รายงานวิจัยฉบับสมบูรณ์

# เครื่องปรับตำแหน่งอัตโนมัติสำหรับแถบผ้าทอควบคุมด้วยการประมวลผลภาพ



โดย

รศ. ดร. สุรพันธุ์ เอื้อไพบูลย์



เสนอ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

15 กรกฎาคม 2553

# สารบัญ

# บทที่ 1

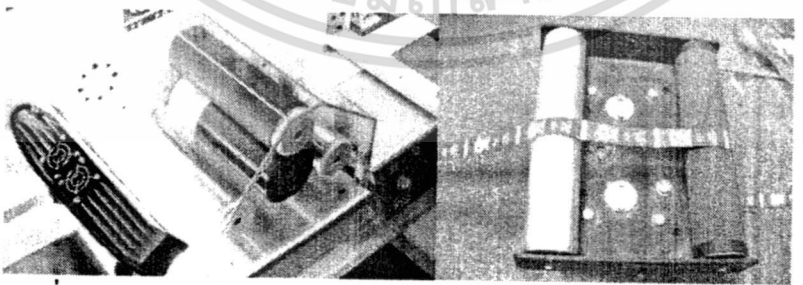
# บทนำ

**1. บทนำ**

ในอุตสาหกรรมสิ่งทอนั้น จะมีการนำเอาผืนผ้าที่ทอเสร็จแล้ว มีลักษะเป็นม้วน ลำเลียงไปด้วยสายพานและลูกกลิ้งเพื่อนำไปตัดให้ได้ขนาดหน้ากว้างต่างๆ กันไปตามความต้องการของลูกค้าผู้สั่งซื้อ ส่วนใหญ่แล้ว การตัดจะใช้เทคโนโลยีสองประเภท คือตัดด้วยลำแสงเลเซอร์และตัดด้วยคลื่นอัลตร้าโซนิค เพราะจะทำให้ขอบของผ้าไม่มีความคมเนื่องจากถูกละลายด้วยความร้อน และเนื่องจากเป็นการตัดแบบไม่สัมผัสชิ้นงาน ทำให้สามารถทำงานได้อย่างรวดเร็วมาก ดังแสดงในรูปที่ 1.1

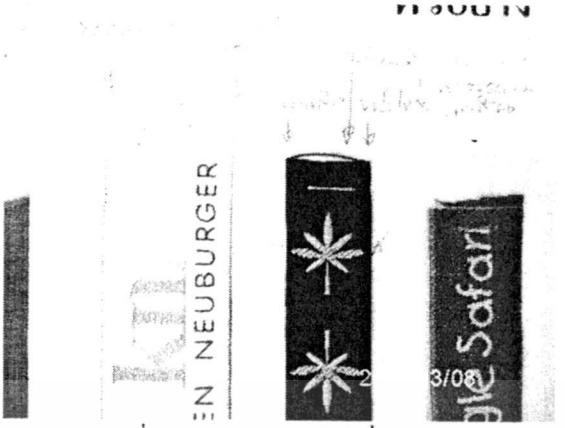
รูปที่ 1.1 กระบวนการตัดผืนผ้าทอ

แต่ปัญหาอุปสรรคที่จะพบมากในกระบวนการตัด ก็คือความแม่นยำในการที่จะควบคุมแถบผ้าให้วิ่งผ่านเครื่องตัดได้อย่างเที่ยงตรงที่สุด โดยปราศจากการแกว่งไปมาของแถบผ้าที่กำลังวิ่งด้วยความเร็วที่สูง (ประมาณ 2 เมตรต่อวินาที) ในการแก้ปัญหานี้ ทางผู้ประกอบการจะต้องสั่งอุปกรณ์ชิ้นหนึ่งที่เรียกกันว่า "Dancer" จากต่างประเทศมาใช้งาน ดังแสดงในรูปที่ 1.2 สายงานผลิตหนึ่งๆก็จะใช้ประมาณ 4-5 เครื่อง ราคาเครื่องละ 2 แสนกว่าบาท



รูปที่ 1.2 เครื่องปรับตำแหน่งอัตโนมัติสำหรับแถบผ้าทอ หรือเรียกว่า Dancer แต่ละแบบ

ถ้าหากว่าเกิดการแกว่งไปมาของผ้าในขณะที่กำลังตัด ก็จะเกิดผลเสียหายต่อแถบผ้าทำให้ขอบไม่ตรงในลักษณะต่างๆกันออกไป ดังแสดงในรูปที่ 1.3




รูปที่ 1.3 ลักษณะแถบผ้าที่เสียหาย

ด้วยเหตุนี้ ทางผู้วิจัยเห็นว่าน่าจะได้มีการพัฒนาสร้างเครื่องปรับตำแหน่งอัตโนมัติสำหรับแถบผ้าทอ หรือเรียกว่า Dancer นี้ขึ้นมาใช้งานได้เองภายในประเทศ ด้วยราคาที่ลดลงเหลือประมาณ 1 ใน 3 ของอุปกรณ์ที่นำเข้ามาจากต่างประเทศ

**2. วัตถุประสงค์ของโครงการวิจัย**

วิจัย ออกแบบ และพัฒนาสร้างต้นแบบเครื่องปรับตำแหน่งอัตโนมัติสำหรับแถบผ้าทอควบคุมด้วยการประมวลผลภาพ

**3. ประโยชน์ที่คาดว่าจะได้รับ**

ลดการนำเข้าอุปกรณ์สำหรับอุตสาหกรรมสิ่งทอ ทดแทนด้วยการผลิตขึ้นมาใช้งานเอง ประหยัดเงินตราของประเทศ ทำให้การเสียดุลการค้าลดลง

**4. ขอบเขตของโครงการวิจัย**

สร้างต้นแบบเครื่องปรับตำแหน่งอัตโนมัติสำหรับแถบผ้าทอควบคุมด้วยการประมวลผลภาพ ที่สามารถนำไปใช้ควบคุมกาววิ่งของแถบผ้าทอไม่ให้มีการแกว่งไปมาได้

**5. ขั้นตอนในการดำเนินโครงการวิจัย**

ออกแบบระบบไฟฟ้า ออกแบบชิ้นส่วนทางกล วงจรควบคุมอุปกรณ์รับภาพ ซอฟท์แวร์ประมวลผลหาตำแหน่งแถบผ้าด้วยไมโครคอนโทรลเลอร์ วงจรส่วนควบคุมทางกล วงจรควบคุมเซอร์โวมอเตอร์แบบปิดสำหรับชดเชยการแกว่งไปมาของแถบผ้าทอ ออกแบบระบบโครงข่ายเซนเซอร์ เพื่อที่จะทำให้อุปกรณ์นี้ ทำงานร่วมกันได้มากกว่าสองตัวขึ้นไป ไมโครคอนโทรลเลอร์ที่ใช้ในงานวิจัยได้เลือกใช้ Propeller P8X32A ของ Parallax เนื่องจากมีการทำงานแบบขนาน 8 CPU พร้อมกัน และมีราคาถูก รายละเอียดจะได้กล่าวต่อไปในบทที่ 2

# บทที่ 2
# ทฤษฎีที่เกี่ยวข้อง

สำหรับบอร์ด ET-BASE PX32 นี้จะเป็นบอร์ด Training โดยใช้ MCU เบอร์ P8X32A –D40 ของ PARALLAX โดยจะเป็น MCU ที่มีความไวสูง ขนาด 32 บิต 8 Cog Multiprocessor โครงสร้างของ MCU จะเป็นตัวถังแบบ DIP 40 PIN สามารถทำงานได้ที่ความถี่สูงสุด 80MHzทำงานที่แรงดัน 2.7-3.6VDCการพัฒนาโปรแกรมจะใช้Software tool "Propeller V1.06" ซึ่ง Software ตัวนี้สามารถใช้เขียนโปรแกรม, Compile Code และ Download Code ผ่านทาง RS232 ได้เลย(ไม่สามารถ Debug ดูการทำงานเป็น STEP ได้) โดยภาษาที่ใช้ในการเขียนโปรแกรมจะเป็นภาษา SPIN ซึ่งจะช่วยให้พัฒนาโปรแกรมได้ง่ายและรวดเร็วขึ้น เนื่องจากใน โปรแกรม Propeller นี้ จะมี Library ต่างๆสำหรับใช้ติดต่อระหว่างอุปกรณ์รอบข้างกับตัว MCU P8X32A ไว้ให้เรียบร้อยแล้ว ซึ่งเวลาใช้งานผู้ใช้ก็สามารถเรียก Library มาใช้ได้เลยตัวอย่าง
เช่น Library RS232 , Library VGA , Library TV , Library Keyboard , Mouse เป็นต้น

**1. คุณสมบัติของ MCU P8X32A**

1.1) เป็น MCU 32 bit 8 Cog Multiprocessor (8 CPU ใน Chip เดียว)

1.2) ตัวถังแบบ DIP 40 PIN มี Port I/O 32 Pin

1.3) ตัว MCU ทำงานที่แรงดัน 2.7-3.6 VDC และ I/O Port สามารถขับกระแส Source/Sink ได้ 40 mA ที่ 3.3 VDC

1.4) ทำงานได้ที่ความถี่สูงสุด 80 MHz สามารถเลือกใช้งาน External Clock หรือ Internal Clock ได้ มี PLL อยู่ภายใน

1.5) มี RAM ภายในสำหรับเก็บ Code 32 Kbyte ซึ่งเวลาตัดไฟเลี้ยง MCU ออก Code ก็จะถูกลบ ดังนั้นเวลาใช้งานจริง
จะต้องต่อ I2C EEPROM ไว้ภายนอกเพื่อเก็บ Code

1.6) Pin ที่ถูกกำหนดให้ทำงานเป็น Input จะสามารถรับระดับแรงดัน Input ได้ไม่เกิน VDD(2.7-3.6V) เท่านั้น

1.7) ความเร็วในการทำงานภายในของ Chip จะอยู่ที่ 20 MIPS/cog

**2. คุณสมบัติของบอร์ด ET- BASE PX32 V1.0**

2.1) ใช้ MCU P8X32A ตัวถังแบบ DIP 40 PIN

2.2) มี I2C EEPROM #24LC256 (32Kb) สำหรับใช้เก็บ Code Program

2.3) ตัวบอร์ดรับแรงดัน Input 6-12 VDC โดยมีชุด Regulator ปรับแรงดันลงให้เหลือ 5V และ 3.3V อยู่ในบอร์ด

2.4) ใช้ Crystal 5.00 MHz (External) สามารถใช้ PLL ภายในตัว Chip คูณเพิ่มความถี่ให้สูงขึ้นได้ถึง 80 MHz

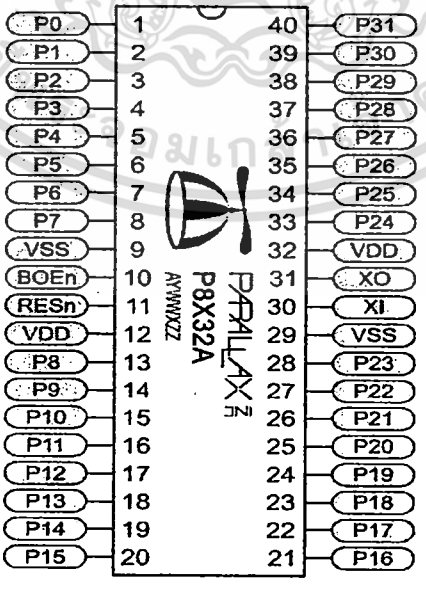
2.5) พัฒนาโปรแกรมด้วยภาษา SPIN โดยใช้ Tool "Propeller" (Freeware) ซึ่ง Tool นี้จะใช้สำหรับเขียนโปรแกรม , Compile และ Download ภายในตัว และจะมี Library ให้ใช้ในการInterface ระหว่างตัว MCU กับอุปกรณ์รอบข้างที่จัดไว้ให้บนบอร์ดเรียบร้อยแล้ว และสามารถ
เข้าไป Download Library เพิ่มเติมได้ที่ http://www.parallax.com/
ในส่วนของการใช้งานภาษา SPIN ก็สามารถดูได้จาก Help ของโปรแกรม Propeller

2.6) การ Download Code จะ Download ผ่านทาง RS232 โดยสามารถเลือกจากตัว Propeller ได้ว่า จะ Download Code ไปเก็บไว้ยัง EEPROM (External) หรือ RAM ภายใน MCU

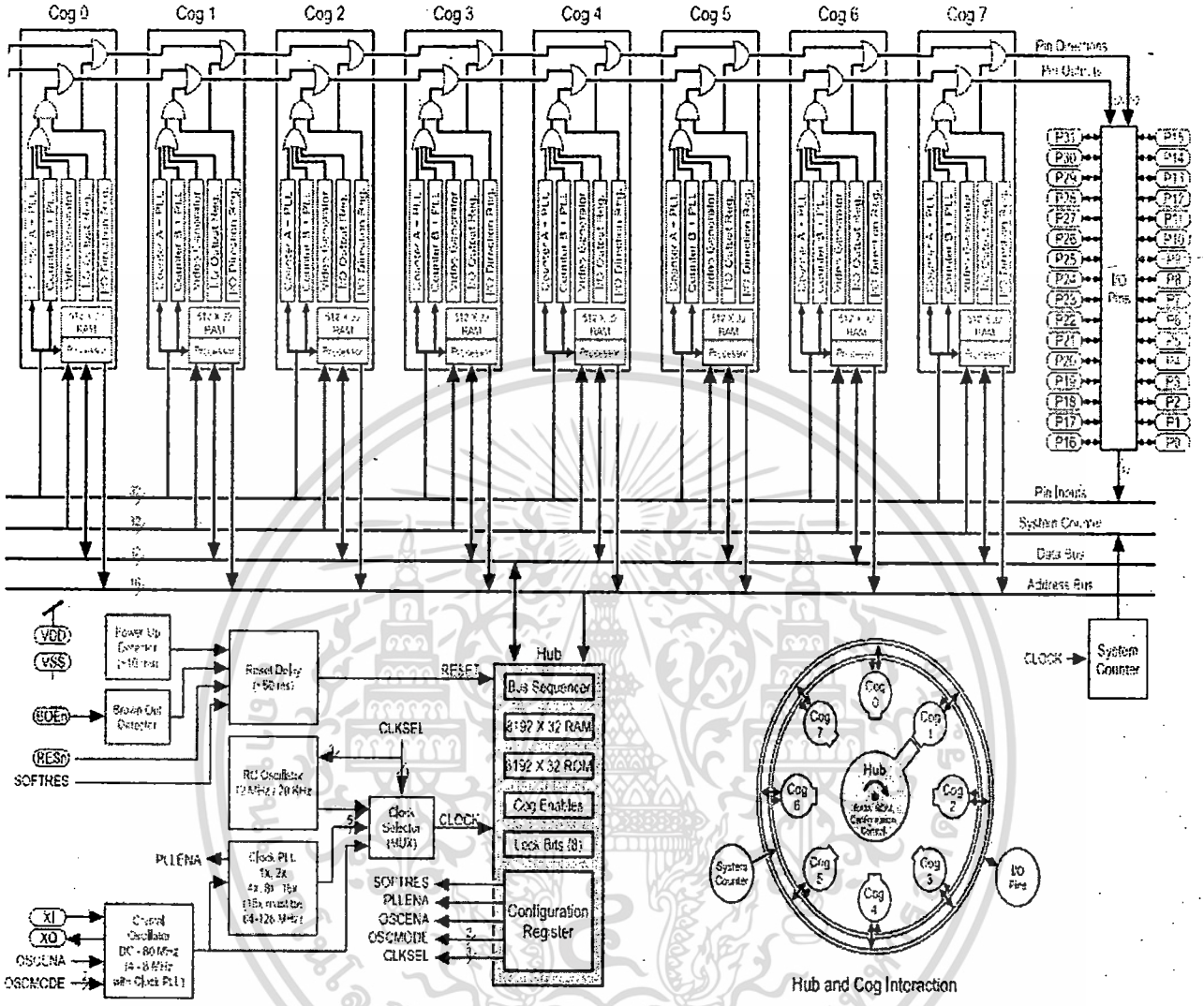
2.7) บอร์ดนี้ได้จัดสรร I/O ไว้ตายตัวสำหรับ Interface กับอุปกรณ์รอบข้างดังนี้

