวมทั้งอธิบายขั้นตอนต่างๆ ของฟังก์ชันที่ใช้ในการติดต่อ และกล่าวถึงการเขียนโปรแกรมหน้าจอสำหรับใช้ในการคงบคุมอุปกรณ์ในระบบ ซึ่งจะทำให้เข้าใจถึงขั้นตอนการทำงานของโปรแกรมควบคุมและการใช้งานได้ดีขึ้น

การเขียนโปรแกรมควบคุมอุปกรณ์บนระบบบัสมาตรฐาน IEEE-488 (GPIB)

การเขียนโปรแกรมควบคุมอุปกรณ์บนระบบบัสมาตรฐาน IEEE-488 (GPIB) จะมีขั้นตอนในการติดต่อระหว่างอุปกรณ์ทั้งหมดบนระบบบัสโดยเฉพาะ โดยการอาศัยสัญญาณ ATN (Attention) ซึ่งเป็นสัญญาณที่ใช้ในการควบคุมระบบ ขั้นตอนดังกล่าวสามารถจะแบ่งออกได้เป็น 2 ช่วง คือ

1. ช่วงที่สัญญาณ ATN มีลอจิกเป็น Low

ข้อมูลที่ถูกส่งลงบนบัสในช่วงที่สัญญาณ ATN มีลอจิกเป็น Low จะหมายถึงคำสั่งที่เป็นรหัสคำสั่งมาตรฐานของ IEEE-488 ตามตารางที่ 2 โดยรหัสคำสั่งในคอลัมน์ที่ 0 และ 1 จะเป็นคำสั่งที่ใช้ในการควบคุมระบบ โดยที่ภายในช่วงระยะเวลาที่สัญญาณ ATN มีลอจิกเป็น Low นั้น สามารถที่จะส่งคำสั่งมาตรฐานนเหล่านั้นลงไปกี่คำสั่งก็ได้

2. ช่วงที่สัญญาณ ATN มีลอจิกเป็น High

ข้อมูลที่ถูกส่งลงบนบัสในช่วงที่สัญญาณ ATN มีลอจิกเป็น High จะหมายถึงข้อมูลจริงที่เป็นรหัส ASCII ซึ่งจะเป็นคำสั่งหรือฟังก์ชันที่ใช้ควบคุมการทำงานของแต่ละอุปกรณ์ที่ต่ออยู่ในระบบ โดยรหัสคำสั่งดังกล่าวจะแตกต่างกันออกไปขึ้นอยู่กับชนิดของอุปกรณ์และบริษัทผู้ผลิตอุปกรณ์นั้นๆ

ในการส่งข้อมูลที่เป็นรหัสคำสั่งมาตรฐานของ IEEE-488 หรือข้อมูลที่เป็นคำสั่งที่ใช้ควบคุมฟังก์ชันการทำงานของอุปกรณ์ การส่งข้อมูลนั้นจะต้องมีขบวนการในการตรวจสอบว่าระหว่างตัวรับและตัวส่ง ได้ทำการรับหรือส่งข้อมูลถูกต้องเรียบร้อยหรือไม่ ซึ่งขบวนการดังกล่าวก็คือขบวนการแฮนด์เช็คดังที่ได้อธิบายไปแล้วในบทที่ 2

สำหรับขั้นตอนการเขียนโปรแกรมติดต่อของตัวควบคุมกับอุปกรณ์ตัวรับและตัวส่งสามารถแบ่งออกได้เป็น 2 กรณี คือ

1. กรณีตัวควบคุมติดต่อกับตัวส่ง

เมื่อตัวควบคุมจะทำการติดต่อกับอุปกรณ์ที่ทำหน้าที่เป็นตัวส่ง ตัวควบคุมจะทำหน้าที่เป็นตัวรับ สำหรับขั้นตอนการติดต่อแสดงได้ดังแผนผังเวลาในภาพที่ 23 โดยตัวควบคุมจะทำการเปลี่ยนสัญญาณ ATN ให้มีลอจิกเป็น Low แล้วทำการส่งข้อมูลที่เป็นรหัสคำสั่งมาตรฐานของ IEEE-488 ซึ่งเป็นข้อมูลที่บอกแอดเดรสของอุปกรณ์และมีความหมายดังนี้

UNL (3Fh) (Unlistener) ตัวควบคุมจะทำการเคลียร์อุปกรณ์ทุกตัวออกจากการเป็นตัวรับในระบบ หลังจากคำสั่งนี้จะทำให้ไม่มีอุปกรณ์ที่ต่ออยู่บนบัสทำหน้าที่เป็นตัวรับเลย

OTA (40h) (Our Talker Address) ตัวควบคุมจะทำการขอติดต่อกับอุปกรณ์ที่ทำหน้าที่เป็นตัวส่ง จากตัวอย่างในภาพที่ 23 ตัวส่งจะมีแอดเดรสเป็น 0 เพราะว่าในการติดต่อกับอุปกรณ์ที่ทำหน้าที่เป็นตัวส่ง จะต้องทำการบวกแอดเดรสของตัวส่งด้วยค่า 40h (0+40h) โดยตัวส่งก็จะรับรู้ว่าตัวควบคุมติดต่อมาและต้องการให้ทำการส่งข้อมูลลงบนบัส

MLA (My Listener Address) ตัวควบคุมจะแจ้งแอดเดรสของตัวเองในการติดต่อกับตัวส่ง โดยตัวควบคุมจะทำหน้าที่เป็นตัวรับ จากตัวอย่างในภาพที่ 23 ตัวรับจะมีแอดเดรสเป็น 21 เพราะในการติดต่อกับอุปกรณ์ที่เป็นตัวรับจะต้องทำการบวกแอดเดรสของตัวรับด้วย 20h (21+20h = 35h)

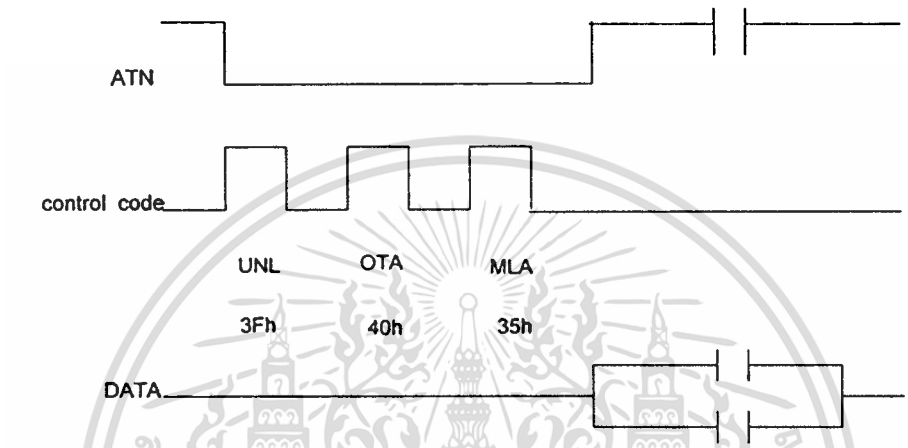
เมื่อสิ้นสุดคำสั่งที่เป็นรหัสคำสั่งมาตรฐานของ IEEE-488 แล้ว ตัวควบคุมจะเปลี่ยนสัญญาณ ATN ให้มีลอจิกเป็น High เป็นการสิ้นสุดขั้นตอนเริ่มต้นในการติดต่อกับตัวส่ง ส่วนข้อมูลที่ถูกส่งลงบนบัสในช่วงที่สัญญาณ ATN เป็น High จะเป็นข้อมูลที่เป็นรหัส ASCII ที่ได้รับจากอุปกรณ์ที่เป็นตัวส่ง

2. กรณีตัวควบคุมติดต่อกับตัวรับ

เมื่อตัวควบคุมจะทำการติดต่อกับอุปกรณ์ที่ทำหน้าที่เป็นตัวรับ ตัวควบคุมจะทำหน้าที่เป็นตัวส่ง สำหรับขั้นตอนการติดต่อแสดงได้ดังแผนผังเวลาในภาพที่ 24 ตัวควบคุมจะทำการ

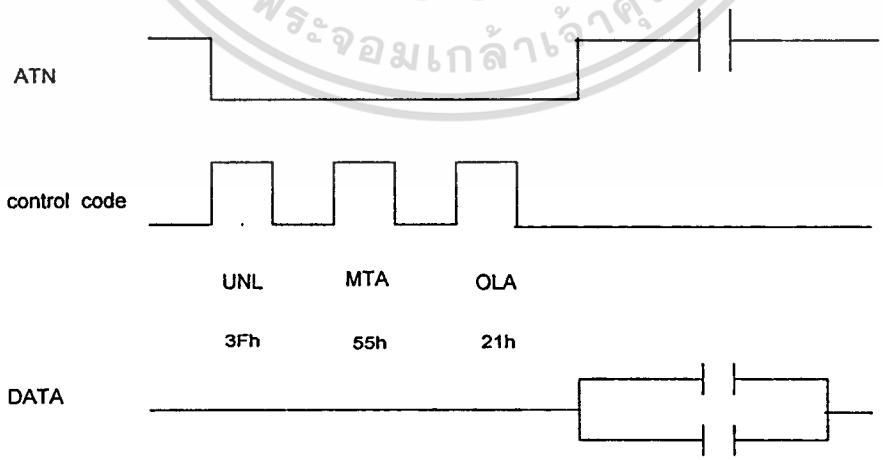
เปลี่ยนสัญญาณ ATN ให้มีลอจิกเป็น Low หลังจากนั้นจะทำการส่งข้อมูลที่เป็นรหัสคำสั่งมาตรฐานของ IEEE-488 ซึ่งเป็นข้อมูลที่บอกแอดเดรสของอุปกรณ์และมีความหมายดังนี้ คือ

ภาพที่ 23



แสดงแผนผังเวลาเมื่อตัวควบคุมทำการติดต่อกับตัวส่ง

ภาพที่ 24



แสดงแผนผังเวลาเมื่อตัวควบคุมทำการติดต่อกับตัวรับ

UNL (3Fh) (Unlistener) ตัวควบคุมจะทำการเคลียร์อุปกรณ์ทุกตัวออกจากการเป็นตัวรับในระบบ

MTA (55Fh) (My Tallker Address) ตัวควบคุมจะแจ้งแอดเดรสของตัวเองในการติดต่อกับอุปกรณ์ที่เป็นตัวรับ โดยตัวควบคุมจะทำหน้าที่เป็นตัวส่งที่มีแอดเดรสเป็น 21 (21+40h = 55h)

OLA (21h) (Our Listener Address) ตัวควบคุมทำการติดต่อกับอุปกรณ์ที่ทำหน้าที่เป็นตัวรับที่มีแอดเดรสเป็น 1 (1+20h = 21h) ตัวรับที่มีแอดเดรสดังกล่าวจะรับรู้ว่าตัวควบคุมต้องการติดต่อ เพื่อให้ทำการรับข้อมูลที่ส่งลงมาบนบัส

คำสั่งตามมาตรฐาน IEEE-488 ในช่วงที่สัญญาณ ATN มีลอจิกเป็น Low อาจมีมากกว่านี้ และอาจจะมีการเรียงลำดับของคำสั่งแตกต่างกันไปขึ้นอยู่กับการใช้งานและการออกแบบโปรแกรม

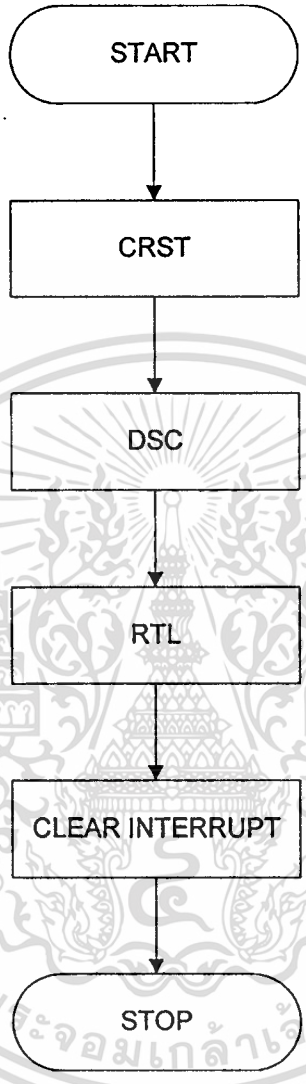
ฟังก์ชันการควบคุมอุปกรณ์เครื่องมือวัดในระบบบัส GPIB

การเขียนโปรแกรมเพื่อทำการควบคุมอุปกรณ์เครื่องมือวัดในระบบ สามารถแสดงฟังก์ชันต่าง ๆ ที่ใช้ในการควบคุมอุปกรณ์ได้ด้วยแผนผังรูปภาพดังต่อไปนี้

1. Initial Controller Card

ฟังก์ชันนี้เป็นการ Initial Card โดยการรีเซท IC uPD7210 ด้วยการส่งคำสั่ง CRST (Chip Reset) หลังจากนั้นทำการส่งคำสั่ง DSC (Disable System Control) และคำสั่ง RTL (Return To Local Generation) แล้วทำการเคลียร์อินเทอร์รัพท์โดยการส่งค่า 0 ลงไปที่รีจิสเตอร์ Interrupt Mask ทั้งสอง ขั้นตอนเหล่านี้สามารถแสดงได้ดังแผนภูมิที่ 25 ในการ Initial Card นั้นสามารถที่จะเรียกใช้คำสั่งอื่นเพิ่มเติมอีกก็ได้

