การควบคุมโดยใช้มาตรฐาน IEEE-488 (GPIB)

THE CONTROLLING METHOD BY USING IEEE-488 (GPIB)

ผู้จัดทำ

นางสาวกฤษรา อำพลจันทร์ รหัส 34101008

นางสาวนุช จาดจวบสินธ์ รหัส 34104184

อาจารย์ที่ปรึกษา

รศ.ดร.มนัส สังวรศิลป์

ปริญญานิพนธ์นี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมอิเลคทรอนิคส์

สถาบัน เทคโนโลยีพระจอม เกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชาอิเลคทรอนิคส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมโดยใช้ IEEE-488

ผู้จัดทำ

นางสาวกฤษรา อำพลจันทร์ รหัส 34101008

นางสาวนุช จาดจวบสินธ์ รหัส 34104184

(รศ.ดร.มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การควบคุมโดยใช้ IEEE-488 (GPIB)

โดย นางสาวกฤษรา อำพลจันทร์
นางสาวนุช จาดจวบสินธ์

รศ.ดร.มนัส สังวรศิลป์ อ.ที่ปรึกษา

ปีการศึกษา 2537

บทคัดย่อ

ปริญญานิพนธ์นี้กล่าวถึง การควบคุมอุปกรณ์ต่างๆ โดยใช้มาตรฐาน IEEE-488 (GPIB) อันเป็นระบบมาตรฐานในการส่งรับข้อมูลของ IEEE มาตรฐานการส่งรับข้อมูลแบบนี้เป็นการรับหรือส่งข้อมูลระหว่างตัวควบคุม (Controller) กับอุปกรณ์ที่ทำหน้าที่เป็นตัวส่ง (Talker) หรือ เป็นตัวรับข้อมูลนั้น (Listener) โดยใช้การรับส่งข้อมูลแบบขนาน 8 bit โดยทั่วไปในระบบที่เป็นมาตรฐาน อุปกรณ์แต่ละชนิดจะมีคำสั่งเฉพาะ (programming protocol) ที่ใช้ติดต่อกับตัวควบคุมผ่าน IEEE 488 ในปริญญานิพนธ์นี้เป็นการเขียนโปรแกรมที่เรียกว่า Instrument driver (คุณสมบัติ คือ เป็นที่รวบรวมฟังก์ชันการทำงานต่างๆ ของอุปกรณ์ชนิดนั้นๆ) ผู้ใช้ไม่จำเป็นต้องเรียนรู้ programming protocal ของอุปกรณ์ชนิดนั้นเลย เพราะมันถูกบรรจุอยู่ภายใน function ของ Instrument driver ของอุปกรณ์นั้นแล้ว ทำให้สะดวกที่จะนำไปประยุกต์เขียนเป็นโปรแกรมใช้งานต่างๆ ต่อไป โดยได้เขียน Instrument driver ของอุปกรณ์ต่อไปนี้คือ Power supply HM8142, Function Generator HM8130, Oscilloscope HM1007 และ Multimeter HM8112-2 และยังได้ทดลองสร้างโปรแกรมประยุกต์ใช้งานผ่าน Instrument driver เหล่านี้ด้วย

The Controlling Method by Using IEEE-488 (GPIB)

By Miss Kritsara Umponjun
Miss Nuch Jadjuabsint

Adviser Assoc.Dr.Sungwarasin Manus

Education year 1994

ABSTRACT

This thesis concerns about the controlling method by using IEEE-488 (GPIB). It is a standard system for transfering data. By sending or receiving information between Controller(PC) and Talker/Listener in parallel 8 bit of databus.

In the standard system each instrument has it own programming protocal for communicating with the controller and other devices via IEEE 488 (GPIB) .By using the instrument driver, it frees the user from learning the programming protocal of an instrument.The software routines of an instrumint driver control an instrument. To the user,an instrument driver includes one or more function to perform high-level Instrument related tasks by including the function calls in an application program. We write the drivers for the Power Supply HM8142 ,Function Generation HM8130 ,Multimeter HM8112-2 and Oscilloscope HM1007. We also write the sample program by using instrument driver to communication with the instrument.

กิตติกรรมประกาศ

ทางผู้จัดทำปริญญานิพนธ์ขอกราบขอบพระคุณ รศ.ดร.มนัส สังวรศิลป์ และอาจารย์เทอดศักดิ์ ลิ่วหาทอง เป็นอย่างสูง ซึ่งได้ให้คำแนะนำในการจัดทำโครงงานและปริญญานิพนธ์ฉบับนี้จนเสร็จสมบูรณ์ และขอขอบคุณ พี่ๆ เพื่อนๆ ทุกคน ที่ได้กรุณาให้คำแนะนำและความช่วยเหลือในด้านต่างๆ ตลอดจนคอยเป็นกำลังใจ จนทำให้ปริญญานิพนธ์นี้สำเร็จลงได้ด้วยดี

สารบัญ

บทนำ
การควบคุมด้วย IEEE-488

ระบบอุตสาหกรรมในอดีตมักใช้อุปกรณ์ไม่มากชิ้น หากมีการเชื่อมต่อก็สามารถทำได้โดยง่าย ต่อมาการอุตสาหกรรมได้เจริญรุดหน้าไปอย่างรวดเร็วดังนั้นระบบอุตสาหกรรมจึงมีขนาดใหญ่และ ซับซ้อนขึ้นด้วยการเชื่อมต่อระหว่างอุปกรณ์หลายๆชิ้น จะต้องเสียค่าใช้จ่ายจำนวนมากไปด้วย หากจะมีการเชื่อมต่อระหว่างอุปกรณ์หลายๆชิ้นจะต้องเสียค่าใช้จ่ายจำนวนมาก ดังนั้นจึงมีการคิดค้นระบบเชื่อมต่อ ซึ่งเป็นมาตรฐานขึ้นมา นั่นคือ IEEE-488 นั่นเอง

ประวัติการพัฒนาIEEE-488

ดังที่ได้กล่าวมาแล้วข้างต้นว่า การเชื่อมต่อทางอุตสาหกรรมในอดีต เป็นไปด้วยความยากลำบากและเสียค่าใช้จ่ายมาก ดังนั้นบริษัทผู้ผลิตเครื่องมือวัดต่างๆในประเทศสหรัฐอเมริกาจึงร่วมกันจัดหาระบบเชื่อมต่อมาตรฐานขึ้นมา ซึ่งในประเทศเยอรมันก็มีการพัฒนาระบบเชื่อมต่อมาตรฐานเช่นกันโดยความร่วมมือของ IEC (International Eletrotechnical Commission) จนกระทั่งในปี 1972 สหรัฐอเมริกาโดยการนำของ IEEE (Institute of electrical and Electronics Enginners : IEEE) จึงได้มีการประชุมเพื่อวางแผนพิจารณาระบบเชื่อมต่อมาตรฐานร่วมกัน

บริษัทฮิวเล็ตแพกการ์ด ผู้ผลิตเครื่องมือวัดรายใหญ่ในอเมริกา ได้ทำการพัฒนาระบบเชื่อมต่อมาตรฐานอยู่ก่อนแล้วชื่อว่า HPIB (Hewwlett Packard Interface Bus)จึงได้เสนอโครงการให้ IEEE เพื่อพิจารณา และได้รับการยอมรับในปี 1975 โดย IEEE จัดให้เป็นมาตรฐานลำดับที่ 488 ดังนั้น จึงได้ชื่อว่า IEEE-Std 488-1975 ซึ่งต่อมาได้มีการปรับปรุงเป็น IEEE-Std 488-1978 หรือที่นิยมเรียกกันว่า GPIB (Generral Purpose Interface Bus)

บทที่ 1

IEEE-488 (GPIB)

โครงสร้างของ IEEE-488

ในระบบพื้นฐานของ GPIB จะประกอบด้วยอุปกรณ์ คือ ผู้ส่ง(Talker) , ผู้รับ (Listener) และผู้ควบคุม (Controller)

- Talker ทำหน้าที่ส่งข้อมูล โดยในระบบสามารถมี Talker ได้หลายตัวแต่จะมีเพียงตัวเดียวเท่านั้นที่กำลังทำงานอยู่

- Listener ทำหน้าที่เป็นตัวรับข้อมูล โดยในระบบเดียวกันสามารถมี Listener ได้หลายตัวเช่นเดียวกัน แต่ Listener สามารถทำงานได้ครั้งละหลายๆตัวได้

- Controller ทำหน้าที่ควบคุมอุปกรณ์ต่างๆในระบบ โดยจะกำหนดให้ Talker ทำการส่งข้อมูล หรือ กำหนดให้ Listener ทำการรับข้อมูล

อุปกรณ์ที่มี GPIB นั้นสามารถแบ่งตามหน้าที่ได้ดังนี้

1. ทำหน้าที่เป็น Talker เท่านั้น เช่น เครื่องมือวัด เป็นต้น

2. ทำหน้าที่เป็น Listener เท่านั้น เช่น เครื่องพิมพ์(Printer), เครื่องบันทึก (Recorder) เป็นต้น

3. ทำหน้าที่เป็นทั้ง Talker และ Listener เช่น คอมพิวเตอร์, เครื่องมือวัดที่สามารถควบคุมได้จากภายนอก เป็นต้น

4. ทำหน้าที่เป็น Talker Listener และ Controller ในตัวเดียวกัน เช่น คอมพิวเตอร์ที่ทำหน้าที่ควบคุมระบบ

ขีดจำกัดของ IEEE-488

1. จำนวนอุปกรณ์ในระบบ(Talker,Listener,Controller) ที่ต่อกับสายสัญญาณ 1 เส้น จะต้องไม่เกิน 15 เครื่อง

2. สายเคเบิ้ลที่ต่อระหว่างอุปกรณ์ จะต้องยาวไม่เกิน 4 เมตร และ ความยาวรวม

ของสายเคเบิลในระบบจะต้องไม่เกิน 20 เมตร

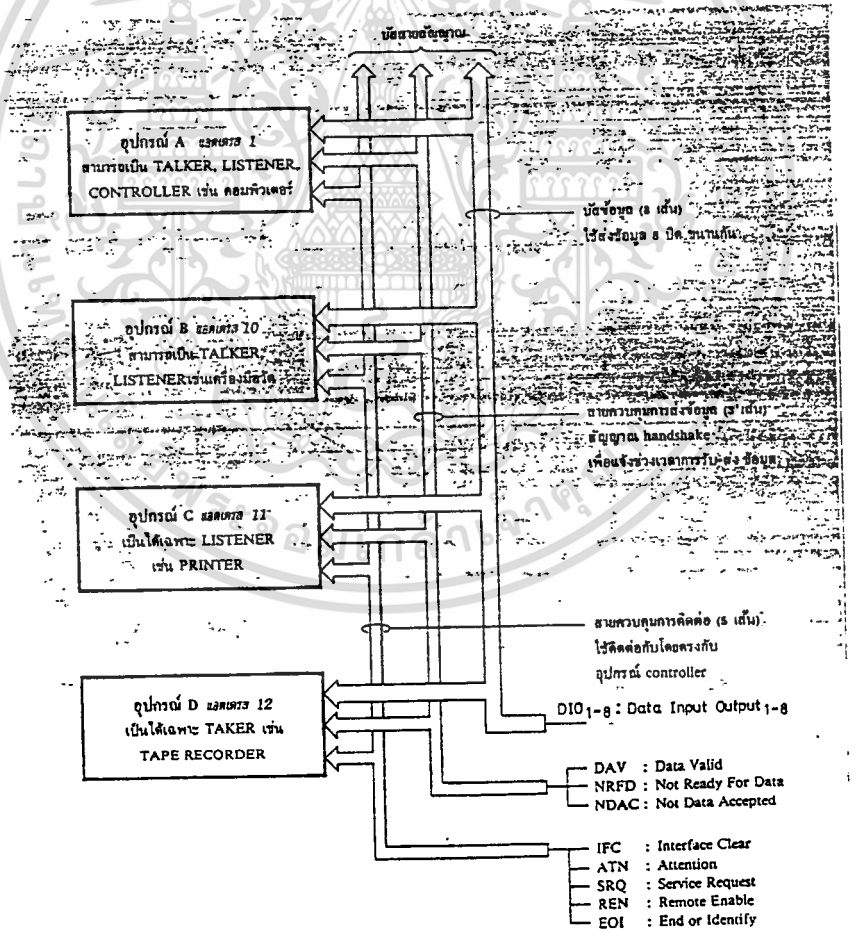
3. ความเร็วในการส่งข้อมูลจะต้องไม่เกิน 1Mb/Sec (1 ล้านไบต์ต่อวินาที)

4. ต้องมีการจ่ายไฟให้กับอุปกรณ์มากกว่าครึ่งหนึ่งของระบบ

รายละเอียดเกี่ยวกับ IEEE-488

ลักษณะทางกายภาพของ IEEE-488 นั้นคือ เป็นสายสัญญาณแบบ 24 เส้นขนานกัน และมีขั้วต่ออยู่ทางปลายทั้งสองของสาย เพื่อต่อกับอุปกรณ์ หรือ ต่อกันเพื่อให้สายสัญญาณมีความยาวเพิ่มขึ้น ในจำนวนสายสัญญาณ 24 เส้นนี้ มีเพียง 16 เส้นเท่านั้น ที่ทำหน้าที่นำสัญญาณ ส่วนที่เหลืออีก 8 เส้น ทำหน้าที่กราวนด์ (ground) และ ชีลด์ (shield)

โดยจำนวนสายที่ใช้นำสัญญาณ 16 เส้นนั้นยังแบ่งได้เป็น 3 ประเภทตามรูปที่ 1 คือ



รูปที่ 1.1

1. บัสข้อมูล (Data Bus) จำนวน 8 สาย คือ

DIO1- DIO8

2. สายสัญญาณควบคุม (Control Line) จำนวน 5 สาย คือ

IFC (Interface Clear)

ATN (Attetention)

SRQ (Service Request)

REN (Remote Enable)

EOI (End Or Identify)

3. สายแฮนด์เชค (HAND SHAKE) 3 สาย คือ

DAV (Data Valid)

NRFD (Not Ready For Data)

NDAC (Not Data Accepted)

<u>ความหมายของสัญญาณต่างๆภายใน IEEE-488</u>

ดังที่ได้กล่าวมาแล้วว่าสายสัญญาณต่างๆใน GPIB ได้แบ่งออกเป็น 3 กลุ่ม ในหัวข้อนี้จะอธิบายความหมายของสัญญาณต่าง ๆ ดังนี้

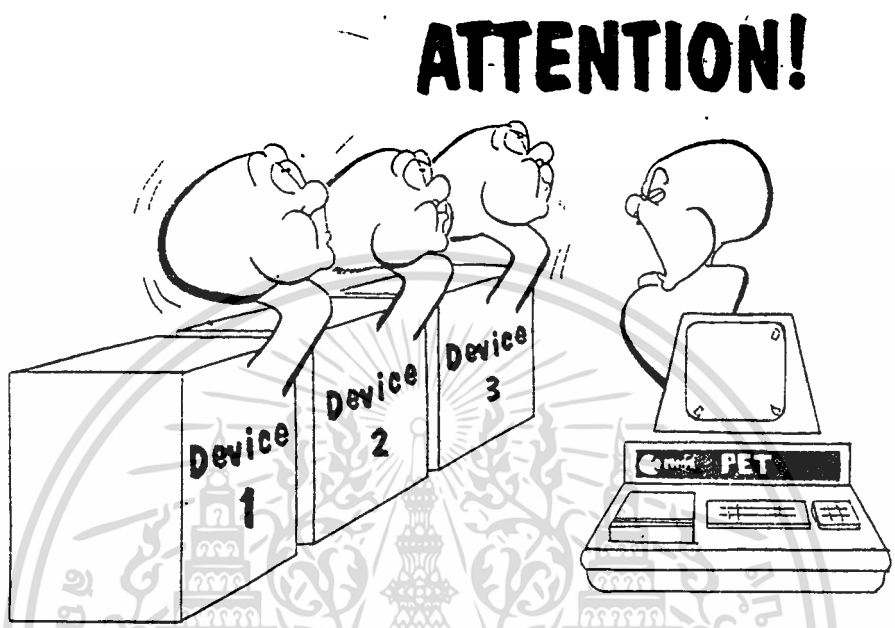
กลุ่มสัญญาณข้อมูล

1. DIO1-DIO8 (Data Input/Output) สายสัญญาณทั้ง 8 เส้นนี้ทำหน้าที่เป็นทางผ่านของข้อมูลในระบบ

กลุ่มสัญญาณควบคุมการเชื่อมต่อ (Interface)

1. IFC (Interface Clear) เป็นสัญญาณรีเซต หรือ เคลียร์ระบบ กำเนิดได้โดยตัวควบคุม (Controller)เท่านั้น เมื่ออุปกรณ์ในบัสได้รับสัญญาณเคลียนี้ จะกลับคืนสู่สภาวะเริ่มต้นใหม่ ซึ่งเป็นสภาวะแรกเริ่มก่อนการกำหนดฟังก์ชันเหมือนแรกเปิดสวิตช์

2. ATN (Attention) เป็นสัญญาณที่ถูกส่งโดยอุปกาณ์ที่เป็นตัวควบคุมเช่นเดียวกัน ใช้ในการสั่งให้อุปกาณ์ทุกตัวในระบบเตรียมพร้อมเพื่อรอรับคำสั่งต่อไป



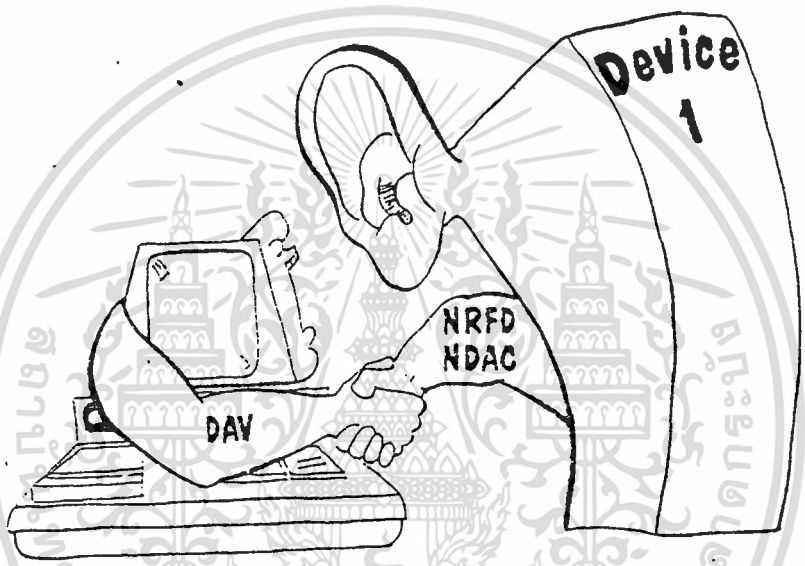
3. SRQ (Service Request) เป็นสัญญาณที่ถูกส่งจากอุปกรณ์ต่าง ๆ เพื่อเป็นการบอกแก่ระบบว่าขณะนี้อุปกรณ์ดังกล่าวต้องการการติดต่อจากตัวควบคุม

4. REN (Renote Enable) สัญญาณนี้ เป็นสัญญาณที่ถูกส่งมาจากตัวควบคุมเพียงตัวเดียวเท่านั้น เพื่อใช้สั่งให้อุปกรณ์ต่างๆเปลี่ยนจากโหมดที่ใช้งานปกติมาเป็นการควบคุมโดยตัวควบคุมแทน

5. EOI (End or Identify) เป็นสัญญาณที่ถูกส่งได้ โดยอุปกรณ์ที่เป็นตัวควบคุม (Controller) หรืออุปกรณ์ที่เป็นตัวส่ง (Talker)ก็ได้ ใช้สำหรับแสดงว่าข่าวสารที่ส่งเป็นชุดนั้นได้เสร็จสิ้นลงแล้ว

กลุ่มสัญญาณควบคุมการรับ-ส่งข้อมูล

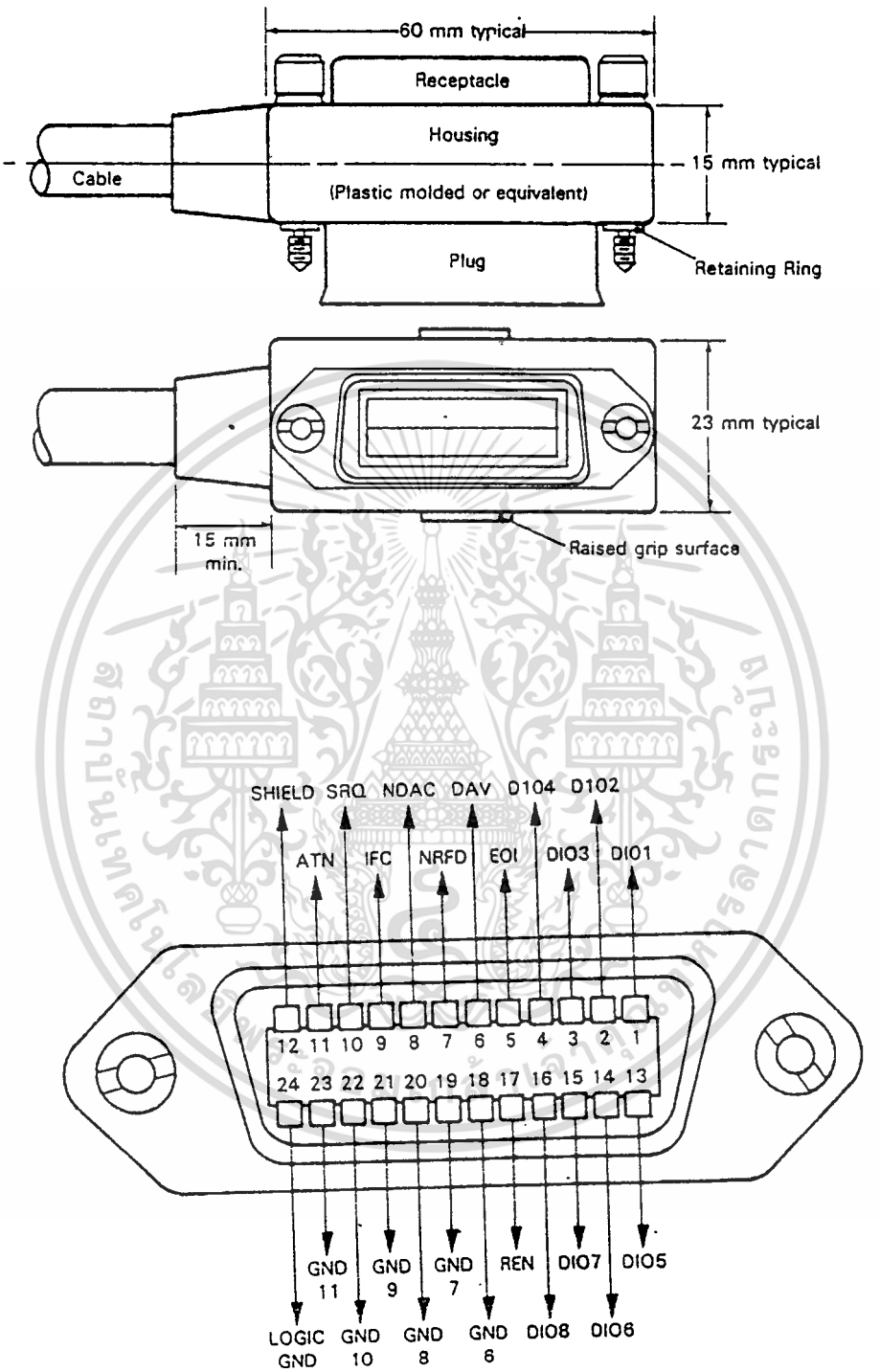
1. DAV (Data Valid) เมื่อสัญญาณนี้ถูกดึงเป็นลอจิก "Low" โดยอุปกรณ์ที่เป็นตัวส่ง (Talker) เป็นการแจ้งแก่ระบบบัสว่า ขณะนี้ตัวส่ง ได้ทำการส่งข้อมูล ลงไปที่สายสัญญาณข้อมูลเรียบร้อยแล้ว



2. NRFD (Not Ready ForData) เมื่อสัญญาณนี้มีลอจิกเป็น "Low" จะเป็นการแสดงว่าในขณะนี้ระบบบัสยังไม่พร้อมที่จะรับข้อมูล เนื่องจากอุปกรณ์ในระบบยังพร้อมไม่หมดทุกตัว ซึ่งสัญญาณเส้นนี้จะไม่เป็น "Hi" จนกว่าอุปกรณ์ทุกตัวให้ลอจิกที่เป็น "Hi" ครบถ้วนแล้ว สัญญาณนี้มีประโยชน์ในกรณีที่อุปกรณ์ในระบบมีความเร็วแตกต่างกัน

3. NDAC (Not Data Accepted) สัญญาณเส้นนี้เป็นสัญญาณที่ถูกควบคุมโดยอุปกรณ์ที่เป็นตัวรับ (Listener) โดยสัญญาณนี้จะมีลอจิกเป็น "Low" ในขณะที่อุปกรณ์ที่เป็นตัวรับกำลังเก็บข้อมูลจากสายข้อมูล(Data Bus)และจะเป็น "Hi" เมื่ออุปกรณ์นั้นได้ทำการอ่านข้อมูลเสร็จเรียบร้อยแล้ว

โดยสัญญาณลอจิกที่ใช้ใน DATA BUS (D1-D8) ของ IEEE-488 นี้มีลักษณะเป็นคอมพลีเมนต์ทั้งหมด คือ"1"เท่ากับ "Low" และ "0" เท่ากับ "Hi" ซึ่งตรงข้ามกับในวงจรที่เราคุ้นเคย

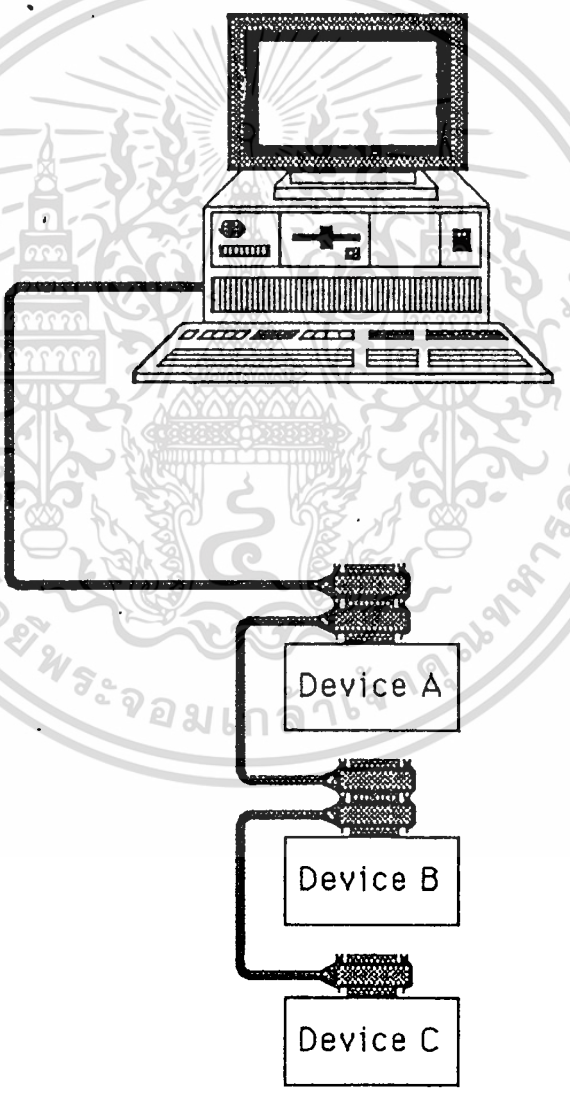


รูปที่ 1.2 ขั้วต่อของ GPIB และการจัดขาของสัญญาณต่าง ๆ

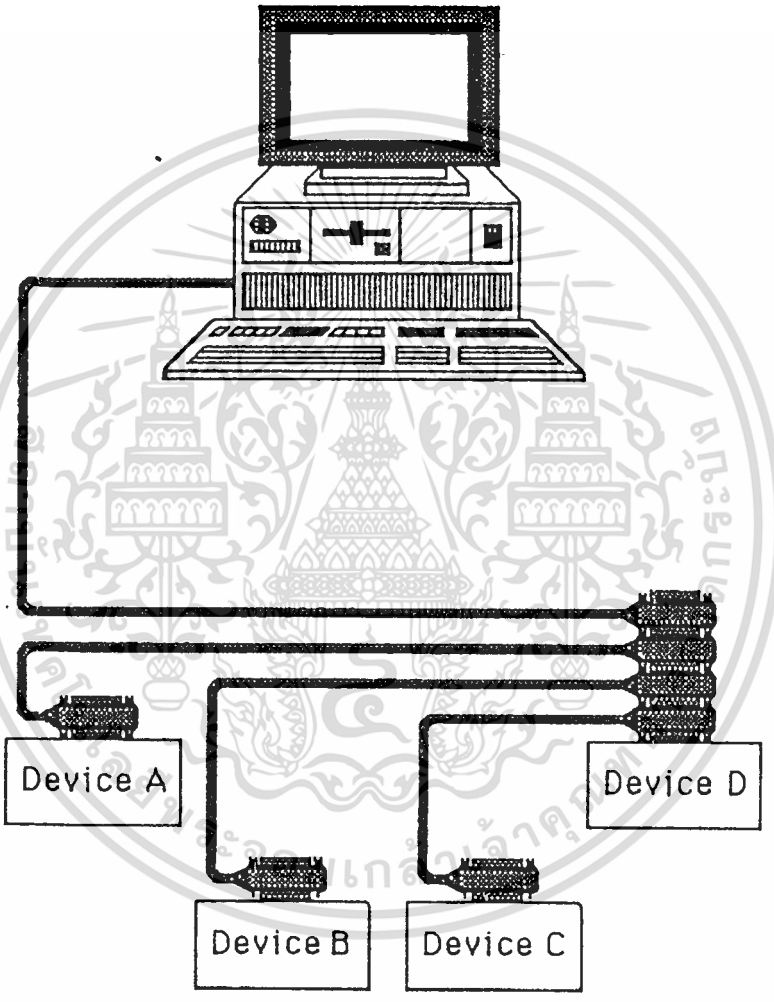
การเชื่อมต่ออุปกรณ์ต่างๆในระบบ IEEE-488 BUS