1 . Port Key Board(PS2) , 2. Port Mouse (PS2) , 3. Port RS232 , 4. Port VGA , 5. Port TV(AV) , 6. Port MIC , 7. PortHeadphone , 8. Port I/O 8 PIN สำหรับใช้งานด้านอื่นๆ

**3. โครงสร้างและบล็อกไดอะแกรมของ MCU P8X32A**




**รูปที่ 2.1 แสดงโครงสร้างของ Chip P8X32A(DIP 40 PIN)**




รูปที่ 2.2 แสดงบล็อกไดอะแกรมของ Chip P8X32A

จากรูปที่ 2.1 จะเห็นว่าโครงสร้างภายในของ Chip จะประกอบไปด้วย Processor 8 ตัว โดยจะเรียกว่า "Cog" ซึ่งถูกออกแบบให้มีการทำงานที่ความไวสูง สิ้นเปลืองพลังงานต่ำ ตัวถังมีขนาดเล็ก มีความคล่องตัวในการทำงานผ่านทาง Processor ทั้ง 8 ตัวสูง โดยสามารถทำงานได้ในเวลาเดียวกันพร้อมๆกัน และเป็นอิสระต่อกัน ซึ่ง Chip นี้จะใช้การ Share ทรัพยากรผ่านทางศูนย์กลาง HUB เพื่อให้ Cog แต่ละ Cog สามารถใช้งานทรัพยากรร่วมกันได้ในส่วนของระบบ Clock จะถูก Share ไปยัง Cog

แต่ละ Cog โดยจะอ้างอิงที่ฐานเวลาเดียวกัน ส่วนการ Interrupt จะไม่ถูกใช้กับ Chip นี้ แต่จะใช้การกำหนดตำแหน่งที่จะกระโดดไปทำงานให้กับ Cog ต่างๆโดยตรงเลย

ในรูปที่2.2 จะเป็นโครงสร้างตัวถังแบบ DIP 40 PIN ของ Chip โดยจำแนก PIN ต่างๆได้ดังนี้

**ตารางที่2.1** รายละเอียด PIN

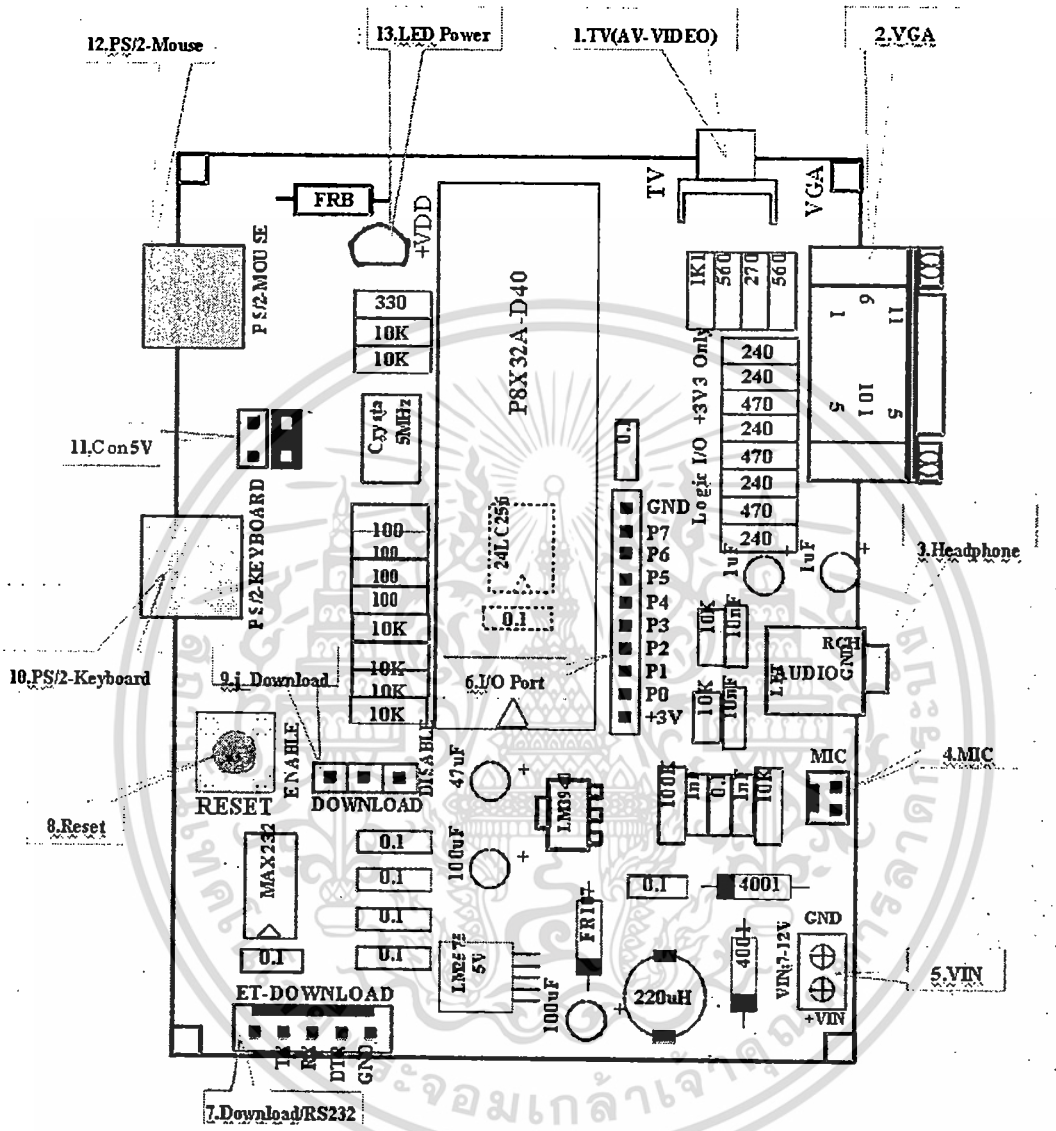
| PIN Name | Direction | Description |
|---|---|---|
| P0-P31 | I/O | เป็น Port I/O โดยมีระดับ Logic อยู่ที่ประมาณครึ่งหนึ่งของ VDD หรือ 1.6 VDC ที่แรงดัน 3.3 VDC ใน 32 Pin นี้จะมีอยู่ด้วยกัน 4 Pin ซึ่งจะถูกกำหนดให้ทำงานใน<br>หน้าที่พิเศษหลังจาก Power-up หรือ Reset คือ<br>P28 – I2C SCL ซึ่งจะใช้ต่อไปยัง EEPROM ภายนอก<br>P29 – I2C SDA ซึ่งจะใช้ต่อไปยัง EEPROM ภายนอก<br>P30 – Serial Tx ใช้ต่อสำหรับ Download Code และส่งข้อมูล ผ่านทาง RS232<br>P31 – Serial Rx ใช้ต่อสำหรับ Download Code และรับข้อมูล ผ่านทาง RS232 |
| VDD | --- | 3.3 V Power (2.7-3.6 VDC) |
| VSS | --- | Ground |
| BOEn | I | Brown-Out Enable(Active Low) จะใช้ต่อไปยัง VDD หรือ VSS ถ้าขานี้เป็น Low ขา RESn จะกลายเป็น Output<br>แต่จะยังสามารถ Drive Low เพื่อ Reset Chip ได้ ถ้าเป็น high ขา RESn จะทำหน้าที่เป็น Input |
| RESn | I/O | Reset(Active Low) เมื่อเป็น Low ตัว Chip และ Cog ทั้งหมด จะถูก Disable PIN I/O จะถูกปล่อยลอย และ Chip<br>จะ Restart ภายในเวลา 50ms หลังจาก Logic ที่ RESn เปลี่ยนจาก Low เป็น High |
| XI | I | Crystal Input |
| XO | O | Crystal Output |

**4. โครงสร้างของบอร์ด ET-BASE PX32 V1.0**

สำหรับบอร์ด ET-BASE PX32 V1.0 นี้ ได้กำหนด Port ในการ Interface กับอุปกรณ์รอบข้างเฉพาะอย่างไว้ให้เรียบร้อยแล้ว จะเหลือในส่วนของ Port I/O (P0-P7)อีก 8 Pin เท่านั้น ที่จัดไว้สำหรับให้ผู้ใช้นำไปใช้ต่อกับอุปกรณ์อื่นๆ
เพิ่มเติมได้ เช่น LED , SW, SD Card , LCD เป็นต้น

**ตารางที่2.2** แสดง Port ที่ถูกต่อใช้งานบนบอร์ด ET-BASE PX32

| Port Number | หน้าที่ |
|---|---|
| P0-P7 | ถูกกำหนดให้เป็น I/O ที่ผู้ใช้สามารถนำไปต่อใช้งานอื่นๆเพิ่มเติมได้ |
| P8-P9 | ถูกกำหนดให้ใช้สำหรับต่อ Microphone |
| P10-P11 | ถูกกำหนดให้เป็น AUDIO Out Stereo สามารถต่อไปยังเครื่องขยาย หรือ Headphone ได้ |
| P12-P15 | ถูกกำหนดให้เป็น Port Video Out (NTSC/PAL)สำหรับต่อ เข้าช่อง AV ของ TV |
| P16-P23 | ถูกกำหนดให้ใช้เป็น Port VGA สำหรับต่อเข้ากับจอ PC หรือ จอ LCD |
| P24-P25 | ถูกกำหนดให้ใช้สำหรับต่อ PS/2 Mouse |
| P26-P27 | ถูกกำหนดให้ใช้สำหรับต่อ PS/2 Key Board |
| P28-P29[System] | ถูกต่อเข้ากับ I2C-EPPROM เพื่อใช้เก็บ Code Program (P28:Clock ; P29:Data) |
| P30-P31[System] | ถูกกำหนดให้ใช้เป็น Port RS232 เพื่อ Download Program และรับ-ส่งข้อมูลทาง RS232(P30:Rx;P31:Tx) |




รูปที่2.3 แสดงโครงสร้างของบอร์ด ET-BASE PX32 V1.0

หมายเลข1 : TV(AV-VIDEO) เป็นขั้วต่อ VIDEO OUT เพื่อใช้ต่อกับ TV โดยต่อเข้าที่ขั้วต่อ VIDEO IN ของ TV

หมายเลข2 : VGA เป็นขั้วต่อ VGA OUT (DB15) ใช้สำหรับต่อไปยัง Monitor ของ PC เช่น จอ LCD

หมายเลข3 : Headphone เป็นขั้วต่อ AUDIO OUT ใช้สำหรับต่อหูฟัง หรือต่อเข้ากับชุดขยายสัญญาณ

หมายเลข4 : MIC เป็นขั้วต่อ Microphone

หมายเลข5 : VIN เป็นขั้วต่อไฟเลี้ยงบอร์ด DC 7V-12 V ในการต่อจะต้องระวัง ต่อขั้วบวก-ลบให้ถูกต้องด้วย

หมายเลข6 : I/O Port เป็น Port I/O 8 Pin จัดไว้สำหรับให้ผู้ใช้สามารถต่อ I/O อื่นๆเพิ่มเติมได้ อาทิเช่น LCD, Key, LED หรือ SD Card เป็นต้น

ข้อควรระวัง สำหรับ I/O Port นี้ในกรณีที่ใช้งานเป็น Input จะรับแรงดันที่เข้ามายัง Port ได้ ไม่เกิน VDD หรือ 3.3 V ห้ามใช้แรงดัน 5 Vมาต่อเข้าโดยตรงเพราะจะทำให้ Port Pin เสียหายได้ ควรใช้ R มาต่อ Dividerให้เหลือ3.3V เสียก่อน

หมายเลข 7 : Download/RS232 เป็นขั้วต่อใช้สำหรับ Download Program และใช้ ในการสื่อสาร รับ-ส่ง ข้อมูลผ่านทาง RS232 โดยมี Line Driver Max232 เป็นตัวปรับระดับสัญญาณ Rx,Tx จาก 3.3V ไปเป็น 12V เพื่อให้ต่อใช้งานร่วมกับ PC ได้

หมายเลข8 : Reset เป็น SW.สำหรับใช้ Reset การทำงานของ MCU โดยจะทำงานที่Logic 0

หมายเลข9 :Jumper Download เมื่อจะ Download Program ให้ Set jumper มาทางด้าน Enable ซึ่ง Jumper นี้จะทำหน้าที่ต่อขา RES ของ MCU เข้ากับขา DTR ของขั้ว Download ถ้า Set Jumper มาทางด้าน Disable จะเป็นการตัดขา RES ของ MCU ออกจากขา DTR ของขั้ว Download เพื่อป้องกันสัญญาณ Reset จาก PC หลังจากดาวน์โหลดเสร็จแล้ว และยังไม่ได้ถอดสาย Download ออก

หมายเลข10 : PS/2-Keyboard เป็นขั้วต่อ PS/2 ใช้สำหรับต่อ keyboard

หมายเลข11 : Con 5V เป็นขั้วต่อ DC 5 V Output สำหรับใช้ต่อไฟ 5 V จากบอร์ดไปใช้งานภายนอกได้

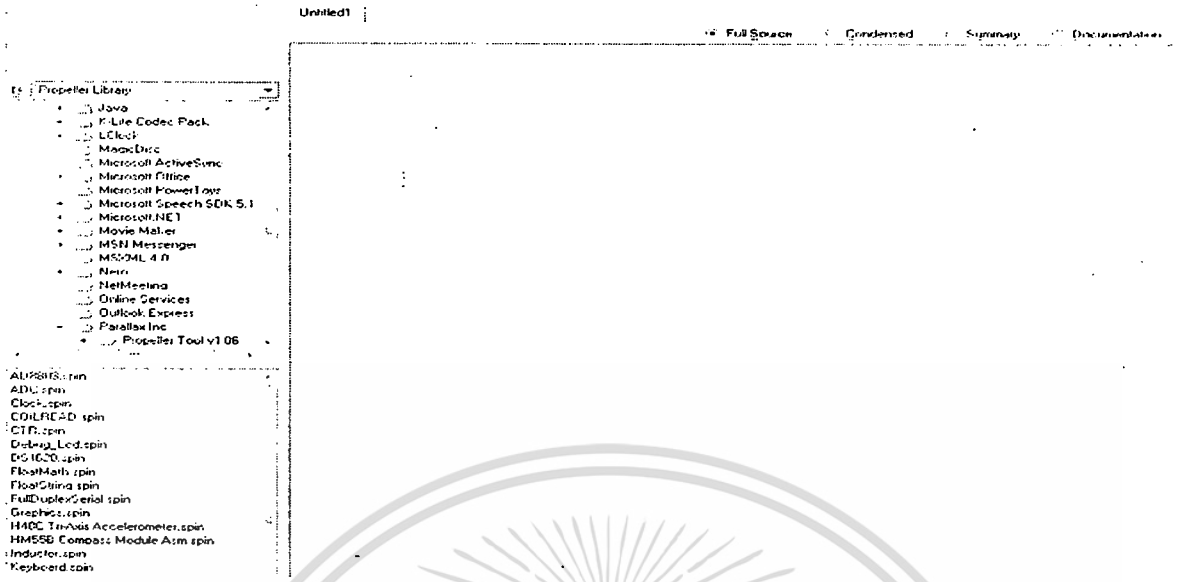
หมายเลข12 : PS/2-Mouse เป็นขั้วต่อ PS/2 ใช้สำหรับต่อ Mouse

