Chương I: GIỚI THIỆU PIC 16F87X

I. GIỚI THIỆU:

1. Đặc Tính Kỹ Thuật Và Sơ Đồ Chân Của PIC16F87x:

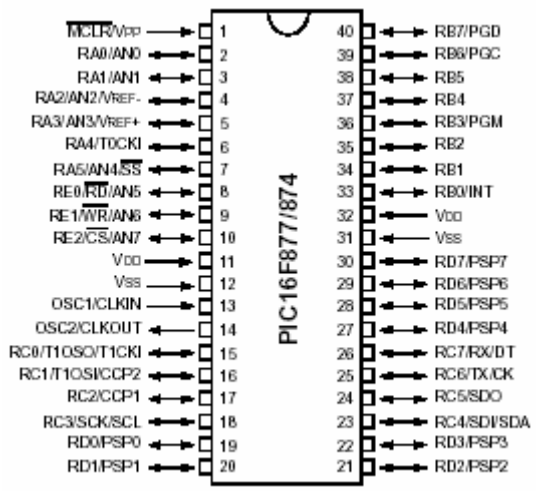
*a. **Đặc tính kỹ thuật của PIC16F87x**:*

- Vi điều khiển PIC16F87x là loại CPU có đặc tính cao được tích hợp trên công nghệ RISC.
- Tập lệnh gồm có 35 lệnh, mỗi lệnh là một từ đơn.
- Tất cả các lệnh (ngoại trừ các lệnh rẽ nhánh) được thực hiện trong 2 chu kỳ máy.
- Tần số xung nhịp có thể đạt tới 20 MHz.
- Bộ nhớ chương trình được tích hợp theo công nghệ FLASH với dung lượng 8Kx14 từ (8192 lệnh), 368x8 byte bộ nhớ RAM, 256x8 byte bộ nhớ EEPROM.
- 14 nguồn ngắt (Bao gồm cả ngắt cứng và ngắt mềm).
- Ngăn xếp phần cứng 8 mức
- Gồm 3 chế độ định địa chỉ: trực tiếp, gián tiếp và định địa chỉ tương đối.
- Reset khi mở nguồn.
- Gồm 3 bộ định thời (Timer 0, Timer 1 và Timer 2).
- Bộ định thời đáp ứng theo sự kiện của ngoại vi (Watchdog Timer).
- Mã bảo vệ lập trình được.
- Tiết kiệm năng lượng ở chế độ chờ (SLEEP Mode).
- Thay đổi nguồn xung nhịp.
- Khả năng thiết kế đầy đủ.
- Nguồn cung cấp từ 2V đến 5.5 V.
- Tích hợp mạch lập trình trong thông qua cổng nối tiếp (ICSP) và lập trình với nguồn đơn 5 V.
- Dòng điện mức cao ở các đường dữ liệu có thể đạt tới 25mA.
- PIC16F87x gồm 3 bộ định thời/ bộ đếm là timer0, timer1 và timer2. Trong đó Timer0 và timer2 là timer 8 bit con timer 1 là timer 16 bit. Tất cả các timer đều có thể thực hiện như một bộ đếm (counter). Tất cả các timer đều có thể đặt trước tỷ lệ.
- 2 bộ bắt giữ ngõ vào, so sánh và điều rộng xung. Bắt giữ ngõ vào 16 bit với độ phân giải 12,5ns, so sánh 16 bit với độ phân giải 200ns, điều rộng xung với độ phân giải 10 bit.
- 8 kênh ADC 10 bit.
- Đường truyền nối tiếp bất đồng bộ với mode chủ (SPI) và mode chủ tớ (I^2C).

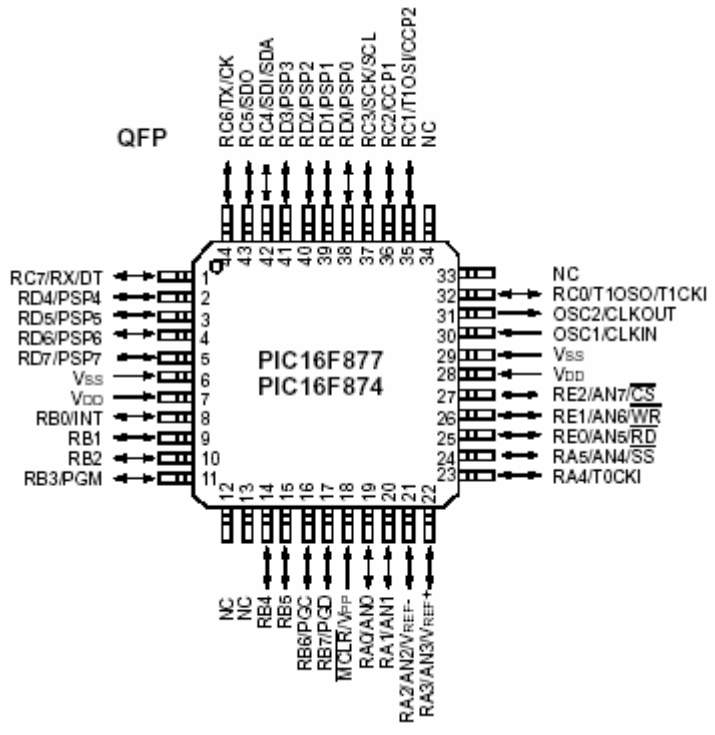
- Truyền nhận đa năng đồng bộ và bất đồng bộ với 9 bit địa chỉ.
- Giao tiếp dữ liệu song song 8 bit.

b. ***Sơ đồ chân của PIC16F87x***:

PIC16F87x được tích hợp dưới dạng như hình vẽ 1.1



a) Tích hợp dạng PDIP



b) Tích hợp dạng QFP

Hình 1.1 Sơ đồ chân của PIC16F87x

Chi tiết các chân của MCU PIC16F877 được mô tả ở bảng sau:

Bảng mô tả chân của MCU PIC16F877:

Số chân	Tên chân linh kiện	Chức năng
1	$\overline{MCLR}$/Vdd	- Reset CPU - Cấp nguồn Vdd cho chip ở chế độ lập trình