แผนภูมิที่ 25

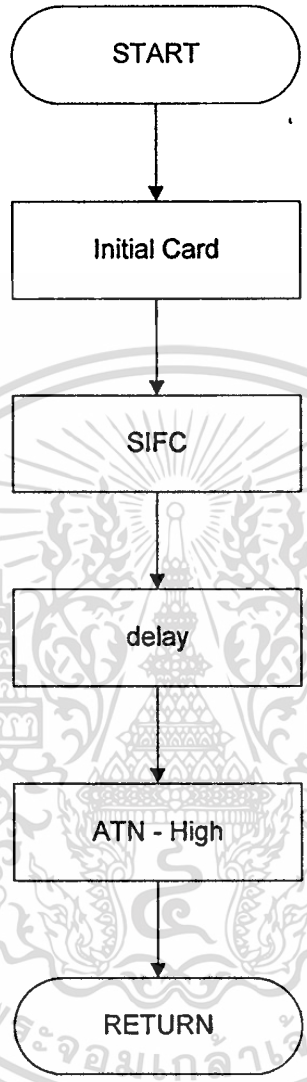


แสดงขั้นตอนการ Initial Controller Card

2. SIFC (Send Interface Clear)

การเรียกใช้ฟังก์ชันนี้จะต้องเริ่มด้วยการ Initial Card เสียก่อน แล้วจึงใช้คำสั่งนี้เพื่อทำการเคลียร์อุปกรณ์ทุกตัวที่ต่ออยู่บนบัสออกจากระบบ เพื่อให้ระบบกลับสู่สถานะการเตรียมพร้อม ขั้นตอนดังกล่าวแสดงในแผนภูมิที่ 26

แผนภูมิที่ 26

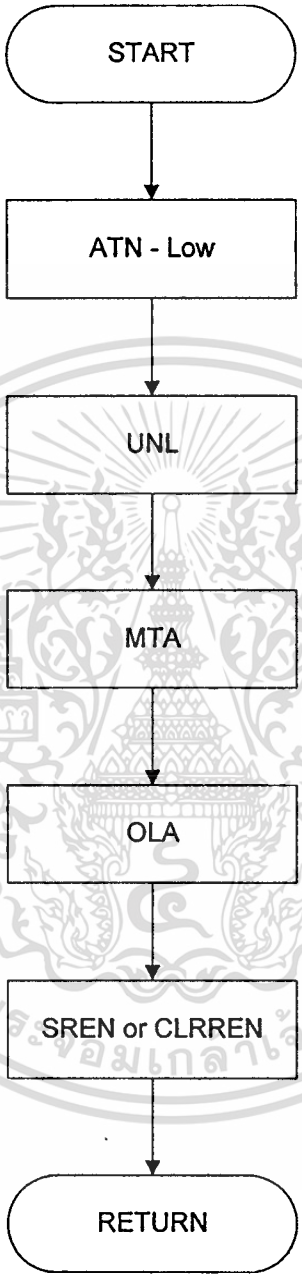


แสดงขั้นตอนการทำงานของฟังก์ชัน SIFC

3. SRE (Send Remote Enable) และ CLRREN (Clear Remote Enable)

ฟังก์ชัน SRE เป็นการสั่งให้อุปกรณ์ที่มีแอดเดรสตามที่กำหนดถูกควบคุมโดยตัวควบคุม และ ฟังก์ชัน CLRREN เป็นการสั่งให้อุปกรณ์ที่มีแอดเดรสตามที่กำหนดออกจากการถูกควบคุมโดยตัวควบคุม มีขั้นตอนดังแสดงในแผนภูมิที่ 27

แผนภูมิที่ 27

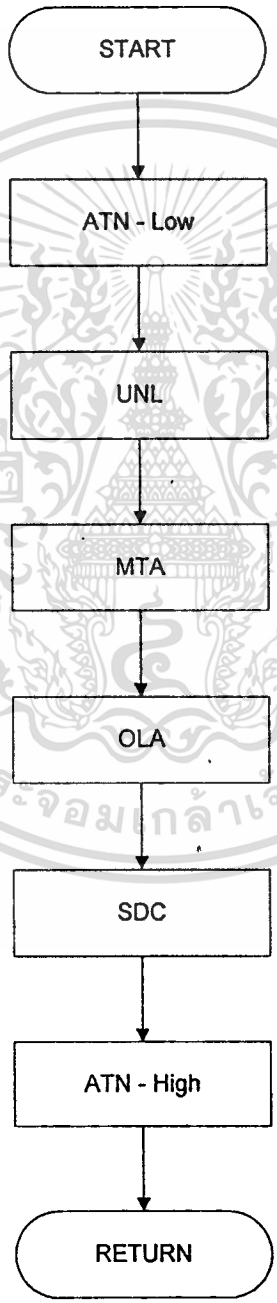


แสดงขั้นตอนการทำงานของฟังก์ชัน SRE และ CLRREN

4. SDC (Select Device Clear)

คำสั่งนี้จะทำการเคลียร์อุปกรณ์ออกจากระบบโดยการกำหนดตัวอุปกรณ์ที่จะทำการเคลียร์ มีขั้นตอนดังแสดงในแผนภูมิที่ 28

แผนภูมิที่ 28

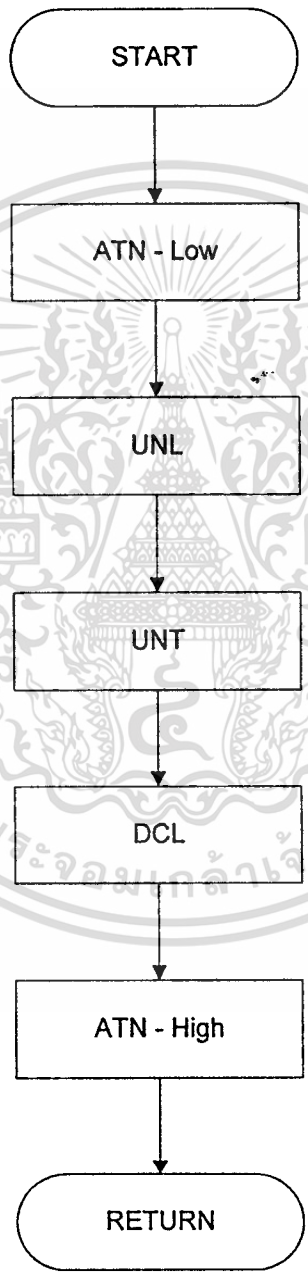


แสดงขั้นตอนการเคลียร์อุปกรณ์บางตัวที่ถูกเลือก

5. DCL (Device Clear)

คำสั่งนี้จะทำการเคลียร์อุปกรณ์ทุกตัวออกจากระบบ มีขั้นตอนดังแสดงในแผนภูมิที่ 29

แผนภูมิที่ 29

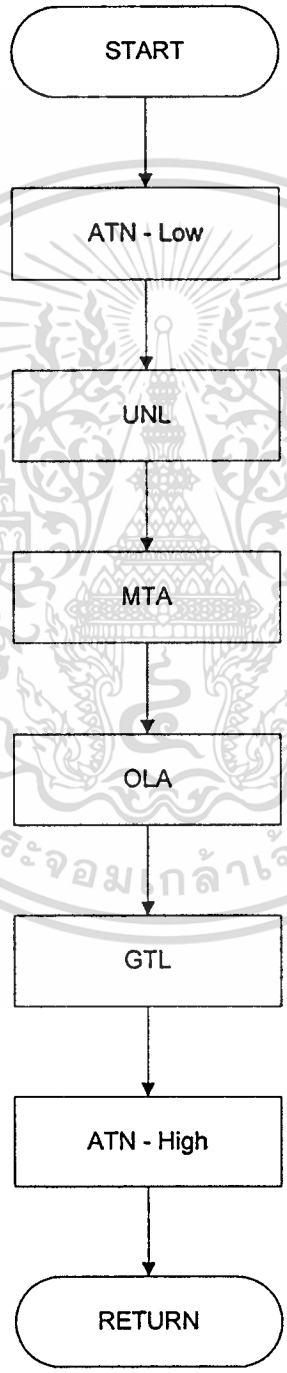


แสดงขั้นตอนการเคลียร์อุปกรณ์ทุกตัวออกจากระบบ

6. GTL (Go To Local)

เป็นการเคลียร์อุปกรณ์ตัวรับทุกตัวออกจากระบบ มีขั้นตอนดังแสดงในแผนภูมิที่ 30

แผนภูมิที่ 30



แสดงขั้นตอนการเคลียร์อุปกรณ์ตัวรับทุกตัวออกจากระบบ

7. การส่งข้อมูลจากตัวควบคุมไปยังอุปกรณ์ที่เป็นตัวรับ

การส่งข้อมูลจากตัวควบคุมไปยังอุปกรณ์ที่เป็นตัวรับ จะต้องผ่านขั้นตอนการติดต่อกับอุปกรณ์เสียก่อน เพื่อเป็นการแจ้งแอดเดรสของตัวควบคุมและอุปกรณ์ที่เป็นตัวรับ แล้วทำการตรวจสอบสถานะของรีจิสเตอร์ Interrupt Status1 ว่าอุปกรณ์ที่เป็นตัวรับพร้อมที่จะทำการส่งรับข้อมูลแล้วหรือยัง โดยการตรวจสอบว่าถ้าบิทที่ 1 (DO) มีค่าเป็น 1 แสดงว่าตัวรับพร้อมที่จะรับข้อมูลแล้ว สามารถส่งข้อมูลลงไปได้เลย แต่ถ้าบิทดังกล่าวยังเป็น 0 ก็ให้ทำการตรวจสอบสถานะบิทดังกล่าวต่อไป ขั้นตอนดังกล่าวสามารถแสดงได้ดังแผนภูมิที่ 31

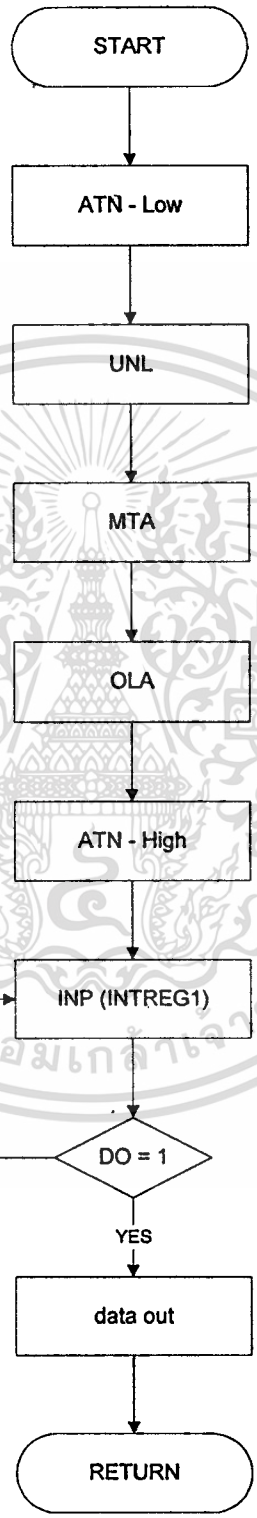
8. การส่งข้อมูลจากอุปกรณ์ที่เป็นตัวส่งไปยังตัวควบคุม

การส่งข้อมูลจากอุปกรณ์ที่เป็นตัวส่งไปยังตัวควบคุม จำเป็นจะต้องมีขั้นตอนการติดต่อระหว่างอุปกรณ์กับตัวควบคุมก่อนเช่นกัน จึงจะสามารถทำการอ่านข้อมูลจากรีจิสเตอร์ได้ โดยจะต้องทำการตรวจสอบสถานะของรีจิสเตอร์ Interrupt Status1 ว่าบิทที่ 0 (DI) มีค่าเป็น 1 หรือไม่ เพื่อแสดงว่าอุปกรณ์ตัวส่งพร้อมที่จะส่งข้อมูลแล้ว ตัวควบคุมซึ่งขณะนี้จะทำหน้าที่เป็นตัวรับสามารถทำการรับข้อมูลได้โดยผ่านรีจิสเตอร์ Data In และทำการตรวจสอบข้อมูลตัวสุดท้ายว่ามีค่าเป็น 0x0D (Carriage Return) หรือไม่ เพื่อเป็นการบอกว่าเป็นการสิ้นสุดของชุดข้อมูลให้ตัวควบคุมทำการหยุดอ่านข้อมูลนั้น ขั้นตอนดังกล่าวแสดงได้ดังแผนภูมิที่ 32

การส่งคำสั่งในการติดต่อกับอุปกรณ์ที่ต่ออยู่ในระบบบัส GPIB นั้น สามารถที่จะใช้รหัสคำสั่งตามมาตรฐาน IEEE-488 หรือจะใช้คำสั่งที่เป็นฟังก์ชันที่มีมาให้ของไอซี uPD7210 ก็ได้ สำหรับคำสั่งอื่นๆ ก็สามารถเรียกใช้งานได้ในลักษณะเดียวกัน

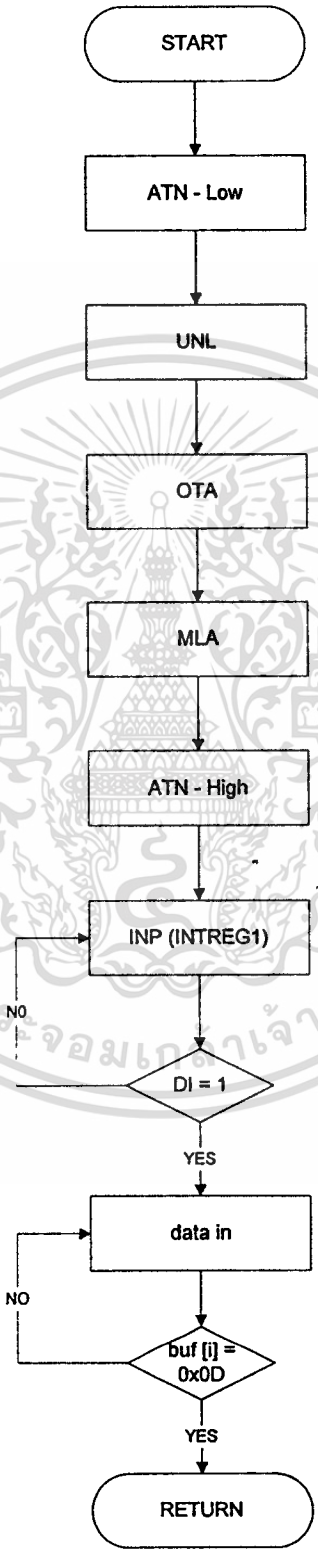
การเขียนโปรแกรมควบคุมอุปกรณ์ในระบบนั้น สามารถแสดงขั้นตอนการเขียนโปรแกรมได้ดังแผนภูมิที่ 33