สำหรับการเชื่อมต่อระหว่างอุปกรณ์ต่างๆในระบบ IEEE-488 Bus นั้น มีอยู่ 2 วิธีคือ

1. การเชื่อมต่อแบบเรียงต่อเนื่องกัน (Daisy Chain Configuration)
2. การเชื่อมต่อแบบกระจาย (Star Configuration)



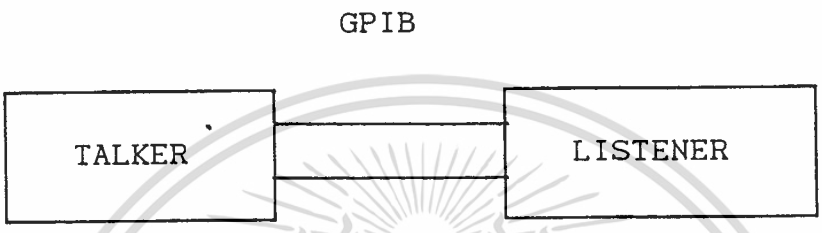
รูปที่ 1.3 การเชื่อมต่อแบบเรียงต่อเนื่องกัน (Daisy Chain Configuration)



รูปที่ 1.4 การเชื่อมต่อแบบกระจาย (Star Configuration)

ขบวนการแฮนด์เชค (Handshake Prodedure)

เมื่อมีการเชื่อมต่อระหว่างอุปกรณ์ต่าง ๆ ในระบบ ดังนั้นเมื่อระบบจะทำงานจึงต้องมีการตรวจสอบเสียก่อน ซึ่งการตรวจสอบนี้เราเรียกว่าขบวนการแฮนด์เชค (Handshake Procedure)



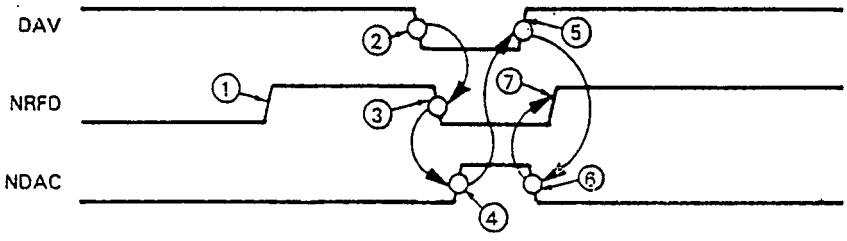
รูปที่ 1.5

พิจารณาระบบที่ไม่ซับซ้อนนักโดยกำหนดให้มีตัวส่ง(Talker)และตัวรับ (Listener) อย่างละ 1 ตัว

การที่จะมีการสื่อสารระหว่างตัวส่งและตัวรับจะกระทำได้โดยการที่ตัวส่งจะทำหน้าที่ส่งสัญญาณออกมาเพื่อให้ตัวรับทราบว่ามีข้อมูลเสร็จในสายสัญญาณ และตัวรับก็จะส่งสัญญาณเพื่อให้ตัวส่งทราบว่าได้ทำการรับข้อมูลเป็นที่เรียบร้อยแล้ว

การส่งสัญญาณในการสื่อสารระหว่างตัวส่งและตัวรับนั้น จะส่งมาทางสายสัญญาณ 3 สาย ซึ่งสายสัญญาณดังกล่าวนั้นเป็นสายสัญญาณในกลุ่มควบคุมการรับ-ส่งข้อมูล คือ NRFD (Not Ready For Data),DAV (Data Valid)และ NDAC (Not Data Accepted)

DAV เป็นสัญญาณที่ถูกส่งโดยตัวส่ง NRFD และ NDAC เป็นสัญญาณที่ถูกส่งโดยตัวรับ โดยตัวรับจะใช้สัญญาณ NDAC เพื่อชี้ให้เห็นว่าพร้อม หรือไม่พร้อมที่จะรับข้อมูล

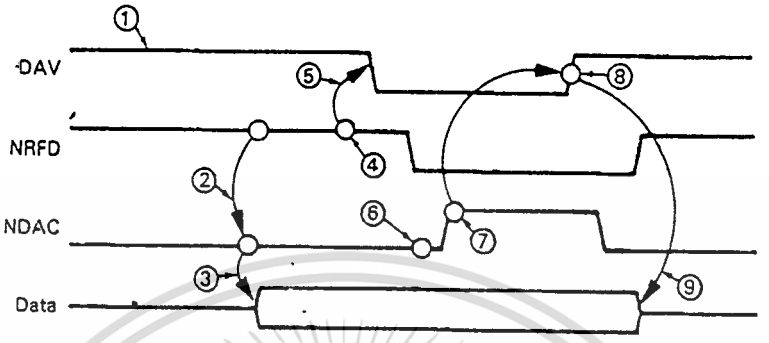


Note: The timing shown is relative.

รูปที่ 1.6

ขบวนการแฮนด์เชคเริ่มขึ้นเมื่อ ตัวส่งมีข้อมูลที่จะถ่ายทอดลงสายสัญญาณไปยังตัวรับ เริ่มด้วยการที่ตัวรับจะทำให้สัญญาณ NRFD เป็น "Hi" (มีค่าเป็น 0) นั่นคือเป็นการบอกให้ทราบว่าตัวรับพร้อมที่จะให้ตัวส่งทำการถ่ายข้อมูลลงในสายสัญญาณแล้ว (ในจุดที่ 1) ตัวส่งจะทำการถ่ายข้อมูลลงในสายสัญญาณ DIO1-DIO8 หลังจากทำการรอเวลา เพื่อให้ตัวส่งถ่ายข้อมูลแล้ว ตัวส่งจะทำให้สัญญาณ DAV มีลอจิกเป็น "Low" (มีค่าเป็น 1) เพื่อที่จะบอกให้ทราบว่าขณะนี้ได้มีข้อมูลในสายสัญญาณทั้ง 8เส้นแล้ว (ในจุดที่ 2) ต่อมาเมื่อผู้รับทราบว่ามีข้อมูลอยู่ในบัสก็จะทำให้สัญญาณ NRFD ให้มีค่าเป็น "Low" เพื่อบอกให้ตัวส่งทราบว่ายังไม่พร้อมที่จะให้ตัวส่งทำการส่งข้อมูลชุดต่อไป และตัวรับพร้อมที่จะเก็บข้อมูลจากสายสัญญาณ (ในจุดที่ 3) หลังจากตัวรับทำการเก็บข้อมูลจากสายสัญญาณเสร็จแล้ว ตัวรับจะทำให้สัญญาณ NDAC ให้มีสถานะเป็น "Hi"เพื่อบอกให้ทราบว่าได้ทำการเก็บข้อมูลเสร็จแล้ว (ในจุดที่ 4) เมื่อตัวส่งตรวจพบว่าสัญญาณ NDAC มีลอจิกเป็น "Hi"ก็จะมาทำให้สัญญาณ DAV ให้มีลอจิกเป็น "Hi" ด้วยเพื่อไม่ให้ตัวรับทำการรับข้อมูลในบัสอีก (ในจุดที่ 5) เมื่อตัวรับทราบว่า สัญญาณ DAV มีลอจิกเป็น"Hi" ก็จะทำให้สัญญาณ NDAC ให้เป็น "Low" ทำให้ข้อมูลในบัสถูกกำจัดออกไป หลังจากนั้นก็จะทำให้สัญญาณ NRFD ให้เป็น "Hi" เพื่อบอกให้ทราบว่าพร้อมที่จะรับข้อมูลชุดต่อไปแล้วขบวนการแฮนด์เชคจึงเสร็จสิ้นลง และจะเริ่มขึ้นใหม่เมื่อมีสัญญาณชุดใหม่เข้ามา

ขบวนการแฮนด์เชคของตัวควบคุมซึ่งทำหน้าที่ เป็นตัวส่ง

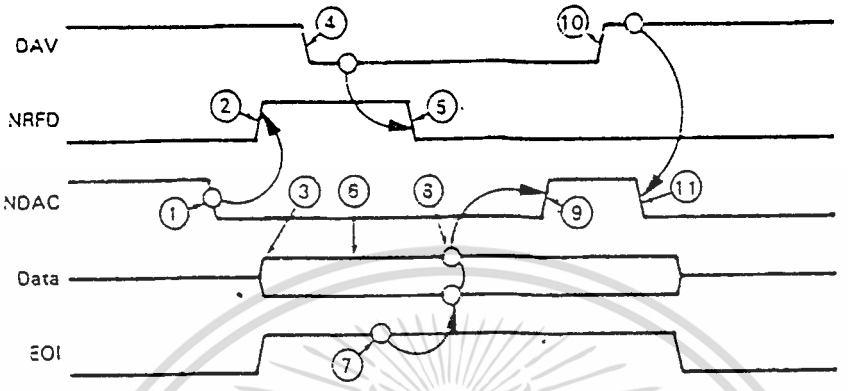


รูปที่ 1.7

ขบวนการแฮนด์เชคเริ่มขึ้นโดยการที่ควบคุม ทำการเซตค่าให้สัญญาณ DAV มีค่าเป็น "Hi"(ในจุดที่ 1) ต่อมาเป็นการตรวจสอบว่าสัญญาณ NDAC และ NRFD มีลอจิกเป็น"Hi" ทั้งคู่หรือไม่ (ในจุดที่ 2) ถ้าทั้งคู่เป็น"Hi"นั่นคือเกิดการผิดปกติ คืออุปกรณ์นั้นไม่พร้อมที่จะทำงาน จึงทำการออกจากขบวนการแฮนด์เชค

กลับมาดูในจุดที่ 2 หากสัญญาณใดสัญญาณหนึ่งมีลอจิกเป็น "Low" ตัวควบคุมจะทำการส่งข้อมูลลงใน GPIB Bus(ในจุดที่ 3) หลังจากนั้นตัวควบคุมจะทำการตรวจสอบว่าสัญญาณ NRFD มีลอจิกเป็น "Hi" (ในจุดที่ 4) หลังจากนั้นตัวควบคุมจะทำให้สัญญาณ DAV ให้มีลอจิกเป็น "Low" เพื่อให้ตัวรับทราบว่ามีข้อมูลอยู่ในบัสข้อมูล (ในจุดที่ 5) ตัวควบคุมจะรอเวลาเพื่อให้ตัวรับทำการเก็บข้อมูลเมื่อครบกำหนดเวลาตัวควบคุมจะทำการเชคว่าตัวรับทำงานเกินเวลาที่กำหนดหรือไม่หากเกินเวลาก็จะทำการตรวจสอบต่อไปว่าสัญญาณ NDAC ทำลอจิกเป็น "Hi" ใช่หรือไม่ก็จะทำการตรวจสอบอีกครั้งว่าเกินเวลาหรือไม่ กลับไปดูที่จุดที่ 6 หากสัญญาณ NDAC เป็น "Hi" (ในจุดที่ 7) ตัวควบคุมจะทำการตอบสนองโดยการทำสัญญาณ DAV ให้มีลอจิกเป็น "Hi" เพื่อบอกให้ตัวรับทราบว่าให้หยุดการเก็บข้อมูลจากบัส (ในจุดที่ 8) หลังจากนั้นข้อมูลในบัสข้อมูลจะถูกนำออกไป (จุดที่ 9) จึงเป็นการเสร็จสิ้นขบวนการแฮนด์เชค

ขบวนการแฮนด์เชคของตัวควบคุมซึ่งทำหน้าที่เป็นตัวรับ



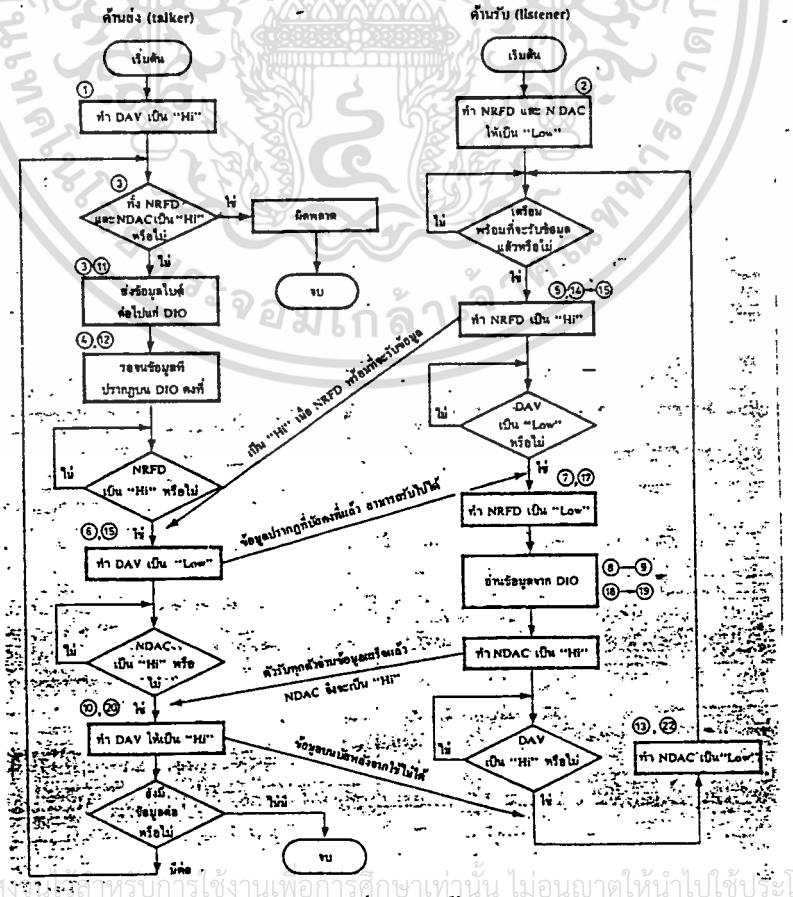
รูปที่ 1.8

ขบวนการแฮนด์เชคเริ่มขึ้น หลังจากการที่ตัวควบคุมรับรู้ว่าตัวส่งจะทำการส่งข้อมูลเมื่อตัวควบคุมทราบว่าตัวส่งจะทำการส่งจะทำการส่งข้อมูล ตัวควบคุมรับรู้ว่าตัวส่งจะทำให้สัญญาณ NDAC มีลอจิกเป็น"Low"เพื่อบอกให้ทราบว่าตัวควบคุมยังไม่ได้ทำการเก็บข้อมูล (ข้อมูลในที่นี้คือ แอดเดรสหรือ ข้อมูลทั่วไปซึ่งตัวส่งทำการส่งมาในบัสข้อมูล) นั่นคือในจุดที่ 1

ต่อมาตัวควบคุมจะทำการเซตค่าสัญญาณ NRFD ให้เป็น "Hi" เพื่อที่จะให้ตัวส่งทราบว่าขณะนี้ตัวควบคุมพร้อมที่จะรับข้อมูลแล้ว (ในจุดที่ 2) เมื่อสัญญาณ NRFD มีค่าเป็น "Hi" และะสัญญาณ NDAC มีค่าเป็น "Low"แล้ว ตัวส่งจะทราบทันทีว่าทำการส่งข้อมูลลงมาในบัสข้อมูลได้แล้ว (ในจุดที่ 3) หลังจากนั้นตัวควบคุมจะทำการรอเวลาให้ตัวส่งทำการส่งข้อมูลลงในบัสข้อมูลให้เสร็จ โดยจะทำการตรวจสอบไปด้วยว่าเกินเวลาที่กำหนดหรือยังหากเกินเวลาที่กำหนดก็จะออกจากขบวนการแฮนด์เชค หากไม่เกินกำหนดเวลาจะทำการตรวจสอบเวลาว่าเกินเวลาที่ต้องคอยแล้วหรือยัง หากยังไม่เกินเวลาก็จะทำการเชคต่อไปจนกระทั่งหมดเวลาหรือจนกว่าสัญญาณ DAV เป็น "Low"

เมื่อสัญญาณ DAV เป็น "Low" แล้วตัวควบคุมจะตอบรับ โดยการทำให้สัญญาณ NRFD มีค่าเป็น "Low " (ในจุดที่ 5) นั่นเป็นการบอกว่าตัวควบคุมพร้อมที่จะเริ่มเก็บข้อมูล (ในจุดที่ 6) หลังจากนั้นตัวควบคุมจะทำการตรวจสอบสัญญาณ EOI เพื่อตรวจสอบว่าข้อมูลที่ตัวส่งทำการส่งลงมานั้นหมดแล้วหรือยัง(ถ้าตัวส่งทำการส่งข้อมูลลงในบัสหมดแล้วจะมำการตั้งค่าสัญญาณ EOI ให้เป็น "Low") (ในจุดที่ 7) หากสัญญาณ EOI เป็น "Low" ตัวควบคุมจะเซตสถานะว่า ขณะนี้ตัวส่ง ได้ทำการส่งข้อมูลลงมาในบัสหมดแล้ว หากสัญญาณ EOI เป็น "Hi" ก็ไม่ทำการเซตสถานะ หลังจากนั้นตัวควบคุมจะทำการเก็บข้อมูลในบัส (ในจุดที่ 8) แล้วทำให้สัญญาณ NDAC มีค่าเป็น "Hi" เพื่อบอกให้ทราบว่าตัวควบคุมได้ทำการอ่านข้อมูลเสร็จเรียบร้อยแล้ว (ในจุดที่ 9) เมื่อตัวควบคุมทำให้สัญญาณ NDAC เป็น "Hi" แล้วตัวส่งจะลบข้อมูลในบัสข้อมูล โดยการทำให้สัญญาณ DAV เป็น "Hi" (ในจุดที่ 10) ซึ่งก่อนหน้านี้ตัวควบคุมจะทำการตรวจสอบอยู่แล้วว่าสัญญาณ DAV เป็น "Hi" หรือยัง

เมื่อสัญญาณ DAV เป็น "Hi" แล้วตัวควบคุมจะทำให้สัญญาณ NDAC มีค่าเป็น "Low" (ในจุดที่ 11) แล้วจึงออกจากขบวนการแฮนด์เชค



รูปที่ 1.8 - 1

คำสั่งใช้งานของ GPIB

การสั่งการต่างๆ เพื่อกำหนดหน้าที่การทำงาน และกำหนดฟังก์ชัน เช่น กำหนดช่วงการวัด โหมดการวัด หรืออื่นๆ แก่เครื่องวัดที่ต่ออยู่ เหล่านี้นั้น ต้วควบคุมจะเป็นต้วกำหนดโดยการส่งรหัสคำสั่งไปที่อุปกรณ์โดยผ่าน DI1-DI6 รหัสคำสั่งนี้จะถูกส่งไปในช่วงที่สายสัญญาณ ATN เป็น LOW

คำสั่งสำหรับกำหนดหน้าที่การทำงานต่าง ๆ ตามมาตราฐานของ GPIB มีอยู่ด้วยกัน 128 คำสั่งดังแสดงในตารางที่ 1 ต่อไปนี้ โดยแบ่งเป็น 5 กลุ่มคำสั่ง

รหัสที่ใช้ในระบบ GPIB บัสนั้นใช้ร่วมกัน ทั้งรหัสข้อมูล และรหัสคำสั่ง นั่นคือรหัสเดียวกันมีความหมายได้ 2 อย่าง คือ เมื่อ ATN เป็น LOW จะหมายถึงรหัสคำสั่ง แต่ถ้า ATN เป็น HIGH รหัสนี้จะแทนข้อมูลที่เป็น ASCII แทน ซึ่งในตาราง ก็ได้แบ่งความหมายออกเป็น 2 คอลัมน์

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	MSG[1]	ASCII	MSG[1]	ASCII	MSG[1]	ASCII	MSG[1]	ASCII	MSG	ASCII	MSG
0	NUL		DLE		SP	00	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	13	]	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)

ตารางที่ 1.1

1. กลุ่มคำสั่งเจาะจงจุดหมาย (addressed command group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ที่เป็นตัวส่งหรือตัวรับที่กำหนดไว้ล่วงหน้าแล้ว คำสั่งนี้ประกอบด้วย

GTL (got to local) สั่งให้อุปกรณ์กลับสู่สภาพการควบคุมปกติด้วยมือ

SDC (selected device clear) สั่งให้อุปกรณ์เคลียร์ตัวเองสู่สภาพเริ่มต้นใหม่

PPC (paralled poll configure) เป็นคำสั่งสำหรับทำการ จัดสายสัญญาณของการทำการจัดสรรสายสัญญาณ ของการทำกระบวนการตรวจสอบสภาพอุปกรณ์ โดยวิธีขนานโดยใช้กับกลุ่มคำสั่งรอง

GET (group excute trigger) ใช้สั่งเริ่มต้นการทำงานของอุปกรณ์ทีละหลายตัว

TCT (take control) เป็นการกำหนดให้อุปกรณ์ตัวส่งทำหน้าที่เป็นตัวควบคุม

2. กลุ่มคำสั่งครอบคลุม (universal command group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ทุกตัวที่ต่ออยู่ในบัส ประกอบด้วย

LLO (local lockout) เป็นการสั่งให้อุปกรณ์ล็อกอยู่ที่สภาวะควบคุมโดยปุ่มปรับที่หน้าปัดตามปกติ

DCL (devide clear) สั่งให้อุปกรณ์ทุกตัวกลับไปสู่สภาวะเริ่มต้น

PPU (parallel poll unconfigure) ใช้ยกเลิกกระบวนการตรวจสอบสภาพแบบขนานทั้งหมด

SPE (serial poll enable) เปลี่ยนโหมดของการตรวจสอบสภาพเป็นแบบอนุกรม ในโหมดนี้จะเป็นการส่งสถานะของเครื่องแทนการส่งข้อมูล

SPD (serial poll disable) ยกเลิกโหมดการตรวจสอบแบบอนุกรม

3. กลุ่มคำสั่กำหนดอุปกรณ์ตัวรับ (listener address group) เป็นคำสั่งสำหรับกำหนดให้อุปกรณ์เป็นตัวรับ ตามรหัสหมายเลขจาก 0-30 และมีคำสั่ง UNT(untalker) สำหรับยกเลิก

4. กลุ่มคำสั่งกำหนดอุปกรณ์ตัวส่ง (talker address group) สำหรับกำหนดให้อุปกรณ์เป็นตัวส่ง ตามรหัสหมายเลขจาก 0-30 และมีคำสั่ง UNL (unlisten) สำหรับ

ยกเลิกเช่นกัน

คำสั่งกลุ่มที่ 1 ถึง 4 นั้น จัดเป็นกลุ่มคำสั่งหลัก ที่มีความหมายตายตัวยังมีคำสั่งอีกกลุ่มที่ขึ้นอยู่กับการกำหนดภายหลังนั่นคือกลุ่มคำสั่งรอง

5. กลุ่มคำสั่งรอง (secondary command group) เป็นคำสั่งที่กำหนดรายละเอียดย่อยของอุปกรณ์แต่ละตัวที่ต่ออยู่ในระบบ ให้มีการทำงานอย่างไร ตามจุดประสงค์ใช้งานของเครื่องมือนั้น เช่นเดียวกับการปรับปุ่มต่าง ๆ ด้วยมือนั่นเองคำสั่งรองนี้จะตามหลังคำสั่งหลักคือจะใช้หลังจากอุปกรณ์ต่างๆ ถูกกำหนดวางตัวในระบบเรียบร้อยแล้ว

คำสั่งต่างๆ ที่กล่าวไป ซึ่งใช้ในการกำหนดสภาวะการทำงานของอุปกรณ์ แต่ละสภาวะที่กำหนดไปนั้นเป็นอย่งไร และมีจุดประสงค์เพื่ออะไร ดังต่อไปนี้

Device clear/ Interface Clear

Device clear ใช้ในการทำให้อุปกรณ์ที่ต่ออยู่ในบัสกลับไปสู่สภาวะเริ่มต้น ยังไม่มีการกำหนดฟังก์ชันใดๆ สภาวะเริ่มต้นนี้จะแตกต่างกันไปแล้วแต่ว่าอุปกรณ์นั้น ออกแบบไว้อย่างไร Device clear มีอยู่ 2 ลักษณะคือ เคลียร์หมดทุกตัวที่ต่ออยู่ (DCL) กับเคลียร์เจาะจงอุปกรณ์ตัวใดตัวหนึ่ง (SDC)

แต่ว่าในการเคลียร์อุปกรณ์ให้อยู่ในสภาวะเริ่มต้นนั้นไม่ได้หมายความว่า Interface function ของ GPIB จะถูกเคลียร์อุปกรณ์ให้ไปอยู่ในสภาวะเริ่มต้นด้วยแต่อย่างใด interface function คือสภาพการ interface ที่ได้กำหนดไว้ในระบบประกอบด้วยฟังก์ชันต่างๆ ดังแสดงในตารางที่ 2

ฟังก์ชัน	สัญลักษณ์	การกลับสู่จุดเริ่มต้นโดยIFC
source handshake	SH	ได้
acceptor handshake	AH	ได้
talker or enlarge talker	T or TE	ได้
listener or enlarge listener	L or LT	ได้
service request	SR	ไม่ได้
reomte /local	RL	ไม่ได้
parallel poll	pp	ไม่ได้
device clear	DC	ได้
device trigger	DT	ได้
controller	C	ได้

ตารางที่ 1.2

Remote / Local

remote เป็นการกำหนดให้อุปกรณ์ที่อยู่ในระบบเช่น เครื่องมือวัดให้อยู่ในการควบคุมของอุปกรณ์ตัวอื่นแทน ซึ่งปุ่มปรับต่างๆ บนหน้าปัดเครื่องจะไม่มีผลต่อการทำงาน ส่วน local เป็นการควบคุมการทำงานของเครื่องมือวัดด้วยปุ่มปรับบนหน้าปัดตามปกติ

การใช้ remote มีประโยชน์ในแง่ที่ขณะที่ตัวการควบคุมเช่น คอมพิวเตอร์กำลังติดต่ออุปกรณ์ตัวนั้นอยู่ หากไม่มีการตัดการควบคุมโดยปุ่มปรับบนหน้าปัดออก ถ้ามีใครมาปรับแต่งก็จะทำให้การทำงานผิดพลาดไปได้ การทำงานของ GPIB ใน remote and local มี 4. สักษณะดังนี้