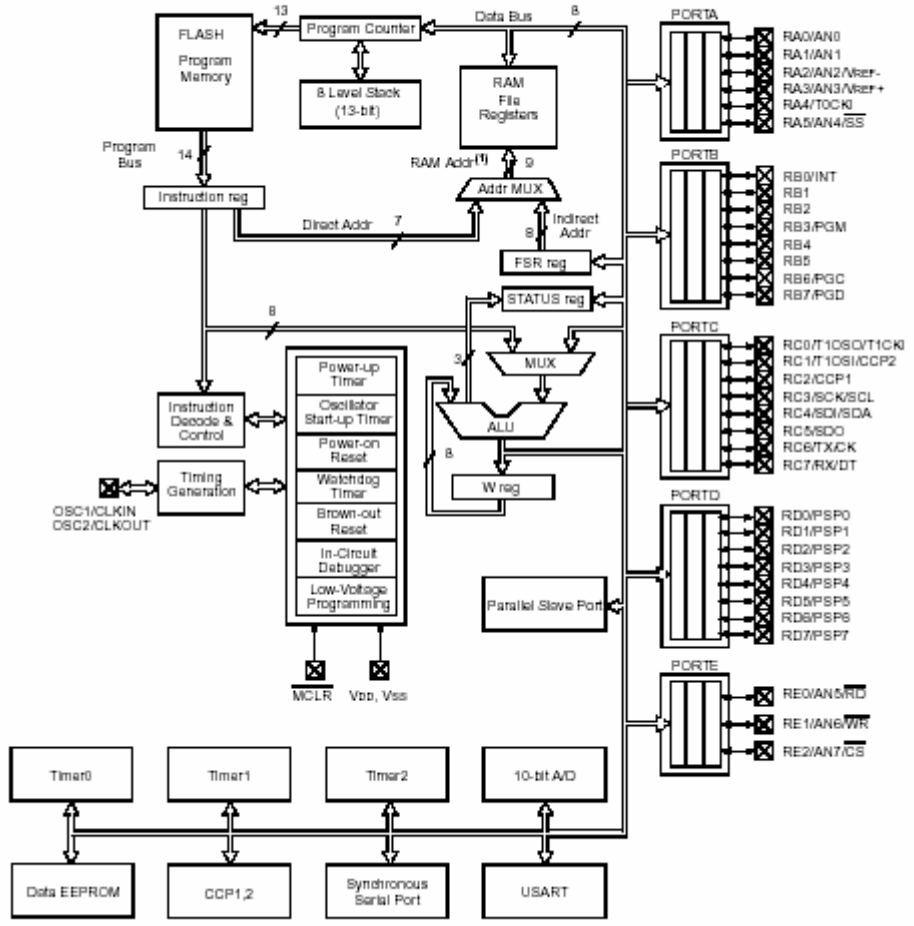
2	RA0/AN0	- Bit D0 của cổng giao tiếp song song (Port A) - Ngõ vào analog 0 của ADC
3	RA1/AN1	- Bit D1 của cổng giao tiếp song song (Port A) - Ngõ vào analog 1 của ADC
4	RA2/AN2/VREF-	- Bit D2 của cổng giao tiếp song song (Port A). - Ngõ vào analog 2 của ADC. - Cực điện thế thấp của nguồn điện áp chuẩn cho ADC (dùng điện áp chuẩn bên ngoài).
5	RA3/AN3/VREF+	- Bit D3 của cổng giao tiếp song song (Port A). - Ngõ vào analog 3 của ADC. - Cực điện thế cao của nguồn điện áp chuẩn cho ADC (dùng điện áp chuẩn bên ngoài).
6	RA4/T0CKI	- Bit D4 của cổng giao tiếp song song (Port A). - Nguồn cấp xung nhịp từ bên ngoài cho timer 0
7	RA5/AN4/$\overline{SS}$	- Bit D5 của cổng giao tiếp song song (Port A). - Ngõ vào analog 4 của ADC. - Chọn tớ (Slave) cho cộng nối tiếp bất đồng bộ
8	RE0/$\overline{RD}$/AN5	- Bit D0 của cổng giao tiếp song song (Port E). - Ngõ vào analog 5 của ADC. - Cho phép đọc dữ liệu song song từ các ngoại vi.
9	RE1/$\overline{RD}$/AN6	- Bit D1 của cổng giao tiếp song song (Port E). - Ngõ vào analog 6 của ADC. - Cho phép ghi dữ liệu song song từ các ngoại vi.
10	RE2/$\overline{RD}$/AN7	- Bit D2 của cổng giao tiếp song song (Port E). - Ngõ vào analog 7 của ADC. - Cho phép chọn ngoại vi.
11	Vdd	- Nguồn cung cấp cho vi điều khiển
12	Vss	- Mass nguồn cung cấp
13	OSC1/CLKIN	- Cấp xung nhịp.
14	OSC2/CLKOUT	- Cấp xung nhịp.
15	RC0/T1OSO/T1CKI	- Bit D0 của cổng giao tiếp song song (Port C). - Bộ phát xung nhịp từ timer1 - Cấp xung nhịp từ bên ngoài cho timer1
16	RC1/T1OSI/CCP2	- Bit D1 của cổng giao tiếp song song (Port C). - Cấp xung nhịp từ bên ngoài cho timer1 - Bộ bắt giữ ngõ vào 2 hoặc so sánh ngõ ra 2 hoặc ngõ ra điều rộng xung 2.
17	RC2/CCP1	- Bit D2 của cổng giao tiếp song song (Port C). - Bộ bắt giữ ngõ vào 1 hoặc so sánh ngõ ra 1 hoặc ngõ ra điều rộng xung 1.
18	RC3/SCK/SCL	- Bit D3 của cổng giao tiếp song song (Port C). - Xung nhịp ngõ vào hoặc ngõ ra trong chế độ truyền

		đồng bộ trong cả 2 chế độ SPI và I^2C.
19	RD0/PSP0	- Bit D0 của cổng giao tiếp song song (Port D).
20	RD1/PSP1	- Bit D1 của cổng giao tiếp song song (Port D).
21	RD2/PSP2	- Bit D2 của cổng giao tiếp song song (Port D).
22	RD3/PSP3	- Bit D3 của cổng giao tiếp song song (Port D).
23	RC4/SDI/SDA	- Bit D4 của cổng giao tiếp song song (Port C). - Đường nhập dữ liệu trong mode SPI hoặc đường xuất nhập dữ liệu trong mode I^2C.
24	RC5/SDO	- Bit D5 của cổng giao tiếp song song (Port C). - Đường xuất dữ liệu trong mode SPI.
25	RC6/TX/CK	- Bit D6 của cổng giao tiếp song song (Port C). - Đường truyền dữ liệu nối tiếp bất đồng bộ trong chế độ truyền bất đồng bộ. - Xung nhịp trong chế độ truyền nối tiếp đồng bộ.
26	RC7/RX/DT	- Bit D7 của cổng giao tiếp song song (Port C). - Đường nhận dữ liệu nối tiếp bất đồng bộ trong chế độ truyền bất đồng bộ. - Đường dữ liệu trong chế độ truyền nối tiếp đồng bộ.
27	RD4/PSP4	- Bit D4 của cổng giao tiếp song song (Port D).
28	RD5/PSP5	- Bit D5 của cổng giao tiếp song song (Port D).
29	RD6/PSP6	- Bit D6 của cổng giao tiếp song song (Port D).
30	RD7/PSP7	- Bit D7 của cổng giao tiếp song song (Port D).
31	VSS	- Nguồn cung cấp cho vi điều khiển
32	VDD	- Mass nguồn cung cấp
33	RB0/INT	- Bit D0 của cổng giao tiếp song song (Port B). - Ngắt ngoài.
34	RB1	- Bit D1 của cổng giao tiếp song song (Port B).
35	RB2	- Bit D2 của cổng giao tiếp song song (Port B).
36	RB3/PGM	- Bit D3 của cổng giao tiếp song song (Port B). - Chế độ lập trình
37	RB4	- Bit D4 của cổng giao tiếp song song (Port B).
38	RB5	- Bit D5 của cổng giao tiếp song song (Port B).
39	RB6/PGC	- Bit D6 của cổng giao tiếp song song (Port B).
40	RB7/PGD	- Bit D7 của cổng giao tiếp song song (Port B).
		-

c. ***Cấu trúc của PIC16F87x***

PIC16F87x có cấu trúc như hình 1.2

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Hình 1.2 Sơ đồ cấu trúc bên trong của PIC16F877.

Các khối trong cấu trúc của PIC như sau:

- Port A: là cổng giao tiếp dữ liệu song song và một số chức năng khác như trong bảng mô tả chân.
- Port B: là cổng giao tiếp dữ liệu song song và một số chức năng khác như trong bảng mô tả chân.
- Port C: là cổng giao tiếp dữ liệu song song và một số chức năng khác như trong bảng mô tả chân.
- Port D: là cổng giao tiếp dữ liệu song song và một số chức năng khác như trong bảng mô tả chân.
- Port E: là cổng giao tiếp dữ liệu song song và một số chức năng khác như trong bảng mô tả chân.
- Program memory: là bộ nhớ chứa chương trình ứng dụng, được chế tạo theo công nghệ FLASH, cho phép đọc ghi nhanh và có thể nạp xoá nhiều lần.
- Program counter: Bộ đếm chương trình làm nhiệm vụ chứa địa chỉ của các lệnh chứa trong bộ nhớ chương trình để thực thi.
- 8 level stack 13 bit: Là ngăn xếp 13 bit với 8 mức. Ngăn xếp dùng để chứa các dữ liệu trung gian khi chương trình bị ngắt và tuân theo quy luật vào trước ra sau (FILO).