แผนภูมิที่ 31



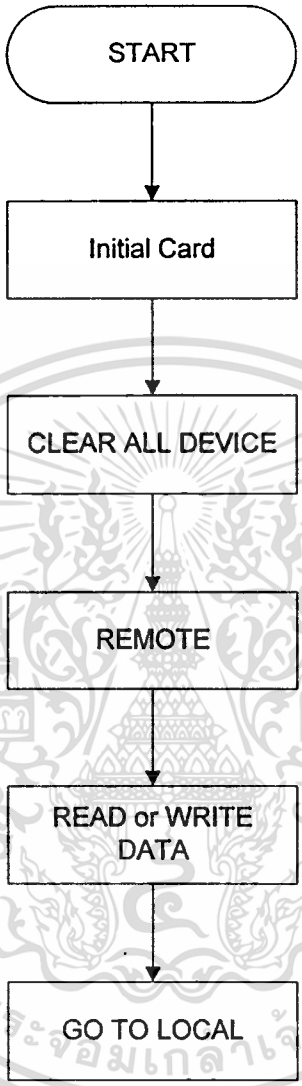
แสดงขั้นตอนการส่งข้อมูลจากตัวควบคุมไปยังอุปกรณ์ที่เป็นตัวรับ

แผนภูมิที่ 32



แสดงขั้นตอนการส่งข้อมูลจากอุปกรณ์ที่เป็นตัวส่งไปยังตัวควบคุม

แผนภูมิที่ 33



แสดงขั้นตอนการติดต่ออุปกรณ์เครื่องมือวัด

การเขียนโปรแกรมหน้าจอสำหรับผู้ใช้งานในการควบคุมอุปกรณ์ในระบบ [4]

การเขียนโปรแกรมควบคุมอุปกรณ์ในระบบบัส GPIB นั้น ได้ทำการพัฒนาโปรแกรมขึ้นโดยใช้ภาษาซี เนื่องจากเป็นภาษาที่มีประสิทธิภาพมาก สามารถสั่งงานได้เร็วกว่าภาษาอื่นๆ เกือบทุกภาษา ยกเว้นภาษาแอสแซมบลี้ และมีความอ่อนตัวในการเขียนโปรแกรมติดต่อกับส่วน

ต่างๆ ของไมโครคอมพิวเตอร์ได้ดี จึงเหมาะสมที่จะใช้งานในด้านการอินเตอร์เฟส สำหรับภาษาซีที่ใช้ในการพัฒนาโปรแกรมนั้นเป็นมาตรฐานของ ANSI C สำหรับการเขียนโปรแกรมหน้าจอที่ใช้ควบคุมระบบเครื่องมือวัดนั้น โปรแกรมส่วนนี้ได้พัฒนาขึ้นโดยใช้โปรแกรม LabWindows / CVI เนื่องจากว่าโปรแกรมดังกล่าวสามารถรองรับโปรแกรมภาษาซีบนมาตรฐานของ ANSI C และยังมีฟังก์ชันต่างๆ ให้เลือกใช้อีกมาก โดยเฉพาะส่วนกราฟิกที่ใช้ในการแสดงอินพุทเอาท์พุทของโปรแกรมและส่วนกราฟิกที่ใช้สร้างหน้าต่างในการควบคุมระบบเครื่องมือวัด

ในระบบเครื่องมือวัดที่ทำการวิจัยนั้น ประกอบด้วยอุปกรณ์ทั้งหมด 4 อุปกรณ์ คือ

1. Power Supply
2. Function Generator
3. Digital Mutimeter
4. Storage Oscilloscope

การออกแบบหน้าต่างควบคุมระบบเครื่องมือวัด ได้ทำการออกแบบให้ผู้ใช้งานสามารถควบคุมอุปกรณ์ที่ต่ออยู่ในระบบได้พร้อมกัน โดยกำหนดให้อุปกรณ์ทำการรับคำสั่งในการควบคุมการทำงานฟังก์ชันต่างๆ จากช่องรับข้อมูลที่ออกแบบขึ้น สำหรับหน้าต่างควบคุมระบบเครื่องมือวัดที่ทำการออกแบบแสดงได้ดังภาพที่ 34

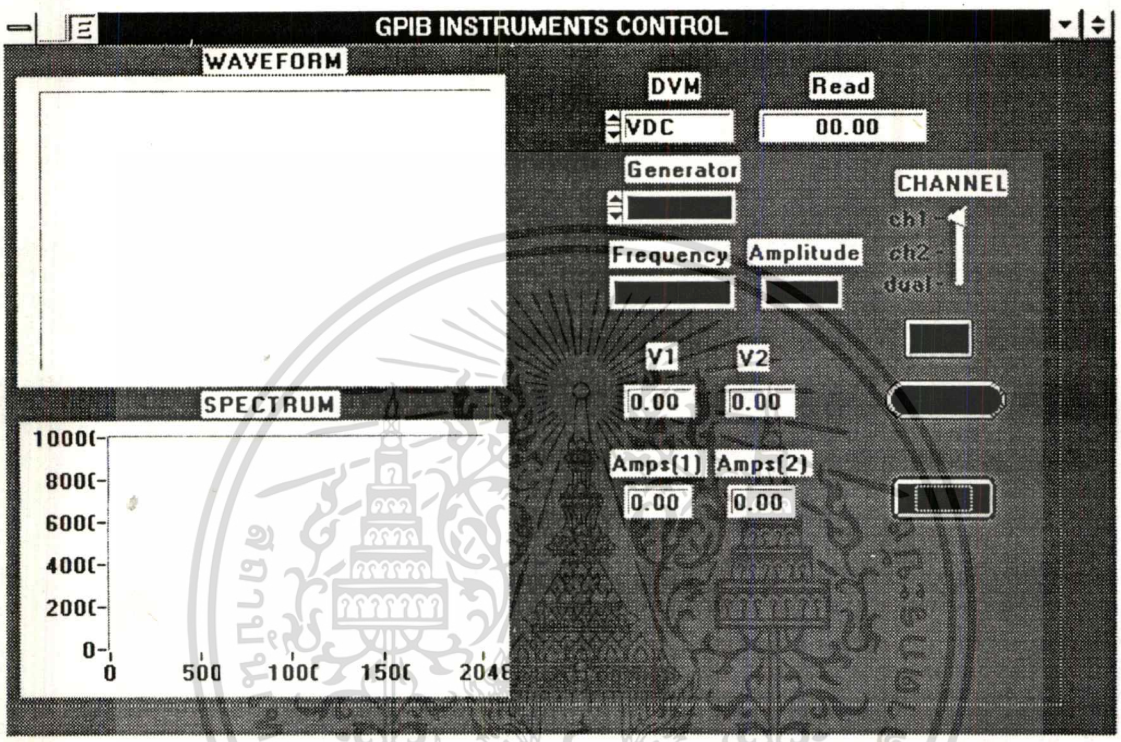
ในหน้าต่างที่ใช้ควบคุมระบบเครื่องมือวัด สามารถแบ่งได้เป็นส่วนการควบคุมอุปกรณ์ที่ต่ออยู่ในระบบเครื่องมือวัด ซึ่งประกอบด้วยอุปกรณ์ทั้ง 4 ตัวดังได้กล่าวข้างต้น และส่วนที่เป็นการประยุกต์ใช้งานในการวิเคราะห์หาสเปคตรัมของสัญญาณที่วัดได้จาก Storage Oscilloscope ซึ่งแต่ละส่วนจะประกอบด้วยส่วนประกอบต่างๆ ดังนี้ คือ

1 ส่วนที่ควบคุม Digital Mutimeter ประกอบด้วยช่องรับข้อมูล 1 ช่องและช่องแสดงผลค่าที่วัดได้จาก Digital Multimeter อีก 1 ช่อง โดยสามารถแยกคุณสมบัติของช่องรับข้อมูลและช่องแสดงผล คือ

ชื่อ	DVM
รูปแบบ	Edit Ring
หน้าที่	ส่งข้อมูล เพื่อใช้ในการควบคุมฟังก์ชันการทำงานต่างๆ ของ Digital Multimeter
การกำหนดค่า	เลือกฟังก์ชันที่ต้องการจากข้อมูลที่กำหนดไว้
ชื่อ	Read
รูปแบบ	Edit String

หน้าที่ รับข้อมูล เพื่อแสดงผลค่าที่อ่านได้จาก Digital Multimeter

ภาพที่ 34



แสดงหน้าต่างควบคุมระบบเครื่องมือวัดที่ทำการออกแบบ

2 ส่วนที่ควบคุม Function Generator ประกอบด้วยช่องรับข้อมูล 3 ช่อง และมีคุณสมบัติ คือ

ชื่อ Generator

รูปแบบ Edit Ring

หน้าที่ ส่งข้อมูล เพื่อใช้ในการควบคุมฟังก์ชันการกำเนิดสัญญาณต่างๆ ของ Function Generator

การกำหนดค่า เลือกฟังก์ชันที่ต้องการจากข้อมูลที่กำหนดไว้

ชื่อ Frequency

รูปแบบ Edit String

หน้าที่ ส่งข้อมูล เพื่อใช้ในการกำหนดความถี่ของสัญญาณ

การกำหนดค่า กำหนดค่าโดยการป้อนค่าความถี่ที่ต้องการ

ชื่อ Amplitude

รูปแบบ Edit String

หน้าที่ ส่งข้อมูล เพื่อใช้ในการกำหนดแอมปลิจูดของสัญญาณ

การกำหนดค่า กำหนดค่าโดยการป้อนค่าแอมปลิจูดที่ต้องการ

3 ส่วนที่ควบคุม Power Supply ประกอบด้วยช่องรับข้อมูล 4 ช่อง และมีคุณสมบัติ คือ

ชื่อ V1

รูปแบบ Edit String

หน้าที่ ส่งข้อมูล เพื่อกำหนดค่าศักดาไฟฟ้าของแหล่งจ่ายไฟตัวที่ 1

การกำหนดค่า กำหนดค่าโดยการป้อนค่าศักดาไฟฟ้าที่ต้องการ

ชื่อ V2

รูปแบบ Edit String

หน้าที่ ส่งข้อมูล เพื่อกำหนดค่าศักดาไฟฟ้าของแหล่งจ่ายไฟตัวที่ 2

การกำหนดค่า กำหนดค่าโดยการป้อนค่าศักดาไฟฟ้าที่ต้องการ

ชื่อ Amps(1)

รูปแบบ Edit String

หน้าที่ ส่งข้อมูล เพื่อกำหนดค่ากระแสไฟฟ้าของแหล่งจ่ายไฟตัวที่ 1

การกำหนดค่า กำหนดค่าโดยการป้อนค่ากระแสไฟฟ้าที่ต้องการ

ชื่อ Amps(2)

รูปแบบ Edit String

หน้าที่ ส่งข้อมูล เพื่อกำหนดค่ากระแสไฟฟ้าของแหล่งจ่ายไฟตัวที่ 2

การกำหนดค่า กำหนดค่าโดยการป้อนค่ากระแสไฟฟ้าที่ต้องการ

4 ส่วนที่ควบคุม Storage Oscilloscope ประกอบด้วยช่องรับข้อมูล 1 ช่อง และช่องแสดงผล 1 ช่อง มีคุณสมบัติ คือ

ชื่อ CHANNEL

รูปแบบ Edit Ring Slide

หน้าที่ ส่งข้อมูล เพื่อกำหนดช่องสัญญาณของ Storage Oscilloscope ที่จะทำการอ่านค่าข้อมูลที่จะนำมาแสดงผล

การกำหนดค่า เลือกฟังก์ชันที่ต้องการจากข้อมูลที่กำหนดไว้

ชื่อ WAVEFORM

รูปแบบ Edit Graph

หน้าที่ แสดงผลของข้อมูลที่อ่านเข้ามาจากช่องสัญญาณต่างๆ ในรูปกราฟ

5 ส่วนที่เป็นการประยุกต์ใช้งาน เป็นการวิเคราะห์หาสเปคตรัมของสัญญาณที่วัดได้จาก Storage Oscilloscope โดยใช้วิธี Fast Fourier Transform ซึ่งในส่วนนี้จะประกอบด้วยช่องแสดงผล 1 ช่องและมีคุณสมบัติ คือ

ชื่อ SPECTRUM

รูปแบบ Edit Graph

หน้าที่ แสดงสเปคตรัมของสัญญาณที่ทำการอ่านค่ามาจาก Storage Oscilloscope

การใช้งานโปรแกรมควบคุมระบบเครื่องมือวัด จะมีลักษณะเป็นการติดต่อระหว่างผู้ใช้งานกับเครื่องมือวัดที่ต่ออยู่ในระบบผ่านหน้าต่างที่ทำการควบคุม การทำงานของโปรแกรมควบคุมนั้นจะรับข้อมูลที่เป็นคำสั่งต่างๆ ในการควบคุมอุปกรณ์จากช่องรับข้อมูลบนหน้าต่าง เพื่อทำการส่งข้อมูลลงไปบนบัสของระบบ และการสั่งงานโปรแกรมจะใช้เมาส์เป็นตัวช่วยกำหนดว่าช่องรับข้อมูลใดที่จะทำการรับข้อมูล โปรแกรมสำหรับควบคุมระบบเครื่องมือวัดนี้สามารถแสดงขั้นตอนต่างๆ ได้ดังแผนภาพในแผนภูมิที่ 35

