Experiment 41 การเขียนโปรแกรมประยุกต์การใช้บริการอินเตอร์รัพท์จากหลายแหล่ง

จุดประสงค์ เพื่อเรียนรู้เทคนิคการเขียนโปรแกรมประยุกต์เพื่อให้ไมโครคอนโทรลเลอร์สามารถรับการอินเตอร์รัพท์ที่มาจากหลายแหล่งโดยสามารถตรวจสอบสถานะและกำหนดการทำงานแต่ละแหล่งได้

ทฤษฎีพื้นฐาน

จากใบงานทดลองที่ผ่านมาเราได้เรียนรู้และทดลองการใช้งานการบริการอินเตอร์รัพท์เพียงแหล่งเดียว แต่งในการประยุกต์ใช้งานจริงในระบบควบคุมการผลิตหรืออุปกรณ์เครื่องจักรกลต่างๆ จำเป็นต้องใช้การอินเตอร์รัพท์ที่มาจากหลายแหล่ง เช่นการกดปุ่มสวิทช์ฉุกเฉินเพื่อหยุดการทำงาน การใช้งานร่วมกับ Timer เพื่อสร้างสัญญาณเวลานับลำดับการทำงานในสายงานการผลิต เป็นต้น สิ่งที่สำคัญจะต้องเรียนรู้เป็นพื้นฐานคือรีจิสเตอร์ต่าง ๆ ที่เป็นตัวควบคุมการทำงานของอินเตอร์รัพท์ต่าง ๆ และรีจิสเตอร์ที่เป็นตัวเก็บสถานะแฟล็กที่เกิดอินเตอร์รัพท์แต่ละแหล่ง โดยจะต้องศึกษาแต่ละบิทในรีจิสเตอร์ ว่าควบคุมอะไรและเป็นแฟล็กสถานะของอินเตอร์รัพท์จากแหล่งใดบ้าง หากเข้าใจดีแล้วการใช้งานคงไม่ยาก มีหลักการที่เป็นขั้นตอนดังนี้ คือ

1) ใช้คำสั่งกำหนดตำแหน่งของโปรแกรมอินเตอร์รัพท์ไว้ล่วงหน้า เช่น

```
ON INTERRUPT GOTO INT          INT คือโปรแกรมบริการอินเตอร์รัพท์
```

2) กำหนดค่าในรีจิสเตอร์ควบคุมอินเตอร์รัพท์ในบิทที่เกี่ยวข้องกับการใช้งานไว้ล่วงหน้า เช่น

```
INCONT = %10110000
OPTION_REG = %11000101          เป็นต้น ให้ศึกษาแต่ละบิทมีความหมายว่าอะไร
```

3) เขียนโปรแกรมในส่วนบริการอินเตอร์รัพท์ ในโปรแกรมส่วนแรกที่สำคัญคือการตรวจสอบสถานะแฟล็ก ว่าเกิดอินเตอร์รัพท์จากแหล่งใด โดยตรวจการเกิดลอจิก 1 ที่บิทแฟล็ก เมื่อตรวจได้แล้วก็ให้ไปทำงานตามโปรแกรมย่อย(Subroutines) ที่กำหนดไว้ในเงื่อนไข เมื่อเสร็จงานแล้วให้ล้างแฟล็กนั้นทิ้งไปก่อนโดยกำหนดให้บิทสถานะเป็นลอจิก 0 แล้วส่งกลับมาทำงานปกติ ไม่ว่าจะเกิดอินเตอร์รัพท์จากแหล่งใดก็ตามเราต้องใช้โปรแกรมบริการอินเตอร์รัพท์เดียวกันเท่านั้น ในหลักการนี้เราสามารถพัฒนาโปรแกรมเพื่อทำงานกับอินเตอร์รัพท์จากหลายแหล่งได้

ตัวอย่าง การตรวจสอบสถานการณ์เกิดอินเตอร์รัพท์ที่มาจากพอร์ท RB0/INT

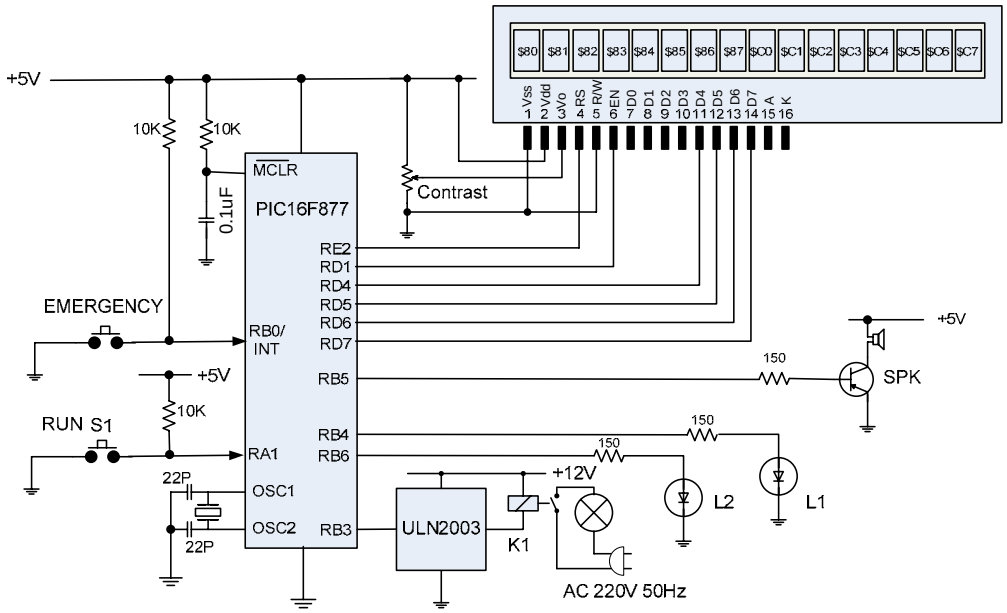
```
IF (INTCON.1 = 1) THEN
    gosub emer_stop
ENDIF
```

ตัวอย่าง การล้างบิทแฟล็กสถานะในรีจิสเตอร์

```
INTCON.1 = 0
```

ในการใช้งานจริง การเกิดอินเตอร์รัพท์จากหลายแหล่งพร้อมกันมีน้อยมาก ๆ หรือจะไม่มีโอกาสเกิดได้เลย ดังนี้ถือเป็นโชคดีที่ไม่ต้องเขียนโปรแกรมตรวจสอบสถานะที่ยุ่งซับซ้อน ในทางปฏิบัติจึงเขียนตรวจสอบสถานะที่เรียงลำดับกันไป

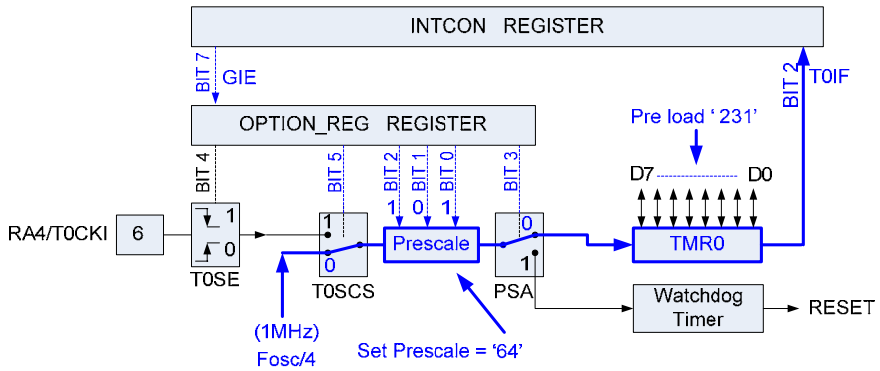
วงจรทดลองตาม Experiment 41



รูปที่ 1 วงจรทดลองตาม Experiment 41

การทำงานของวงจรและระบบควบคุม

วงจรตามรูปที่ 1 เป็นตัวอย่างระบบควบคุมลำดับเวลาการทำงานของอุปกรณ์ต่าง ๆ ได้แก่ หลอดไฟ L1 L2 จอแสดงผลแบบ LCD และรีเลย์ K1 โดยใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุม การทำงานของอุปกรณ์ต่างเป็นไปตามจังหวะเวลาจริง (Real time) โดยการสร้างฐานเวลามาจากตัวคริสตัลออสซิลเลเตอร์ภายในตัวชิพ MCU โดยการใช้ฮาร์ดแวร์ในส่วนของ Prescale , Timer0 เป็นตัวสร้างฐานเวลา 1 Hz และ นำมาป้อนเข้าตัวแปรสร้างฐานเวลาเป็นนาที และชั่วโมงตามลำดับ โดยวิธีการเขียนโปรแกรมควบคุมการจัดการทั้งหมด ในการเขียนโปรแกรมควบคุมดังกล่าวต้องนำการอินเตอร์รัพท์จากการเกิด Overflow ของ Timer0 มาใช้ด้วย อินเตอร์รัพท์อีกส่วนหนึ่ง คือ สวิทช์ฉุกเฉิน EMERGENCY S1 เป็นสวิทช์ที่ใช้หยุดการทำงานของระบบควบคุมอย่างกะทันหัน โดยที่ค่าการนับต่าง ๆ และช่วงเวลายังไม่สูญหายเพียงแต่พักค่าเวลา เมื่อแก้ไขงานเสร็จก็สามารถสั่งให้เดินระบบต่อได้ โดยการกด S1 ค่าเวลาที่นับค้างไว้ก็จะเดินต่อ ในการทำให้ได้ฐานเวลา 1 Hz จากคริสตัลออสซิลเลเตอร์ 4 MHz นั้นจำเป็นต้องทราบข้อมูลเบื้องต้นดังนี้ คือ



รูปที่ 2 แสดงเส้นทางสัญญาณ 1 MHz จาก OSC. ผ่าน Prescale เข้าสู่ Timer0

จากไดอะแกรมตามรูปที่ 2 เป็นการกำหนดให้ Timer0 ทำงานเป็นตัวตั้งเวลา(Timer) โดยมีสัญญาณ OSC. 1 MHz จากภายในที่ ผ่านการหารด้วย 4 มาแล้ว เพื่อให้ Timer0 เกิด Overflow ที่ 1 Hz พอดี จะต้องกำหนดค่าที่ Prescale ให้หารสัญญาณ 1 MHz ด้วย 64 ซึ่งจะได้ผลลัพธ์ 15625 พอดี จากนั้นต้องกำหนดให้ Timer0 ทำการหารด้วย 25 (โดยวิธี ใส่ค่าเบื้องต้นไว้ 231 ซึ่งจะนับอีก 25 ค่าก็จะเต็มครบ 256 ค่าซึ่งเกิด Overflow พอดี) ซึ่งเป็นค่าเดียวที่หารลงตัวได้ผลลัพธ์ 625 พอดี ดังนั้นการกำหนดค่าดังกล่าวตามไดอะแกรมรูปที่ 2 จะเกิดการอินเตอร์รัพท์จำนวน 625 ครั้ง ต่อวินาที ดังนั้นในโปรแกรมส่วนบริการอินเตอร์รัพท์ จะต้องเขียนโปรแกรมเพื่อหารค่า 625 ให้ลงตัวเหลือ 1 ครั้งต่อวินาที หรือ 1 Hz นั่นเอง จากนั้นจะต้องเขียนโปรแกรมสร้างฐานเวลานาที และชั่วโมงไว้เพื่อใช้งานต่อไป ในหลักการตามโปรแกรมนี้เราสามาถเขียนโปรแกรมเพื่อตั้งเวลาได้เป็นเป็นพันเป็นหมื่นชั่วโมงก็ยังได้ ตามในโปรแกรมตัวอย่างนี้สามารถตั้งเวลาได้สูงสุด 256 ชั่วโมง หรือประมาณ 10 วัน