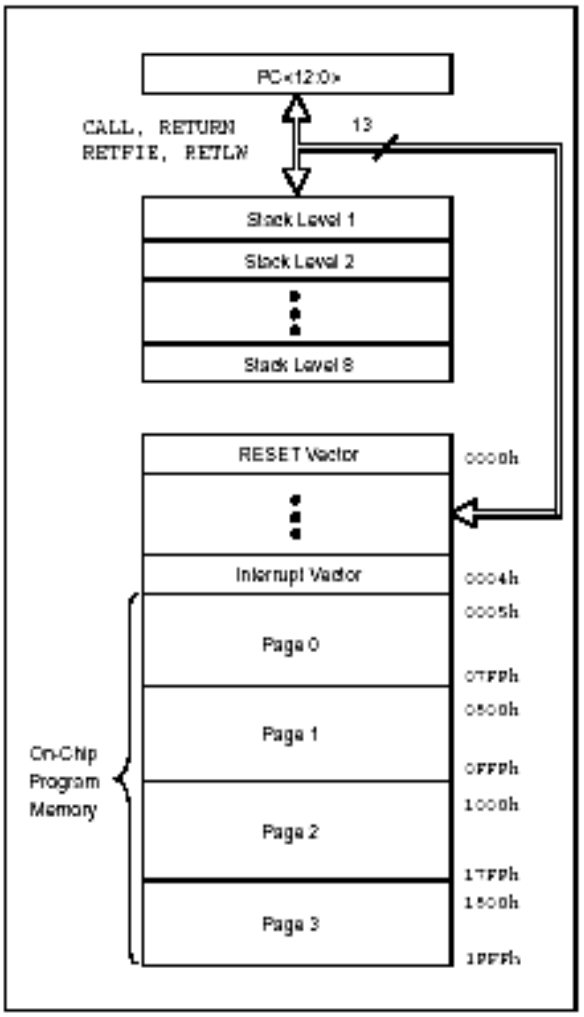
- RAM: là bộ nhớ dùng để lưu trữa các giá trị trung gian trong quá trình tính toán khi thực thi chương trình.
- Addr MUX: Bộ dồn kênh địa chỉ.
- Instruction Reg: Thanh ghi lệnh. Đây là thanh ghi 14 bit tương ứng với một lệnh của PIC.
- Instruction decode & control: Bộ giải mã lệnh và điều khiển.
- Timing generation: Bộ phát xung thời gian.
- Status reg: Thanh ghi trạng thái. Thanh ghi này dùng để lưu trữ các trạng thái của CPU.
- ALU: Khối xử lý số học, khối này thực hiện các phép toán số học trong PIC.

2. <u>Tổ Chức Bộ Nhớ Trong PIC16F87x</u>:

Bộ nhớ trong MCU PIC16F87x gồm 3 phần: Bộ nhớ chương trình, bộ nhớ dữ liệu RAM và bộ nhớ dữ liệu EEPROM được tổ chức như sau:

a. ***<u>Tổ chức của bộ nhớ chương trình và ngăn xếp</u>***:

Bộ nhớ chương trình và ngăn xếp trong MCU PIC16F87x được chế tạo theo công nghệ FLASH gồm 8 mức ngăn xếp 13 bit và 8Kx14Word, được tổ chức như hình 1.3



Hình 1.3 Tổ chức bộ nhớ chương trình và ngăn xếp trong MCU PIC16F877

Trong bộ nhớ chương trình thì các word có địa chỉ từ 0000h đến 0004h là các vetor ngắt. Khi có sự kiện ngắt xảy ra thì bộ đếm chương trình (PC) sẽ nhảy đến vùng

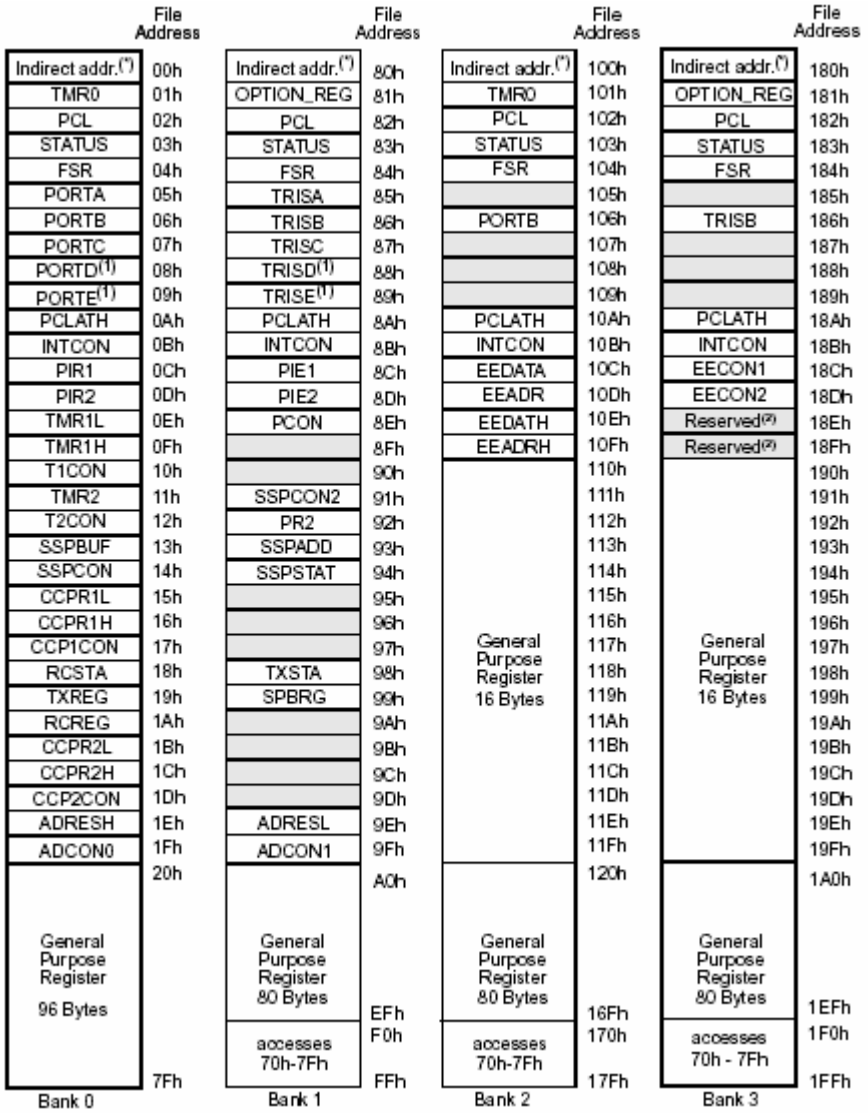
địa chỉ này. Mỗi vector ngắt tương ứng với một sự kiện ngắt (ví dụ khi reset MCU thì PC sẽ nhảy tới vector có địa chỉ 0000h). Chương trình ứng dụng sẽ được lưu trữ từ word có địa chỉ 0005h tới 1FFFh.

b. ***Bộ nhớ dữ liệu RAM***:

Bộ nhớ dữ liệu RAM được chia thành 4 dãy chứa các thanh ghi đa dụng và các thanh ghi chức năng đặc biệt. Việc chọn lựa các dãy được thực hiện thông qua 2 bit RB0 và RB1 trong thanh ghi trạng thái.

RB1	RB0	Dãy
0	0	Dãy 0
0	1	Dãy 1
1	0	Dãy 2
1	1	Dãy 3

Mỗi dãy có thể mở rộng lên tới 128 byte. Vùng địa chỉ thấp của mỗi dãy dùng để chứa các thanh ghi chức năng đặc biệt, vùng còn lại dùng để chứa các thanh ghi đa dụng. Bộ nhớ dữ liệu RAM được tổ chức như hình 1.4.



Hình 1.4 Phân chia vùng nhớ dữ liệu RAM trong MCU PIC16F877

Các thanh ghi chức năng đặc biệt là các thanh ghi được CPU và các ngoại vi sử dụng để điều khiển như mong muốn hoạt động của thiết bị. Chi tiết các thanh ghi điều chức năng đặc biệt được giới thiệu trong bảng sau:

Địa chỉ	Tên thanh ghi	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Giá trị khi Reset
Dãy 0										
00h	INDF	Dùng nội dung của thanh ghi FSR để định địa chỉ bộ nhớ dữ liệu.								0000 0000
01h	TMR0	Thanh ghi khối timer 0								xxxx xxxx
02h	PCL	Byte thấp của bộ đếm chương trình (PC)								0000 0000
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
04h	FSR	Con trỏ bộ nhớ dữ liệu gián tiếp								xxxx xxxx
05h	PORT A	-	-	Chốt dữ liệu cho port A khi xuất						--0x 0000
06h	PORT B	Chốt dữ liệu cho port Bkhi xuất								
07h	PORT C	Chốt dữ liệu cho port Ckhi xuất								
08h	PORT D	Chốt dữ liệu cho port D hi xuất								
09h	PORT E	-	-	-	-	-	RE2	RE1	RE0	---- -xxx
0Ah	PCLATH	-	-	-	Ghi bộ đệm trên 5 bit cho PC					---0 0000
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
0Dh	PIR2	-	(5)	-	EEIF	BCLIF	-	-	CCP2IF	-r-9 0--0
0Eh	TMR1L	Byte thấp của thanh ghi giữ trong thanh ghi 16 bit TMR1								xxxx xxxx
0Fh	TMR1H	Byte cao của thanh ghi giữ trong thanh ghi 16 bit TMR1								xxxx xxxx
10h	T1CON	-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TRM1ON	--00 0000
11h	TMR2	Thanh ghi khối timer 2								0000 0000
12h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 000
13h	SSPBUF	Thanh ghi đệm truyền nhận dữ liệu công nối tiếp đồng bộ								xxxx xxxx
14h	SSPCON	WCON	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	--00 0000
15h	CCPR1L	Thanh ghi bắt giữ ngõ vào, so sánh và điều rộng xung 1 (Bit có trọng số nhỏ nhất)								xxxx xxxx
16h	CCPR1H	Thanh ghi bắt giữ ngõ vào, so sánh và điều rộng xung 1 (Bit có trọng số cao nhất)								xxxx xxxx
17h	CCP1CON	-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
19h	TXREG	Thanh ghi truyền dữ liệu nối tiếp								0000 0000
1Ah	RCREG	Thanh ghi nhận dữ liệu nối tiếp								0000 0000
1Bh	CCPR2L	Thanh ghi bắt giữ ngõ vào, so sánh và điều rộng xung 2 (Bit có trọng số nhỏ nhất)								
1Ch	CCPR2H	Thanh ghi bắt giữ ngõ vào, so sánh và điều rộng xung 2 (Bit có trọng số cao nhất)								
1Dh	CCP2CON	-	-	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000
1Eh	ADRESH	Thanh ghi chứa byte cao của ngõ ra ADC								
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/ $\overline{DONE}$	-	ADON	0000 00-0
Dãy 1										
80h	INDF	Dùng nội dung của thanh ghi FSR để định địa chỉ bộ nhớ dữ liệu.								0000 0000
81h	OPTION-REG	$\overline{RBPU}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111
82h	PCL	Byte thấp của bộ đếm chương trình (PC)								0000 0000
83h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
84h	FSR	Con trỏ bộ nhớ dữ liệu gián tiếp								xxxx xxxx
85h	TRISA	-	-	Thanh ghi dữ liệu trực tiếp của port A						--11 1111
86h	TRISB	Thanh ghi dữ liệu trực tiếp của port B								1111 1111
87h	TRISC	Thanh ghi dữ liệu trực tiếp của port C								1111 1111
88h	TRISD	Thanh ghi dữ liệu trực tiếp của port D								1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	-	Thanh ghi dữ liệu trực tiếp của port E			0000 x111
8Ah	PCLATH	-	-	-	Ghi bộ đệm trên 5 bit cho PC					---0 0000
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
8Dh	PIE2	-	(5)	-	EEIE	BCLIE	-	-	CCP2IE	-r-9 0--0
8Eh	PCON	-	-	-	-	-	-	$\overline{POR}$	$\overline{BOR}$	---- --qq
8Fh	-									
90h	-									
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000

92h	PR2	Thanh ghi chu kỳ timer 2								1111 1111
93h	SSPADD	Thanh ghi địa chỉ công nối tiếp đồng bộ trong mode I^2C								0000 0000
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000
95h	-									
96h	-									
97h	-									
98h	TXSTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010
99h	SPBRG	Thanh ghi phát tốc độ baud								0000 0000
9Ah	-									
9Bh	-									
0Ch	-									
9Dh	-									
9Eh	ADRESL	Thanh ghi chứa byte thấp của ngõ ra ADC								xxxx xxxx
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000
Dãy 2										
100h	INDF	Dùng nội dung của thanh ghi FSR để định địa chỉ bộ nhớ dữ liệu.								0000 0000
101h	TMR0	Thanh ghi khối timer 0								xxxx xxxx
102h	PCL	Byte thấp của bộ đếm chương trình (PC)								0000 0000
103h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
104h	FSR	Con trỏ bộ nhớ dữ liệu gián tiếp								xxxx xxxx
105h	-									
106h	PORT B	Chốt dữ liệu cho port Bkhi xuất								
107h	-									
108h	-									
109h	-									
10Ah	PCLATH	-	-	-	Ghi bộ đệm trên 5 bit cho PC					---0 0000
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
10Ch	EEDATA	Thanh ghi chứa dữ liệu byte thấp của eeprom								xxxx xxxx
10Dh	EEADR	Thanh ghi chứa byte thấp địa chỉ của eeprom								xxxx xxxx
10Eh	EEDATH	-	-	Thanh ghi chứa dữ liệu byte cao của eeprom						xxxx xxxx
10Fh	EEADRH	-	-	-	Thanh ghi chứa byte cao địa chỉ của eeprom					xxxx xxxx
Dãy 3										
180h	INDF	Dùng nội dung của thanh ghi FSR để định địa chỉ bộ nhớ dữ liệu.								0000 0000
181h	OPTION-REG	$\overline{RBPU}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111
182h	PCL	Byte thấp của bộ đếm chương trình (PC)								0000 0000
183h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
184h	FSR	Con trỏ bộ nhớ dữ liệu gián tiếp								xxxx xxxx
185h	-									
186h	TRISB	Thanh ghi dữ liệu trực tiếp của port B								1111 1111
187h	-									
188h	-									
189h	-									
18Ah	PCLATH	-	-	-	Ghi bộ đệm trên 5 bit cho PC					---0 0000
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
18Ch	EECON1	EEPGD	-	-	-	WREER	WREN	WR	RD	x---x000
18Dh	EECON2	Thanh ghi điều khiển EEPROM								---- ----
18Eh	-									
18Fh	-									

b.1 ***Thanh ghi trạng thái***:

Thanh ghi trạng thái dùng để chúa trạng thái của khối xử lý số học ALU, Khi sử dụng thanh ghi này cần chú ý vì có một số bit trong thanh ghi sẽ thay đổi tuỳ vào trạng thái của ALU.

Thanh ghi trạng thái có địa chỉ: 03h, 83h, 103h, 183h

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

Bit 7 Bit 0

- Bit 7 (IRP): Bit chọn dãy thanh ghi (Sử dụng trong chế độ dịnh địa chỉ gián tiếp). Bit IRP có thể đọc, ghi.

+ IRP = 1: Dãy 2, 3 (100h – 1FFh).
+ IRP = 0: Dãy 0, 1 (00h - FFh).
- Bit 6,5 (RP1:RP0): 2 bit chọn dãy thanh ghi (Sử dụng trong chế độ định địa chỉ trực tiếp). Bit IRP có thể đọc.

RP1:RP0	Dãy	Địa chỉ
1 1	Dãy 3	180h – 1FFh
1 0	Dãy 2	100h – 17Fh
0 1	Dãy 1	80h – FFh
0 0	Dãy 0	00h – 7Fh

- Bit 4 ($\overline{TO}$): Bit báo tràn bộ định thời. Bit $\overline{TO}$ chỉ được đọc.
+ $\overline{TO}$ = 1 khi MCU tiêu thụ công suất hoặc được tác động bằng lệnh CLRWDT hoặc lệnh SLEEP.
+ $\overline{TO}$ = 0 khi tràn bộ định thời.
- Bit 3 ($\overline{PD}$): Bit báo MCU không tiêu thụ năng lượng. Bit $\overline{PD}$ là bit chỉ đọc.
+ $\overline{PD}$ = 1 khi MCU tiêu thụ năng lượng hoặc được tác động bằng lệnh CLRWDT.
+ $\overline{PD}$ = 0 khi thực hiện lệnh SLEEP (MCU ở trạng thái chờ).
- Bit 2 (Z): Bit Zero. Bit Z là bit có thể đọc ghi.
+ Z = 1 nếu kết quả của 1 phép toán số học hay luận lý (logic) là 0.
+ Z = 1 nếu kết quả của 1 phép toán số học hay luận lý (logic) là khác 0.
- Bit 1 (DC): là cờ nhớ và mượn, bit này thay đổi bởi các lệnh ADDWF, ADDLF, SUBWF, SUBLF.
+ DC = 1 nếu kết quả của bit thứ 4 (Bit 3 trong một byte hoặc 1 word) có nhớ.
+ DC = 0 nếu kết quả của bit thứ 4 (Bit 3 trong một byte hoặc 1 word) không có nhớ.
- Bit 0 (C): cờ báo tràn (cờ nhớ). Bit này thay đổi bởi các lệnh ADDWF, ADDLF, SUBWF, SUBLF.
+ C =1 nếu kết quả có tràn ở bit có trọng số cao nhất.
+ C =0 nếu kết quả không có tràn ở bit có trọng số cao nhất.

b.2 ***Thanh ghi chọn lựa (Option Register)***:

Thanh ghi chọn lựa có địa chỉ 81h và 181h, thanh ghi này là loại thanh ghi có thể đọc ghi.