แผนภูมิที่ 35

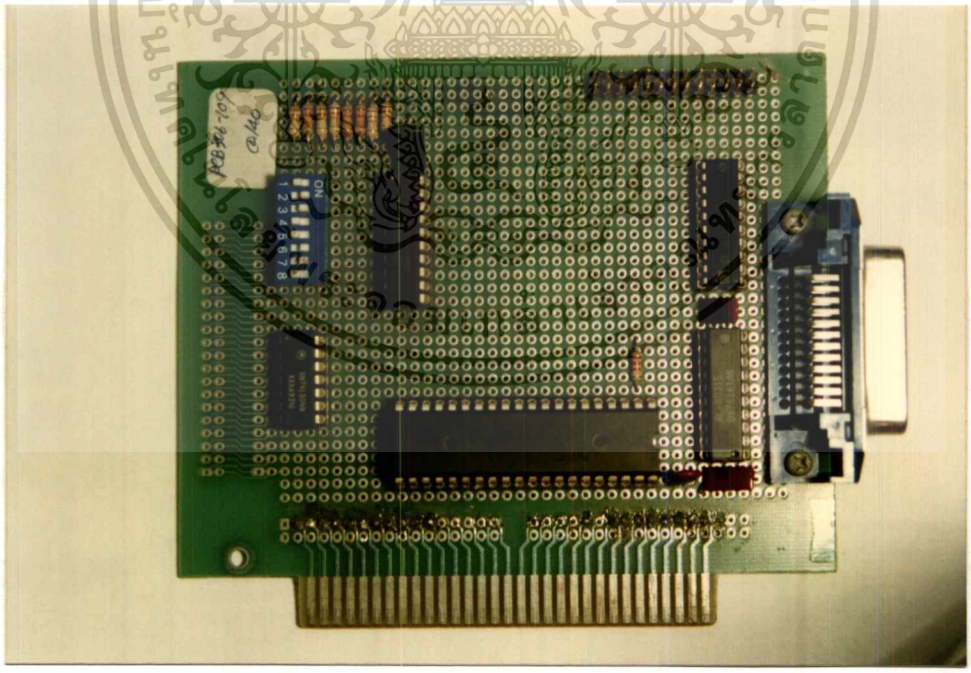
แสดงขั้นตอนต่างๆ ของโปรแกรมสำหรับควบคุมระบบเครื่องมือวัด

บทที่ 6
การทดสอบและการประยุกต์ใช้งาน

กล่าวนำ

บทนี้จะแสดงถึงการทดสอบระบบเครื่องมือวัดที่ โดยจะทำการทดลองควบคุมอุปกรณ์ที่ต่ออยู่ในระบบบัส IEEE-488 (GPIB) ผ่านการ์ดอินเตอร์เฟสที่สร้างขึ้น โดยใช้โปรแกรมควบคุมระบบเครื่องมือวัดที่พัฒนาขึ้นและผลการประยุกต์ใช้งานโปรแกรมดังกล่าว ซึ่งการ์ดที่สร้างขึ้นแสดงดังภาพที่ 36

ภาพที่ 36

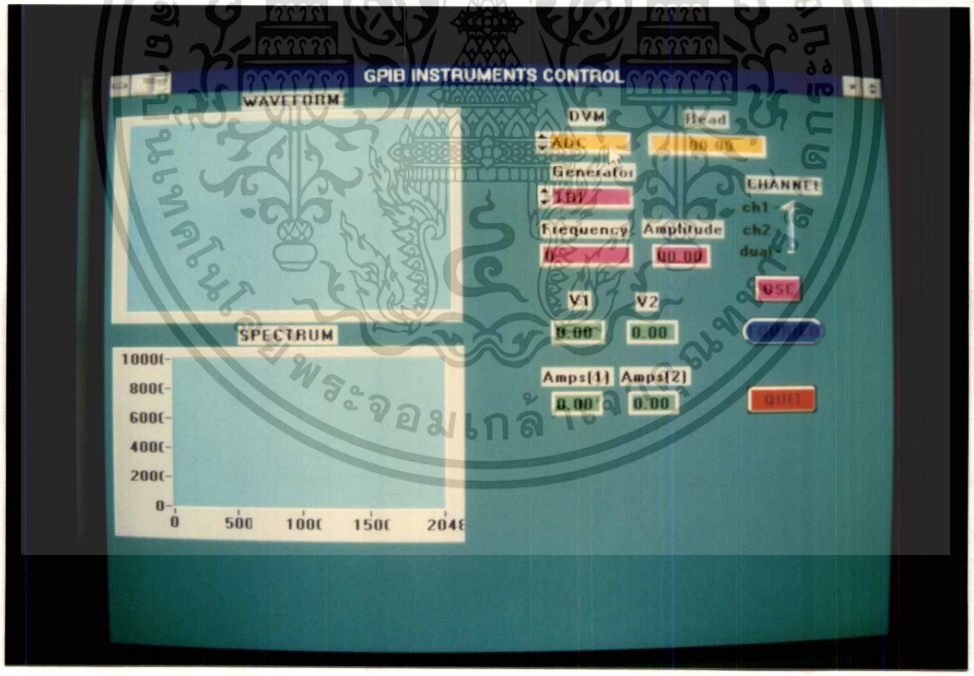


แสดงการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 (GPIB) ที่สร้างขึ้น

ผลการทดสอบการควบคุม Digital Multimeter

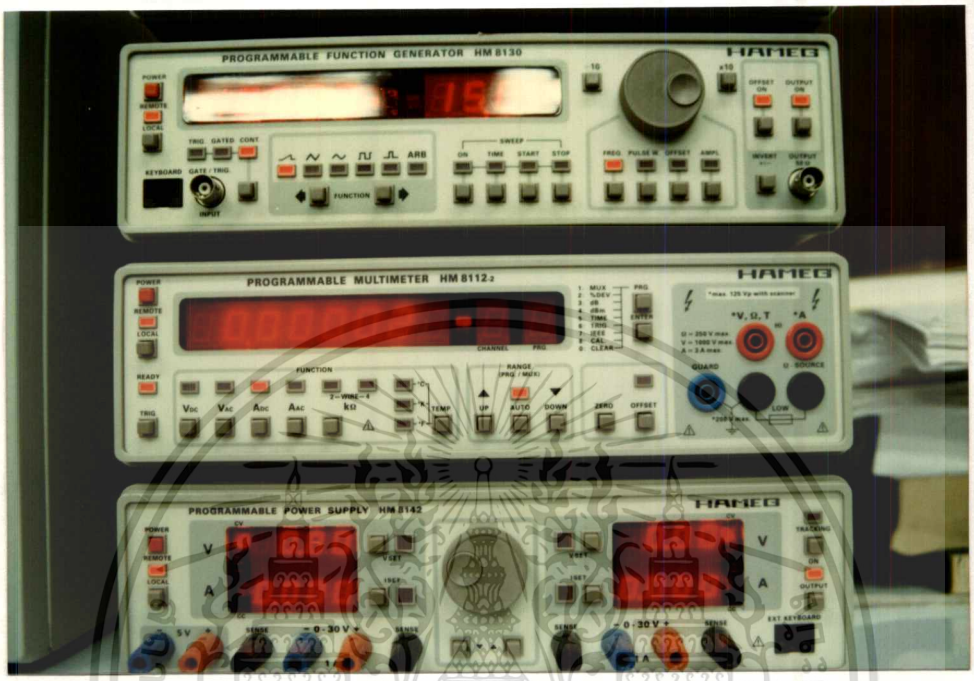
ในการทดลองได้ทำการควบคุม Digital Multimeter ให้ทำงานตามฟังก์ชันการวัดต่างๆ โดยการกำหนดฟังก์ชันการทำงานจากหน้าจอสำหรับการควบคุมระบบเครื่องมือวัด จะเห็นได้ว่าสามารถที่จะควบคุม Digital Multimeter ให้ทำงานตามฟังก์ชันการทำงานที่ควบคุมได้ ซึ่งการกำหนดฟังก์ชันการทำงานของ Digital Multimeter แสดงไว้ในภาพที่ 37 และผลจากการส่งคำสั่งลงไปควบคุมแสดงดังภาพที่ 38 ผลการทดสอบจะเห็นว่า สามารถที่จะควบคุมการทำงานของ Digital Multimeter ให้ทำงานตามฟังก์ชันที่กำหนดได้อย่างถูกต้องและการแสดงค่าที่ได้จากการวัดก็สามารถทำได้อย่างถูกต้องเช่นกัน ดังแสดงในภาพที่ 39

ภาพที่ 37



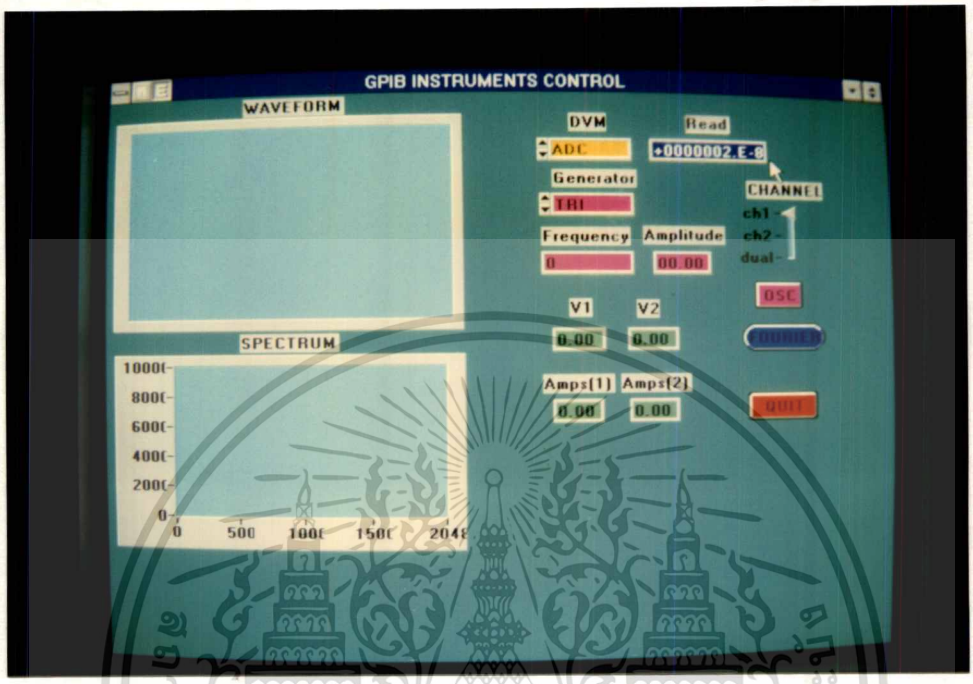
แสดงการกำหนดฟังก์ชันการทำงานของ Digital Multimeter

ภาพที่ 38



แสดงฟังก์ชันการทำงานของ Digital Multimeter หลังจากได้รับการควบคุม
จากโปรแกรมควบคุมระบบ

ภาพที่ 39

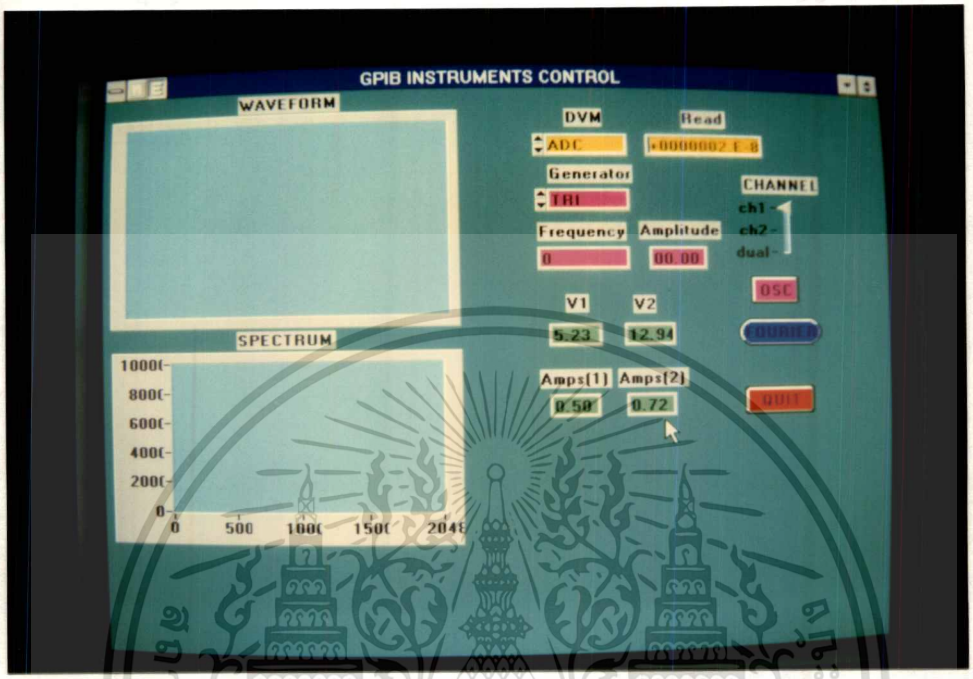


แสดงค่าที่อ่านได้จาก Digital Multimeter บนหน้าจอควบคุม

ผลการทดสอบการควบคุม Power Supply

การทดลองควบคุม Power Supply ได้ทำการทดลองส่งคำสั่งควบคุมให้แหล่งจ่ายศักดาไฟฟ้าและกระแสไฟฟ้าตัวที่ 1 และ 2 ให้จ่ายศักดาไฟฟ้าและกระแสไฟฟ้าตามที่กำหนด จากการทดลองจะเห็นว่าสามารถควบคุมการทำงานของ Power Supply ให้ทำงานได้ตามต้องการ แต่สำหรับค่ากระแสไฟฟ้าที่กำหนดของแหล่งจ่ายกระแสไฟฟ้าทั้งสอง เครื่องจะไม่สามารถแสดงให้เห็นได้ในโหมดรีโมท แต่สามารถที่จะตรวจสอบความถูกต้องของการกำหนดค่ากระแสไฟฟ้าได้ในโหมด Local ผลการทดสอบแสดงได้ดังภาพที่ 40, 41 และ 42