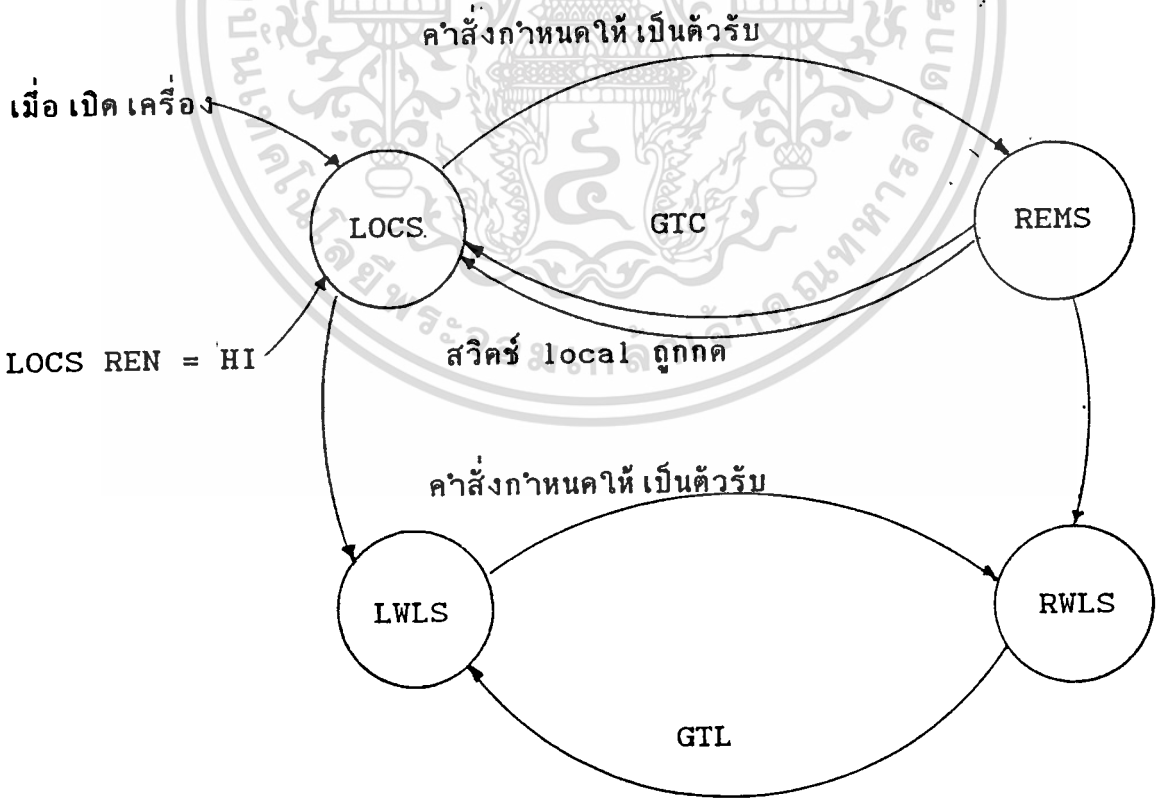
1. LOCS ก็คือ local นั่นเอง เป็นสภาพการควบคุมที่ปุ่มตามปกติ จะอยู่ในสภาพนี้

ตอนเปิดเครื่อง หรือ REN เป็น HIGH หรือเมื่อได้รับคำสั่ง GTL

2. REMS คือ remote หมายถึงการตัดการควบคุมโดยปุ่มหน้าปัดออก จะเกิดขึ้นเมื่อ REN เป็น LOW และจะถูกล็อกไว้ เว้นแต่ว่าสวิตช์ local ที่ตัวอุปกรณ์จะถูกเปลี่ยนไปตำแหน่ง local

3. RWLS เป็นสภาพ remote ที่ถูกล็อกเอาไว้เช่นกัน แต่ว่าจะตัดการควบคุมตรงสวิตช์ local ที่ตัวอุปกรณ์ออกไป สภาพ remote โดย RWLS จึงมีความสำคัญสูงกว่า REMS อย่างไรก็ตามยังถูกยกเลิกได้ด้วยคำสั่ง LLO

4. LWLS มีสภาพเช่นเดียวกับ local แต่จะแตกต่างกันตรงที่สภาพlocal โดย LWLS นี้เมื่อได้รับคำสั่งกำหนดอุปกรณ์ตัวรับจะเปลี่ยนไปอยู่ในสภาพแบบล็อกหรือ RWLS ทันทีในการที่จะมาที่สภาพ LWLS นี้ได้ก็มี 2 กรณีคือ เมื่ออยู่ในสภาพlocalธรรมดา (LOCS) แล้วได้รับคำสั่ง LLO หรืออยู่ใน RWLS แล้วได้รับคำสั่ง GTL



รูปที่ 1.8.2

การขอบริการและการตรวจสอบ (Service Request and Polling)

เมื่อตัวควบคุมได้รับ SRQ เป็น LOW จะให้อุปกรณ์ส่งข้อมูลแสดงสถานะการทำงานซึ่งมีอยู่ 2 วิธีคือ

1. การตรวจสอบแบบอนุกรม ซึ่งมีขั้นตอนดังนี้

1.1 ATN ถูกดึงเป็น LOW หลังจากได้รับ LOW จากสายสัญญาณ SRQ

1.2 คำสั่ง UNL ถูกสั่งไปยังอุปกรณ์

1.3 ตัวควบคุมจะแจ้งรหัสตัวรับของตน และกำหนดรหัสตัวส่งของอุปกรณ์ที่จะตรวจสอบไปที่บัส

1.4 ตามด้วยคำสั่ง SPE และสาย ATN กลายเป็น HI ซึ่งอุปกรณ์ที่ถูกเรียกจะส่งข้อมูลแสดงสถานะออกมา 1ไบต์โดยบิตที่ 7 จะเป็นตัวชี้ว่าอุปกรณ์เส้นเป็นตัวขอบริการ ถ้าใช่จะเป็น LOW ส่วนบิตอื่นๆ ก็ใช้บอกข้อมูลอื่นๆซึ่งมิได้กำหนดเฉพาะ

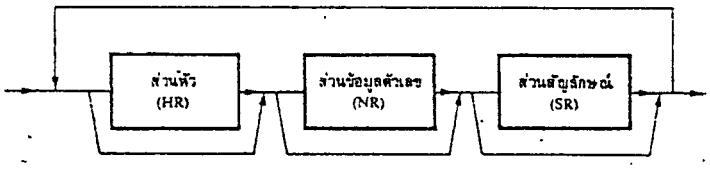
1.5 สาย ATN ถูกดึงเป็น LOW อีกที เพื่อส่งคำสั่งยกเลิกการตรวจสอบคือ SPD

1.6 จากนั้นคำสั่ง UNI ก็ถูกส่งไปยังอุปกรณ์เพื่อยกเลิกการเป็นตัวส่ง ซึ่งถ้าหากSQR ยังคงเป็น LOW อยู่ ก็จะมีการตรวจสอบไปยังอุปกรณ์ตัวอื่นๆต่อไปตามขั้นตอนเดิม

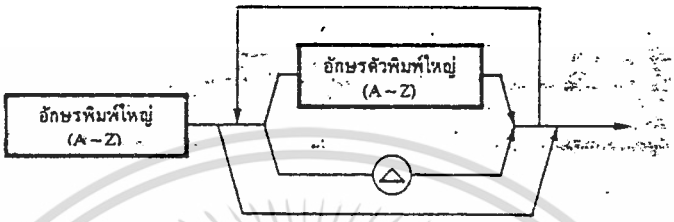
2.การตรวจสอบแบบขนาน สามารถทำได้เร็วกว่าแบบอนุกรมทั้งนี้เพราะสามารถอ่านข้อมูล เพราะสามารถอ่านข้อมูลเพียงไบต์เดียวก็สามารถรู้ได้ทันที ว่าอุปกรณ์ตัวใดเป็นผู้ขอบริการ

รูปแบบของข้อมูล

โดยทั่วไปข้อมูลจากอุปกรณ์ (device message) แบ่งได้ออกเป็น 3 ส่วนดังแสดงในรูปที่ 9 อันได้แก่ส่วนหัว (HR) ซึ่งจะอยู่ส่วนหน้าสุด เป็นตัวบอกชนิดข้อมูล ส่วนประกอบ HR แสดงในรูปที่ 10 จะเห็นว่าประกอบด้วยตัวอักษรภาษาอังกฤษตัวพิมพ์ใหญ่ช่องว่างที่เป็นเว้นวรรค ( ) ปกติจะมีอักษรประมาณ 1-3 ตัว

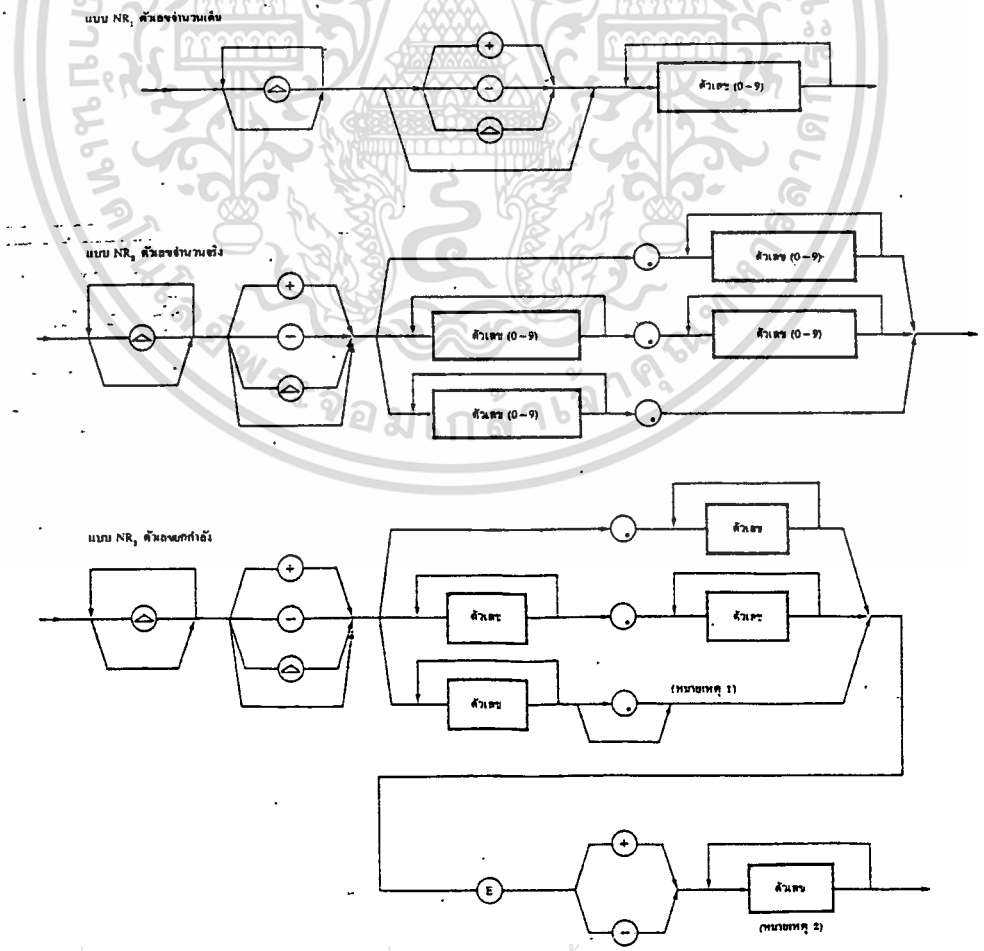


รูปที่ 1.9



รูปที่ 1.10

ส่วนที่สองคือเนื้อหาข้อมูล (NR) ซึ่งใช้แสดงค่าตัวเลข มีอยู่ 3 แบบ คือ NR1,NR2 และ NR3 ดังแสดงในรูปที่ 11 ส่วนท้ายของ NR ยังอาจมีตัวอักษรแสดงหน่วยตามมา

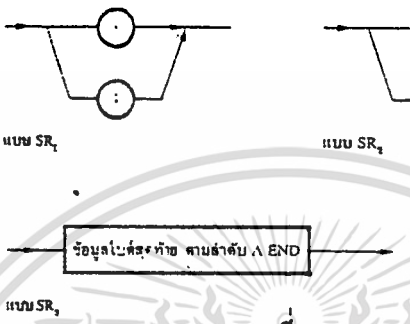


หมายเหตุ 1 : คุณสมบัตินี้ตัวส่งต้องส่งออกไปด้วยทุกครั้ง และตัวรับก็ต้องถือว่ามีค่านี้อยู่

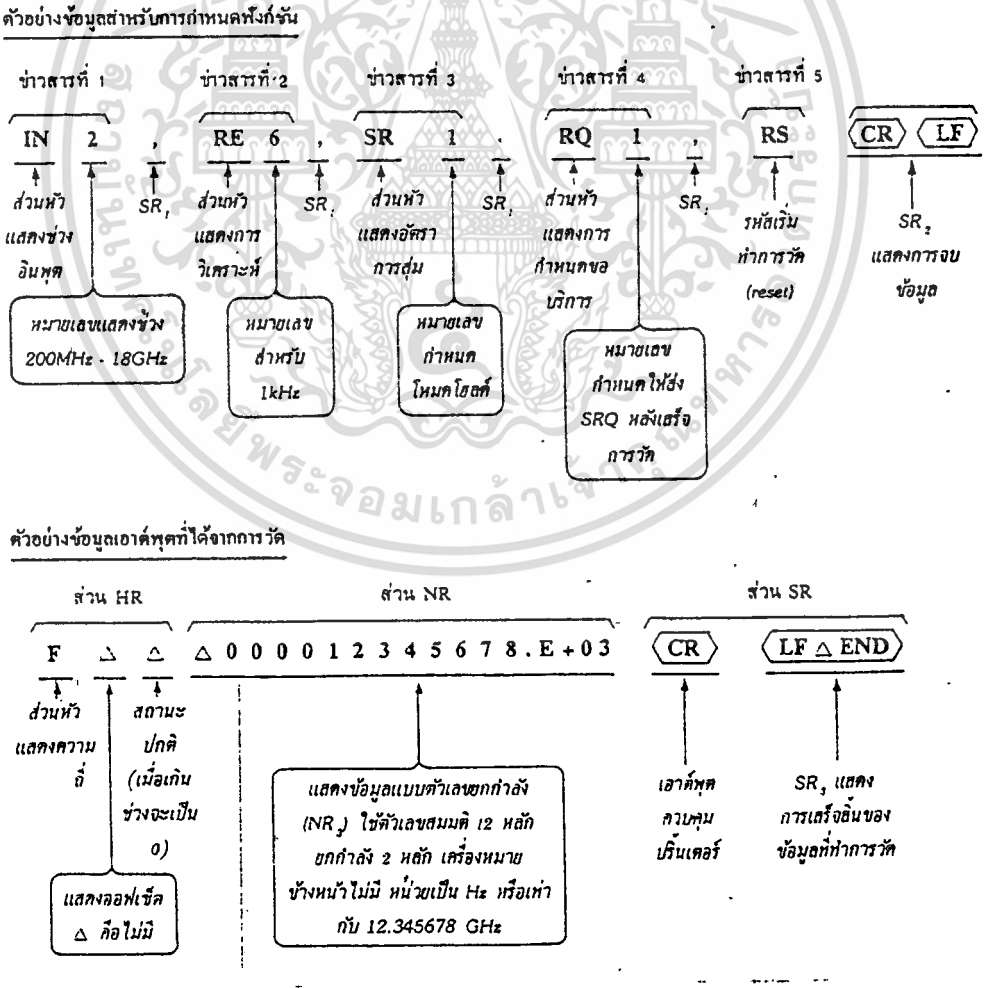
หมายเหตุ 2 : เอกซ์โปเนนชาล 2 หลักเป็นมาตรฐาน แต่สามารถใช้ 1 หลักหรือ 3 หลักก็ได้

รูปที่ 1.11

ส่วนที่ 3 คือสัญญาณแบ่งข้อมูลแต่ละชุด (SR) ดังแสดงในรูปที่ 12 โดย SR1 ใช้แสดงการต่อเนื่องของข้อมูล (ข้อมูลยังมีต่อ) SR2 และ SR3 แสดงการสิ้นสุดของข้อมูล แต่ SR3 เป็นการบอกการเสร็จสิ้นข้อมูลทั้งหมดจากการวัด และรูปที่ 13 เป็นตัวอย่างข้อมูลสำหรับการกำหนดฟังก์ชันให้เครื่องวัดความถี่และข้อมูลความถี่ที่วัดได้



รูปที่ 1.12



รูปที่ 1.13

บทที่ 2
CODE การทำงานของอุปกรณ์ในระบบ

ในโครงงานนี้ใช้ Card IEEE-488 ในการควบคุมเครื่องมือวัดอันได้แก่ Programmable function generator HM8130, Oscilloscope HM1007, Programmable multimeter HM8112-2 และ Power supply HM8142 ซึ่งใช้ NI488.2 routine และ NI-488 function ในการติดต่อกับอุปกรณ์ผ่าน card ได้ ซึ่งรูปแบบของภาษาซีที่ใช้ในการติดต่อผ่าน NI-488 Function มีดังตารางต่อไปนี้

ตารางที่ 2.1 C Language NI-488 Function

Call Syntax	คำอธิบาย
ibbna (ud,bname)	เปลี่ยนอุปกรณ์ที่กำลังติดต่ออยู่
ibcac (ud,v)	Become Active Controller
ibclr (ud)	Clear specified device
ibcmd (ud,cmd,cnt)	Send commands from string
ibconfig (ud,option,value)	Configue the software
ud = ibdev (bd_index,pad sad,tmo,eot,eos)	Open and initialize an unused devices when the device name is unknown
ibdma (ud,v)	Enable/disable DMA
ibeos (ud,v)	Change/disable EOS mode(writ)
ibeot (ud,v)	Enable/disable END message
ibevent (ud,&event)	Return the next event
ud = ibfind (udname)	Open device and return unit descriptor

ตารางที่ 2.1 C Language NI-488 Functions (ต่อ)

Call Syntax	คำอธิบาย
ibgts (ud,v)	Go from active Controller to standby
ibist (ud,v)	Set/clear ind. status bit for parallel polls
iblines (broad,lines)	Get status of GPIB lines
ibln (pad,sad,listen)	Check for presence of device on bus
ibloc (ud)	Go to local
ibonl (ud,v)	Place device or board online/off
ibpad (ud,v)	Change Primary Address
ibpct (ud)	Pass Control
ibppc (ud,v)	Parallel Poll Configue
ibrd (ud,rd,cnt)	Read data to string
ibrda (ud,rd,cnt)	Read data asynchronously tostring
ibrdf (ud,flname)	Read data to file
ibrpp (ud,&ppr)	Conduct a Parallel Poll
ibrsc (ud,v)	Request/release system control
ibrsp (ud,&spr)	Return serial poll byte
ibrsv (ud,v)	Request service,set/change serial poll
ibsad (ud,v)	Change/disable Secondary Address
ibsic (ud)	Send Interface Clear for 100 usec

ตารางที่ 2.1 C Language NI-488 Functions (ต่อ)

Call Syntax	คำอธิบาย
ibsre (ud,v)	Set/clear Remote Enable line
ibsrq (func)	Reqister an SRQ "interrupt routine"
ibstop (ud)	Abort asynchronous operation
ibtmo (ud,v)	Change/disable time limit
ibtrap (mask,mode)	Configue Application Monitor
ibtrg (ud)	Trigger select device
ibwait (ud,mask)	Wait for select event
ibwrt (ud,wrt,cnt)	Write data from string
ibwrta (ud,wrt,cnt)	Write data asychronously from string
ibwrtf (ud,flname)	Write data from file

คำสั่งที่ใช้กับ Function Generator

เมื่อเชื่อมต่อ Function Generator กับ Computor โดยใช้บัสมาตรฐาน IEEE-488 คำสั่งที่Computerจะสามารถควบคุมการทำงานของ Funtion Generator มีดังต่อไปนี้

Commands without data

SIN	Sinewave signal function
TR	Triangle signal function
SQR	Squarewave signal function
PLS	Pulse signal function
RMP	Sawtooth,positive-going
RMN	Sawtooth,negative-going
ARB	Arbitrary function
SW1/0	Sweep mode on/off
CTM	Continuous mode
GTM	Gated mode
TRM	Triggered mode
OT1/0	Output signal on/off
OF1/0	Offset on/off
DFR	Display signal frequency
DST	Display start frequency
DSP	Display stop frequency
DWT	Display pulse width
DSW	Display sweep time

DAM	Display output amplitude
DOF	Display offset voltage
RMO	Disable REMOTE status
LK1	Enable local-inhibit status
LK2	Disable local-inhibit status
TRG	Triggers a signal period (sweep off) or a complete sweep (sweep on)
CLR	Reset the function generator and returns all setting to their default
ARC	Delete all arbitrary data and reset the internal arbitrary counter to zero
ARE	Terminates the aribitrary editor

Available commands

FRQ:<Data>	Set frequency to	<....>Hz
STT:<Data>	Set start frequency to	<....>Hz
STP:<Data>	Set stop frequency to	<....>Hz
SWT:<Data>	Set sweep time to	<....>s
WDT:<Data>	Set pulse width to	<....>s
AMP:<Data>	Set amplitude to	<....>V
OFS:<Data>	Set offset to	<....>V

Commands with whole-number values

STO =<Data>	Store values	<0...8>
RCL =<Data>	Read values	<0...9>

Reset commands

FRQ?	Frequency
STT?	Start frequency
STP?	Stop frequency
SWT?	Sweep time
WDT?	Pulse width
AMP?	Output voltage
OFS?	Offset
ARD?	Arbitrary data
ID?	Device ID
VER?	Equipment version
STA?	Device status

คำสั่งที่ใช้กับ Oscilloscope

DIG [channal-code]	Extraction of stored data from the scope's memory,with channel selection Transfer of data to the controller
GET [channal-code]	Activation of the RESET TRIGGER function at the scope,without channel selection
STA	Query for osilloscope channel setting
TET	Text transfer to the interface with the signal-data to the device

คำสั่งที่ใช้กับ Programmable Multimeter

VD เลือกฟังก์ชันการวัดเป็น direct voltage

VA เลือกฟังก์ชันการวัดเป็น alternating voltage

O2 เลือกวัดค่า resistance แบบ 04=4-wire

ID เลือกวัดค่า direct current

IA เลือกวัดค่า alternating current

TC,TK,TF เลือกวัดค่าอุณหภูมิตามฟังก์ชัน

RX เป็นการเลือกตั้งค่าขอบเขตของการวัดในฟังก์ชันต่างๆ โดย X จะแปรค่าเป็นดังนี้

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

A0/A1 (A/Zero) switches off autoranging;
A1 autoranging on

TX ตั้งค่า integration time และจำนวนทศนิยมที่จะแสดงบนจอ display ซึ่งจะส่งค่าได้ถึง 6 1/2 digit ผ่าน IEEE 488 Bus

Z0 Starts an offset correction

S0 (S/Zero) Starts the continuous measuring sequence.

S1 Stop the cont. measuring sequence,

L0 (L/Zero)Short format: DMM จะส่งค่าเฉพาะ data block แรก

L1 Long format: DMM จะส่งค่าทั้งหมดที่สามารถอ่านได้

P1 แสดงค่า offset-corrected value R = X-C

P2 % Deviation R = 100 x (X-C)/C

P3 dB R = 20 x log(X/C)

P4 dBm R = 20 x log(X/C) with

C = 0.775 V into 600 Ohm for voltages

and C = 1.29 mA for current

PxEN Enter meas. value for constant x at P 2-4; X=1,2,3

ID? ส่งค่า instruments identification "HM8112-2"

STA? ส่งค่าแสดงสถานะของอุปกรณ์ (ทั้งสอง data block)

D0/D1 0=Display off; 1=Display active

คำสั่งที่ใช้ Power Supply

RM1/RM0 : REMOTE ON/OFF

LK1/LK0 : LOCAL ON/OFF

SU1 : SET VOLTAGE 1

SU2 : SET VOLTAGE 2

SI1 : SET CURRENT 1

SI2 : SET CURRENT 2

RU1 : RECALL VOLTAGE 1

RU2 : RECALL VOLTAGE 2

RI1 : RECALL CURRENT 1

RI2 : RECALL CURRENT 2

MU1 : MEASURE VOLTAGE 1

MU2 : MEASURE VOLTAGE 2

MI1 : MEASURE CURRENT 1

MI2 : MEASURE CURRENT 2

MX1/MX0 : MIXED MODE ON/OFF

OP1/OP0 : OUTPUT ON/OFF

TRU : TRACK VOLTAGE

TRI : TRACK CURRENT

SR1/SR0 : SERVICE REQUEST ON/OFF

ABT : ARBITARY WAVEFORM MODE

RUN : START ARBITARY

STP : STOP ARBITARY

ABX : EXIT ARBITARY

VER : GET VERSION

STA : GET STATUS

ID? : GET IDENTIFIER

บทที่ 3
LabWindows/CVI

ในการควบคุมอุปกรณ์ต่างๆ ภายในระบบนี้เราควบคุมผ่าน Card IEEE-488 ผ่านโปรแกรมที่มีชื่อว่า LabWindows/CVI ต่อไปนี้จะเป็นการแนะนำกว้างๆ เกี่ยวกับ LabWindows/CVI

ลักษณะของ LabWindows/CVI

LabWindows/CVI เป็นโปรแกรมที่ช่วยให้เราสามารถเขียนและพัฒนาโปรแกรมต่อไปได้โดยเขียนในรูปแบบของภาษาซี และมีลักษณะเป็นหน้าต่างเหมือนในโปรแกรม Windows ในตัวโปรแกรม LabWindows/CVI เอง จะมี function และ library ต่างๆ ที่จะช่วยในการเขียนและพัฒนาโปรแกรมต่อไป ไม่ว่าจะเป็นโปรแกรมเกี่ยวกับการกระทำ data acquisition หรือ การประยุกต์ควบคุมอุปกรณ์ต่างๆ

ลักษณะของ LabWindows/CVI จะมี interactive environment (คือลักษณะทั่วไปที่ใช้ในการ editing, compiling, linking และ debugging โปรแกรม ANSI C) และภายใต้ environment เหล่านี้เราสามารถที่จะใช้ function ต่างๆ ใน library ของ LabWindows/CVI ได้

ลักษณะของแต่ละ function จะมีลักษณะเป็นหน้าต่างที่เรียกว่า function panel ในการใช้เราจะตั้งค่าต่างๆ ผ่าน function panel นี้ แล้วมันจะแสดง code ที่ตั้งค่าเหล่านี้โดยอัตโนมัติ ดังนั้นเราสามารถที่จะคัดลอก code ไปใส่ไว้ในโปรแกรมของเราได้เลย

จะเห็นว่าประสิทธิภาพของ LabWindows/CVI นี้จะขึ้นอยู่กับ library ซึ่งใน library ต่างๆ เหล่านี้จะมี function ที่สามารถนำมาใช้ได้ ตัวอย่างการใช้

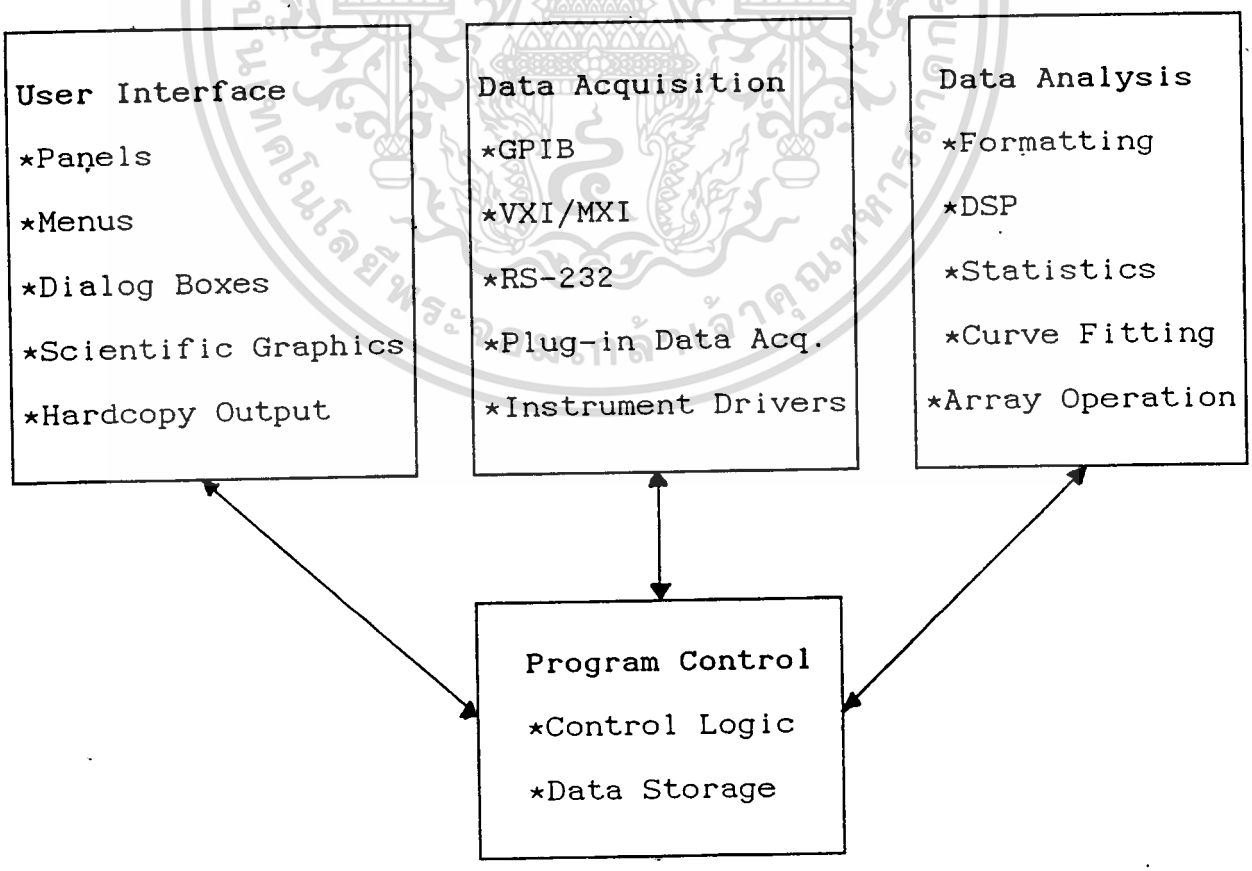
- สำหรับการกระทำ data acquisition จะมี library เกี่ยวกับ Instrument Library, GPIB/GPIB 488.2 Library, Data Acquisition Library และ VXI Library
- สำหรับการกระทำ data analysis จะมี library เกี่ยวกับ Formatting และ I/O Library, Analysis Library และ Advanced Analysis Library

