TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
KHOA ĐIỆN – ĐIỆN TỬ

PGS. TS NGUYỄN VĂN TIỀM
Bộ môn Điều khiển học
ĐT: 0904226592
Email: nguyenvantiem@utc.edu.vn

BÀI GIẢNG

HỆ THỐNG ĐIỀU KHIỂN NHÚNG

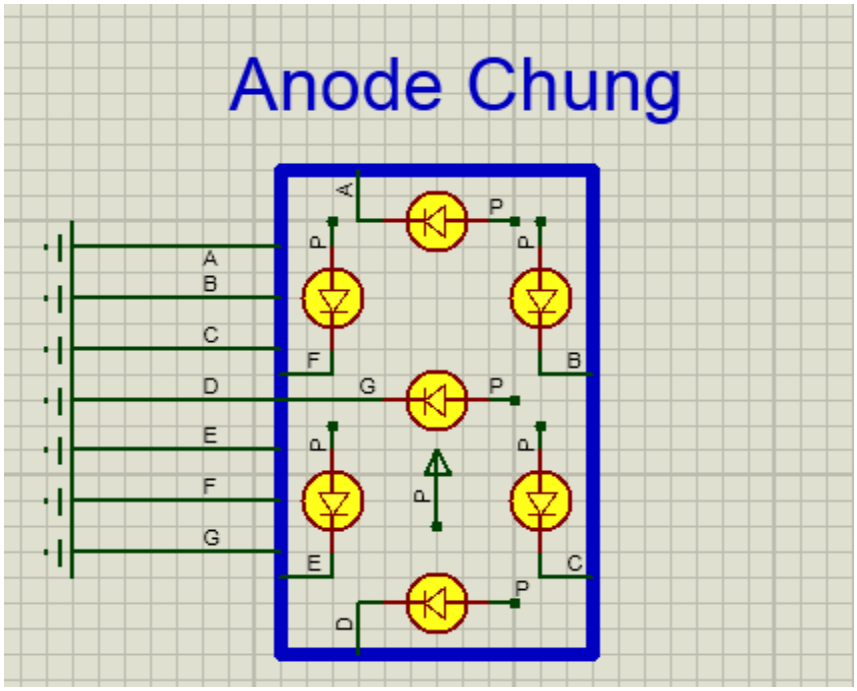
Phần 2

BÀI 1: LED 7 THANH

I. Cấu tạo

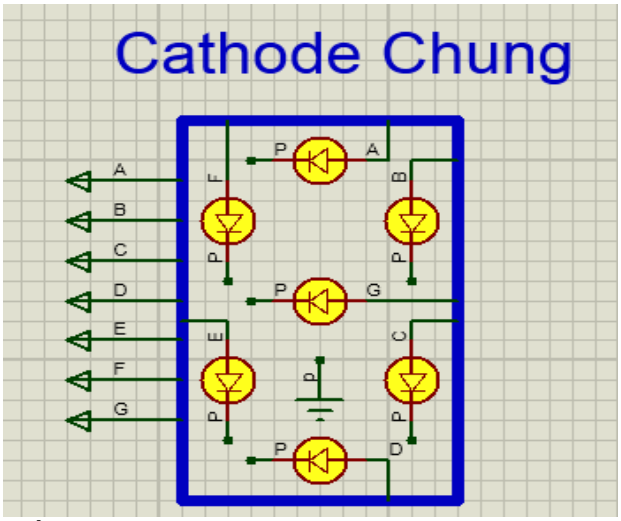
- Led 7 thanh được cấu tạo từ các led đơn xắp xếp theo các thanh nét để có thể biểu diễn chữ số hoặc kí tự đơn giản.
- Các thanh được cấu tạo bằng một led đơn, đặt tên thanh bằng các chữ cái a,b,c,d,e,f,g,h. (h là dấu chấm dot)
  - ➢ Cần 8 tín hiệu để điều khiển từng nét của led (từng diode phát quang)
- Nếu led 7 đoạn có Anode (cực +) chung, đầu chung này được nối với +Vcc, các chân còn lại dùng để điều khiển trạng thái sáng tắt của các led đơn, led chỉ sáng khi tín hiệu đặt vào các chân này ở mức 0.
- Nếu led 7 đoạn có Cathode (cực -) chung, đầu chung này được nối xuống Ground, các chân còn lại dùng để điều khiển trạng thái sáng tắt của các led đơn, led chỉ sáng khi tín hiệu đặt vào các chân này ở mức

1. Anode chung



Đối với LED Anode chung thì khi chúng ta cấp nguồn âm vào thì nó sẽ sáng led như trên hình.

1. Cathode chung



- Đối với LED Cathode chung thì khi chúng ta cấp nguồn dương cho nó thì nó sẽ sáng như trên hình.

II. Mã led 7 thanh

- Mã LED 7 đoạn có Anode chung, muốn thanh nào sáng ta xuất ra chân Cathode của LED đơn đó mức 0. Từ đó ta có bảng giải mã LED 7 đoạn Anode chung như sau:

Số	Số nhị phân								HEX
	7	6	5	4	3	2	1	0	
	dp	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	8F
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90
A	1	0	0	0	1	0	0	0	88
B	1	0	0	0	0	0	1	1	83
C	1	1	0	0	0	1	1	0	C6
D	1	0	1	0	0	0	0	1	A1
E	1	0	0	0	0	1	1	0	86
F	1	0	0	0	1	1	1	0	8E

- Mã LED 7 đoạn Cathode chung, muốn thanh nào sáng ta xuất ra chân Anode của LED đơn đó mức 1. Từ đó ta có bảng giải mã LED 7 đoạn Cathode chung như sau:

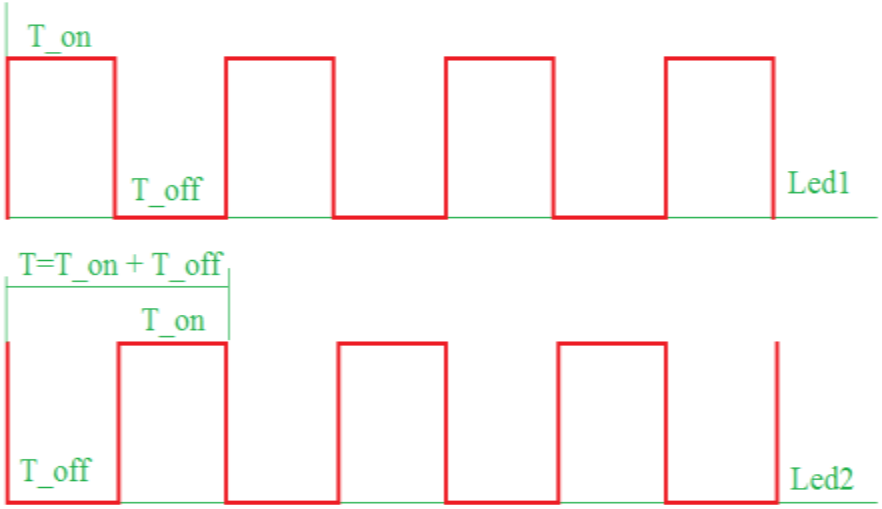
Số	Số nhị phân								HEX
	7	6	5	4	3	2	1	0	
	dp	g	f	e	d	c	b	a	
0	0	0	1	1	1	1	1	1	3F
1	0	1	1	1	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F
A	0	1	1	1	0	1	1	1	77
B	0	1	1	1	1	1	0	0	7C
C	0	0	1	1	1	0	0	1	39
D	0	1	0	1	1	1	1	0	5E
E	0	1	1	1	1	0	0	1	79
F	0	1	1	1	0	0	0	1	71

III. Giao tiếp cơ bản giữa vi điều khiển và led 7 thanh

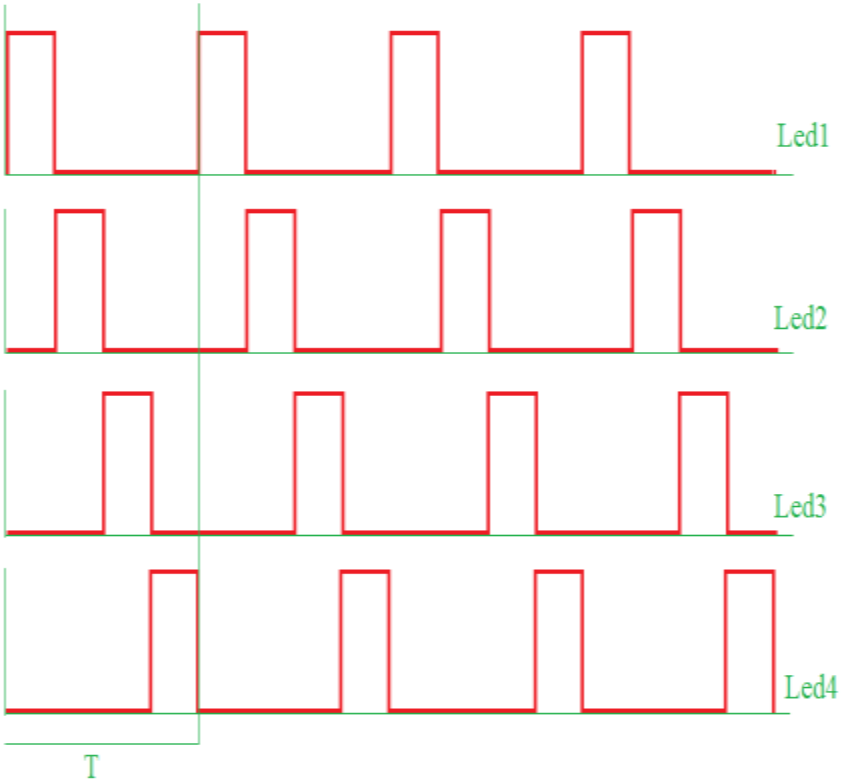
1. Phương pháp quét led:

Khác với hiển thị số,kí tự trên 1 con led 7 thanh,khi ta muốn hiển thị 2 hay nhiều led 7 thanh cùng một lúc thì thì ta phải sử dụng phương pháp quét led Việc quét led 7 thanh liên quan đến việc **quét tần số và thời gian hiển thị của 1 led.**

- Khi quét 2led 7 thanh thì sơ đồ xung của phương pháp quét như sau:



Tương tự ta có giản đồ xung khi quét 4led 7 thanh:



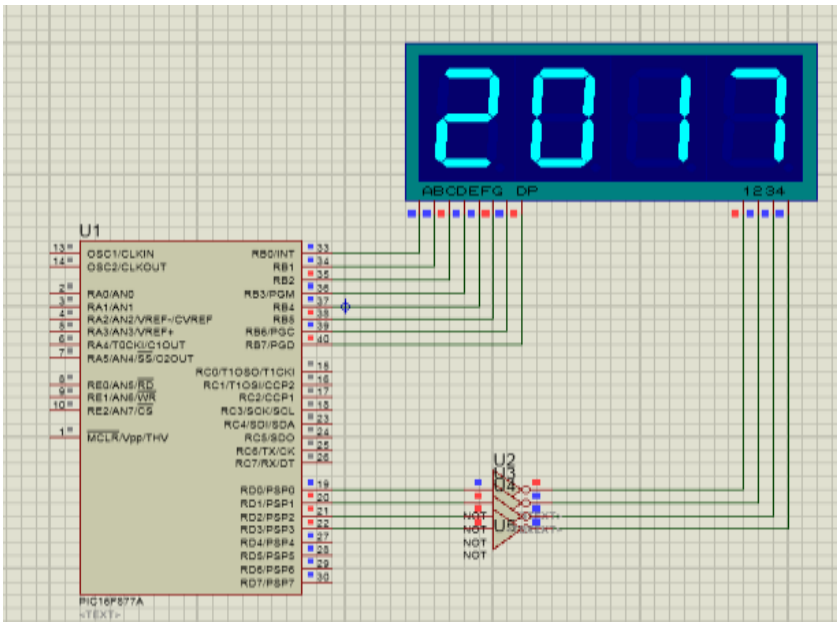
- Như vậy so sánh 2 giản đồ ta nhận thấy bản chất của phương pháp quét led là trong một thời điểm chỉ có 1 led 7thanh sáng. Hay nói cách khác là trong một chu kì T các led thay nhau lần lượt sáng.
  + Nếu quét 2 led thì 2 chân của VDK sẻ đảm nhận nhiệm vụ đóng mở điện cấp cho led còn 8 chân sẻ đưa data ra led.
  + Nếu quét 4 led thì 4 chân của VDK sẻ đảm nhận nhiệm vụ đóng mở điện cấp cho led còn 8 chân sẻ đưa data ra led.
  + Tương tự cho số led >4 nhưng phải đảm bảo đủ chân dùng VDK bằng cách này và tần số quét phù hợp.

2. **Tần số quét led:**

- Ví dụ: Nếu chúng ta bật tắt 1 bóng đèn mỗi giây 1 lần thì mắt ta sẽ cảm nhận được bóng đèn chớp tắt nhưng nếu chúng ta bật tắt bóng đèn >24 lần trong 1 giây thì mắt ta sẻ cảm nhận thấy bóng đèn luôn luôn sáng. 24 lần đấy trong 1 giây đấy gọi là tần số đóng mở đèn. Hay gọi là tần số đóng ngắt đèn là >24Hz.
- Trong phương pháp quét led thông thường quét ở tần số khoảng 50Hz đến 100Hz là phù hợp tùy theo số lượng led và tất nhiên led sẽ không bị chớp chớp.
- Với giản đồ xung quét 4 led trên với tần số quét led 50Hz (quét 50 lượt trong 1 giây) thì thời gian quét xong 4 led (1 lượt) là T=1/50=0.02 giây .Vì quét 4 led nên mỗi led sẽ sáng 0.02/4s=5ms lần lượt. Như vậy chúng ta sẽ tạo hàm trễ 5ms để làm trễ cho mỗi lần led sáng.
- Với giản đồ xung quét 2 led trên với tần số quét led 50Hz thì vậy chúng ta sẽ tạo hàm trễ 10ms

- Không quét với tần số >100hz. Tần số càng lớn thì thời gian sáng của 1 led sẽ nhỏ lại và không đủ điều kiện cho led sáng (vì chưa sáng nổi đã tắt) hay led sẽ mờ đi không được đẹp.

Ví dụ: hiển thị năm 2017 lên led 7 thanh




```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = ON
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000
```

//khai báo mảng mã quét led anode và cathode hiển thị các số {0,1,2,3,4,5,6,7,8,9}. Khi dùng loại nào thì lôi mã tương ứng vào chương trình con

```
unsigned char Anode[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
unsigned char Cathode[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90};
unsigned char nghin, tram, chuc, donvi;
void hien_thi(unsigned int value) {
```

//tách các chữ số ra lấy giá trị của một hàng.

```
    nghin = value / 1000;
    tram = (value - nghin * 1000) / 100;
    chuc = (value - nghin * 1000 - tram * 100) / 10;
    donvi = value % 10;
//hiển thị chữ số hàng nghìn
    PORTD = 0b00001110;      //
    PORTB = Cathode[nghin]; // gán địa chỉ PORTB= mã led ở vị trí hàng nghìn
    __delay_ms(5);                //
//hiển thị chữ số hàng trăm
    PORTD = 0b11111101;
    PORTB = Cathode[tram];
    __delay_ms(5);
//hiển thị chữ số hàng chục
    PORTD = 0b11111011;
    PORTB = Cathode[chuc];
    __delay_ms(5);
//hiển thị chữ số hàng đơn vị
    PORTD = 0b11110111;
    PORTB =Cathode[donvi];
    __delay_ms(5);
}
void main() {
    TRISB = 0;
    TRISD = 0;
    PORTB = 0xff;
    PORTD = 0xff;
    while (1) {
        hien_thi(2017);
    }
}
```

GIẢI THÍCH: hàm __delay_ms(5):

VD nhỏ : Nếu chúng ta bật tắt 1 bóng đèn mỗi giây 1 lần thì mắt ta sẻ cảm nhận được bóng đèn chớp tắt nhưng nếu chúng ta bật tắt bóng đèn >24 lần trong 1 giây thì mắt ta sẻ cảm nhận thấy bóng đèn luôn luôn sáng. 24 lần đấy trong 1 giây đấy gọi là tần số đóng mở đèn. Hay gọi là tần số đóng ngắt đèn là >24Hz.

- Trong phương pháp quét led thông thường quét ở tần số tầm 50Hz đến 100Hz là đẹp tùy theo số lượng led và tất nhiên led sẽ không bị chớp chớp.
- Với giản đồ xung quét 4 led trên với tần số quét led 50Hz (quét 50 lượt trong 1 giây) thì thời gian quét xong 4 led (1 lượt) là T=1/50=0.02 giây .Vì quét 4 led nên mỗi led sẻ sáng 0.02/4s=5ms lần lượt. Như vậy chứng ta sẻ tạo hàm trể 5ms để làm trể cho mỗi lần led sáng.

Ví dụ 2: dùng led 7 hiển thị giá trị tăng dần từ 0-100

Tương tự như ví dụ trên, ta chỉ cần cho khai báo một biến int count=0.

Như vậy ở đây ta sử dụng lệnh count++ để đếm từng phần tử của mảng cho đến tới phần tử thứ 100 rồi quay lại 0. Quá trình sẽ được lặp đi lặp lại.

….

```
void main() {
   TRISB = 0;
   TRISD = 0;
   PORTB = 0xff;
   PORTD = 0xff;
   while (1)
   {
      hien_thi(count);
      count++;
      if (count == 100)
      {
         count = 0;
      }
   }
```

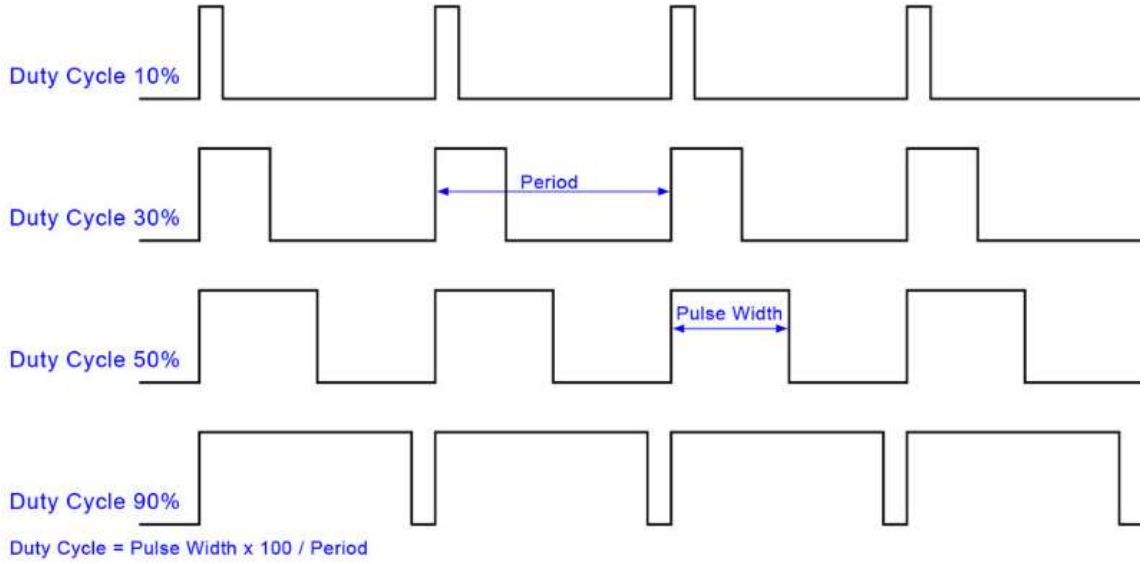
BÀI 2: PWM PIC16F877A

I. Tổng quan về PWM

1. Định nghĩa:

Phương pháp điều xung PWM (Pulse Width Modulation) là phương pháp điều chỉnh điện áp ra tải, hay nói cách khác, là phương pháp điều chế dựa trên sự thay đổi độ rộng của chuỗi xung vuông, dẫn đến sự thay đổi điện áp ra.

Các PWM khi biến đổi thì có cùng 1 tần số và khác nhau về độ rộng của sườn dương hay sườn âm.