การทำงานของโปรแกรม ในส่วนของโปรแกรมหลักจะไม่ทำอะไรเลยเพียงหมุนวนลูปอยู่เท่านั้น เพื่อรอการเกิดอินเตอร์รัพท์ และกระโดดไปทำงานในส่วนบริการอินเตอร์รัพท์และตรวจสอบบิทแฟล็ก ว่าจะมาจากแหล่งใดบ้าง หากมาจากการกดสวิทช์ฉุกเฉินก็ไปทำงานในโปรแกรมย่อยฉุกเฉิน หากเกิดจาก Overflow ของ Timer0 ก็ทำในส่วนของการตั้งเวลาไป ในการเอาค่าฐานเวลาไปใช้งาน ตามโปรแกรมใน Experiment นี้กำหนดให้หลอด L2 ติดสว่างที่นาทีที่ 2 และไปดับที่นาทีที่ 5 รีเลย์ K1 เริ่มทำงานที่นาทีที่ 12 และไปหยุดทำงานนาทีที่ 15 ขณะที่ทำงานอยู่นั้นหากกดสวิทช์ฉุกเฉินที่ทำงานทุกอย่างหยุดหมดเวลาให้ค่างไว้ค่าเดิม มีข้อความแสดงที่จอ หยุดฉุกเฉิน และรอให้ผู้ใช้กด S1 เพื่อให้ทำงานต่อ

โปรแกรมคำสั่ง

```
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
spk       var  portb.5
L1        var  portb.4
L2        var  portb.6
k1        var  portb.3
s1        var  porta.1
i         var  word
prsc1     var  word
ss        var  byte
mm        var  byte
hh        var  byte
trisa      = %111111
adcon1  = 7
'------------------------
on interrupt  goto int
intcon        = %10110000        'open timer0 overflow int.and RB0/INT
option_reg = %11000101        'open internal osc/4 prescale 64
'
      low L1
      low k1
      prsc1  = 0
      ss      = 0
      mm     = 0
      hh      = 0
      tmr0    = 231        'pre load timer
                           'for 25 count overflow
```

มีต่อหน้าถัดไป

```
'----start main Program --
'
start:
        goto start
        end
'
'--End of Main Program---
'
'
        disable
int:
        if (intcon.1) = 1 then
          gosub emer_stop
        endif
        prsc1 = prsc1+1
        if prsc1 > 625 then             'count for 1 sec.
             prsc1 = 0
             ss = ss+1
             if ss > 59 then
                ss = 0
                mm = mm+1
                if mm > 59 then
                   mm = 0
                   hh = hh+1
                   if hh > 23 then
                      hh = 0
                   endif
                endif
             endif
             toggle   L1
             gosub display              'disp 1 sec. period
          endif
        gosub action
        tmr0     = 231
        intcon.2 = 0
        resume
'
display:
        lcdout $fe,1,dec hh
        lcdout $fe,$85,dec mm,$fe,$c0,dec ss
        return
'
delay:
        for i = 1 to 5000
           pauseus 1
        next i
        return
'
action:
        if (mm >= 2 and mm <= 5) then
           high L2
        else
           low  L2
        endif
        if (mm >= 12 and mm <= 15) then
           high  K1
        else
           low  k1
        endif
        return
```

ต่อ

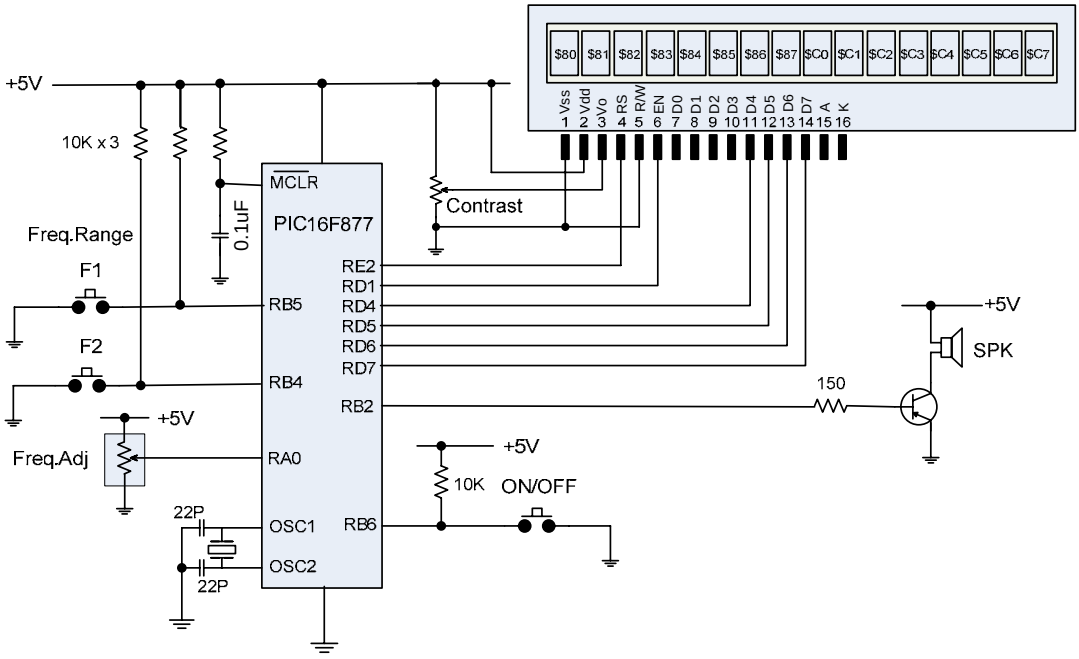
```
'
emer_stop:
        lcdout $fe,1,"EMERG.",$fe,$c0,"STOP!"
        low L1
        low k1
        pause 2000
        lcdout $fe,1,"Press S1"
        loop1:if S1 = 1 then pause 100:goto loop1
        intcon.1 = 0
        return
```

Experiment 42 การเขียนโปรแกรมสร้างสัญญาณความถี่ต่อเนื่องจาก Timer0 จากการประยุกต์ใช้ไมโครคอนโทรลเลอร์ทำงานหลายหน้าที่พร้อม ๆ กัน

จุดประสงค์ เพื่อเรียนรู้เทคนิคการเขียนโปรแกรมประยุกต์เพื่อให้ไมโครคอนโทรลเลอร์สามารถผลิตความถี่ต่อเนื่องขณะที่สามารถประมวลผลคำสั่งอื่น ด้วยวิธีการใช้บริการอินเตอร์รัพท์ได้

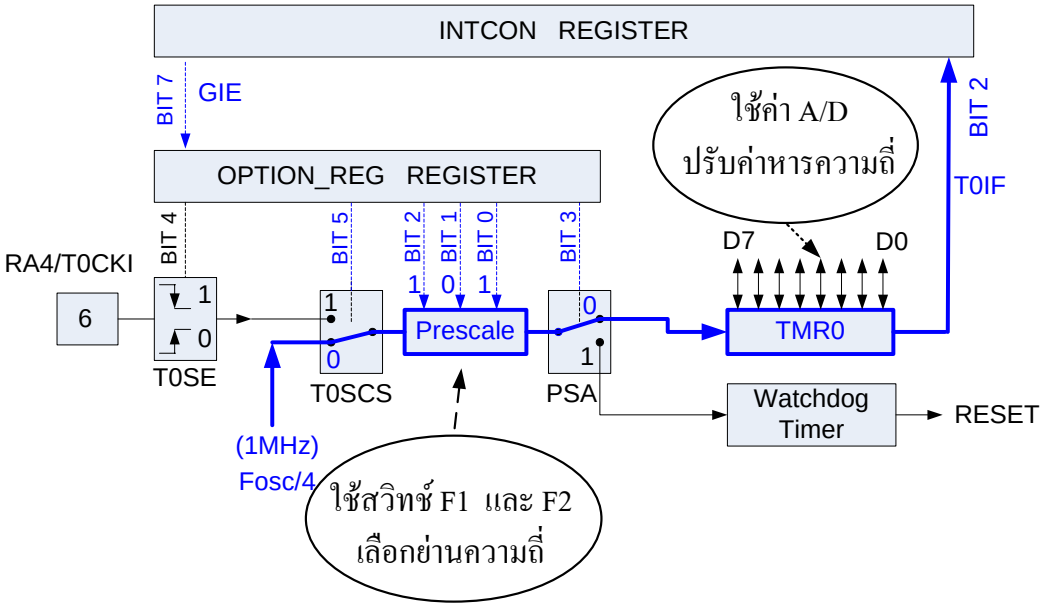
ทฤษฎีพื้นฐาน

ในบางกรณีที่เราต้องใช้ไมโครคอนโทรลเลอร์ประมวลผลทำงานหลาย หน้าที่ไปพร้อม ๆ กัน เช่น การผลิตความถี่ควบคุมอย่างต่อเนื่อง ขณะที่อีกหน้าที่หนึ่งกำลังรอรับค่าการปรับความถี่จากผู้ใช้โดยไม่ขาดตอน การประยุกต์ใช้ในระบบการควบคุมไฟฟ้าในรถยนต์ เช่นในขณะที่กำลังส่งสัญญาณไฟกระพริบอย่างต่อเนื่อง อีกหน้าที่หนึ่งกำลังรอรับการติดต่อรับคำสั่งจากตัวควบคุมภายนอกอยู่ เป็นต้น การทำงานวิธีการดังกล่าวนี้เป็นลักษณะที่เรียกว่า “Multi – tasking” เนื่องจากในระบบไมโครคอนโทรลเลอร์มีขนาดเล็กเกินไปที่จะเขียนระบบปฏิบัติการแบบ Multi-tasking จริง ๆ ได้ ดังนั้นวิธีการที่จะนำมาใช้ เราจึงต้องดึงความสามารถในด้านอินเตอร์รัพท์มาใช้ให้มากที่สุดเท่าที่จะทำได้ ในใบงานตาม Experiment 42 นี้ จะเป็นตัวอย่างในการเขียนโปรแกรมให้ไมโครคอนโทรลเลอร์ทำงานแบบ Multi-tasking โดยใช้วิธีการอินเตอร์รัพท์ โดยการทำงานเป็น Signal generator ผลิตความถี่ออกมาอย่างต่อเนื่อง ในขณะเดียวกัน เราสามารถกดสวิทช์เลือกย่านความถี่ และใช้ Potentiometer แปลงสัญญาณเป็น A/D ไปปรับความถี่ได้



รูปที่ 1 วงจรทดลองตาม Experiment 42

หลักการทำงานของโปรแกรม คือ โปรแกรมจะผลิตความถี่ออกมาที่ พอร์ท RB2 โดยการทำงานของ Timer0 ร่วมกับ Prescale การผลิตความถี่ออกมาต่อเนื่องโดยการทำงานแบบอินเตอร์รัพท์ สวิทช์ F1 F2 ใช้ปรับย่านความถี่โดยวิธีการเลือกค่าอัตราส่วนการหารความถี่ใน Prescale ส่วนการปรับค่าความถี่ในแต่ละย่านจะใช้ค่าอัตราส่วนการหารความถี่ใน Timer0 โดยการนำค่ามาจาก การรับค่า A/D จาก Potentiometer



รูปที่ 2 แสดงการเลือกย่านความถี่ และปรับค่าความถี่ในโปรแกรมผลิตความถี่ต่อเนื่อง

โปรแกรมคำสั่ง

```
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
spk       var  portb.2
f1        var  portb.5
f2        var  portb.4
on_off    var  portb.6
'
stat      var  bit
adc       var  byte
v1        var  word
freq      var  word
'
trisa     =  %111111
trisb     =  %11110000
adcon1 =  0
'-----------------------
on interrupt  goto int
intcon        = %10100000    'open timer0 overflow int.
option_reg  = %11000101     'open internal osc/4 prescale 64
'
```