- สำหรับการกระทำ data presentation จะมี library เกี่ยวกับ User Interface Library
- สำหรับการกระทำการประยุกต์การติดต่อสื่อสารต่างๆ จะมี library เกี่ยวกับ Dynamic Data Exchange (DDE) และTransmission Control Protocol(TCP)

ใน Instrument Library นั้น จะเป็นที่เก็บของ driver ของอุปกรณ์ต่างๆ ที่ผ่านการควบคุมทาง GPIB, VXI, RS-232 เช่น oscilloscope, multimeters ซึ่งอุปกรณ์แต่ละชนิดนี้ ถ้าต่างชนิด ต่างยี่ห้อกัน ก็จะมีคำสั่งควบคุมอุปกรณ์เหล่านี้ต่างกันไปด้วย ขึ้นอยู่กับบริษัทที่ผลิตอุปกรณ์นั้นๆ ดังนั้นถ้าเราจำเป็นที่จะต้องควบคุมอุปเหล่านี้ เราจะต้องเขียน driver ของอุปกรณ์ชนิดนั้นๆ ขึ้นมาก่อน

โครงสร้างของโปรแกรมใน LabWindows/CVI

ในการเขียนโปรแกรมหนึ่งขึ้นมาบน LabWindows/CVI นั้น จะประกอบด้วยโครงสร้างส่วนต่างๆ ดังนี้



รูปที่ 3.1

User Interface

การออกแบบการติดต่อระหว่างผู้ใช้กับโปรแกรมเรา ในการออกแบบการ interface จะเป็นตัวกำหนดว่าเราจะให้โปรแกรมเรานั้นทำอะไร

User Interface จะกำหนดถึงการใส่ค่า input ต่างๆ ลงในโปรแกรม รวมถึงการแสดงค่า output ต่างๆ ออกมา โดยจะผ่านทาง menu, panel, control และ dialog box

Program Control

จะเป็นส่วนที่ควบคุมการทำงานของ user interface, data acquisition และ data analysis

Data Analysis

หลังจากที่ได้รับข้อมูลเข้ามาแล้ว ก็จะเป็นการนำเอาข้อมูลนั้นมาประมวลผล รวมถึงการคำนวณหาค่าต่างๆ ไม่ว่าจะเป็นการ formatting, scaling, signal processing, statistics และ curve fitting

Data Acquisition

เป็นส่วนที่จะรับข้อมูลเข้ามาจากอุปกรณ์ต่างๆ ซึ่งส่วนนี้จะจัดเตรียมข้อมูลดิบไว้เพื่อใช้ในการ analysis และการ นำเสนอสู่ผู้ใช้ ในขั้นตอนอื่นของโปรแกรมต่อไป

มีฟังก์ชันสำหรับใช้ในการควบคุมผ่านทาง GPIB, RS-232 และ อุปกรณ์VXI รวมถึง Instrument Drivers ซึ่งในการติดต่อกับอุปกรณ์ใดๆ จำเป็นจะต้องมีส่วนนี้ก่อน

บทที่ 4
Instrument Driver

ดังที่กล่าวไว้ในบทที่แล้วในการที่จะเขียนโปรแกรมขึ้นมาสักชุดหนึ่งโดยเฉพาะในการทำงานที่เป็นระบบแล้ว จำเป็นที่จะต้องเขียน Instrument Driver ขึ้นมาก่อน เพราะ Instrument Driver นี้ จะเหมือนเป็นตัวกลางที่ใช้ในการเชื่อมต่อระหว่างโปรแกรมหลักของเรากับอุปกรณ์ที่เราต้องการติดต่อด้วย ดังรายละเอียดที่จะอธิบายถึง Instrument Driver ต่อไปนี้

Instrument Library และ Instrument Driver

Instrument Library เป็น library หนึ่งใน LabWindows/CVI ที่บรรจุ Instrument driver ไว้ Instrument driver จะช่วยอำนวยความสะดวกในการใช้อุปกรณ์นั้นๆ โดยผู้ใช้ไม่ต้องไปยุ่งกับคำสั่งเฉพาะ (programming protocol) ของอุปกรณ์นั้นๆ เลย โปรแกรมใน Instrument driver จะทำการควบคุมอุปกรณ์นั้นๆ โดยเฉพาะ

นอกเหนือจากการควบคุมอุปกรณ์นั้นๆ แล้ว Instrument driver ยังทำการจัดรูปแบบข้อมูลดิบจาก instrument ให้เป็นข้อมูลที่ผู้ใช้สามารถเข้าใจได้ เช่น driver สามารถเปลี่ยน binary array เป็น ASCII string หรือเปลี่ยน ASCII string ในรูป coordinate x-y ให้เป็น array ของ integer เพื่อให้สามารถ plot ได้

Instrument driver ประกอบด้วย 4 ไฟล์ ดังนี้

-Instrument driver program จะอยู่ในรูปของ .lib, .obj หรือ .c ไฟล์ ในโปรแกรมที่สร้างขึ้นนี้ใช้ .c ไฟล์

-Instrument include (.h) file จะรวบรวม function declaration ต่างๆ และการกำหนดค่าคงที่

-Instrument function panel file (.fp) กำหนดข้อมูลในการกำหนดรูปแบบของ function tree, function panel และ help text

-ASCII text file (.doc) จะบรรจุ documentation ต่างๆ

วิธีใช้ Instrument Driver

สำหรับผู้ใช้แล้ว Instrument driver จะนำเสนอฟังก์ชันต่างๆ ที่ใช้ในการควบคุมอุปกรณ์นั้นๆ เฉพาะแล้วแต่ฟังก์ชันไป

ผู้ใช้จะเลือก Instrument driver จาก Instrument menu ขึ้นมา หลังจากนั้นผู้ใช้ก็จะเลือกใช้ function ต่างๆ ใน instrument driver นั้น โดยรูปแบบ function ของ instrument driver จะเป็นในลักษณะของ function panel (มีลักษณะเป็นหน้าต่าง แล้วมีปุ่มต่างๆ ให้สามารถตั้งค่าของ function นั้นได้) ซึ่งความสามารถของ instrument driver function panel มีดังนี้

1 กระทำการควบคุมอุปกรณ์นั้น

2 สามารถสร้าง function call ซึ่งเราสามารถนำไปใช้ใน program application ได้

กล่าวโดยสรุป instrument driver จะรวมเอาฟังก์ชันต่างๆ ที่ใช้ในการควบคุมอุปกรณ์นั้นๆ ไว้ด้วยกัน โดยผู้ใช้สามารถควบคุมอุปกรณ์นั้นๆ ได้โดยไม่ต้องรับรู้ถึงคำสั่งเฉพาะ (programming protocal) ของอุปกรณ์นั้นๆ เลย

บทที่ 5
ผลการทดลอง

สำหรับการทำ project ในเทอมนี้เป็นการเขียนโปรแกรม instrument driver สำหรับอุปกรณ์ดังต่อไปนี้คือ Programmable Function Generator HM8130 Power Supply HM8142, Programmable Multimeter HM8112-2 และ Analog Digital Oscilloscope HM1007

ขั้นตอนในการเขียนโปรแกรม Instrument Driver มีดังต่อไปนี้

1 ตั้งชื่อโปรแกรม instrument driver นั้น

2 กำหนด function สำหรับอุปกรณ์นั้นๆ

3 สร้าง function tree และใส่ help information สำหรับแต่ละ function ด้วย

4 สำหรับแต่ละ function ใน driver

-กำหนด parameter ของแต่ละ function ไม่ว่าจะเป็นชนิด ขอบเขต หรือ error code

-สร้าง function panel ในแต่ละ function

-เพิ่มเสริมคำสั่งควบคุมเพื่อให้เหมาะกับการใช้งาน

-เขียน code สำหรับ function นั้นๆ

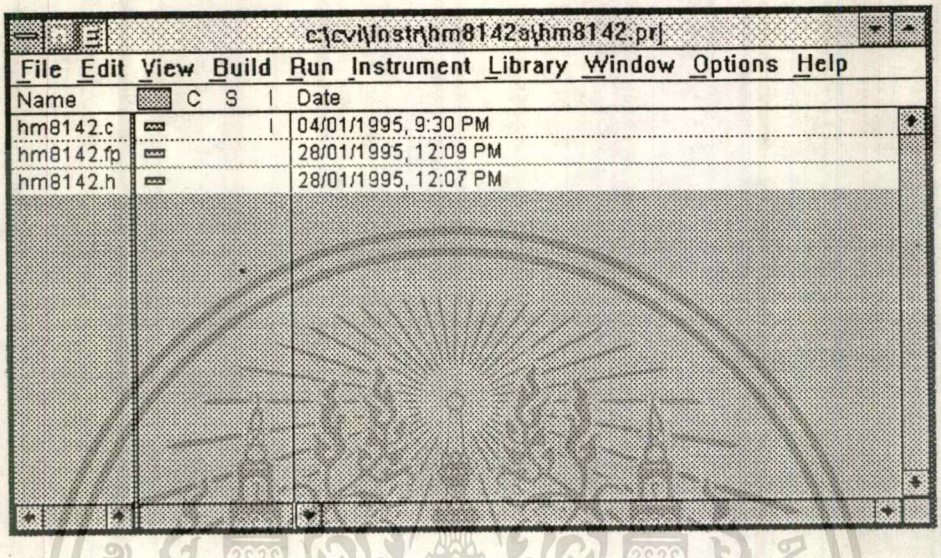
-ทดสอบ source code ที่เขียนขึ้น

5 สร้าง include file สำหรับ source code ที่เขียนขึ้นนั้น รวมถึงการประกาศตัวแปรและการกำหนดค่าคงที่ต่างๆ

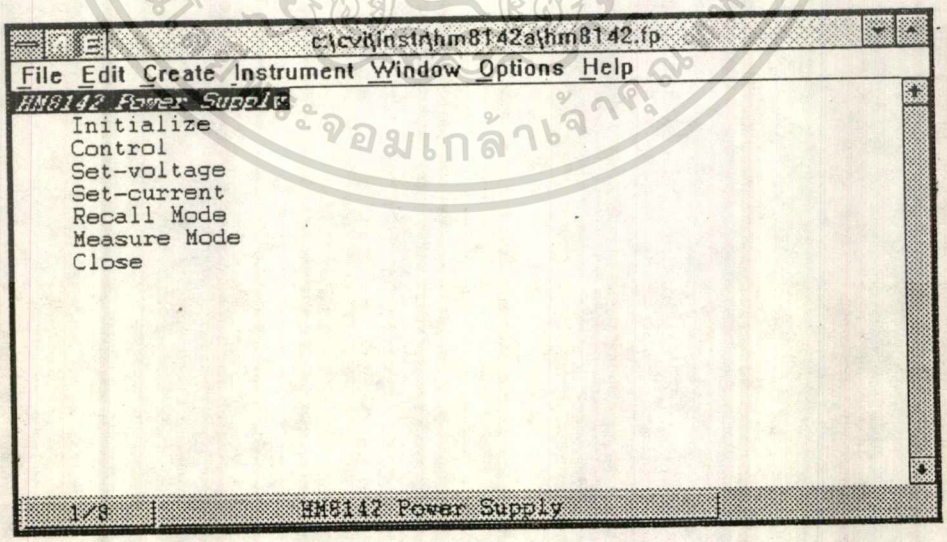
6 ทดสอบโปรแกรม driver ที่สร้างขึ้น

ซึ่งขั้นตอนในการเขียนโปรแกรม driver นี้ได้นำเอาหน้าจอที่เขียนขึ้นมาแสดงให้เห็นดังต่อไปนี้

Power Supply HM8142



รูปที่ 5.1 แสดง project file ซึ่งประกอบด้วย .c, .fp และ .h เป็นการนำโปรแกรมแต่ละส่วนมา link กันเพื่อให้สามารถทำงานร่วมกันได้

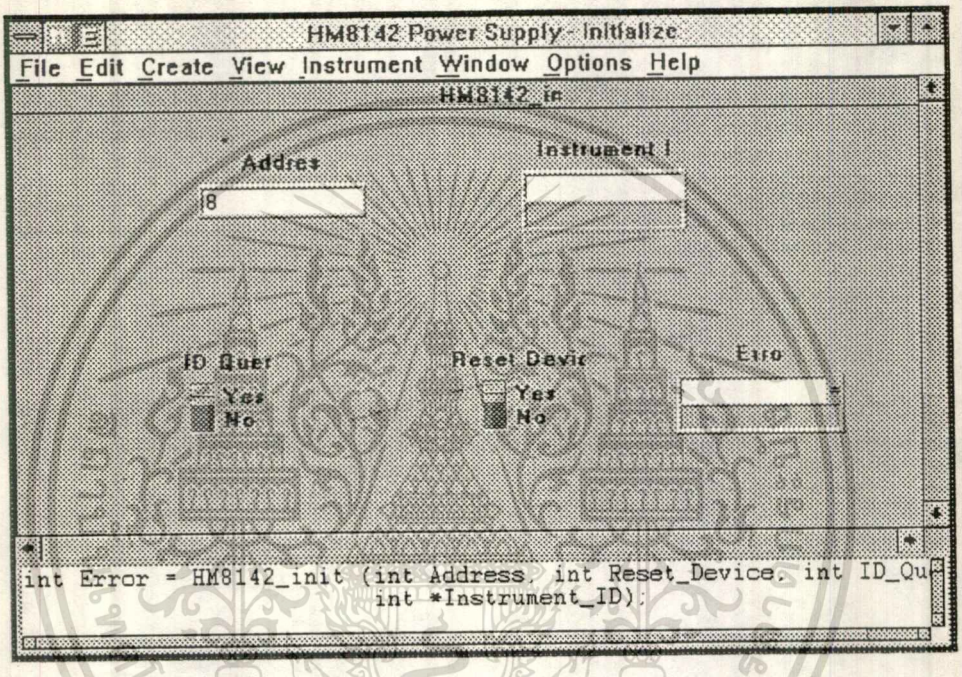


รูปที่ 5.2 แสดง function tree

function tree ของ power supply ประกอบด้วย function ต่างๆ ดังรูปข้างต้น

ต่อไปนี้เป็นการแสดง function ต่างๆ ที่ใช้ควบคุม Power supply ตามโครงสร้างของ Function tree

1. **Initialize** เป็นการ set ค่า address ของ power supply เพื่อให้การทำงานของ power supply เข้าสู่การควบคุมของ controller



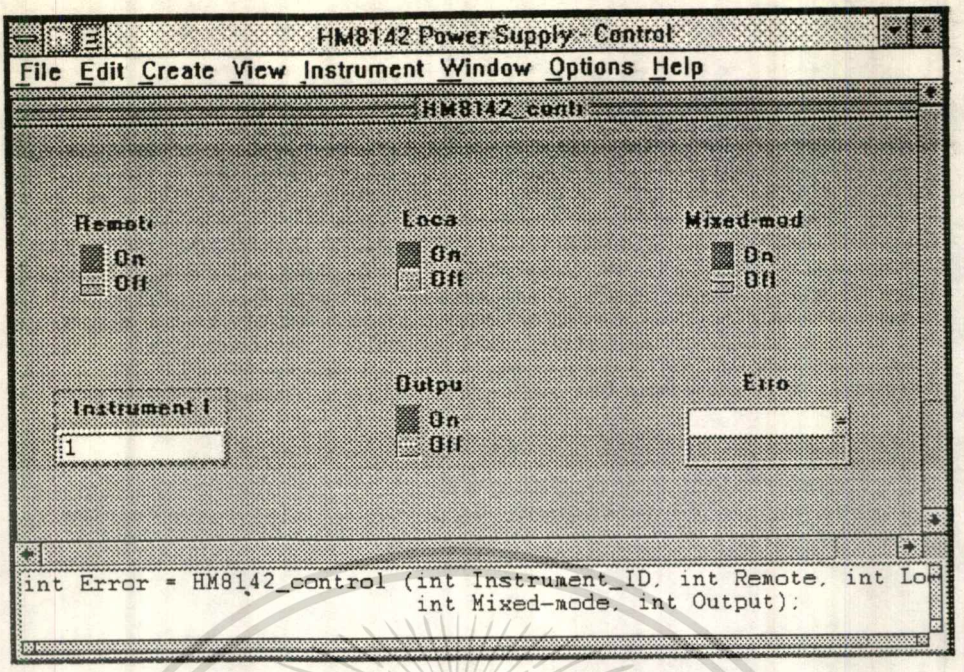
รูปที่ 5.3 Power Supply - Initialize

2. **Control** เป็นการตั้งค่า remote แบบต่างๆ

-ถ้าตั้งแบบ local จะไม่สามารถออกจากการควบคุมโดยการกดปุ่มที่ตัวเครื่องได้

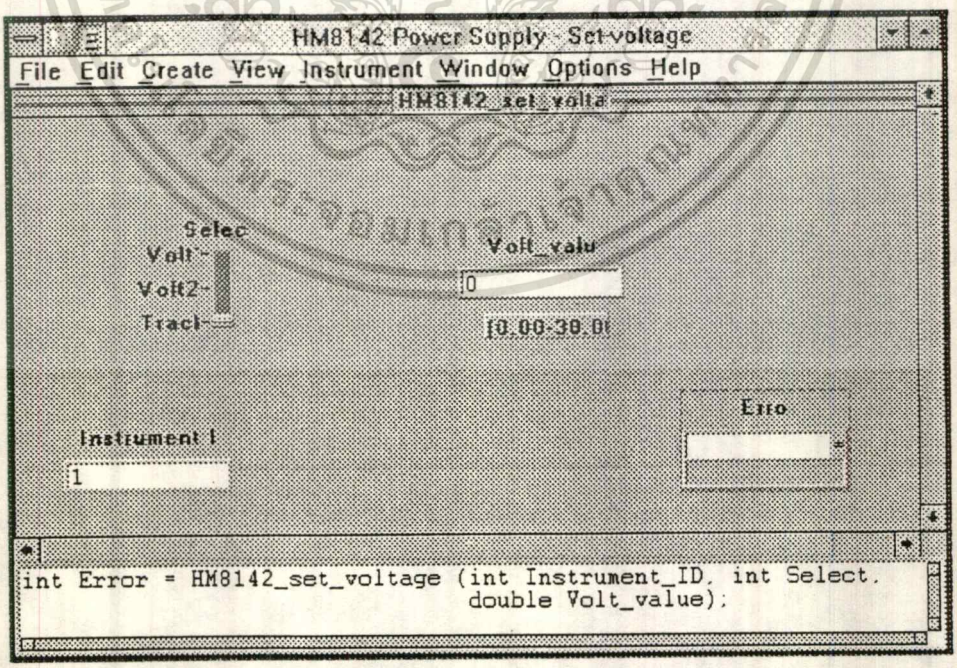
-ถ้าตั้งแบบ mixed mode สามารถควบคุมได้ทั้งจากตัวเครื่องอุปกรณ์โดยตรงหรือจากตัวควบคุมก็ได้

รวมถึงการ On/Off ของ Output



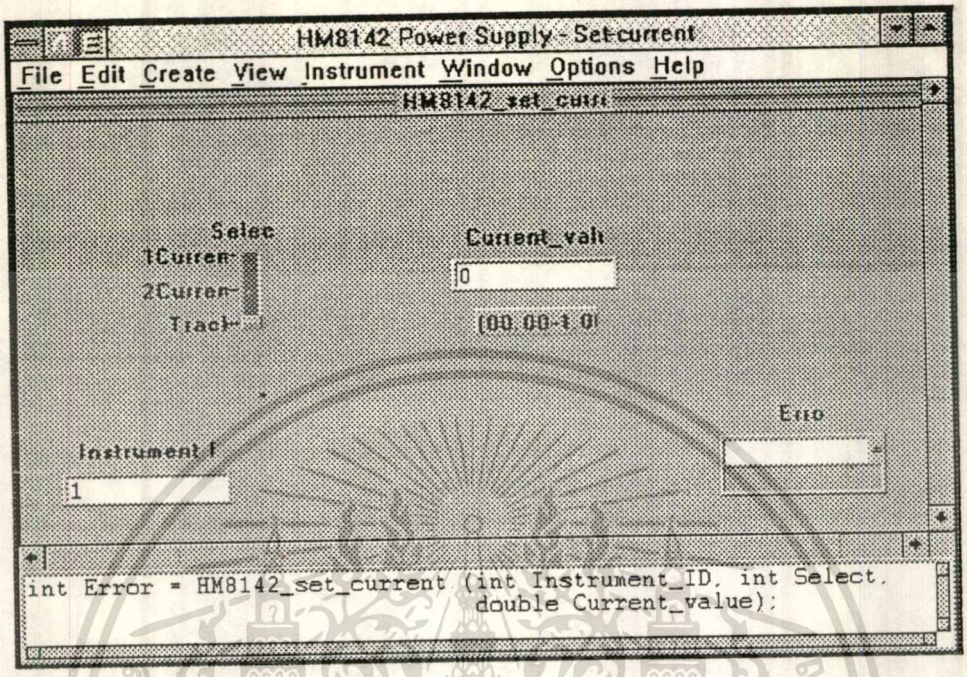
รูปที่ 5.4 Power Supply - Control

3. Set-voltage เป็นการตั้งค่า voltage ที่ต้องการโดยสามารถเลือกใช้ voltage1, voltage2 หรือ track voltage เป็นการตั้งค่าให้เหมือนกันทั้ง voltage1 และ voltage2