2. Các phương pháp điều chế xung pwm

Để tạo được ra PWM thì hiện nay có hai cách thông dụng :

- Bằng phần cứng:có thể tạo bằng phương pháp so sánh hay là từ trực tiếp từ các IC dao động tạo xung vuông như : 555, LM556...
- Bằng phần mềm: Trong phần mềm được tạo bằng các chip có thể lập trình được. Tạo bằng phần mềm thì độ chính xác cao hơn là tạo bằng phần cứng. Nên người ta hay sử dụng phần mềm để tạo PWM

3. Ứng dụng

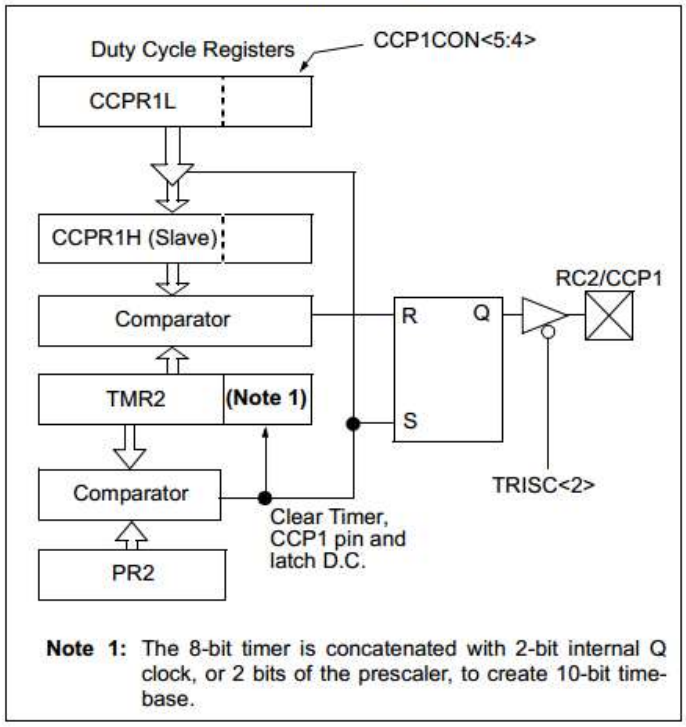
- ***Trong động cơ*** : điều khiển tốc độ động cơ
- ***Trong các bộ biến đổi xung áp:*** điều chỉnh dòng điện và điện áp ra tải.Bộ biến đổi xung áp có nhiều loại như là biến đổi xung áp nối tiếp và bộ biến đổi xung áp song song.
- Ngoài những cái trên thì PWM còn được sử dụng trong các bộ chuyển đổi DC -AC , hay trong biến tần, nghịch lưu.

II. PWM trong pic16f877A

- PWM (Pulse -Width Modulation) có nghĩa là điều chế độ rộng xung. Xung điều chế sẽ được đưa ra các pin CCP tương ứng
- Trong PIC 16F877A có 2 bộ PWM là CCP1 và CCP2.

1. Sơ đồ khối của pwm trong PIC16f877A:

FIGURE 8-3: SIMPLIFIED PWM BLOCK DIAGRAM

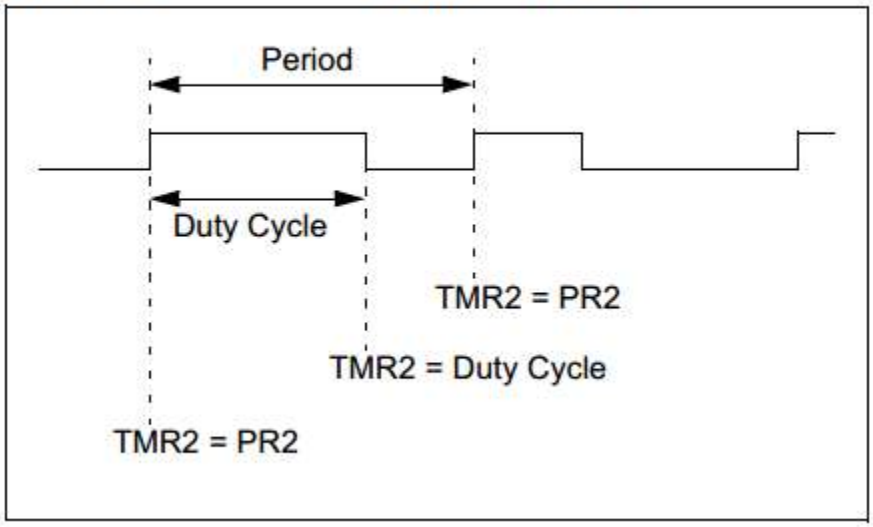


- Khối PWM gồm có 2 mạch so sánh: mạch so sánh 8 bit với mạch so sánh 10 bit. Mạch so sánh 8 bit sẽ so sánh giá trị đếm của timer2 với giá trị của thanh ghi PR2 (period register),giá trị trong timer2 tăng từ giá trị đặt trước cho đến khi bằng giá trị của PR2 thì mạch so sánh sẽ set đầu ra của mạch flip-flop-RS
  RS làm ngõ ra CCPx lên mức 1. Đồng thời nạp giá trị 10 bit từ thanh ghi CCPRxL

sang thanh ghi CCPRxH, timer2 bị reset và bắt đầu đếm lại cho đến khi giá trị của timer2 bằng giá trị của CCPRxH thì mạch so sánh
sẽ reset flip flop RS làm ngõ ra CCPx về mức 0. Quá trình này lặp lại.

2. Các tham số của PWM

FIGURE 8-4: PWM OUTPUT



- **Chu kì xung(pwm period)**

PWM period = [(PR2)+1]*4*T_{OSC}*(giá trị bộ chia tần số của TMR2).

TOSC : là chu kỳ của thạch anh tạo dao động=20M
Khi giá trị của timer 2 (TMR2) bằng giá trị của thanh ghi PR2 thì 3 sự kiện theo sau sẽ xảy ra:
· Thanh ghi TMR2 bị xóa
· Tín hiệu ngõ ra CCPx lên mức 1, ngoại trừ hệ số chu kỳ bằng 0% thì CCPx vẫn ở mức 0.
· Hệ số chu kỳ PWM được chuyển từ thanh ghi CCPRxL sang thanh ghi CCPRxH

- **Độ rộng xung (pwm duty cycle)**

Hệ số chu kỳ được thiết lập bởi giá trị lưu trong thanh ghi 10 bit gồm CCPRxL 8 bit và 2 bit còn lại là
bit thứ 4 và thứ 5 ở trong thanh ghi CCPxCON – kí hiệu là CCPxCON<5:4>.
Giá trị của hệ số chu kỳ là 10 bit nên có thể thay đổi từ 0 đến 1023 tạo ra 1024 cấp giá trị điều khiển.
Giá trị 10 bit thì 8 bit có trọng số lớn lưu trong thanh ghi CCPRxL và 2 bit còn lại có trọng số thấp thì ở CCPxCON<5:4>.
Hệ số chu kỳ của PIC16F877A được tính theo công thức:

PWM duty cycle = (CCPRxL:CCPxCON<5:4>)*T_{OSC}*(giá trị bộ chia tần số TMR2)

3. Tìm hiểu về thanh ghi

- Thanh ghi CCPxCON: (CCP1CON VÀ CCP2CON)

REGISTER 8-1: CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS: 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 Unimplemented: chưa được thực hiện, cài đặt =0
bit 5-4 CCPxX:CCPxY: PWM Least Significant bits
2bit MSB chứa giá trị tính độ rộng xung(duty cycle) của pwm. (8bit còn lại chứa trong thanh ghi CCPRxL)
bit 3-0 CCPxM3:CCPxM0: CCPx chọn chế độ hoạt động
0000 = Capture/Compare/PWM disabled
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin isunaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode
ở đây, ta dùng chế độ pwm nên pwm mode= 11xx. Nghĩa là ép buộc CCPxM3=1;CCPxM2=1; còn CCPxM1,CCPxM0 có thể thay đổi tùy ý mà không ảnh hưởng tới chế độ pwm

- **Thanh ghi T2CON**

ở chế độ pwm, ta cần chú ý tới giá trị bộ chia tần số prescaler của Timer2

REGISTER 7-1: T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Bit7: không quan tâm và mặc định bằng 0
bit 6-3 TOUTPS3:TOUTPS0: Timer2 Output Postscale Select bits: các bit này chọn tỷ số chia tần cho postscale
0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•••
1111 = 1:16 Postscale
bit 2 :**TMR2ON**: chọn chế độ timer
1 = Timer2 is on
0 = Timer2 is off

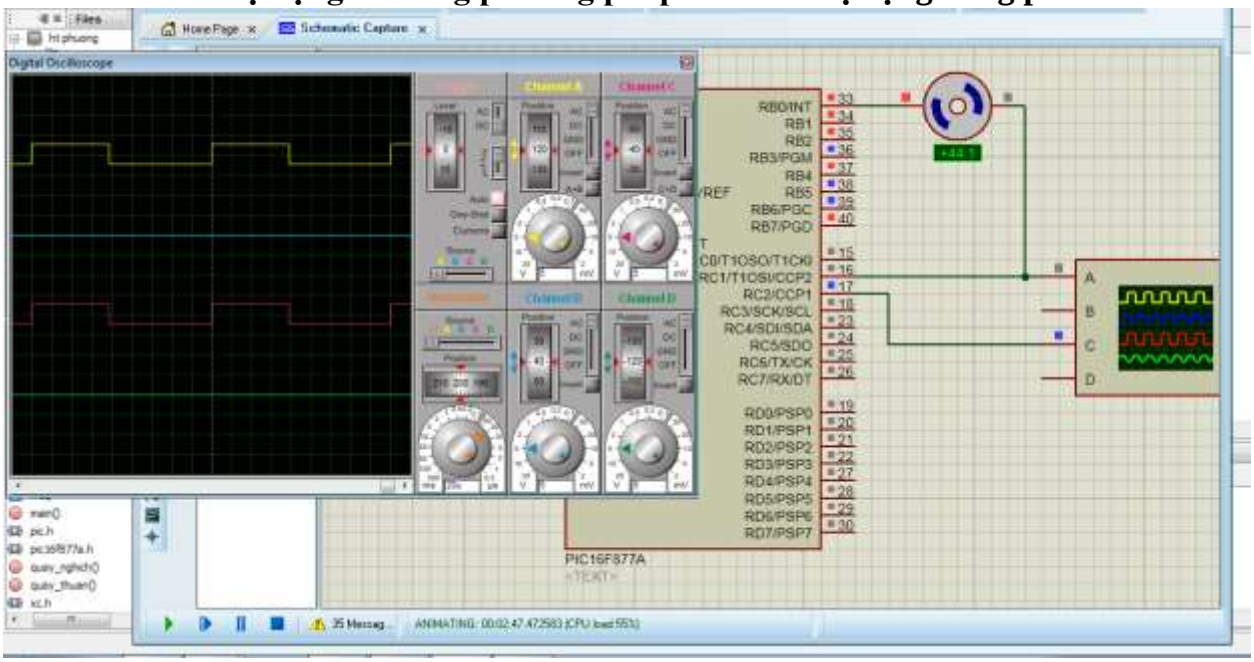
bit 1-0 :T2CKPS1:T2CKPS0: chọn bộ chia tần số Prescaler
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

4. Các bước cài đặt bộ pwm

- **Bước 1**: thiết lập thời gian của một chu kì của xung điều chế cho pwm(period) bằng cách đưa giá trị thích hợp vào thanh ghi PR2
- **Bước 2**: Thiết lập độ rộng xung cần điều chế (duty cycle) bằng cách đưa giá trị vào thanh ghi CCPRxL và các bit CCP1CON<5:4>.
- **Bước 3**: điều khiển các pin của CCP là output bằng cách clear các bit tương ứng trong thanh ghi TRISC.
- **Bước 4**: thiết lập giá trị bộ chia tần số prescaler của Timer2 và cho phép Timer2 hoạt động bằng cách đưa giá trị thích hợp vào thanh ghi T2CON.
- **Bước 5**: cho phép CCP hoạt động ở chế độ pwm

5. Ví dụ

Điều khiển tốc độ động cơ dùng phương pháp điều chế độ rộng xung pwm




```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = ON
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
```

```
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000
#define TMR2PRESCALE 4
long freq;
int PWM_Max_Duty()
{
return (_XTAL_FREQ / (freq * TMR2PRESCALE);
}
```
//tạo chương trình con PWM1_Init(long fre), PWM2_Init(long fre): cài đặt tần số băm xung
```
PWM1_Init(long fre)
{
   PR2 = (_XTAL_FREQ / (fre * 4 * TMR2PRESCALE)) - 1;// 
```
tính PR2 theo công thức tính chu kì xung
```
        freq = fre;
}
PWM2_Init(long fre) {
   PR2 = (_XTAL_FREQ / (fre * 4 * TMR2PRESCALE)) - 1;
        freq = fre;
}
```

//cài đặt hàm pwm
```
PWM1_Start() {
```
// chọn chế độ pwm mode cho chân CCP1 trên thanh ghi CCP1CON
```
   CCP1CONbits.CCP1M3 = 1;
   CCP1CONbits.CCP1M2 = 1;
```
//chọn bộ chia tần chọn bộ chia tần số Prescaler=1;4;16. ở đây ta viết chung cho tất cả 3 trường hợp. và chỉ việc thay đổi phần define TMR2PRESCALE bên trên

```
#if TMR2PRESCALAR == 1
        T2CKPS0 = 0
        T2CKPS1 = 0
#elif TMR2PRESCALAR == 4
        T2CKPS0 = 1;
        T2CKPS1 = 0;
#elif TMR2PRESCALAR == 16
        T2CKPS0 = 1;
        T2CKPS1 = 1;
#endif
        TMR2ON = 1;  // 
```
kích hoạt timer2
```
        TRISC2 = 0;    //
```
cho phép ccp1 hoạt động

```
}

PWM2_Start() {
   CCP2CONbits.CCP2M3 = 1;
   CCP2CONbits.CCP2M2 = 1;
#if TMR2PRESCALE == 1
          T2CKPS0 = 0;
          T2CKPS1 = 0;
#elif TMR2PRESCALE == 4
          T2CKPS0 = 1;
          T2CKPS1 = 0;
#elif TMR2PRESCALE == 16
          T2CKPS0 = 1;
          T2CKPS1 = 1;
#endif
          TMR2ON = 1;
          TRISC1 = 0;
}
```
//cài đặt chọn chân pwm và độ rộng xung
```
void duty_cycle(unsigned char ccp, unsigned int duty) {
   switch (ccp) {
      case 1:
      {
         if (duty < 1024) {
            duty = ((float) duty / 1023) * PWM_Max_Duty();
                  CCP1X = duty & 2;
                  CCP1Y = duty & 1;
                  CCPR1L = duty >> 2;
         }
      }
         break;
      case 2:
      {
         if (duty < 1024) {

            duty = ((float) duty / 1023) * PWM_Max_Duty();
                  CCP2X = duty & 2;
                  CCP2Y = duty & 1;
                  CCPR2L = duty >> 2;
         }
      }
         break;
   }
```

```
}
void quay_thuan()
{
   duty_cycle(2, 100);
   RB0=1;
}
void quay_nghich()
{
      duty_cycle(2, 100);
      RB0=0;
}
void main() {
         TRISB=0X00;
         PWM1_Init(1000);
         PWM2_Init(1000);
         PWM1_Start();
         PWM2_Start();
   while (1) {
      quay_thuan();
   }
}
```

LƯU Ý: trên đây là bài toán tổng thể khi sử dụng pwm1,pwm2. Với mỗi bài toán bất kì, ta chỉ cần thay đổi các thông số như bộ chia tần số,tần số băm xung trong hàm pwm_init(???)

BÀI 3: NGẮT

I. Tổng quan về ngắt

Ngắt hiểu theo nghĩa đơn giản là các sự kiện ngẫu nhiên làm gián đoạn quá trình của một sự kiện đang xảy ra.

- **Ví dụ**: Trong giờ học trên lớp, ta đang học bài, có chuông điện thoại ,ta phải dừng hoạt động học bài lại để trả lời điện thoại. Sự kiện điện thoại reo chuông được gọi là sự kiện ngắt, việc ta trả lời điện là chương trình phục vụ ngắt. Việc đang học bài được xem là chương trình chính.
  Ngắt được thực hiện khi và chỉ khi cài đặt cho phép nó. Như trong ví dụ trên, nếu sự kiện ngắt- điện thoại reo xảy ra, nếu giáo viên và bản thân ta cho phép mình trả lời điện thoại khi đang học bài thì khi có điện thoại ta mới nghe.
- Vi điều khiển cũng có ngắt. Cách xử lý của nó cũng tương tự như ví dụ trên. Hoạt động của vi điều khiển khi có sự kiện ngắt xảy ra và ngắt đó đã được cho phép:
  - Thực hiện nốt lệnh đang thực hiện

- Dừng chương trình đang thực hiện
- Lưu lại địa chỉ của lệnh kế tiếp trong chương trình đang thực hiện vào bộ nhớ stack
- Nhảy tới địa chỉ 0x04 trong bộ nhớ chương trình
  Tại đây, vi điều khiển sẽ thực hiện chương trình con phục vụ ngắt do người lập trình đã lập trình từ trước.
- Sau khi thực hiện xong chương trình con phục vụ ngắt, vi điều khiển lấy lại địa chỉ của lệnh kế tiếp đã được lưu và thực hiện tiếp chương trình đang thực hiện dở lúc chưa có ngắt

- Các thuật ngữ dùng cho xử lý ngắt trong vi điều khiển:
  - Nguồn ngắt: nguồn ngắt là nguyên nhân gây ra ngắt.
  - Sự kiện ngắt: khi nguồn ngắt xảy ra
  - Chương trình con phục vụ ngắt: là chương trình vi điều khiển xử lý khi có sự kiện ngắt xảy ra do người lập trình lập trình ra
  - Bit cho phép ngắt: tức việc cho phép vi điều khiển chạy chương trình con phục vụ ngắt khi có sự kiện ngắt xảy ra. Trong vi điều khiển PIC, mỗi ngắt có bit cho phép của nó. Bit này tận cùng bằng chữ E (enable), nằm trong các thanh ghi chuyên dụng. Muốn cho phép ngắt đó, ta phải đưa bit cho phép ngắt tương ứng lên giá trị 1. Ngắt chỉ thực sự được cho phép ngắt khi ta cho bit cho phép ngắt toàn cục GIE (Global Interrupt Enable) lên mức 1. Một số các ngắt khác, như các ngắt ngoại vi bao gồm ADC, PWM v.v Muốn cho phép nó còn phải đưa bit cho phép ngắt ngoại vi lên mức 1.
  - Cờ ngắt: là bit phản ánh trạng thái của sự kiện ngắt. Mỗi ngắt có một bit cờ. Khi bit cờ này bằng 1 nghĩa là sự kiện ngắt tương ứng với cờ đó xảy ra. Các bit này tận cùng bằng từ F (Flag- cờ).
    Lưu ý là dù một ngắt có được cho phép hay không thì cờ ngắt vẫn được set lên 1 khi có sự kiện ngắt xảy ra.

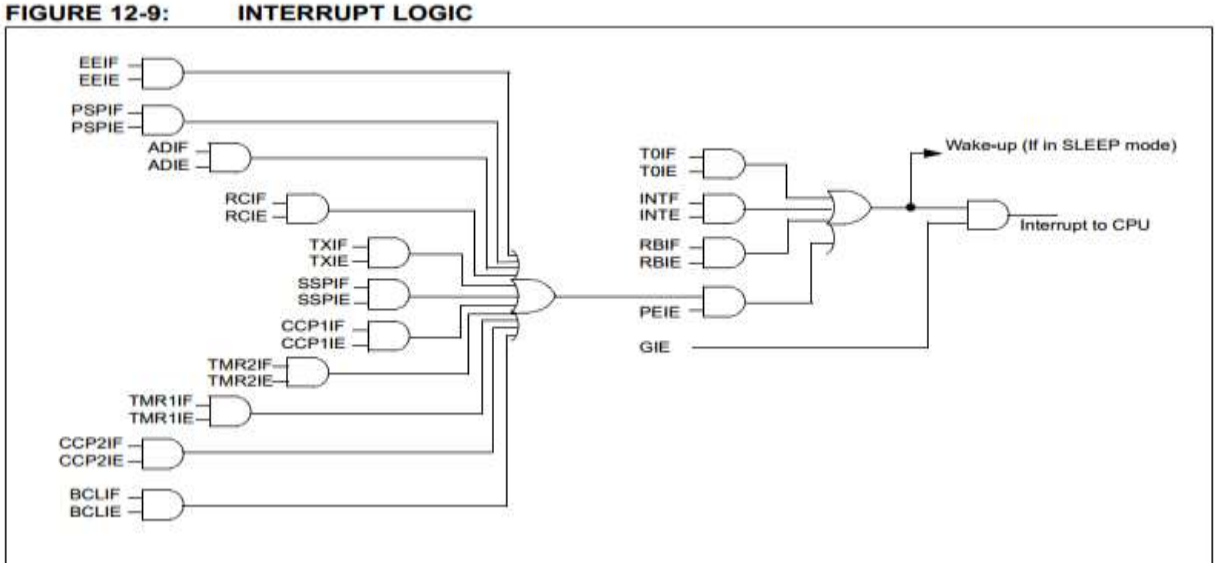
II. Ngắt trên Pic

Vi điều khiển PIC16F877A có 15 nguồn ngắt. Được chia làm 2 lớp ngắt:

- **Lớp ngắt cơ bản:** bao gồm các ngắt cơ bản như ngắt tràn timer 0, ngắt ngoài, ngắt thay đổi trạng thái của các chân PortB (RB4-RB7). Bit cho phép ngắt và bit cờ tương ứng là TMR0IE,TMR0IF; INTE, INTF; RBIE và RBIF. Để ý là để cho phép ngắt thực sự xảy ra phải có bit cho phép ngắt toàn cục GIE.
- **Lớp ngắt ngoại vi**: bao gồm các ngắt ngoại vi như ngắt tràn timer 1 (TMR1IE, TMR1IF), ngắt tràn Timer 2(TMR2IE, TMR2IF), ngắt hoàn thành việc chuyển đổi ADC (ADCIE, ADCIF), ngắt hoàn thành việc nhận kí tự trong truyền thông RS232 (RCIE, RCIF), ngắt hoàn thành việc truyền kí tự trong truyền thông RS232 (TXIE, TXIF) v.v Để ý là muốn thực sự cho phép các ngắt này ngoài bit cho phép ngắt toàn cục được set phải set cả bit cho phép ngắt ngoại vi PEIE.