มีต่อหน้าถัดไป

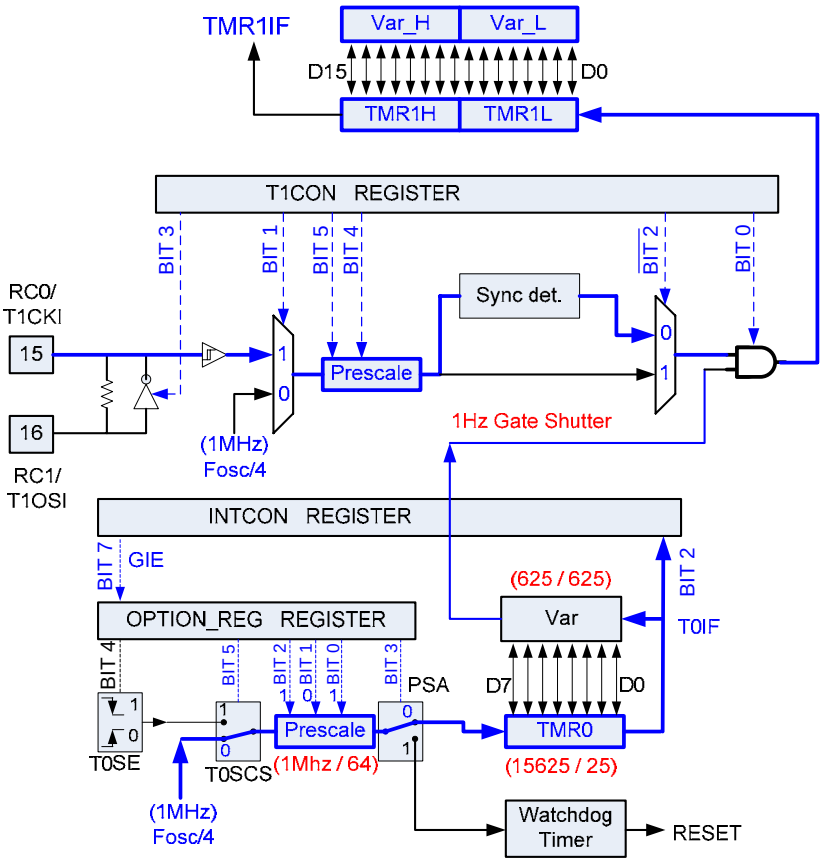
```
'
        stat     =  1
        adcin 0,adc
        tmr0    = adc          'pre load timer
                               'for 25 count overflow
'----start main Program --
'
start:
        if f1 = 0 then option_reg = %11000101
        if f2 = 0 then option_reg = %11000000
        if on_off = 0 then
             toggle  stat
             pause 300
        endif
        adcin 0,adc
        goto start
        end
'
'--End of Main Program---
'
'
        disable
int:
       v1 = v1+1
       if v1   >= 10 then     'divide by 10
             v1 = 0
             if stat  = 1 then
                  toggle   spk
             else
                   low spk
              endif
         endif
         tmr0       = adc
         intcon.2  = 0
         resume
```

Experiment 43 การเขียนโปรแกรมนับความถี่ต่อเนื่องจากสัญญาณภายนอก (Frequency Counter)

จุดประสงค์ เพื่อเรียนรู้เทคนิคการเขียนโปรแกรมประยุกต์เพื่อให้ไมโครคอนโทรลเลอร์นับความถี่ โดยใช้โมดูล Timer 0 ทำงานร่วมกับ Timer 1 และการอินเตอร์รัพท์

ทฤษฎีพื้นฐาน

การนับจำนวน และการนับความถี่มีความหมายแตกต่างกัน การนับจำนวนจะไม่มีคาบเวลาเข้ามาเกี่ยวข้อง คือมีแต่จำนวนว่านับมาได้เท่าไร การนับความถี่จะมีคาบเวลาเข้ามากำหนดว่า จะนับจำนวนลูกคลื่นพัลส์ ที่เข้ามาในช่วงเวลา เช่น ใน 1 วินาที หรือ Hertz (Hz) ใน 1 นาที ได้แก่ความเร็วการหมุนของเพลาเป็นจำนวนรอบต่อนาที (RPM : Revolution Per Minute) หรือความเร็วการวิ่งของรถยนต์เป็น กิโลเมตร ต่อ ชั่วโมง (KM per Hour) เป็นต้น ในการใช้ไมโครคอนโทรลเลอร์กับภาษา Pic Basic Pro เขียนโปรแกรมเพื่อนับความถี่ มี 2 วิธี คือ การใช้คำสั่ง COUNT ตามที่ได้ศึกษาและปฏิบัติมาแล้ว วิธีนี้จะใช้ไม่ค่อยได้ผลกรณีที่การทำงานของโปรแกรมจะต้องทำหลายหน้าที่พร้อม ๆ กันในลักษณะ multi tasking เนื่องจากขณะที่โปรแกรมกำลังทำงานในคำสั่ง COUNT ในช่วงคาบเวลานั้นโปแกรมไม่สามารถทำคำสั่งอื่นได้เลย หากกรณีที่ต้องตรวจจับสัญญาณอิพุทที่มีความไวสูง ๆ อาจผิดพลาดได้



รูปที่ 1 แสดงโครงสร้างและการใช้ Timer 0 และ Timer 1 ในการเขียนโปรแกรมให้ทำงานเป็นเครื่องนับความถี่ (Frequency Counter)

ในใบงานตาม Experiment 43 นี้ เราจะใช้ขีดความสามารถของการอินเตอร์รัพท์ และ Timer module ที่มีอยู่ โดยการใช้ Timer 0 และการอินเตอร์รัพท์เป็นตัวกำหนดคาบเวลาการนับ (Counter period) โดยส่งคาบเวลา 1 Hz ไปบังคับเปิด และปิดประตูการนับลูกพัลส์ที่เข้ามานับสะสมที่ Timer 1 หลักการ คือ จะต้องเขียนโปรแกรมให้ Timer 0 ผลิตสัญญาณ 1 Hz ออกมา เขียนโปรแกรมกำหนดรีจิสเตอร์ T1CON ให้Timer 1 ทำงานในโหมด Counter โดยมีขา RC0 เป็นตัวรับสัญญาณพัลส์จากภายนอก การใช้งาน Timer 1 ใช้งานเช่นเดียวกันกับ Timer 0 โดยมีข้อแตกต่างกันเล็กน้อยดังนี้ คือ

ตัว Timer 1 มีขนาด 16 บิท รองรับการตั้งเวลา และการนับได้ถึง 65536 ค่า หรือ FFFFH สามารถใช้คริสตัลความถี่ต่ำไม่เกิน 200 KHz สร้างความถี่ฐานเวลาได้โดยต่อที่ขา RC0 กับ RC1 หรือจะป้อนสัญญาณพัลส์เข้าที่ขา RC0 ได้โดยตรง หรือจะใช้ความถี่ Machine Cycle Fosc. / 4 หรือ 1 MHz (กรณีใช้คริสตัลขนาด 4 MHz) Timer 1 สามารถเลือกตัวหารความถี่ (Prescaler) ได้ 4 ค่า คือ 1:1 1:2 1:4 และ 1:8 โดยใช้บิทควบคุมในรีจิสเตอร์ T1CON ซึ่งเป็นรีจิสเตอร์ที่ควบคุมการทำงานของ Timer 1 นอกจากนั้น Timer 1 ยังมีบิทควบคุมการทำงาน และหยุดทำงานได้ และสามารถกำหนดให้สัญญาณ Count สามารถ Synchronized กับสัญญาณนาฬิกาของ Machine Cycle ได้ด้วย

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

รูปที่ 2 แสดงโครงสร้างของรีจิสเตอร์ T1CON ที่ใช้ควบคุมการทำงานของ Timer 1

Timer 1 สามารถรองรับการทำงานอินเตอร์รัพท์เมื่อการนับถึง Overflow โดยมีสัญญาณบิทแฟล็กเกิดขึ้นที่ตำแหน่ง TMR1IF (บิท 0)ในรีจิสเตอร์ PIR1 แต่การใช้งานอินเตอร์รัพท์จะต้องกำหนดบิทในรีจิสเตอร์ที่

INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIE2	—	(5)	—	EEIE	BCLIE	—	—	CCP2IE

เกี่ยวข้องก่อน คือ รีจิสเตอร์ INTCON PIE1 และ PIR1 โดยใช้คำสั่งดังต่อไปนี้ วางไว้ที่หัวของโปรแกรม

INTCON = %11000000

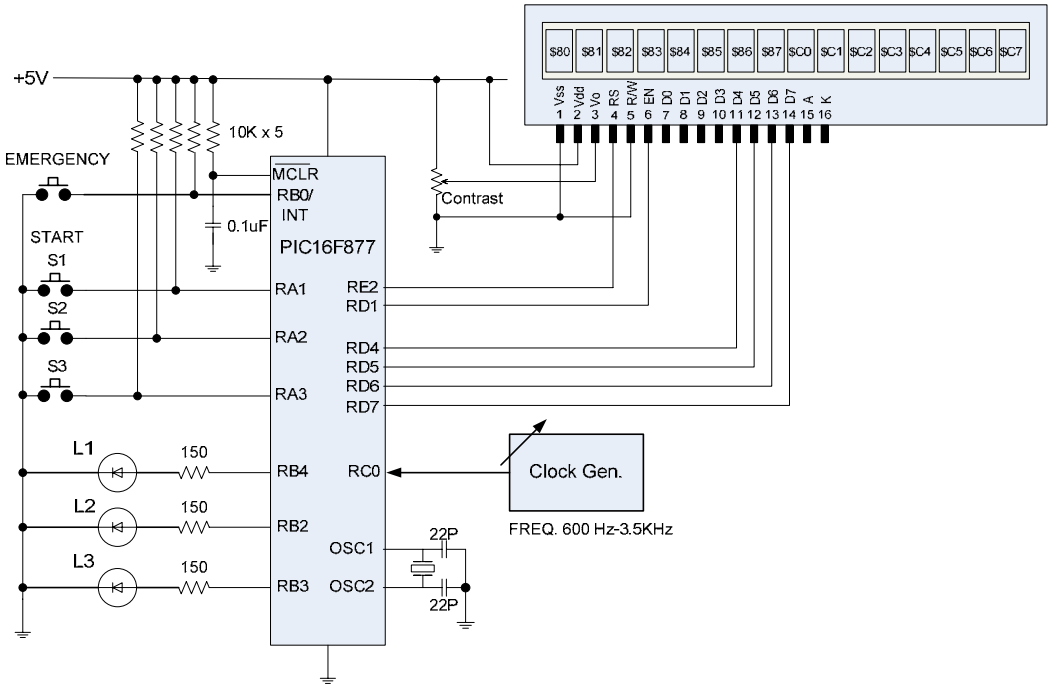
PIE1 = %00000001

PIR1 = %00000001

เมื่อเกิดอินเตอร์รัพท์เนื่องจาก Timer 1 เกิด Overflow หลังจากทำงานในโปรแกรมบริการอินเตอร์รัพท์เสร็จแล้วจะต้องเคลียร์บิทแฟล็กให้มีค่าเป็น 0 ดังนี้

PIR1.0 = 0

แต่ถ้าไม่ได้งานอินเตอร์รัพท์ไม่ต้องกำหนด และถึงแม้จะไม่กำหนดการใช้งานอินเตอร์รัพท์ เมื่อเกิด Overflow ใน Timer 1 ก็ยังคงมีบิทแฟล็กเกิดขึ้น ซึ่งจะเกิดผลดีคือ เราสามารถนำไปเป็นเงื่อนไขในการกำหนดการเปลี่ยนย่านการนับได้



รูปที่ 3 วงจรทดลองตาม Experiment 43

การทำงานของวงจร

เป็นวงจรรับสัญญาณจาก Clock Gen. เพื่อวัดค่าความถี่แดงค่าออกที่จอ LCD มีสวิทช์ควบคุมใช้งานตามจุดประสงค์ต่าง ๆ 4 ตัว คือ Emergency เป็นสวิทช์กดเพื่อหยุดการทำงานฉุกเฉิน ต่อที่ขา RB0/INT ทำงานแบบอินเตอร์รัพท์แบบกดทำงานทันที S1 เป็นสวิทช์กดควบคุมการยกเลิกการทำงานฉุกเฉิน S2 เป็นสวิทช์ควบคุมการทำงานของหลอด L2 ทำงานแบบ Jogging คือ กดหลอดติด ปล่อยหลอดดับ S3 เป็นสวิทช์ควบคุมการทำงานของหลอด L3 ทำงานแบบ Toggle คือกดครั้งแรกติด กดครั้งที่สองดับ แบบนี้สลับกันไป