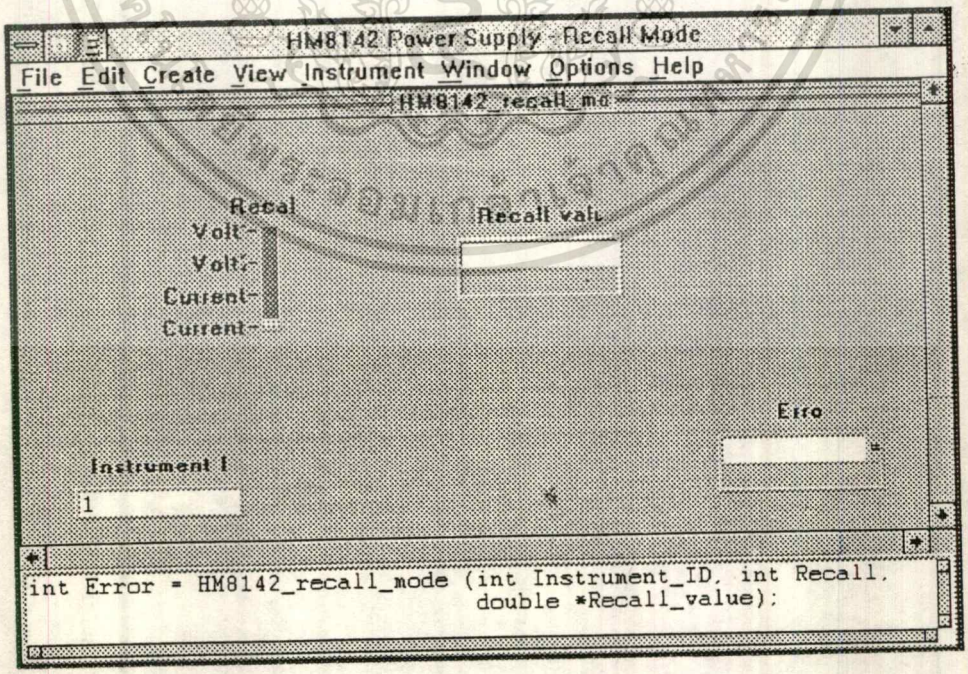
รูปที่ 5.5 Power Supply - Set voltage

4. Set-current เป็นการตั้งค่ากระแส (0-1A) โดยเลือกตั้งค่าที่ current1, current2 และ track current



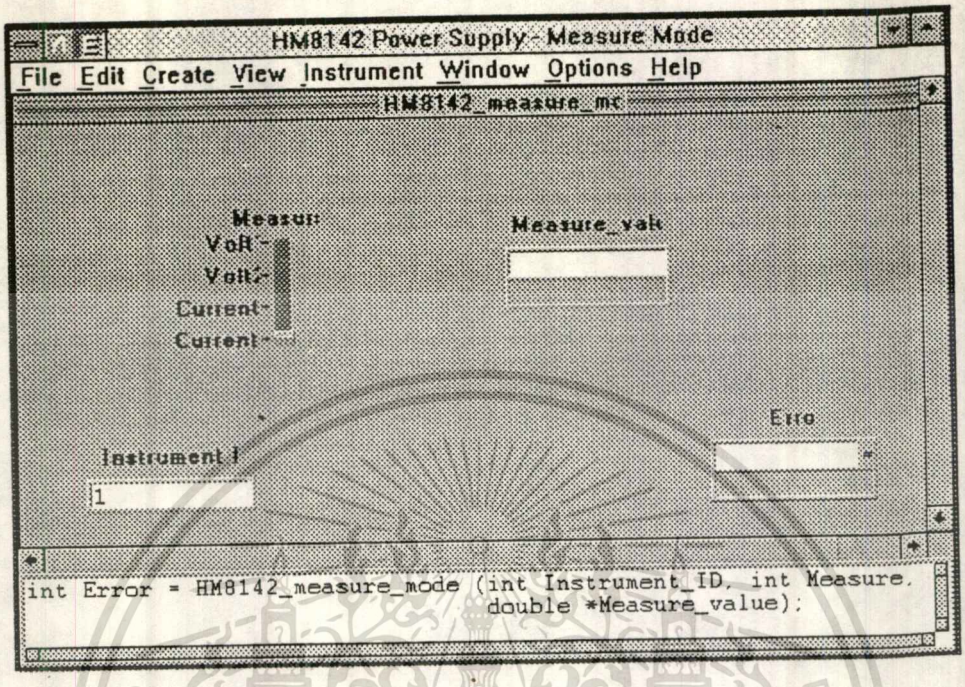
รูปที่ 5.6 Power Supply - Set current

5. Recall Mode เป็นการแสดงค่า voltage หรือ กระแส ที่ตั้งจากโปรแกรม



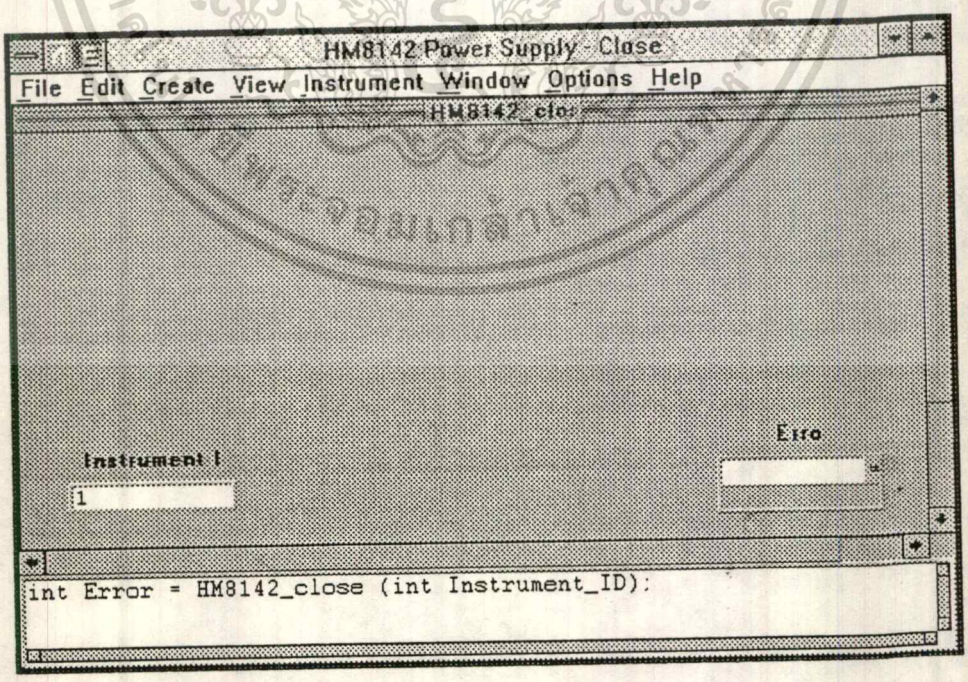
รูปที่ 5.7 Power Supply - Recall mode

6. Measure Mode เป็นการแสดงค่า voltage หรือ กระแส ที่ power supply จ่ายออกมาจริง



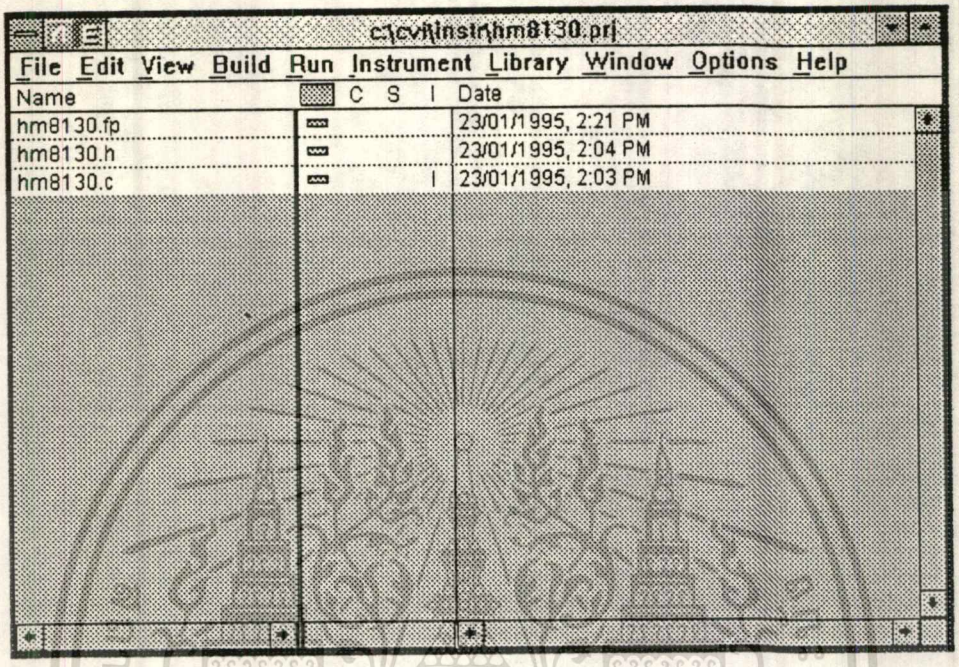
รูปที่ 5.8 Power Supply - Measure mode

7. Close เป็นการออกจากการควบคุม

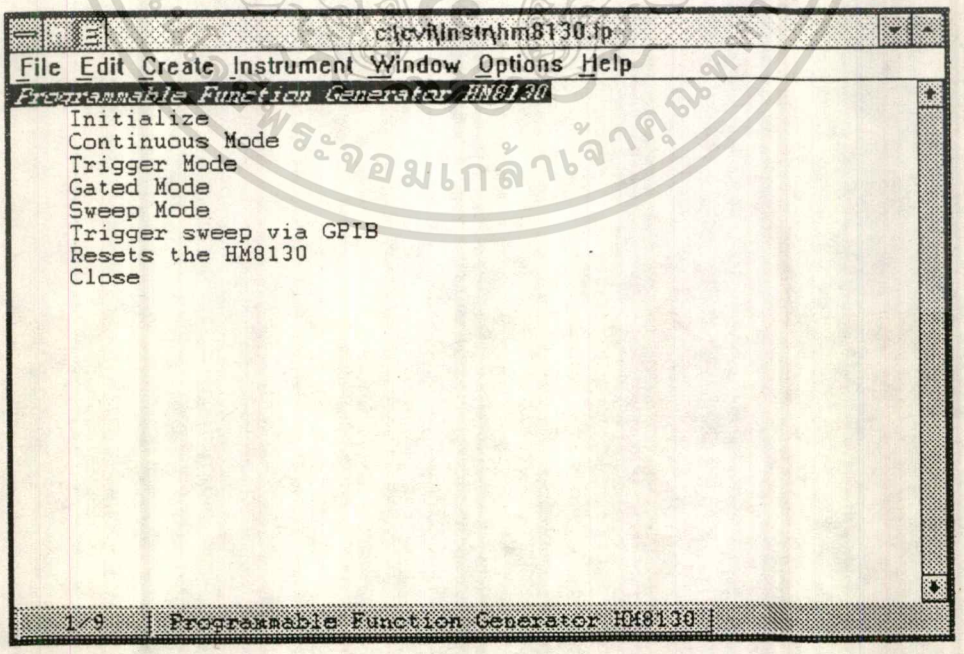


รูปที่ 5.9 Power Supply - Close

Programmable Function Generator HM8130



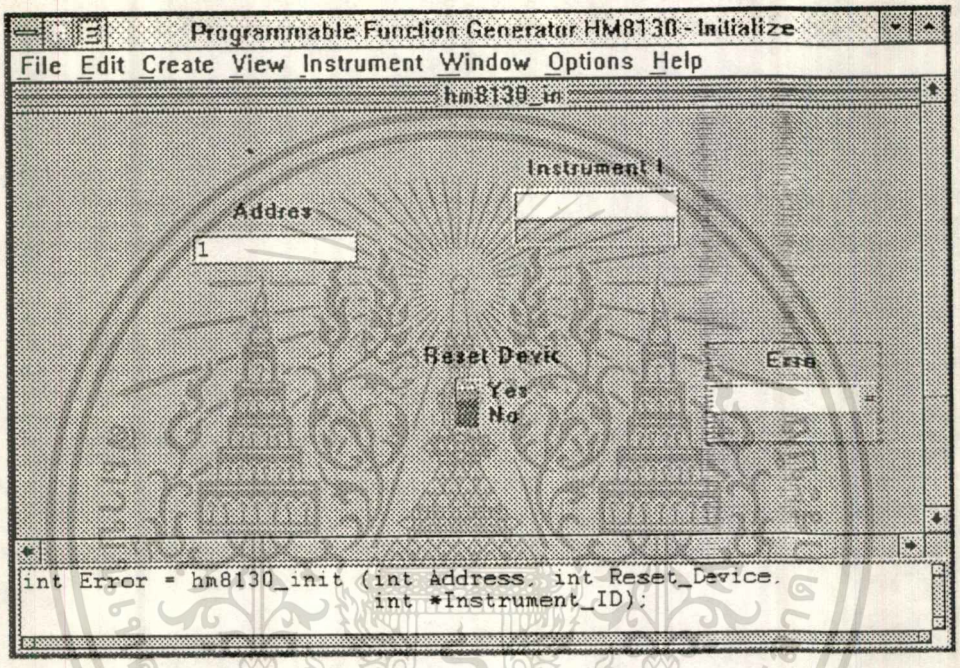
รูปที่ 5.10 เป็นการแสดง project file ของ HM8130



รูปที่ 5.11 เป็นการแสดง function tree ของ Function generator

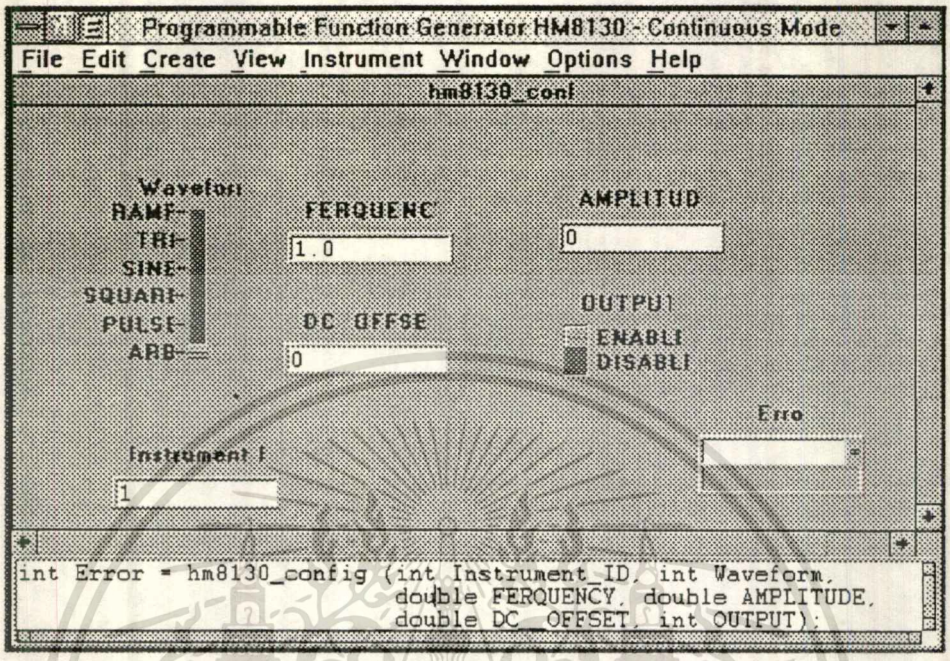
function ต่างๆ ภายใน function tree ประกอบด้วยส่วนต่างๆ ดังต่อไปนี้

1. Initialize เป็นการตั้งค่า address ของ function generator และ ความต้องการที่จะเข้าสู่ความควบคุมหรือไม่



รูปที่ 5.12 Function Generator - Initialize

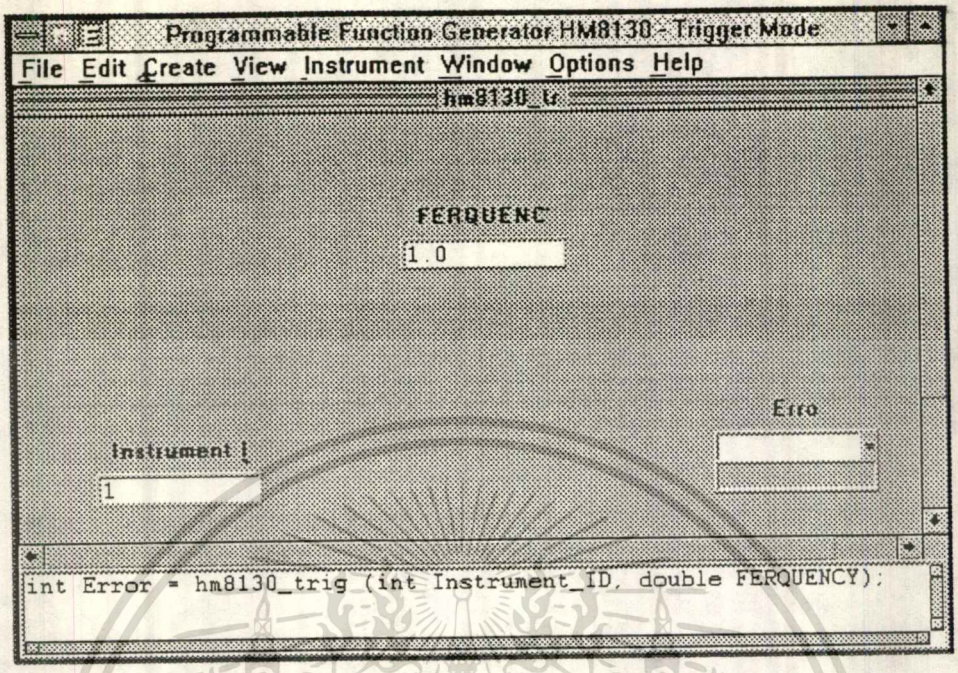
2. Continuous Mode เป็นการ set ให้ function generator ทำงานในโหมดปกติ ประกอบด้วยการกำหนด waveform การตั้งค่าความถี่ และ amplitude การกำหนดค่า dc offset การกำหนดสถานะของ output (enable/disable)



รูปที่ 5.13 Function generator - Continuous mode

3. Trigger Mode จะมีการป้อน trigger signal เข้าไปที่ตัวอุปกรณ์นั้น ดังนั้นสัญญาณ output ที่กำเนิดออกมาจะเหมือนถูกควบคุมโดยสัญญาณ trigger ที่ใส่เข้าไปที่ตัวอุปกรณ์นั้น คือสัญญาณ output จะมีได้เมื่อได้รับสัญญาณ trigger และสัญญาณ output นั้นจะค่อยๆ ลดค่า amplitude เหมือนลักษณะสัญญาณ burst และจะมีค่าขึ้นมาใหม่เมื่อได้รับสัญญาณ trigger อีก

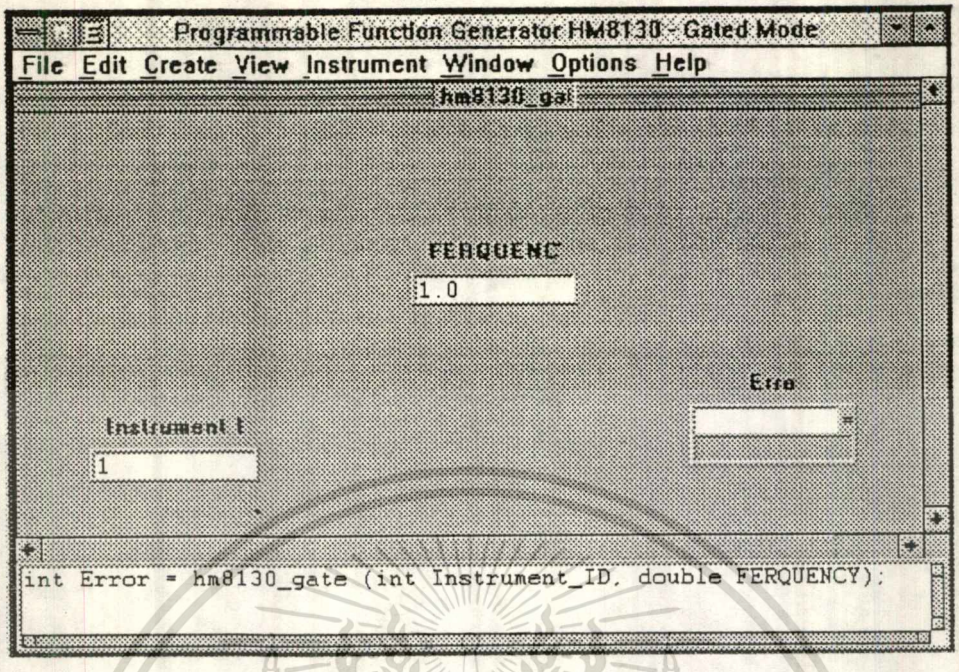
ใน function นี้จะเป็นการตั้งค่าความถี่ของสัญญาณที่จะถูก trig



รูปที่ 5.14 Function Generator - Trigger Mode

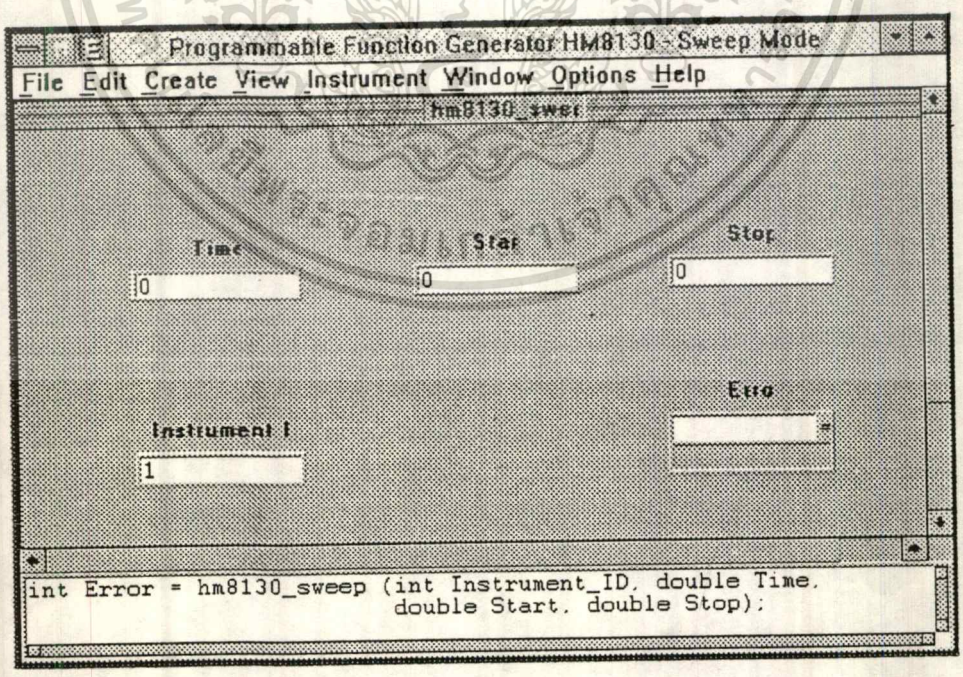
4. Gated Mode สัญญาณ output จะถูกควบคุมโดยสัญญาณที่ใส่เข้าไปที่ gate / trigger input ที่ตัวอุปกรณ์ การทำงานในโหมดนี้จะเป็นในลักษณะแบบ asynchronous ถ้าหากว่าสัญญาณ input ที่ใส่เข้าไปเป็นในลักษณะของสัญญาณ TTL (คือ มีค่า high กับ low) จะมีสัญญาณ output ต่อเมื่อสัญญาณ input เป็น high และจะไม่มีสัญญาณ output เมื่อสัญญาณ input เป็น low

ในฟังก์ชันนี้จะเป็นการตั้งค่าความถี่ของสัญญาณ output



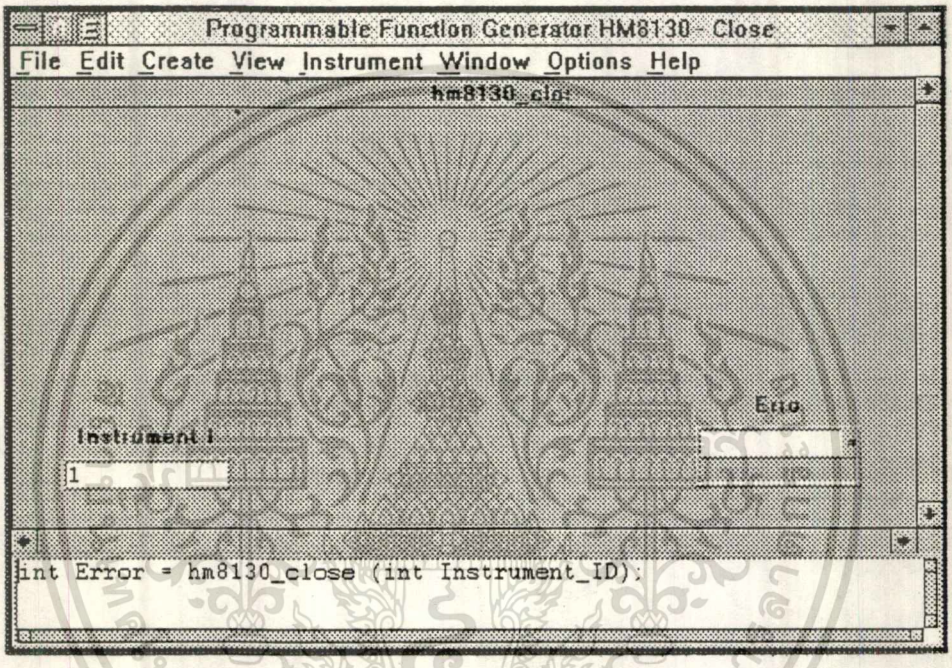
รูปที่ 5.15 Function Generator - Gated Mode

5. Sweep Mode เป็นการตั้งค่าความถี่เริ่มต้นและความถี่สุดท้าย รวมถึงช่วงเวลาในการกวาดความถี่เริ่มต้นถึงความถี่สุดท้ายที่ตั้งไว้นั้น



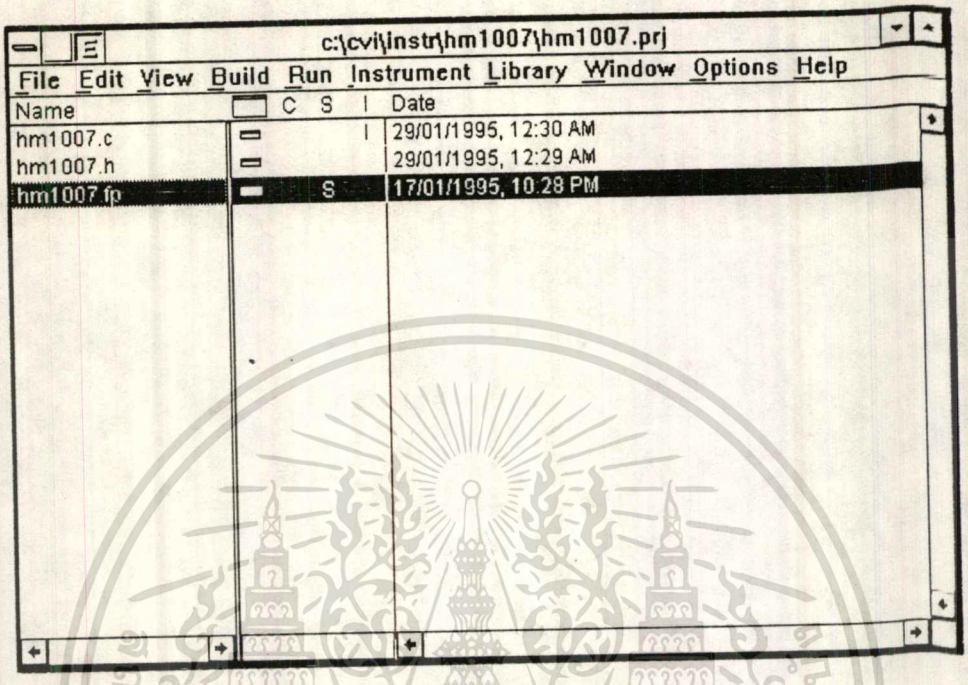
รูปที่ 5.16 Function Generator - Sweep mode

6. Trigger sweep via GPIB เป็นการตั้งค่าความถี่ในการ sweep
7. Resets เป็นการ reset ค่าต่างๆ ภายใน Function generator
8. Close เป็นการออกจากการควบคุม

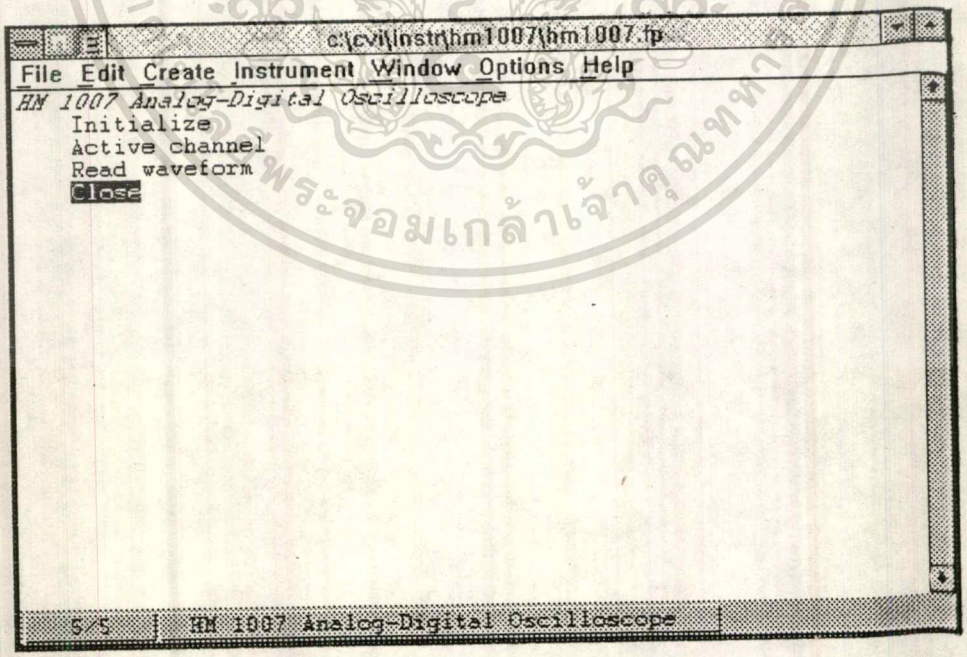


รูปที่ 5.17 Function Generator - Close

Analog Digital Oscilloscope HM1007



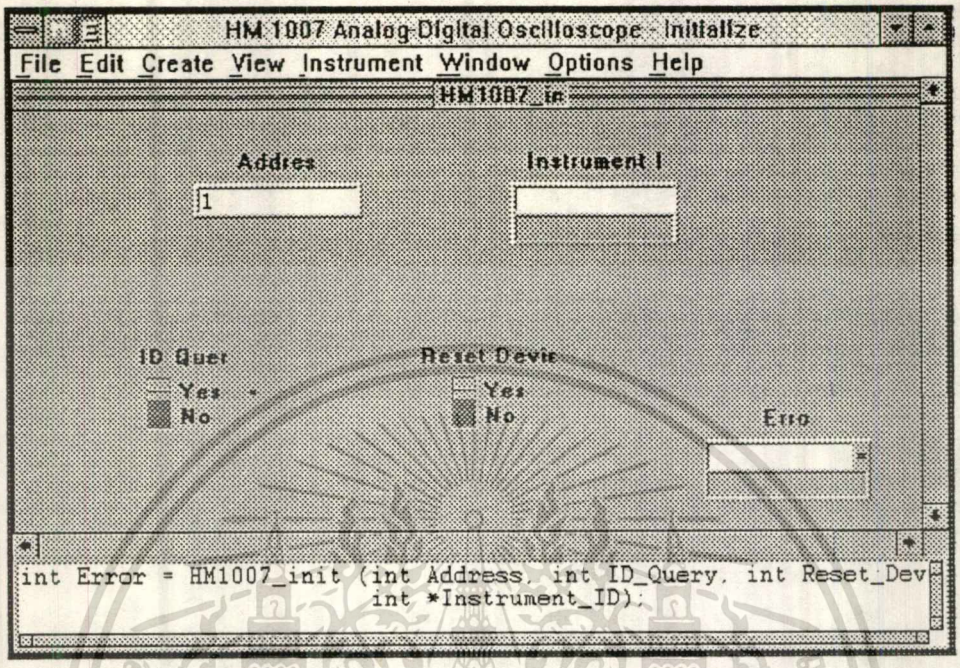
รูปที่ 5.18 เป็นการแสดง project file ของ Oscilloscope



รูปที่ 5.19 เป็นการแสดง function tree ของ Oscilloscope

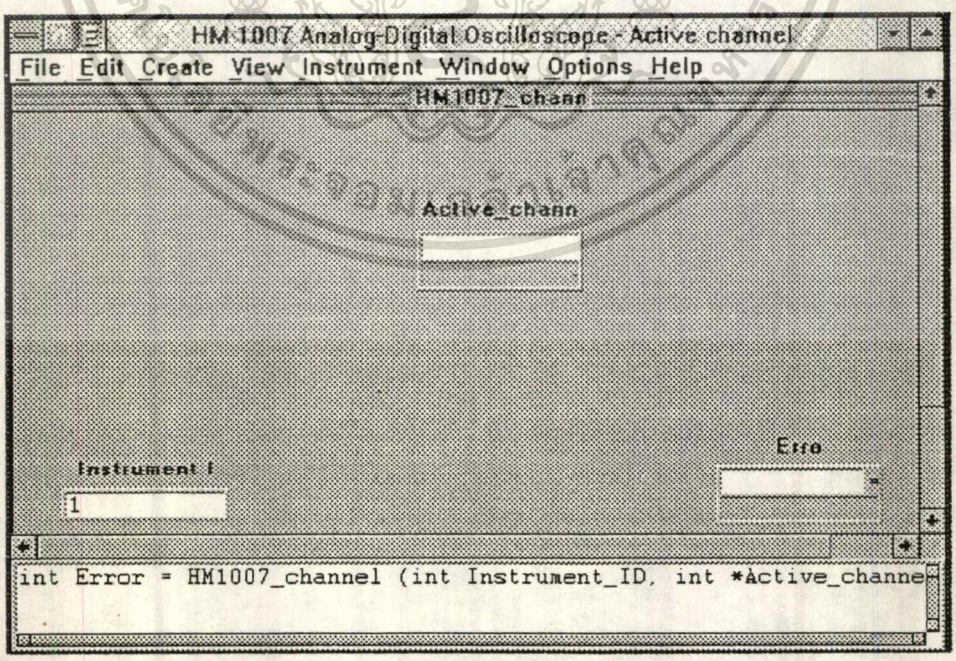
function ต่างๆ ภายใน function tree ประกอบด้วยส่วนต่างๆ ดังนี้