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Bit 7 Bit 0

- RBPU : Bit cho phép port B được treo lên nguồn.
+ RBPU = 1: Cấm treo port B lên nguồn.
+ RBPU = 0: cho phép treo port B lên nguồn bởi một cổng chốt dữ liệu riêng.
- INTEDG: Bit thay đổi cạnh (Cạnh lên hay cạnh xuống) của ngắt ngoài.
+ INTEDG =1: Cho phép ngắt cạnh lên của chân RB0/INT.
+ INTEDG =0: Cho phép ngắt cạnh xuống của chân RB0/INT.
- T0CS: Chọn nguồn xung nhịp cho timer 0.

+ T0CS = 1: Chọn nguồn xung nhịp từ bên ngoài (Chân RA4/T0CKI).
+ T0CS = 0: Chọn nguồn xung nhịp từ bên trong MCU.
- T0SE: Chọn cạnh nguồn xung nhịp cho timer 0.
+ T0SE =1: Chọn cạnh xuống của xung nhịp cho timer 0 (chân RA4/T0CKI).
+ T0SE =0: Chọn cạnh lên của xung nhịp cho timer 0 (chân RA4/T0CKI).
- PSA: Bit ấn định tỷ lệ cho timer.
+ PSA =1 : Ấn định tỷ lệ cho WDT.
+ PSA =0 : Ấn định tỷ lệ cho timer 0.
- PS2:PS0: Bit chọn tốc độ tỷ lệ

PS2:PS0	Tốc độ timer 0	Tốc độ WDT
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

b.3 ***Thanh ghi INTCON***:

Thanh ghi INTCON có địa chỉ 0Bh, 8Bh, 10Bh, 18Bh là thanh ghi điều khiển ngắt. Thanh ghi INTCON có thể đọc ghi được.

GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Bit 7 Bit 0

- GIE: Bit cho phép ngắt toàn cục.
+ GIE = 1 cho phép tất cả các ngắt làm việc.
+ GIE = 0 cấm tất cả các ngắt làm việc.
- PEIE: Bit cho phép ngắt từ các ngoại vi.
+ PEIE = 1 cho phép tất cả các ngắt của ngoại vi.
+ PEIE = 0 cấm tất cả các ngắt của ngoại vi.
- T0IE: Bit cho phép ngắt của timer 0 khi tràn.
+ T0IE = 1 cho phép ngắt của timer 0.
+ T0IE = 0 cấm ngắt của timer 0.
- INTE: Bit cho phép ngắt từ nguồn ngắt ngoài.
+ INTE = 1: Cho phép ngắt RB0/INT là việc.
+ INTE = 0: Cấm phép ngắt RB0/INT là việc.
- RBIE: Bit cho phép thay đổi ngắt của port B.
+ RBIE = 1: Cho phép thay đổi ngắt Port B.
+ RBIE = 0: Không cho phép thay đổi ngắt Port B.
- T0IF: Cờ ngắt báo tràn timer 0.
+ T0IF = 1: Thanh ghi TMR0 tràn.

+ T0IF = 0: Thanh ghi TMR0 không tràn.

- INTF: Cờ báo ngắt ngoài RB0/INT

+ INTF = 1: Ngắt ở chân RB0/INT đã xảy ra, bit này phải được xoá bằng chương trình.

+ INTF = 0: Ngắt ở chân RB0/INT chưa xảy ra.

- RBIF: Cờ báo thay đổi ngắt Port B.

+ RBIF = 1: RB7:RB4 thay đổi trạng thái, bit này phải được xoá bằng phần mềm.

+ RBIF = 0: RB7:RB4 không thay đổi trạng thái.

b.4 ***Thanh ghi PIE1***:

Thanh ghi PIE1 có địa chỉ 8Ch chứa các bit cho phép các ngắt riêng của ngoại vi.

PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

Bit 7 Bit 0

- PSPIE: Bit cho phép ngắt của cổng vào ra song song.

+ PSPIE =1: Cho phép ngắt cổng xuất nhấp song song.

+ PSPIE =0: không cho phép ngắt cổng xuất nhấp song song.

- ADIE: Bit cho phép ngắt của bộ chuyển đổi ADC.

+ ADIE =1: Cho phép ngắt của ADC.

+ ADIE =0: không cho phép ngắt của ADC.

- RCIE: bit cho phép ngắt của đường dữ liệu USART.

+ RCIE =1: Cho phép ngắt của nhận dữ liệu USART.

+ RCIE =0: không cho phép ngắt của đường nhận dữ liệu USART.

- TXIE: Bit cho phép ngắt đường truyền USART.

+ TXIE =1: Cho phép ngắt của truyền dữ liệu USART.

+ TXIE =0: không cho phép ngắt của đường truyền dữ liệu USART.

- SSPIE: bit cho phép ngắt của đường dữ liệu nối tiếp đồng bộ.

+ SSPIE =1: Cho phép ngắt của truyền dữ liệu nối tiếp đồng bộ.

+ SSPIE =0: không cho phép ngắt của đường truyền dữ liệu nối tiếp đồng bộ.

- CCP1IE: Cho phép ngắt của bộ bắt giữ ngõ vào, so sánh ngõ ra và PWM 1.

+ CCP1IE = 1: cho phép.

+ CCP1IE = 0 Cấm.

- TMR2IE: Cho phép ngắt của timer 2.

+ TMR2IE = 1: cho phép.

+ TMR2IE = 0: Cấm.

- TMR1IE: Cho phép ngắt của timer 1.

+ TMR1IE = 1: cho phép.

+ TMR1IE = 0: Cấm.

b.5 ***Thanh ghi PIR1***:

Thanh ghi PIR1 có địa chỉ 0Ch chứa các bit cờ của các ngắt riêng của ngoại vi.

PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

Bit 7 Bit 0

- PSPIF: Bit cờ của ngắt của cổng vào ra song song.

+ PSPIF =1: Đọc hoặc ghi cổng xuất nhấp song song (Bit này phải được xoá bằng phần mềm).

+ PSPIF =0: không đọc ghi công song song.

- ADIF: Bit cờ của ngắt của bộ chuyển đổi ADC.

+ ADIF = 1: ADC chuyển đổi xong.

+ ADIF = 0: ADC chưa chuyển đổi xong.

- RCIE: bit cờ ngắt nhận dữ liệu USART.

+ RCIF =1: Bộ đệm đầy.

+ RCIF =0: Bộ đệm rỗng.

- TXIF: bit cờ ngắt truyền dữ liệu USART.

+ TXIF =1: Bộ đệm đầy.

+ TXIF =0: Bộ đệm rỗng.

- SSPIF: Cờ ngắt của cổng nối tiếp đồng bộ.

+ SSPIF = 1: khi có điều kiện ngắt SSP xảy ra, bit này phải được xoá bằng phần mềm.

+ SSPIF = 0: khi không có điều kiện ngắt SSP xảy ra.

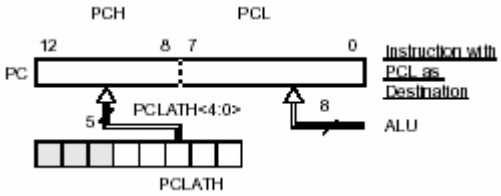
- CCP1IF: Cờ ngắt của bộ bắt giữ ngõ vào, so sánh ngõ ra và PWM 1.

+ CCP1IF = 1: khi có sự kiện bắt giữ ngõ vào hoặc so sánh ngõ ra ở thanh ghi TMR1.