Sơ đồ logic của ngắt:

FIGURE 12-9: INTERRUPT LOGIC



1. Ngắt ngoài

Hoạt động:

- Trong PIC 16F877A có 1 ngõ vào ngắt ngoài tại chân INT (RB0). Khi có một sự kiện trên chân INT (cạnh xuống hoặc cạnh lên) sẽ sinh ra một ngắt (nếu được cho phép).
- Nguồn ngắt: là xung đi vào chân RB0 của vi điều khiển PIC
- Sự kiện ngắt: sự kiện ngắt xảy ra khi có xung đi vào chân RB0 của vi điều khiển. Xung là xung sườn dương hay sườn âm phụ thuộc bit cài đặt chọn dạng xung, bit INTEDG (bit 6 của thanh ghi PTION_REG) là 1 hay 0.
- Bit cho phép ngắt: Để cho phép ngắt ngoài, bit cho phép ngắt ngoài INTIE (bit 4 của thanh ghi INTCON) phải được set lên 1. Ngoài ra, bit cho phép ngắt toàn cục GIE (bit 7 của thanh ghi INTCON) cũng phải được set lên 1.
- Cờ ngắt: bit cờ ngắt ngoài là bit INTIF (bit 1 của thanh ghi INTCON) được tự động set lên 1 khi có sự kiện ngắt ngoài xảy ra. Cờ này phải được xóa bằng chương trình (cụ thể là trong chương trình con phục vụ ngắt) để vi điều khiển quản lý chính xác các lần ngắt kế tiếp.

  Để sử dụng chức năng ngắt ngoài ta phải làm như sau:

. chọn chân RB0 là chân đầu vào (TRISRB0=1)

· Chọn kiểu ngắt: (cạnh lên hoặc cạnh xuống).

· Kích hoạt ngắt ngoài.

· Kích hoạt ngắt toàn cục.

- **Các thanh ghi liên quan**

- Thanh ghi INTCON

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7: GIE: Global Interrupt Enable bit: **ngắt toàn cục**
1 = Enables all unmasked interrupts: cho phép ngắt toàn cục
0 = Disables all interrupts: không cho phép ngắt
bit 6: PEIE: **ngắt cục bộ**
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupt
bit 5: T0IE: TMR0 Overflow Interrupt Enable bit: **ngắt tràn timer0**
1 = Enables the TMR0 interrupt// cho phép
0 = Disables the TMR0 interrupt// không cho phép
bit 4: INTE: RB0/INT External Interrupt Enable bit //**bit cho phép ngắt ngoài trên RB0**
1 = Enables the RB0/INT external interrupt// **cho phép**
0 = Disables the RB0/INT external interrupt// **không cho phép**
bit 3: RBIE: RB Port Change Interrupt Enable bit **// ngắt thay đổi trạng thái của các chân PortB (RB4-RB7).**
1 = Enables the RB port change interrupt// **cho phép**
0 = Disables the RB port change interrupt// **không cho phép**
bit 2: T0IF: TMR0 Overflow Interrupt Flag bit// **cờ báo ngắt khi tràn timer0**
1 = TMR0 register has overflowed
0 = TMR0 register did not overflow
bit 1: INTF: RB0/INT External Interrupt Flag bit**// cờ báo ngắt ngoài**
1 = The RB0/INT external interrupt occurred // **ngắt ngoài xảy ra**
0 = The RB0/INT external interrupt did not occur**// không xảy ra**
bit 0 **RBIF**: RB Port Change Interrupt Flag bit// **cờ báo ngắt thay đổi trạng thái portb**
1 = At least one of the RB7:RB4 pins changed state**// ít nhất 1 trong 4 chân thay đổi trạng thái**
0 = None of the RB7:RB4 pins have changed state**// không chân nào thay đổi trạng thái**

- **Thanh ghi OPTION**

REGISTER 2-2: OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 6 :**INTEDG**: Interrupt Edge Select bit //**bit chọn chế độ ngắt cạnh lên hay cạnh xuống**
1 = Interrupt on rising edge of RB0/INT pin **// ngắt cạnh lên**
0 = Interrupt on falling edge of RB0/INT pin **// ngắt cạnh xuống**
Các bit còn lại phục vụ chức năng khác:
Bit 7: RBPU: PORTB pull-up enable bit
=1:Không cho phép chức năng pull-up của portB

=0: Cho phép

Bit 5: TOCS Timer0 Clock Source select bit

=1: clock lấy từ chân RA4/TOCKI

=0: Dùng xung clock bên trong

Bit 4: TOSE Timer0 Source Edge Select bit

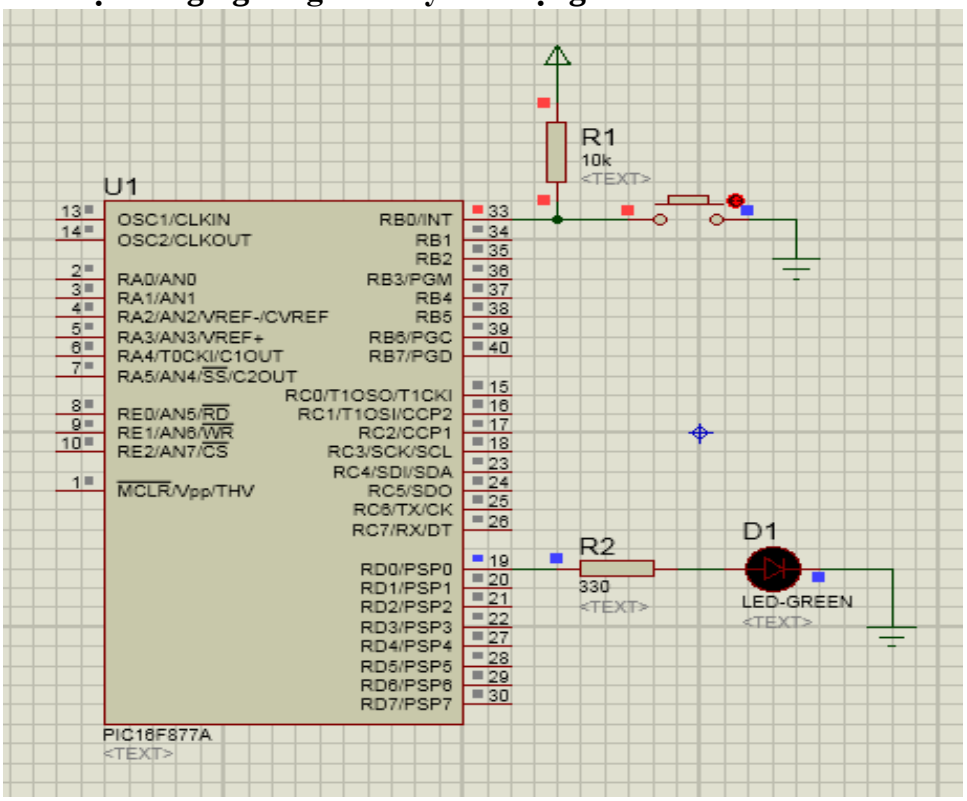
=1: tác động cạnh lên

=0: tác động cạnh xuống

Bit 3: PSA Prescaler Assignment Select bit: chọn bộ chia tần

Bit 2-0: PS2:PS0 Prescaler Rate Select bit: thiết lập tỉ số chia tần

- **Ví dụ: dùng ngắt ngoài thay đổi trạng thái của chân RD0**




```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = ON
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
```

```
#define _XTAL_FREQ 20000000
    void interrupt ISR()
    {
        if(INTCONbits.INTF==1) //khi có cờ báo ngắt
        {
          {
            PORTDbits.RD0 =!PORTDbits.RD0; //đảo trạng thái chân RD0
            INTCONbits.INTF=0;// Xóa cờ báo ngắt
          }
        }

    }
    void main ()
    {
      TRISD0 = 0;
      PORTD = 0X00;
      TRISBbits.TRISB0 = 1;
      PORTB = 0X00;
      INTCONbits.GIE = 1;// ngat toan cuc
      INTCONbits.INTE = 1;// ngat cuc bo
      OPTION_REGbits.INTEDG = 1;//ngắt cạnh lên
      while(1)
      {
      }
    }
```

Như vậy, trong một bài toán dùng ngắt ngoài, ta áp dụng giống chương trình trên và thay đổi trong chương trình ngắt.

Một số ngắt còn lại như ngắt timer0, timer1 timer2 sẽ đề cập ở bài timer

BÀI 4: TIMER 0/COUNTER 0/NGẮT TIMER 0

I. TIMER0

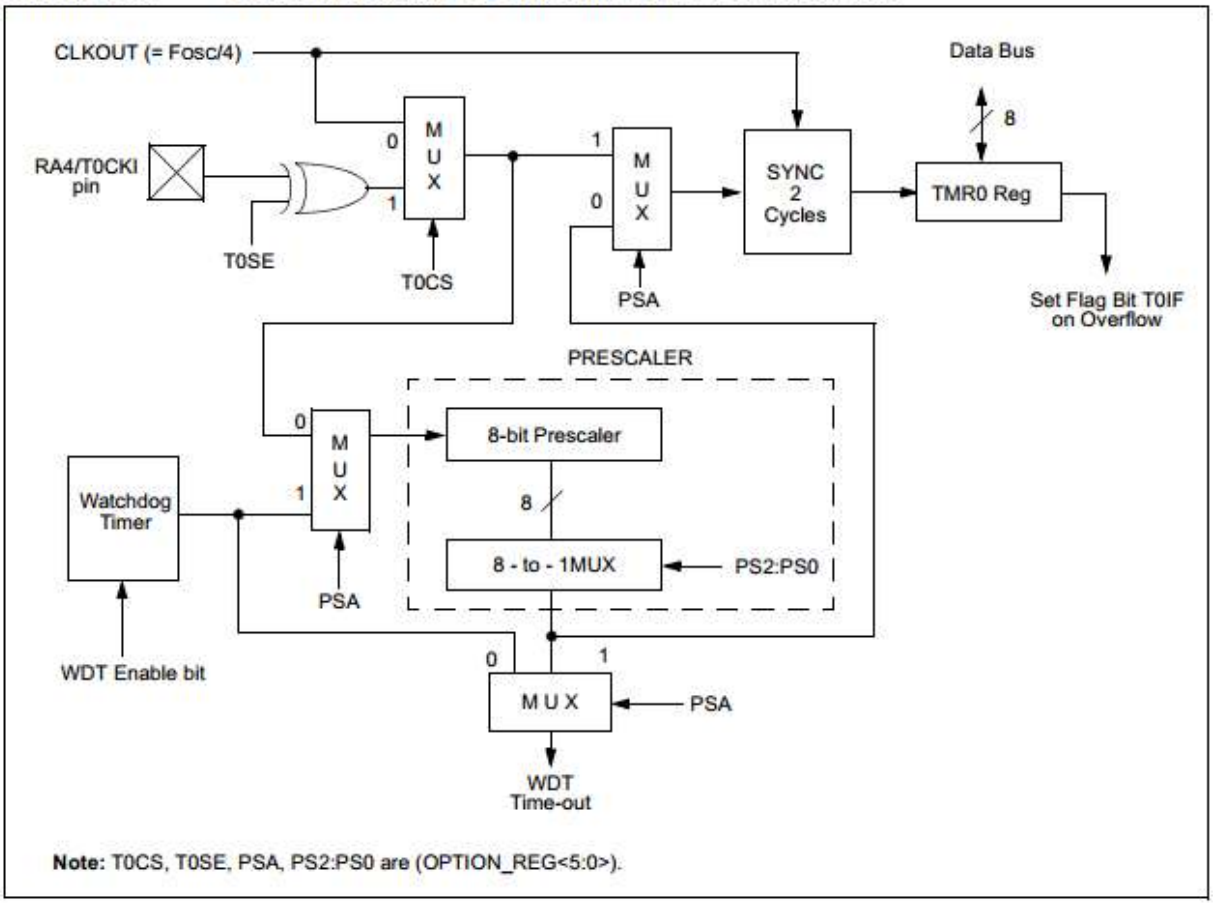
1. Đặc điểm

Bộ timer0/counter0 có những đặc điểm sau:

- Là timer/counter 8 bit.
- Có thể đọc và ghi giá trị đếm của timer/counter
- Có bộ chia trước 8 bit cho phép lập trình lựa chọn hệ số chia bằng phần mềm.
- Cho phép lựa chọn nguồn xung clock bên trong hoặc bên ngoài.
- Phát sinh ngắt khi bị tràn từ FFH về 00H.
- Cho phép lựa chọn tác động xung CK cạnh lên hoặc cạnh xuống.

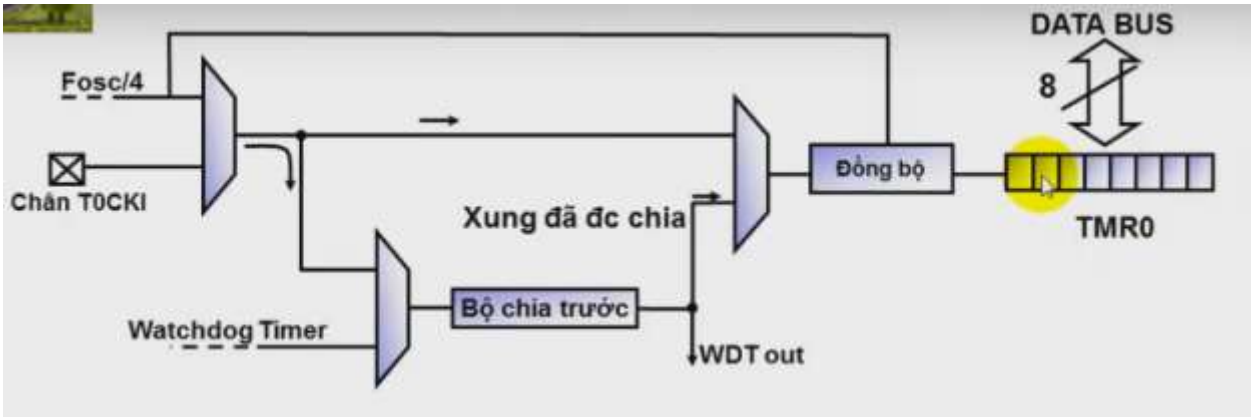
2. Sơ đồ khối của timer0 và bộ chia trước với WDT:

FIGURE 5-1: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER



Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

- **Viết gọn lại:**



- **Giải thích**:
  Nguồn xung vào của Timer0 có thể lấy từ nguồn dao động nội(Fosc/4) hoặc nguồn xung đưa vào chân RA4/T0CKI. Xung này được đưa thẳng đến bộ đếm hoặc thông qua bộ chia trước (bộ chia trước có thể sử dụng cho Timer0 hoặc watchdog timer). Giá trị đếm được lưu vào thanh ghi TMR0(8bit)
  Khi bộ chia trước sử dụng cho watchdog timer thì xung sẽ được đưa thẳng vào bộ đếm luôn(coi như bộ chia 1:1)

Thanh ghi TMR0 cho phép đọc ghi. Khi ghi vào TMR0, giá trị đếm trong bộ chia trước sẽ bị xóa nhưng tỷ lệ chia vẫn giữ nguyên. Nếu timer0 được dùng với chức năng định thời thì phải sau 2Tcy thì timer0 mới bắt đầu đếm.
Khi giá trị trong TMR0 đạt tới giá trị tối đa,thì nó sẽ bị tràn, phải xóa cờ báo tràn TMR0IF trong trình phục vụ ngắt.

3. Thanh ghi

- **Timer0 được cấu hình bởi thanh ghi OPTION:**

REGISTER 5-1: OPTION_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

- **bit 7 RBPU:** PORTB Pull-up Enable bit **// bit điều khiển điện trở treo của portb**.
  1 = PORTB pull-ups are disabled
  0 = PORTB pull-ups are enabled by individual port latch values
- **bit 6 INTEDG**: Interrupt Edge Select bit// bit chọn ngắt cạnh lên hay xuống của ngắt ngoài
  1 = Interrupt on rising edge of RB0/INT pin
  0 = Interrupt on falling edge of RB0/INT pin
- **bit 5 T0CS**: TMR0 Clock Source Select bit: **bit lựa chọn nguồn xung cho TMR0**
  1 = Transition on RA4/T0CKI pin **// đếm xung ngoại đưa đến chân T0CKI.**
  0 = Internal instruction cycle clock (CLKOUT) **// đếm xung clock nội bên trong.**
- **bit 4 T0SE**: TMR0 Source Edge Select bit // **bit lựa chọn cạnh tích cực**
  1 = Increment on high-to-low transition on RA4/T0CKI pin **// tích cực cạnh xuống ở chân T0CKI.**
  0 = Increment on low-to-high transition on RA4/T0CKI pin**// tích cực cạnh lên ở chân T0CKI.**
- **bit 3 PSA**: Prescaler Assignment bit **//bit gán bộ chia**
  1 = Prescaler is assigned to the WDT // **gán bộ chia cho WDT**
  0 = Prescaler is assigned to the Timer0 module// **Gán bộ chia cho timer0**
- **bit 2-0 PS2:PS0**: Prescaler Rate Select bits // **bit chọn tỉ lệ bộ chia trước**

PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

- Nếu bit T0CS bằng 1 thì chọn chế độ đếm xung ngoài Counter. Trong chế độ đếm xung
  ngoài thì xung đếm đưa đến chân RA4/T0CKI. Bit T0SE = 0 thì chọn cạnh lên, ngược lại thì chọn cạnh xuống.
- Bộ chia trước không thể đọc/ghi có mối quan hệ với Timer0 và Watchdog Timer.

4. Ngắt của Timer0

- Khi giá trị đếm trong thanh ghi TMR0 tràn từ FFh về 00h thì phát sinh ngắt, cờ báo ngắt TMR0IF lên 1. Ngắt có thể ngăn bằng bit cho phép ngắt TMR0IE.
- Trong chương trình con phục vụ ngắt Timer0 phải xóa cờ báo ngắt TMR0IF. Ngắt của TMR0 không thể kích CPU thoát khỏi chế độ ngủ vì bộ định thời sẽ ngừng khi CPU ở chế độ ngủ.

5. Timer0 đếm xung ngoại

- Muốn đếm xung ngoại thì xung được đưa đến ngõ vào T0CKI, việc đồng bộ tín hiệu xung ngõ vào T0CKI với xung clock bên trong được thực hiện bằng cách lấy mẫu ngõ ra bộ chia ở những chu kì Q2 và Q4 của xung clock bên trong. Điều này rất cần thiết cho T0CKI ở trạng thái mức cao ít nhất 2 TOSC và ở trạng thái mức thấp ít nhất 2 TOSC

6. Bộ chia trước

- Bộ chia trước có thể gán cho Timer0 hoặc gán cho Watchdog Timer. Các bit PSA và PS2:PS0 chọn đối tượng gán và tỉ lệ chia.
- Khi được gán cho Timer0 thì tất cả các lệnh ghi cho thanh ghi TMR0 sẽ xoá bộ chia trước.
- Khi được gán cho WDT thì lệnh CLRWDT sẽ xoá bộ chia trước cùng với Watchdog Timer.