การทำงานของโปรแกรม

เป็นโปรแกรมประยุกต์การใช้งานอินเตอร์รัพท์จากหลายแหล่ง ควบคู่กันไปกับการใช้งาน Timer module ทั้งสองตัวเพื่อทำหน้าที่วัดค่าสัญญาณความถี่พัลส์ ที่ป้อนเข้าทางขา RC0 โดยให้ทุกงานทำไปพร้อม ๆ กันในลักษณะ multi tasking โดยการบริหารเวลาแบบให้การเกิดอินเตอร์รัพท์เป็นตัวกำหนดจ่ายการทำงานในแต่ละ Task โดยจะไม่ทีการทำ Task ในโปรแกรมหลักเลย งานแต่ละ Task ได้แก่ งานนับค่าความถี่โดยนับค่าจำนวนลูกพัลส์ในรีจิสเตอร์ TMR1L และ TMR1H แล้วกำหนดค่าลงในตัวแปร freq ซึ่งอยู่ในโปรแกรมย่อย count_process งานหยุดภาวการณ์ทำงานเมื่อกดสวิทช์ฉุกเฉิน ซึ่งอยู่ในโปรแกรมย่อย emer_stop และ delay งานแสดงค่าความถี่ที่วัดได้ออกทางจอ LCD ซึ่งอยู่ในโปรแกรมย่อย display งานตรวจสอบเองไขการกดสวิทช์ควบคุมทั้ง 4 ตัว ซึ่งอยู่ในโปรแกรมย่อย action การทำงานของโปรแกรมใน Task ต่าง ๆ จะเกิดขึ้นทุก ๆ การเกิดอินเตอร์รัพท์ เมื่อ Timer0 เกิด Overflow ตามโปรแกรมจะกำหนดค่าไว้เท่ากับ 1 / 625 วินาที การทำงานแต่ละ Task จะถูกปิดกั้นด้วยวิธีการเปิด – ปิดอินเตอร์รัพท์กั้นไว้ เพื่อไม่ให้มีผลกระทบกันเรื่องเวลา โดยเฉพาะคาบเวลาการเปิด - ปิด คาบเวลาการนับความถี่ วิธีการเช่นนี้เราจะได้การทำงานของโปรแกรมที่ทำงานอย่างมีประสิทธิภาพ สามารถทำงานตอบสนองได้อย่างรวดเร็วในทุก ๆ Task

โปรแกรมคำสั่ง

```
'=========================
'Freq. counter using Timer1
'
'1 sec. gate shutter period
'=========================
DEFINE LCD_DREG PORTD
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTE
DEFINE LCD_RSBIT 2
DEFINE LCD_EREG PORTD
DEFINE LCD_EBIT 1
spk     var  portb.5
L1      var  portb.4
L2      var  portb.2
L3      var  portb.3
s1      var  porta.1
s2      var  porta.2
s3      var  porta.3
i       var  word
prsc1   var  word
freq    var  word
trisa    = %111111
adcon1 = 7
```

มีต่อหน้าถัดไป

```
'-----------------------
on interrupt  goto int
intcon         = %10110000   'open timer0 overflow int.and RB0/INT
option_reg  = %11000101   'open internal osc/4 prescale 64
t1con          = %00000010   'set timer1 for ext.clk, sync,
'                      prescale 1:1
       low L1
       low L2
       low L3
       prsc1  = 0
       tmr0   = 231         'pre load timer
                            'for 25 count overflow
'----start main Program --
'
start:
'       gosub action
       goto start
       end
'
'--End of Main Program---
'
'
       disable
int:
       prsc1 = prsc1+1
       if prsc1 > 625 then   'count for 1 sec.
            t1con.0 = 0
            prsc1    = 0
            gosub count_process
            toggle   L1
            gosub display   'disp 1 sec. period
       endif
       gosub action
       tmr0      = 231
       intcon.2 = 0
       resume
'
display:
       lcdout $fe,1,"Freq="
       lcdout $fe,$c0,dec freq,$fe,$c6,"Hz"
       return
'
delay:
       for i = 1 to 5000
          pauseus 1
       next i
       return
'
action:
       t1con.0 = 0   'close gate shutter
       if (intcon.1) = 1 then
         gosub emer_stop
       endif
       if s2 = 0 then
          high L2
       else
          low l2
       endif
       if s3 = 0 then
          toggle L3
          idle:if s3 = 0 then pause 50:goto idle
       endif
       t1con.0 = 1   'open gate shutter
       return
```

มีต่อหน้าถัดไป

```
'
count_process:
      freq.byte0 = tmr1l
      freq.byte1 = tmr1h
      tmr1l      = 0
      tmr1h      = 0
      return
'
emer_stop:
            lcdout $fe,1,"EMERG.",$fe,$c0,"STOP!"
            low L1
            low L2
            low L3
            pause 2000
            lcdout $fe,1,"Press S1"
            loop1:if S1 = 1 then pause 100:goto loop1
            tmr1l      = 0
            tmr1h      = 0
            intcon.1   = 0
        return
```

แนวคิดในการนำไปประยุกต์ใช้งาน

1) สร้างเครื่องวัดความถี่ แบบสามารถกำหนดย่านวัดได้แบบ Auto range

2) สร้างเครื่องวัดและควบคุมความเร็วของมอเตอร์ดีซี โดยรับค่าสัญญาณพัลส์จาก Encoder นำมาวัดและควบคุมความเร็วแบบดิจิตอลเซอร์โว

3) วัดค่าและควบคุมความเร็วของมอเตอร์ล้อหุ่นยนต์ เพื่อรักษาแนวและความเร็วการเคลื่อนที่ โดยให้ศึกษาการออกแบบฮาร์ดแวร์เพิ่มเติมที่สามารถมัลติเพล็กซ์ค่าความถี่จาก Encoder ทั้ง 2 ล้อมาสลับวัดกันได้

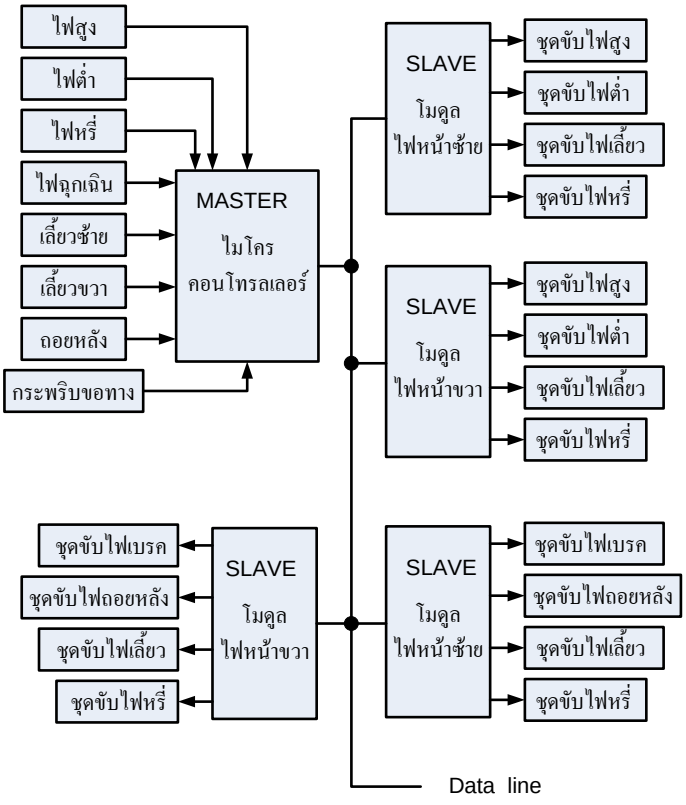
4) ให้พัฒนาโปรแกรมต่อไปให้สามารถวัดค่าความถี่สูง ๆ มากได้ โดยวิธีการลดค่า Shutter period ในการเปิด – ปิด Timer 1 ให้มีค่าน้อยกว่า 1 วินาที แล้วนำค่าที่วัดได้มาคูณคำนวณชดเชยภายหลัง

Experiment 44 การเขียนโปรแกรมให้รับการสื่อสารจากพอร์ทอนุกรมขณะประมวลผลคำสั่งต่อเนื่อง
(กรณีศึกษา : ระบบสื่อสารควบคุมไฟฟ้าในรถยนต์แบบใช้พอร์ทอนุกรม)

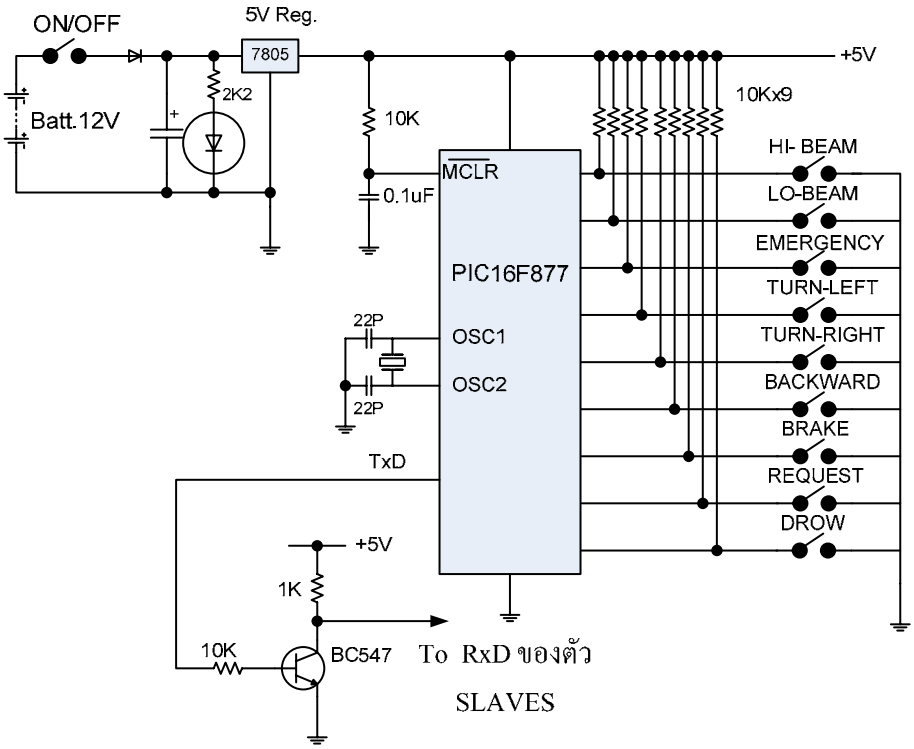
จุดประสงค์ เพื่อเรียนรู้เทคนิคการเขียนโปรแกรมประยุกต์เพื่อให้ไมโครคอนโทรลเลอร์สามารถรับการสื่อสารสัญญาณควบคุมผ่านทางพอร์ทอนุกรม ขณะที่กำลังประมวลผลคำสั่งและทำงานต่อเนื่อง โดยใช้การทำงานแบบอินเตอร์รัพท์

ทฤษฎีพื้นฐาน

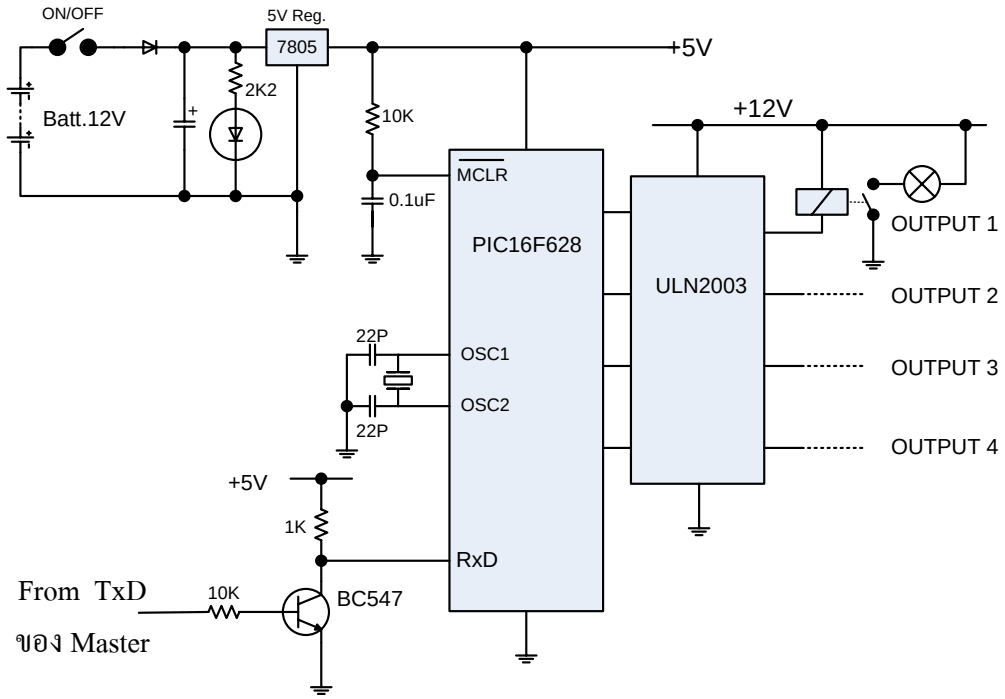
การพัฒนาให้ไมโครคอนโทรลเลอร์ทำงานหลายหน้าที่คล้ายกับการทำงานแบบ(multi-tasking) โดยวิธีการใช้ขีดความสามารถด้านอินเตอร์รัพท์มีความจำเป็นต้องทำความเข้าใจให้ลึกซึ้ง เนื่องจากระบบการควบคุมที่นำไปใช้งานจริง มีความจำเป็นมาก ระบบสื่อสารควบคุมไฟฟ้าในรถยนต์แบบใช้พอร์ทอนุกรมเป็นอีกตัวอย่างหนึ่งที่น่านำมาเรียนรู้ เนื่องจากความรู้ที่ได้จะเป็นพื้นฐานสำคัญในการออกแบบระบบควบคุมไฟฟ้าในรถยนต์รุ่นใหม่ ๆ ที่ใช้ระบบสื่อสารสั่งการด้วยสายสัญญาณเส้นเดียวแบบ half – duplex ซึ่งมีใช้กันในรถยนต์ราคาแพงทั่วไป หากมีการพัฒนานำไปใช้บ้าง ก็จะทำให้จำนวนสายไฟสัญญาณและแสงสว่างลดลงจากเป็นมัดหลายสิบเส้น เหลือเพียงเส้นเดียวกับสายไฟบวก 12 V