หมายเลข13 : LED Power แสดงสะถานะการทำงานของแหล่งจ่ายไฟ

**5. การใช้งานโปรแกรม Propeller เบื้องต้น**

**5.1) ทำการติดตั้งโปรแกรม Propeller V1.06 ลงในเครื่อง**

**5.2) หลังจากติดตั้งแล้วให้เปิดโปรแกรม Propeller ขึ้นมาจะได้หน้าต่างดังรูปที่ 3.4**



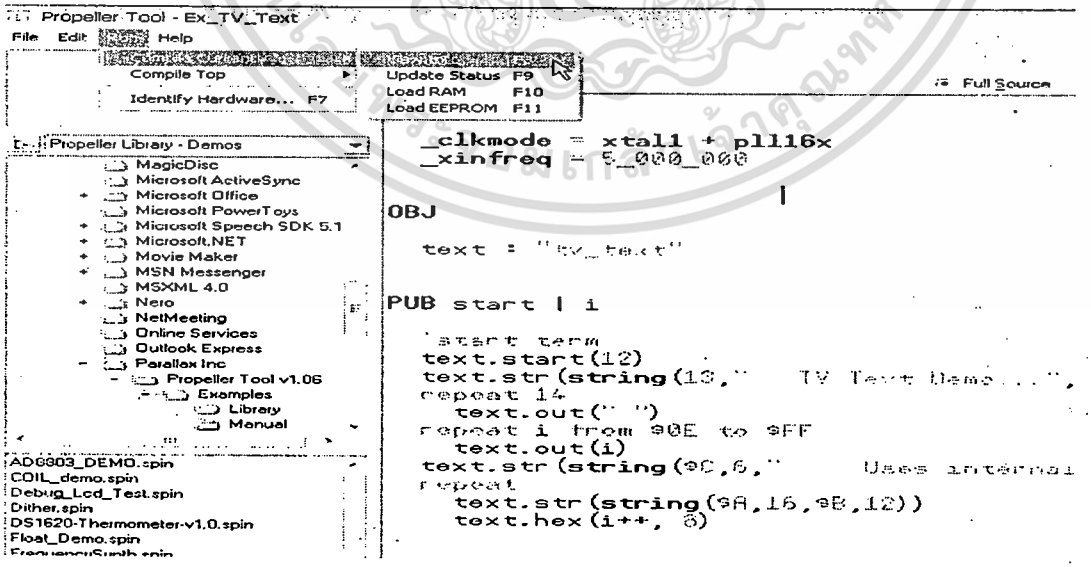

รูปที่2.4 หน้าต่าง Propeller

จากรูปในหน้าต่างด้านขวามือ(แท็บUntitled1) จะเป็นพื้นที่ว่างสำหรับใช้เขียนโปรแกรม

5.3) หลังจากเขียนโปรแกรม เสร็จก็ให้ไปที่เมนู File เลือก Save As เพื่อทำการ Save File เป็นนามสกุลจุด spin

5.4) เมื่อ Save File เรียบร้อยแล้ว ก็ให้ไปที่เมนู RUN และเลือกที่ Compile Current จากนั้นก็จะมีเมนูให้ผู้ใช้ เลือกต่ออีก

ดังแสดงในรูปที่ 2.5 ถ้าเลือก



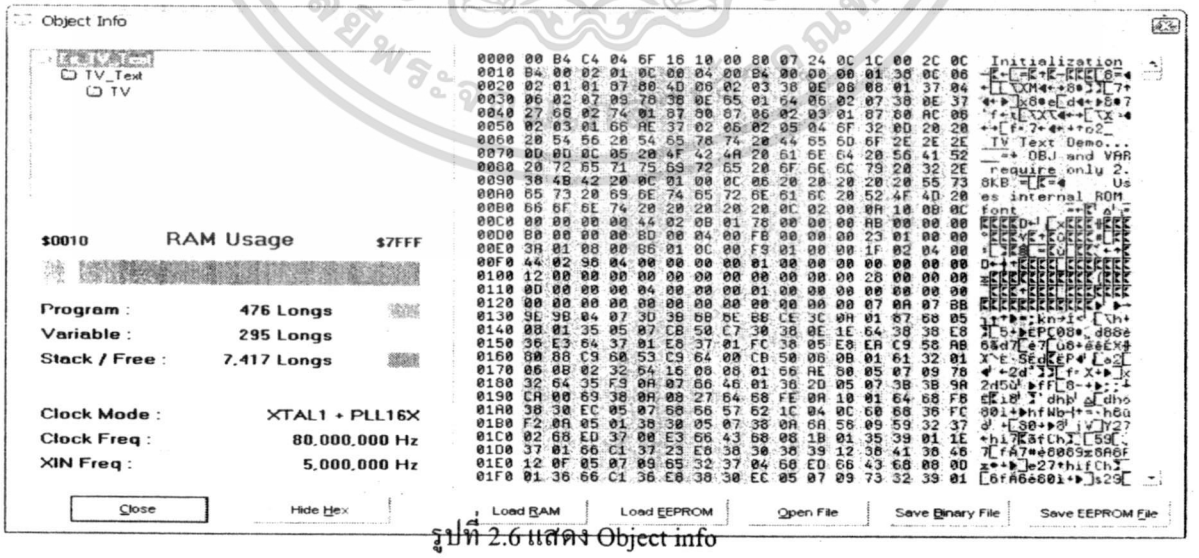

รูปที่ 2.5 เมนู Run ใน Propeller

View Info... = จะเป็นการ Compile โปรแกรมอย่างเดียว และจะมีหน้าต่าง Object info รายงาน

ข้อมูลต่างๆของโปรแกรมที่ เขียนขึ้น ดังรูปที่5.3 ซึ่งก็จะแสดงพื้นที่ RAM ที่ถูกใช้ไป แสดงความถี่ Clock ที่ผู้ใช้เลือกใช้งาน ซึ่งข้อมูลของตัวอย่างโปรแกรมในรูปที่ 5.3 นี้ สามารถอธิบายได้คือ

- เป็นข้อมูลของไฟล์ที่ชื่อ Ex_TV_Text โดยภายในไฟล์นี้ ได้เรียกใช้งาน Function ที่อยู่ใน File Library ที่ชื่อ TV_Text และ ใน File library TV_Text ก็ได้เรียกใช้ Function ใน File library ที่ชื่อTV อีกต่อหนึ่ง
- ไฟล์ Ex_TV_Text นี้ ใช้เนื้อที่ RAM ในส่วนที่เป็นโปรแกรมไปทั้งหมด 476 Long และในส่วนที่เป็นตัวแปร ไปทั้งหมด 295 Long และเหลือพื้นที่ RAM ที่ยังว่างอยู่ทั้งหมด 7,417 Long โดย 1 Long จะมีค่าเท่ากับ 32 bit หรือ 4 byte
- ในโปรแกรมนี้ได้เลือกใช้ Clock Mode ในโหมด XTAL1+PLL16X โดยความถี่ Clock ที่ใช้จริงก็จะเท่ากับ 80 Mhz โดยใช้ Crystal จากภายนอก 5 MHz (5MHz x PLL16 = 80MHz)

หลังจากหน้าต่าง Object Info แสดงขึ้นมาแล้ว ให้ผู้ใช้เลือกคลิกที่ปุ่ม Load Ram หรือ ปุ่ม Load EEPROM เพื่อทำการ Download Program โดยถ้าเลือกปุ่ม Load RAM โปรแกรมจะถูก Load เข้า ไปเก็บไว้ยังพื้นที่ RAM ของ MCU โดยตรง ซึ่งการโหลดวิธีนี้ โปรแกรมจะถูกลบเมื่อมีการกด SW.Reset หรือเอาไฟเลี้ยงบอร์ดออก , ถ้าเลือกปุ่ม Load EEPROM โปรแกรมจะถูก Load เข้าไปเก็บไว้ยัง EEPROM ที่ต่ออยู่ภายนอกก่อน จากนั้น ตัว MCU ก็จะทำการดึง Code จาก EEPROM เข้าไป RUN ใน RAM ให้อัตโนมัติ ซึ่งการโหลดวิธี นี้ โปรแกรมจะไม่ถูกลบเมื่อมีการกด SW. Reset หรือเอาไฟเลี้ยงบอร์ดออก




รูปที่ 2.6 แสดง Object info

Load RAM = จะเป็นการ Compile และ Download โปรแกรมลงใน RAM เลย จะไม่แสดง

หน้าต่าง Object Infoให้เห็นโปรแกรมจะถูกลบ ถ้ามีการกด SW. Reset หรือ ตัดไฟเลี้ยงออก

Load EEPROM = จะเป็นการ Compile และ Download โปรแกรมลงใน EEPROM ที่ต่ออยู่ภายนอกจะไม่แสดงหน้าต่าง Object Infoให้เห็นเช่นกัน โปรแกรมจะยังคงอยู่ เมื่อ มีการ Reset หรือตัดไฟเลี้ยงออก

5.5) สำหรับการ Set Baud Rate หรือ เลือก COM Port เพื่อใช้ในการ Download Code นั้นจะไม่จำเป็นเพราะ ตัวโปรแกรม Propeller นั้นจะทำการ Set Baud Rate และหา Com Port ให้โดยอัตโนมัติ แต่ถ้าผู้ใช้ต้องการจะกำหนด Com Port เองก็สามารถเข้าไปแก้ไขได้โดยไปที่เมนู Edit แล้วเลือก Preferences จะมีหน้าต่างขึ้นมาให้เลือก ที่แท็บ Operation ในช่อง Serial Port Search : ให้เลือก Comport ที่จะใช้งาน (ปกติ = Auto)

5.6) สำหรับตัวโปรแกรม Propeller นี้จะใช้การพัฒนาโปรแกรมด้วยภาษา SPIN ซึ่งรูปแบบการใช้งานของภาษา SPIN นี้ผู้ใช้สามารถดูคำสั่งการใช้งานได้ โดยไปที่เมนู Help แล้วเลือกที่ Propeller Manual (pdf)

**6. การทดสอบ RUN ตัวอย่างโปรแกรม**

1) ทำการติดตั้งโปรแกรม Propeller Tool V1.06.exe ให้เรียบร้อย และทำการ Copy ตัวอย่างของ ETT จาก CD ROM มาวางไว้ใน PC ให้เรียบร้อย

2) ต่อสาย Download จาก Com Port ของ PC มาเข้าที่ขั้วต่อ Download ของบอร์ด และ Set jumper Download (หมายเลข9)มาทางด้าน Enable

3) ในที่นี้จะขอทดสอบการแสดงผลด้วย TV ดังนั้น ให้ต่อสายสัญญาณจาก TV OUT ของบอร์ดไปเข้าที่ขั้วต่อ VIDEO IN ของ TV แล้วทำการเปิด TV ในช่อง AV รอไว้ จากนั้นจ่ายไฟเลี้ยงให้กับตัวบอร์ด

4) จากตัวอย่างของ ETT ให้เข้าไปที่ Folder " Ex1_TV " จากนั้นเลือกที่ Folder "Ex1.2_TvText _Graphic" แล้วดับเบิ้ลคลิกที่ไฟล์ "Ex1.2_TV_TextGPH.spin" โปรแกรม Propeller ก็จะทำการเปิดไฟล์นี้ขึ้นมา

5) จากนั้นให้ไปที่เมนู RUN แล้วเลือกที่ Compile Current แล้วคลิกเลือก Load RAM หรือ Load EEPROM ก็ได้ โปรแกรมก็จะถูก Compile และ Download ลงบน MCU

6) หลังจาก โหลดโปรแกรมเสร็จให้สังเกตที่หน้าจอ TV จะเห็น ข้อความ "Hello!" แสดงขึ้นที่หน้าจอ

7) ถ้าต้องการทดสอบตัวอย่างอื่นๆอีกก็ให้ทำตามขั้นตอนที่กล่าวไปข้างต้น พร้อมกับหาอุปกรณ์มาต่อเข้ากับ Port ที่จะใช้ทดสอบด้วย

นอกจากตัวอย่างของ ETT ที่เขียนมาให้แล้ว ผู้ใช้ยังสามารถทดสอบ RUN ตัวอย่างที่มีอยู่ในโปรแกรม Propeller บน
บอร์ดของ ETT ได้ด้วย โดยทำดังนี้

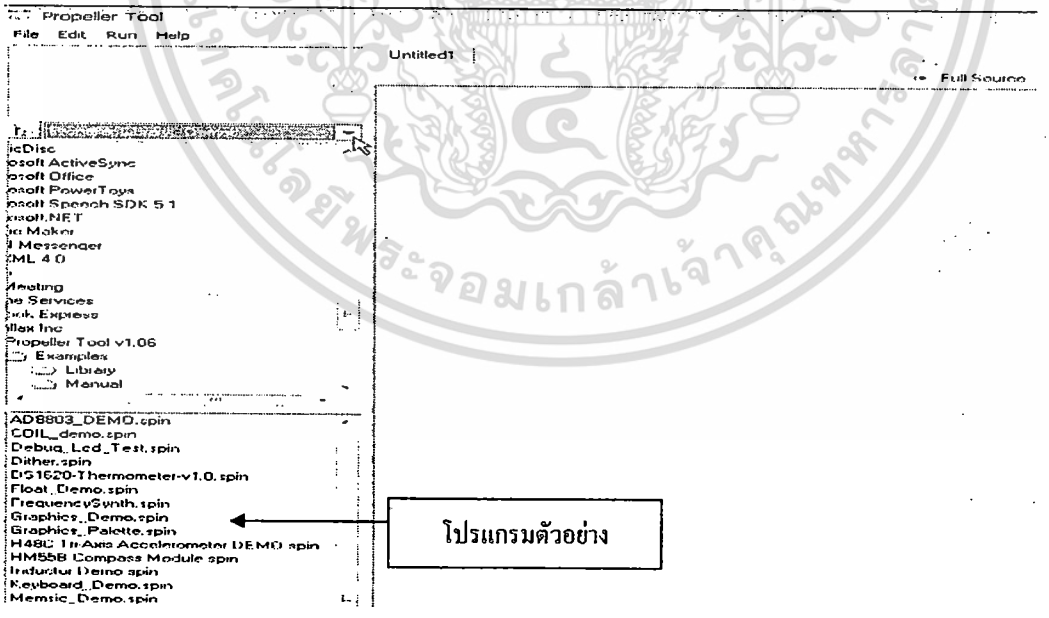
1) เปิดโปรแกรม Propeller ขึ้นมา ในช่องด้านซ้ายมือ ดูในรูปที่6.1 ในตำแหน่งที่ Mouse ชี้ ให้เลือกที่ Propeller Library - Demos จากนั้นจะเห็นไฟล์ตัวอย่างที่เป็นนามสกุลจุด SPIN แสดงอยู่ในช่องด้านล่างซ้ายมือ

2) ดับเบิลคลิกไฟล์ตัวอย่างที่ต้องการจะ RUN ขึ้นมา โดยควรจะเลือกไฟล์ที่สนับสนุนการ Interface ที่มีอยู่บนบอร์ดของอีทีที ที่จัดไว้ให้ ตัวอย่างไฟล์ที่ Interface ทาง Port VGA เช่น VGA_DEMO.spin , ตัวอย่างไฟล์ที่ Interface ทาง Port Mic ,Audio เช่น Microphone to Headphones.spin, SingingDemoSeven.spin เป็นต้น