+ CCP1IF = 0: khi không có sự kiện bắt giữ ngõ vào hoặc so sánh ngõ ra ở thanh ghi TMR1.

c. ***PCL và PCLATH***:

Bộ đếm chương trình có độ rộng 13 bit. Byte thấp được chứa trong thanh ghi PCL, đây là thanh ghi có thể đọc ghi. Những bit cao của bộ đếm chương trình (PCH) không đọc được nhưng có thể được ghi gián tiếp thông qua thanh ghi PCLATH. Hình 1.5 mô tả hoạt động ghi dữ liệu cho bộ đếm chương trình.



Hình 1.5 hoạt động ghi dữ liệu cho PC

Bộ đếm chương trình thường được nạp giá trị khi có ngắt xảy ra hoặc khi gọi chương trình con. Khi có ngắt xảy ra hoặc gọi chương trình con, PC sẽ cất giá trị hiện tại của nó vào ngăn xếp (Stack) và nạp vào giá trị mới là địa chỉ của vector ngắt hoặc chương trình con để thực thi, sau khi thực hiện xong chương trình xử lý ngắt hoặc chương trình con, PC sẽ lấy giá trị trong stack để thực hiện tiếp chương trình. Trong PIC16F87x có bộ ngăn xếp cứng gồm 8 mức với độ rộng 13 bit hoạt động theo cơ chế vào trước ra sau.

d. ***Trang bộ nhớ***:

Tất cả các MCU trong họ PIC16F87x đều có thể định địa chỉ liên tục trong khối bộ nhớ 8K Word. Tuy nhiên các lệnh nhảy và rẽ nhánh trong PIC chỉ cho phép thực hiện trong phạm vi 11 bit địa chỉ (2K word) của bộ nhớ (Một trang bộ nhớ). Do đó khi

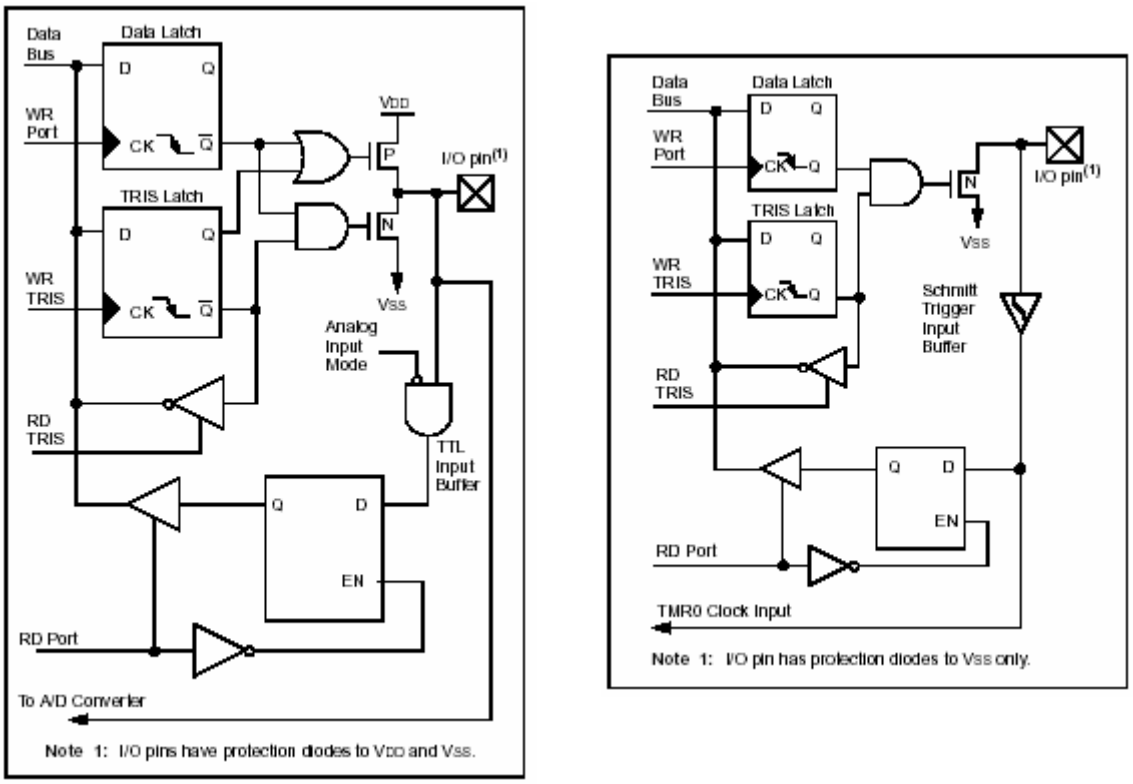
cần nhảy hoặc rẽ nhánh trong phạm vi xa hơn một trang bộ nhớ, người lập trình phải can thiệp vào PC thông qua thanh ghi PCLATH để chọn trang nhớ. Vì các bit

3. <u>Cổng vào ra</u>:

Cổng vào ra là nơi giao tiếp giữa MCU với các thiết bị bên ngoài. MCU PIC16F87x có 5 cổng vào ra: Port A, Port B, Port C, Port D và Port E. Một số chân ở các cổng vào ra của PIC được tích hợp nhiều chức năng, chức năng xuất nhập thông thường và các chức năng đặc biệt.

a. ***<u>Port A</u>***:

Port A là cổng xuất nhập 2 chiều gồm 6 ngõ. Thanh ghi dữ liệu tương ứng là TRISA. Bit TRISA được set lên 1 tương ứng với Port A là ngõ nhập, Bit TRISA được reset về 0 tương ứng với Port A là cổng xuất dữ liệu. Việc đọc trạng thái các chân của port A chính là đọc thanh ghi và việc xuất dữ liệu ra các chân của Port A chính là việc ghi dữ liệu lên cổng chốt dữ liệu. Chân RA4 tích hợp thêm chức năng là chân cấp xung nhịp cho Timer 0. Chân RA4 là ngõ vào Schmitt Trigger và là ngõ ra cực máng để hở. Các chân còn lại của Port A đều tương thích TTL và CMOS. Hình 1.6 mô tả cấu trúc của Port A.



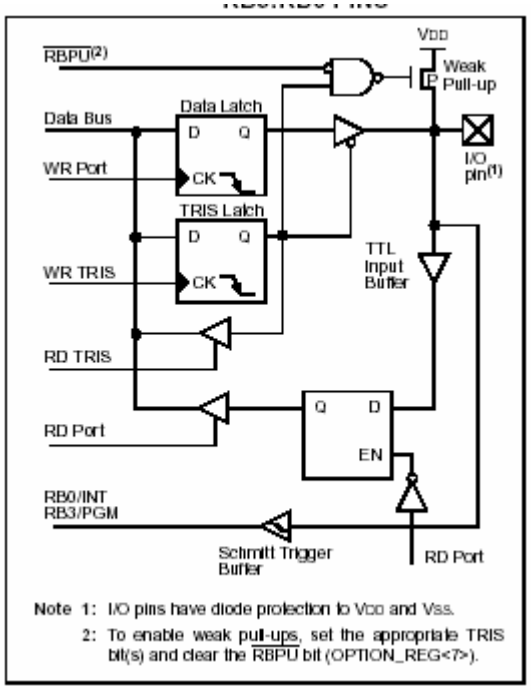
a) Sơ đồ khối các chân RA5, RA3:RA0 b) Sơ đồ khối chân RA4

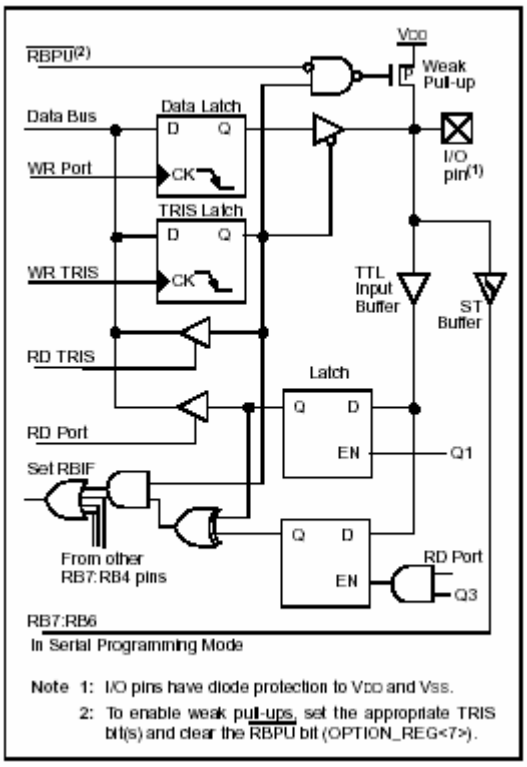
Hình 1.6 Cấu trúc của Port A

b. ***<u>Port B</u>***:

Port B là cổng xuất nhập 2 chiều gồm 8 ngõ. Thanh ghi dữ liệu tương ứng là TRISB. Bit TRISB được set lên 1 tương ứng với Port B là ngõ nhập, Bit TRISB được reset về 0 tương ứng với Port B là cổng xuất dữ liệu. Việc đọc trạng thái các chân của port B chính là đọc thanh ghi và việc xuất dữ liệu ra các chân của Port B chính là việc

ghi dữ liệu lên cổng chốt dữ liệu. Port B có 3 chân được tích hợp thêm chức năng lập trình điện áp thấp là RB3/PGM, RB6/PGC, RB7/PGD. Các chân của Port B đều có điện trở kéo lên nguồn trong MCU và đều tương thích TTL. Hình 1.7 mô tả cấu trúc của Port B.



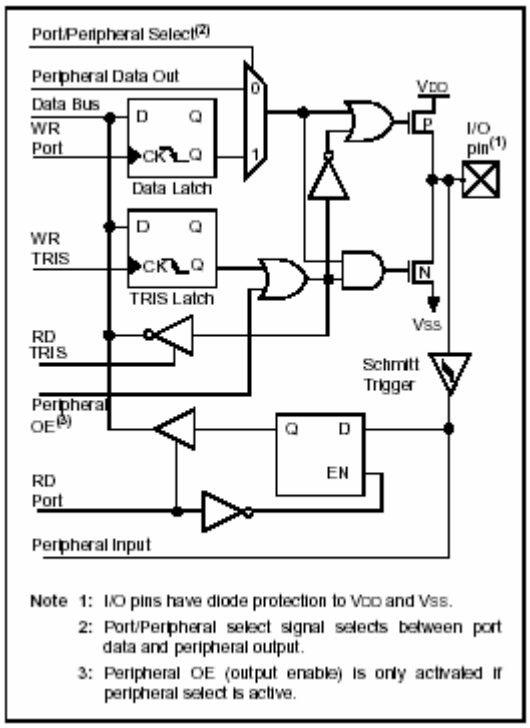


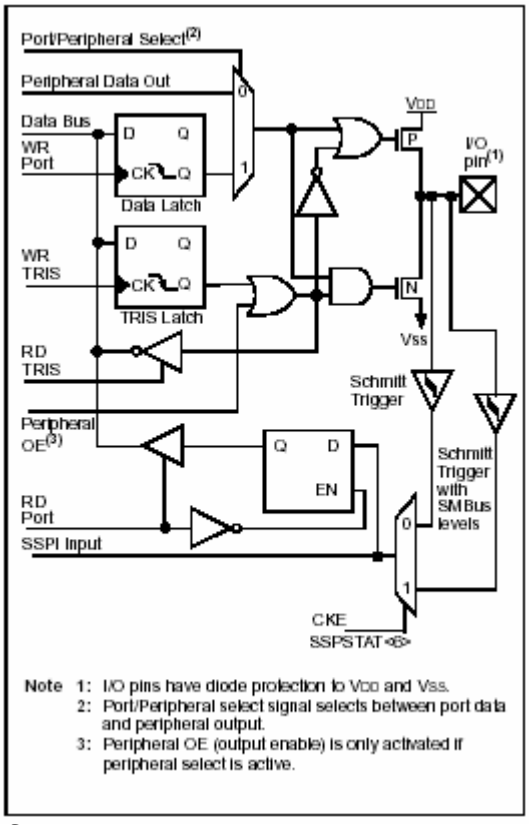
a) Sơ đồ khối RB3:RB0 b) Sơ đồ khối của RB7:RB4

Hình 1.7 Cấu trúc của Port B

c. ***Port C***:

Port C có cấu trúc như hình 1.8



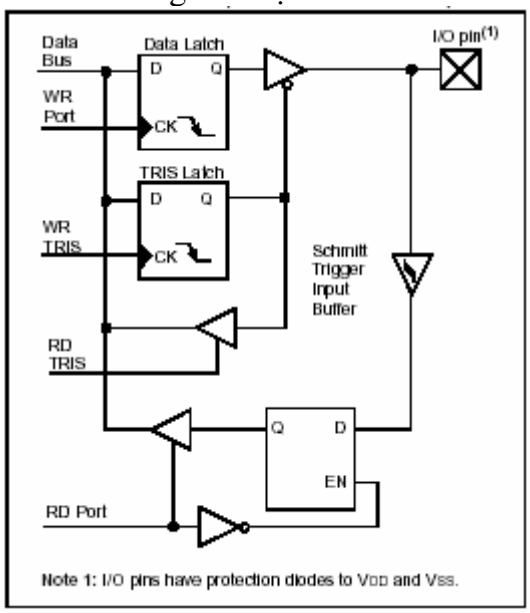


a) Sơ đồ khối các chân RC2:RC0, RC7:RC5 b) Sơ đồ khối các chân RC4:RC3

Hình 1.8 Sơ đồ cấu trúc Port C

d. ***Port D***:

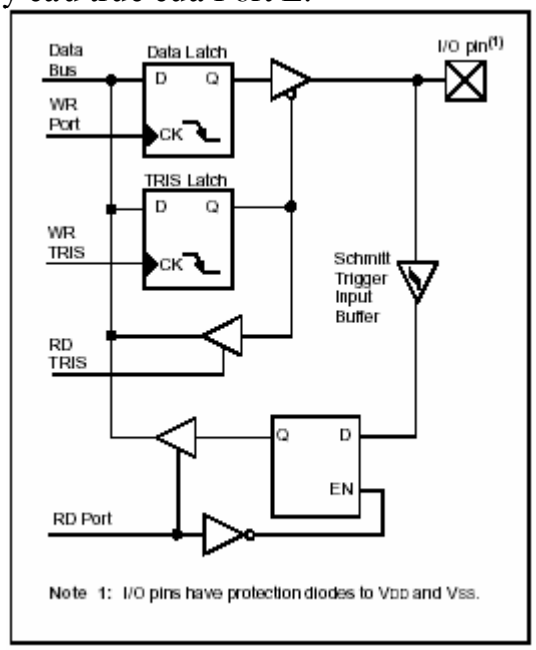
Port D gồm 8 chân với bộ đệm ngõ vào Schmitt trigger, mỗi chân của Port D có thể định cấu hình là ngõ vào hoặc ngõ ra riêng biệt. Port D cũng có thể được định cấu hình như là một cổng xuất nhập song song 8 bit nhờ bit PSPMODE. Trong chế độ này thì Port D tương thích TTL. Hình 1.9 giới thiệu cấu trúc của Port D.



Hình 1.9 Cấu trúc Port D ở chế độ xuất nhập.

e. ***Port E***:

Hình 1.10 trình bày cấu trúc của Port E.