รูปที่ 1 แสดงบล็อกไดอะแกรมและเส้นทางการสื่อสารระบบควบคุมไฟฟ้าในรถยนต์
แบบใช้สายสื่อสารเส้นเดียวแบบ Half – Duplex



รูปที่ 2 แสดงวงจรต้นแบบที่ใช้งานจริงในส่วน Master Control



รูปที่ 3 แสดงวงจรต้นแบบที่ใช้งานจริงในส่วนของโมดูล SLAVES ทั้ง 4 ชุด
ที่ควบคุมไฟหน้า 2 ชุด และไฟท้าย 2 ชุด

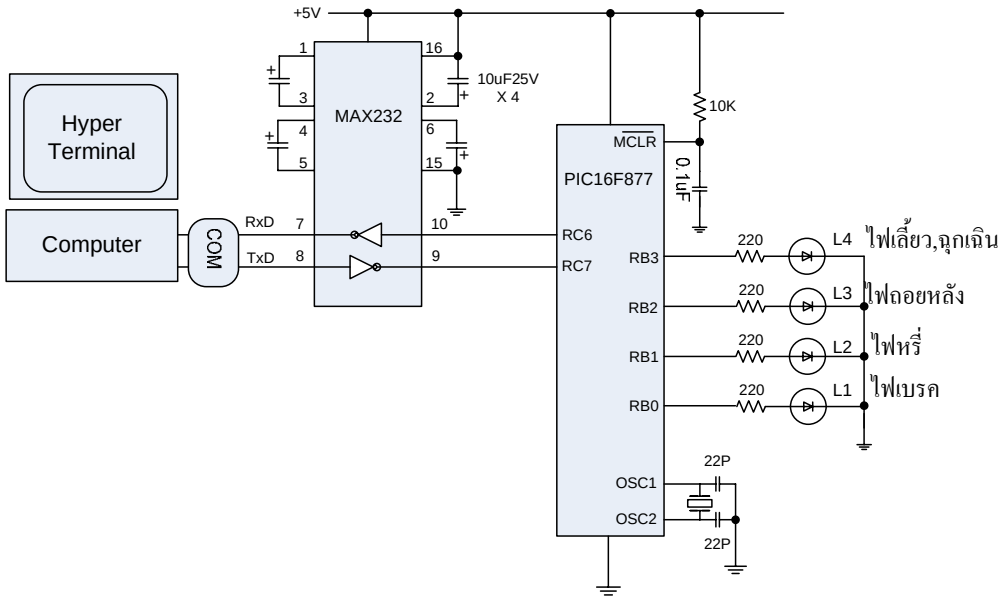
เนื่องจากโครงงานนี้เป็นโครงงานวิจัยที่สำเร็จแล้ว ที่พัฒนาต้นแบบโดยผู้เขียนตำรา ร่วมกับที่นักศึกษาสาขาวิศวกรรมศาสตร์ที่เคยสอนอยู่ ไม่สามารถนำของจริงมาทดลองได้ แต่จะนำหลักการเขียนโปรแกรมมาใช้กับบอร์ดทดลอง โดยใช้สวิทช์คีย์บอร์ดของคอมพิวเตอร์เป็น MASTER CONTROL โมดูล และบอร์ดทดลองเป็น SLAVE MODULE โดยจะเขียนโปรแกรมใส่ในโมดูล SLAVE ที่สมมุติขึ้นมา ส่วนในการทดลองของโมดูล MASTER จะใช้สวิทช์ในบอร์ดทดลองเป็นโมดูล MASTER และจะตรวจสอบการทำงานโดยส่งโค้ดไปแสดงที่ Hyper Terminal แทน เมื่อพัฒนาโปรแกรมทั้งโมดูล MASTER และ SLAVE ได้ทั้งหมดแล้ว หากต้องการเชื่อมต่อกันทั้งระบบจะต้องใช้บอร์ดทดลอง 2 ชุดโดยต่อสาย RS-232C ต่อเชื่อมเข้าด้วยกัน

การทำงานของโมดูล SLAVE สมมุติว่าเป็นโมดูลควบคุมไฟท้าย ที่รับการติดต่อจากตัว MASTER โดยผ่านพอร์ทอนุกรม RS-232C แบบ Full – Duplex ที่ไม่ใช้แบบ Half – Duplex เนื่องจากจะต้องมีการแก้ไขวงจรที่เป็นฮาร์ดแวร์บางจุด ผู้เขียนเห็นว่าเรื่องนี้ไม่ใช่สาระสำคัญ หากเข้าใจในระบบการเขียนโปรแกรมสื่อสารตามใบงานทดลองนี้แล้ว การทำงานของโมดูล SLAVE มีดังนี้ คือ จะคอยรับการติดต่อจากโมดูลตัว MASTER ซึ่งเป็นคอมพิวเตอร์ การติดต่อจะเป็นโค้ดรหัสที่เป็นตัวอักขระ ตัวลูกจะนำโค้ดเหล่านั้นมาตีความหมายกำหนดการประมวลผลทำงานและส่งเอ้าท์พุทไปขับรีเลย์ เพื่อให้ชุดหลอดไฟท้ายทำงานตามโค้ดที่ตัว MASTER ส่งมา โค้ดรหัสที่กำหนดขึ้นเป็นไปตามตาราง ที่ 1

ตารางที่ 1 (ตัวอย่าง) โค้ดรหัสการทำงานของโมดูลไฟท้ายรถยนต์

1	ถามสภาพการทำงานของโมดูล
2	ตอบกลับ หากทำงานปกติ
Q	เปิดไฟหรี่
W	ปิดไฟหรี่
E	เปิดไฟถอยหลัง
R	ปิดไฟถอยหลัง
T	เปิดไฟเลี้ยว
Y	ปิดไฟเลี้ยว
U	เปิดไฟฉุกเฉิน
I	ปิดไฟฉุกเฉิน
A	เปิดไฟเบรก
S	ปิดไฟเบรก

จากการวิเคราะห์การทำงานของโมดูล SLAVE จะพบว่า เมื่อตัว MASTER ส่งโค้ดมาให้ทำงาน ทุกงานต้องทำต่อเนื่องจนกว่าจะส่งโค้ดมาให้หยุดทำงาน การทำงานต่อเนื่องส่วนมากจะมีการเปิด และปิดหลอดไฟ ซึ่งไม่ค่อยมีปัญหากับการทำงานแบบหลายหน้าที่มากนัก แต่มีบางงานที่เป็นปัญหา คือการทำงานของไฟเลี้ยว และไฟฉุกเฉิน เนื่องจากต้องคอยปิด และเปิดตลอดเวลา ในขณะที่คอยรับโค้ดจากตัว MASTER อยู่ หากการออกแบบเขียนโปรแกรมไม่คำนึงถึงเรื่องดังกล่าว จะเป็นปัญหาในการนำไปใช้งานจริง ดังนั้น ในกรณีนี้จำเป็นต้องใช้หลักการอินเตอร์รัพท์เข้ามาใช้ร่วมกับโมดูล Timer เพื่อให้การทำงานเป็นลักษณะหลายหน้าที่ (multi-tasking)ได้



รูปที่ 4 วงจรโมดูล SLAVE ปฏิบัติทดลองตาม Experiment 44

โปรแกรมคำสั่งสำหรับทดสอบการทำงานของโมดูล SLAVE

```
'====================
'Slave  Module
'====================
'
include "modedefs.bas"
'
brk      var  portb.0
pkg      var  portb.1
rwd      var  portb.2
turn     var  portb.3
s_in     var  portc.7
s_out    var  portc.6
'
stat     var  bit
div      var  word
code     var  byte
'
trisa      = %111111
trisb      = %11100000
adcon1 = 7
'-----------------------
on interrupt  goto int
intcon         = %10100000       'open timer0 overflow int.
option_reg = %11000101       'open internal osc/4 prescale 64
'
       tmr0   = 231         'pre load timer
       stat    = 0
       low  brk
       low  pkg
       low  rwd
       low  turn
```

มีต่อหน้าถัดไป

```
'----start main Program --
'
start:
        gosub recive
        gosub action
        goto start
        end
'
'--End of Main Program---
'
'
        disable
int:
       div = div+1
       gosub recive
       if div >= 50 then     'divide by 50
           div  = 0
          if stat = 1 then
              toggle   turn
          else
              low turn
          endif
        endif
        tmr0     = 231
        intcon.2 = 0
        resume
'
recive:
        serin s_in,2,2,time,code
        return
action:
        if code = "1" then
            serout s_out,2,["SystemOK",10,13]
            code = "0"
        endif
        if code = "q"  then high pkg
        if code = "w" then low  pkg
        if code = "e"  then high rwd
        if code = "r"  then low  rwd
        if code = "t"  then stat = 1
        if code = "y"  then stat = 0
        if code = "u" then stat = 1
        if code = "i"  then stat = 0
        if code = "a"  then high brk
        if code = "s"  then low  brk
        return
'
time:return
```

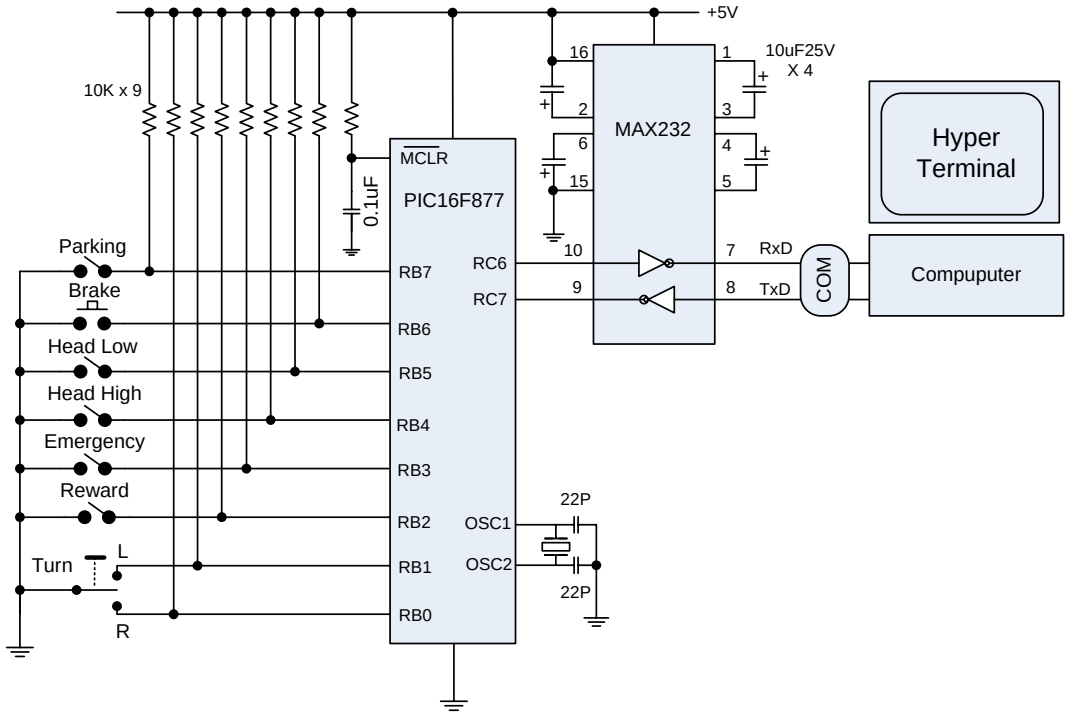
หมายเหตุ

1) เนื่องจากการใช้โปรแกรม Hyper Terminal กับโปรแกรมดาวน์โหลดตัวชิพ ใช้พอร์ท RS-232C เดียวกัน จะใช้งาน 2 โปรแกรมพร้อมกันไม่ได้ หากต้องการดาวน์โหลดตัวชิพ ต้องปิดโปรแกรม Hyper Terminal ก่อน และถ้าต้องการใช้โปรแกรม Hyper Terminal ต้องปิดโปรแกรมดาวน์โหลดก่อน แต่หากจะใช้แยกพอร์ทกันก็สามารถเปิดใช้พร้อมกันได้