1. Initialize



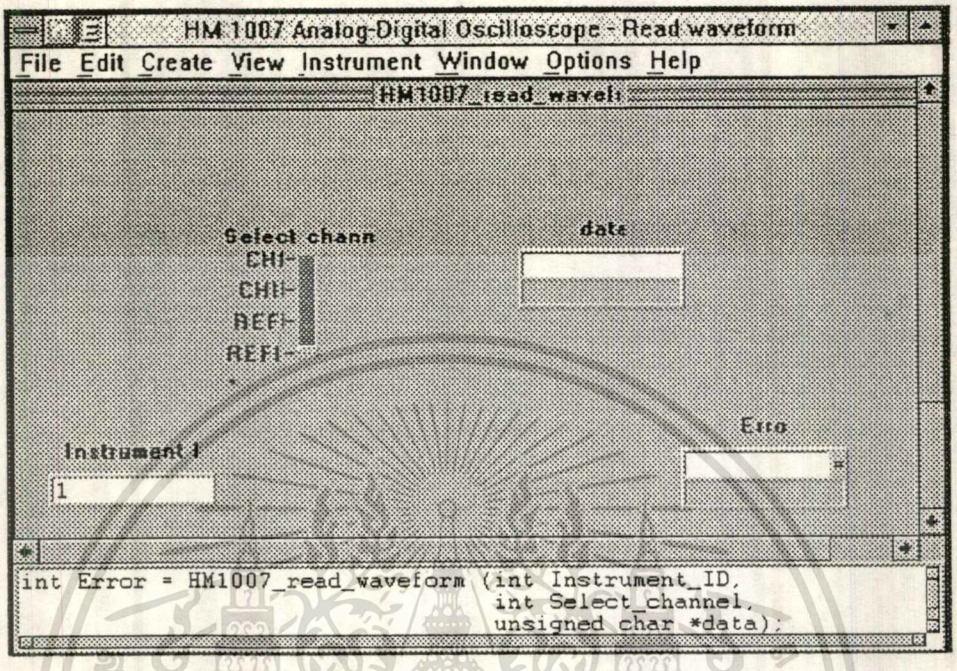
รูปที่ 5.20 Oscilloscope - Initialize

2. Active Channel เป็นการเลือก channel ของ oscillocope



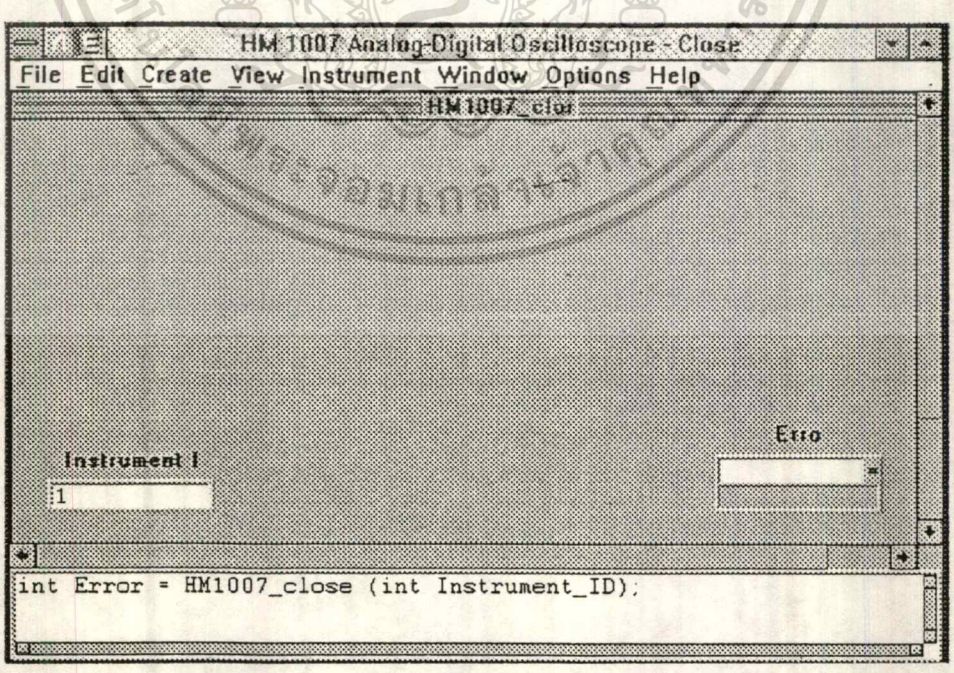
รูปที่ 5.21 Oscilloscope - Active channel

3. **Read waveform** เป็นการอ่านค่าจาก oscilloscope



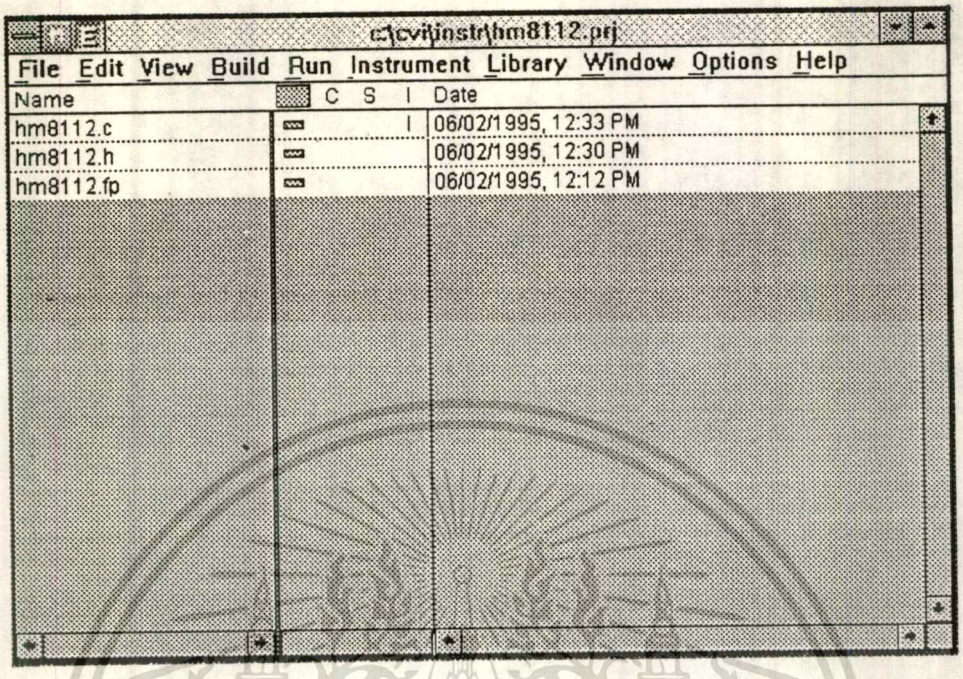
รูปที่ 5.22 Oscilloscope - Read waveform

4. **Close** เป็นการออกจากการควบคุม

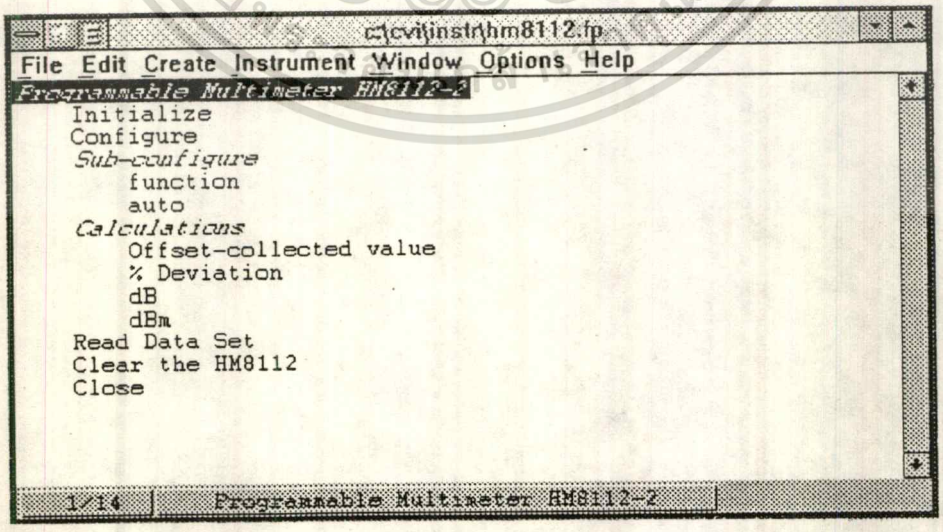


รูปที่ 5.23 Oscilloscpe - Close

Programmable Multimeter HM8112-2



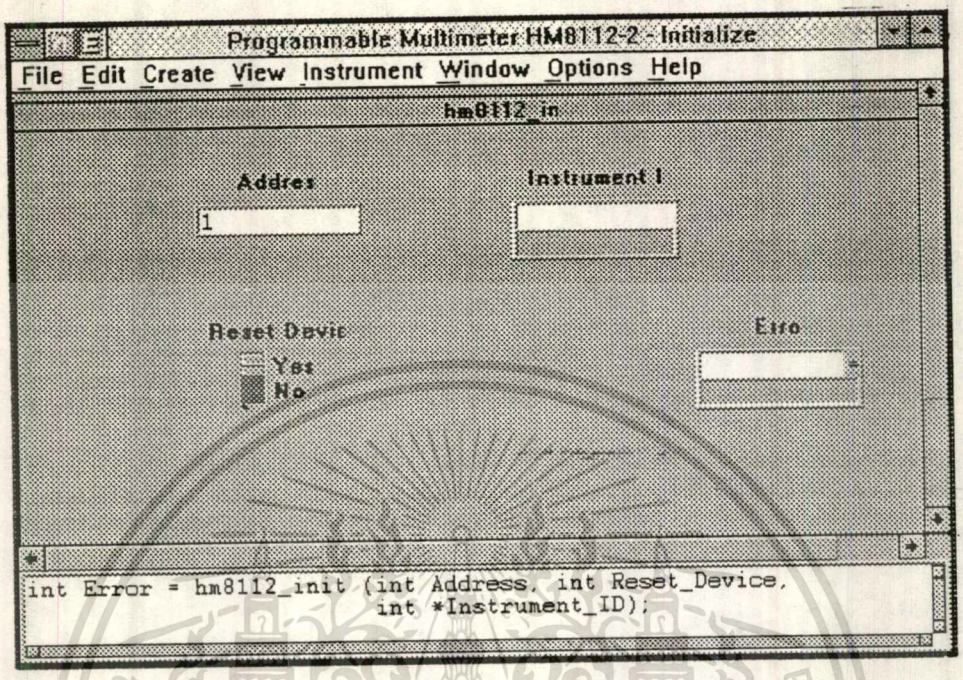
รูปที่ 5.24 แสดง Project file ของ Multimeter



รูปที่ 5.25 แสดง function tree ของ Multimeter

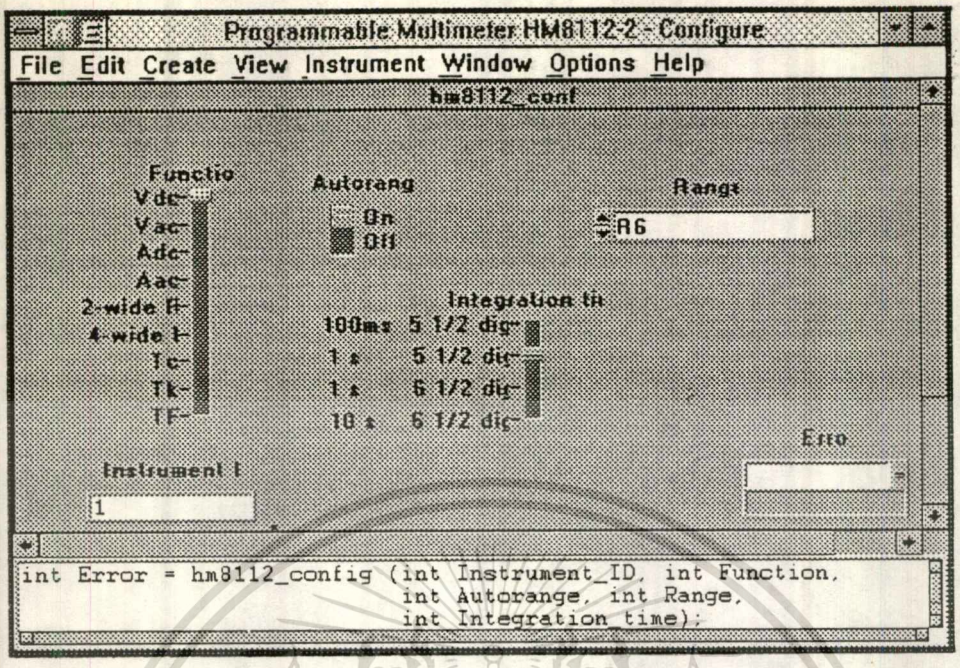
function ต่างๆ ภายใน function tree มีดังนี้

1. Initialize



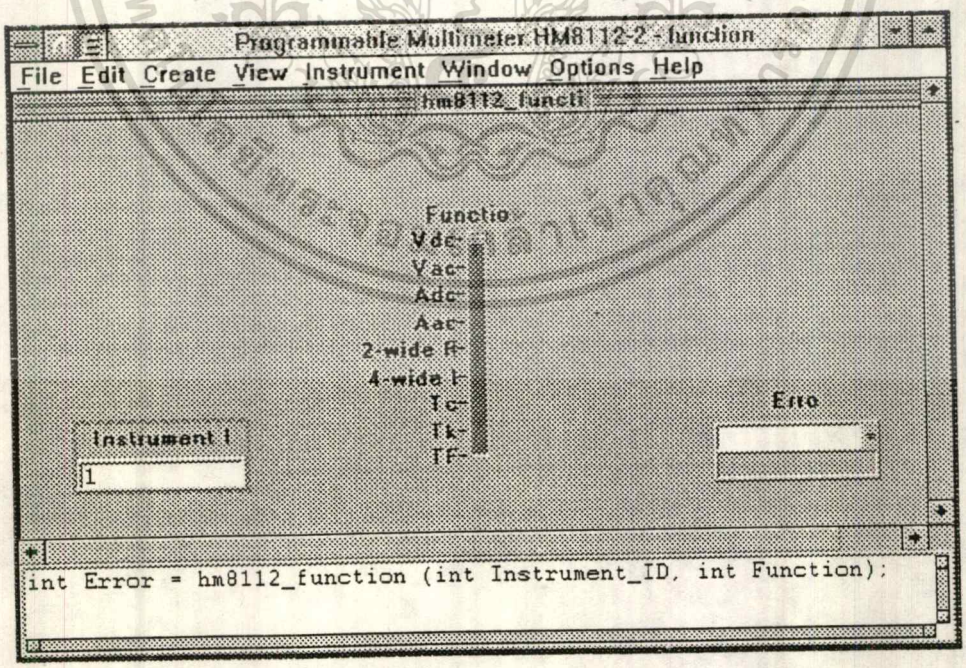
รูปที่ 5.26 Multimeter - Initialize

2. Configure ประกอบด้วยการตั้งค่า function ในการวัดของ meter รวมถึงการหนดขอบเขตการวัด และการตั้งค่า Integration time ของ multimeter

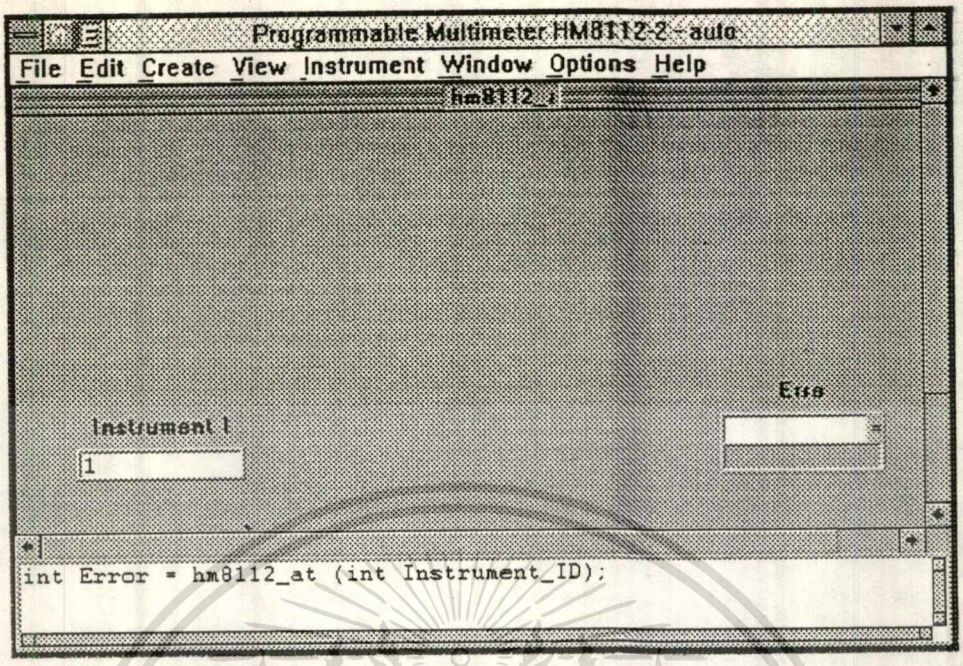


รูปที่ 5.27 Multimeter - Configure

3. Sub-configure เป็นการแยกการตั้งค่า function ในการวัดออกจากการตั้งค่าขอบเขตอัตโนมัติ ออกเป็น 2 functions ย่อย



รูปที่ 5.28 Multimeter - function



รูปที่ 5.29 Multimeter - auto

4. Calculations เป็นการคำนวณค่าต่างๆ ที่อ่านได้จากมิเตอร์ประกอบด้วย ค่า offset, %Deviation, dB และ dBm

5. Read Data Set เป็นการอ่านค่าจาก multimeter

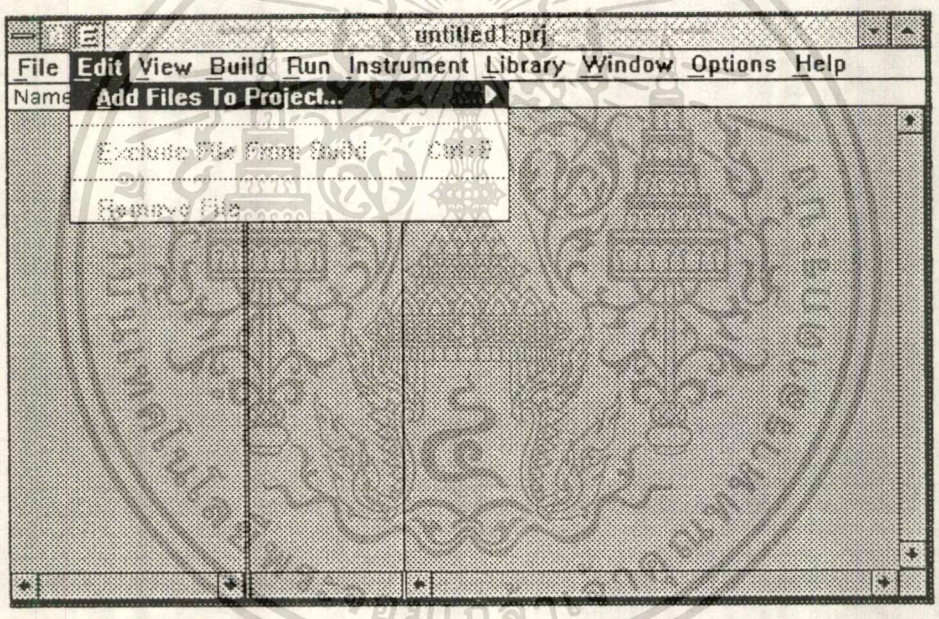
6. Clear the Hm8112

นอกจากการเขียน instrument driver ยังได้ทำตัวอย่างของการควบคุมขึ้นมาด้วย คือเขียนเป็น project file ขึ้นมาซึ่งขั้นตอนในการเขียน project file อย่างคร่าวๆ มีดังนี้

ผลการทดลอง Project file

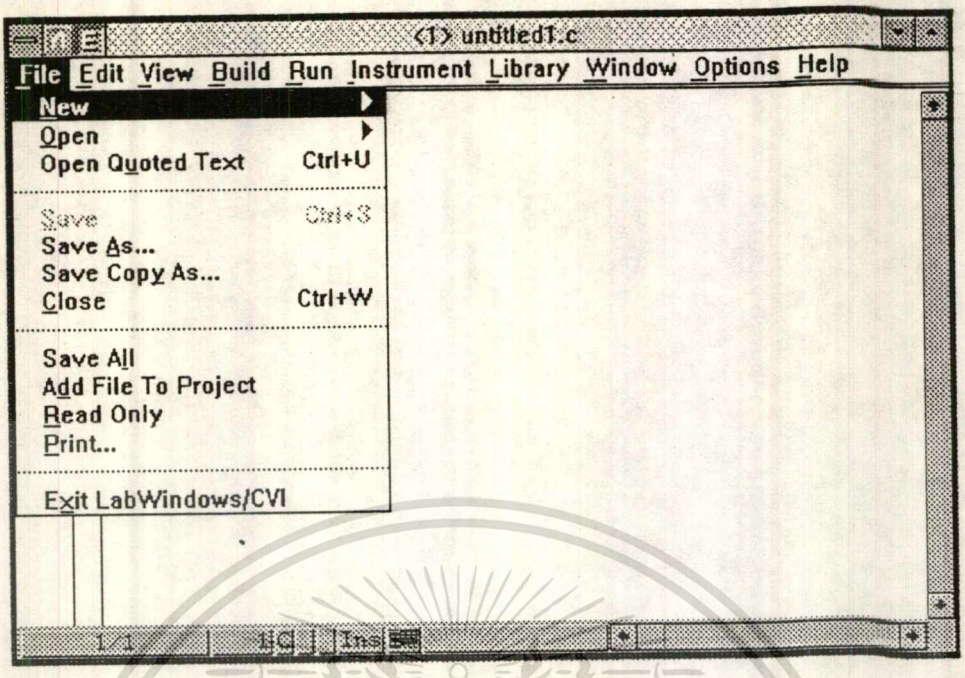
ส่วนสำคัญในการสร้างโปรแกรมใช้งาน (Project file) ประกอบด้วย หน้าต่างหลัก (Project window), หน้าต่างตัวโปรแกรม (Source window) และหน้าต่างที่ใช้เชื่อมต่อกับผู้ใช้ (User Interface Editor window)

ใน Project window เราสามารถรวม window ต่างๆ ที่สร้างขึ้นไว้ที่หน้าต่างเดียวกันเพื่อให้แต่ละ window ทำงานเชื่อมต่อกันได้ ซึ่งวิธีการรวมจะใช้คำสั่ง Add file to Project



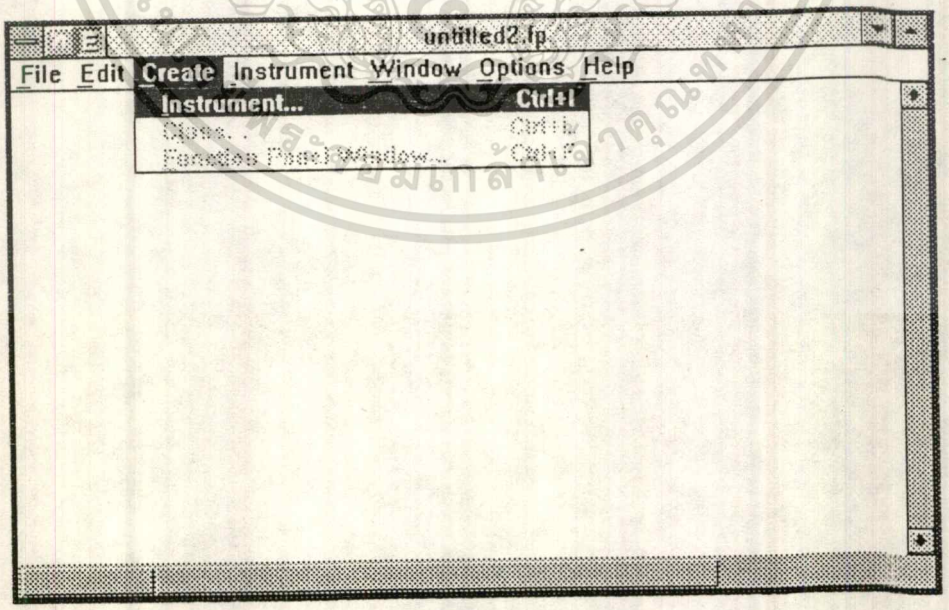
รูปที่ 5.30 Project window

Source window (.C) สามารถเขียนโปรแกรมภาษาซีลงไปที่หน้าต่างนี้ได้ จะเป็นส่วนที่ควบคุมการทำงานของหน้าต่างอื่นๆ บ่งบอกการดำเนินไปของโปรแกรมด้วย



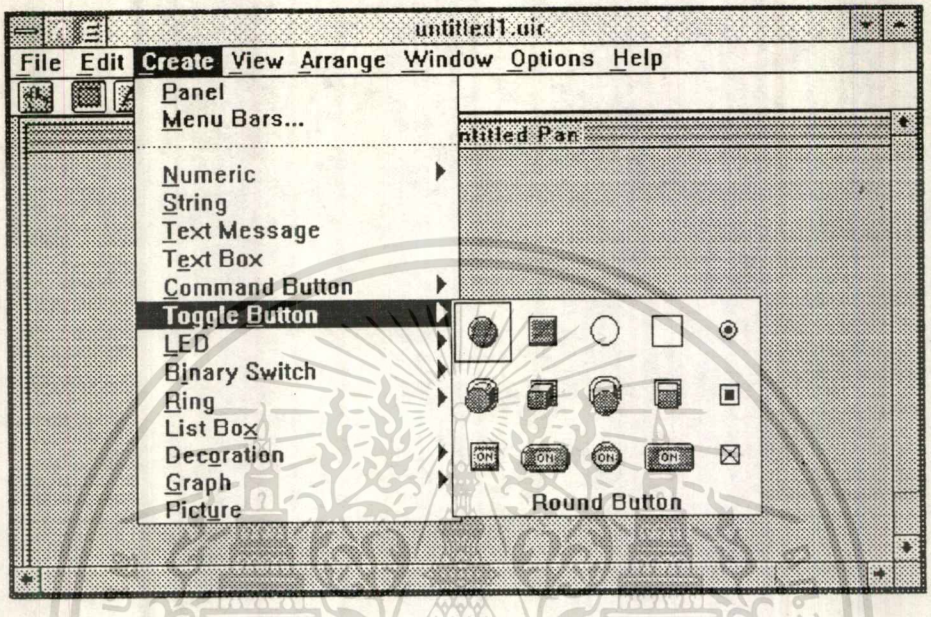
รูปที่ 5.31 Source window

ในกรณีที่มีหน้าต่างของ .fp นั่นคือแสดงว่า โปรแกรมมีการใช้งานผ่าน function tree ของInstrument driver เหมือนเป็นตัวเชื่อมให้ติดต่อกับอุปกรณ์นั้นๆ ได้

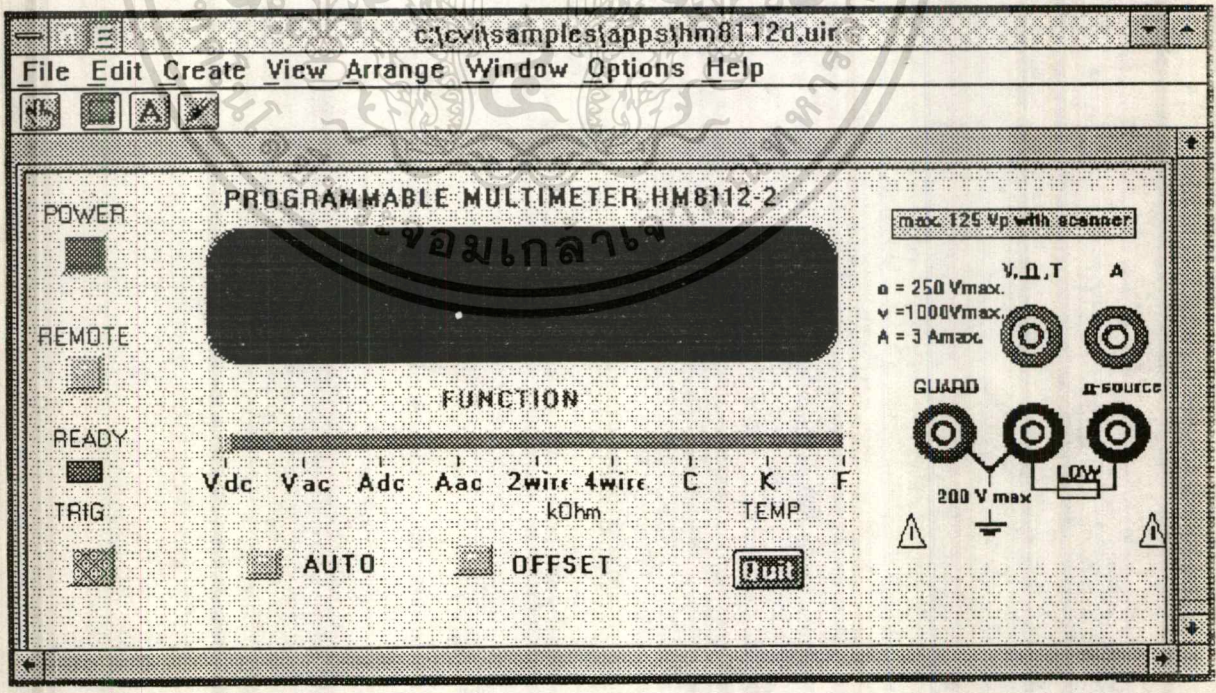


รูปที่ 5.32 Function Panel window

User Interface Editor window เป็นการสร้างส่วนที่ติดต่อกับผู้ใช้ผ่านหน้าจอคอมพิวเตอร์ วิธีการสร้างรูปบนหน้าจอจะใช้คำสั่ง creat ดังรูป