3) ไปที่เมนู RUN แล้วเลือกที่ Compile Current แล้วคลิกเลือก Load RAM หรือ Load EEPROM ก็ได้ โปรแกรมก็จะถูกCompile และ Download ลงบน MCU

4) สังเกตการณ์ทำงานของโปรแกรม จากอุปกรณ์ที่นำมาต่อ Interface

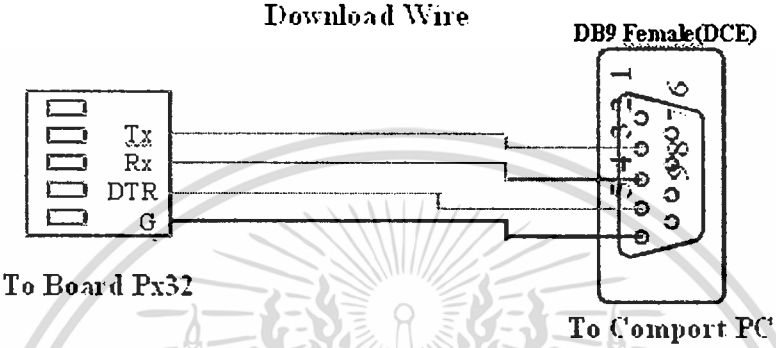
ในกรณีที่ผู้ใช้เปิดโปรแกรมขึ้นมาซ้อนกันหลายๆหน้าต่าง ตัวโปรแกรม Propeller จะทำการ Compile และ Download โปรแกรม เฉพาะในส่วนของไฟล์ที่ถูก Active แสดงให้เห็นอยู่ที่หน้าจอเท่านั้น




**รูปที่ 2.7 แสดงหน้าต่างโปรแกรมตัวอย่าง**

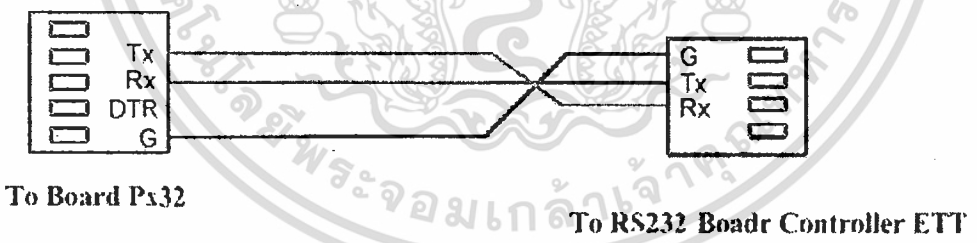
**7. การต่อสาย Download และการต่อสายสื่อสาร RS232**

-สำหรับสาย Download Program และสายสำหรับรับ-ส่งข้อมูลทาง RS232 ระหว่างบอร์ด PX32 กับ PC จะใช้สายเส้นเดียวกันโดยการเข้าสายแสดงดังรูปที่ 7.1




รูปที่ 2.8 แสดงการเข้าสาย Download & สายสื่อสาร RS232 ระหว่างบอร์ด Px32 และ PC

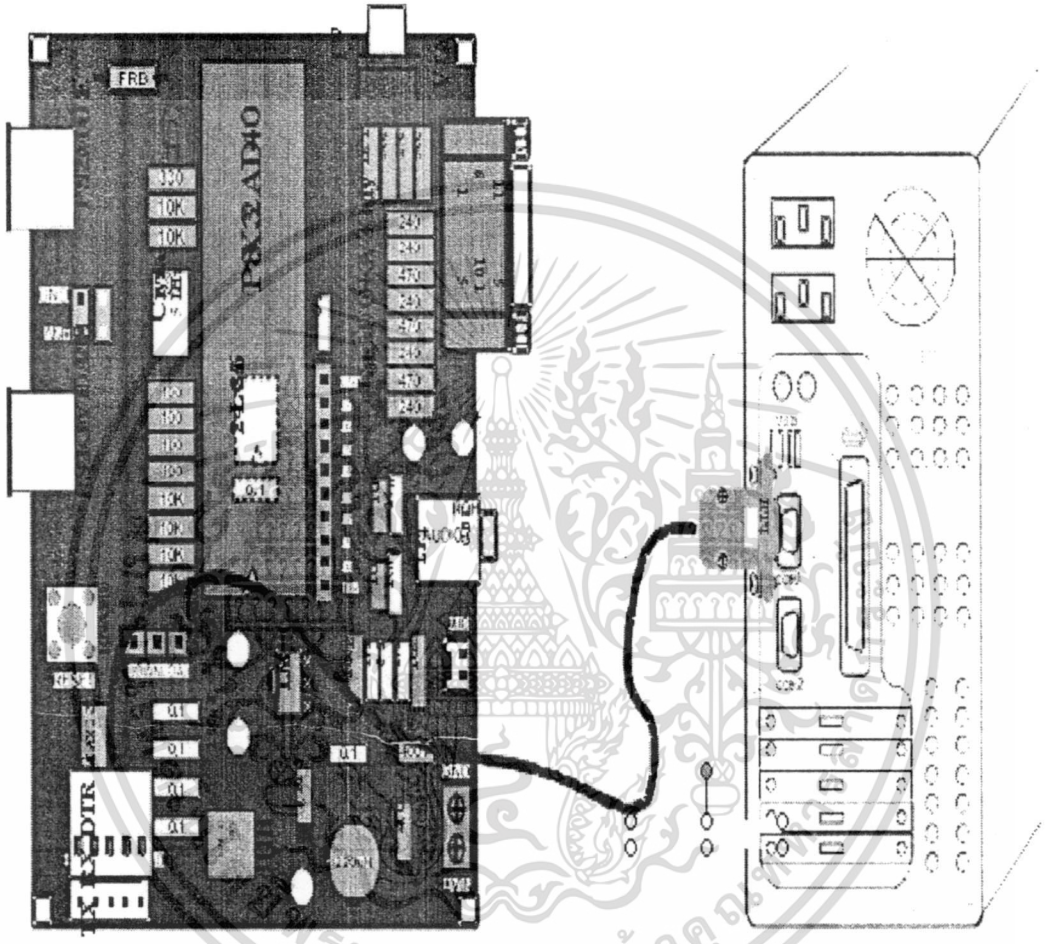
ในกรณีที่จะ รับ-ส่ง ข้อมูลทาง RS232 ระหว่างบอร์ด PX32 กับบอร์ด Controller อื่นๆ สามารถใช้การเข้าสายดังแสดงในรูปที่ 2.9




รูปที่ 2.9 แสดงการเข้าสาย RS232 สำหรับ รับ-ส่ง ข้อมูล ระหว่างบอร์ด PX32 กับบอร์ด Controller อื่นๆ

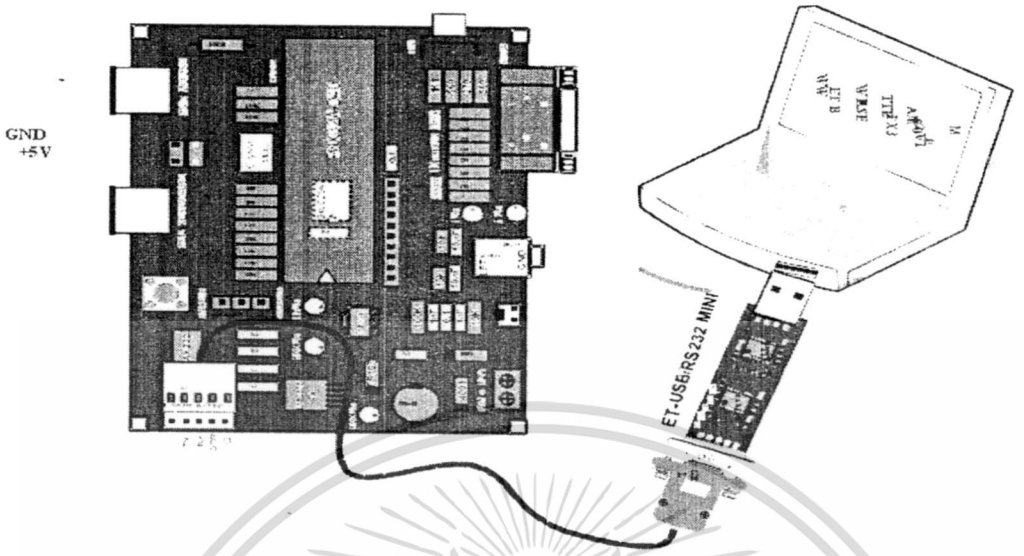
-เมื่อจะทำการ Download Program ลงบนบอร์ด Px32 ก็ให้ทำการต่อสาย download ทางด้าน Connector 5 Pin เข้ากับขั้วต่อของบอร์ด PX32 ส่วนทางด้าน Connector DB9 ก็ให้ต่อเข้ากับ Com Port ของ PC ที่ใช้งานดังรูปที่ 7.3 และทำการ Set Jumper Download(หมายเลข9) บนบอร์ด PX32 มาทาง

ด้าน Enable , เมื่อจะทำการรับ-ส่ง ข้อมูล ผ่านทาง RS232 ของบอร์ด PX32 กับ PCหรือกับบอร์ด Controller อื่นๆ หลังจาก Download โปรแกรมเรียบร้อยแล้ว ก็ควรจะSet Jumper Download (หมายเลข9) บนบอร์ด PX32 มาทางด้าน Disable




รูปที่ 2.10 แสดงการต่อสาย Download เข้ากับ Com-Port ของ PC โดยตรง

ในกรณีที่ใช้เครื่อง PC หรือ Note Book ที่ไม่มีขั้วต่อ Com-Port ผู้ใช้จะต้องใช้ชุดแปลงจาก USB ไปเป็น RS232 เสียก่อน โดยถ้าใช้ชุดแปลงของ ETT รุ่น ET-USB/RS232 Mini ก็สามารถต่อใช้งานดังแสดงในรูปที่ 2.11




รูปที่ 2.11 แสดงการต่อสาย Download โดยผ่านตัวแปลงชุด ET-USB/RS232 Mini

**8. ข้อกำหนดที่ควรรู้ในการพัฒนาโปรแกรมด้วยภาษา SPIN บน Tool Propeller**

8.1) สำหรับภาษา SPIN นี้เวลาใช้งานเราจะมองออกเป็น Block ซึ่งมีอยู่ด้วยกันทั้งหมด 6 Block โดยเวลาจะเขียนก็จะต้องพิมพ์ชื่อเฉพาะของแต่ละ Block ที่ถูกกำหนดไว้ตายตัวด้วย โดยเวลาพิมพ์ชื่อเฉพาะของแต่ละ Block นี้ มีข้อกำหนดว่าจะต้องเริ่มพิมพ์ที่ Colum ซ้ายสุดของหน้าต่างที่จะใช้เขียนโปรแกรมเสมอ โดยชื่อ Block ทั้ง 6 Block มีดังนี้

1.) CON - Block นี้จะใช้สำหรับกำหนดค่าคงที่ให้กับตัวแปรที่จะใช้งาน ซึ่งตัวแปรที่ประกาศใน Block นี้สามารถเรียกใช้ได้ภายในไฟล์เดียวกัน ตัวอย่างเช่น

```
CON

        _clkmode        = xtal1 + pll16x
        _xinfreq        = 5_000_000
             tt         = 1.0
```

2.) VAR - Block นี้จะใช้สำหรับประกาศตัวแปรที่จะใช้ในการเขียนโปรแกรม โดยตัวแปรประกาศใน Block นี้สามารถเรียกใช้งานได้ภายใน Project File เดียวกัน ตัวอย่างเช่น

```
VAR
        long    testerror
        long    vlong
        word    vword
        byte    vbyte
```

3.) OBJ - Block นี้จะใช้สำหรับกำหนดชื่อไฟล์จากภายนอกเข้ามา เพื่อให้สามารถเรียกใช้ฟังก์ชันที่มีอยู่ในไฟล์จากภายนอกได้ (ไฟล์จากภายนอกที่ไม่ใช่ไฟล์ Library ของตัว Propeller เวลาจะเรียกใช้ฟังก์ชันที่อยู่ในไฟล์นั้นๆ ผู้ใช้จะต้อง Copy File นั้นมาไว้ที่เดียวกับไฟล์ Project ที่ผู้ใช้เขียนอยู่ด้วย ) การดูฟังก์ชันที่อยู่ภายใน ไฟล์ Library ของ Propeller ทำได้โดย ดูรูปที่ 6.1 ในช่องที่ Mouse ชี้อยู่ให้เลือก "Propeller Library" จากนั้นในหน้าต่างด้านล่าง ก็จะแสดงชื่อไฟล์ .spin ออกมาให้เห็นซึ่งไฟล์เหล่านี้ก็คือ File Library ของตัว Propeller ที่มี ไว้ให้เรียกใช้ เมื่อดับเบิลคลิกที่ File Library ไฟล์ก็จะถูกเปิดขึ้นมา และจะมีฟังก์ชั่นต่างๆถูกเขียนไว้ซึ่งฟังก์ชั่นเหล่านั้นผู้ใช้ สามารถเรียกมาใช้ในโปรแกรมที่ผู้ใช้เขียนอยู่ได้ ตัวอย่างเช่น

```
OBJ

    term    : "tv_terminal"
```

เวลาจะเรียกใช้งานฟังก์ชันที่อยู่ในไฟล์ Library " tv_terminal" ก็ให้แทนด้วย term.*ชื่อฟังก์ชัน* เช่น term.out(12) เป็นต้น

4.) PUB - สำหรับ Block นี้จะมีไว้สำหรับให้ผู้ใช้เขียนโปรแกรมที่ต้องการลงไป และสามารถเรียกใช้งานฟังก์ชันที่เขียนอยู่ใน block นี้ ได้ทั้งภายใน Project File เดียวกัน หรือต่าง Project File กันก็ได้ ในทุกๆ Project File ที่สร้างขึ้น จะต้องมี Block PUB อยู่อย่างน้อย 1 Block โดย Block PUB Block แรก ตัว Propeller จะถือเป็น Block main ของโปรแกรม ส่วน Block PUB ที่อยู่ต่อจาก Block PUB แรก จะถูกมองเป็นเพียงโปรแกรมย่อยมีไว้สำหรับให้โปรแกรมหลักเรียกใช้ โดยใน Block นี้ เมื่อพิมพ์คำว่า PUB แล้ว จะต้องตั้งชื่อให้กับ Block PUB ด้วยซึ่งชื่อนี้ก็จะ
เป็นเหมือนชื่อของFunction นั่นเอง ตัวอย่างเช่น

```
PUB stop

    vga.stop
```

เวลาจะประกาศตัวแปรใช้ภายใน PUB นั้นๆ ก็ให้ใส่ | ต่อจากชื่อ แล้วจึงตามด้วยชื่อตัวแปร ซึ่งตัวแปรที่ ประกาศนี้จะใช้งานได้เฉพาะภายใน PUB เท่านั้น และตัวแปรที่ประกาศจะมีขนาด 32 bit วลาจะรับค่าจากภายนอกเข้ามาใช้ยังฟังก์ชัน ก็ให้ใส่ **(ชื่อตัวแปร1,ตัวแปร2 )** ต่อจากชื่อ โดยภายในวงเล็บ

ก็ให้ใส่ชื่อตัวแปรที่ใช้รับการส่งผ่านค่าเข้ามาในฟังก์ชันด้วยเวลาจะส่งผ่านค่าจากฟังก์ชันออกไปยังภายนอก ก็ให้ใส่ : ต่อจากชื่อ แล้วตามด้วยชื่อตัวแปรที่จะส่งผ่านค่าออกไป ดังตัวอย่าง

```
PUB Ticks(Pin) : Microseconds | cnt1, cnt2

  outa[Pin]~
  dira[Pin]~~
  outa[Pin]~~
  outa[Pin]~
  dira[Pin]~

  waitpne(0, |< Pin, 0)
  cnt1 := cnt
  waitpeq(0, |< Pin, 0)
  cnt2 := cnt
  Microseconds := (||(cnt1 - cnt2) / (clkfreq / 1_000_000)) >> 1
```