2) การทำงานคำสั่ง SERIN จะหยุดรอจนกว่าจะมีข้อมูลเข้าตัวแปร ในโปรแกรมจึงกำหนด Option ให้หยุดรอแค่ 2 mS หากไม่มีก็ให้ผ่านไปโดยการหลอกให้กระโดดไปที่โปรแกรมย่อย time

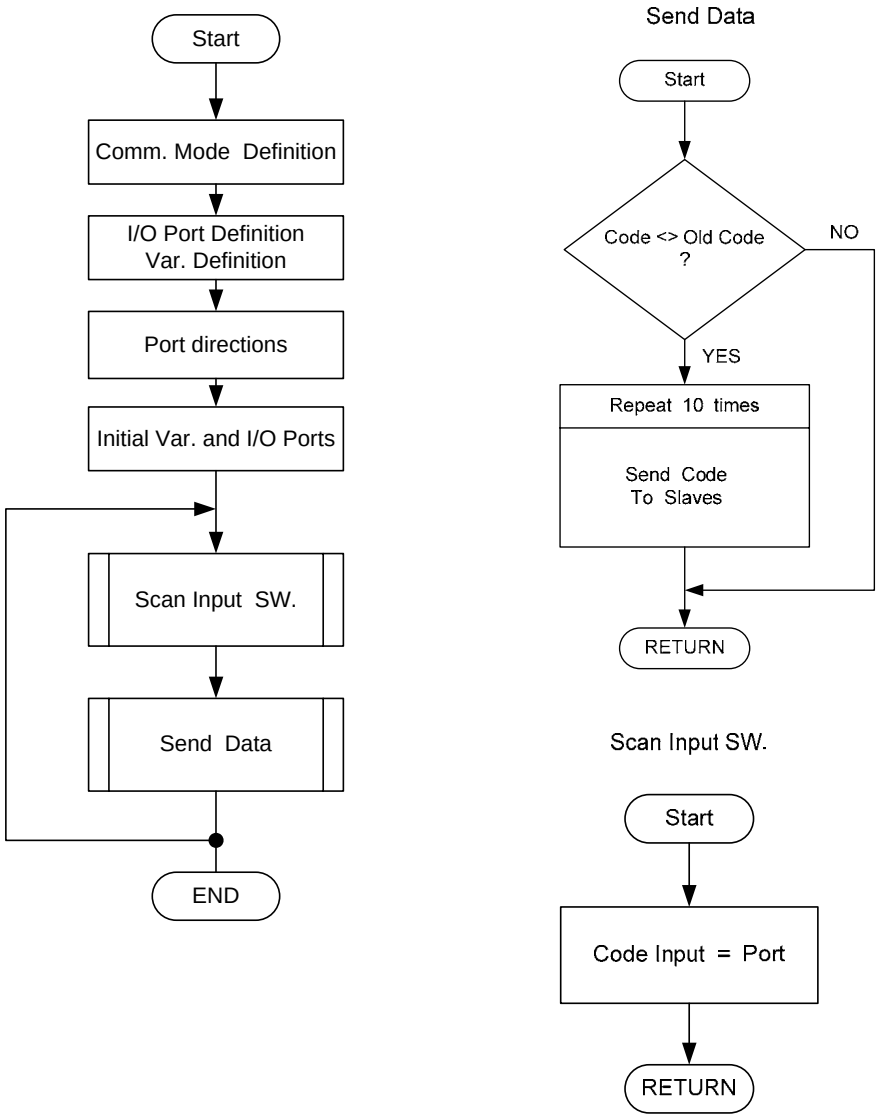
2) ตัวแปร code ที่อยู่ในโปรแกรมหากรับค่าใดมาแล้วจะคงค่านั้นตลอดไป เช่นถ้าส่งโค้ด “1” หากไม่รีบเปลี่ยนค่าเป็นอื่น ก็จะมีข้อความ SystemOK ออกมาที่จอ Hyper Terminal ตลอดเวลา

การทำงานของโมดูล MASTER วงจรประกอบด้วยสวิทช์ที่ต่อกับส่วนควบคุมไฟสัญญาณต่าง ๆ ที่แผงคอนโซลหน้ารถ ที่คันบังคับต่าง ๆ ได้แก่ สวิทเปิด – ปิดไฟไฟหน้า ไฟสูง – ไฟต่ำ ไฟหรี่ ไฟฉุกเฉิน ไฟเลี้ยวซ้าย – ขวา ไฟถอยหลัง ไฟกระพริบขอทาง ไฟเบรก เป็นต้น ลักษณะการทำงานของสวิทช์เหล่านี้ มีหลายแบบ ได้แก่ แบบคันโยกทำงานค้างไฟสูง ไฟต่ำ ไฟฉุกเฉิน แบบกดติดปล่อยดับ เช่นไฟเบรก เป็นต้น การทำงานของโปรแกรม คือ โปรแกรมจะสแกนรับการทำงานของสวิทช์ทุกตัว นำมาเป็นเงื่อนไขการประมวลผลการทำงานและส่งโค้ดไปยังตัว SLAVES ทุกตัวทางพอร์ทอนุกรม เนื่องจากการออกแบบโปรแกรมในโมดูล MASTER ต้องการให้ทำงานครบฟังก์ชั่นการควบคุม จำเป็นต้องกำหนด พอร์ท B ทั้งพอร์ทต่อกับสวิทช์ ควบคุมทั้งหมด โดยแต่ละบิทต่ออยู่กับสวิทช์แต่ละตัวต่ออยู่กับขา I/O ของพอร์ท B ดังนั้นการโยก หรือกดสวิทช์ใด ๆ ก็คือการเปลี่ยนค่าที่พอร์ท B นั่นเอง เราเอาค่าที่พอร์ท B ที่เกิดจากการเปิดปิดสวิทช์ มาเป็นค่าโค้ด ส่งออกไปยังตัว SLAVE โมดูลทางพอร์ทอนุกรม เมื่อโมดูลได้รับโค้ดจะแปลความหมายโค้ดแต่ละบิทไปสั่งทำงานขับหลอดไฟสัญญาณอีกทีหนึ่ง



รูปที่ 5 วงจรโมดูล MASTER ทดลองตาม Experiment 44

โปรแกรม Flowchart



รูปที่ 6 Main Program ของโมดูล Master

การทำงานของโปรแกรม กำหนดให้พอร์ท Bทุกบิทเป็นอินพุท โปรแกรมจะวนลูปไปรับค่าที่พอร์ท B และนำค่าที่พอร์ท B มากำหนดเป็นตัวแปร code แล้วนำค่า code ส่งออกทางพอร์ทอนุกรม 3 ครั้งติดต่อกันโดยกำหนดให้แต่ละครั้งห่างกัน 4 mS. ที่ต้องทำซ้ำเช่นนี้ก็เพื่อย้ำให้โมดูล SLAVE รับค่าได้ถูกต้อง หากโค้ดไม่มีค่าเปลี่ยน จะไม่มีการส่ง ไม่ให้โมดูล SLAVE ต้องเป็นภาระเรื่องเวลาที่ต้องมารับข้อมูลซ้ำ ๆ มากนัก หรือหากจะทำให้มีข้อมูลซ้ำส่งเป็นช่วง ๆ ตลอดเวลาก็แล้วแต่ผู้ออกแบบ

ในการนำเอาโมดูล MASTER และ SLAVES มาต่อพ่วงให้ทำงานร่วมกัน จำเป็นต้องปรับเงื่อนไขการทำงานของตัว SLAVES ให้สอดคล้องกับโค้ดที่ได้รับจากตัว MASTER จากการออกแบบวงจรตามรูปที่ 5 สามารถวิเคราะห์สถานการณ์ทำงานของสวิทช์ควบคุมได้จากค่าข้อมูลแต่ละบิทที่พอร์ท B ตามรูปที่ 7

D7	D6	D5	D4	D3	D2	D1	D0
Parking	Brake	Head LO	Head HI	Emerg.	Reward	Turn L	Turn R

0 = ทำงาน
1 = หยุดทำงาน

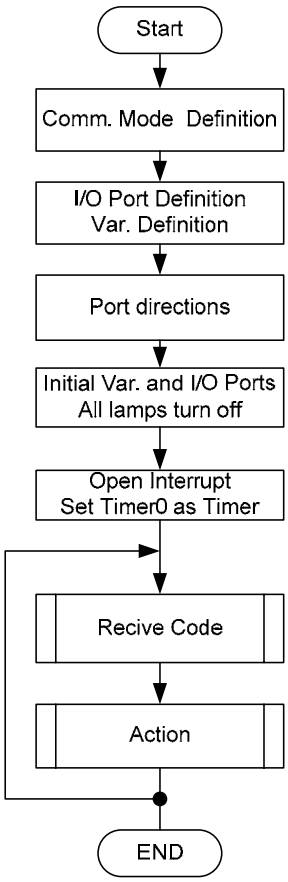
รูปที่ 7 แสดงรูปแบบโค้ดที่ส่งจากโมดูล MASTER ไปยังโมดูล SLAVES

ดังนั้น โปรแกรมทดสอบการทำงานของโมดูล SLAVE จะต้องปรับแก้ในส่วนการนำโค้ดมาตรวจสอบเงื่อนไขเพื่อให้ทำงานตามที่โมดูล MASTER ส่งมา

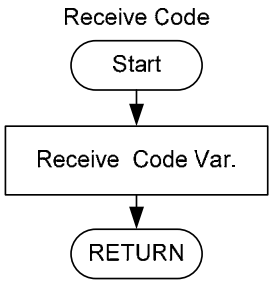
โปรแกรมคำสั่งสำหรับทดสอบการทำงานของโมดูล MASTER

```
'====================
'Master  Module
'====================
'
include "modedefs.bas"
'
parking         var  portb.7
head_LO      var  portb.5
head_HI       var  portb.4
brake           var  portb.6
emerg          var  portb.3
turn_L          var  portb.1
turn_R          var  portb.0
rwd              var  portb.2
s_out            var  portc.6
'code            var  byte
old_code       var  byte
i                  var  byte
trisa  = %000000
trisb  = %11111111
       pause 1000
       serout s_out,2,["Ready !!",10,13]
'
'----start main Program --
'
start:
       gosub scan
       gosub send
       goto start
       end
'
'--End of Main Program---
send:
       if code <> old_code then
          for i = 1 to 3
             serout s_out,2,[code,10,13]
             pause 4
          next i
          old_code = code
       endif
       return
'
scan:
       code = portb
       return
```

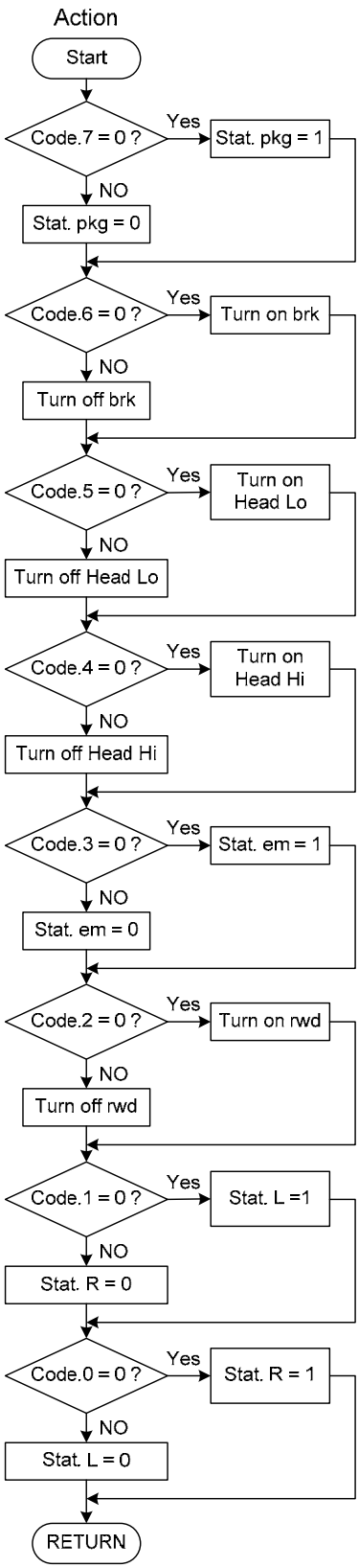
โปรแกรม Flowchart การทำงานของโมดูล Slave ที่ทำงานตามโค้ดที่โมดูล Master ส่งมา