7. Cách tính giá trị đưa vào TMR0

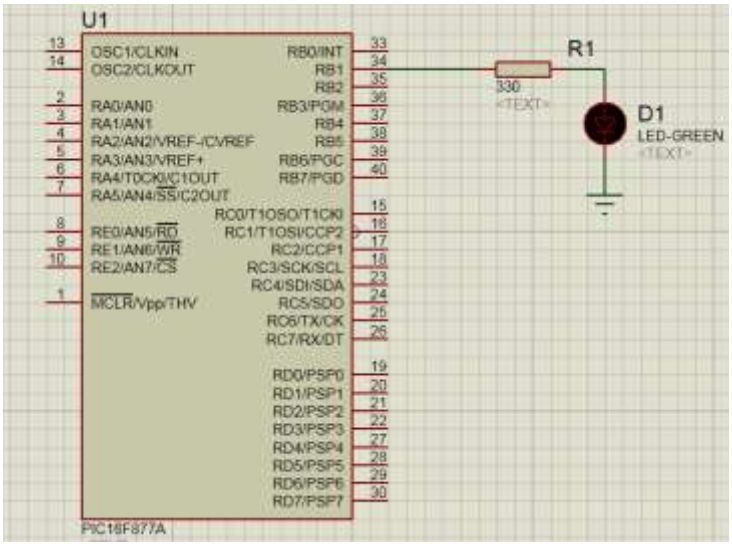
- Tần số thạch anh của pic=20M
- Pic có bộ chia 4 nên tần số lệnh =20M/4=5M.
  - Trong 1s nó sẽ thực hiện được 5 triệu lệnh.
  - Trong 1ms nó sẽ thực hiện được 5000 lệnh

  Muốn dùng timer0 delay 1ms thì bộ timer phải đếm được từ 0-5000
  Do timer0 8bit nên giá trị đếm cao nhất là 255

Do vậy nên ta phải dùng bộ chia trước để giảm giá trị đếm của timer xuống(có 8 bộ chia)
Vậy để timer delay được 1ms thì số lần đếm của timer0=5000/ tỉ lệ bộ chia

Ví dụ :dùng bộ chia 32 thì số lần đếm = 5000/32=156 lần đếm
ở đây ta không thể dùng bộ chia 2,4,8,16 vì số lần đếm vượt quá 255. Vậy nên dùng từ bộ chia 32 trở đi, vậy ta cần nạp 1 giá trị trước cho timer cho tới khi nó tràn(đếm tới 255) thì đủ 156 lần đếm với bộ chia 32 ta cần nạp giá trị ban đầu cho timer0 là 255-156=100 vào TMR0

8. VÍ DỤ: DELAY BẰNG TIMER0, BẬT TẮT LED



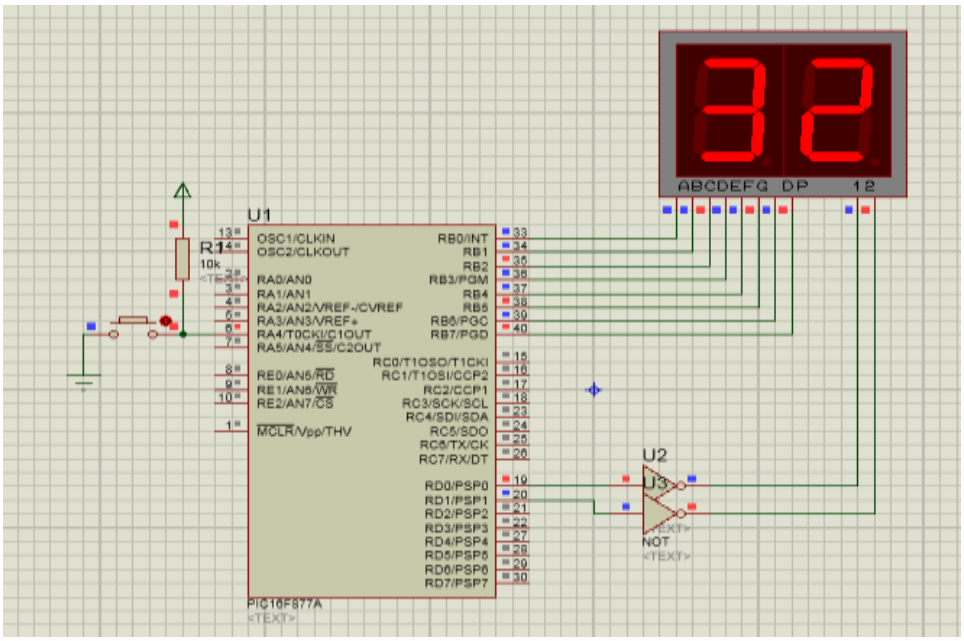

```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
#include "lcd.h"

// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = ON
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000
void delay_timer0_ms(unsigned int t)
{
```

```
   while(t--) // lặp lại t lần cho tới khi nào nó =0 thì thoát khỏi vòng lặp while
   {
   // delay 1ms dùng timer0
   TMR0=100;// nạp giá trị ban đầu cho timer0
   TMR0IF=0;//xóa cờ tràn
   while(!TMR0IF);
   }
}
void timer0_init()
{
   //khoi tao timer
   OPTION_REGbits.T0CS=0;//dem xung noi
   OPTION_REGbits.PSA=0;// bo chia duoc su dung cho timer
   //chọn bộ chia 1:32
   OPTION_REGbits.PS2=1;
   OPTION_REGbits.PS1=0;
   OPTION_REGbits.PS0=0;
}
void main()
{
   TRISB=0;     timer0_init();
   while(1)
   {  RB1=~RB1;// Đảo trạng thái chân RB1
      delay_timer0_ms(500); //Dùng hàm delay timer tạo trễ 500ms
   }
}
```

II. Counter0:

- Muốn timer0 hoạt động ở chế độ counter, ta chỉ cần set bit T0SC trong thanh ghi OPTION. Khi đó, xung tác động lên bộ đếm được lấy từ chân RA4/T0CK1. Bit T0SE cho phép lựa chọn cạnh tác động vào bộ đếm(cạnh tác động là cạnh lên nếu T0SE=0, là cạnh xuống nếu T0SE=1)
- OPTION_REGbits.T0CS = 1; // Đếm xung ngoại
- OPTION_REGbits.PSA = 1; // sử dụng bộ chia trước WDT
- Chỉ cần 2 bit này chúng ta có thể sử dụng chức năng Counter của Timer 0 trong nhiều ứng dụng ví dụ như là : đếm sản phẩm, đếm xung từ encoder….
- Timer0 là thanh ghi 8bit nên giá trị đếm được tối đa là 256
- Muốn chương trình chạy được thì phải cấp xung vào chân RA4/T0CKI là chân chân xung của Timer 0 như vậy thì khi có cạnh lên hoặc cạnh xuống thì chương trình của chúng ta sẽ bắt đầu đếm.
- **Ví dụ** về bài toán đếm sản phẩm hiển thị lên led 7 thanh
  Ta có thể dùng một con led phát và 1 led thu hồng ngoại, khi sản phẩm đi qua thì mạch gắn với led thu sẽ có nhiệm vụ quét ra một xung để biết có sản phẩm đi

qua và xung đó được đưa vào chân RA4. ở bài này mô phỏng bằng phím nhấn. mỗi lần nhấn tương ứng với một xung được tạo




```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = ON
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000

unsigned int dem = 0,nghin,tram,chuc,donvi;
const unsigned char maled[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,
0x90};
void hien_thi()
{
   chuc = dem/10;
   donvi = dem%10;
   PORTB = maled[chuc];
   RD0=0;
   __delay_ms(10);
   RD0=1;
   PORTB = maled[donvi];
   RD1=0;
```

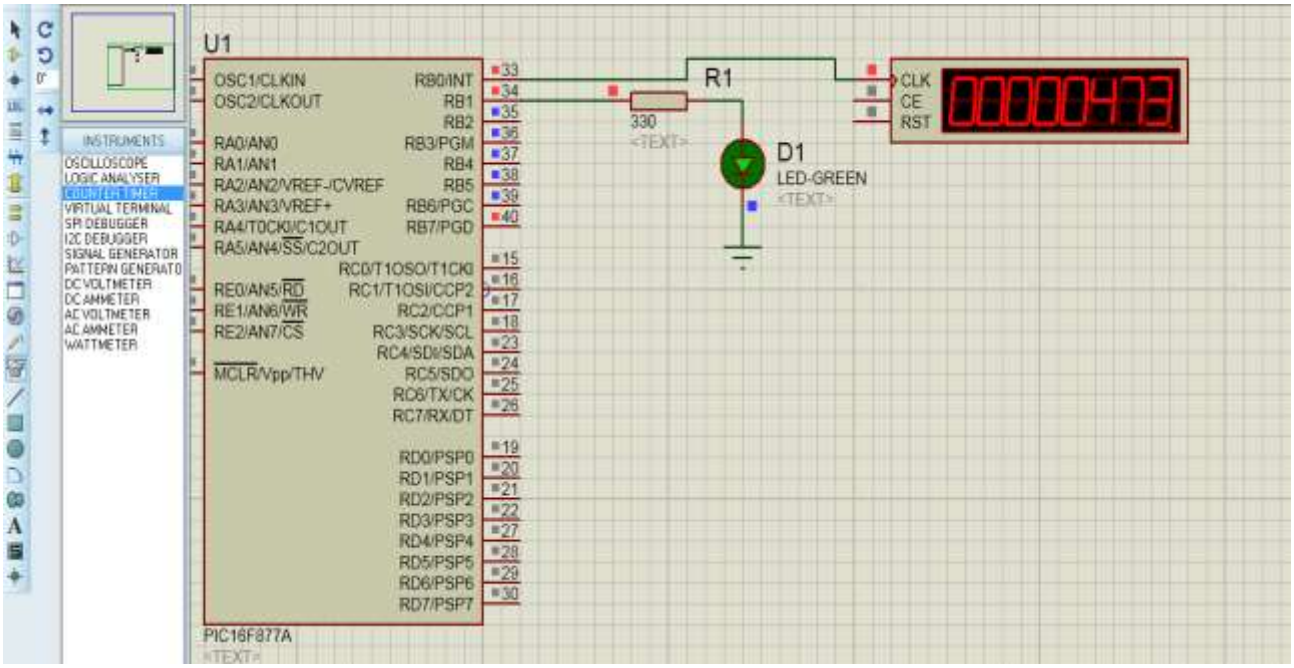
```
   __delay_ms(10);
   RD1=1;
}
void main()
{
TRISA4=1;//Tao xung dau vao
   TRISB = 0;
   PORTB = 0;
   TRISD0 = 0;
   TRISD1 = 0;
   OPTION_REGbits.T0CS = 1;          // đếm xung ngoại
   OPTION_REGbits.PSA = 1;           // sử dụng bộ chia trước WDT
   while (1)
   {
      hien_thi();
      dem=TMR0; // Gán giá trị đếm cho TMR0
      if(dem>=100)
      {
         TMR0=0;
         dem=0;
      }
   }
}
```

III. NGẮT TIMER 0

- Nguồn ngắt: là trạng thái tràn của thanh ghi bộ đếm timer 0, TMR0 vi điều khiển PIC
  - Sự kiện ngắt: sự kiện ngắt xảy ra khi có sự tràn của TMR0, tức là khi TMR0=255 rồi bị xóa
  - Bit cho phép ngắt: Để cho phép ngắt này, bit cho phép ngắt TMR0IE (bit 5 của thanh ghi INTCON) phải được set lên 1. Ngoài ra, bit cho phép ngắt toàn cục GIE (bit 7 của thanh ghi INTCON) cũng phải được set lên 1.
  - Cờ ngắt: bit cờ ngắt ngoài là bit TMR0IF (bit 2 của thanh ghi INTCON) được tự động set lên 1 khi có sự kiện ngắt ngoài xảy ra. Cờ này phải được xóa bằng chương trình (cụ thể là trong chương trình con phục vụ ngắt) để vi điều khiển quản lý chính xác các lần ngắt kế tiếp.

  Ví dụ:

  Dùng ngắt timer0 đảo trạng thái PORTB



Code

```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = ON
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000

void timer0_init()
    {
       //khoi tao timer
       OPTION_REGbits.T0CS=0;//dem xung noi
       OPTION_REGbits.PSA=0;// bo chia duoc su dung cho timer
       // chon ty le bo chia truoc là 1:32
       OPTION_REGbits.PS2=1;
       OPTION_REGbits.PS1=0;
       OPTION_REGbits.PS0=0;

       TMR0=100;// nap truoc gia tri 100 cho timer0
       TMR0IF=0;// xoa co tran timer0
```

```
   TMR0IE=1;// cho phep ngat timer0
   GIE=1;// ngat toan cuc
}
void interrupt timer0()
{
   if(TMR0IF)// khi cờ báo tràn =1
   {
      TMR0=100;// Nạp lại giá trị cho timer0
      PORTB=~PORTB;// viết code bất kì bạn muốn thực hiện khi có ngắt, ở đây
tôi đổi trạng thái PORTB
      TMR0IF=0;// xóa cờ báo ngắt
   }
}

void main()
{
   TRISB=0;
   timer0_init();
   while(1)
   {
   }
}
```

Để mô phỏng, ta sử dụng công cụ couter-timer, kích chuột vào chọn kiểu đo tần số(frequency)

Trong ngắt trên, cứ 1ms nó xảy ra một lần, nên 1ms, PORTB sẽ đảo giá trị 1 lần. nó bằng 0 trong 1ms đầu, sau đó =1 trong 1ms sau và cứ như vậy lặp đi lặp lại. vậy nên chu kì =2ms. Tần số=1/2ms ra khoảng 500hz

Thì từ đó dùng bộ couter timer ta sẽ thấy giá trị trong mô phỏng gần đúng với giá trị tính toán

BÀI 5: TIMER 1/COUNTER 1/NGẮT TIMER 1

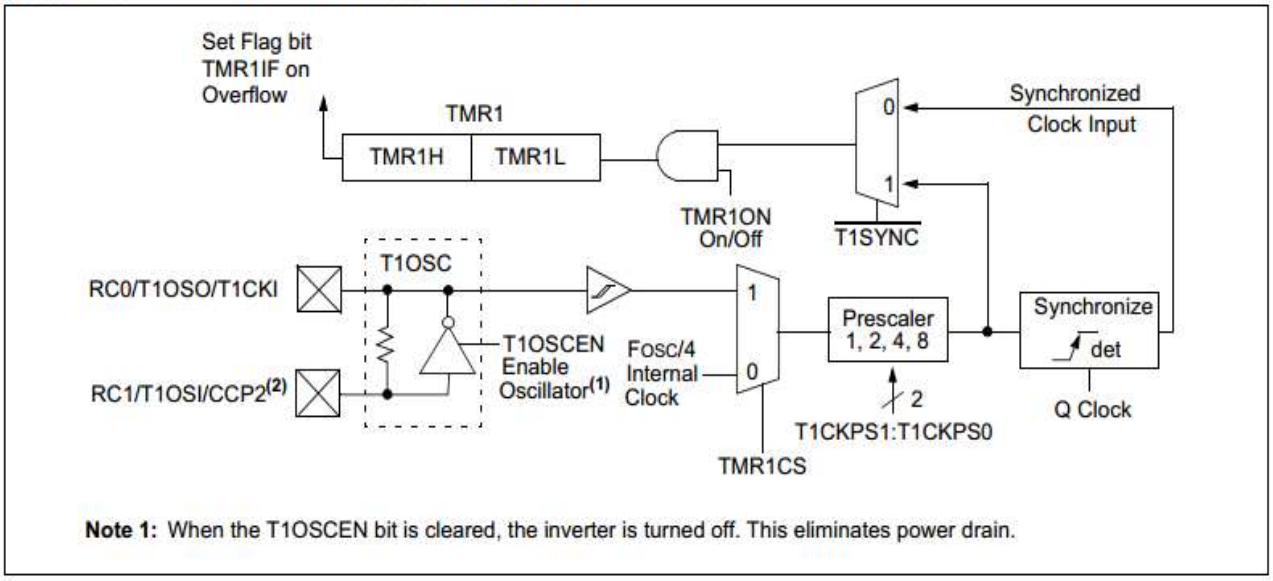
I. TIMER 1

1. Đặc điểm của timer 1

- Là timer/counter 16bit gồm 2 thanh ghi TMR1H và TMR1L có thể đọc và ghi.
- Timer 1 có thể hoạt động ở chế độ định thời hay đếm được lựa chọn bởi bit TMR1CS.
- Trong chế độ định thời T1 tăng giá trị ở mỗi chu kỳ lệnh, chế độ đếm bộ đếm tăng mỗi khi có cạch clock ngõ vào prescale bên ngoài.
- Có bộ prescale chia tần.
- Xảy ra hiện tượng ngắt khi tràn từ FFFFh -> 0000h.

2. Sơ đồ khối

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

- Khi bit T1OSCEN (T1CON<3>) được set, Timer1 sẽ lấy xung clock từ hai chânRC1/T1OSI/CCP2 và RC0/T1OSO/T1CKI làm xung đếm. Timer1 sẽ bắt đầu đếm sau cạnh xuống đầu tiên của xung ngõ vào. Khi đó PORTC sẽ bỏ qua sự tác động của hai bitTRISC<1:0> và PORTC<2:1> được gán giá trị 0. Khi clear bit T1OSCEN Timer1 sẽ lấy xung đếm từ oscillator hoặc từ chân RC0/T1OSO/T1CKI.
- Timer1 có hai chế độ đếm là đồng bộ (Synchronous) và bất đồng bộ (Asynchronous).Chế độ đếm được quyết định bởi bit điều khiển (T1CON<2>).
- Khi T1SYNC=1 xung đếm lấy từ bên ngoài sẽ không được đồng bộ hóa với xung clock bên trong, Timer1 sẽ tiếp tục quá trình đếm khi vi điều khiển đang ở chế độ sleep và ngắt doTimer1 tạo ra khi bị tràn có khả năng "đánh thức" vi điều khiển. Ở chế độ đếm bất đồng bộ,Timer1 không thể được sử dụng để làm nguồn xung clock cho khối CCP(Capture/Compare/Pulse width modulation).
- Khi T1SYNC =0 xung đếm vào Timer1 sẽ được đồng bộ hóa với xung clock bên trong. Ởchế độ này Timer1 sẽ không hoạt động khi vi điều khiển đang ở chế độ sleep.

3. Thanh ghi

Các thanh ghi liên quan đến timer1 bao gồm:

- INTCON (địa chỉ 0bh, 8bh, 10bh, 18bh): cho phép ngắt hoạt động (GIE và PEIE).
- PIR1 (địa chỉ 0ch): chứa cờ ngắt timer1 (TMR1IF).
- PIE1(địa chỉ 8ch): cho phép ngắt timer1 (TMR1IE).
- TMR1L (địa chỉ 0eh): chứa giá trị 8 bit thấp của bộ đếm timer1.
- TMR1H (địa chỉ 0eh): chứa giá trị 8 bit cao của bộ đếm timer1.
- TLCON: (địa chỉ 10h): xác lập các thông số cho timer1.