ในตัวอย่างนี้ชื่อ PUB คือ Ticks ส่วน (Pin) เป็นการประกาศตัวแปรเพื่อใช้รับค่าจากภายนอกเข้ามาเก็บไว้ที่ตัวแปร Pin ส่วน : Microsecond เป็นการประกาศตัวแปรเพื่อใช้ส่งผ่านค่าออกจากฟังก์ชัน และสุดท้าย | cnt1,cnt2 เป็นการประกาศตัวแปรไว้ใช้งานภายใน Block PUB Ticks เวลาใช้งานจริงก็ประกาศในส่วนที่จะใช้งานก็พอ ไม่จำเป็นต้องประกาศจนครบทั้งหมดก็ได้

5.) PRI -สำหรับ Block นี้จะมีไว้สำหรับให้ผู้ใช้เขียนโปรแกรมที่ต้องการลงไปเหมือนกับ Block PUB แต่จะต่างตรงที่ Function ที่เขียนอยู่ใน Block นี้จะถูกมองเป็นโปรแกรมย่อยสำหรับให้เรียกใช้ภายใน Project File เดียวกันเท่านั้น ไม่สามารถเรียกใช้งานคนละ Project File ได้ ส่วนการประกาศใช้งานตัวแปรก็จะเหมือนกับ Block PUB

```
PRI bound(i, delta) : b | d

  d := bx_div[i]
  b := bx_min[i] + (bx_acc[i] := bx_acc[i] + delta
```

6.) DAT - สำหรับ Block นี้จะมีไว้สำหรับ กำหนด data table หรือ สำหรับ เขียน Assembly Code ตัวอย่างเช่น

```
DAT

'-------------------------------------
' Assembly language VGA driver
'-------------------------------------

                        org

entry                   mov     taskptr,#tasks

                        mov     x,#8
:init                   jmpret  taskret,taskptr
                        djnz    x,#:init


vgacolors               long    $C000C000
                        long    $C0C00000
                        long    $08A808A8
                        long    $0808A8A8
                        long    $50005000
                        long    $50500000
```

Block ทั้ง 6 นี้เวลาจะใช้งานผู้ใช้สามารถประกาศใช้เฉพาะ Block ที่จะใช้งานเท่านั้นก็ได้ไม่จำเป็นต้องประกาศทุก Block และในแต่ละ Block สามารถประกาศใช้ซ้ำกันได้มากกว่า 1 Block โดยใน Block PUB เวลาประกาศใช้ซ้ำกัน ชื่อของฟังก์ชันที่พิมพ์ต่อจาก PUB จะต้องไม่เหมือนกัน

8.2) ตัวอักษรที่พิมพ์ไม่ว่าจะเป็นการพิมพ์คำสั่ง หรือ การพิมพ์ชื่อตัวแปรก็ตาม สามารถพิมพ์โดยใช้ ตัวพิมพ์ใหญ่และตัวพิมพ์เล็กแทนกันได้ โดยตัว Propeller จะมองเป็นตัวแปรตัวเดียวกัน จะไม่มองแยกกันคนละตัวเหมือนภาษา C

8.3) เวลาแทนค่าเลข ฐาน10 เกิน 3 หลักจะต้องคั่นด้วย _ เสมอ เช่น x := 1_200

8.4) การใช้เครื่องหมายเท่ากับ ถ้าใช้ใน Block CON จะใช้ = ,ถ้าใช้ใน Block OBJ จะใช้: ,ถ้าใช้ใน Block PUB จะใช้ := ,

8.5) การใช้เลขฐานใน Propeller : $ นำหน้าค่าคงที่ เช่น $56 จะใช้แสดงถึงตัวเลขฐาน 16 ,% นำหน้าค่าคงที่(1และ0) เช่น %1001 จะแสดงถึงตัวเลขฐาน2 , %% นำหน้าค่าคงที่ (0,1,2,3) เช่น %%2130 จะมองเป็น Quaternary Number ก็คือให้มองเลข 1หลัก มีขนาด 2 บิต จากตัวอย่าง %%2130 = 10 01 11 00 (ฐาน2) = 9C (ฐาน16)

8.6) ในการใช้งานคำสั่งเกี่ยวกับ Loop เช่น repeat หรือ คำสั่งเงื่อนไข เช่น if หลังจากพิมพ์คำสั่งแล้ว ส่วนของโปรแกรมที่จะพิมพ์ในบรรทัดต่อมาเพื่อให้ทำงานภายใต้คำสั่ง repeat หรือ if จะต้องพิมพ์ให้เยื้องกับคำสั่งเหล่านี้มาทางขวามืออย่างน้อย 1 ช่องว่าง ถ้าพิมพ์ให้อยู่ในระดับเดียวกัน หรือ

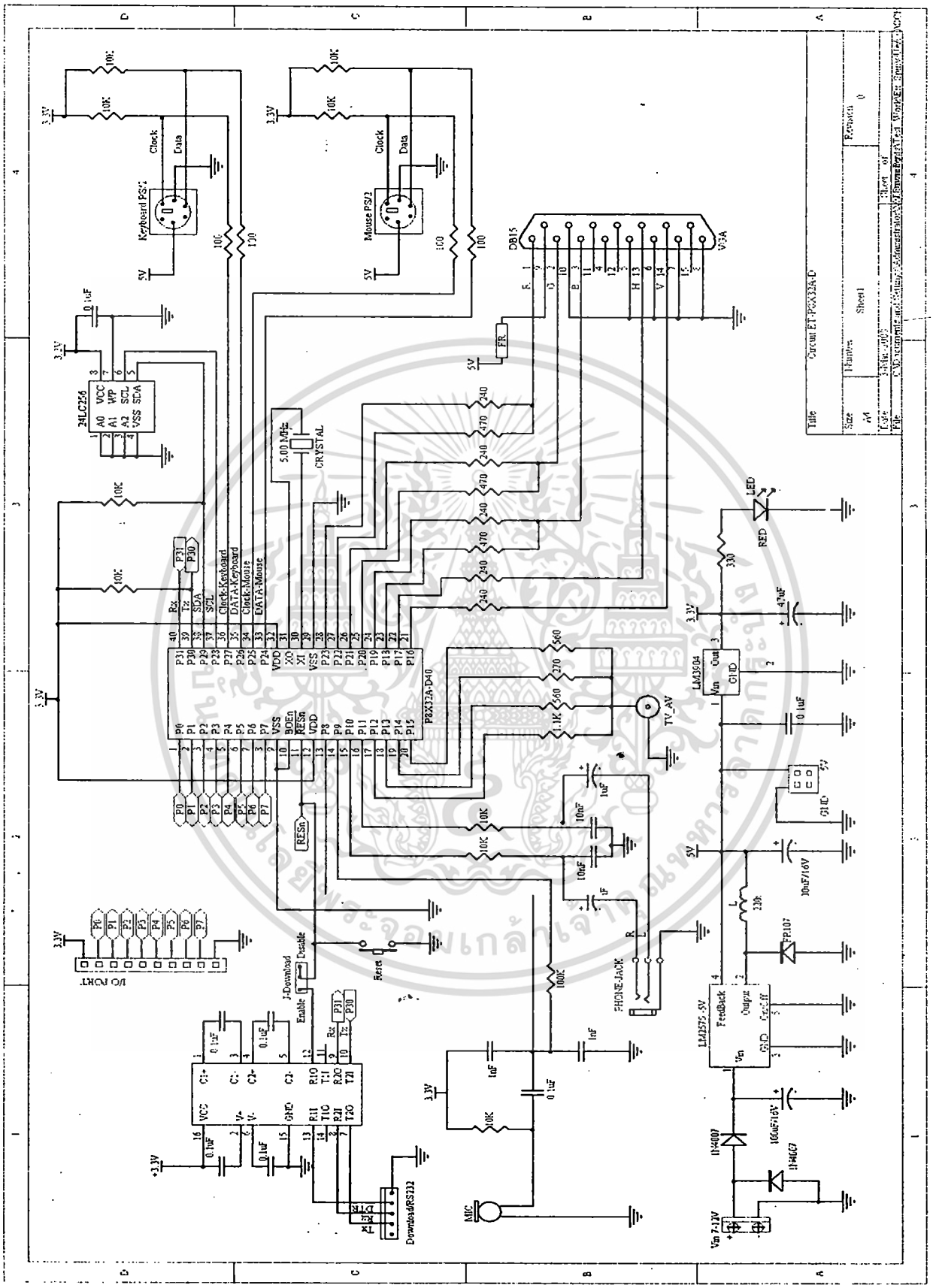
เยื้องไปทางด้านซ้ายมือ ตัว Propeller จะถือว่าโปรแกรมในบรรทัดนั้นอยู่ภายนอกคำสั่ง repeat หรือ if เป็นต้น ตัวอย่างเช่น

```
PRI BlinkingLED
  Pin := 07
  DirA[Pin] := Out
  Repeat
    OutA[Pin] := High           'LED ON
    WaitCnt(40_000_000 + Cnt)   'ONE-HALF SECOND WAIT
    OutA[Pin] := Low            'LED OFF
    WaitCnt(40_000_000 + Cnt)   'ONE-HALF SECOND WAIT
```

8.7) ในการเขียน Comment อธิบายโปรแกรมจะนำด้วยสัญลักษณ์ ' หรือ '' แล้วตามด้วย Comment หรือเขียนไว้ในเครื่องหมายปีกกา {comment} หรือ {{comment}}

8.8) ใน Block PUB ที่ใช้เป็น Block main ของโปรแกรม ถ้าโปรแกรมที่เขียนใน Block เป็นแบบ Loop เปิด ไม่มีการทำงานวนซ้ำภายใน Loop ใดๆ ก็ควรจะจบโปรแกรมด้วยคำสั่ง Repeat เพื่อไม่ให้ MCU วนกลับไป Reset เริ่มโปรแกรมใหม่

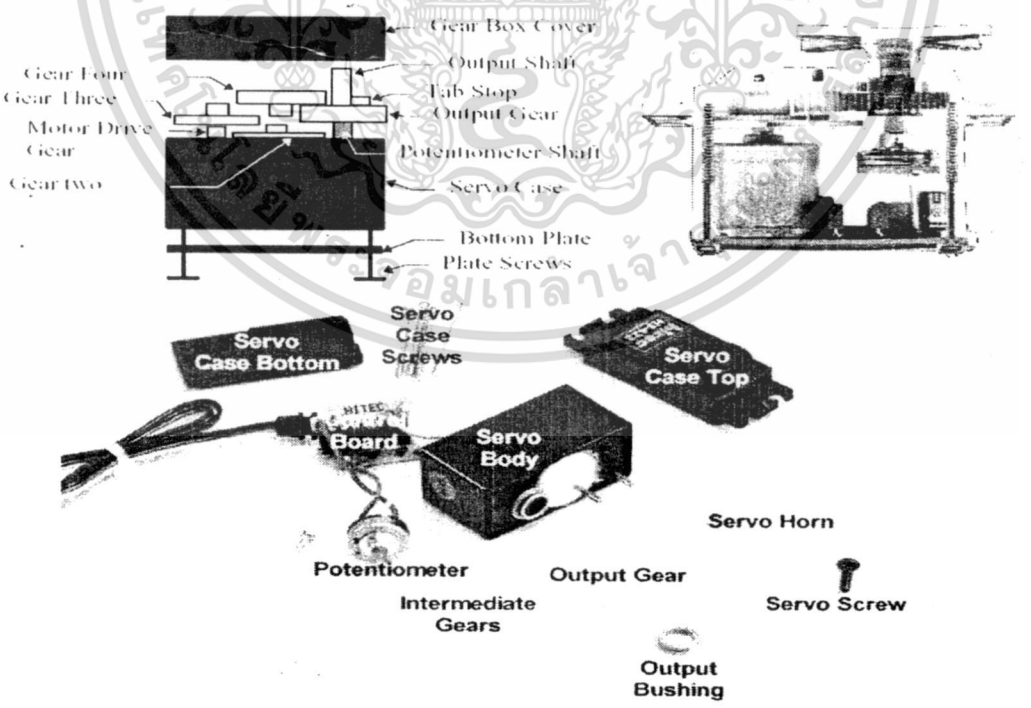
สำหรับรายละเอียดมากกว่านี้ในการใช้งานคำสั่งต่างๆของ ภาษา SPIN ให้ดูได้ที่ Help ของโปรแกรม Propeller โดยเข้าไปที่เมนู Help แล้วเลือกที่ Propeller Manual(pdf)