ภาพที่ 40



แสดงหน้าจอควบคุมให้ Power Supply ทำงานตามที่กำหนด

ภาพที่ 41

แสดงหน้าปัทม์ของ Power Supply หลังจากได้รับคำสั่งควบคุม
จากโปรแกรมควบคุมระบบ

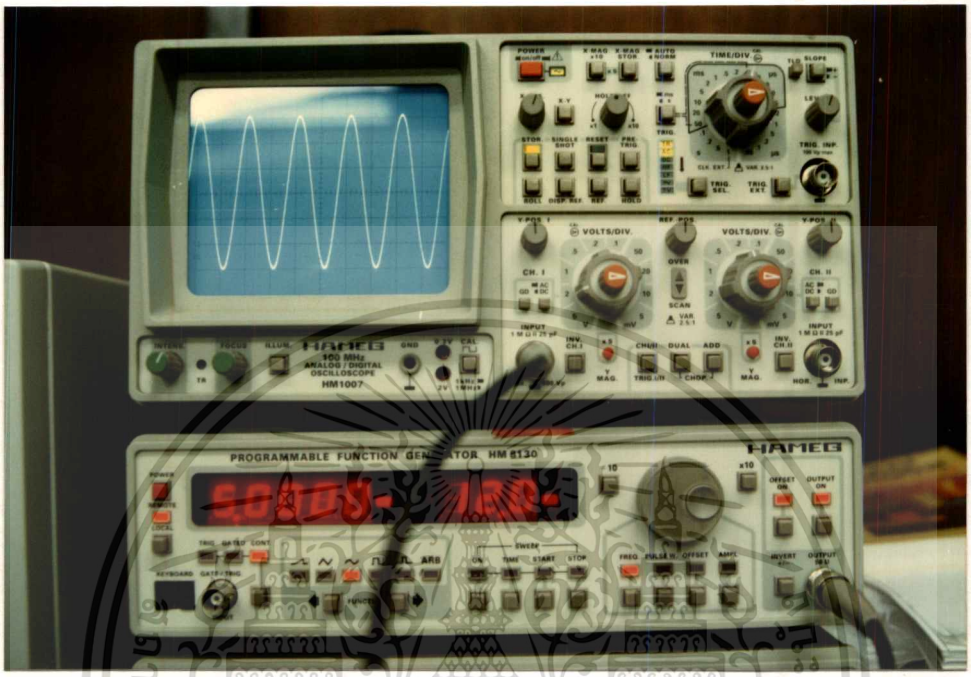
ภาพที่ 42

แสดงค่ากระแสไฟฟ้าของ Power Supply ที่ถูกกำหนดจากโปรแกรมควบคุมระบบ

ผลการทดสอบการควบคุม Storage Oscilloscope

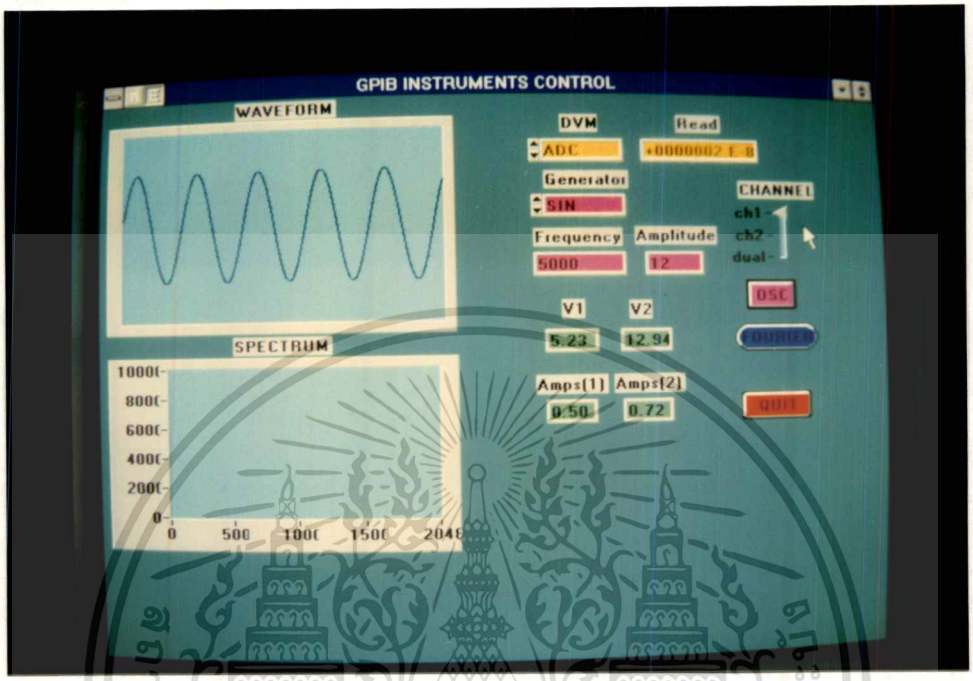
การทดสอบการควบคุม Storage Oscilloscope เป็นการอ่านข้อมูลจาก Channel ต่างๆ ของ Storage Oscilloscope เพื่อนำมาแสดงบนหน้าต่างที่ใช้ทำการควบคุม ผลการทดสอบนั้นสามารถที่จะแสดงรูปคลื่นสัญญาณได้เหมือนสัญญาณที่แสดงบนจอของ Storage Oscilloscope ดังแสดงในภาพที่ 43 และ 44

ภาพที่ 43



แสดงหน้าจอของ Storage Oscilloscope และผลการควบคุม Function Generator

ภาพที่ 44

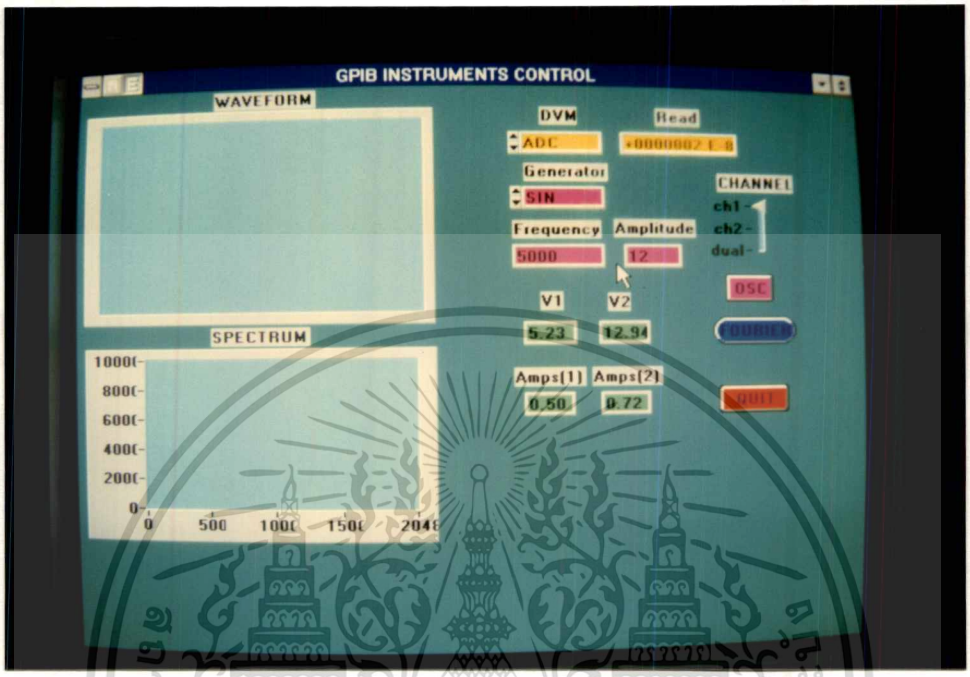


แสดงหน้าต่างที่แสดงสัญญาณที่ได้จาก Storage Oscilloscope

ผลการทดสอบการควบคุม Function Generator

การทดลองควบคุม Function Generator ได้ทำการทดลองควบคุม Function Generator ให้ทำงานตามฟังก์ชันที่ต้องการ โดยการควบคุมผ่านหน้าต่างที่ใช้ควบคุมระบบวัดเครื่องมือวัด จะเห็นได้ว่าสามารถที่จะควบคุม Function Generator ให้ทำงานได้ตามฟังก์ชันที่กำหนด ดังจะเห็นจากผลการทดสอบที่แสดงในภาพที่ 45 และ 43

ภาพที่ 45

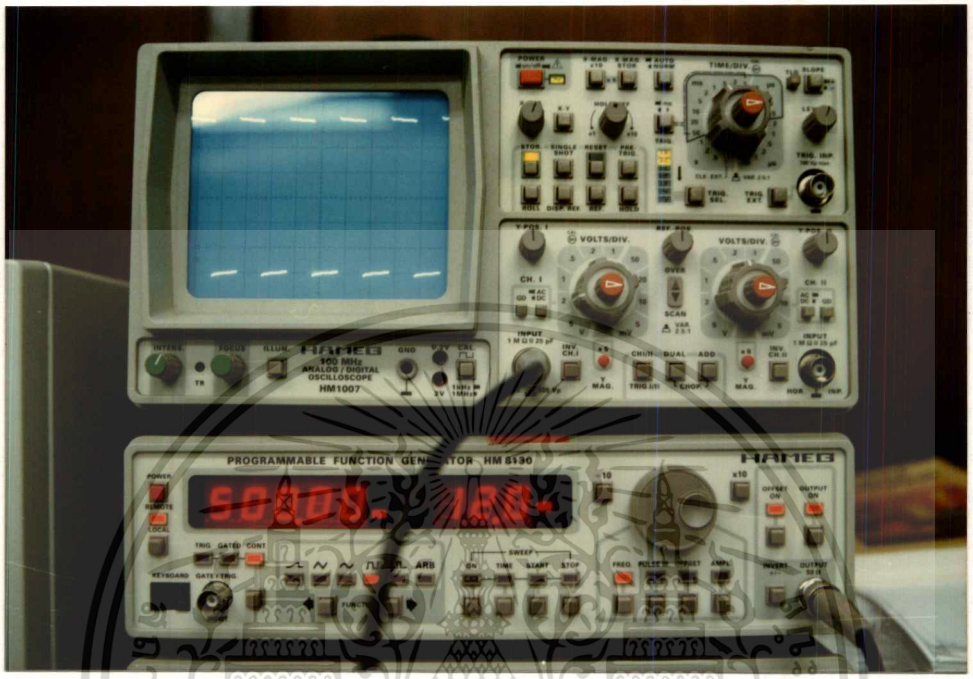


แสดงหน้าต่างที่ทำการควบคุม Function Generator

ผลการทดสอบการประยุกต์ใช้งาน

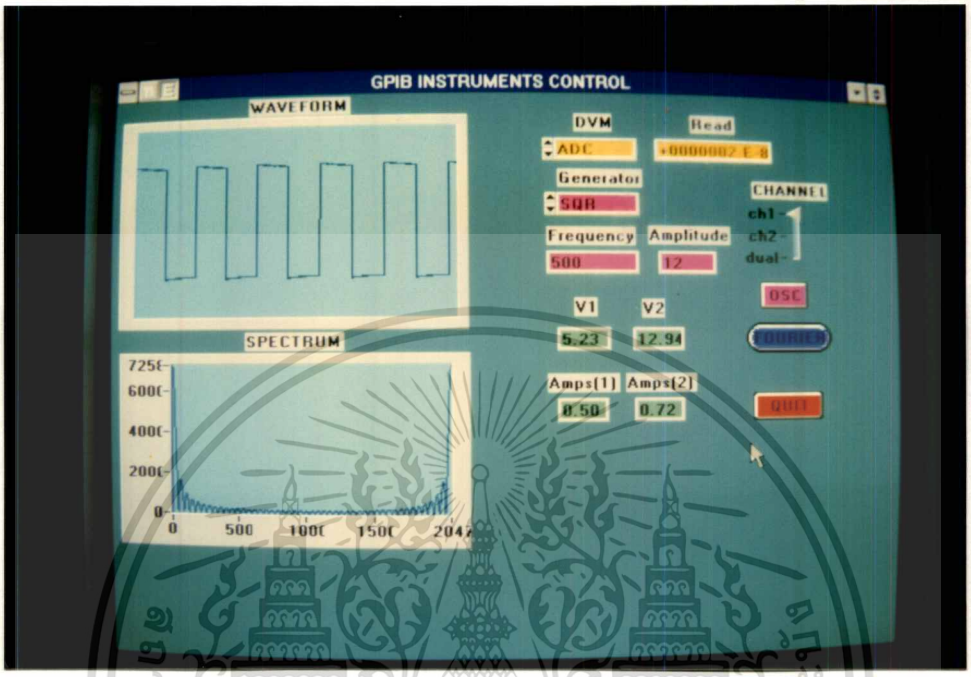
ในการทดสอบการประยุกต์ใช้งานเป็นการนำเอาสัญญาณที่วัดได้ Storage Oscilloscope มาทำการวิเคราะห์หาสเปคตรัมของสัญญาณโดยวิธี FFT (Fast Fourier Transform) ซึ่งฟังก์ชัน FFT นี้เป็นฟังก์ชันที่มีมาใน Lab Windows/CVI อยู่แล้ว สามารถที่จะเรียกใช้ได้เลย และหลังจากที่นำสัญญาณที่อ่านมาได้ทำการวิเคราะห์หาสเปคตรัมของสัญญาณแล้ว จึงจะนำค่าของชุดข้อมูลที่ได้จากการวิเคราะห์มาพล็อตแสดงสเปคตรัมของสัญญาณดังแสดงในภาพที่ 46 และ 47