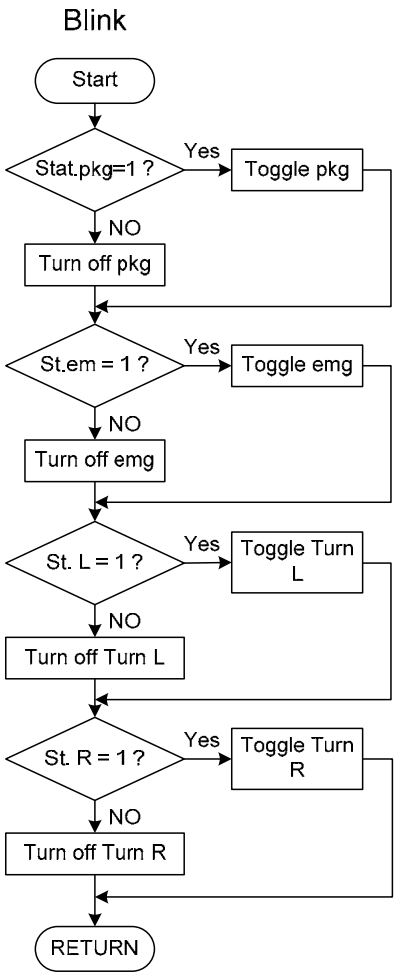
Main program ของโมดูล Slave



โปรแกรมย่อยสำหรับรับข้อมูลจาก Master



รูปที่ 8 โปรแกรม Flowchart การทำงานของโมดูล Slave



โปรแกรมย่อยส่วนสร้างไฟกระพริบ

รูปที่ 9 โปรแกรม Flowchart การทำงานของโมดูล Slave (ต่อ)

โปรแกรมคำสั่งโมดูล SLAVE ที่ปรับแก้ไขตามโค้ดที่โมดูล MASTER ส่งมา

```
'====================
'Slave  Module
'====================
'
include "modedefs.bas"
'
brk          var  portb.0
pkg          var  portb.1
rwd          var  portb.2
turn_L       var  portb.3
turn_R       var  portb.4
emerg        var  portb.5
head_LO var  portb.6
head_HI  var  portb.7
_in          var  portc.7
s_out        var  portc.6
st_p         var  bit
st_em        var  bit
st_L         var  bit
st_R         var  bit
stat         var  bit
div          var  word
code         var  byte
```

มีต่อหน้าถัดไป

```
'
trisa      = %111111
trisb      = %00000000
adcon1 = 7
'-----------------------
on interrupt  goto int
intcon        =  %10100000   'open timer0 overflow int.
option_reg =  %11000101   'open internal osc/4 prescale 64
‘
       tmr0   = 231         'pre load timer
       stat    = 0
       st_p    = 0
       st_em = 0
       st_L    = 0
       st_R   = 0
       low  brk
       low  pkg
       low  rwd
       low  turn_L
       low  turn_R
       low  emerg
       low  head_LO
       low  head_HI
‘
'----start main Program --
'
start:
       gosub recive
       gosub action
       goto start
       end
‘
'--End of Main Program---
'
       disable
int:
      div  =  div+1
      gosub recive
      if div >= 50 then     'divide by 50
          div  = 0
          gosub blink
      endif
      tmr0     = 231
      intcon.2 = 0
      resume
'------------------------------
recive:
       serin s_in,2,2,time,code
       return
'------------------------------
action:
       if code.7 = 0  then
          st_p = 1
       else
          st_p = 0
       endif
       '-------------------
       if code.6 = 0  then
          high brk
       else
          low  brk
       endif
       '-------------------
       if code.5 = 0  then
          high  head_LO
       else
          low   head_LO
       endif
```

มีต่อหน้าถัดไป

```
        '------------------
        if code.4 = 0  then
           high head_HI
        else
           low  head_HI
        endif
        '------------------
        if code.3 = 0  then
           st_em = 1
        else
           st_em = 0
        endif
        '------------------
        if code.2 = 0  then
           high rwd
        else
           low  rwd
        endif
        '------------------
        if code.1 = 0  then
           st_L = 1
        else
           st_L = 0
        endif
        '------------------
        if code.0 = 0  then
           st_R = 1
        else
           st_R = 0
        endif
        return
'
time:return
'
blink:
         if st_p = 1 then         ‘parking  sw
            toggle   pkg
         else
            low pkg
         endif
         '----------------
         if st_em = 1 then       ‘emergency  sw
            toggle   emerg
         else
            low emerg
         endif
         '----------------
         if st_L = 1 then
            toggle   turn_L
         else
            low turn_L
         endif
         '-----------------
         if st_R = 1 then
            toggle   turn_R
         else
            low turn_R
         endif
         '-----------------
         Return
'================================
```

<u>หมายเหตุ</u> การทำงานของโปรแกรมในส่วนของไฟกระพริบ ได้แก่ ไฟเลี้ยว และไฟฉุกเฉินจะแยกดวงไฟกัน แต่ในการออกแบบให้สอดคล้องกับการใช้งานจริงต้องปรับให้เหมาะสม

Experiment 45 การเขียนโปรแกรมประยุกต์การสื่อสารจากพอร์ทอนุกรมโดยใช้โมดูล USART ในตัวชิพทำงานร่วมกับการบริการอินเตอร์รัพท์ (กรณีศึกษา : ระบบสื่อสารควบคุมไฟฟ้าในรถยนต์แบบใช้พอร์ทอนุกรม)

จุดประสงค์ เพื่อเรียนรู้เทคนิคการเขียนโปรแกรมประยุกต์การใช้ฟังก์ชั่นโมดูล USART ของไมโครคอนโทรลเลอร์ สื่อสารสัญญาณควบคุมผ่านทางพอร์ทอนุกรม โดยทำงานร่วมกับการบริการอินเตอร์รัพท์

ทฤษฎีพื้นฐาน

การสื่อสารอนุกรมเป็นสิ่งจำเป็นของทั้งระบบคอมพิวเตอร์และไมโครคอนโทรลเลอร์ ซึ่งมีอยู่หลายมาตรฐานหลายแบบ เช่น RS-232C, USB, CAN, I2C เป็นต้น ปัจจุบันไมโครคอนโทรลเลอร์ได้สร้างฟังก์ชั่นโมดูลเหล่านี้เป็นฮาร์ดแวร์ฝังอยู่ในการทำงานของตัวชิพ USART (Universal Synchronous Asynchronous Receiver Transmitter) เป็นโมดูลหนึ่งที่ฝังไว้ในฮาร์ดแวร์การทำงานของไมโครคอนโทรลเลอร์ PIC16F87X ทำหน้าที่สื่อสารข้อมูลทางพอร์ทอนุกรม สามารถทำงานได้ทั้งโหมดซิงโครนัส (Synchronous) และอะซิงโครนัส (Asynchronous) ได้ 3 รูปแบบดังนี้

1) Asynchronous Full Duplex
2) Synchronous – Master Half Duplex
3) Synchronous – Slave Half Duplex

ขา I/O ที่ใช้ในการสื่อสารของโมดูล USART สำหรับไมโครคอนโทรลเลอร์ PIC ตัวถังแบบ PDIP 40 ขา คือ RC6 เป็นขา TX และ RC7 เป็นขา RX

รีจิสเตอร์ที่ใช้ควบคุมการทำงานของโมดูล USART มี 3 ตัว ได้แก่

รีจิสเตอร์ TXSTA : Transmit Status and Control Register (Address : 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in SYNC mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be parity bit

TXSTA เป็นรีจิสเตอร์ที่ใช้ควบคุม และเก็บสถานะ การทำงานในภาคส่งข้อมูล

รีจิสเตอร์ RCSTA : Receive Status and Control Register (Address: 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode:
Don't care
Synchronous mode - master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode - slave:
Don't care

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit, but must be calculated by user firmware)

RCSTA เป็นรีจิสเตอร์ที่ใช้ควบคุมและแสดงสถานการณ์ทำงานภาครับข้อมูล

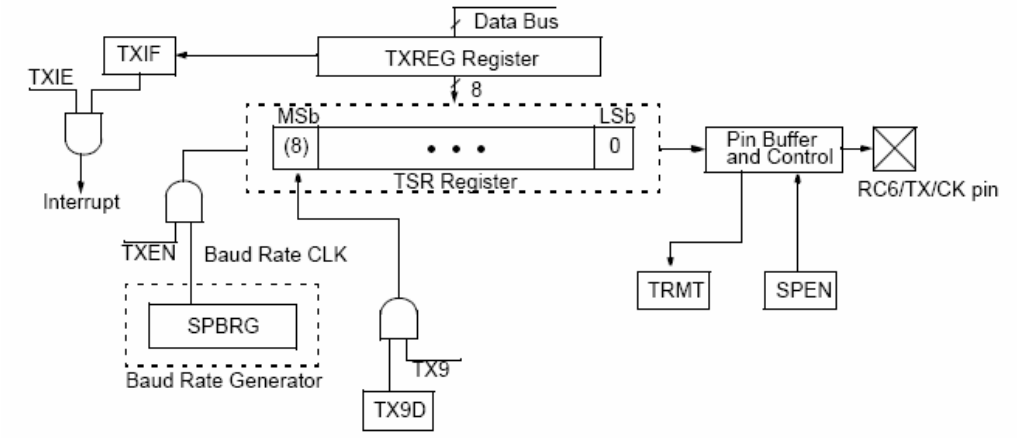
รีจิสเตอร์ที่เกี่ยวข้องอีกหนึ่งตัว คือ SPBRG เป็นรีจิสเตอร์สำหรับกำหนดค่า Baud Rate ของการส่งและรับข้อมูล

โมดูล USART สามารถใช้งานอินเตอร์รัพท์ภาคส่ง โดยจะต้องมีรีจิสเตอร์เข้ามาเกี่ยวข้องกับการทำงานดังนี้ คือ

ตารางที่ 1 การกำหนดค่าใช้งานในการส่งข้อมูล ค่าบิทที่แลเงาไว้เป็นค่าที่ไม่ได้ใช้

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

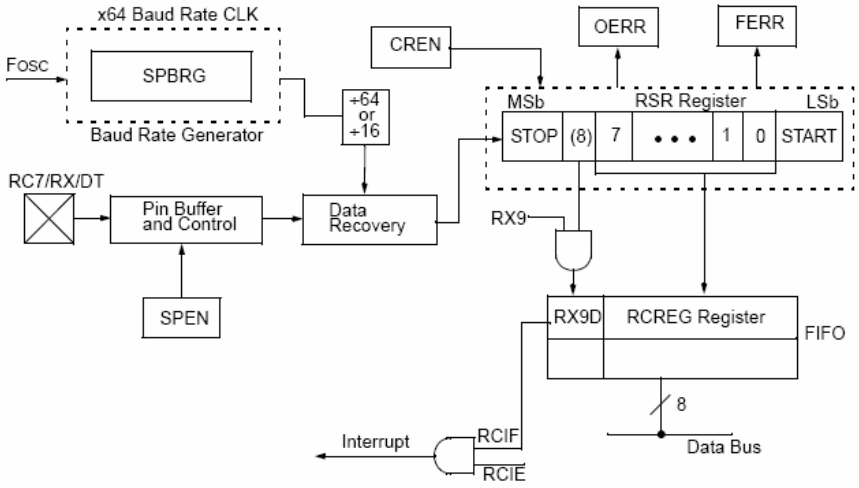
บล็อกไดอะแกรมการทำงานของภาคส่งข้อมูล



รูปที่ 1 บล็อกไดอะแกรมแสดงการทำงานของภาคส่งข้อมูลโมดูล USART

ในการส่งข้อมูลทุกอย่างจะเบ็ดเสร็จด้วยคำสั่ง SEROUT หรือ HSEROUT ในภาษา PIC BASIC PRO สามารถใช้งานได้สะดวกและมีประสิทธิภาพ และง่ายกว่าการกำหนดด้วยการเขียนโปรแกรมจัดการกับรีจิสเตอร์ TXSTA ที่มีขั้นตอนจัดการหลายขั้นตอน