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

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

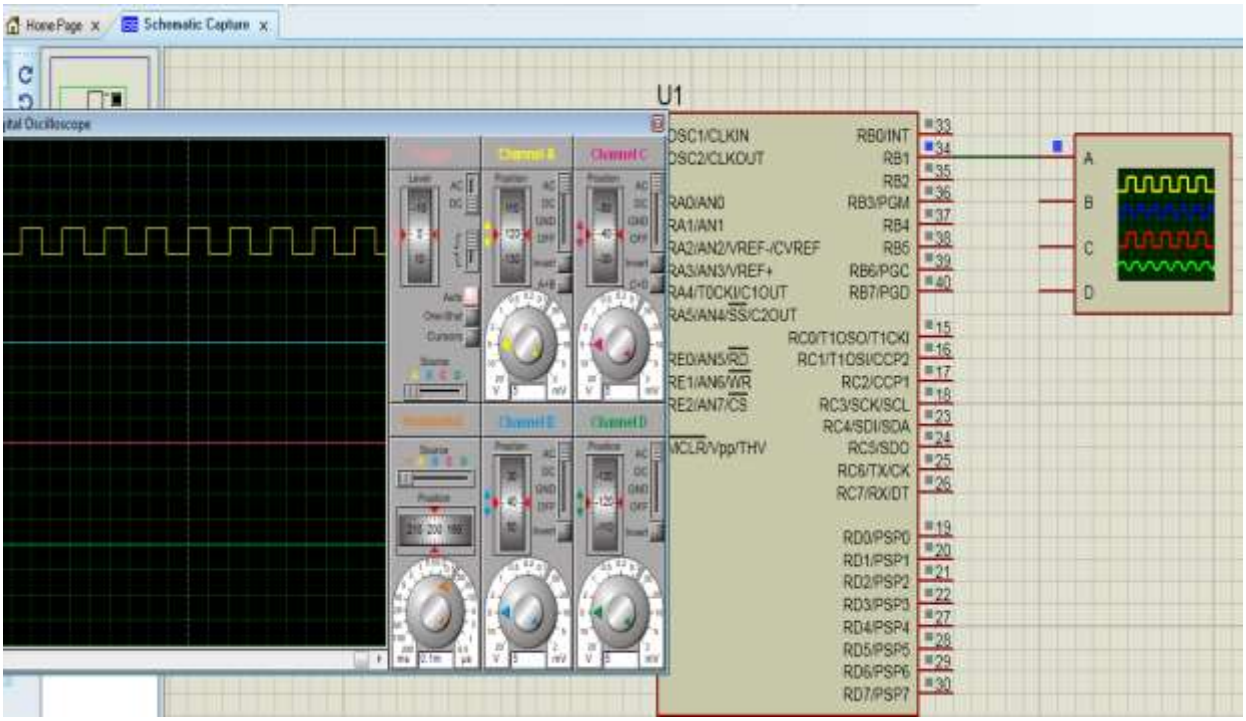
- **Bit7-6**: không dùng, cho =0
- **Bit 5-4 T1CKPS1:T1CKPS0**: chọn tỉ lệ bộ chia trước timer1
  11 = 1:8 Prescale value
  10 = 1:4 Prescale value
  01 = 1:2 Prescale value
  00 = 1:1 Prescale value
- **bit 3** T1OSCEN: Timer1 Oscillator Enable Control bit
  1 = Oscillator is enabled
  0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)
- **bit 2 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:Timer1 uses the internal clock when TMR1CS = 0.
- **bit 1 TMR1CS**: chọn xung vào timer1 là xung nội hay ngoại
  1 = xung ngoại,dùng chân RC0/T1OSO/T1CKI
  0 = xung nội(FOSC/4)
- **bit 0 TMR1ON**: bật timer1
  1 = Enables Timer1
  0 = Stops Timer1

4. ví dụ: tạo hàm delay bằng timer1

```
#include <xc.h>

#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = ON
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000

void delay_timer1_ms(unsigned int t)
```

```
{
   while(t--)
   {
      // Delay 1ms bang cach kiem tra co tran
      // nap gia tri TMR1H:TMR1L=65536-5000=60536
      // Kiem tra co tran. khi co tran thi dem du 5000 xung, sau do ta chi can dong co tran
lai
      TMR1H=60536>>8;// dua gia tri 60536 vao 8 bit cao cua thanh ghi TMR1H sau
dich ve 8 bit thap
      TMR1L=60536&0X00FF;// xoa di 8 bit cao va giu lại 8 bit thap nap vao thanh ghi
TMR1L cho timer hoat dong
      TMR1ON=1;// bat timer0
      while(TMR1IF);// doi co tran =1
      TMR1IF=0;// xoa co tran
      TMR1ON=0;// timer1 ngung dem
   }
}
void timer1_init()
{
   TMR1CS=0;// dung xung noi
   // dung bo chia ty le 1:1
   T1CKPS1=0;
   T1CKPS0=0;
}
void main()
{
   TRISB=0;
   timer1_init();
   while(1)
   {
      RB1=~RB1;
      delay_timer1_ms(500);
   }
}
```

- **ví dụ 2: tạo một xung vuông với tần số 1k Hz dùng timer1**



Code:

```
#include <xc.h>

#include <pic.h>
#include <pic16f877a.h>
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = ON
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000
unsigned int Count = 0;
unsigned char f;

void pwm_timer_init(unsigned char f) {
   TRISBbits.TRISB1 = 0;
   TMR1 = (65536-5000*1000/f);
   T1CONbits.TMR1CS = 0;
   T1CONbits.T1CKPS1 = 0;
   T1CONbits.T1CKPS0 = 0;
   T1CONbits.T1SYNC = 1;
   T1CONbits.TMR1ON = 1;
```

```
    while(1)
    {
    if (PIR1bits.TMR1IF == 1) {
       PIR1bits.TMR1IF = 0;
       TMR1 = (65536-5000*1000/f);
       Count++;
       if (Count == 1) {
          Count = 0;
          RB1 = ~RB1;
       }
    }
    }
}
void main(void) {
    pwm_timer_init(1000);
    while (1) {
    }
}
```

chú ý: ví dụ này làm tổng quát cho trường hợp bộ chia 1:1 của timer 16bit. Bạn có thể chọn bộ chia khác để phù hợp với mỗi bài toán

II. COUNTER 1

1. Hoạt động của Timer1 ở chế độ Counter đồng bộ

Khi bit TMR1CS bằng 1 thì T1 hoạt động ở chế độ Counter:

- Nếu bit T1OSCEN bằng 1 thì đếm xung từ mạch dao động của T1.
- Nếu bit T1OSCEN bằng 1 thì đếm xung cạnh lên đưa đến ngõ vào RC0/T1OSO/T1CKI.
- Nếu bit T1SYNC bằng 0 thì ngõ vào xung ngoại được đồng bộ với xung bên trong. Ở chế độ đồng bộ, nếu CPU ở chế độ ngủ thì Timer1 sẽ không đếm vì mạch đồng bộ ngừng hoạt động.

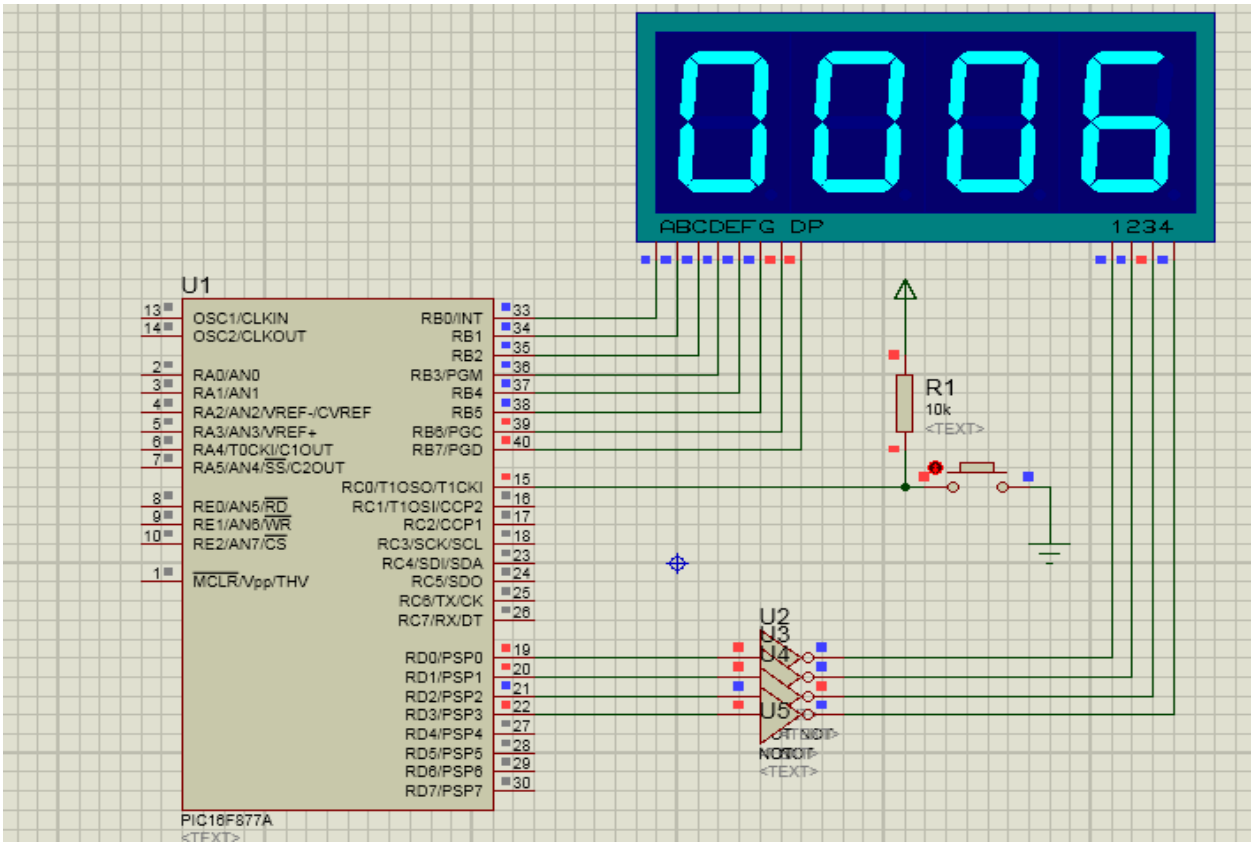
2. Hoạt động của Timer1 ở chế độ Counter bất đồng bộ

- Nếu bit T1SYNC bằng 1 thì xung ngõ vào từ bên ngoài không được đồng bộ. Bộ đếm tiếp tục tăng bất đồng bộ với xung bên trong. Bộ đếm vẫn đếm khi CPU ở trong chế độ ngủ và khi tràn sẽ phát sinh ngắt và đánh thức CPU. Ngắt T1 có thể ngăn được.
- Để sử dụng chế độ Counter của Timer 1 chúng ta chỉ cần cấu hình các thanh ghi và bit dưới đây là chúng ta có thể sử dụng chế độ Counter của Timer 1 và các thanh ghi và bit cần cấu hình như sau :

- T1CONbits.TMR1CS = 1;
  T1CONbits.T1SYNC = 0;
  T1CONbits.TMR1ON = 1;
  - Giải thích về 3 thanh ghi và bit ở trên như sau :
    T1CONbits.TMR1CS = 1 : Chọn nguồn xung clock từ bên ngoài ở chân RC0/T1OSO/T1CKI (cạnh lên).
    T1CONbits.T1SYNC = 0 : Đồng bộ ngõ vào clock ở từ bên ngoài.
    T1CONbits.TMR1ON = 1: Cho phép Timer1 đếm.
  - Ngoài việc cấu hình các thanh ghi và bit ở trên chúng ta cần quan tâm tới 1 việc quan trọng nữa là cấp nguồn Clock và chân RC0 thì khi có cạnh lên hoặc cạnh xuống thì chế độ Counter sẽ bắt đầu đếm.

3. Ví dụ

- Một dây chuyền sản xuất cần đếm 5000 sản phẩm. hiển thị số lượng đếm được lên lcd
  Ta dùng counter1 để đếm(vì timer1 là thanh ghi 16 bit nên counter1 đếm tối đa được 2^16




```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
```

```
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = ON
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000

unsigned int dem = 0,nghin,tram,chuc,donvi;
const unsigned char maled[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
0x80, 0x90};
void hien_thi ()
{
   nghin = dem/1000;
   tram = (dem - nghin*1000)/100;
   chuc = (dem - nghin*1000 - tram*100)/10;
   donvi = dem%10;
   PORTB = maled[nghin];
   RD0=0;
   __delay_ms(5);
   RD0=1;
   PORTB = maled[tram];
   RD1=0;
   __delay_ms(5);
   RD1=1;
   PORTB = maled[chuc];
   RD2=0;
   __delay_ms(5);
   RD2=1;
   PORTB = maled[donvi];
   RD3=0;
   __delay_ms(5);
   RD3=1;
}
void main(void)
{
   T1CONbits.TMR1CS = 1;
   T1CONbits.T1SYNC = 0;
   T1CONbits.TMR1ON = 1;
   TRISB = 0;
```

```
   PORTB = 0;
   TRISDbits.TRISD0 = 0;
   TRISDbits.TRISD1 = 0;
   TRISDbits.TRISD2 = 0;
   TRISDbits.TRISD3 = 0;
   PORTD = 0;
   while (1)
   {
      hien_thi();
      dem=TMR1;
      if(dem>=1000)
      {
         TMR1=0;
         dem=0;
      }
   }
}
```

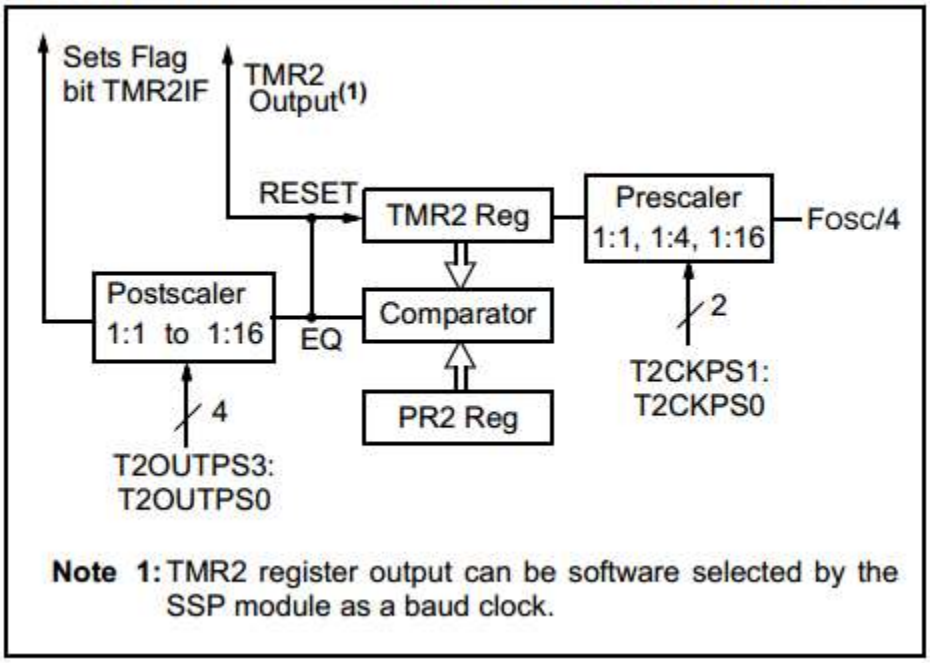
TỔNG QUAN VỀ TIMER2

1. Đặc điểm

Timer 2 là bộ định thời 8 bit được hỗ trợ bởi 2 bộ chia tần số press scaler và postcaler.

2. Sơ đồ khối

FIGURE 7-1: TIMER2 BLOCK DIAGRAM



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

Thanh ghi chứa giá trị đếm của timer2 là TMR2. Bit cho phép ngắt timer2 tác động là TMR2ON(T2CON<2>). Cờ ngắt của timer 2 là bit TMR2IF. Xung ngõ vào (1/4 Fosc) được đưa qua bộ chia tần số prescaler 4bit (tỉ số chia tần là 1:1, 1:4, 1:16) và được điều khiển bởi các bit T2CKPS1:T2CKPS0 của thanh ghi T2CON

Timer2 còn được hỗ trợ bởi thanh ghi PR2. Giá trị đếm trong thanh ghi TMR2 sẽ tăng từ 00h đến giá trị chứa trong thanh ghi PR2, sau đó được reset về 00h. khi reset, thanh ghi PR2 nhận giá trị mặc định là FFh

Ngõ ra của timer 2 được đưa qua bộ chia tần số postcaler với các mức chia từ 1:1 đến 1:16 được điều khiển bởi 4 bit : T2OUTPS3:T2OUTPS0. Ngõ ra của postcaler đóng vai trò quyết định cho việc điều khiển cờ ngắt.

3. Thanh ghi

Các thanh ghi liên quan tới timer 2 bao gồm:

- **INTCON**: cho phép các ngắt(GIE,PEIE)

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

- **PIR1**: chứa cờ ngắt timer2 (TMR2IF).

PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Bit 1: TMR2IF: cờ báo ngắt Timer2

Các bit còn lại liên quan tới cờ ngắt của các khối chức năng khác

- **PIE1**: chứa bit điều khiển timer2 (TMR2IE)

PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

- **TMR2:** chứa giá trị đếm của timer2
- **T2CON**: xác lập các thông số cho timer2

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Bit 7: không dùng,đặt =0

Bit 6:3 (TOUTPS3:TOUTPS0): Chọn tỷ số bộ chia tần postcaler

0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•
•
•
1111 = 1:16 Postscale

Bit 2 (TMR2ON): Cho phép timer2 hoạt động
=1: cho phép
=0: không cho phép
Bit 1:0 (T2CKPS1:T2CKPS0): Bit chọn tỷ lệ bộ chia tần số prescaler
00: prescaler is 1
01: prescaler is 4
1x: prescaler is 16

- **PR2**: thanh ghi hỗ trợ cho timer2

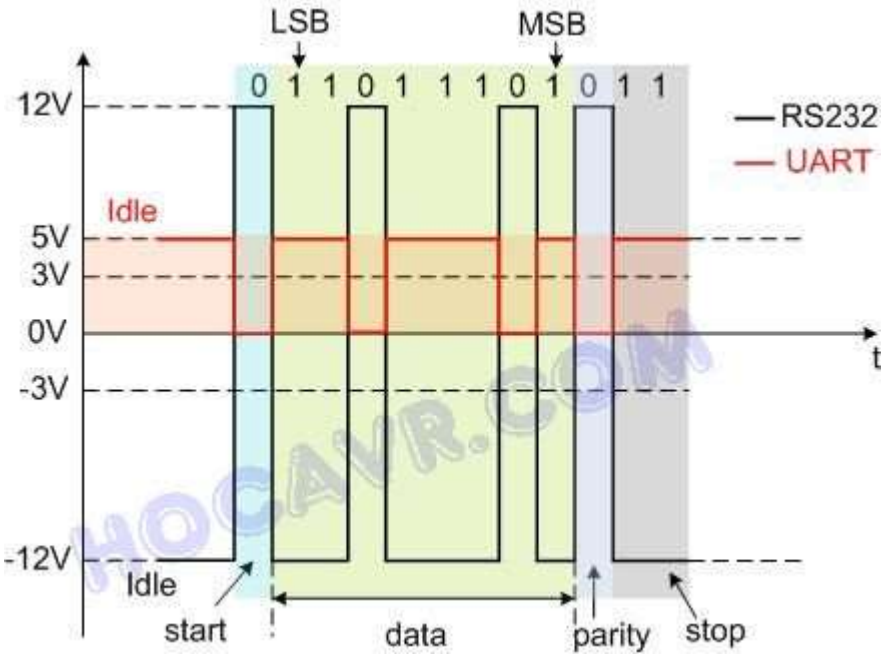
4. Ví dụ: cách làm tương tự với timer0 và timer1

BÀI 6:CHUẨN GIAO TIẾP UART

I. Tổng quan về uart

1. Khái niệm

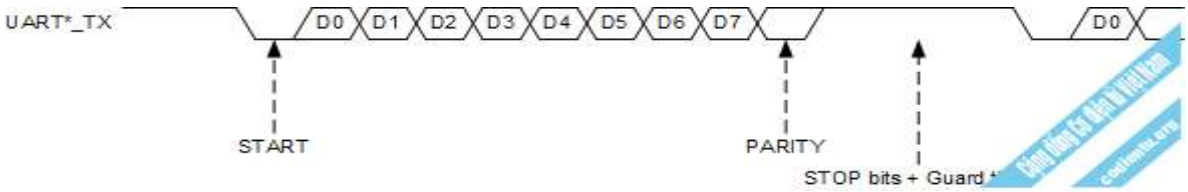
- Thuật ngữ USART trong tiếng anh là viết tắt của cụm từ: Universal Synchronous & Asynchronous serial Reveiver and Transmitter, nghĩa là bộ truyền nhận nối tiếp đồng bộ và không đồng bộ. USART hay UART cần phải kết hợp với một thiết bị chuyển đổi mức điện áp để tạo ra một chuẩn giao tiếp nào đó. Ví dụ, chuẩn RS232 (hay COM) trên các máy tính cá nhân là sự kết hợp của chip UART và chip chuyển đổi mức điện áp.



Hình 2.27: Tín hiệu tương đương của UART và RS232.

- Việc truyền UART được dùng giữa 2 vi xử lý với nhau, đồng nghĩa với việc mỗi vi xử lý có thể tự tạo ra xung clock cho chính nó xử dụng.

- Để bắt đầu cho việc truyền dữ liệu bằng UART, một START bit được gửi đi, sau đó là các bit dữ liệu và kết thúc quá trình truyền là STOP bit.



Hình 2.28: Quá trình bắt đầu gửi tín hiệu.

- Như hình ta có thể thấy, khi ở trạng thái chờ mức điện thế ở mức 1 (high). Khi bắt đầu truyền START bit sẻ chuyển từ 1 xuống 0 để báo hiệu cho bộ nhận là quá trình truyền dữ liệu sắp xảy ra. Sau START bit là đến các bit dữ liệu D0-D7. Sau khi truyền hết dữ liệu thì đến Bit Parity để bộ nhận kiểm tra tính đúng đắn của dữ liệu truyền Cuối cùng là STOP bit là 1 báo cho thiết bị rằng các bit đã được gửi xong. Thiết bị nhận sẽ tiến hành kiểm tra khung truyền nhằm đảm báo tính đúng đắn của dữ liệu.

2. Các thông số cơ bản trong truyền nhận UART:

- **Baund rate (tốc độ baund):** Khoảng thời gian dành cho 1 bit được truyền. Phải được cài đặt giống nhau ở gửi và nhận. Để việc truyền và nhận không đồng bộ xảy ra thành công thì các thiết bị tham gia phải "thống nhất" nhau về khoảng thời dành cho 1 bit truyền, hay nói cách khác tốc độ truyền phải được cài đặt như nhau trước, tốc độ này gọi là tốc độ Baud.

Tốc độ baud là số bit truyền trong 1 giây. Ví dụ nếu tốc độ baud được đặt là 19200 thì thời gian dành cho 1 bit truyền là 1/19200 ~ 52.083us.