รูปที่ 5.33 User Interface Editor window



รูปที่ 5.34 แสดงตัวอย่างรูปที่สร้างเสร็จแล้ว

```
#include <gpib.h>
#include <formatio.h>
#include "HM8142.h"

static int address[HM8142_MAX_INSTR + 1];
static int bd[HM8142_MAX_INSTR + 1];
static int instr_cnt;
static int HM8142_err;

/* cmd is a buffer for GPIB I/O strings */
static char cmd[50];

/*INSTRUMENT_DEPENDENT COMMAND ARRAYS================================*/
static char* time_code[16];

/*= UTILITY ROUTINES ================================================*/
int HM8142_open_instr (int addr);
int HM8142_close_instr (int instrID);
int HM8142_invalid_short_range (short val, short min, short max,
    int err_code);
int HM8142_invalid_integer_range (int val, int min, int max,
    int err_code);
int HM8142_invalid_longint_range (long val, long min, long max,
    int err_code);
int HM8142_invalid_real_range (double val, double min, double max,
    int err_code);
int HM8142_device_closed (int instrID);
int HM8142_read_data (int instrID, char *buf, long cnt);
int HM8142_write_data (int instrID, char *buf, long cnt);
int HM8142_read_data_file (int instrID, char *filename);
int HM8142_write_data_file (int instrID, char *filename);
int HM8142_poll (int instrID, char *response);
int HM8142_set_timeout (int instrID, int tmo_code);
void HM8142_setup_arrays (void);

/*====================================================================*/
/* Function: Initialize                                               */
/* Purpose:  This function opens the instrument, queries the          */
/*           instrumentforits ID,and initializes the instrument       */
/*           to a known state.                                        */
/*====================================================================*/
int HM8142_init (int addr, int id_query, int rest, int *instrID)

{
    int  ID;

    if (HM8142_invalid_integer_range (addr, 0, 30, -1) != 0)
        return HM8142_err;
    if (HM8142_invalid_integer_range (id_query, 0, 1, -2) != 0)
        return HM8142_err;
    if (HM8142_invalid_integer_range (rest, 0, 1, -3) != 0)
        return HM8142_err;

    ID = HM8142_open_instr (addr);
    if (ID <= 0)
        return HM8142_err;

    if (id_query) {
        if (HM8142_write_data (ID, "ID?", 3) != 0) {
            HM8142_close_instr (ID);
            return HM8142_err;
        }
```



```
    switch (local){
    case 0:
```

```
      Fmt (cmd,"%s<LKO\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      break;
case 1:
      Fmt (cmd,"%s<LK1\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      break;
    }
   }
 else
 {
  Fmt (cmd,"%s<RMO\n");

  if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
      return HM8142_err;
   }
if (remote==1)
 {
switch (mixed_mode){
case 0:
      Fmt (cmd,"%s<MXO\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
      break;
case 1:
      Fmt (cmd,"%s<MX1\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
      break;
    }
   }
 else
   {
    Fmt (cmd,"%s<RMO\n");
    if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
        return HM8142_err;
    }
 if (remote==1)
   {
   switch (output){
   case 0:
      Fmt (cmd,"%s<OP0\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      break;
   case 1:
       Fmt (cmd,"%s<OP1\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
           return HM8142_err;
       break;
      }
    }
   else
    {
     Fmt (cmd,"%s<RMO\n");
     if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
     }
 return HM8142_err;
 }
```

```
/*============================================================================*/
/*Function:Set_voltage                                                        */
/*Purpose: This function select voltage 1,voltage 2 or track and set*/
/*           voltage.                                                         */
/*============================================================================*/
int HM8142_set_voltage (int instrID,int select,double volt_value)
{
  if (HM8142_invalid_integer_range(instrID,1,HM8142_MAX_INSTR,-1)!=0)
      return HM8142_err;
  if (HM8142_invalid_integer_range (select,0,2,-2)!=0)
      return HM8142_err;
 switch (select){
  case 0:
       if (HM8142_invalitch (select){
  case 0:
       if (HM8142_invalid_real_range (volt_value,0.00,30.00,-3)!=0)
           return HM8142_err;
       Fmt (cmd,"%s<SU1 %f[p2]\n",volt_value);
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
           return HM8142_err;
       break;
  case 1:
       if (HM8142_invalid_real_range (volt_value,0.00,30.00,-3)!=0)
           return HM8142_err;



        Fmt (cmd,"%s<SU2 %f[p2]\n",volt_value);
        if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
            return HM8142_err;
        break;
   case 2:
        if (HM8142_invalid_real_range (volt_value,0.00,30.00,-3)!=0)
            return HM8142_err;
        Fmt (cmd,"%s<TRU %f[p2]\n",volt_value);
        if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
            return HM8142_err;
        break;
     }
   return HM8142_err;
  }
 /*============================================================================
 /*Function:Set_current
 /*Purpose:This function set the current and select current1,current2
 /*          or track.
 /*============================================================================
 int HM8142_set_current (int instrID,int select,double current_value)
 {
   if (HM8142_invalid_integer_range(instrID,1,HM8142_MAX_INSTR,-1)!=0
       return HM8142_err;
   if (HM8142_invalid_integer_range (select,0,2,-2)!=0)
       return HM8142_err;
   switch (select){
    case 0:
         if (HM8142_invalid_real_range(current_value,0.00,1.00,-3)!=0
            return HM8142_err;
         Fmt (cmd,"%s<SI1 %f[p2]\n",current_value);
         if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
             return HM8142_err;
         break;
    case 1:
         if (HM8142_invalid_real_range(current_value,0.00,1.00,-3)!=0
```

```
            return HM8142_err;
         Fmt (cmd,"%s<SI2 %f[p2]\n",current_value);
         if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
            return HM8142_err;
         break;
    case 2:
         if(HM8142_invalid_real_range(current_value,0.00,1.00,-3)!=0)
             return HM8142_err;
         Fmt (cmd,"%s<TRI %f[p2]\n",current_value);
         if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
             return HM8142_err;
         break;
     }
  return HM8142_err;
 }
/*============================================================================*
/*Function:Recall                                                             *
/*Purpose:This function recall voltage and current.                           *
/*============================================================================*
int HM8142_recall_mode (int instrID,int recall,double *recall_value)
{
  if (HM8142_invalid_integer_range(instrID,1,HM8142_MAX_INSTR,-1)!=0)
     return HM8142_err;
  if (HM8142_invalid_integer_range (recall,0,3,-2)!=0)
     return HM8142_err;
 switch (recall){
 case 0:
      Fmt (cmd,"%s<RU1\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      if (HM8142_read_data (instrID,cmd,30)!=0)
         return HM8142_err;
      if (Scan(cmd,"%s[i3]>%f[p3]",recall_value)!=1)
         {

          HM8142_err=236;
          return HM8142_err;
          }
       break;
 case 1:
      Fmt (cmd,"%s<RU2\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      if (HM8142_read_data (instrID,cmd,50)!=0)
         return HM8142_err;
      if (Scan(cmd,"%s[i3]>%f[p3]",recall_value)!=1)
         {
          HM8142_err=236;
          return HM8142_err;
          }
      break;
 case 2:
      Fmt (cmd,"%s<RI1\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      if (HM8142_read_data (instrID,cmd,30)!=0)
         return HM8142_err;
      if (Scan(cmd,"%s[i3]>%f[p3]",recall_value)!=1)
         {
          HM8142_err=236;
          return HM8142_err;
          }
```

```
      break;
 case 3:
      Fmt (cmd,"%s<RI2\n");
      if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
         return HM8142_err;
      if (HM8142_read_data (instrID,cmd,30)!=0)
         return HM8142_err;
      if (Scan(cmd,"%s[i3]>%f[p3]",recall_value)!=1)
        {
         HM8142_err=236;
         return HM8142_err;
        }
      break;
   }
 return HM8142_err;

}
/*============================================================================*
/*Function:Measure                                                            *
/*Purpose:This function measure voltage and current.                          *
/*============================================================================*
int HM8142_measure_mode(int instrID,int measure,double *measure_value
{
  if (HM8142_invalid_integer_range (instrID,1,HM8142_MAX_INSTR,-1)!=0
     return HM8142_err;
  if (HM8142_invalid_integer_range (measure,0,3,-2)!=0)
     return HM8142_err;
 switch(measure){
 case 0:
       Fmt(cmd,"%s<MU1\n");
       if (HM8142_write_data(instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
       if (HM8142_read_data(instrID,cmd,30)!=0)
          return HM8142_err;
       if (Scan(cmd,"%s[i3]>%f[p3]",measure_value)!=1)
          {
           HM8142_err=236;
           return HM8142_err;
           }
       break;
 case 1:
       Fmt(cmd,"%s<MU2\n");

       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
           return HM8142_err;
       if (HM8142_read_data (instrID,cmd,30)!=0)
           return HM8142_err;
       if (Scan(cmd,"%s[i3]>%f[p3]",measure_value)!=1)
          {
           HM8142_err=236;
           return HM8142_err;
           }
       break;
 case 2:
       Fmt(cmd,"%s<MI1\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
           return HM8142_err;
       if (HM8142_read_data (instrID,cmd,30)!=0)
           return HM8142_err;
       if (Scan(cmd,"%s[i3]>%f[p3]",measure_value)!=1)
          {
```

```
            HM8142_err=236;
            return HM8142_err;
            }
        break;
case 3:
        Fmt(cmd,"%s<MI2\n");
        if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
           return HM8142_err;
        if (HM8142_read_data (instrID,cmd,30)!=0)
           return HM8142_err;
        if (Scan(cmd,"%s[i3]>%f[p3]",measure_value)!=1)
           {
            HM8142_err=236;
            return HM8142_err;
           }
        break;
      }
   return HM8142_err;
}
/*=============================================================================*
/*Function:Abitrary mode                                                       *
/*Purpose:use for generation of virtually any desired waveforms.               *
/*=============================================================================*
int HM8142_arbitrary_mode (int instrID,int time[512],
    double voltage[512],int number,int reapted_time,int mode)
{
  int i,offset,leng;
  char data_string[4096];
  char time_string[3];
  char volt_string[7];


  if (HM8142_invalid_integer_range(instrID,1,HM8142_MAX_INSTR,-1)!=0)
     return HM8142_err;
  if (HM8142_invalid_integer_range (number,1,511,-4)!=0)
     return HM8142_err;
  if (HM8142_invalid_integer_range (reapted_time,1,255,-5)!=0)
     return HM8142_err;
  if (HM8142_invalid_integer_range (mode,0,2,-6)!=0)
     return HM8142_err;
  if (number == 512)
      number--;

  for (i=0;i<number;i++)
   {
     if (HM8142_invalid_integer_range (time[i],0,15,-2)!=0)
        return HM8142_err;
     if (HM8142_invalid_real_range (voltage[i],0.00,30.00,-3)!=0)
        return HM8142_err;
    }
  offset = 0;

  CopyString (data_string,offset,"ABT ",0,4);
  offset +=4;

  for (i=0;i<number;i++)
   {
     Fmt (time_string,"%s<%s",time_code[time[i]]);
     leng = StringLength (time_string);
     CopyString (data_string,offset,time_string,0,leng);
     offset +=leng;
```

```
    Fmt (volt_string,"%s<%f[p2w5] ",voltage[i]);
    leng = StringLength (volt_string);
    CopyString (data_string,offset,volt_string,0,leng);
    offset +=leng;
   }

 Fmt (cmd,"%s<N%i\n",reapted_time);
 leng = StringLength (cmd);
 CopyString (data_string,offset,cmd,0,leng);
 leng = StringLength (data_string);
 if (HM8142_write_data (instrID,data_string,leng)!=0)
    return HM8142_err;

switch(mode){
 case 0:
       Fmt (cmd,"%s<RUN\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
       break;
 case 1:
       Fmt (cmd,"%s<STP\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
       break;
 case 2:
       Fmt (cmd,"%s<ABX\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
       break;
   }
  return HM8142_err;
 }
/*============================================================
int HM8142_arbitrary_test (int instrID,unsigned char arb_text,
    int select)
{
  if (HM8142_invalid_integer_range (instrID,1,HM8142_MAX_INSTR,-1)
      return HM8142_err;
  if (HM8142_invalid_integer_range (select,0,1,-3)!=0)
      return HM8142_err;

  Fmt (cmd,"%s<ABT %s\n",arb_text);
  if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
     return HM8142_err;

  switch(select){
  case 0:
       Fmt (cmd,"%s<RUN\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
       break;
  case 1:
       Fmt (cmd,"%s<STP\n");
       if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
          return HM8142_err;
       break;
   }
  return HM8142_err;
 }
```

```
/*=============================================================================*
/*Function:Check_status                                                        *
/*Purpose:This function check version,status,identifier.                       *
/*=============================================================================
==================*/
int HM8142_check_status (int instrID,int get,char *reply_value)
{
  if (HM8142_invalid_integer_range(instrID,1,HM8142_MAX_INSTR,-1)!=0)
      return HM8142_err;
  if (HM8142_invalid_integer_range (get,0,2,-2)!=0)
     return HM8142_err;

  switch(get){
  case 0:
        Fmt(cmd,"%s<VER\n");
        if (HM8142_write_data (instrID,cmd,NumFmtdBytes())!=0)
           return HM8142_err;
        if (HM8142_read_data (instrID,cmd,30)!=0)
           return HM8142_err;
        if (Scan(cmd,"%s[i3]>%f[p3]",reply_value)!=1)
           {
            HM8142_err=236;
            return HM8142_err;
            }
        break;
    }
   return HM8142_err;
  }

/*----------- INSERT INSTRUMENT-DEPENDENT ROUTINES HERE ------------*
/*=============================================================================*
/* Function: Close                                                             *
/* Purpose:  This function closes the instrument.                              *
/*=============================================================================*
int HM8142_close (int instrID)
{
   if(HM8142_invalid_integer_range(instrID,1,HM8142_MAX_INSTR,-1)!= 0
        return HM8142_err;

    if (HM8142_device_closed (instrID))
        return HM8142_err;

    HM8142_close_instr (instrID);
    return HM8142_err;
}



/*= UTILITY ROUTINES ==========================================*

/*=============================================================================*
/* Function: Open Instrument                                                   *
/* Purpose:  This function locates and initializes an entry in the             *
/*           Instrument Table and the GPIB device table for the                *
/*           instrument.  If successful, the instrument ID is                  *
/*           returned,                                                         *
/*           else a -1 is returned.  The size of the Instrument                *
/*           Table can                                                         *
/*           be changed in the include file by altering the constant           *
/*           HM8142_MAX_INSTR.                                                 *
```

```
/*=============================================================================*/
int HM8142_open_instr (int addr)
{
    int i, instrID;

    instrID = 0;

/* Check to see if the instrument is already in the Instrument Table*


    for (i=1; i<= HM8142_MAX_INSTR; i++)
        if (address[i] == addr) {
            instrID = i;
            i = HM8142_MAX_INSTR;
         }

/* If it is not in the instrument table, open an entry for the      *
/* instrument.                                                      *

    if (instrID <= 0)
        for (i=1; i<= HM8142_MAX_INSTR; i++)
            if (address[i] == 0) {
                instrID = i;
                i = HM8142_MAX_INSTR;
             }

/* If an entry could not be opened in the Instrument Table, return  *
/* an error.                                                        *

    if (instrID <= 0) {
        HM8142_err = 220;
        return -1;
     }

/*  If the device has not been opened in the GPIB device table      *
/*   (bd[ID] = 0),then open it.                                     *
            */

    if (bd[instrID] <= 0)    {
        if (instr_cnt <= 0)
            CloseInstrDevs("HM8142");
        bd[instrID] = OpenDev ("", "HM8142");
        if (bd[instrID] <= 0)    {
            HM8142_err = 220;
            return -1;
        }
        instr_cnt += 1;
        address[instrID] = addr;
     }

/*  Change the primary address of the device    */

    if (ibpad (bd[instrID], addr) < 0)   {
        HM8142_err = 233;
        return -1;
    }
    return instrID;
}
/*=============================================================================*/
```

```
/* Function: Close Instrument                                          *
/* Purpose:  This fnction closes the instrument by removing it from *
/*           the GPIB device table and setting the address and the  *
/*           bd to zero in the Instrument Table.  The return value  *
/*           is equal to the global error variable.                 *

/*=====================================================================*
int HM8142_close_instr (int instrID)
{
    if (bd[instrID] != 0) {
        CloseDev (bd[instrID]);
        bd[instrID] = 0;
        address[instrID] = 0;
        instr_cnt -= 1;
     }
    else
        HM8142_err = 221;

    return HM8142_err;
 }
/*=====================================================================*
/* Function: Invalid Short Range                                       *
/* Purpose:  This function checks a short to see if it lies between *
/*           a minimum and maximum value.  If the value is out of   *
/*           range, set the global error variable to the value      *
/*           err_code.  If the value is OK, error = 0.              *
/*=====================================================================*
int HM8142_invalid_short_range (short val, short min, short max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    HM8142_err = err_code;
    return -1;
  }
  return 0;
}
/*=====================================================================*
/* Function: Invalid Integer Range                                     *
/* Purpose:  This function checks an integer to see if it lies      *
/*           between a minimum and maximum value.  If the value is  *
/*           out of range, set                                       *
/*           the global error variable to the value err_code.If the *
/*           value is OK, error = 0.                                 *
/*=====================================================================*
int HM8142_invalid_integer_range (int val, int min, int max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    HM8142_err = err_code;
    return -1;
  }
  return 0;
}
/*=====================================================================*
/* Function: Invalid Long Integer Range                          =======*
/* Function: Invalid Long Integer Range                                *
/* Purpose:  This function checks a long integer to see if it lies  *
/*           between a minimum and maximum value.  If the value is  *
/*           out of range,set the global error varisble to the value*
/*           err_code.  If the value is OK, error = 0.  The return  *
```

```
/*          value is equal to the global error value.                  *

/*=============================================================*
int HM8142_invalid_longint_range (long val, long min, long max,
    int err_code)
{
    if (val < min || val > max) {
        HM8142_err = err_code;
        return -1;
    }
    return 0;
}
/*=============================================================*
/* Function: Invalid Real Range                                *
/* Purpose:  This function checks a real number to see if it lies  *
/*           between a minimum and maximum value.  If the value is  *
/*           out of range,set the global error variable to the value*
/*           err_code.  If the value is OK, error = 0.          *
/*=============================================================*
int HM8142_invalid_real_range (double val, double min, double max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    HM8142_err = err_code;
    return -1;
  }
  return 0;
}

/*=============================================================*
/* Function: Device Closed                                     *
/* Purpose:  This function checks to see if the module has been    *
/*           initialized.  If the device has not been opened, a 1 is*
/*           returned, 0 otherwise.                            *
/*=============================================================*
int HM8142_device_closed (int instrID)
{
    if (bd[instrID] <= 0) {
        HM8142_err = 232;
        return -1;
    }
    return 0;
}
/*=============================================================*
/* Function: Read Data                                         *
/* Purpose:  This function reads a buffer of data fromthe instrument*
/*           .The return value is equal to the global error variable*
/*=============================================================*
int HM8142_read_data (int instrID, char *buf, long cnt)
{
    if (ibrd(bd[instrID], buf, cnt) <= 0)
        HM8142_err = 231;
    else
        HM8142_err = 0;

    return HM8142_err;
}
/*=============================================================*
/* Function: Write Data                                        *
/* Purpose:  This function writes a buffer of data to the instrument*
/*           .The return value is equal to the global error variable*
/*=============================================================*
```

```
int HM8142_write_data (int instrID, char *buf, long cnt)
{
    if (ibwrt(bd[instrID], buf, cnt) <= 0)
        HM8142_err = 230;
    else
        HM8142_err = 0;

    return HM8142_err;
}
/*=============================================================*
/* Function: Read Data File                                    *
/* Purpose:  This function reads a buffer of data from theinstrument*
/*           and stores it to the file specified by "filename".  *
/*           Filename must either be a string, such as "C:\lw\instr\*
/*           ile" or a pointe to such a string.  The return value is*
/*           equal to the global error variable.                 *
/*=============================================================*
int HM8142_read_data_file (int instrID, char *filename)
{
    if (ibrdf (bd[instrID], filename) <= 0)
        HM8142_err = 229;
    else
        HM8142_err = 0;

    return HM8142_err;
}
/*=============================================================*
/* Function: Write Data File                                   *
/* Purpose:  This function writes a buffer of data from the file    *
/*           specified by "filename" to the instrument. Filename    *
/*           must either be a string, such as "C:\lw\instr\file" or *
/*           a pointer to such a string.  The return value is equal *
/*           to the global error  variah a string.  The return value
/*           to the global error  variable.                      *
/*=============================================================*
int HM8142_write_data_file (int instrID, char *filename)
{

    if (ibwrtf (bd[instrID], filename) <= 0)
        HM8142_err = 228;
    else
        HM8142_err = 0;

    return HM8142_err;
}
/*=============================================================*
/* Function: Serial Poll                                       *
/* Purpose:  This function performs a serial poll on the instrument.*
/*           The status byte of the instrument is placed in the    *
/*           response variable. The return value is equal to the   *
/*           global error variable.                              *
/*=============================================================*
int HM8142_poll (int instrID, char *response)
{
    if (ibrsp (bd[instrID], response) <= 0)
        HM8142_err = 226;
    else
        HM8142_err = 0;

    return HM8142_err;
}
```

```
/*=============================================================================
/* Function: Set Timeout                                                      
/* Purpose:  This function changes or disables the timeout of the
/*           device.Refer to the LabWindows Standard Libraries
/*           Reference Manual for timeout codes.
/*=============================================================================
int HM8142_set_timeout (int instrID, int tmo_code)
{
    if (ibtmo (bd[instrID], tmo_code) <= 0)
        HM8142_err = 239;
    else
        HM8142_err = 0;

    return HM8142_err;
}
/*=============================================================================
/* Function: Setup Arrays
/* Purpose:  This function is called by the init routine to
/*           initialize global arrays.
/*           This routine should be modified for each instrument to
/*           include instrument-dependent commmand arrays.
/*=============================================================================
void HM8142_setup_arrays ()
{

 time_code[0] = "0";    /* 100 us    */
 time_code[1] = "1";    /*   1 ms    */
 time_code[2] = "2";    /*   2 ms    */
 time_code[3] = "3";    /*   5 ms    */
 time_code[4] = "4";    /*  10 ms    */
 time_code[5] = "5";    /*  20 ms    */
 time_code[6] = "6";    /*  50 ms    */
 time_code[7] = "7";    /* 100 ms    */
 time_code[8] = "8";    /* 200 ms    */
 time_code[9] = "9";    /* 500 ms    */
 time_code[10]= "A";    /*   1 s     */
 time_code[11]= "B";    /*   2 s     */
 time_code[12]= "C";    /*   5 s     */
 time_code[13]= "D";    /*  10 s     */
 time_code[14]= "E";    /*  20 s     */
 time_code[15]= "F";    /*  50 s     */

}
/*= THE END ==================================================================
```

```
#include <gpib.h>
#include <formatio.h>
#include <math.h>
#include "hm8130.h"

static int address[hm8130_MAX_INSTR + 1];
static int bd[hm8130_MAX_INSTR + 1];
static int instr_cnt;
static int hm8130_err;

/*= INTERNAL DATA ==========================================================
/* cmd is a buffer for GPIB I/O strings */
static char cmd[70];

/*= UTILITY ROUTINES =======================================================
int hm8130_open_instr (int addr);
int hm8130_close_instr (int instrID);
int hm8130_invalid_integer_range (int val, int min, int max, int
    err_code);
int hm8130_invalid_real_range (double val, double min, double max,
    int err_code);
int hm8130_device_closed (int instrID);
int hm8130_read_data (int instrID, char *buf, long cnt);
int hm8130_write_data (int instrID, char *buf, long cnt);

/*==========================================================================
/* Function: Initialize
/* Purpose:  This function opens the instrument and initializes the
/*           instrument to a known state.
/*==========================================================================
int hm8130_init (int addr, int rest, int *instrID)
{
    int  ID;

    if (hm8130_invalid_integer_range (addr, 0, 30, -1) != 0)
        return hm8130_err;
    if (hm8130_invalid_integer_range (rest, 0, 1, -2) != 0)
        return hm8130_err;

    ID = hm8130_open_instr (addr);
    if (ID <= 0)
        return hm8130_err;

    if (rest) {
        if (hm8130_write_data (ID, "INIT ", 5) != 0) {
            hm8130_close_instr (ID);
            return hm8130_err;
        }
    }
    if (hm8130_write_data (ID, "PATH OFF ", 9) != 0) {
        hm8130_close_instr (ID);
        return hm8130_err;
    }


    *instrID = ID;
    return hm8130_err;
}
/*==========================================================================
/* Function: Continuous Mode
```

```
/* Purpose:  This function configures the basic setup of the HM8130,*/
/*           including mode waveformtype, frequency,amplitude,offset*/
/*                                                                  */
/*           and output.                                            */
/*=================================================================*/
int hm8130_config (instrID, wvf, freq, ampl,offset,output)
int instrID;
int wvf;
double freq;
double ampl;

double offset;
int output;
{
  if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!=0)
       return hm8130_err;
    if (hm8130_invalid_integer_range (wvf, 0, 5, -2) != 0)
       return hm8130_err;
    if (hm8130_invalid_real_range (freq, 0.001, 1.0E+7, -3) != 0)
       return hm8130_err;
    if (hm8130_invalid_real_range (ampl, 0, 20, -4) != 0)
       return hm8130_err;
    if (hm8130_invalid_real_range (offset, -7.5, 7.5, -5) != 0)
       return hm8130_err;
    if (hm8130_invalid_integer_range (output, 0, 1, -6) != 0)
       return hm8130_err;
    if (hm8130_device_closed (instrID) != 0)
       return hm8130_err;

 /* Check that the amplitude and offset are within the correct range *
     if (ampl + offset > 20.0) {
         hm8130_err = 309;
         return hm8130_err;
      }
    switch(wvf)
    {   case 0:
           Fmt (cmd, "%s<RMP ");
           break;
        case 1:
           Fmt (cmd, "%s<TRI ");
           break;
        case 2:
           Fmt (cmd, "%s<SIN ");
           break;
        case 3:
           Fmt (cmd, "%s<SQR ");
           break;
        case 4:
           Fmt (cmd, "%s<PLS ");
           break;
        case 5:
           Fmt (cmd, "%s<ARB ");
           break;
      }
      Fmt (cmd, "%s[a]<CTM DFR FRQ:%f DOF OFS:%f DAM AMP:%f OT%d ",
                        freq,offset,ampl,output);
      if (hm8130_write_data (instrID, cmd, StringLength (cmd)) != 0)
          return hm8130_err;
      return hm8130_err;
  }
  /*=================================================================*/
  /* Function: Trigger Mode
```

```
/* Purpose:  This function sets frequency in trigger mode.          *
/*==================================================================*
int hm8130_trig (int instrID, double freq)
{
  if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!= 0
       return hm8130_err;
    if (hm8130_invalid_real_range (freq, 0.001, 500000, -2) != 0)
       return hm8130_err;
    if (hm8130_device_closed (instrID) != 0)
      return hm8130_err;
    Fmt (cmd, "%s<TRM DFR FRQ:%f ",freq);
    if (hm8130_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
       return hm8130_err;
    return hm8130_err;
}
/*==================================================================*
/* Function: Gated Mode                                             *
/* Purpose:  This function sets frequency in gated mode.            *

/*==================================================================*
int hm8130_gate (int instrID, double freq)
{
   if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!=0
       return hm8130_err;
    if (hm8130_invalid_real_range (freq, 0.001, 500000, -2) != 0)
       return hm8130_err;
    if (hm8130_device_closed (instrID) != 0)
      return hm8130_err;
    Fmt (cmd, "%s<GTM DFR FRQ:%f ",freq);
    if (hm8130_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
       return hm8130_err;
    return hm8130_err;
}
/*==================================================================*
/* Function: Sweep Mode                                             *
/* Purpose:  This function sets parameter in sweep mode.            *
/*==================================================================*
int hm8130_sweep (int instrID, double time, double str, double stp)
{
  if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!=0)
       return hm8130_err;
    if (hm8130_invalid_real_range (time, 0.02, 100, -2) != 0)
d_real_range(thm8130.02r;100, -2) != 0)
       return hm8130_err;
    if (hm8130_invalid_real_range (str, 0.01, 1.0E+7, -3) != 0)
       return hm8130_err;
    if (hm8130_invalid_real_range (stp, 0.01, 1.0E+7, -4) != 0)
       return hm8130_err;
    if (hm8130_device_closed (instrID) != 0)
       return hm8130_err;
    Fmt (cmd, "%s<SW1 DSW SWT:%f DST STT:%f DSP STP:%f ", time, str,
       stp);
    if (hm8130_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
       return hm8130_err;
    return hm8130_err;
}
/*==================================================================*
/* Function: Trigger Sweep via GPIB                                 *
/* Purpose:  This function triggers a signal period or a complete   *
/*           sweep.                                                 *
/*==================================================================*
```

```
int hm8130_trigsw (int instrID)
{
  if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!=0)
        return hm8130_err;
    if (hm8130_device_closed (instrID) != 0)
        return hm8130_err;
    Fmt (cmd, "%s<TRG ");
    if (hm8130_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
        return hm8130_err;
    return hm8130_err;
}
/*====================================================================*
/* Function: Reset                                                    *
/* Purpose:  This function resets the instrument.                     *
/*====================================================================*
int hm8130_reset (int instrID)
{
  if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!=0)
        return hm8130_err;

    if (hm8130_device_closed (instrID) != 0)
        return hm8130_err;
    Fmt (cmd, "%s<CLR ");
    if (hm8130_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
        return hm8130_err;
    return hm8130_err;
}
/*====================================================================*

/* Function: Close                                                    *
/* Purpose:  This function closes the instrument.                     *
/*====================================================================*
int hm8130_close (int instrID)
{
   if (hm8130_invalid_integer_range(instrID,1,hm8130_MAX_INSTR,-1)!=0
        return hm8130_err;

    if (hm8130_device_closed (instrID))
        return hm8130_err;

    hm8130_close_instr (instrID);
   return hm8130_err;
}
/*= UTILITY ROUTINES ================================================

int hm8130_open_instr (int addr)
{
    int i, instrID;

    instrID = 0;

    for (i=1; i<= hm8130_MAX_INSTR; i++)
        if (address[i] == addr) {
            instrID = i;
            i = hm8130_MAX_INSTR;
         }

    if (instrID <= 0)
        for (i=1; i<= hm8130_MAX_INSTR; i++)
            if (address[i] == 0) {
                instrID = i;
```

```
                i = hm8130_MAX_INSTR;
            }
    if (instrID <= 0) {
        hm8130_err = 220;
        return -1;
     }

    if (bd[instrID] <= 0)    {
        if (instr_cnt <= 0)
            CloseInstrDevs("hm8130");
        bd[instrID] = OpenDev ("", "hm8130");
        if (bd[instrID] <= 0)    {
            hm8130_err = 220;
            return -1;
        }
        instr_cnt += 1;
        address[instrID] = addr;
     }

    if (ibpad (bd[instrID], addr) < 0)   {
        hm8130_err = 233;
        return -1;
    }

    return instrID;
}
/*=============================================================================
/* Function: Close Instrument
/*=============================================================================
int hm8130_close_instr (int instrID)
{
    if (bd[instrID] != 0) {
        CloseDev (bd[instrID]);
        bd[instrID] = 0;
        address[instrID] = 0;
        instr_cnt -= 1;
     }

    else
        hm8130_err = 221;

    return hm8130_err;
 }

/*=============================================================================
/* Function: Invalid Integer Range
/*=============================================================================
int hm8130_invalid_integer_range (int val, int min, int max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    hm8130_err = err_code;
    return -1;
  }
  return 0;
}

/*=============================================================================
/* Function: Invalid Real Range
/*=============================================================================
int hm8130_invalid_real_range (double val, double min, double max,
    int err_code)
```

```
{
  if ((val < min) || (val > max)) {
    hm8130_err = err_code;
    return -1;
  }
  return 0;
}
/*=========================================================================
======================*/
/* Function: Device Closed
/*=========================================================================
int hm8130_device_closed (int instrID)
{
    if (bd[instrID] <= 0)  {
        hm8130_err = 232;
        return -1;
    }
    return 0;
}
/*=========================================================================
/* Function: Read Data
/*=========================================================================
int hm8130_read_data (int instrID, char *buf, long cnt)
{
    if (ibrd(bd[instrID], buf, cnt) <= 0)
        hm8130_err = 231;
    else
        hm8130_err = 0;

    return hm8130_err;
}
/*=========================================================================
/* Function: Write Data
/*=========================================================================
int hm8130_write_data (int instrID, char *buf, long cnt)
{
    if (ibwrt(bd[instrID], buf, cnt) <= 0)
        hm8130_err = 230;
    else
        hm8130_err = 0;

    return hm8130_err;
}
/*
/*= THE END ===============================================================
```

```
#include <math.h>
#include <gpib.h>
#include <formatio.h>
#include <analysis.h>
#include "HM1007.h"

static int address[HM1007_MAX_INSTR + 1];
static int bd[HM1007_MAX_INSTR + 1];
static int instr_cnt;
static int HM1007_err;

/*= INTERNAL DATA ==========================================================
/* cmd is a buffer for GPIB I/O strings */
static char cmd[50];
static char* channel_code[16];
static int flag;
static int test;


/*= UTILITY ROUTINES =======================================================
int HM1007_open_instr (int addr);
int HM1007_close_instr (int instrID);
int HM1007_invalid_short_range (short val, short min, short max,
    int err_code);
int HM1007_invalid_integer_range (int val, int min, int max,
    int err_code);
int HM1007_invalid_longint_range (long val, long min, long max,
    int err_code);
int HM1007_invalid_real_range (double val, double min, double max,
    int err_code);
int HM1007_device_closed (int instrID);
int HM1007_read_data (int instrID, char *buf, long cnt);
int HM1007_write_data (int instrID, char *buf, long cnt);
int HM1007_read_data_file (int instrID, char *filename);
int HM1007_write_data_file (int instrID, char *filename);
int HM1007_poll (int instrID, char *response);
int HM1007_set_timeout (int instrID, int tmo_code);
void HM1007_setup_arrays (void);

/*==========================================================================
/* Function: Initialize
/* Purpose:  This function opens the instrument, queries the
/*           instrument
/*           for its ID, and initializes the instrument to a known
/*           state.
/*==========================================================================
int HM1007_init (int addr, int id_query, int rest, int *instrID)

{
    int  ID;

    if (HM1007_invalid_integer_range (addr, 0, 30, -1) != 0)
        return HM1007_err;
    if (HM1007_invalid_integer_range (id_query, 0, 1, -2) != 0)
        return HM1007_err;
    if (HM1007_invalid_integer_range (rest, 0, 1, -3) != 0)
        return HM1007_err;
    ID = HM1007_open_instr (addr);
    if (ID <= 0)
```

```
        return HM1007_err;

    if (id_query) {
        if (HM1007_write_data (ID, "ID?", 3) != 0) {
            HM1007_close_instr (ID);
            return HM1007_err;
        }
        if (HM1007_read_data (ID, cmd, 50) != 0) {
            HM1007_close_instr (ID);

            return HM1007_err;
        }

        Scan (cmd, "HM/1007");
        if (NumFmtdBytes () != 7)  {
            HM1007_err = 223;
            HM1007_close_instr (ID);
            return HM1007_err;
        }
    }
/*-- End of ID Query ---------------------------------------------*
    if (rest) {
        if (HM1007_write_data (ID, "INIT",4) != 0) {
            HM1007_close_instr (ID);
            return HM1007_err;
        }
    }
/*-- End of Reset ------------------------------------------------*

    if (HM1007_write_data (ID, "PATH OFF", 8) != 0) {
        HM1007_close_instr (ID);
        return HM1007_err;
    }

    HM1007_setup_arrays ();
    *instrID = ID;
    return HM1007_err;
}
/*============================================================*
/* Function: Active channel                                   *
/* Purpose: This function shows the active channels of the scope.   *
/*============================================================*
int HM1007_channel (int instrID,int *active_channel)
{
 int temp_num;

 if (HM1007_invalid_integer_range (instrID,1,HM1007_MAX_INSTR,-1)!=0)
    return HM1007_err;

 Fmt (cmd,"%s<STA\n");
 if (HM1007_write_data (instrID,cmd,NumFmtdBytes())!=0)
    return HM1007_err;
 if (HM1007_read_data (instrID,cmd,20)!=0)
    return HM1007_err;
 temp_num = (int)cmd[0];
 temp_num/= 2;
 if (HM1007_invalid_integer_range (temp_num,1,4,236)!=0)
    return HM1007_err;
 *active_channel = temp_num;
 return HM1007_err;
}
```

```
    for (i=1; i<= HM1007_MAX_INSTR; i++)
        if (address[i] == addr) {
            instrID = i;
            i = HM1007_MAX_INSTR;
         }
    if (instrID <= 0)
        for (i=1; i<= HM1007_MAX_INSTR; i++)
            if (address[i] == 0) {
                instrID = i;
                i = HM1007_MAX_INSTR;
             }
    if (instrID <= 0) {
        HM1007_err = 220;
        return -1;
     }

    if (bd[instrID] <= 0)    {
        if (instr_cnt <= 0)
            CloseInstrDevs("HM1007");
        bd[instrID] = OpenDev ("", "HM1007");
        if (bd[instrID] <= 0)    {
            HM1007_err = 220;
            return -1;
        }
        instr_cnt += 1;
        address[instrID] = addr;
     }
    if (ibpad (bd[instrID], addr) < 0)   {
        HM1007_err = 233;
        return -1;
    }

    return instrID;
}
/*============================================================*
/* Function: Close Instrument                                 *
/*============================================================*
int HM1007_close_instr (int instrID)
{
    if (bd[instrID] != 0) {
        CloseDev (bd[instrID]);
        bd[instrID] = 0;
        address[instrID] = 0;
        instr_cnt -= 1;
     }
    else
        HM1007_err = 221;

    return HM1007_err;
 }
/*============================================================*
/* Function: Invalid Short Range                              *
/*============================================================*
int HM1007_invalid_short_range (short val, short min, short max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    HM1007_err = err_code;
    return -1;
  }
  return 0;
```

```
/*============================================================================*
/*Funtion :Read waveform                                                      *
/*Purpose:This function extraced of stored data frome the scope's            *
/*        memory,with channel selection.                                      *
/*        Transfer of specified data blocks to the controller.                *
/*============================================================================*
int HM1007_read_waveform (int instrID,int select_channel,
    unsigned char *data)
{
  int  points_per_ch;

  if (HM1007_invalid_integer_range (instrID,1,HM1007_MAX_INSTR,-1)!=0
      return HM1007_err;
  if (HM1007_invalid_integer_range (select_channel,1,4,-2)!=0)
      return HM1007_err;
  if (HM1007_device_closed (instrID))
      return HM1007_err;
  points_per_ch = 2048;

  if (select_channel==4)
      select_channel = 8;
  if (select_channel ==3)
      select_channel = 4;
  Fmt (cmd,"%s<DIG %s\n",channel_code[select_channel]);
  if (HM1007_write_data (instrID,cmd,NumFmtdBytes())!=0)
      return HM1007_err;
  if (HM1007_read_data(instrID,data,points_per_ch)!=0)
      return HM1007_err;
/*  if (Scan (data,"%c[z]>%*d",points_per_ch,data_array)!=0)
    {
    HM1007_err=236;
    return HM1007_err;
     }*/
 return HM1007_err;
}

/*============================================================================*
/* Function: Close                                                            *
/* Purpose:  This function closes the instrument.                             *
/*============================================================================*
int HM1007_close (int instrID)
{

    if (HM1007_device_closed (instrID))
        return HM1007_err;

    HM1007_close_instr (instrID);
    return HM1007_err;
}
/*= UTILITY ROUTINES =========================================================*

/*============================================================================*
/* Function: Open Instrument                                                  *
/*============================================================================*
int HM1007_open_instr (int addr)
{
)   int i, instrID;
{
    int i, instrID;
    instrID = 0;
```

```
}
/*============================================================================
/* Function: Invalid Integer Range
/*============================================================================
int HM1007_invalid_integer_range (int val, int min, int max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    HM1007_err = err_code;
    return -1;
  }
  return 0;
}
/*============================================================================
/* Function: Invalid Long Integer Range
/*============================================================================
int HM1007_invalid_longint_range (long val, long min, long max,
    int err_code)
{
    if (val < min || val > max) {
        HM1007_err = err_code;
        return -1;
    }
    return 0;
}
/*============================================================================
/* Function: Invalid Real Range
/*============================================================================
int HM1007_invalid_real_range (double val, double min, double max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    HM1007_err = err_code;
    return -1;
  }

  return 0;
}
/*============================================================================
/* Function: Device Closed
/*============================================================================
int HM1007_device_closed (int instrID)
{
    if (bd[instrID] <= 0)  {
        HM1007_err = 232;
        return -1;
    }
    return 0;
}
/*============================================================================
/* Function: Read Data
/*============================================================================
int HM1007_read_data (int instrID, char *buf, long cnt)
{
    if (ibrd(bd[instrID], buf, cnt) <= 0)
        HM1007_err = 231;
    else
        HM1007_err = 0;

    return HM1007_err;
```

```
/*=============================================================================
/* Function: Write Data
/*=============================================================================
int HM1007_write_data (int instrID, char *buf, long cnt)
{
    if (ibwrt(bd[instrID], buf, cnt) <= 0)
        HM1007_err = 230;
    else
        HM1007_err = 0;

    return HM1007_err;
}
/*=============================================================================
/* Function: Read Data File
/*=============================================================================
int HM1007_read_data_file (int instrID, char *filename)
{
    if (ibrdf (bd[instrID], filename) <= 0)
        HM1007_err = 229;
    else
        HM1007_err = 0;

    return HM1007_err;
}
/*=============================================================================
/* Function: Write Data File
/*=============================================================================
int HM1007_write_data_file (int instrID, char *filename)
{
    if (ibwrtf (bd[instrID], filename) <= 0)
        HM1007_err = 228;
    else
        HM1007_err = 0;

    return HM1007_err;
}
/*=============================================================================
/* Function: Serial Poll
/*=============================================================================
int HM1007_poll (int instrID, char *response)
{
    if (ibrsp (bd[instrID], response) <= 0)
        HM1007_err = 226;
    else

        HM1007_err = 0;

    return HM1007_err;
}
/*=============================================================================
/* Function: Set Timeout
/*=============================================================================
int HM1007_set_timeout (int instrID, int tmo_code)
{
    if (ibtmo (bd[instrID], tmo_code) <= 0)
        HM1007_err = 239;
    else
        HM1007_err = 0;

    return HM1007_err;
```

```
/*=====================================================================
/* Function: Setup Arrays
/*=====================================================================
void HM1007_setup_arrays ()
{
    channel_code[0] = "0";  /* Not a valid number   */
    channel_code[1] = "1";  /* CH1                  */
    channel_code[2] = "2";  /* CH2                  */
    channel_code[3] = "3";  /* CH1 + CH2            */
    channel_code[4] = "4";  /* REF1                 */
    channel_code[5] = "5";  /* CH1+REF1             */
    channel_code[6] = "6";  /* CH2+REF1             */
    channel_code[7] = "7";  /* CH1+CH2+REF1         */
    channel_code[8] = "8";  /* REF2                 */
    channel_code[9] = "9";  /* CH1+REF2             */
    channel_code[10]= "A";  /* CH2+REF2             */
    channel_code[11]= "B";  /* CH1+CH2+REF2         */
    channel_code[12]= "C";  /* REF1+REF2            */
    channel_code[13]= "D";  /* CH1+REF1+REF2        */
    channel_code[14]= "E";  /* CH2+REF1+REF2        */
    channel_code[15]= "F";  /* CH1+CH2+REF1+REF2    */
}
/*= THE END ==========================================================
```

```
#include <gpib.h>
#include <formatio.h>
#include "hm8112.h"

static int address[hm8112_MAX_INSTR + 1];
static int bd[hm8112_MAX_INSTR + 1];
static int instr_cnt;
static int hm8112_err;

/*= INTERNAL DATA ========================================================*
/* cmd is a buffer for GPIB I/O strings */
static char cmd[32];
static char in_data[32];

/*= INSTRUMENT-DEPENDENT COMMAND ARRAYS ==================================*
static char *desire_func[9];

/*= UTILITY ROUTINES =====================================================*
int hm8112_open_instr (int addr);
int hm8112_close_instr (int instrID);
int hm8112_invalid_integer_range (int val, int min, int max,
    int err_code);
int hm8112_invalid_real_range (double val, double min, double max,
    int err_code);
int hm8112_device_closed (int instrID);
int hm8112_read_data (int instrID, char *buf, long cnt);
int hm8112_write_data (int instrID, char *buf, long cnt);
void hm8112_setup_arrays (void);

/*========================================================================*
/* Function: Initialize                                                   *
/* Purpose:  This function opens the instrument and initializes the       *
/*           instrument to a known state.                                 *
/*========================================================================*
int hm8112_init (int addr, int rest, int *instrID)

{
    int  ID;

    if (hm8112_invalid_integer_range (addr, 0, 30, -1) != 0)
        return hm8112_err;
    if (hm8112_invalid_integer_range (rest, 0, 1, -3) != 0)
        return hm8112_err;

    ID = hm8112_open_instr (addr);
    if (ID <= 0)
        return hm8112_err;


    if (rest) {
        if (hm8112_write_data (ID, "INIT", 4) != 0) {
            hm8112_close_instr (ID);
            return hm8112_err;
        }
    }
    if (hm8112_write_data (ID, "PATH OFF", 8) != 0) {
        hm8112_close_instr (ID);
        return hm8112_err;
    }

    hm8112_setup_arrays ();
```

```
    *instrID = ID;
    return hm8112_err;
}

/*============================================================*
/*Function: General Configuration                             *
/*Purpose : This function configures the desired measurement  *
/*          function,                                         *

/*          range and measurement.                            *
/*============================================================*
int hm8112_config (int instrID, int func, int aut,int range,int inTim

{
   if (hm8112_invalid_integer_range (instrID, 1, hm8112_MAX_INSTR, -1
      return hm8112_err;
   if (hm8112_invalid_integer_range (func, 0, 8, -2) != 0)
      return hm8112_err;
   if (hm8112_invalid_integer_range (aut, 0, 1, -3) != 0)
      return hm8112_err;
   if (hm8112_invalid_integer_range (range, 1, 6, -4) != 0)
      return hm8112_err;
   if (hm8112_invalid_integer_range (inTim, 1, 4, -5) != 0)
      return hm8112_err;

   if (hm8112_device_closed (instrID) != 0)
      return hm8112_err;

   /* auto-range */
     if (aut == 1)
        Fmt (cmd,"%sA1T%dSOQOMOPO", desire_func[func], inTim);
   /*manual-range*/
     else
        Fmt (cmd,"%sR%dA0T%dSOQOMOPO", desire_func[func],range,inTim

   if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
      return hm8112_err;
return hm8112_err;
}
/*============================================================*
/*Function: Select function                                   *
/*Purpose : This function configures the desired measurementfunction*
/*============================================================*
int hm8112_function (int instrID, int func)

{
 if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=0
      return hm8112_err;
   if (hm8112_invalid_integer_range (func, 0, 8, -2) != 0)
      return hm8112_err;

   Fmt (cmd,"%sSOQOMOPO",desire_func[func]);
   if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
      return hm8112_err;
return hm8112_err;
}

/*============================================================*
/*Function: Autorange                                         *
/*Purpose : This function configures the desired measurement functio*
/*============================================================*
```

```
int hm8112_at (int instrID)

{
  if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=0
      return hm8112_err;
   if (hm8112_device_closed (instrID) != 0)
      return hm8112_err;

   Fmt (cmd,"%s<A1");
   if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
      return hm8112_err;
return hm8112_err;
}


/*=====================================================================*
/* Function:Offset                                                     *
/* Purpose: Calculates the offset-corrected value                      *
/*=====================================================================*
int hm8112_offset (int instrID)
{
 if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=0
      return hm8112_err;

   if (hm8112_device_closed (instrID) != 0)
      return hm8112_err;

   Fmt (cmd,"%s<P1");
   if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
      return hm8112_err;
return hm8112_err;
}
/*=====================================================================
/* Function: %dev
/* Purpose : Calculates the %deviation value
/*=====================================================================
int hm8112_dev (int instrID)
{
 if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=0
      return hm8112_err;

   if (hm8112_device_closed (instrID) != 0)
      return hm8112_err;

   Fmt (cmd,"%s<P2");
   if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
      return hm8112_err;
return hm8112_err;
return hm8112_err;
}

/*=====================================================================
/* Function: dB
/* Purpose : Calculates the dB value
/*=====================================================================
int hm8112_db (int instrID)
{
   if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!-
      return hm8112_err;
```

```
    if (hm8112_device_closed (instrID) != 0)
       return hm8112_err;

    Fmt (cmd,"%s<P3");
    if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
       return hm8112_err;
return hm8112_err;
}
/*=============================================================================
/* Function: dBm
/* Purpose : Calculates the dBm value
/*=============================================================================
int hm8112_dbm (int instrID)
{
  if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=
       return hm8112_err;

    if (hm8112_device_closed (instrID) != 0)
       return hm8112_err;

    Fmt (cmd,"%s<P4");
    if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0)
       return hm8112_err;
return hm8112_err;
}
/*=============================================================================
/* Function: Read
/* Purpose:  This function reads data set of the instrument.
/*=============================================================================
int hm8112_read (int instrID, int dat)
{
  if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=
       return hm8112_err;
    if (hm8112_invalid_integer_range (dat, 0, 1, -2) != 0)
       return hm8112_err;

    if (hm8112_device_closed (instrID) != 0)
       return hm8112_err;

    Fmt (cmd,"L%d", dat);
    if (hm8112_write_data (instrID, cmd, NumFmtdBytes ()) != 0);
       return hm8112_err;
    if (hm8112_read_data (instrID, in_data, 32) != 0)
       return hm8112_err;
return hm8112_err;
}
/*=============================================================================
/* Function: Clear
/* Purpose : This function clears/resets the instrument;
/*=============================================================================
int hm8112_clear (int instrID)
{
  if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=
       return hm8112_err;

    if (hm8112_device_closed (instrID) != 0)
          return hm8112_err;

    if (ibclr (bd[instrID])<= 0)
       hm8112_err = 224;
    else
       hm8112_err =0;
```

```
    return hm8112_err;
}

/*=============================================================================*
/* Function: close                                                             *
/* Purpose:  This function closes the instrument.                              *
/*=============================================================================*
int hm8112_close (int instrID)
{
  if (hm8112_invalid_integer_range (instrID,1,hm8112_MAX_INSTR,-1)!=0
        return hm8112_err;

    if (hm8112_device_closed (instrID))
        return hm8112_err;

    hm8112_close_instr (instrID);
    return hm8112_err;
}
/*= UTILITY ROUTINES ==========================================================*
/*=============================================================================*
/* Function: Open Instrument                                                   *
/*=============================================================================*
int hm8112_open_instr (int addr)
{
    int i, instrID;

    instrID = 0;

/* Check to see if the instrument is already in the Instrument Table*

    for (i=1; i<= hm8112_MAX_INSTR; i++)
        if (address[i] == addr) {
            instrID = i;
            i = hm8112_MAX_INSTR;

         }


    if (instrID <= 0)
        for (i=1; i<= hm8112_MAX_INSTR; i++)
            if (address[i] == 0) {
                instrID = i;
                i = hm8112_MAX_INSTR;
             }
    if (instrID <= 0) {
        hm8112_err = 220;
        return -1;
     }
    if (bd[instrID] <= 0)    {
        if (instr_cnt <= 0)
            CloseInstrDevs("hm8112");
        bd[instrID] = OpenDev ("", "hm8112");
        if (bd[instrID] <= 0)    {
            hm8112_err = 220;
            return -1;
        }
                return -1;
    }
        instr_cnt += 1;
        address[instrID] = addr;
```

```
     }

    if (ibpad (bd[instrID], addr) < 0)   {
        hm8112_err = 233;
        return -1;
    }

    return instrID;
}
/*=====================================================================*
/* Function: Close Instrument                                          *
/*=====================================================================*
int hm8112_close_instr (int instrID)
{
    if (bd[instrID] != 0) {
        CloseDev (bd[instrID]);
        bd[instrID] = 0;
        address[instrID] = 0;
        instr_cnt -= 1;
     }
    else
        hm8112_err = 221;

    return hm8112_err;
 }
/*=====================================================================*
/* Function: Invalid Integer Range                                     *
/*=====================================================================*
int hm8112_invalid_integer_range (int val, int min, int max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    hm8112_err = err_code;
    return -1;
  }
  return 0;
}
/*=====================================================================*
/* Function: Invalid Real Range                                        *
/*=====================================================================/
int hm8112_invalid_real_range (double val, double min, double max,
    int err_code)
{
  if ((val < min) || (val > max)) {
    hm8112_err = err_code;
    return -1;
  }
  return 0;
}
/*=====================================================================*
/* Function: Device Closed                                             *
/*=====================================================================*
int hm8112_device_closed (int instrID)
{
    if (bd[instrID] <= 0)  {
        hm8112_err = 232;
        return -1;
    }
    return 0;
}
/*=====================================================================*
```

```
/* Function: Read Data                                                    *
/*=======================================================================*
int hm8112_read_data (int instrID, char *buf, long cnt)
{
    if (ibrd(bd[instrID], buf, cnt) <= 0)
        hm8112_err = 231;
    else
        hm8112_err = 0;

    return hm8112_err;
}
/*=======================================================================
/* Function: Write Data
/*=======================================================================
int hm8112_write_data (int instrID, char *buf, long cnt)
{
    if (ibwrt(bd[instrID], buf, cnt) <= 0)
        hm8112_err = 230;
    else
        hm8112_err = 0;

    return hm8112_err;
}
/*=======================================================================
/* Function: Setup Arrays
/*=======================================================================
void hm8112_setup_arrays ()
{
    desire_func[0] = "VD";
    desire_func[1] = "VA";
    desire_func[2] = "ID";
    desire_func[3] = "IA";
    desire_func[4] = "O2";
    desire_func[5] = "O4";
    desire_func[6] = "TC";
    desire_func[7] = "TK";
    desire_func[8] = "TF";

}
/*= THE END ============================================================
```