ภาพที่ 46



แสดงสัญญาณบนหน้าจอของ Storage Oscilloscope

ภาพที่ 47

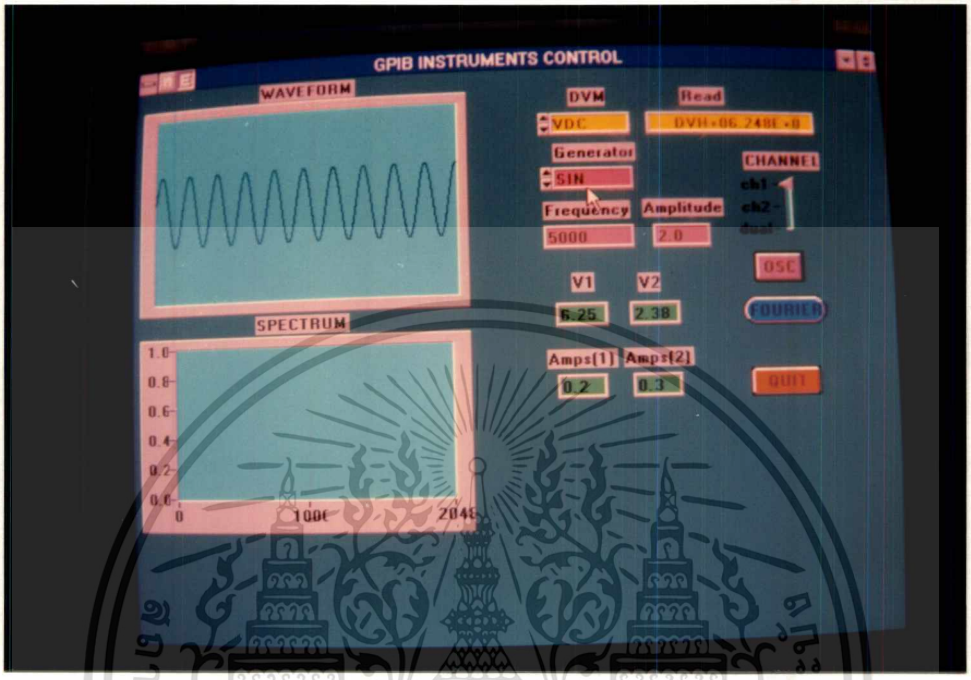


แสดงสเปคตรัมของสัญญาณที่ได้จากการวิเคราะห์สัญญาณแบบ FFT

การทดสอบการต่ออุปกรณ์ต่างบริษัทร่วมกันในระบบ

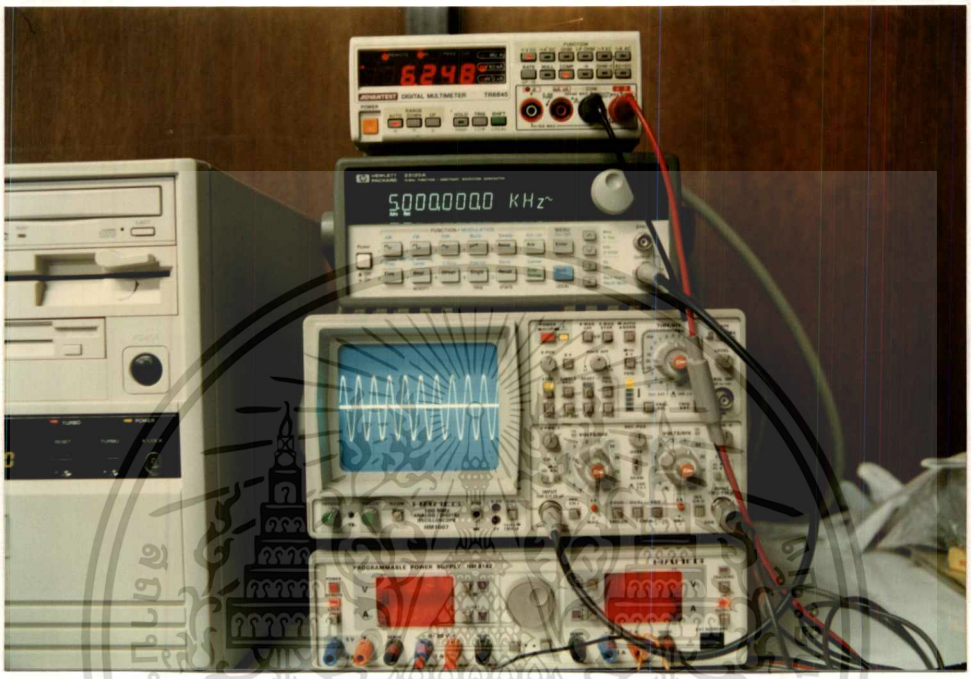
การทดสอบการต่ออุปกรณ์ต่างบริษัทร่วมกันในระบบควบคุมเครื่องมือวัดที่ได้พัฒนาขึ้น ได้ทำการเปลี่ยนอุปกรณ์ใหม่คือ Digital Multimeter รุ่น TR6845 ของบริษัท ADVANTEST และ Function Generator รุ่น 33120A ของบริษัท Hewlett Packard ในการควบคุมระบบเครื่องมือวัดนั้น ต้องทำการกำหนดแอดเดรสของอุปกรณ์ทั้งสองให้ตรงกับโปรแกรมควบคุมก่อน และต้องทำการแก้ไขโปรแกรมส่วนที่เป็นคำสั่งในการควบคุมฟังก์ชันการทำงานของแต่ละอุปกรณ์ด้วย เนื่องจากว่าคำสั่งในการควบคุมฟังก์ชันการทำงานของอุปกรณ์แต่ละบริษัทนั้นไม่เหมือนกัน การทดสอบควบคุมระบบเครื่องมือวัดดังกล่าวโดยใช้การ์ดอินเตอร์เฟสและโปรแกรมควบคุมที่พัฒนาขึ้นนั้น แม้ว่าในระบบที่ทำการทดสอบจะประกอบด้วยอุปกรณ์ที่ต่างบริษัทกัน แต่ก็สามารถที่จะควบคุมให้ทำงานได้อย่างถูกต้องโดยการควบคุมผ่านหน้าต่างที่ใช้ควบคุมระบบวัดเครื่องมือวัดเช่นกัน ดังจะเห็นได้จากผลการทดสอบที่แสดงในรูปที่ 48 และรูปที่ 49

ภาพที่ 48



แสดงการกำหนดฟังก์ชันการทำงานของอุปกรณ์ทั้งสอง
ผ่านหน้าต่างที่ใช้ควบคุมเครื่องมือวัด

ภาพที่ 49



แสดงผลการควบคุมระบบเครื่องมือวัดที่ประกอบด้วยอุปกรณ์ที่ต่างบริษัทกัน

สรุป

จากการทดลองในการควบคุมอุปกรณ์ต่างๆ ในระบบเครื่องมือวัดบนระบบบัสมาตรฐาน IEEE-488 (GPIB) ผ่านการ์ดอินเตอร์เฟสที่ทำการสร้างขึ้น จะเห็นว่าสามารถที่จะควบคุมอุปกรณ์เครื่องมือวัดที่ต่ออยู่ในระบบให้ทำงานได้เป็นไปอย่างถูกต้องโดยการควบคุมผ่านหน้าต่างสำหรับการควบคุม ซึ่งการ์ดอินเตอร์เฟสและโปรแกรมที่พัฒนาขึ้นมาสามารถใช้งานได้เป็นอย่างดี

บทที่ 7
บทสรุปและแนวทางพัฒนา

การติดต่อสื่อสารระหว่างคอมพิวเตอร์กับเครื่องมือวัดต่างๆ ในทางอุตสาหกรรมนั้น ส่วนใหญ่จะนิยมใช้ระบบบัสมาตรฐาน IEEE-488 (GPPB) เป็นมาตรฐานในการติดต่อ เนื่องจากการติดต่อสื่อสารผ่านบัสดังกล่าวมีความเร็วสูงและสามารถเชื่อมต่ออุปกรณ์ได้หลายตัวพร้อมกันในระบบ ซึ่งการติดต่อสื่อสารระหว่างคอมพิวเตอร์ที่ทำหน้าที่เป็นตัวควบคุมระบบและเครื่องมือวัดที่ต่ออยู่ในระบบจำเป็นต้องมีการ์ดอินเตอร์เฟสเพื่อใช้ในการสื่อสารข้อมูลระหว่างกัน โดยการ์ดดังกล่าวจะทำหน้าที่เปลี่ยนระบบบัสจาก PCAT เป็นระบบบัสแบบ IEEE-488 (GPPB) ซึ่งเป็นระบบบัสแบบขนาน

วิทยานิพนธ์ฉบับนี้ได้กล่าวถึงการศึกษาออกแบบบ และสร้างการ์ดอินเตอร์เฟสตามมาตรฐาน IEEE-488 และทำการพัฒนาโปรแกรมควบคุมระบบขึ้น เพื่อให้ง่ายและสะดวกต่อการใช้งาน โดยในส่วนของการ์ดอินเตอร์เฟสที่ทำการสร้างขึ้นประกอบด้วยวงจร 2 ส่วน คือ วงจรถอดรหัสแอดเดรสจากไมโครคอมพิวเตอร์ และวงจรส่วนควบคุมการอินเตอร์เฟสกับอุปกรณ์ในระบบ รวมถึงโปรแกรมที่ใช้ในการควบคุมอุปกรณ์ในระบบ

ในการทดสอบการทำงานของการ์ดอินเตอร์เฟสและโปรแกรมที่พัฒนาขึ้น ได้ทำการทดสอบโดยการส่งคำสั่งต่างๆ ลงไปควบคุมอุปกรณ์ที่ต่ออยู่ในระบบให้ทำงานตามคำสั่งที่ส่งลงไปบนบัสของระบบ ซึ่งผลที่ได้แสดงให้เห็นว่าการ์ดอินเตอร์เฟสที่สร้างขึ้นและโปรแกรมที่พัฒนาขึ้นสามารถใช้งานได้เป็นอย่างดี และในส่วนของการประยุกต์ใช้งานเป็นการวิเคราะห์หาสเปคตรัมของสัญญาณโดยวิธี FFT (Fast Fourier Transform) ซึ่งสัญญาณที่ได้จากการวิเคราะห์สามารถนำไปใช้งานในการประมวลสัญญาณได้ต่อไป

งานวิจัยนี้สามารถนำไปพัฒนาเพื่อใช้กับการวัดคุณสมบัติของอุปกรณ์อิเลกทรอนิกส์ เช่น ทรานซิสเตอร์ ฯลฯ ได้ หรือจะนำไปใช้วัดการตอบสนองความถี่ของลำโพงหรือเครื่องขยายสัญญาณได้ รวมทั้งอาจจะนำไปประยุกต์ใช้งานในด้านการแพทย์ก็ได้ โดยการนำเอาสัญญาณที่ได้จากร่างกายผู้ป่วยมาทำการวิเคราะห์หาสเปคตรัมของสัญญาณ เพื่อที่จะนำข้อมูลที่ได้ไปใช้ในการวินิจฉัยทางการแพทย์ หรือนำไปประยุกต์ใช้ในห้องปฏิบัติการด้านเซมิคอนดักเตอร์ ในการวัดคุณสมบัติของตัวอุปกรณ์ต่างๆ ที่สร้างขึ้น เป็นต้น

บรรณานุกรม

[1] กนก เจนจิระพงศ์เวช. "GPIB (IEEE 488) บัสอินเตอร์เฟส มาตรฐานและการใช้" วารสารเซมิฯ. ฉบับที่ 78 (พฤษภาคม,สิงหาคม,กันยายน 2530): 214-218, 201-207, 196-201.

[2] ธานินทร์ ถาวรศาสนวงศ์, ทินกร ดุ๊ก. "การอินเตอร์เฟส IBM PC" กรุงเทพฯ. ฟิสิกส์เซ็นเตอร์การพิมพ์, ม.ป.ป.

[3] Eugene Fisher, C.W. Jensen. "PET/CBM and the IEEE 488 Bus (GPIB)" Berkley. Osborne McGraw-Hill, Inc., 1982.

[4] National Instruments Corporation. "LabWindows/CVI User Interface Reference Manual" Austin. 1994.