**รูปที่ 2.12 วงจร ET-BASE PX32 V1.0**

**9. Servo motor**

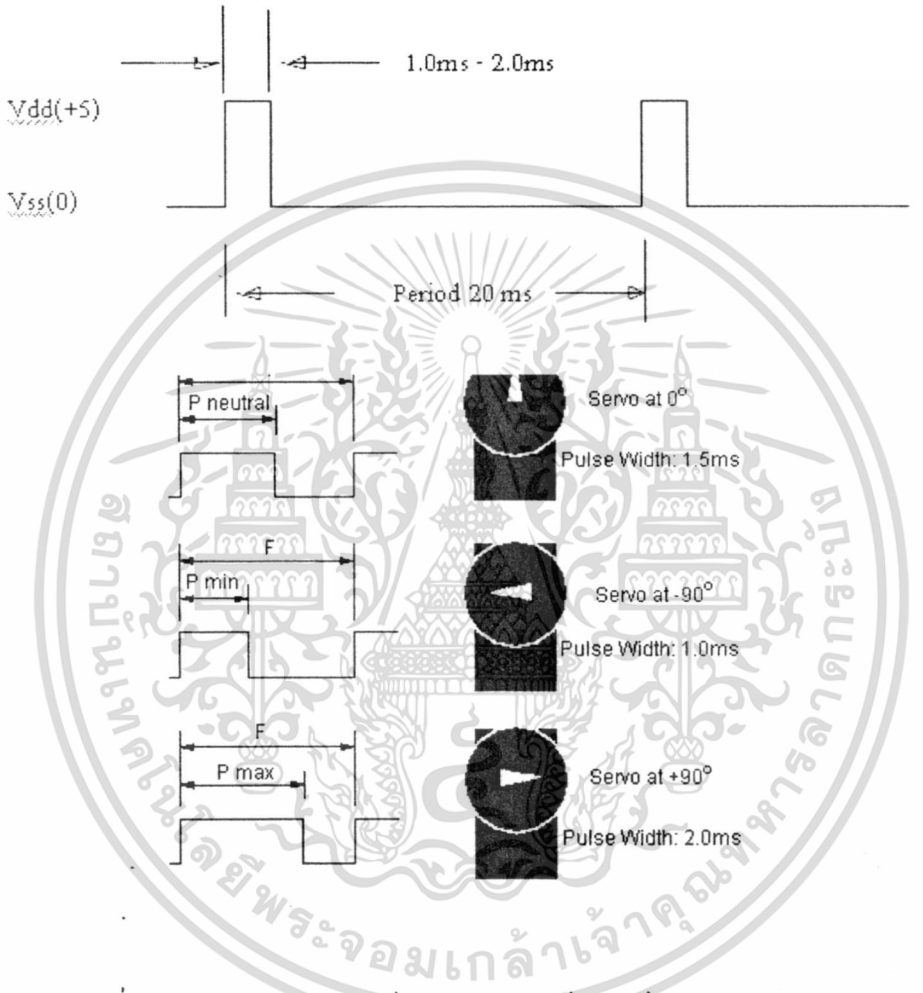
Servo motor คือ มอเตอร์ไฟฟ้ากระแสตรง (DC motor) ที่ถูกประกอบรวมกับ ชุดเกียร์ และส่วนควบคุม ต่างๆ ไว้ในโมดูลเดียวกัน หรือ ภายในกล่องพาสติกเดียวกัน โดยมอเตอร์ชนิดนี้จะมีสายต่อใช้งานเพียง 3 เส้นเท่านั้น คือ VCC,GNDและ สายสัญญาณควบคุม(Control Line) ซึ่งสามารถควบคุมให้มอเตอร์หมุนซ้าย หรือ ขวาได้จากสายสัญญาณเพียงเส้นเดียวโดยสัญญาณที่ใช้ควบคุมนี้จะเป็นสัญญาณ พัลส์วิดมอด (PWM) แบบ TTL Level ระดับแรงดันที่จ่ายให้มอเตอร์นี้จะอยู่ในช่วงประมาณ 4 ถึง 6 โวลท์ ขึ้นอยู่กับคุณสมบัติของมอเตอร์แต่ละตัวข้อดีของมอเตอร์ชนิดนี้ก็คือ จะมีขนาดเล็กน้ำหนักเบา,ให้แรงบิตสูง ,กินพลังงานน้อยและสามารถควบคุมด้วยแรงดันลอจิกที่เป็น TTL ได้โดยตรงไม่จำเป็นต้องต่อวงจรขับ(Driver) อื่นๆ เพราะ มอเตอร์ชนิดนี้จะมีวงจรควบคุมบรรจุไว้ภายในอยู่แล้ว ซึ่งมอเตอร์ชนิดนี้สามารถควบคุมให้หมุนไปในตำแหน่ง หรือ ทิศทางองศาที่ต้องการได้ โดยอาศัยสัญญาณความกว้างพัลส์ ที่ป้อนให้มอเตอร์แต่เซอร์โวมอเตอร์นี้จะหมุนได้แค่เพียงในช่วงประมาณ 180° หรือครึ่งรอบเท่านั้นหรือบางรุ่นอาจหมุนได้ถึง 210° แต่จะไม่สามารถหมุนเป็นวงรอบได้เนื่องจากโครงสร้างภายในจะประกอบด้วยตัวต้านทานชนิดปรับค่าได้ (VR)ที่ทำหน้าที่ตรวจสอบตำแหน่งการหมุนของมอเตอร์และตัวต้านทานนี้จะถูกยึดติดกับแกนหมุนของมอเตอร์ซึ่งจากการที่ตัวต้านทานปรับค่านี้ไม่สามารถหมุนเป็นวงรอบได้ ดังนั้น เซอร์โวมอเตอร์จึงถูกออกแบบให้หมุนได้เพียงแค่ประมาณ 180° หรือ ครึ่งรอบเท่านั้น เพื่อป้องกันความเสียหายที่จะเกิดกับตัวต้านทานปรับค่าได้แต่ถ้าหากเราต้องการให้มอเตอร์หมุนเป็นวงรอบ (360°) นั้นก็สามารถทำได้ โดยจะต้องทำการปรับแต่ง (Modify) ดัดแปลงชิ้นส่วนบางอย่างของมอเตอร์ ซึ่งวิธีการต่างๆ จะได้กล่าวไว้ในภายหลัง




รูปที่2.13 ส่วนประกอบต่างๆของ Servo Motor

**หลักการทำงานของ Servo motor**

การควบคุมการทำงานของ เซอร์โวมอเตอร์ ทำได้โดย การป้อนสัญญาณความกว้างพัลส์ ให้กับมอเตอร์ซึ่งตำแหน่งและทิศทางการหมุนของมอเตอร์นี้จะขึ้นอยู่กับขนาดของความกว้างของพัลส์นั้น โดยทั่วไปแล้วความกว้างของสัญญาณพัลส์จะมีจุดให้อ้างอิง 3 จุด ดังรูป คือ




รูปที่ 2.14 แสดงความสัมพันธ์ระหว่างความกว้างพัลส์และมุมของServo

**- สัญญาณความกว้างพัลส์ขนาด 1.5 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม0 องศา หรือจุดกึ่งกลางของมอเตอร์**

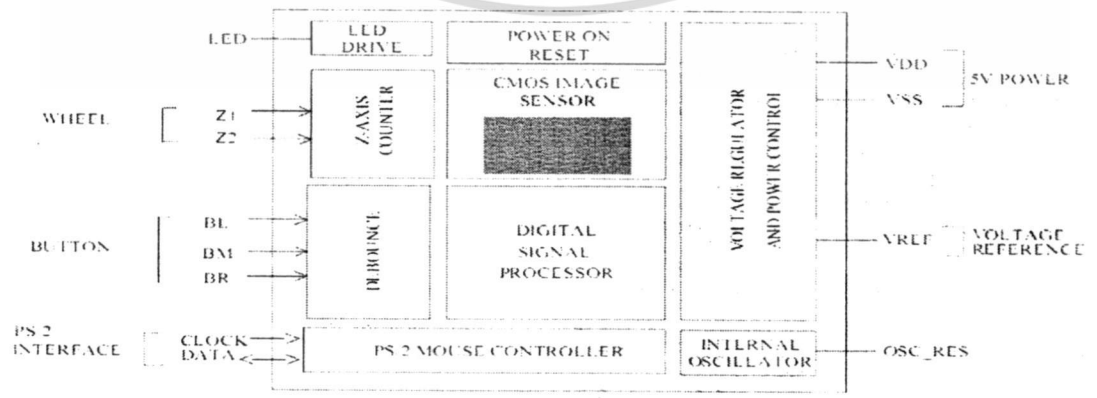
**- สัญญาณความกว้างพัลส์ขนาด 1 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม -90 องศา หรือในทิศทางทวนเข็มนาฬิกา**

**- สัญญาณความกว้างพัลส์ขนาด 2 ms จะควบคุมให้เซอร์โวมอเตอร์หมุนไปอยู่ที่ตำแหน่งมุม +90 องศา หรือในทิศทางตามเข็มนาฬิกา**

ส่วนการที่จะควบคุมให้มอเตอร์หมุนเป็นมุมอื่นๆนั้นก็สามารถทำได้โดยการป้อนสัญญาณพัลส์เป็นระดับความกว้างต่างๆ โดยอ้างอิงจากจุด ทั้ง 3 จุดที่กล่าวมานี้ ตัวอย่างเช่นถ้าต้องการให้มอเตอร์หมุนไปที่มุม - 45 องศา เราก็จะต้องป้อนสัญญาณพัลส์ที่มีความกว้าง 1.25 ms เป็นต้นและสัญญาณพัลส์นี้จะต้องจ่ายให้มอเตอร์ทุกๆ 20 ms (Period) เพื่อรักษาสภาพตำแหน่งของมอเตอร์ไว้โดยหลักการก็คือจะอาศัยการเปรียบเทียบช่วงเวลาของความกว้างพัลส์ที่จ่ายให้กับมอเตอร์ทางขาสัญญาณควบคุมกับค่าเวลาของวงจร RC ภายในบอร์ดควบคุมในตัวของมอเตอร์ ซึ่งค่าเวลาของวงจร RC นี้จะมีการเปลี่ยนแปลงตามการหมุนของมอเตอร์เนื่องจากตัวต้านทานปรับค่าจะถูกยึดติดอยู่กับแกนหมุนของมอเตอร์ซึ่งการหมุนของมอเตอร์จะทำให้ค่าความต้านทานของตัวต้านทานปรับค่า (VR) เปลี่ยนแปลงไปเป็นผลทำให้ค่าเวลาของวงจร RC เปลี่ยนแปลงตามไปด้วยโดยในขณะที่เราป้อนสัญญาณความกว้างพัลส์ให้กับมอเตอร์ทางขาสัญญาณควบคุมสัญญาณนี้จะถูกนำไปเปรียบเทียบกับค่าเวลาของวงจร RC หากค่าทั้ง 2 ไม่เท่ากันมอเตอร์ก็จะหมุนทำให้ค่าเวลาของวงจร RC เปลี่ยนแปลงจนกระทั่งค่าเวลาความกว้างพัลส์ของวงจร RC เปลี่ยนแปลงจนเท่ากับสัญญาณพัลส์ทางขาควบคุม (Control line) มอเตอร์จึงจะหยุดหมุน

**10. PAN 3400 PS/2 Optical Mouse Single Chip**

เป็น Sensor ที่มีขนาดเล็กเท่ากับ 11.68 x 10.16 mm มีความสามารถตรวจจับพื้นที่ผิวสัมผัสใน 2มิติ ทั้งแนวแกน x และ แกน y มีความเร็วในการตรวจจับ 600 ครั้งต่อตารางนิ้ว และอัตราเคลื่อนไหวเท่ากับ 28 นิ้ว/วินาที (inches per second :ips) ภายใน sensor จะมีตัวที่ช่วยในการประมวลผลภาพ คือ Image Acquisition System (IAS) และ Digital Signal Processor (DSP) โดยเมื่อ sensor ทำการตรวจจับพื้นที่จะได้ input ที่เป็น image แล้วจะส่งต่อให้กับ Image Acquisition System (IAS) เพื่อทำการเตรียม imageให้พร้อมสำหรับการประมวลผลต่อไปโดยจะส่งต่อให้กับ Digital Signal Processor (DSP) เพื่อประมวลผลภาพโดย DSP จะทำการพิจารณาทั้งทิศทางและระยะทางที่ sensor ตรวจจับได้




**รูปที่** 2.15 Block Diagram PAN3400

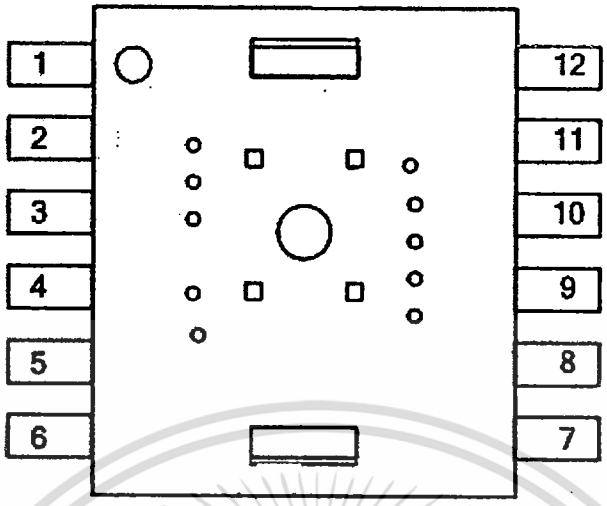
1) คุณสมบัติของ PAN 3400 PS/2 Optical Mouse Single Chip

- Single 5.0 volt power supply
- Compatible with Microsoft® Intelli 3D PS/2 and IBM® PS/2 mouse
- Precise optical motion estimation technology
- Complete 2-D motion sensor
- No mechanical parts
- Accurate motion estimation over a wide range of surfaces
- High speed motion detection up to 28 inches/sec
- Resolution is 600 CPI
- Power saving mode during times of no movement
- Support three buttons (R, M, L) and three axes (X, Y, Z)
- Z axis support mechanical input
- Internal ±10% accurate oscillator, external crystal-less

PAN 3400 Sensor มีขนาด 12 ขา โดยแต่ละขาจะมีลักษณะ Dual Inline Package (DIP) โดยแต่ละขาจะรับสัญญาณต่างกันไปดังตารางที่ 2.3

**ตารางที่ 2.3** Pin Description

| Pin # | Name | Type | Definition |
|---|---|---|---|
| 1 | BL | IN | Button left key input, internal pull-up 50K ohm, press connect to low |
| 2 | LED | OUT | LED control |
| 3 | OSC_RES | IN | Connect to resistor input |
| 4 | VSS | GND | Chip ground |
| 5 | VDD5V | PWR | Chip power VDD, 5.0V |
| 6 | VREF | BYPASS | Analog voltage reference |
| 7 | CLOCK | I/O | PS/2 mouse clock line |
| 8 | DATA | I/O | PS/2 mouse data line |
| 9 | BR | IN | Button right key input, internal pull-up 50K ohm, press connect to low |
| 10 | BM | IN | Button middle key input, internal pull-up 50K ohm, press connect to low |
| 11 | Z2 | IN | Z axis, support mechanical scroller input, internal pull-down 50K ohm |
| 12 | Z1 | IN | Z axis, support mechanical scroller input, internal pull-down 50K ohm |




รูปที่ 2.16 PAN3400 Sensor

ตารางที่ 2.4 Data Format

| Byte | Bit | Symbol | Description |
|---|---|---|---|
| 1 | 0 | BL | 1 = Left button pressed |
| | 1 | BR | 1 = Right button pressed |
| | 2 | BM | Always = 0, reserved for middle button |
| | 3 | 1 | Always = 1, reserved for future use |
| | 4 | Xs | X data sign, 1 = negative |
| | 5 | Ys | Y data sign, 1 = negative |
| | 6 | $X_{ov}$ | X data overflow, 1 = overflow |
| | 7 | $Y_{ov}$ | Y data overflow, 1 = overflow |
| 2 | 0-7 | X0 ~ X7 | X data (D0 - D7). A positive value indicates motion to the right; a negative value indicates motion to the left. Bit 0=LSB. |
| 3 | 0-7 | Y0 ~ Y7 | Y data (D0 - D7), A positive value indicates device motion upward; a negative value indicates motion downward. Bit 0 = LSB. |