บทสรุปและวิจารณ์

ปริญญานิพนธ์ในปีการศึกษานี้ประกอบไปด้วย

1. ศึกษา IEEE 488 GPIB (General Purpose Interface Bus)

2. ศึกษา LabWindows/CVI (Getting Started with LabWindows CVI) ซึ่งเป็นโปรแกรมช่วยในการเขียนโปรแกรมใช้งานต่างๆ มีลักษณะเป็นหน้าต่าง (windows) และภายในหน้าต่างเขียนโดยใช้ภาษาซี

3. ศึกษาและทำการเขียนโปรแกรม Instrument driver ของอุปกรณ์ต่อไปนี้คือ Programmable Function Generator HM8130, Power Supply HM8142, Analog Digital Programmable Oscilloscope HM1007 และ Programmable Multimeter HM8112-2

4. ทดลองทำการเขียนโปรแกรมประยุกต์ใช้งานควบคุมอุปกรณ์ เพื่อแสดงให้เห็นการใช้งานได้จริงของ Instrument driver ที่ได้เขียนขึ้นมา โปรแกรมดังกล่าวเขียนบน LabWindows/CVI

Instrument driver ที่ทำการเขียนขึ้นมานี้เป็นเพียงส่วนประกอบหนึ่งใน LabWindows/CVI เท่านั้น แต่เป็นส่วนที่สำคัญที่ช่วยในการเขียนโปรแกรมควบคุมอุปกรณ์นั้นๆ ต่อไป กล่าวคือ Instrument driver จะเหมือนเป็นตัวเชื่อมหรือตัวแปลงฟังก์ชันการทำงานพื้นฐานของอุปกรณ์นั้นๆ ให้อยู่ในระดับภาษาที่สามารถใช้งานได้สะดวกในการเขียนโปรแกรมในขั้นต่อไป

แนวทางในการพัฒนาต่อในปีการศึกษาต่อไป

-จะเห็นได้ว่า LabWindows/CVI ใช้งานได้กว้างมาก ถ้าสามารถศึกษาและทราบถึง function ต่างๆ ภายในโปรแกรมนี้ก็สามารถเขียนและพัฒนาใช้งานต่อไปได้มากมาย เพราะ ไม่ใช่แค่การควบคุมอุปกรณ์เท่านั้น แต่เรายังสามารถเขียนโปรแกรมวิเคราะห์หาค่าและคุณสมบัติของสิ่งที่ต้องการศึกษาได้อีกด้วย

บรรณานุกรม

1. Eugene Fisher and C.W.Jinson,"PET/CBM and the IEEE 488 Bus (GPIB)," Osborne/McGraw-Hill Berkeley, California
2. National Instruments Corporation, "NI-488.2TM Software Reference Manual for MS-DOS,"
3. กนก เจนจิระพงศ์เวช, วารสารเซมิคอนดัคเตอร์ฉบับที่ 78 บทความเรื่อง "GPIB [IEEE 488] บัสอินเตอร์เฟส มาตรฐานและการใช้",บริษัทซีเอ็ดยูเคชั่น จำกัด
4. จรูญ มิ่งขวัญ, ศักดาคม เลาจเวชกุล, อาทิตย์ ศรีแก้ว, "การควบคุมโดยใช้มาตรฐาน IEEE-488 (GPIB)", ปีการศึกษา 2536
5. National Instruments Corporation, " LabWindows/CVI Getting Started with Labwindows/CVI"
6. National Instruments Corporation, " LabWindows/CVI Instrument Driver Developer's Guide"