ภาคผนวก

ภาคผนวก ก

โปรแกรมควบคุมระบบเครื่องมือวัด

ภาคผนวก ก

โปรแกรมควบคุมระบบเครื่องมือวัด

```
/* FILE  IOTEST1.C */
#include <ansi_c.h>
#include <utility.h>
#include <gpib.h>
#include <stdio.h>
#include <stdlib.h>
#include <userint.h>
#include <math.h>
#include <string.h>
#include <iotest1.h>
#include <analysis.h>

#define PortAdd      0x340
#define DATAIN       PortAdd
#define DATAOUT      PortAdd
#define INTREG1      PortAdd+1
#define INTREG2      PortAdd+2
#define SPREG        PortAdd+3
#define ADSTREG      PortAdd+4
#define ADMDREG      PortAdd+4
#define CPTREG       PortAdd+5
#define ACMD         PortAdd+5
#define AUXREG       PortAdd+5
#define ADRREG       PortAdd+6
#define ADR0REG      PortAdd+6
```

```
#define ADR1REG   PortAdd+7
#define EOSREG    PortAdd+7
#define DIPSW     PortAdd+8

#define GTL       0x01     /* GPIB go to local */
#define SDC       0x04     /* GPIB selected device clear */
#define PPC       0x05     /* GPIB parallel poll configure */
#define GET       0x08     /* GPIB group execute trigger */
#define LLO       0x11     /* GPIB local lock out */
#define DCL       0x14     /* GPIB device clear */
#define PPU       0x15   /* GPIB parallel poll unconfigure */
#define SPE       0x18     /* GPIB serial poll enable */
#define SPD       0x19     /* GPIB serial poll disable */
#define UNL       0x3f
#define UNT       0x5f
#define IEP         0      /* 7210 Immedate Execute pon */
#define SPPFO     0x01     /* 7210 Set Parallel Flag */
#define CRST      0x02   /* 7210 Chip reset */
#define FINHS     0x03     /* 7210 Finish Handshake */
#define SEOI      0x06     /* 7210 Sent Eoi */
#define SPPF1     0x09     /* 7210 Set Parallel Pol Flag */
#define GTSTB     0x10     /* 7210 Goto Standby */
#define TCA       0x11     /* 7210 Take Control Asynchonously */
#define LISN      0x13     /* 7210 Listen */
#define CLRIFC    0x16     /* 7210 Clear IFC */
#define CLRREN    0x17     /* 7210 Clear REN */
#define EPP       0x1D     /* 7210 Execute Parallel Poll */
#define SIFC      0x1E     /* 7210 Set IFC */
#define SREN      0x1F     /* 7210 Set REN    */
```

```
struct INBITREG {
/**** Data In ****/
    unsigned bdatain : 8;
/** Interupt Status 1 **/
    unsigned di  : 1;
    unsigned dto  : 1;
    unsigned err  : 1;
    unsigned dec  : 1;
    unsigned end  : 1;
    unsigned det  : 1;
    unsigned apt  : 1;
    unsigned cpt  : 1;

/* Interrupt Status2 */
    unsigned adsc : 1;
    unsigned remc : 1;
    unsigned lokc : 1;
    unsigned co  : 1;
    unsigned rem  : 1;
    unsigned lok  : 1;
    unsigned srqi : 1;
    unsigned intt : 1;

/* Serial  Poll Staurs*/
    unsigned spstatus: 6;
    unsigned pend   : 1;
    unsigned s8     : 1;

/* Address Status */
```

```
    unsigned mjmn  : 1;
    unsigned ta    : 1;
    unsigned la    : 1;
    unsigned tpass : 1;
    unsigned lpass : 1;
    unsigned spms  : 1;
    unsigned atn   : 1;
    unsigned cic   : 1;

/* Command Pass Through */
    unsigned bcpt  : 8;

/*  Address 0  */
    unsigned dt0   : 2;
    unsigned dl0   : 1;
    unsigned ad0   : 5;

/*  Address 1  */
    unsigned eoi   : 1;
    unsigned dt1   : 1;
    unsigned dl1   : 1;
    unsigned ad1   : 5;
    };

struct INBYTEREG
   {
    unsigned char direg  ;
    unsigned char is1reg  ;
    unsigned char is2reg  ;
```

```
    unsigned char spsreg  ;
    unsigned char adsreg  ;
    unsigned char cptreg  ;
    unsigned char adr0reg ;
    unsigned char adr1reg ;
   };

union IGREGS
   {
    struct INBITREG  ib ;
    struct INBYTEREG iy ;
   };

struct OUTBITREG
   {
/* Byte Out */
    unsigned byteout :8;

/* Interupt Mask 1 */
    unsigned di      : 1;
    unsigned dto     : 1;
    unsigned err     : 1;
    unsigned dec     : 1;
    unsigned end     : 1;
    unsigned det     : 1;
    unsigned apt     : 1;
    unsigned cpt     : 1;

/* Interupt Mask 2 */
```

```
    unsigned adsc   : 1;
    unsigned remc   : 1;
    unsigned lokc   : 1;
    unsigned co     : 1;
    unsigned dmai   : 1;
    unsigned dmao   : 1;
    unsigned srqi   : 1;
    unsigned intt   : 1;

/* Serial Poll Mode */
    unsigned spstatus: 6;
    unsigned rsv    : 1;
    unsigned s8     : 1;

/* Address Mode */
    unsigned adm0   : 1;
    unsigned adm1   : 1;
    unsigned adxx   : 2;
    unsigned trm0   : 1;
    unsigned trm1   : 1;
    unsigned lon    : 1;
    unsigned ton    : 1;

/* Auxiliary Mode */
    unsigned auxm   : 8;

/* Address 0/1 */
    unsigned ad1    : 5;
    unsigned dl     : 1;
```

```
    unsigned dt      : 1;
    unsigned ars     : 1;

/* End of String */
    unsigned eos     : 8;
   };

struct OUTBYTEREG
   {
    unsigned char boreg  ;
    unsigned char im1reg  ;
    unsigned char im2reg  ;
    unsigned char spmreg  ;
    unsigned char admreg  ;
    unsigned char auxmreg ;
    unsigned char adr01reg;
    unsigned char eosreg  ;
   };

union OGREGS
   {
    struct OUTBITREG  ob;
    struct OUTBYTEREG oy;
   };

struct AUXBIT
   {
/* Auxiliary A Register */
    unsigned ahs: 2;
```

```
    unsigned a2 : 1;
    unsigned a3 : 1;
    unsigned a4 : 1;
    unsigned aa : 3;

/* Auxiliary B Register */
    unsigned b0 : 1;
    unsigned b1 : 1;
    unsigned b2 : 1;
    unsigned b3 : 1;
    unsigned b4 : 1;
    unsigned ab : 3;

/* Auxiliary E Register */
    unsigned e0 : 1;
    unsigned e1 : 1;
    unsigned ae : 6;

/* Auxiliary F Register */
    unsigned nf : 4;
    unsigned af : 4;

/* Auxiliary P Register */
    unsigned pa : 3;
    unsigned s  : 1;
    unsigned u  : 1;
    unsigned ap : 3;
  };
struct AUXBYTE
```

```
  {
   unsigned char areg;
   unsigned char breg;
   unsigned char ereg;
   unsigned char freg;
   unsigned char preg;
  };
union AUXREGS
  {
   struct AUXBIT  ab;
   struct AUXBYTE ay;
  };
union IGREGS inr ;
union OGREGS outr ;
union AUXREGS aux ;

int Gpib_ifc(void);
int Gpib_ren(void);
int Gpib_write(int, unsigned char*,int);
int Gpib_write2(int, unsigned char*,int);
int Gpib_read(int, unsigned char*);
int Gpib_wrt(unsigned char ) ;
int Gpib_cmd(int, char*);
int Gpib_spoll(int, char*, int*, int*);
int Gpib_sqr(int, int);
int Gpib_ppr(int);
int Gpib_pp2(int, int, int);
int Gpib_spst(void);
int Gpib_idy(void);
```

```
int Gpib_clrren(void);

int InstallMainCallback (MainCallbackPtr eventFunction, void *callbackData, int getIdleEvents);
int ProcessLoop (int panel, int control, int event, void *callbackData, int eventData1,
                 int eventData2);

int temp=0;
int device_num[20];
int dvmstatus;
short func=1;
char Data[20];
char rd[20],count_byte;
int dipsw,master,myadr,mylsna,mytka,delim;
int channel;
int handle,p;
unsigned char buf[2048];
static double buff[2048];
int status = 0;
char comd[7];
int i;
main()
{
        Cls();
        InstallMainCallback (ProcessLoop, 0, 1);
        SetIdleEventRate (0);
        initGPIB();
        Gpib_ifc();
        Gpib_ren();
        comd[0] = UNL;
```

```
            comd[1] = mytka;
            comd[2] = 0x20+12;
            comd[3] = 0x20+11;
            comd[4] = 0x20+8;
            comd[5] = 0x20+7;
            Gpib_cmd(6,comd);

            handle = LoadPanel (0, "iotest1.uir", PANEL);
            DisplayPanel (handle);
            RunUserInterface ();
}

int ProcessLoop (int panel, int control, int event, void *callbackData, int eventData1, int
                 eventData2)
{
    int i;
            unsigned char buf[2050];
            unsigned char buff[4098];
            char ch;
            int err=0;
    if(event == EVENT_IDLE)
    {
      if(status == 1)
      {
       switch (channel) {
            case 0 :{
                    comd[0] = UNL;
                            comd[1] = mytka;
                            comd[2] = 0x20+11;
```

```
                    Gpib_cmd(3,comd);
            Gpib_write(5,"DIG 1",1);
                    for (p=1; p<20000; p++)
            {
                comd[0] = UNL;
            }
                for (p=1; p<20000; p++)
                  {
              comd[0] = UNL;
                  }
                for (p=1; p<20000; p++)
                  {
              comd[0] = UNL;
                  }
                comd[0] = UNL;
                comd[1] = 0x40+11;
            comd[2] = myIsna;
                Gpib_cmd(3,comd);
                i = Gpib_read(2050,buf);
                comd[0] = UNT;
                Gpib_cmd(1,comd);
        if((buf[i-2]==0x0d)||(buf[i-2]==0x0a)) buf[i-2]==NULL; else
        if((buf[i-1]==0x0d)||(buf[i-1]==0x0a)) buf[i-1]==NULL;
        DeleteGraphPlot (handle, PANEL_GRAPH, -1, 1);
        PlotY (handle, PANEL_GRAPH, buf, 2048, VAL_UNSIGNED_CHAR, VAL_THIN_LINE,
                    VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_BLACK);
} break;
        case 1 :{
                    for (i=0;i<10;i++) {
```

```
                comd[0] = UNL;
                        comd[1] = mytka;
                        comd[2] = 0x20+11;
                        Gpib_cmd(3,comd);
                Gpib_write(5,"DIG 2",1);
                        for (p=1; p<20000; p++)
                    {
                      comd[0] = UNL;
                    }
                   for (p=1; p<20000; p++)
                    {
                      comd[0] = UNL;
                    }
                   comd[0] = UNL;
                   comd[1] = 0x40+11;
                   comd[2] = mylsna;
                   Gpib_cmd(3,comd);
                   i = Gpib_read(2050,buf);
                   comd[0] = UNT;
                   Gpib_cmd(1,comd);
        if((buf[i-2]==0x0d)||(buf[i-2]==0x0a)) buf[i-2]==NULL; else
        if((buf[i-1]==0x0d)||(buf[i-1]==0x0a)) buf[i-1]==NULL;
        DeleteGraphPlot (handle, PANEL_GRAPH, -1, 1);
        PlotY (handle, PANEL_GRAPH, buf, 2048, VAL_UNSIGNED_CHAR, VAL_THIN_LINE,
                        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_BLUE);
                        }
} break;
        case 2 : {
                 comd[0] = UNL;
```

```
                comd[1] = mytka;
                comd[2] = 0x20+11;
                Gpib_cmd(3,comd);
        Gpib_write(5,"DIG 3",1);
                for (p=1; p<20000; p++)
            {
              comd[0] = UNL;
            }
          for (p=1; p<20000; p++)
            {
              comd[0] = UNL;
            }
          for (p=1; p<20000; p++)
            {
              comd[0] = UNL;
            }
          comd[0] = UNL;
          comd[1] = 0x40+11;
          comd[2] = mylsna;
          Gpib_cmd(3,comd);
          i = Gpib_read(4096,buff);
          for (p=1; p<20000; p++)
            {
              comd[0] = UNL;
            }
          comd[0] = UNT;
          Gpib_cmd(1,comd);
    if((buff[i-2]==0x0d)||(buff[i-2]==0x0a)) buff[i-2]==NULL; else
    if((buff[i-1]==0x0d)||(buff[i-1]==0x0a)) buff[i-1]==NULL;
```

```
            DeleteGraphPlot (handle, PANEL_GRAPH, -1, 1);
            PlotY (handle, PANEL_GRAPH, buff, 4096, VAL_UNSIGNED_CHAR, VAL_THIN_LINE,
                                VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
     } break;
    }
   }
  } return(0);
}

int OSC(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
   if(event == EVENT_COMMIT)
   {
     status = 0;
   }
   return(0);
}

void initGPIB(void)   /*  start edit pass  */
{
  int err=0;
  delim = 0;
  outr.ob.eos= 0x0a;
  dipsw = inp(DIPSW);
  if((dipsw & 0x80)!=0) err=1;
  master= (dipsw & 0x20) >> 5;
  master= master^1;
  myadr = (dipsw & 0x1f);
  mylsna= 0x20+myadr;
```

```
    mytka = 0x40+myadr;
    outp (ACMD,0x02);   /* change CRST = 0x02  */
    outr.oy.admreg = 0;
    outr.ob.trm0   = 1;
    outr.ob.trm1   = 1;
    outr.ob.adm0   = 1;
    outp(ADMDREG,outr.oy.admreg);
    outr.oy.adr01reg=0;
    outr.ob.ad1 = myadr;
    outp(ADRREG,outr.oy.adr01reg);
    outr.ob.ars = 1;
    outr.ob.dt  = 1;
    outr.ob.dl  = 1;
    outp(ADRREG,outr.oy.adr01reg);
    aux.ay.areg = 0x94;
    outp(AUXREG,aux.ay.areg);
    aux.ay.breg = 0xA4;
    outp(AUXREG,aux.ay.breg);
    aux.ay.ereg = 0x0C;
    outp(AUXREG,aux.ay.ereg);
    aux.ay.freg = 0x28;
    outp(AUXREG,aux.ay.freg);
    aux.ay.preg = 0x60;
    outp(EOSREG,outr.ob.eos);
    outp(INTREG1,0);
    outp(INTREG2,0);
    outp(ACMD,0x00);   /* change IEP = 0  */
}
```

```
int Gpib_ifc()     /* remark n in ()  */
 /* int n;  */
 {
 int ts,tem;
  if(master == 0)
   {
    outp(ACMD,SIFC);
            for ( ts=0; ts<500 ; ts++)
                { tem=1;
                };
    outp(ACMD,CLRIFC);
    outp(ACMD,GTSTB);
   }
  return(master);
}

/* Remote Enable */
int Gpib_ren()
{
 if(master == 0) outp(ACMD,SREN);
 return(master);
}

 /*Write 2 Command*/
int Gpib_write2(int n,unsigned char *buf,int eoi)
{
 unsigned char c;
 int i, j, k;
if(delim == 3)
```

```
{
 j = n-1;
 c = buf[n-1];
}
else
 {
  j = n;
  c = outr.ob.eos;
 }
outp(ACMD,0x1c);
for(i=0;i<j;i++)
{
        k = Gpib_wrt(buf[i]);
  if(k == 1) goto tag;
}
tag : ;
return(k);
}

/*Read Command*/
int Gpib_read(n,buf)
int n;
unsigned char buf[];
{
 int i=0;
 aux.ab.ahs = 0 ;
 outp(AUXREG,aux.ay.areg);
 do
  {
```

```
   inr.iy.is1reg = inp(INTREG1);
   if((inr.iy.is1reg & 0x11) != 0)
    {
     buf[i] =inp(DATAIN);
     if(inr.ib.end == 1)
      if((inp(ADR1REG) & 0x80) == 0)
           if(delim == 0)
            if(buf[i-1] != 0x0D) inr.ib.end =0;
     i++;
    }
  }while ((inr.ib.end == 0) && (i<n));
 return(i);
}

int Gpib_write(int n,unsigned char *buf,int eoi)
/*int n, eoi;
unsigned char buf[];                */
{
 unsigned char c;
 int i, j, k;
 if(delim == 3)
  {
   j = n-1;
   c = buf[n-1];
  }
  else
   {
    j = n;
    c = outr.ob.eos;
```

```
  }
outp(ACMD,0x1c);
for(i=0;i<j;i++)
 {
  k = Gpib_wrt(buf[i]);
  if(k == 1) goto tag;
 }
if(delim == 0) k = Gpib_wrt(0x0D);
if(k == 1)goto tag;
if(eoi ==1) outp(ACMD,SEOI);
k = Gpib_wrt(c);
tag : ;
return(k);
}

int Gpib_cmd(int n,char *buf)
/*int n;
char buf[];*/
{
 int i;
 unsigned char c;
 if(master == 0)
  {
   outp(ACMD,TCA);
   for(i=0;i<n;i++)
    {
     c = buf[i];
     if(c == mylsna) outp(ACMD,LISN);
      else
```

```
        {
         do
          {
           inr.iy.is2reg = inp (INTREG2);
           if(inr.ib.co ==1) outp(DATAOUT,c);
          }while(inr.ib.co == 0);
        }
    }
   outp(ACMD,GTSTB);
  }
 return(master);
}

int Gpib_wrt(unsigned char c)
{
 int err;
 do
  {
   inr.iy.is1reg = inp(INTREG1);
    if(inr.ib.dto == 1) outp(DATAOUT,c);
     else
      if(inr.ib.err == 1)
        {
         err=1;
         goto tag ;
        }
  } while(inr.ib.dto == 0);
 err =0;
 tag : ;
```

```
  return(err);
 }

/* uir function */
int  QUIT(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    if (event == EVENT_COMMIT)
    {
        status = 0;
        QuitUserInterface(0);
     }
    return(0);
}

int  FUNC(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
            int function;
            if (event == EVENT_COMMIT) {
        GetCtrlVal (handle, PANEL_FUNCTION, &function);
            comd[0] = UNL;
            comd[1] = mytka;
            comd[2] = 0x20+7;
            Gpib_cmd(3,comd);
        Gpib_write(2,"A1",1);
        switch (function) {
                case 0:         Gpib_write(3,"VD",1);
                        break;
                case 1:         Gpib_write(3,"VA",1);
                        break;
```

```
            case 2:         Gpib_write(3,"ID",1);
                    break;
            case 3:         Gpib_write(3,"IA",1);
                    break;
            case 4:         Gpib_write(3,"O2",1);
                    break;
            case 5:         Gpib_write(3,"O4",1);
                    break;
            case 6:         Gpib_write(3,"TC",1);
                    break;
            case 7:         Gpib_write(3,"TK",1);
                    break;
            case 8:         Gpib_write(3,"TF",1);
                    break;
            case 9:         Gpib_write(3,"A1",1);
                    break;
          }
    } return (0);
}

int  GENE(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    int generate;
          if (event == EVENT_COMMIT) {
       GetCtrlVal (handle, PANEL_GENERATE, &generate);
          comd[0] = UNL;
          comd[1] = mytka;
          comd[2] = 0x20+12;
          Gpib_cmd(3,comd);
```

```
Gpib_write(3,"OT1",1);
switch (generate) {
        case 0:         Gpib_write(3,"TRI",1);
                break;
        case 1:         Gpib_write(3,"RMP",1);
                break;
        case 2:         Gpib_write(3,"SIN",1);
                break;
        case 3:         Gpib_write(3,"SQR",1);
                break;
        case 4:         Gpib_write(3,"PLS",1);
                break;
        case 5:         Gpib_write(3,"ARB",1);
                break;
        case 6:         Gpib_write(3,"SW1",1);
                break;
        case 7:         Gpib_write(3,"DSW",1);
                break;
        case 8:         Gpib_write(3,"DST",1);
                break;
        case 9:         Gpib_write(3,"DSP",1);
                break;
        case 10:        Gpib_write(3,"DFR",1);
                break;
        case 11:        Gpib_write(3,"DWT",1);
                break;
        case 12:        Gpib_write(3,"DOF",1);
                break;
        case 13:        Gpib_write(3,"DAM",1);
```

```
                break;
        case 14:        Gpib_write(3,"OF1",1);
                break;
        }
    } return (0);
}

int  FREQ(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
        int frequency;
        char data[20];
        if (event == EVENT_COMMIT) {
        comd[0] = UNL;
        comd[1] = mytka;
        comd[2] = 0x20+12;
        Gpib_cmd(3,comd);
      GetCtrlVal (handle, PANEL_FREQUENCY, data);
      Gpib_write2(4,"FRQ:",1);
      Gpib_write(strlen(data), data, 15);
    } return (0);
}

int  CHAN(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{

        if (event == EVENT_COMMIT)
        {
          status = 1;
      GetCtrlVal (handle, PANEL_CHANNEL, &channel);
```

```
    } return (0);
}

int  VOLT1(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
            char volt1[20];
            if (event == EVENT_COMMIT) {
           comd[0] = UNL;
            comd[1] = mytka;
            comd[2] = 0x20+8;
            Gpib_cmd(3,comd);
        GetCtrlVal (handle, PANEL_VOLT1, volt1);
        Gpib_write2(4,"SU1 ",1);
        Gpib_write(strlen(volt1), volt1,15);
        Gpib_write(3,"OP1",1);
    } return (0);
}

int  VOLT2(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
            char volt2[20];
            if (event == EVENT_COMMIT) {
           comd[0] = UNL;
           comd[1] = mytka;
           comd[2] = 0x20+8;
           Gpib_cmd(3,comd);
        GetCtrlVal (handle, PANEL_VOLT2, volt2);
        Gpib_write2(4,"SU2 ",1);
        Gpib_write(strlen(volt2), volt2,15);
```

```
        Gpib_write(3,"OP1",1);
    } return (0);
}

int  AMPS1(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
            char amps1[20];
            if (event == EVENT_COMMIT) {
           comd[0] = UNL;
           comd[1] = mytka;
           comd[2] = 0x20+8;
           Gpib_cmd(3,comd);
       GetCtrlVal (handle, PANEL_AMPS1, amps1);
       Gpib_write2(4,"SI1 ",1);
       Gpib_write(strlen(amps1), amps1,15);
       Gpib_write(3,"OP1",1);
    } return (0);
}

int  AMPS2(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
            char amps2[20];
            if (event == EVENT_COMMIT) {
           comd[0] = UNL;
           comd[1] = mytka;
           comd[2] = 0x20+8;
           Gpib_cmd(3,comd);
       GetCtrlVal (handle, PANEL_AMPS2, amps2);
       Gpib_write2(4,"SI2 ",1);
```

```
        Gpib_write(strlen(amps2), amps2,15);
        Gpib_write(3,"OP1",1);
    } return (0);
}

int  AMPL(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
    char ampl[20];
            if (event == EVENT_COMMIT) {
            comd[0] = UNL;
            comd[1] = mytka;
            comd[2] = 0x20+12;
            Gpib_cmd(3,comd);
        GetCtrlVal (handle, PANEL_AMPLITUDE, ampl);
        Gpib_write2(4,"AMP:",1);
        Gpib_write(strlen(ampl), ampl,15);
    } return (0);
}

int  READ(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
            char buf[2050];
            float temp;
            unsigned char value[2050];
            if (event == EVENT_COMMIT) {
            comd[0] = UNL;
            comd[1] = 0x40+7;
            comd[2] = mylsna;
            Gpib_cmd(3,comd);
```

```
            i = Gpib_read(2050, buf);
            comd[0] = UNT;
            Gpib_cmd(1,comd);
            if((buf[i-2]==0x0d)||(buf[i-2]==0x0a)) buf[i-2]==NULL; else
            if((buf[i-1]==0x0d)||(buf[i-1]==0x0a)) buf[i-1]==NULL;
        SetCtrlVal (handle, PANEL_READ, buf);
        } return (0);
}

int  FOUR(int panel, int control, int event, void *callbackData, int eventData1, int eventData2)
{
        static double buff[2048];
        static double y[2048];
        static double abs[2048];
            comd[0] = UNL;
            comd[1] = mytka;
            comd[2] = 0x20+11;
            Gpib_cmd(3,comd);
        Gpib_write(5,"DIG 1",1);
            for (p=1; p<20000; p++)
              {
               comd[0] = UNL;
              }
            for (p=1; p<20000; p++)
              {
               comd[0] = UNL;
              }
               comd[0] = UNL;
               comd[1] = 0x40+11;
```

```
            comd[2] = mylsna;
            Gpib_cmd(3,comd);
            i = Gpib_read(2048,buf);
            comd[0] = UNT;
            Gpib_cmd(1,comd);
        if((buf[i-2]==0x0d)||(buf[i-2]==0x0a)) buf[i-2]==NULL; else
        if((buf[i-1]==0x0d)||(buf[i-1]==0x0a)) buf[i-1]==NULL;
        for(i=0;i<2048;i++){
      buff[i] =  (double) buf[i]  ;
                    }
  for(i=0;i<2048;i++)
    {
     y[i] = 0;
    }
     FFT (buff, y, 2048);
  for(i=0;i<2048;i++)
    {
        abs [i] = sqrt ( (buff[i] * buff[i]) + (y[i] * y[i]) );
    }
        DeleteGraphPlot (handle, PANEL_SPECTRUM, -1, 1);
        PlotY (handle, PANEL_SPECTRUM, abs, 2048, VAL_DOUBLE, VAL_THIN_LINE,
                        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_BLUE);
  return (0);
}
```

```
/* FILE  IOTEST.H */
/*****************************************************************************/
/* LabWindows/CVI User Interface Resource (UIR) Include File                 */
/* Copyright (c) National Instruments 1993. All Rights Reserved.             */
/*                                                                           */
/* WARNING: Do not add to, delete from, or otherwise modify the contents     */
/*          of this include file.                                            */
/*****************************************************************************/
/* Panels and Controls: */

#define  PANEL                  1
#define  PANEL_quit             2       /* callback function: QUIT */
#define  PANEL_GRAPH            3
#define  PANEL_CHANNEL          4       /* callback function: CHAN */
#define  PANEL_FUNCTION         5       /* callback function: FUNC */
#define  PANEL_GENERATE         6       /* callback function: GENE */
#define  PANEL_FREQUENCY        7       /* callback function: FREQ */
#define  PANEL_VOLT1            8       /* callback function: VOLT1 */
#define  PANEL_AMPS2            9       /* callback function: AMPS2 */
#define  PANEL_AMPS1            10      /* callback function: AMPS1 */
#define  PANEL_VOLT2            11      /* callback function: VOLT2 */
#define  PANEL_AMPLITUDE        12      /* callback function: AMPL */
#define  PANEL_READ             13      /* callback function: READ */
#define  PANEL_FOURIER          14      /* callback function: FOUR */
#define  PANEL_OSC              15      /* callback function: OSC */
#define  PANEL_SPECTRUM         16     /* Menu Bars, Menus, and Menu Items: */
        /* (no menu bars in the resource file) */
```

```
/* Callback Prototypes: */

int  AMPL(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  AMPS1(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  AMPS2(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  CHAN(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  FOUR(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  FREQ(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  FUNC(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  GENE(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  OSC(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  QUIT(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  READ(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  VOLT1(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
int  VOLT2(int panel, int control, int event, void *callbackData, int eventData1, int eventData2);
```