บล็อกไดอะแกรมการทำงานของภาครับข้อมูล



รูปที่ 2 บล็อกไดอะแกรมแสดงการทำงานของภาครับข้อมูลโมดูล USART

ในการรับข้อมูลด้วยการกำหนดค่าในรีจิสเตอร์ควบคุม RCSTA และทำงานร่วมกับการใช้บริการอินเตอร์รัพท์ จะเป็นวิธีที่มีประสิทธิภาพการประมวลผลที่สูงกว่าการใช้คำสั่ง SERIN หรือ HSERIN เนื่องจากในการเขียนโปรแกรมเราไม่จำเป็นต้องใช้คำสั่งในการรับข้อมูลด้วยคำสั่ง SERIN หรือ HSERIN ในโปรแกรมอีกต่อไป เพียงแต่คอยเช็คสถานการณ์เกิดอินเตอร์รัพท์ที่(RCIF) เมื่อมีข้อมูลถูกส่งเข้ามาที่รีจิสเตอร์ RCREG ตามรูปที่ 2 โดยต้องเปิดอินเตอร์รัพท์ด้วยการเซ็ตบิท RCIE PEIE และ GIE ตามตารางที่ 2

ตารางที่ 2 การกำหนดค่าในรีจิสเตอร์ที่เกี่ยวข้องกับการรับข้อมูลกับการอินเตอร์รัพท์ ค่าบิทที่แลเงาไว้เป็นค่าที่ไม่ใช้งาน

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

ในการใช้โมดูล USART จำเป็นต้องกำหนดอัตราความเร็วในการรับ – ส่งให้เข้ากับอุปกรณ์ภายนอกซึ่งต้องเป็นอัตราเร็วที่เป็นมาตรฐานของการสื่อสารผ่านพอร์ท RS-232C โดยการกำหนดค่าที่เหมาะสมไว้ล่วงหน้าก่อนเปิดการรับ – ส่งที่รีจิสเตอร์ SPBRG โดยมีสูตรคำนวณดังนี้ คือ

ตารางที่ 3 สูตรคำนวณหาค่า X ที่จะใช้กำหนดในรีจิสเตอร์ SPBRG

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64(X+1))	Baud Rate = Fosc/(16(X+1))
1	(Synchronous) Baud Rate = Fosc/(4(X+1))	N/A

X = value in SPBRG (0 to 255)

ตัวอย่าง การคำนวณค่า X ที่กำหนดในรีจิสเตอร์ SPBRG สำหรับการรับส่งในโหมด Asynchronous และ CPU ใช้ความถี่ 4 MHz ต้องการอัตราความเร็วการรับส่งข้อมูล 9600 Bit / Sec ในโหมด High speed (BRGH = 1)

สูตร Baud Rate = Fosc / (64 (X+1))

9600 = 4×10^6 / (16(X+1))

(X+1) = 26.04

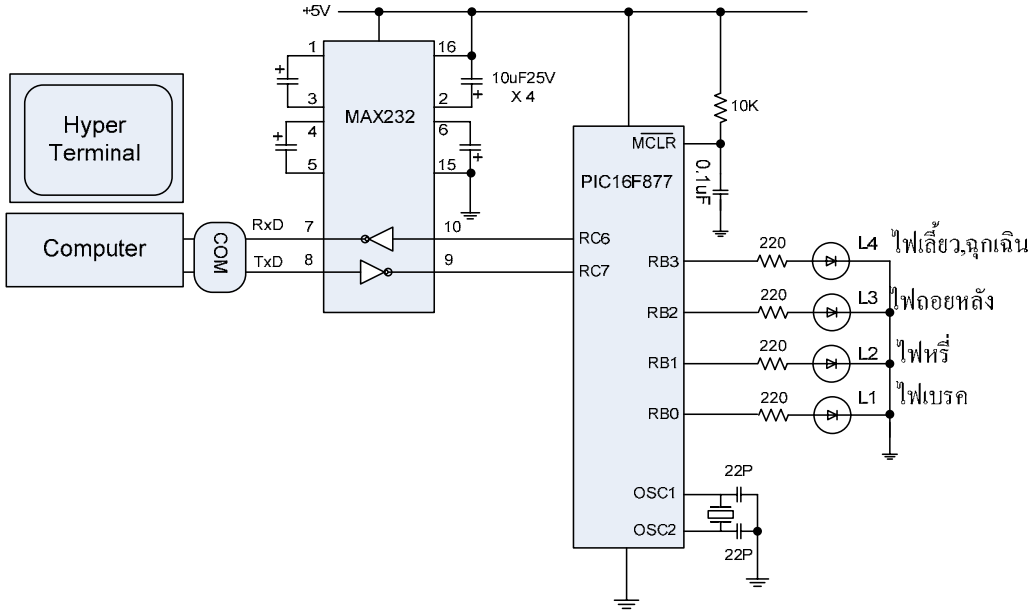
X = 25.04 หรือ 25 เป็นค่าที่ใกล้เคียงที่สุด

ลำดับขั้นตอนในการรับข้อมูลในโหมดอะซิงโครนัส

1) กำหนดค่า Baud Rate SPBRG = 25 ซึ่งจะได้ Baud Rate = 9600 Bit /sec
2) บิท SYNC = 0 , SPEN = 1
3) บิทอินเตอร์รัพท์ GIE = 1 , PEIE = 1 , RCIE = 1
4) รับในโหมด 8 บิท RX9 = 0
5) เริ่มเปิดรับ CREN = 1
6) ตรวจสอบแฟล็กอินเตอร์รัพท์ RCIF จะมีค่าเป็น 1 เมื่อการรับข้อมูลเสร็จสิ้นและเกิดอินเตอร์รัพท์
7) อ่านข้อมูล 8 บิทที่รีจิสเตอร์ RCREG นำไปใช้

บิทแฟล็ก RCIF จะถูกเซ็ตเป็น 1 เมื่อรับข้อมูล และข้อมูลเข้ามาถึงรีจิสเตอร์ RCREG เรียบร้อย เป็นบิทชนิดอ่านได้อย่างเดียว และจะถูกเคลียร์เป็น 0 เมื่อข้อมูลในรีจิสเตอร์ RCREG ถูกอ่านออกไปแล้ว กรณีที่อ่านข้อมูลออกไปไม่ทัน แล้วมีข้อมูลใหม่เข้ามา จะทำให้เกิดการชนกับข้อมูลเก่าในรีจิสเตอร์ RSR ทำให้ข้อมูลเก่าเสียหายไป แต่จะไม่ถ่ายโอนให้กับรีจิสเตอร์ RCREG แต่จะทำให้บิท OERR ถูกเซ็ตเป็น 1 และทำให้การรับข้อมูลหยุดลง เรียกความผิดพลาดนี้ว่า “Overrun Error” ต้องใช้คำสั่งเคลียร์บิท OERR ก่อน การรับข้อมูลจึงจะทำงานต่อไปได้ ความผิดพลาดที่อาจจะเกิดขึ้นได้อีกอย่างหนึ่ง คือ ความผิดพลาดเกิดจากเฟรมการรับข้อมูล โดยพบว่า Stop bit ของเฟรมได้รับเป็น 0 ซึ่งตามปกติต้องมีค่าเป็น 1 ความผิดพลาดนี้จะทำให้บิท FERR ในรีจิสเตอร์ RCSTA ถูกเซ็ตเป็น 1 แต่จะถูกเคลียร์เป็น 0 เมื่อพบว่าเฟรมต่อไปมีความถูกต้อง

วงจรทดลองตาม Experiment 45



รูปที่ 3 วงจรทดลองตาม Experiment 45

โปรแกรมคำสั่ง

```
'====================
'Slave  Module
'File : car3.pbp
'====================
'
include "modedefs.bas"
'
brk       var  portb.0
pkg       var  portb.1
rwd       var  portb.2
turn      var  portb.3
s_in      var  portc.7
s_out     var  portc.6
```

มีต่อหน้าถัดไป

```
stat      var  bit
div       var  word
code      var  byte
'
trisa       =  %111111
trisb       =  %11100000
adcon1  =  7
'-----------------------
on interrupt  goto int
intcon          =  %11100000   'open timer0 overflow and USART int.
option_reg   =   %11000101   'open internal osc/4 prescale 64
pie1.5          = 1                  'Enable USART Recieve int.
txsta           = %00000100     'Asynchronous and High speed
rcsta           = %10010000     'Open TxRx pin and Continuous Recieve
spbrg           = 25                 'set baud rate = 9600 b/s
'
       tmr0 = 251       'pre load timer
       stat = 0
       code = 0
       low  brk
       low  pkg
       low  rwd
       low  turn
'
'----start main Program --
'
start:
       goto start
       end
'
'---End of Main Program---
'
'
       disable
int:
       div = div+1
       if div >= 20 then   'divide by 20
           div  = 0
           gosub blink
       endif
       if pir1.5 = 1 then    'Rx data is completed
           code = rcreg     'Transfer data to code
           rcsta.7 = 0       'Disable USART
       endif
       gosub action
       tmr0       = 100      'Pre load Timer 0
       rcsta.7   = 1          'Enable USART module
       intcon.2 = 0          'Enable Timer int.
       resume
‘
‘
blink:
        if stat = 1 then
            toggle   turn
        else
            low turn
        endif
        return
```

มีต่อหน้าถัดไป

```
'
action:
      if  code = "1"  then
          serout s_out,2,["SystemOK",10,13]
          code = "0"
      endif
      if code = "q" then high pkg
      if code = "w" then low  pkg
      if code = "e" then high rwd
      if code = "r"  then low  rwd
      if code = "t"  then stat = 1
      if code = "y" then stat = 0
      if code = "u" then stat = 1
      if code = "i"  then stat = 0
      if code = "a" then high brk
      if code = "s"  then low  brk
      return
```

การทำงานของโปรแกรม

การทำงานของโปรแกรมทำงานเช่นเดียวกันกับใบงานทดลองที่ 44 แต่ลักษณะการเขียนโปรแกรมจะแตกต่างกันมาก คือโปรแกรมตามใบงานนี้จะใช้ขีดความสามารถพิเศษของ โมดูล USART ในการรับการสื่อสารข้อมูลสั่งการทำงานของระบบไฟฟ้าในรถยนต์ที่ถูกจำลองในบอร์ดไมโครคอนโทรลเลอร์ โดยทำงานร่วมกับการบริการอินเตอร์รัพท์ด้วยการเช็คแฟล็ก(บิท RCIF) ที่เกิดขึ้นเมื่อมีการรับข้อมูลเข้ามาเรียบร้อย ทำให้เราไม่จำเป็นต้องใช้คำสั่ง SERIN เหมือนใบงานที่ 44 ดังนั้นเราจึงไม่จำเป็นต้องเขียนคำสั่งใด ๆ ไว้ในโปรแกรมหลักเหมือนใบงานที่ 44 เพียงแต่ปล่อยให้วนลูปไว้เท่านั้น ทุกอย่างอยู่ที่การบริการอินเตอร์รัพท์ หากไม่เกิดอินเตอร์รัพท์ใด ๆ โปรแกรมจะไม่ทำอะไรเลย ซึ่งจะทำให้การตอบสนองการทำงานของโปรแกรมเป็นไปด้วยความไวสูงสุดในลักษณะที่เรียกว่า “Multi-Tasking” การทำงานโปรแกรมเริ่มด้วยการเปิดการใช้อินเตอร์รัพท์ทั้ง Timer0 เพื่อใช้สร้างไฟกระพริบ และโมดูล USART เพื่อรอรับการสื่อสารทางพอร์ทอนุกรม RS-232 การทำงานของโปรแกรมจะเกิดขึ้นเมื่อมีการอินเตอร์รัพท์ทั้งในส่วนของ Timer0 ที่เกิด Overflow และมีการส่งข้อมูลสื่อสารเข้ามาทางพอร์ทอนุกรม

```
on interrupt  goto int
intcon        =  %11100000   'open timer0 overflow and USART int.
option_reg    =   %11000101  'open internal osc/4 prescale 64
pie1.5        = 1            'Enable USART Receive int.
txsta         = %00000100    'Asynchronous and High speed
rcsta         = %10010000    'Open TxRx pin and Continuous Receive
spbrg         = 25           'set baud rate = 9600 b/s
```

รูปที่ 4 แสดงโปรแกรมส่วนของการขอเปิดใช้บริการอินเตอร์รัพท์