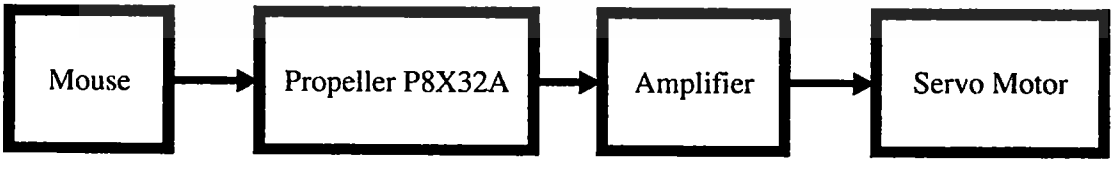
# บทที่ 3

# การออกแบบและวงจรที่ออกแบบ

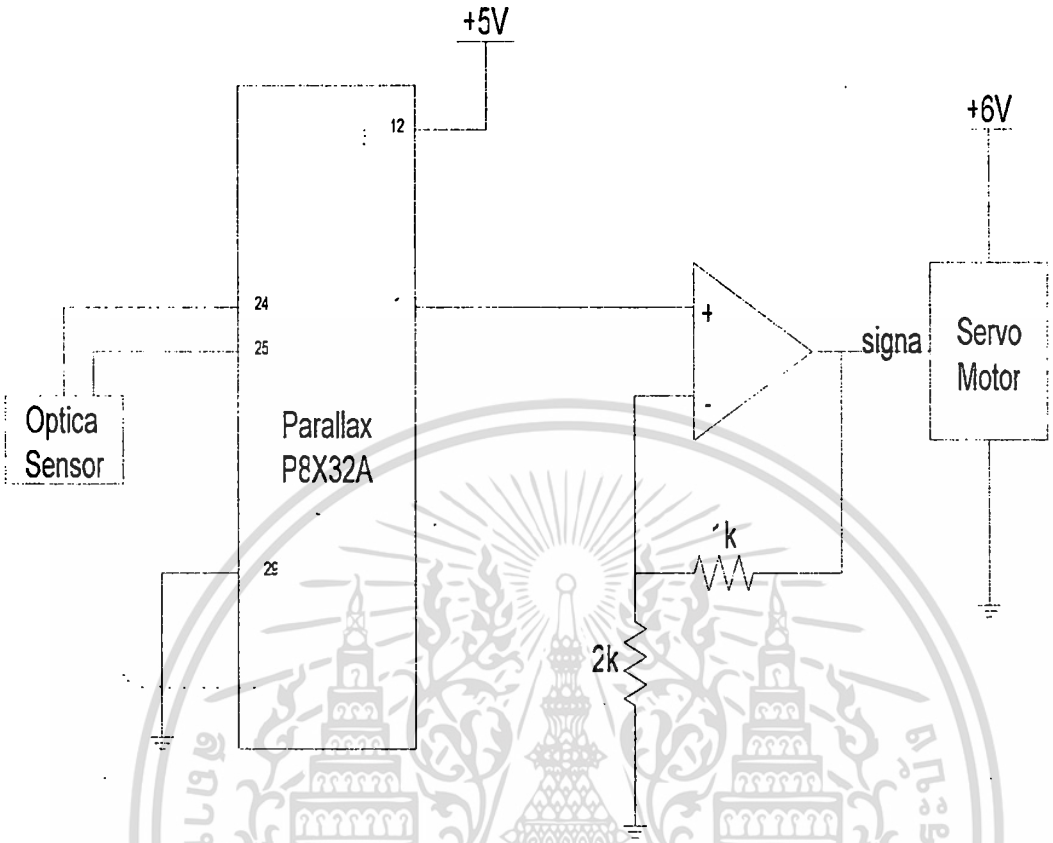
**3.1) การออกแบบวงจร**

ในการออกแบบ เครื่องควบคุมแถบผ้าโดยใช้การรับค่าทางเมาส์ซึ่งเปรียบเสมือนเป็นSensor ที่สามารถจะตรวจจับการเคลื่อนที่ของแถบผ้าว่าเอียงไปทางใด แล้วส่งค่าที่ได้ออกไปให้กับ P8X32A – D40 Microcontroller เพื่อทำการประมวลผลข้อมูลแล้วนำค่าที่ได้นั้นไปสั่งให้ Servo Motor ทำงานตามการเลื่อนของเมาส์ เพื่อให้แถบผ้านั้นไม่เอียงไปทามใดทางหนึ่ง ซึ่งสามารถอธิบายดังได้นี้

1. การนำเมาส์มาใช้ในการรับค่าเสมือนว่าเป็น Sensor ตัวหนึ่ง มาเป็นตัวช่วยในการรับค่าการเปลี่ยนแปลงตำแหน่งของแถบผ้าที่เคลื่อนที่ในแนวนอนหรือแนวแกน x เท่านั้น แล้วส่งค่าการเคลื่อนที่ของแถบผ้านั้นออกไปให้กับ P8X32A – D40 Microcontroller เพื่อประมวลผลข้อมูล
2. เมื่อเรารับค่าที่ได้จากเมาส์ผ่านทางพอร์ต PS2 ส่งค่าการเปลี่ยนตำแหน่งของแถบผ้าให้กับ P8X32A – D40 Microcontroller แล้วทำการประมวลผลข้อมูลให้ข้อมูลที่ได้นั้นสามารถสร้างสัญญาณพัลส์สั่งให้ Servo Motor ทำงานเพื่อให้แถบผ้าไม่เกิดการเอียงไปทางใดทางหนึ่ง โดยสัญญาณที่ส่งออกไปให้ Servo Motor ทำงานผ่านทาง
   P1 ของ P8X32A – D40 Microcontroller
3. Servo Motor จะทำงานเมื่อได้รับพัลส์เข้าใช้ในการสั่งการมอเตอร์ให้หมุนในทิศทางและมุมที่ต้องการมาซึ่งในการออกแบบนี้เรากำหนดค่าพัลส์ระหว่าง 0.7ms -2.3ms โดยที่
   a. เมื่อพัลส์มีค่า 0.7ms Servo Motor จะหมุนไปที่ 180 องศา
   b. เมื่อพัลส์มีค่า 1.5ms Servo Motor จะหมุนไปที่ 90 องศา
   c. เมื่อพัลส์มีค่า 2.3ms Servo Motor จะหมุนไปที่ 0 องศา




รูปที่ 3.1 Block diagram การทำงาน




รูปที่ 3.2 วงจรการทำงานทั้งหมด

จากรูป วงจรการทำงานจะเริ่มต้นด้วยการที่ Optical sensor ตรวจจับการเปลี่ยนแปลงของแถบผ้าที่เคลื่อนไปจากจุดเริ่มต้นในแนวแกน X เมื่อ Sensor ตรวจจับการเคลื่อนที่จากการใช้ LED ส่องไปที่พื้นผิวแถบผ้า จากนั้นจะถูกส่งต่อไปยังส่วนประมวลผลภาพ เพื่อจะแปลงไปเป็นการเคลื่อนไหวในแนวแกน X แล้วจึงส่ง ข้อมูลที่ได้นี้ไปยังส่วนของ Microcontroller ซึ่งทำหน้าที่ในการประมวลผลข้อมูลจาก Sensor mouse เพื่อนำมาใช้ในการกำหนดค่าของ pulse ในการควบคุม Servo motor แต่เนื่องจาก microcontroller Parallax P8X32A สร้างสัญญาณได้มี Amplitude สูงสุดเพียง 3.3 volt ซึ่งไม่เพียงพอต่อการนำไปป้อนสัญญาณเพื่อที่จะสั่งงาน Servo motor ได้ เนื่องจาก Servo motor สามารถทำงานได้ที่สัญญาณ Pulse มีขนาด 5 volt ดังนั้น จึงต้องทำการขยายขนาดของสัญญาณโดยการใช้วงจร Non-inverting Amplifier มาช่วยในการขยายสัญญาณให้มีขนาด 5V เพื่อที่จะนำไปสั่งงาน Servo motor ได้

# บทที่ 4
# การทดลองเครื่องควบคุมแถบผ้า

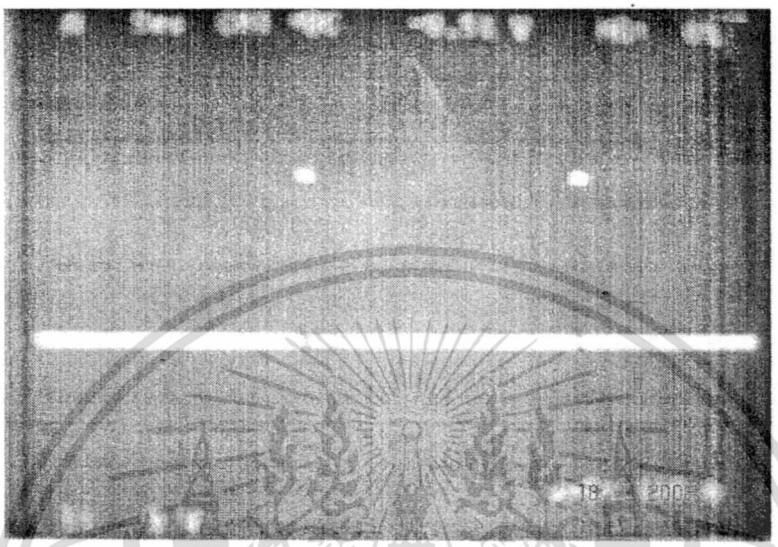
**วิธีการทดลอง**

**ตอนที่ 1** การทดสอบการหมุน Servo Motor ที่คาบเวลาต่างๆ

**ตารางที่ 4.1** การทดสอบ Servo Motor

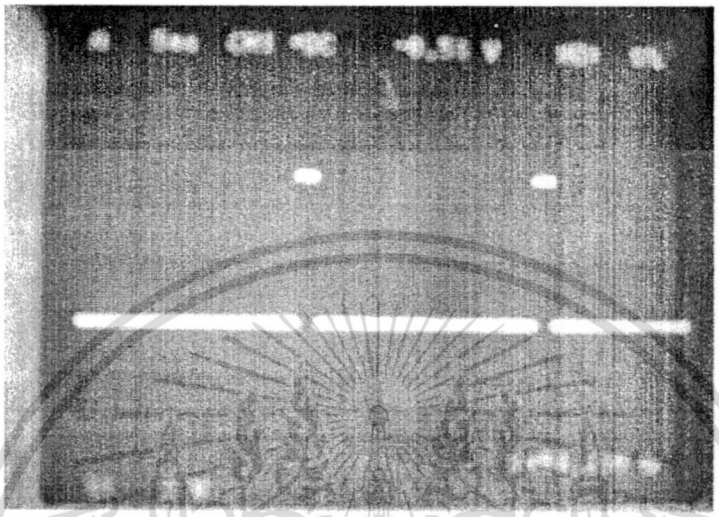
| คาบเวลา(μS) | มุมในการแกว่งของ Servo Motor |
|---|---|
| 700 | 0° |
| 800 | 10° |
| 900 | 22° |
| 1000 | 34° |
| 1100 | 45° |
| 1200 | 56° |
| 1300 | 68° |
| 1400 | 80° |
| 1500 | 90° |
| 1600 | 101° |
| 1700 | 112° |
| 1800 | 124° |
| 1900 | 135° |
| 2000 | 145° |
| 2100 | 157° |
| 2200 | 168° |
| 2300 | 180° |

**ตอนที่ 2** การวัดสัญญาณพัลส์โดยเครื่องออสซิโลสโคป

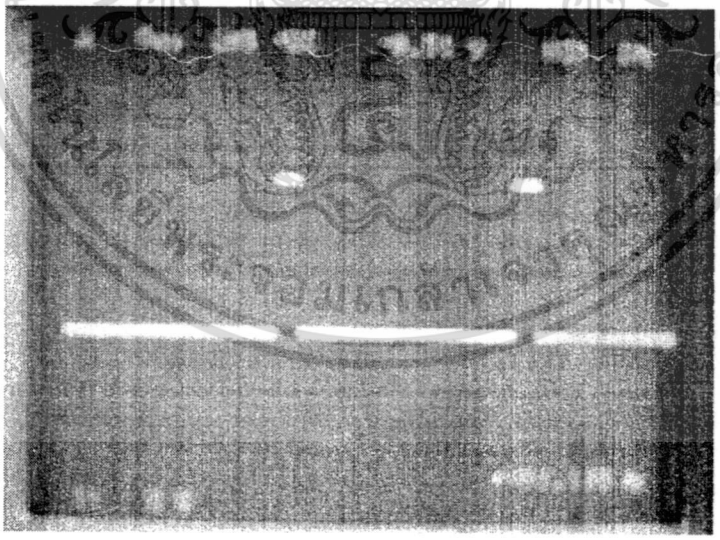


รูปที่ 4.1 สัญญาณพัลส์ขนาด 700 μS

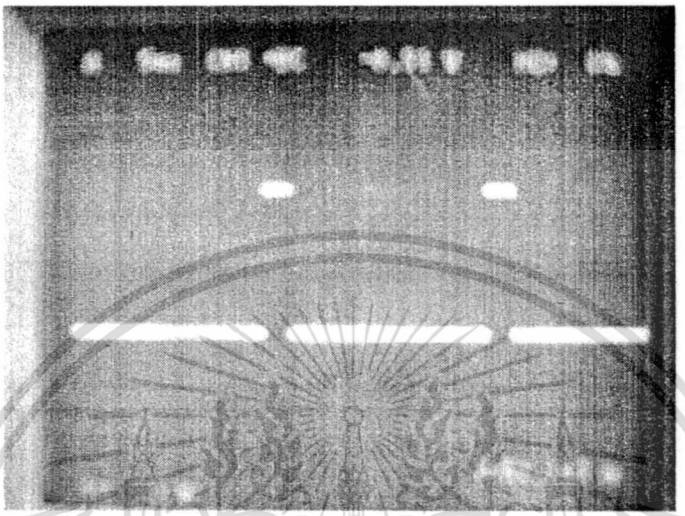
รูปที่ 4.2 สัญญาณพัลส์ขนาด 1100 μS



รูปที่ 4.3 สัญญาณพัลส์ขนาด 1500 μS



รูปที่ 4.4 สัญญาณพัลส์ขนาด 1900 μS



รูปที่ 4.5 สัญญาณพัลส์ขนาด 2300 µs

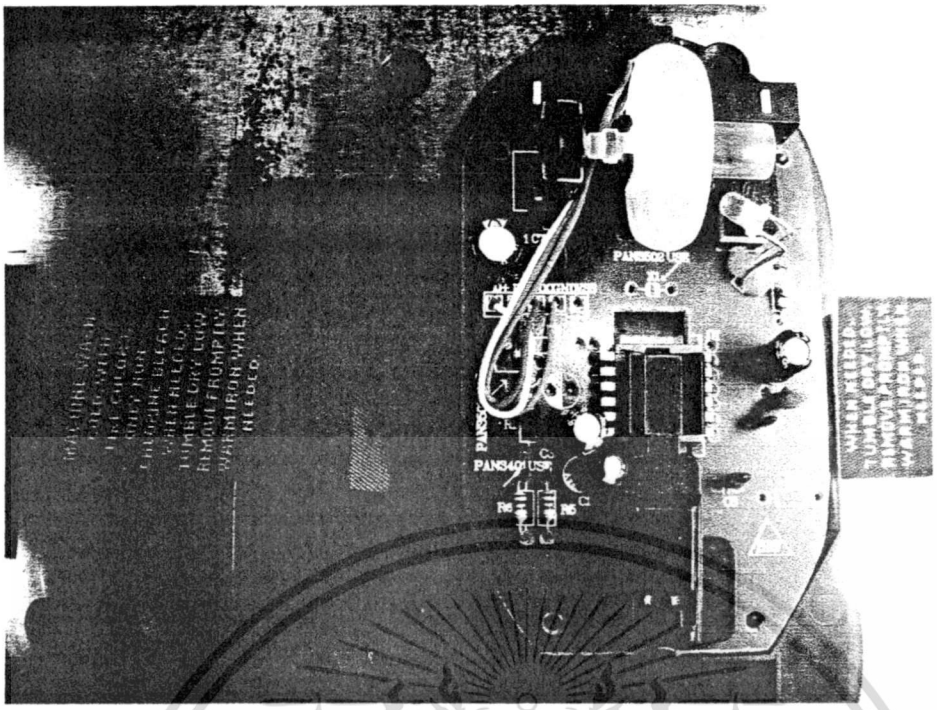
**หมายเหตุ** สัญญาณทั้งหมดวัดโดยใช้ Volt/Div = 2 V และ Time/Div = 5 mS

ตารางที่4.2 ความสัมพันธ์ระหว่างระยะที่แถบผ้าเคลื่อนที่ในแนวแกน X กับความกว้างของ Pulse