- **Frame (khung truyền):** Khung truyền quy định về số bit trong mỗi lần truyền.
  Do truyền thông nối tiếp mà nhất là nối tiếp không đồng bộ rất dễ mất hoặc sai lệch dữ liệu, quá trình truyền thông theo kiểu này phải tuân theo một số quy cách nhất định. Bên cạnh tốc độ baud, khung truyền là một yếu tốc quan trọng tạo nên sự thành công khi truyền và nhận. Khung truyền bao gồm các quy định về số bit trong mỗi lần truyền, các bit "báo" như bit Start và bit Stop, các bit kiểm tra như Parity, ngoài ra số lượng các bit trong một data cũng được quy định bởi khung truyền.
- **Start bit:** là bit đầu tiên được truyền trong 1 Frame. Báo hiệu cho thiết bị nhận có một gói dữ liệu sắp đc truyền đến. Bit bắt buộc.
- **Data**: dữ liệu cần truyền. Bit có trọng số nhỏ nhất LSB được truyền trước sau đó đến bit MSB. Data không nhất thiết phải là gói 8 bit. Trong truyền thông nối tiếp UART, bit có ảnh hưởng nhỏ nhất (LSB – Least Significant Bit, bit bên phải) của data sẽ được truyền trước và cuối cùng là bit có ảnh hưởng lớn nhất (MSB – Most Significant Bit, bit bên trái).
- **Parity bit:** kiểm tra dữ liệu truyền có đúng không. Có 2 loại parity là parity chẵn (even parity) và parity lẻ (odd parity). Parity chẵn nghĩa là số lượng số 1 trong dữ liệu bao gồm bit parity luôn là số chẵn. Ngược lại tổng số lượng các số1 trong parity lẻ luôn là số lẻ. Ví dụ, nếu dữ liệu của bạn là 10111011 nhị phân, có tất cả 6 số 1 trong dữ liệu này, nếu parity chẵn được dùng, bit parity sẽ mang giá trị 0 để đảm bảo tổng các số 1 là số chẵn (6 số 1). Nếu parity lẻ được yêu cầu thì giá trị của parity bit là 1
- **Stop bit**: là 1 hoặc các bit báo cho thiết bị rằng các bit đã được gửi xong. Thiết bị nhận sẽ tiến hành kiểm tra khung truyền nhằm đảm bảo tính đúng đắn của dữ liệu. Bit bắt buộc

II. UART VỚI PIC16F877A

1. Truyền dữ liệu

- Việc truyền dữ liệu thông qua thanh ghi TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

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, bit này không có ứng dụng ở chế độ hoạt động không đồng bộ USART module. Nó được sử dụng để chọn master hoặc slave mode ở chế độ hoạt động đồng bộ.
  CSRC=0: Master mode(xung clock được tạo ra trong nội bộ)
  CSRC=1: Slaver mode(xung clock từ bên ngoài)
- **Bit 6 TX9** : Khi bit này được thiết lập nó cho phép truyền 9 bit,nếu không truyền 8 bit được sử dụng.

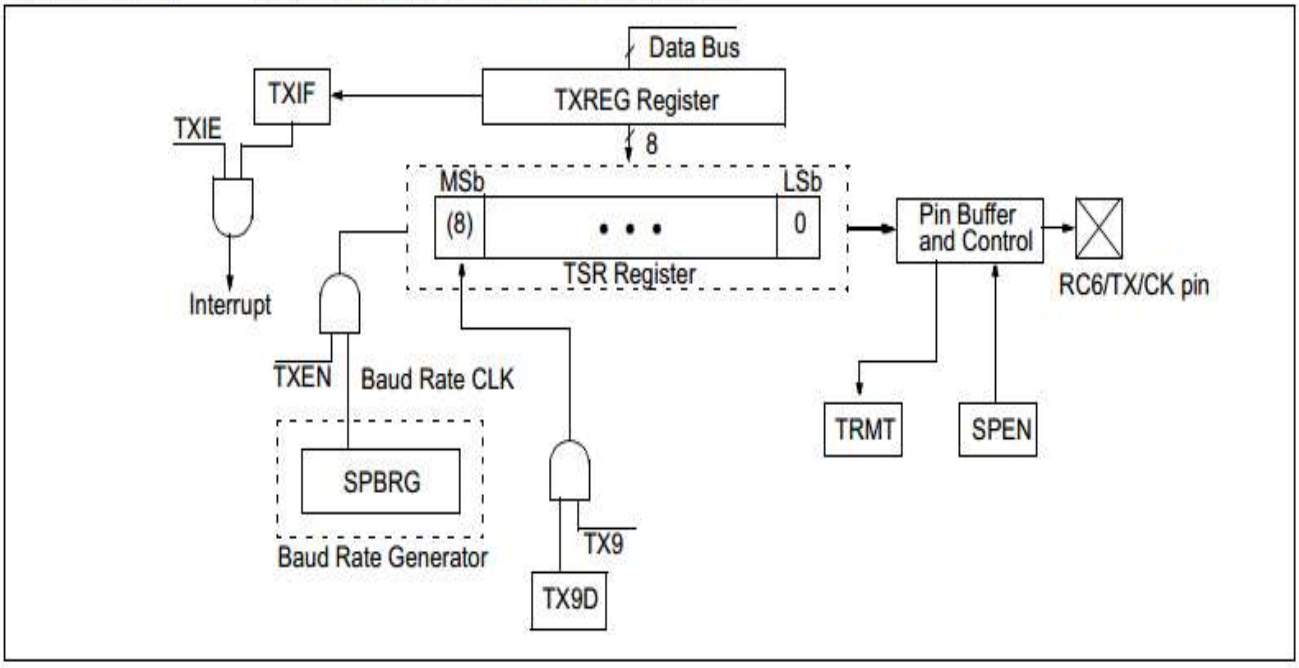
TX9=1: Chọn chế độ 9 bit
TX9=0: chế độ 8 bit

- **Bit 5 TXEN** : bit cho phép truyền.
  TXEN=1: cho phép
  TXEN=0: không cho phép
- **Bit 4 SYNC** : Đây là chế độ USART chọn bit. Thiết lập bit này lựa chọn chế độ đồng bộ hay không đồng bộ
  SYNC=1: đồng bộ
  SYNC=0: không đồng bộ

- **Bit 3 chưa thực hiện** : bit này là chưa thực hiện và sẽ đọc là 0.
- **Bit 2 BRGH** : bit chọn tốc độ baurate. Chỉ dùng cho chế độ truyền thông không đồng bộ
  BRGH=0: Tốc độ thấp
  BRGH=1: Tốc độ cao
- **Bit 1 TRMT** : Đây là Transmit Shift Register (TSR) bit trạng thái. Điều này có thể được sử dụng để kiểm tra xem các dữ liệu được truyền đi hay không. Khi TRS là trống, bit này được thiết lập và khi TSR là đầy thì bit này sẽ là 0.
- **Bit 0 TX9D** : Đây là bit thứ 9 trong chế độ truyền 9 bit. Điều này thường được sử dụng như là bit chẵn lẻ

❖ **Sơ đồ khối của việc truyền dữ liệu:**

FIGURE 10-1: USART TRANSMIT BLOCK DIAGRAM



- Dữ liệu cần truyền được đặt vào thanh ghi TXREG, baund rate được tạo ra, khi TXEN gán bằng 1 dữ liệu từ thanh ghi TXREG đưa vào thanh ghi TSR đồng thời baund rate tác động đến TSR, đẩy dữ liệu cần truyền ra bộ đệm sau đó xuất ra chân TX.
- Bit TXIF dùng để báo trạng thái trong thanh ghi TXREG, nếu có dữ liệu trong TXREG thì TXIF = 1. Nếu dữ liệu được truyền xuống thanh TSR thì TXIF = 0. Tương tự bit TRMT dùng để báo trạng thái thanh ghi TSR.

❖ **Các bước để truyền dữ liệu:**

- Khởi tạo baund rate: ở thanh ghi SPBRG

TABLE 10-1: BAUD RATE FORMULA

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)

- Kích hoạt các cổng nối tiếp không đồng bộ bằng cách xóa bit SYNC và thiết lập bit spen
- Nếu dùng ngắt, thiết lập cho phép TXIE
- Nếu dùng bit thứ 9 thì set TXIE
- Cho phép quá trình truyền thông không đồng bộ bằng cách thiết lập SPEN = 1; SYNC = 0;
- Cho phép truyền dữ liệu bằng cách thiết lập bit TXEN = 1;
- Khi cần truyền dữ liệu thì cần set dữ liệu đó lên TXREG.

2. Nhận dữ liệu

- Được biểu diễn trên thanh ghi RCSTA

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. Thiết lập bit này cho phép truyền thông
  1 = cho phép (cấu hình RC7/RX/DT và RC6/TX/CK)
  0 = không cho phép

- **Bit 6 RX9** : Thiết lập bit này cho phép tiếp nhận 9 bit nếu không nó sẽ ở chế độ tiếp nhận 8 bit.
  RX9=1: Nhận 9 bit
  RX9=0: nhận 8 bit
- **Bit 5 SREN** : cho phép nhận dữ liệu.
  SREN=1: Cho phép nhận
  SREN=0: Không cho phép
- **Bit 4 CREN** : cho phép nhận bit liên tục ở cả chế độ đồng bộ và không đồng bộ
  SREN=1: Cho phép
  SREN=0: không cho phép

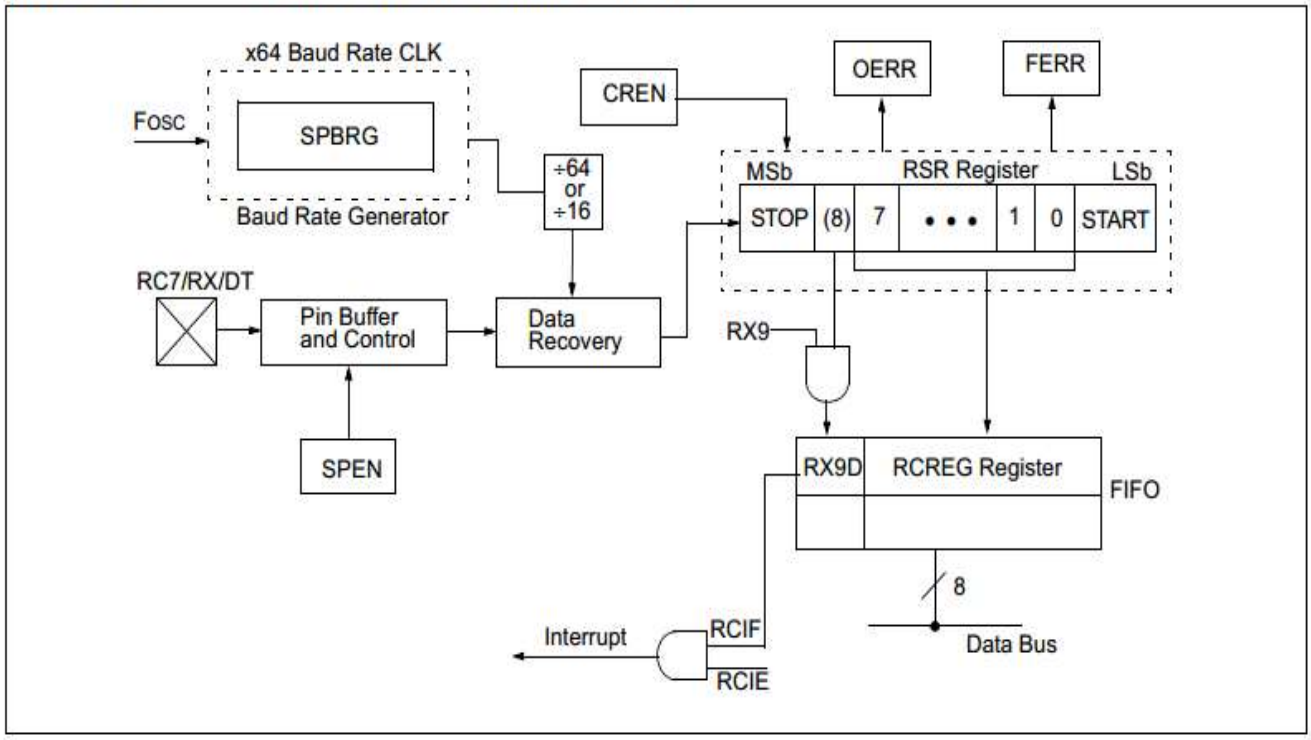
- **Bit 3 ADDEN** : bit cho phép phát hiện địa chỉ. Bit này chỉ áp dụng trong chế độ không đồng bộ 9 bit.
- **Bit 2 FERR** : bit kiểm tra khung truyền.

  FERR =1:lỗi

  FERR =0: không lỗi
- **Bit 1 OERR** : kiểm tra tràn bit.
- **Bit 0 RX9D** : Đây là bit thứ 9 nhận dữ liệu và thường được sử dụng như Parity Bit.

❖ **Sơ đồ khối cho việc nhận dữ liệu:**

FIGURE 10-4: USART RECEIVE BLOCK DIAGRAM



Cơ chế hoạt động tương tự như phần truyền dữ liệu

3. Tốc độ baudrate

TABLE 10-1: BAUD RATE FORMULA

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

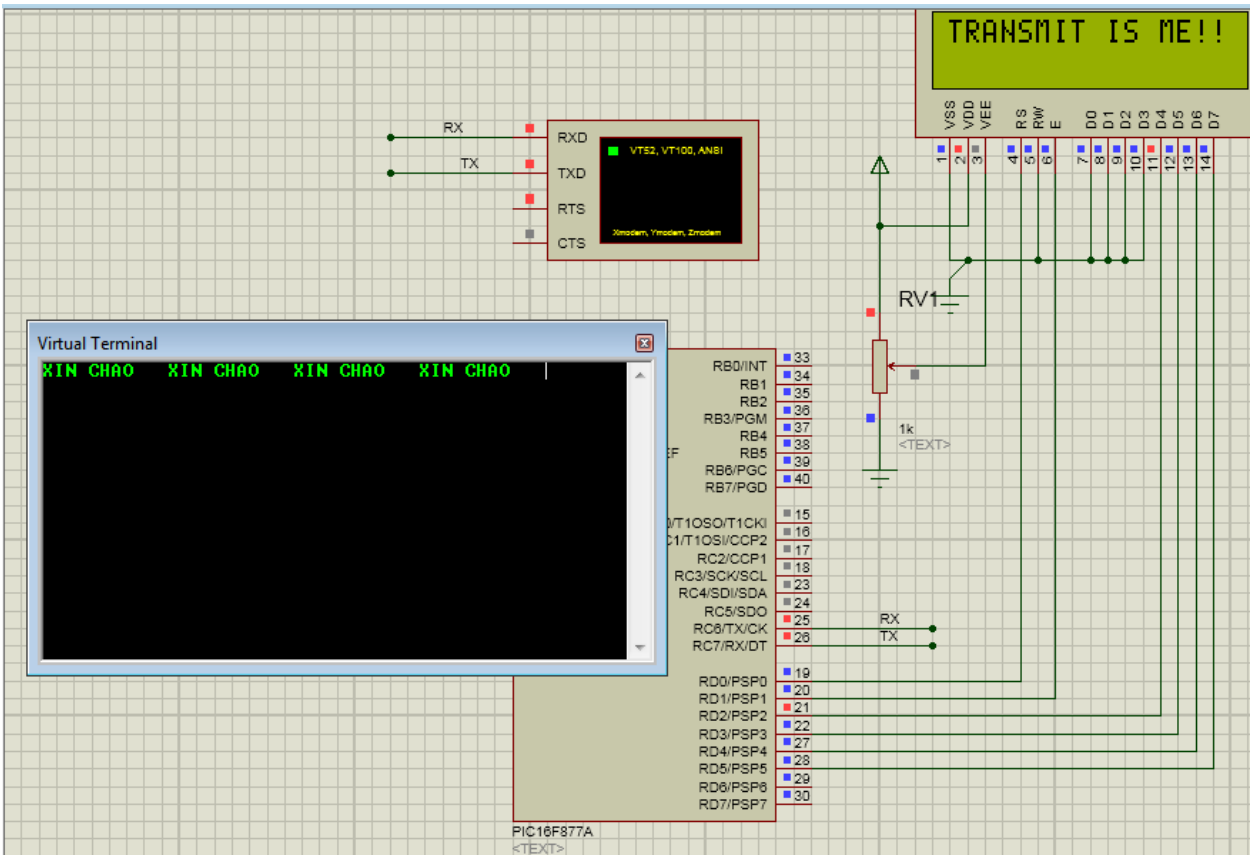
X = value in SPBRG (0 to 255)

Thông thường, với truyền thông nối tiếp không đồng bộ ta sử dụng tốc độ baudrate=9600.

Từ đó ta tính giá trị của X và đưa vào bit SPBRG

4. Ví dụ:

- **VD1: Gửi một chuỗi kí tự bằng uart**

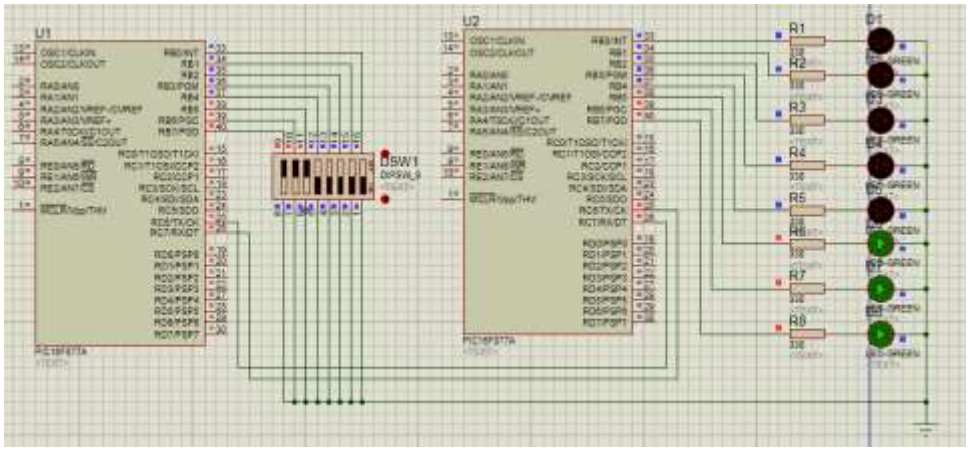



```
#include <xc.h>
#include <pic.h>
#include <pic16f877a.h>
#include "lcd.h"
// CONFIG
#pragma config FOSC = HS
#pragma config WDTE = ON
#pragma config PWRTE = OFF
#pragma config BOREN = ON
#pragma config LVP = ON
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
#define _XTAL_FREQ 20000000
void port_init() {
   TRISC6 = 0;// chân TX truyền dữ liệu
   TRISC7 = 1;// chân RX nhận dữ liệu
   TRISD = 0X00;// cấu hình chân lcd
   PORTD = 0X00;
}
void hien_thi(unsigned int value) {
   unsigned int nghin, tram, chuc, donvi;
```

```
    nghin = value / 1000;
    tram = (value - nghin * 1000) / 100;
    chuc = (value - nghin * 1000 - tram * 100) / 10;
    donvi = value % 10;
    LCD_PutChar(nghin + 0x30);
    LCD_PutChar(tram + 0x30);
    LCD_PutChar(chuc + 0x30);
    LCD_PutChar(donvi + 0x30);
}
```
// hàm cài đặt chế độ truyền
```
void uart_init() {
    BRGH = 1;                   //high speed baud rate
    TXSTAbits.SYNC = 0; //không đồng bộ
    SPBRG = 129;              // baurate=Fosc/(16(X+1)) với X=SPBRG

    TXSTAbits.TXEN = 1;
    RCSTAbits.SPEN = 1; // Serial port enable

    TXSTAbits.TX9 = 0; // không dùng bit thứ 9

    PIE1bits.TXIE = 1; // ngắt tràn
    INTCONbits.GIE = 1; // ngắt toàn cục
    INTCONbits.PEIE = 1; // ngắt ngoại vi
    }
void putchUART(unsigned char data) {
    while (TXIF == 0);
    TXREG = data;
}
void putsUART(unsigned char *data) {
    while (*data) {
        while (TXIF == 0);
        TXREG = *data++;
    }
}
void main() {
    port_init();
    uart_init();
    LCD_Init();
    while (1) {
    LCD_Gotoxy(0, 0);
    LCD_Puts("TRANSMIT IS ME!!!");
    putsUART("XIN CHAO   ");
    }
```