ประวัติผู้เขียน

นายนพดล มณีรัตน์ เกิดเมื่อวันที่ 11 พฤษภาคม พ.ศ. 2511 ที่จังหวัดนครปฐม สำเร็จการศึกษาปริญญาวิทยาศาสตรบัณฑิต สาขาฟิสิกส์ จากมหาวิทยาลัยศรีนครินทรวิโรฒ พิษณุโลก เมื่อปีพ.ศ. 2533 เข้าศึกษาต่อระดับปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2535 มีผลงานทางวิชาการที่ได้รับการยอมรับ นอกจากงานวิทยานิพนธ์ จำนวน 2 เรื่อง ได้แก่ เรื่องเครื่องตรวจสอบสัญญาณระบบบัสตามมาตรฐาน IEEE-488 ลงตีพิมพ์วิศวสารลาดกระบัง, การออกแบบการ์ด IEEE-488 สำหรับไมโครคอมพิวเตอร์ ลงตีพิมพ์วารสารวิจัยและพัฒนา สจธ. ประสพการณ์ในการทำงาน ปีพ.ศ. 2537 เป็นผู้ช่วยวิจัย โครงการวิจัยเรื่องเครื่องวัดสัญญาณคลื่นหัวใจแบบมีหน่วยความจำ ปีพ.ศ. 2538 - 2539 เป็นผู้ช่วยวิจัย โครงการวิจัยเรื่องการใช้คอมพิวเตอร์รับรู้และตรวจค้นหาเซลล์ทางนรีเวช และปีพ.ศ. 2536 - 2538 เป็นอาจารย์ผู้ช่วยสอนวิชาปฏิบัติการอิเลกทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีพ.ศ. 2535 - 2536 ได้รับทุนมูลนิธิเพื่อการศึกษาคอมพิวเตอร์และการสื่อสาร