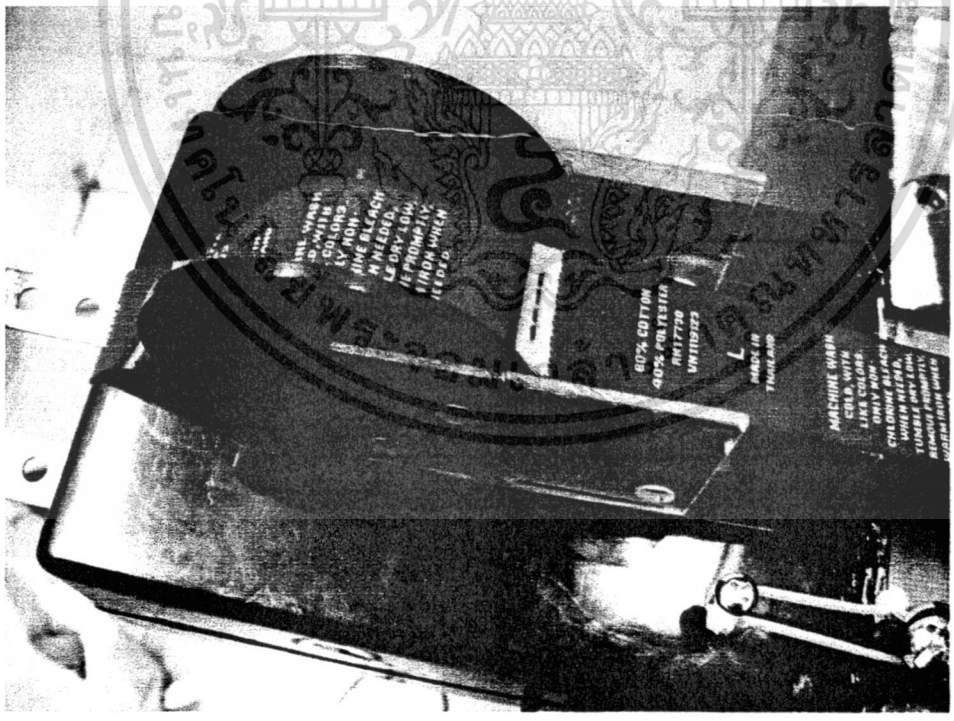
| องศาของการหมุน | ระยะเคลื่อนที่ของแถบผ้า |
|---|---|
| 0 | 0 |
| -10 | 2 mm |
| -20 | 4 mm |
| -30 | 6 mm |
| -40 | 8 mm |
| -50 | 1.0 cm |
| 10 | -2 mm |
| 20 | -4 mm |
| 30 | -6 mm |
| 40 | -8 mm |
| 50 | -1.0 cm |

ตารางที่ 4.3 ความสัมพันธ์ระหว่างการหมุนของมอร์เตอร์กับระยะที่แถบผ้าเคลื่อนที่แนวแกน X

| ระยะเคลื่อนที่ของแถบผ้า (cm) | ความกว้างของ Pulse(ms) |
|---|---|
| -2.0 | 2.5 |
| -1.5 | 2.25 |
| -1.0 | 2.0 |
| -0.5 | 1.75 |
| 0 | 1.5 |
| 0.5 | 1.25 |
| 1.0 | 1.0 |
| 1.5 | 0.75 |
| 2.0 | 0.5 |



รูปที่ 4.6 เซนเซอร์แสง



รูปที่ 4.7 กล่องไมโครคอนโทรลเลอร์และเซอร์โวมอเตอร์

# บทที่ 5
# สรุปผลการทดลอง

จากการศึกษาการทำงานของ Servo motor พบว่าในการควบคุม Servo Motor นั้นเราต้องใช้ PWM (Pulse Width Modulation) เป็นสิ่งที่ควบคุมการทำงานของมอเตอร์ซึ่งค่าของมุมที่เปลี่ยนไปของ Servo motor นั้นขึ้นอยู่กับการกำหนดพัลส์ในการทำงานโดยจากการศึกษานี้พบว่า Servo motor รุ่น S03T/STD/JR นั้นมีช่วงการกำหนดการทำงานของพัลส์ในช่วง 0.7ms-2.3ms โดยที่เมื่อพัลส์มีค่า 0.7ms Servo Motor จะหมุนไปที่ 180 องศาและเมื่อพัลส์มีค่า 2.3ms Servo Motor จะหมุนไปที่ 0 องศาและจากการทดลองวัดสัญญาณพัลส์ที่ได้โดยการใช้ออสซิลโลสโคปจะเห็นได้ว่าสัญญาณเมื่อกำหนดพัลส์ที่ 0.7ms จะมีความกว้างในช่วงที่เป็น +5v น้อยที่สุดและเมื่อกำหนดพัลส์ที่ 2.3ms จะมีความกว้างในช่วงที่เป็น +5v มากที่สุด

จากการทดลองการเคลื่อนที่แถบผ้าไปเพื่อหาความสัมพันธ์ระหว่างระยะการเคลื่อนที่ของแถบผ้ากับการหมุนของ Servo Motor พบว่าเมื่อแถบผ้าเคลื่อนที่ไปทางซ้าย Servo Motor จะหมุนไปทางขวาและเมื่อผ้าเคลื่อนที่ไปทางขวา Servo Motor จะหมุนไปทางซ้าย จะเห็นได้ว่าเมื่อผ้าเคลื่อนที่ไปทุกๆ 2 mm มุมในการหมุนของ Servo Motor จะเปลี่ยนไปครั้งละ 10 องศา

จากการทดลองการเคลื่อนที่แถบผ้าไปเพื่อหาความสัมพันธ์ระหว่างระยะของแถบผ้ากับความกว้างของสัญญาณพัลส์พบว่า เมื่อแถบผ้าเคลื่อนที่ไปทางซ้ายสัญญาณพัลส์จะมีความกว้างเพิ่มขึ้นจากสัญญาณเริ่มต้น โดยสัญญาณเริ่มต้นมีขนาดเท่ากับ 1. 5 ms เพื่อให้ Servo Motor หมุนไปทางขวาเพื่อขยับแถบผ้าให้เลื่อนกลับมา ในทำนองเดียวกันเมื่อแถบผ้าเคลื่อนที่ไปทางขวาสัญญาณพัลส์จะมีความกว้างลดลงจากสัญญาณเริ่มต้น เท่ากับ 1. 5 ms เพื่อให้ Servo Motor หมุนไปทางซ้ายเพื่อขยับแถบผ้าให้เลื่อนกลับมา

จากการที่เราได้ศึกษาในขั้นต้นแล้วจึงมีการทดลองการทำงานโดยการป้อนพัลส์ที่ค่าต่างๆ และทำการสังเกตุวัดมุมของ Servo motor ซึ่งจากการทดลองเราพบว่าค่าของมุมที่ได้นั้นค่องข้างมีความแม่นยำสูงเราจึงสามารถสรุปได้ว่า Servo motor มีความแม่นยำสูงในการทำงาน แต่จากการทดลองยังมีความผิดพลาดเล็กน้อยอันเนื่องมาจาก ในการวัดมุมนั้นอาจมีความแม่นยำที่ไม่เพียงพอเนื่องจากขาดอุปกรณ์ในการวัดที่ดี

# ภาคผนวก

## โปรแกรมภาษา Spin สำหรับติดต่อกับเซนเซอร์แสง

```
CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000                    'Note Clock Speed for your setup!
  PanServo = 1

VAR
  long  mousex,temp

OBJ
  mouse : "Mouse"
  SERVO : "Servo32v32"

PUB Start
        'Preset ALL servo's to center position
    SERVO.Set(PanServo,1500)
  mouse.start(24,25)
  mousex := 1500

  SERVO.Start

  repeat
  temp := mousex + mouse.abs_y*2

  waitcnt (cnt+125_000)
  SERVO.Set(PanServo,temp)
```

## โปรแกรมภาษา Assembly สำหรับติดต่อกับเซอร์โวมอเตอร์

```
CON
  _1uS = 1_000_000 /      1                 'Divisor for 1 uS
  ZonePeriod = 5_000                        '5mS (1/4th of typical servo period of 20mS)
  NoGlitchWindow = 2_500                    '2.5mS Glitch prevention window (set value larger than maximum servo width of 2mS)
VAR
    long    ZoneClocks
    long    NoGlitch
    long    ServoPinDirection
    long    ServoData[31]                   '0-31 Servo Pulse Width information
PUB Start
  ZoneClocks := (clkfreq / _1uS * ZonePeriod)           'calculate # of clocks per ZonePeriod
  NoGlitch  := $FFFF_FFFF-(clkfreq / _1uS * NoGlitchWindow)     'calculate # of clocks for GlitchFree servos. Problem occurs when 'cnt' value rollover is
  cognew(@ServoStart,@ZoneClocks)
```

```
PUB Set(Pin, Width)                                 'Set Servo value
    Width := 500 #> Width <# 3000                       'limit Width value between 1000uS and 2000uS
     Pin :=   0 #>  Pin <# 31                           'limit Pin value between 0 and 31
    ServoData[Pin] := (clkfreq / _1uS * Width)          :    'calculate # of clocks for a specific Pulse Width
    dira[Pin] := 1                                  'set selected servo pin as an OUTPUT
    ServoPinDirection := dira                           'Read I/O state of ALL pins
DAT
            org
ServoStart      mov   Index,          par           'Set Index Pointer
            rdlong _ZoneClocks,       Index         'Get ZoneClock value
            add   Index,          #4                'Increment Index to next Pointer
            rdlong _NoGlitch,         Index         'Get NoGlitch value
            add   Index,          #4                'Increment Index to next Pointer
            rdlong _ServoPinDirection,  Index       'Get I/O pin directions
            add   Index,          #32               'Increment Index to END of Zone1 Pointer
            mov   Zone1Index,         Index         'Set Index Pointer for Zone1
            add   Index,          #32               'Increment Index to END of Zone2 Pointer
            mov   Zone2Index,         Index         'Set Index Pointer for Zone2
            add   Index,          #32               'Increment Index to END of Zone3 Pointer
            mov   Zone3Index,         Index         'Set Index Pointer for Zone3
            add   Index,          #32               'Increment Index to END of Zone4 Pointer
            mov   Zone4Index,         Index         'Set Index Pointer for Zone4
            mov   dira,           _ServoPinDirection  'Set I/O directions
Zone1       mov   ZoneIndex,      Zone1Index        'Set Index Pointer for Zone1
            call  #ResetZone
            call  #ZoneCore
Zone2       mov   ZoneIndex,      Zone2Index        'Set Index Pointer for Zone2
            call  #IncrementZone
            call  #ZoneCore
Zone3       mov   ZoneIndex,      Zone3Index        'Set Index Pointer for Zone3
            call  #IncrementZone
            call  #ZoneCore
Zone4       mov   ZoneIndex,      Zone4Index        'Set Index Pointer for Zone4
            call  #IncrementZone
            call  #ZoneCore
            jmp   #Zone1
ResetZone       mov   ZoneShift1,     #1
            mov   ZoneShift2,     #2
            mov   ZoneShift3,     #4
            mov   ZoneShift4,     #8
            mov   ZoneShift5,     #16
            mov   ZoneShift6,     #32
```

```
            mov   ZoneShift7,      #64
            mov   ZoneShift8,      #128
ResetZone_RET     ret
IncrementZone     shl   ZoneShift1,      #8
            shl   ZoneShift2,      #8
            shl   ZoneShift3,      #8
            shl   ZoneShift4,      #8
            shl   ZoneShift5,      #8
            shl   ZoneShift6,      #8
            shl   ZoneShift7,      #8
            shl   ZoneShift8,      #8
IncrementZone_RET     ret
ZoneCore          mov   ServoByte,       #0            'Clear ServoByte
            mov   Index,          ZoneIndex        'Set Index Pointer for proper Zone
ZoneSync          mov   SyncPoint,       cnt           'Create a Sync Point with the system counter
            mov   temp,           _NoGlitch        'Test to make sure 'cnt' value won't rollover within Servo's pulse width
            sub   temp,           cnt          wc 'Subtract NoGlitch from cnt ; write result in C flag
      if_C  jmp   #ZoneSync                     'If C flag is set get a new Sync Point, otherwise we are ok.
            mov   LoopCounter,    #8           'Set Loop Counter to 8 Servos for this Zone
            movd  LoadServos,     #ServoWidth8     'Restore/Set self-modifying code on "LoadServos" line
            movd  ServoSync,      #ServoWidth8     'Restore/Set self-modifying code on "ServoSync" line
   LoadServos   rdlong ServoWidth8,    Index          'Get Servo Data
   ServoSync    add   ServoWidth8,     SyncPoint        'Determine system counter location where pulse should end
            sub   Index,          #4           'Decrement Index pointer to next address
            sub   LoadServos,     d_field          'self-modify destination pointer for "LoadServos" line
            sub   ServoSync,      d_field          'self-modify destination pointer for "ServoSync" line
            djnz  LoopCounter,    #LoadServos      'Do ALL 8 servo positions for this Zone
            mov   temp,           _ZoneClocks      'Move _ZoneClocks into temp
            add   temp,           SyncPoint        'Add SyncPoint to _ZoneClocks
   ZoneLoop     cmpsub ServoWidth1,     cnt       nr,wc   '(4 - clocks) compare system counter to ServoWidth ; write result in C flag
            muxc  ServoByte,      ZoneShift1       '(4 - clocks) Set ServoByte.Bit0 to "0" or "1" depending on the value of "C"
            cmpsub ServoWidth2,     cnt       nr,wc   '(4 - clocks) compare system counter to ServoWidth ; write result in C flag
            muxc  ServoByte,      ZoneShift2       '(4 - clocks) Set ServoByte.Bit1 to "0" or "1" depending on the value of "C"
            cmpsub ServoWidth3,     cnt       nr,wc   '(4 - clocks) compare system counter to ServoWidth ; write result in C flag
            muxc  ServoByte,      ZoneShift3       '(4 - clocks) Set ServoByte.Bit2 to "0" or "1" depending on the value of "C"
            cmpsub ServoWidth4,     cnt       nr,wc   '(4 - clocks) compare system counter to ServoWidth ; write result in C flag
            muxc  ServoByte,      ZoneShift4       '(4 - clocks) Set ServoByte.Bit3 to "0" or "1" depending on the value of "C"
            cmpsub ServoWidth5,     cnt       nr,wc   '(4 - clocks) compare system counter to ServoWidth ; write result in C flag
            muxc  ServoByte,      ZoneShift5       '(4 - clocks) Set ServoByte.Bit4 to "0" or "1" depending on the value of "C"
            cmpsub ServoWidth6,     cnt       nr,wc   '(4 - clocks) compare system counter to ServoWidth ; write result in C flag
            muxc  ServoByte,      ZoneShift6       '(4 - clocks) Set ServoByte.Bit5 to "0" or "1" depending on the value of "C"
```

```
        cmpsub  ServoWidth7,    cnt         nr,wc   '(4 - clocks) compare system counter to ServoWidth : write result in C flag
        muxc    ServoByte,      ZoneShift7          '(4 - clocks) Set ServoByte.Bit6 to "0" or "1" depending on the value of "C"
        cmpsub  ServoWidth8,    cnt         nr,wc   '(4 - clocks) compare system counter to ServoWidth : write result in C flag
        muxc    ServoByte,      ZoneShift8          '(4 - clocks) Set ServoByte.Bit7 to "0" or "1" depending on the value of "C"
        mov     outa,           ServoByte           '(4 - clocks) Send ServoByte to Zone Port
        cmp     temp,           cnt         nr,wc   '(4 - clocks) Determine if cnt has exceeded width of _ZoneClocks : write result in C flag
        nop                                         '(4 - clocks) We actually had one instruction to spare
if_NC   jmp     #ZoneLoop                           '(4 - clocks) if the "C Flag" is not set stay in the current Zone
ZoneCore_RET    ret

d_field         long    $0000_0200

ServoWidth1     res     1
ServoWidth2     res     1
ServoWidth3     res     1
ServoWidth4     res     1
ServoWidth5     res     1
ServoWidth6     res     1
ServoWidth7     res     1
ServoWidth8     res     1
ZoneShift1      res     1
ZoneShift2      res     1
ZoneShift3      res     1
ZoneShift4      res     1
ZoneShift5      res     1
ZoneShift6      res     1
ZoneShift7      res     1
ZoneShift8      res     1
temp            res     1
Index           res     1
ZoneIndex       res     1
Zone1Index      res     1
Zone2Index      res     1
Zone3Index      res     1
Zone4Index      res     1
SyncPoint       res     1
ServoByte       res     1
LoopCounter     res     1
_ZoneClocks     res     1
_NoGlitch       res     1
_ServoPinDirection  res 1
```