}

Ví dụ 2: truyền nhận dữ liệu giữa 2 vi điều khiển pic



Trong chương trình, tôi tạo một thư viện uart.h

//Khởi tạo uart

```
char UART_Init(const long int baudrate)
{
  unsigned int x;
  x = (_XTAL_FREQ - baudrate*64)/(baudrate*64); //SPBRG khi tốc độ baudrate thấp
  if(x>255)
  {
    x = (_XTAL_FREQ - baudrate*16)/(baudrate*16); //SPBRG cho  High Baud Rate
    BRGH = 1; //bit chọn tốc độ cao
  }
  if(x<256)
  {
    SPBRG = x; // giá trị vào thanh ghi SPBRG ứng với tốc độ baurate
    SYNC = 0; //chế độ không đồng bộ
    SPEN = 1; //cho phép truyền ở chế độ không đồng bộ
    TRISC7 = 1;  //RX
    TRISC6 = 0; //TX
    CREN = 1; //Cho phép nhận
    TXEN = 1; //cho phép truyền
    return 1;
  }
    return 0;
}
```

// kiểm tra dữ liệu truyền(trả về 1 nếu dữ liệu đã có còn trả về 0 nếu dữ liệu rỗng)

```
char UART_TX_Empty()
{
   return TRMT;
}
```

// kiểm tra xem bên nhận đã sẵn sàng nhận hay chưa. Nó sử dụng cờ RCIF để phát hiện việc nhận dữ liệu hoàn tất

```
char UART_Data_Ready()
{
   return RCIF;
}
```

// đọc một kí tự: nó sẽ đợi cho đến khi nhận đủ 8 bit dữ liệu từ bên gửi

```
char UART_Read() {

   while (!RCIF);
   return RCREG;
}
```

// đọc một văn bản, một chuỗi kí tự với chiều dài mong muốn

```
void UART_Read_Text(char *Output, unsigned int length) {
   int i;
   for (int i = 0; i < length; i++)
      Output[i] = UART_Read();
}
```

// viết một kí tự

```
void UART_Write(char data) {
   while (!TRMT);
   TXREG = data;
}
```

// viết một chuỗi kí tự

```
void UART_Write_Text(char *text) {
   int i;
   for (i = 0; text[i] != '\0'; i++)
      UART_Write(text[i]);
}
```

Code truyền:

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <pic16f877a.h>
#include "uart.h"
#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config BOREN = ON
 #pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF

void main()
{
  TRISB = 0xFF; //PORTB as Input
  nRBPU = 0; // trở treo portB
  UART_Init(9600);
  while(1)
  {
    UART_Write(PORTB);
    __delay_ms(100);
  }
}
```

Code nhận:

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <pic16f877a.h>
#include "uart.h"
#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
```

```
#pragma config BOREN = ON
 #pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF

void main()
{
  TRISB = 0x00; //PORTB as Output
  UART_Init(9600);
  while(1)
  {
    if(UART_Data_Ready())
      PORTB = UART_Read();
     __delay_ms(100);

  }
}
```

Chú ý: ở code truyền và nhận ta tạo thêm thư viện uart.h như trên
Tùy vào dữ liệu muốn gửi, chọn hàm phù hợp trong thư viện
Cũng tương tự như ví dụ 1, muốn gửi 1 chuỗi kí tự "hello", bạn dùng hàm UART_Write_Text("hello");
Tương tự với các hàm còn lại

BÀI 7: CHUẨN TRUYỀN THÔNG I2C

I. Chuẩn I2C

1. Giới thiệu

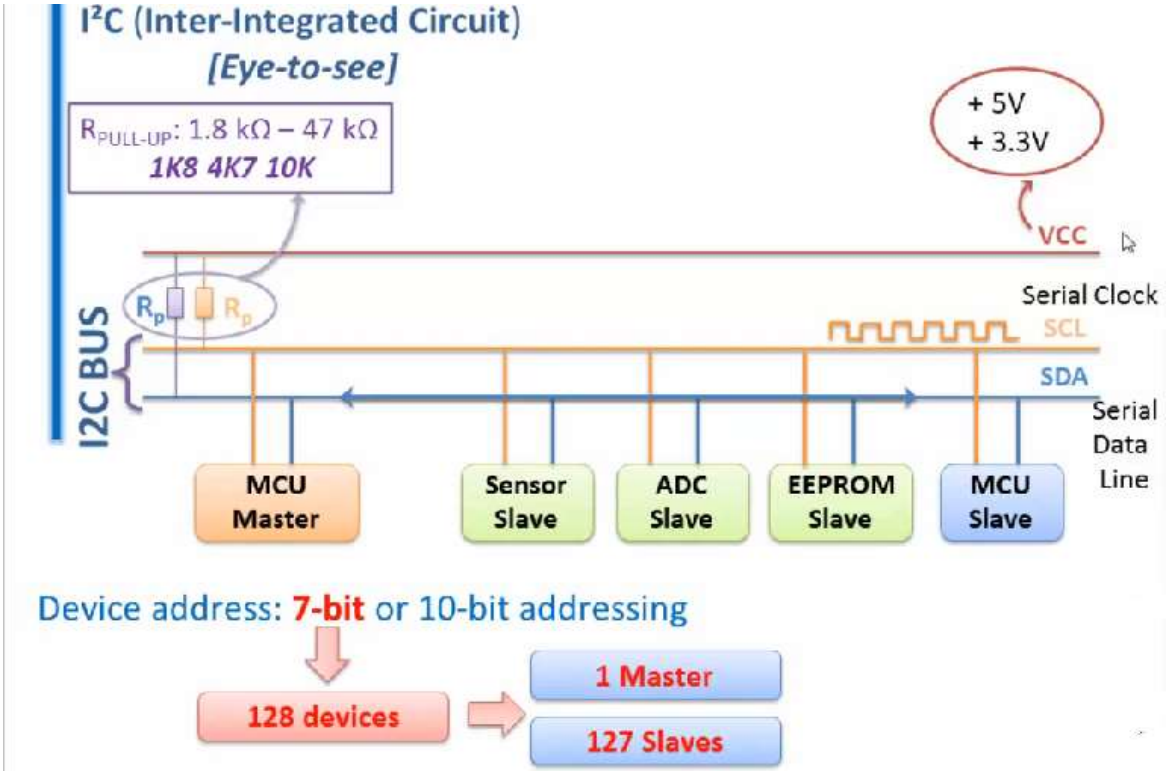
Viết tắt của Inter-Intergrated Circuit – là một bus nối tiếp do Philip phát triển. Trước đây I2C chủ yếu được dùng trong việc chế tạo các hệ thống điện tử sử dụng chip của Philip. Ngày nay I2C được sử dụng rộng rãi trong việc kết nối các thiết bị ngoại vi tốc độ thấp vào các mạch tích hợp.

I2C sử dụng 2 đường truyền tín hiệu 2 chiều (một đường clock và một đường data), sử dụng hiệu điện thế 5V và cùng được kéo lên cao(pull-ups) bằng điện trở. I2C hoạt động theo nhiều mode: mode chuẩn (standard mode) hoạt động ở tốc độ 100kbit/s, mode tốc độ thấp (low-speed mode) hoạt động ở tốc độ 10kbit/s. Tần số clock có thể cho xuống 0. I2C có sử dụng 7 bit để định địa chỉ, do đó trên một bus có thể có 112 nút (16 địa chỉ được sử dụng vào mục đích riêng).

Điểm mạnh của I2C là ở chỗ, một vi điều khiển có thể dùng để điều khiển cả một mạng thiết bị mà chỉ tốn 2 chân của vi điều khiển.

Chính vì nguyên nhân đó mà I2C và SPI là hai chuẩn giao tiếp được sử dụng nhiều nhất trong các IC đặc biệt là các VĐK 8 bít.

2. Sơ đồ tổng quan:



Giải thích:

- Nguồn dùng trong chuẩn giao tiếp I2C thường là 5v hoặc 3.3v
- Chuẩn I2C gồm 2 dây:

SCK: Dây xung clock

SDA: dây truyền dữ liệu

Trên bus I2C có thể ghép nối nhiều thiết bị với nhau. Thông thường sẽ có một 1 thiết bị đóng vai trò master và nhiều slave. Master đóng vai trò truyền tải dữ liệu trên bus I2C. các slave có thể là cảm biến,bộ chuyển đổi ADC, bộ nhớ trong EEPROM hoặc bộ nhớ ngoài RAM hoặc cũng có thể là các con MCU khác

- Khi dùng I2C ta cần điện trở kéo lên nguồn ở 2 dây SDA,SCL. Để đường truyền ổn định nhất, bằng cách thử nghiệm thực tế, người ta dùng điện trở 1k8,4k7,10k
- Dây SCL có nhiệm vụ truyền xung clock từ một con master sang một con slave khác để I2C có thể hoạt động
- Dây SDA để truyền tải dữ liệu:dữ liệu được truyền tải trên dây này là 2 chiều: có thể là dữ liệu truyền từ master sang các slave hoặc ngược lại, dữ liệu được gửi từ slave sang master.
- Có 2 cách định địa chỉ cho I2C là chế độ địa chỉ 7bit và 10 bit

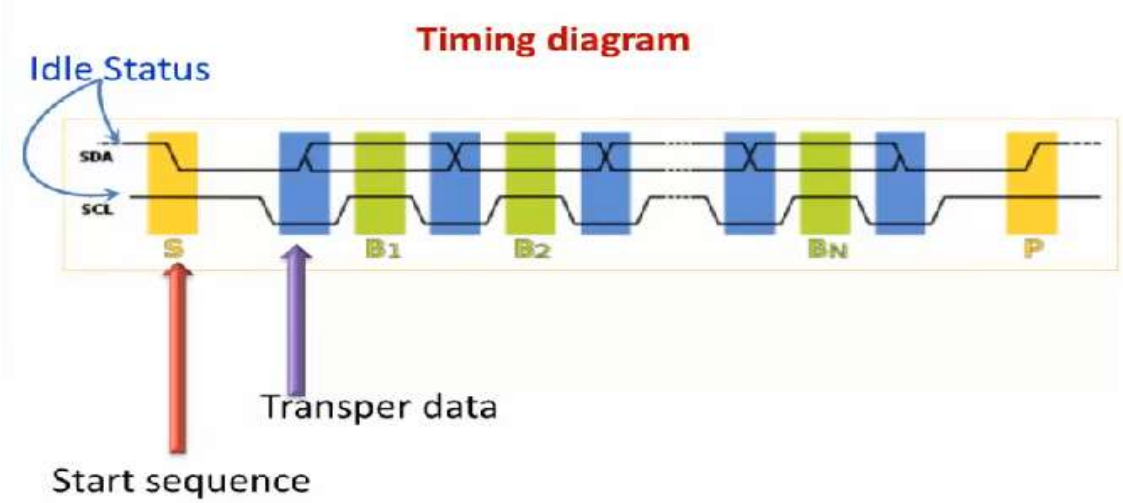
Nếu dùng cách định địa chỉ 7 bit ta có thể ghép nối tối đa 2^7=128 thiết bị. thông thường là 1 master và 127 slave. Tuy nhiên trên I2C người ta quy định 16 bus địa chỉ dự trữ cho I2C nên ta chỉ có tối đa 112 thiết bị

3. Các chế độ

- 1 master-1 slave
- One master- mutil slave

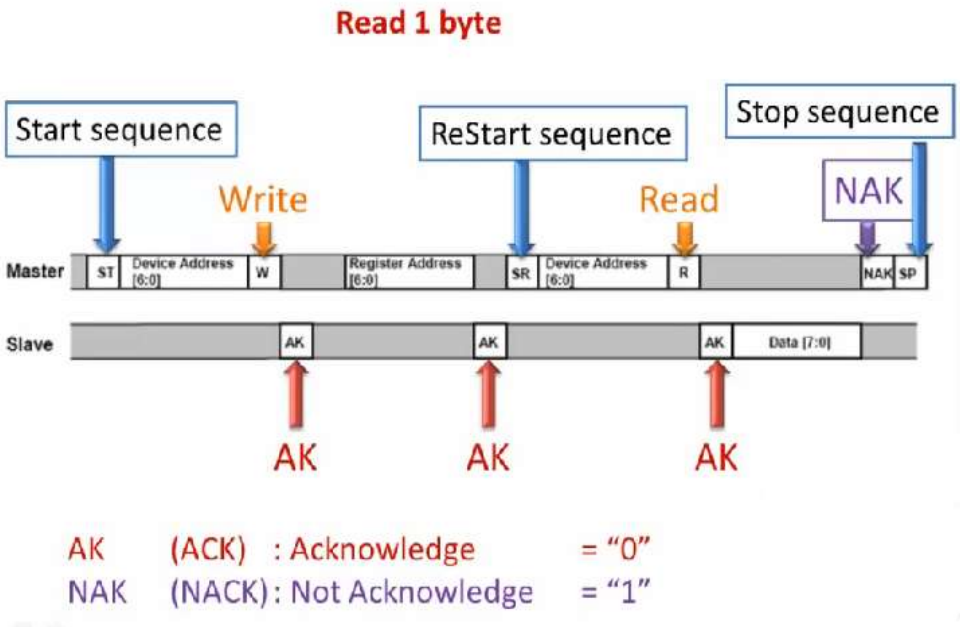
- Mutil master- mutil slave

4. **Tốc độ truyền:**
   - 100kb,10kb,
   - 400kb
   - 1Mb,4.3Mb bit/s
5. **Sơ đồ thời gian cho việc truyền nhận dữ liệu trên I2C**



Để bắt đầu quá trình truyền dữ liệu ta phải có 1 tín hiệu báo rằng bus bắt đầu chuẩn bị giao tiếp(gọi là tín hiệu start). Sau thời điểm đó thì dữ liệu được truyền đi. Mỗi bit được truyền đi thì dây SCL sẽ đếm lên 1 xung clock cho đến khi kết thúc quá trình truyền dữ liệu. khi bit cuối cùng được truyền đi ta cần phải có một tín hiệu báo (stop). sau khi tín hiệu start được phát ra thì bus I2C sẽ trở về chế độ chờ

6. **Đọc dữ liệu**



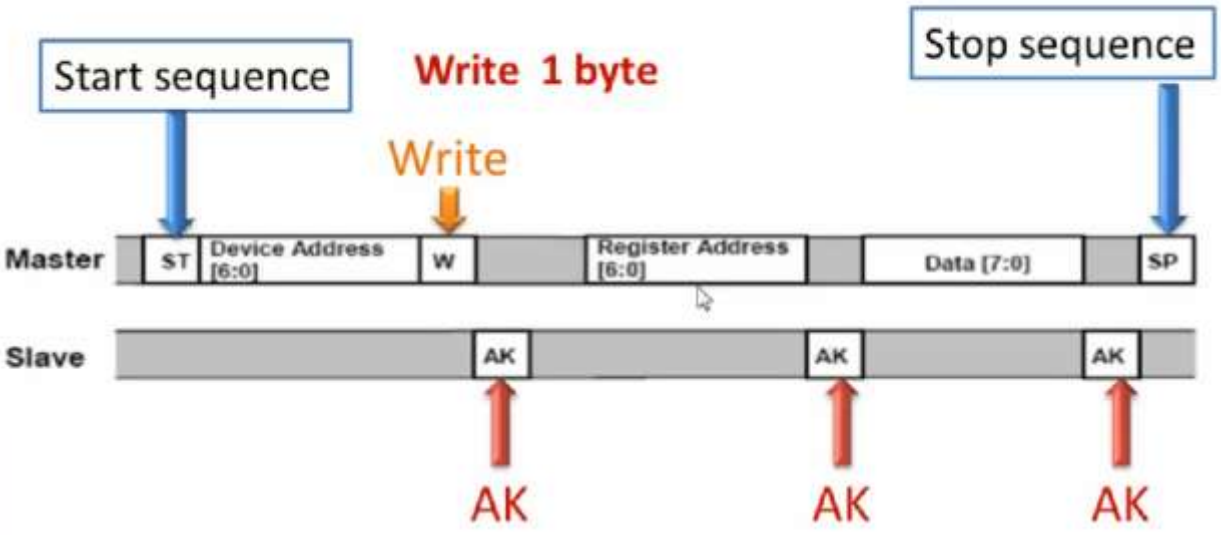
Sau khi gửi tín hiệu start, master sẽ gửi đi 7 bit địa chỉ thiết bị và một bit cuối cùng W(write). Địa chỉ W này là địa chỉ của slave và địa chỉ này do nhà sản xuất quy định. W=0 tương ứng với WRITE và =1 tương ứng với READ.

Slave sau khi nhận được 8 bit địa chỉ, nó sẽ gửi tín hiệu ACK báo cho master biết. Khi đó, master sẽ gửi đi 8 bit địa chỉ thanh ghi chứa trong slave mà master muốn truy cập. sau khi slave nhận được 8 bit địa chỉ tiếp theo thì slave gửi về tín hiệu ACK.

Quá trình của ta là đọc dữ liệu nên ta phải chia làm 2 giai đoạn là lệnh WRITE và READ. Nên ta cần có tín hiệu RESTART. Master lại gửi đi 7 bit địa chỉ thiết bị (trùng với địa chỉ thiết bị Write trước) và bit cuối cùng có giá trị =1 tương ứng với lệnh READ. Tín hiệu ACK được gửi lên và đồng thời slave sẽ gửi cho master 8 bit data. Nếu master nhận được sẽ phát ra tín hiệu NACK và gửi kèm điều kiện stop kết thúc quá trình đọc 1 byte

7. ghi dữ liệu

- tương tự như phần ghi dữ liệu



II. I2C trong VĐK PIC

1. Các thanh ghi sử dụng trong PIC16F877A

Trong Pic 16F877A có 3 thanh ghi điều khiển quá trình truyền và nhận dữ liệu: đó là

SSPSTAT (94h bank 1), SSPCON1 (14H bank 0) và SSPCON2 (91H bank 1).

Trong đó thì:

❖ **SSPSTAT: thanh ghi chứ bit trạng thái của chuẩn giao tiếp MSSP**

SSPSTAT: SYNC SERIAL PORT STATUS REGISTER (ADDRESS: 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

- **Bit 7: SMP:** Sample bit: bit lấy mẫu chọn tốc độ chuẩn

=1:Dùng tốc độ chuẩn 100kHz, 1MHz

=0 dùng tốc độ cao 400KHz)

- **bit 6: CKE:**

=1: cho phép MSBus

=0: không cho phép

- **Bit5**: **D/A**: Data/Address bit:

=0: Chỉ ra các byte vừ nhận được hoặc truyền là địa chỉ

=1: Chỉ ra các byte vừa nhận được hoặc truyền được dữ liệu

- **Bit 4**: **P**: STOP bit:

=1: vừa nhận được stop bit

=0: chưa nhận được stop bit

- **Bit3: S**: START bit

=1: vừa nhận được start bit

=0:chưa nhận được start bit

- **Bit2:** R/W báo rằng quá trình truyền vẫn đang diễn ra
  - Slave mode:
    1 = Read: đọc dữ liệu
    0 = Write: ghi dữ liệu
  - Master mode:
    1 = Transmit is in progress: đang truyền dữ liệu
    0 = Transmit is not in progress: không truyền dữ liệu
- **Bit 1:** Update Address (10-bit I2C mode only)

=1:vi điều khiển cần cập nhật thêm địa chỉ từ thanh ghi SSPADD

=0: không cần cập nhật thêm địa chỉ

- **Bit 0**: BF báo rằng SSPBUF vẫn đang đầy

=1: thanh ghi SSPBUF đang chứa dữ liệu truyền đi hoặc nhận được

=0: thanh ghi SSPBUF không có dữ liệu

❖ **SSPCON: thanh ghi điều khiển chuẩn giao tiếp MMSP**

SSPCON: SYNC SERIAL PORT CONTROL REGISTER (ADDRESS 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

- **bit 7**:WCOL : báo rằng có sự xếp chồng dữ liệu
  - Khi truyền dữ liệu ở chế độ master:

=1: dữ liệu mới được đưa vào thanh ghi SSPBUF trong khi chế độ truyền dữ liệu của I2C chưa sẵn sàng

=0: không có hiện tượng xảy ra

- Khi truyền ở chế độ slave:

=1: dữ liệu được đưa vào thanh ghi SSPBUF trong khi dữ liệu cũ chưa được truyền đi

=0: không có hiện tượng gì xảy ra

- **Bit 6 SSPOV**: Receive Overflow Indicator bit:
    - Khi nhận dữ liệu:

=1: dữ liệu mới được đưa vào thanh SSPBUF trong khi dữ liệu cũ chưa được đọc
=0:không có hiện tượng xảy ra

    - Khi truyền dữ liệu:Bit này không có tác dụng chỉ thị các trạng thái

- **bit 5**: SSPEN: enable chế độ I2C

=1: cho phép cổng giao tiếp MSSP(Các chân SDA,SCL)
=0:Không cho phép cổng giao tiếp MMSP

- **Bit 4**: CKP clock polarity select bit

=1: trạng thái chờ của xung clock ở mức logic cao
=0: trạng thái chờ của xung clock ở mức logic thấp (Để đảm bảo thời gian thiết lập dữ liệu)

- **bit 3-0:** SSPM3:SSPM0: chọn chế độ với chế độ I2C

0110 = I2C Slave mode, 7-bit address
0111 = I2C Slave mode, 10-bit address
1000 = I2C Master mode, clock = FOSC / (4 * (SSPADD+1))
1011 = I2C Firmware Controlled Master mode (slave idle)
1110 = I2C Firmware Controlled Master mode, 7-bit address with START and STOP bit interrupts enabled
1111 = I2C Firmware Controlled Master mode, 10-bit address with START and STOP bit interrupts enabled

❖ **SSPCON2: thanh ghi điều khiển các chế độ hoạt động của chuẩn giao tiếp i2c**

SSPCON2: MSSP CONTROL REGISTER 2 (I^2C MODE) (ADDRESS 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Bit 7: GCEN:
=1: cho phép ngắt khi địa chỉ 000h được nhận vào thanh ghi SSPSR
=0: không cho phép chế độ địa chỉ trên
Bit 6: ACKSTAT: bit này có tas dụng khi truyền dữ liệu ở chế độ i2c master
=1: nhận được xung ACK từ I2C slave
=0: chưa nhận được xung ACK
Bit 5: ACKDT: Acknowledge Data bit: bit này chỉ có tác dụng khi nhận dữ liệu ở chế độ i2c master
=1: nhận được xung ACK
=0: chưa nhận được xung ACK
Bit 4: ACKEN: Acknowledge Sequence Enable bit : bit này chỉ có tác dụng khi nhận dữ liệu ở chế độ i2c master

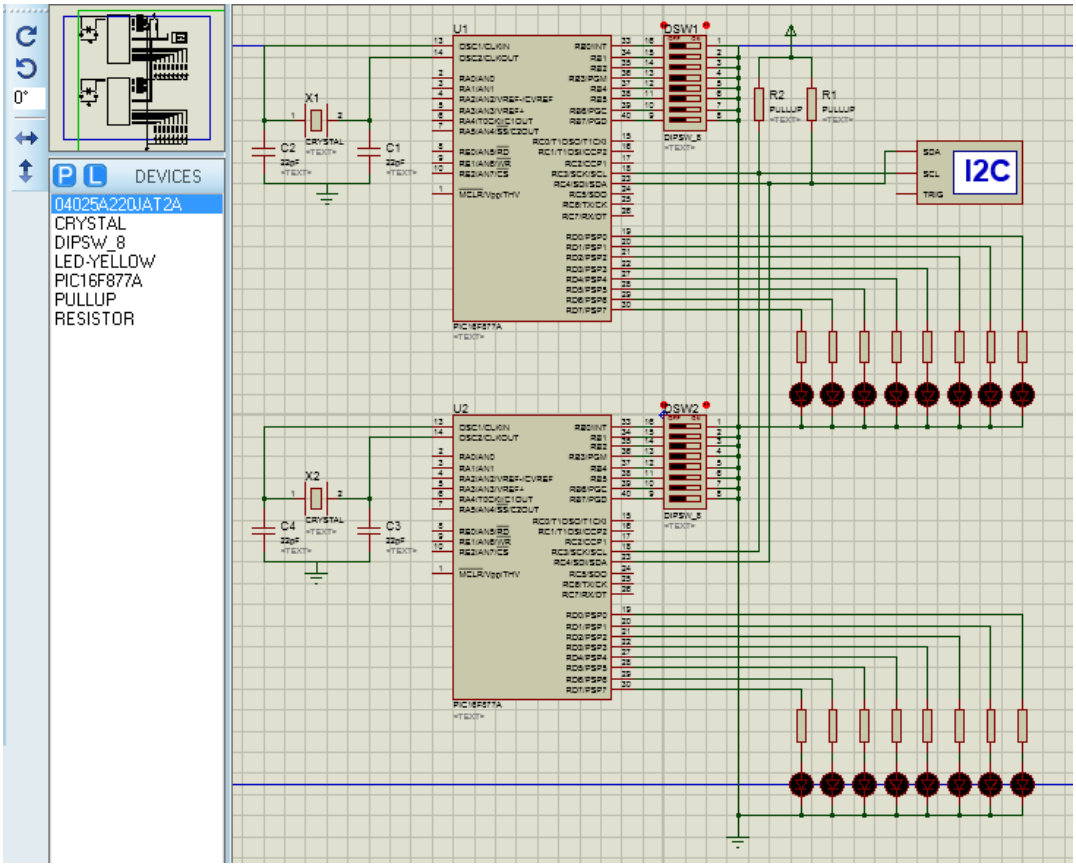
=1: cho phép xung ACK xuất hiện ở 2 pin SDA,SCL khi kết thúc quá trình nhận dữ liệu
=0: không cho phéo tác động trên
Bit 3: RCEN Receive Enable bit: Bit này chỉ có tác dụng ở chế độ I2C masteer mode
=1: cho phép nhận dữ liệu
=0: không cho phép
Bit 2: PEN Stop Condition Enable bit
=1: cho phép thiết lập điều kiện stop ở 2 pin SDA,SCL
=0: không cho phép tác động trên
Bit 1: RSEN Repeated Start Condition Enable bit
=1: cho phép thiết lập điều kiện start lặp lại liên tục ở 2 pin SDA,SCL
=0: Không cho phép tác động trên
Bit 0: SEN Start Condition Enable/Stretch Enable bit
ở chế độ master:
=1: Cho phép thiết lập điều kiện start ở 2 pin SDA,SCL
=0: không cho phép
ở chế độ slave:
=1: cho phép khóa xung clock từ pin SCL của I2C master
=0: không cho phép

Để điều khiển tốc độ baud của chế độ, người ta sử dụng thanh ghi SSPADD. I2C làm việc ở 3 chế độ chuẩn (chỉ tương đối) : 100Kb, 400Kb, 1Mb. Nếu ta dùng thạch anh 4M, và cần sử dụng tốc độ 100Kb ta phải nạp giá trì vào thanh ghi SSPADD là: 28H với tốc độ 400Kb ta cần giá trị là 0AH. Còn để lưu và nhận dữ liệu người ta dùng thanh ghi SSPBUF

Như vây tổng cộng có 5 thanh ghi được dùng đến : SSPSTAT, SSPCON, SSPCON2 (chọn chế độ và điều khiển đường truyền) SSPADD (khởi tạo tốc độ Baud) và SSPBUF dùng để lưu dữ liệu trong hai quá trình Receive, và Transmister.

Ví dụ:

Truyền nhận dữ liệu giữa 2 vdk pic bằng truyền thông i2c (điều khiển 8 led ở con pic16f877a này bằng công tắc DIPSWICH8 của con pic 16f877a khác và ngược lại)



Code:

MASTER:

```
#include <xc.h>
#include <pic16f877a.h>
#define _XTAL_FREQ 8000000
// CONFIG
#pragma config FOSC = XT
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
void I2C_Master_Init(const unsigned long c)
{
  SSPCON = 0b00101000;
  SSPCON2 = 0;
  SSPADD = (_XTAL_FREQ/(4*c))-1;
  SSPSTAT = 0;
  TRISC3 = 1;        //Setting as input
  TRISC4 = 1;        //Setting as input
}
```

```
void I2C_Master_Wait()
{
  while ((SSPSTAT & 0x04) || (SSPCON2 & 0x1F));
}
void I2C_Master_Start()
{
  I2C_Master_Wait();
  SEN = 1;
}
void I2C_Master_RepeatedStart()
{
  I2C_Master_Wait();
  RSEN = 1;
}
void I2C_Master_Stop()
{
  I2C_Master_Wait();
  PEN = 1;
}
void I2C_Master_Write(unsigned d)
{
  I2C_Master_Wait();
  SSPBUF = d;
}
unsigned short I2C_Master_Read(unsigned short a)
{
  unsigned short temp;
  I2C_Master_Wait();
  RCEN = 1;
  I2C_Master_Wait();
  temp = SSPBUF;
  I2C_Master_Wait();
  ACKDT = (a)?0:1;
  ACKEN = 1;
  return temp;
}
void main()
{
  nRBPU = 0;                  //Enable PORTB internal pull up resistor
  TRISB = 0xFF;                //PORTB as input
  TRISD = 0x00;                //PORTD as output
  PORTD = 0x00;                 //All LEDs OFF
  I2C_Master_Init(100000);      //Initialize I2C Master with 100KHz clock
  while(1)
  {
    I2C_Master_Start();         //Start condition
```

```
    I2C_Master_Write(0x30);     //7 bit address + Write
    I2C_Master_Write(PORTB);    //Write data
    I2C_Master_Stop();          //Stop condition
    __delay_ms(200);
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(0x31);     //7 bit address + Read
    PORTD = I2C_Master_Read(0); //Read + Acknowledge
    I2C_Master_Stop();          //Stop condition
    __delay_ms(200);
  }
}
```

SLAVE:

```
#include <xc.h>
#include <pic16f877a.h>
#define _XTAL_FREQ 8000000
#pragma config FOSC = XT
#pragma config WDTE = OFF
#pragma config PWRTE = OFF
#pragma config BOREN = OFF
#pragma config LVP = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config CP = OFF
short z;
void interrupt I2C_Slave_Read()
{
  if(SSPIF == 1)
  {
    SSPCONbits.CKP = 0;
    if ((SSPCONbits.SSPOV) || (SSPCONbits.WCOL))
    {
      z = SSPBUF; // Read the previous value to clear the buffer
      SSPCONbits.SSPOV = 0; // Clear the overflow flag
      SSPCONbits.WCOL = 0;  // Clear the collision bit
      SSPCONbits.CKP = 1;
    }
    if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW)
    {
      z = SSPBUF;
      while(!BF);
      PORTD = SSPBUF;
      SSPCONbits.CKP = 1;
    }
    else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW)
    {
      z = SSPBUF;
```

```
     BF = 0;
     SSPBUF = PORTB ;
     SSPCONbits.CKP = 1;
     while(SSPSTATbits.BF);
    }
   SSPIF = 0;
  }
}
void I2C_Slave_Init(short address)
{
  SSPSTAT = 0x80;
  SSPADD = address;
  SSPCON = 0x36;
  SSPCON2 = 0x01;
  TRISC3 = 1;   //Setting as input
  TRISC4 = 1;   //Setting as input
  GIE = 1;
  PEIE = 1;
  SSPIF = 0;
  SSPIE = 1;
}
void main()
{
  nRBPU = 0;          //Enables PORTB internal pull up resistors
  TRISB = 0xFF;        //PORTB as input
  TRISD = 0x00;        //PORTD as output
  PORTD = 0x00;        //All LEDs OFF
  I2C_Slave_Init(0x30); //Initialize as a I2C Slave with address 0x30
  while(1);
}
```

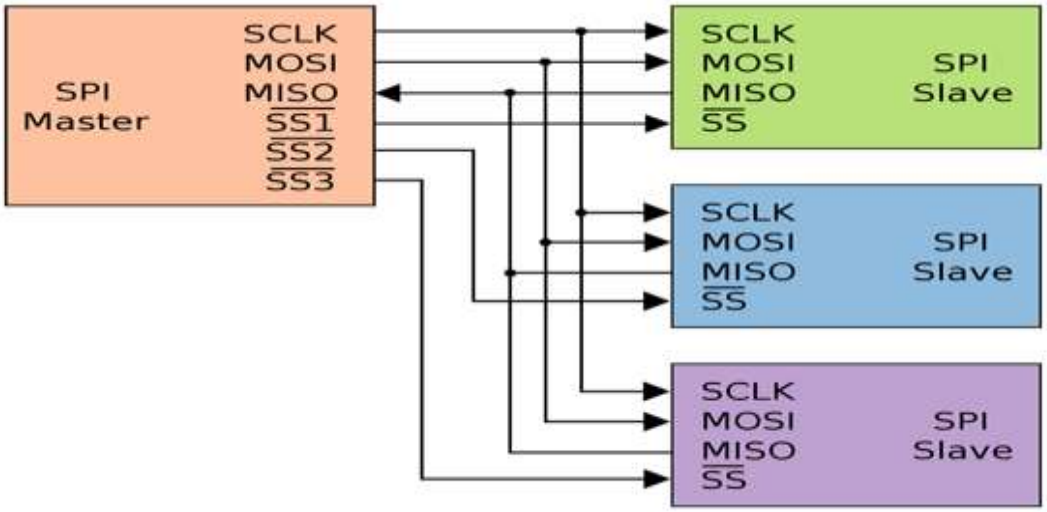
BÀI 8: TRUYỀN THÔNG SPI

I. Tìm hiểu chung

- **SPI (Serial Peripheral Bus)** là một chuẩn truyền thông nối tiếp tốc độ cao do hãng Motorola đề xuất. Đây là kiểu truyền thông Master-Slave, trong đó có 1 chip Master điều phối quá trình tuyền thông và các chip Slaves được điều khiển bởi Master vì thế truyền thông chỉ xảy ra giữa Master và Slave. SPI là một cách truyền song công (full duplex) nghĩa là tại cùng một thời điểm quá trình truyền và nhận có thể xảy ra đồng thời. SPI đôi khi được gọi là chuẩn truyền thông "4 dây" vì có 4 đường giao tiếp trong chuẩn này đó là SCK (Serial Clock), MISO (Master Input Slave Output), MOSI (Master Ouput Slave Input) và SS (Slave Select).
- **SCK:** Xung giữ nhịp cho giao tiếp SPI, vì SPI là chuẩn truyền đồng bộ nên cần 1 đường giữ nhịp, mỗi nhịp trên chân SCK báo 1 bit dữ liệu đến hoặc đi. Đây là điểm khác biệt với truyền thông không đồng bộ mà chúng ta đã biết trong chuẩn

UART. Sự tồn tại của chân SCK giúp quá trình truyền ít bị lỗi và vì thế tốc độ truyền của SPI có thể đạt rất cao. Xung nhịp chỉ được tạo ra bởi chip Master.

- **MISO – Master Input / Slave Output:** nếu là chip Master thì đây là đường Input còn nếu là chip Slave thì MISO lại là Output. MISO của Master và các Slaves được nối trực tiếp với nhau.
- **MOSI – Master Output / Slave Input:** nếu là chip Master thì đây là đường Output còn nếu là chip Slave thì MOSI là Input. MOSI của Master và các Slaves được nối trực tiếp với nhau.
- **SS – Slave Select:** SS là đường chọn Slave cần giap tiếp, trên các chip Slave đường SS sẽ ở mức cao khi không làm việc. Nếu chip Master kéo đường SS của một Slave nào đó xuống mức thấp thì việc giao tiếp sẽ xảy ra giữa Master và Slave đó. Chỉ có 1 đường SS trên mỗi Slave nhưng có thể có nhiều đường điều khiển SS trên Master, tùy thuộc vào thiết kế của người dùng.



Hình: Chuẩn truyền thông SPI

Đôi khi chuẩn SPI được sử dụng chỉ để ghi dữ liệu từ Master ra Slaver thì chân MISO sẽ không được dùng.

2. Cơ chế hoạt động: mỗi chip Master hay Slave có một thanh ghi dữ liệu 8 bits. Cứ mỗi xung nhịp do Master tạo ra trên đường giữ nhịp SCK, một bit trong thanh ghi dữ liệu của Master được truyền qua Slave trên đường MOSI, đồng thời một bit trong thanh ghi dữ liệu của chip Slave cũng được truyền qua Master trên đường MISO. Do 2 gói dữ liệu trên 2 chip được gởi qua lại đồng thời nên quá trình truyền dữ liệu này được gọi là "song công".

II. Truyền thông SPI với pic16f877a

- Chuẩn giao tiếp SPI cho phép truyền nhận đồng bộ. ta dùng 4 chân cho chuẩn giao tiếp này:
  - **RC5/SDO:** Ngõ ra dữ liệu dạng nối tiếp
  - **RC4/SDI/SDA:** Ngõ vào dữ liệu dạng nối tiếp
  - **RC3/SCK/SCL:** xung đồng bộ nối tiếp

- **RA5/AN4/SS/C2OUT:** chọn đối tượng giao tiếp khi ở chế độ slave mode

- **Các thanh ghi liên quan tới chuẩn giao tiếp SPI gồm:**
  - Thanh ghi **PIR**1: Chứa ngắt SSPIE
  - Thanh ghi **PIE1**: chứa bit cho phép ngắt SSPIE
  - Thanh ghi **TRISC**: điều khiển xuất nhập portc
  - Thanh ghi **TRISA**: điều khiển xuất nhập chân RA5/SS
  - Thanh ghi điều khiển **SSPCON**: điều khiển chuẩn giao tiếp SPI

SSPCON: SYNC SERIAL PORT CONTROL REGISTER (ADDRESS 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

Bit 7: WCOL=1: dữ liệ mới được đưa vào thanh ghi SSPBUF trong khi chưa truyền xong dữ liệu trước đó
WCOL=0: không có hiện tượng
Bit 6:SSPOV: Receive Overflow Indicalor bit(chỉ có tác dụng ở chế độ SPI Slave)
=1: dữ liệu trong bufer đệm bị tràn
=0: không có hiện tượng
Bit 5: SSPEN Synchronous Serial Port Enable bit
=1: Cho phép cổng giao tiếp MMSP(SCK,SDO, SDI,SS)
=0: Không cho phép cổng giao tiếp MMSP
Bit 4: CKP Clock Polarity Select bit
=1: trạng thái chờ của xung clock ở mức logic cao
=0: ở mức logic thấp
Bit 3-0: Synchronous Serial Mode Select bit: lựa chọn các chế độ hoạt động của MMSP

0101 Slave mode, xung clock lấy từ pin SCK, không cho phép pin điều khiển $\overline{SS}$ ($\overline{SS}$ là pin I/O bình thường).
0100 SPI Slave mode, xung clock lấy từ pin SCK, cho phép pin điều khiển $\overline{SS}$.
0011 SPI Master mode, xung clock bằng (ngõ ra TMR2)/2.
0010 SPI Master mode, xung clock bằng (F_{OSC}/64).
0001 SPI Master mode, xung clock bằng (F_{OSC}/16).
0000 SPI Master mode, xung clock bằng (F_{OSC}/4).
Các trạng thái không được liệt kê hoặc không có tác dụng điều khiển hoặc chỉ có tác dụng đối với chế độ I2C mode.

- Thanh ghi trạng thái **SSPSTAT**: chỉ cho phép đọc và ghi ở 2 bit trên, 6 bit còn lại chỉ cho phép đọc

SSPSTAT: SYNC SERIAL PORT STATUS REGISTER (ADDRESS: 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

Bit 7: SMB: SMP Sample bit
Master mode:
=1: dữ liệu được lấy mẫu tại thời điểm cuối xung clock
=0: dữ liệu được lấy mẫu tại thời điểm giữa xung clock
Slave mode: bit này phải được xóa về 0
Bit 6: CKE SPI Clock Select bit
=1: SPI master truyền dữ liệu khi xung clock chuyển từ trạng thái tích cực đến trạng thái chờ
=0: SPI master truyền dữ liệu khi xung clock chuyển từ trạng thái chờ đến trạng thái tích cực
Bit 5-1: chỉ dùng cho I2C
Bit 0: BF Buffer Status bit
=1: thanh ghi đệm SSPBUF đã có dữ liệu
=0: thanh ghi đệm SSPBUF chưa có dữ liệu

- Thanh ghi đóng vai trò là buffer truyền nhận **SSPBUF**: dữ liệu truyền nhận sẽ được đưa vào thanh ghi này
- Thanh ghi dịch dữ liệu **SSPSR**:dùng để ghi dữ liệu vào ra

- Khi sử dụng chuẩn giao tiếp SPI, trước tiên ta cần đưa các giá trị thích hợp vào 2 thanh ghi SSPCON và SSPSTAT.
- Trong quá trình nhận dữ liệu, khi dữ liệu đưa vào từ chân RC4/SDI/SDA trong thanh ghi SSPSR đã sẵn sàng(nhận đủ 8 bit) thì dữ liệu được đưa vào thanh ghi SSPBUF, bit chỉ thị trạng thái của bộ đệm BF(SSPSTAT<0>) sẽ được set báo hiệu bộ đệm đã đầy, đồng thời cờ ngắt SSPIF được set. Bit BF tự động reset về 0 khi dữ liệu trong thanh ghi SSPBUF được đọc vào. Bộ đệm kép cho phép đọc các byte tiếp theo trước khi byte dữ liệu đó được đọc vào.
- Quá trình truyền dữ liệu tương tự nhưng ngược lại. dữ liệu cần truyền đưa vào thanh ghi SSPBUF đồng thời đưa vào thanh ghi SSPSR, khi đó, cờ hiệu BF được set. Dữ liệu được dịch từ thanh ghi SSPSR và đưa ra ngoài chân RC5/SDO. Ngắt sẽ xảy ra khi quá trình dịch dữ liệu hoàn tất.