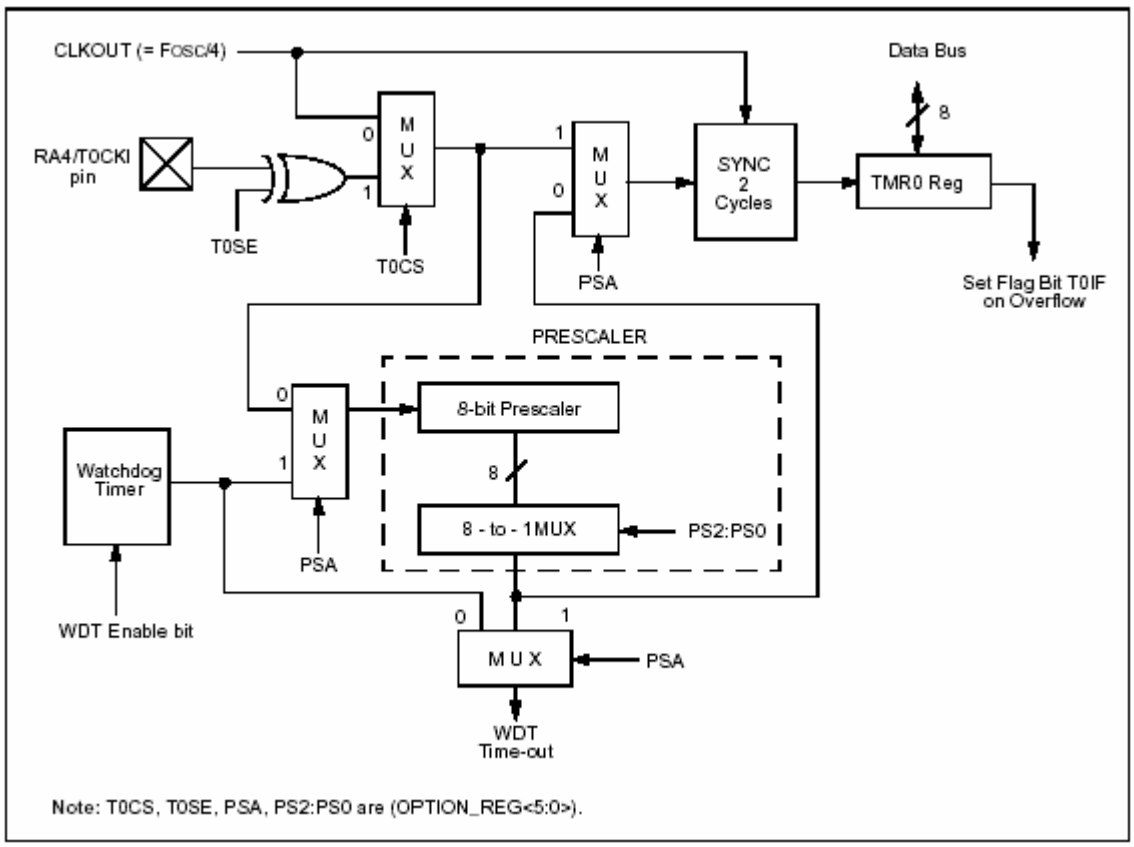
Hình 1.10 Cấu trúc Port E ở chế độ xuất nhập.

4. Timer 0:

Khối timer 0 (TMR0) có thể hoạt động như một bộ định thời hoặc như một bộ đếm với các chức năng như sau:

- Bộ định thời hoặc bộ đếm 8 bit.
- Có thể đọc hoặc ghi.
- Bộ định tỷ lệ 8 bit lập trình được bằng phần mềm.
- Chọn lựa được nguồn xung nhịp (Xung nhịp ngoài hoặc xung nhịp trong MCU).
- Ngắt tràn timer từ FFh tới 00h.
- Chọn lựa được cạnh tác động của xung nhịp bên ngoài.

Hình 1.11 trình bày cấu trúc của timer 0.



Hình 1.11 Cấu trúc của timer 0

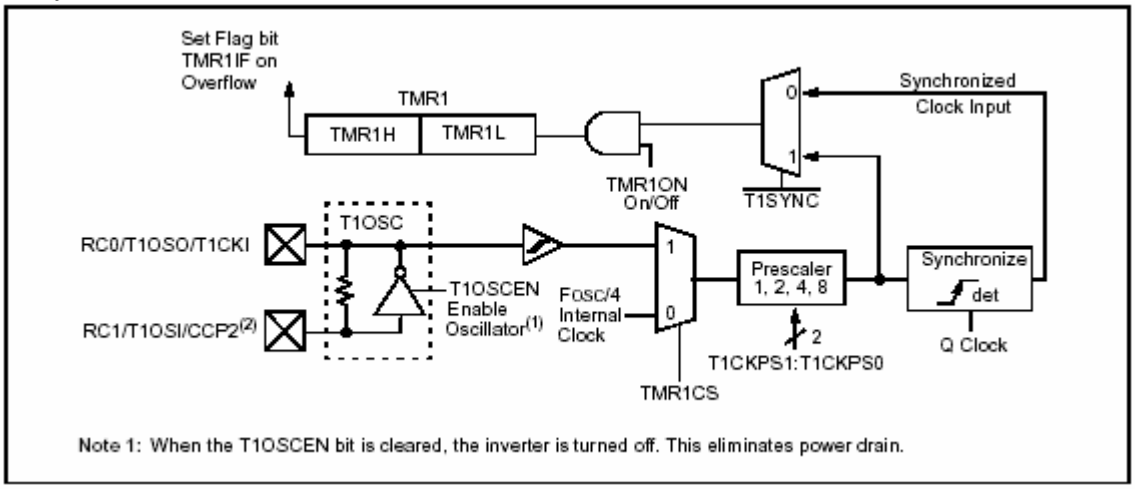
Chế độ định thời hay bộ đếm được thiết lập bit T0CS trong thanh ghi OPTION. Nếu bit T0CS bị xoá thì timer 0 làm việc như một bộ định thời 8 bit, nếu bit T0CS được đặt lên 1 thì timer 0 hoạt động như một bộ đếm 8 bit.

Ngắt TMR0 sẽ báo khi thanh ghi TMR0 tràn từ FFh xuống 00h. Khi thanh ghi TMR0 bị tràn thì bit T0IF trong thanh ghi INTCON sẽ lên 1. Bit T0IF phải được xoá bằng phần mềm trong chương trình con phục vụ ngắt ngay sau khi sự kiện ngắt xảy ra.

5. <u>Timer 1</u>:

Khối timer 1 (TMR1) là một bộ định thời hoặc bộ đếm 16 bit gồm 2 thanh ghi là TMR1H và TMR1L. Cặp thanh ghi này là loại thanh ghi đọc ghi được, giá trị của cặp thanh ghi này được tăng từ 0000h tới FFFFh và trở về 0000h. Nếu được cho phép, ngắt TMR1 sẽ báo khi giá trị đếm của TMR1 tràn từ FFFFh xuống 0000h. Khi ngắt xảy ra thì bit TMR1IF sẽ lên 1, bit này phải được xoá bằng phần mềm trong chương trình con phục vụ ngắt ngay sau khi ngắt xảy ra. TMR1 có thể làm việc ở 2 chế độ:

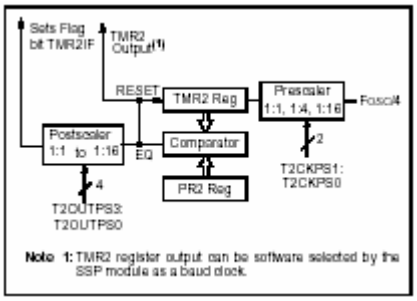
chế độ định thời và chế độ bộ đếm. Chế độ làm việc của TMR1 được quy định bởi bit TMR1CS trong thanh ghi T1CON. Nếu bit TMR1CS=1 TMR1 hoạt động như một bộ định thời 16 bit, nếu bit TMR1CS=0 TMR1 hoạt động như một bộ đếít6 bit. Hình 1.12 trình bày cấu trúc của TMR1.



Hình 1.12 Cấu trúc của TMR1.

6. Timer 2:

Timer 2 (TMR2) là một timer 8 bit có thể định được tỷ lệ, nó có thể được dùng nền thời gian PWM ở chế độ PWM của khối CCP. Thanh ghi TMR2 có thể được đọc ghi hoặc xoá bởi các thiết bị reset. Xung nhịp ngõ vào (Fosc/4) có tỷ lệ là 1:1, 1:4, 1:16 được chọn bởi bit T2CKPS1:T2CKPS0 của thanh ghi T2CON. Hình 1.13 trình bày cấu trúc của khối timer 2.



Hình 1.13 Cấu trúc của timer 2

7. Bộ Bắt Giữ Ngõ Vào, So Sánh Ngõ Ra Và Điều Rộng Xung (PWM):

PIC16F87x có 2 bộ bắt giữ ngõ vào, so sánh ngõ ra và PWM (CCP), Mỗi bộ gồm một thanh ghi 16 bit có thể hoạt động như:

- Thanh ghi bắt giữ ngõ vào 16 bit.
- Thanh ghi so sánh ngõ ra 16 bit.
- Thanh ghi chu kỳ tác động chủ/tớ PWM.

Cả hai khối CCP1 và CCP2 hoạt động tương tự như nhau ngoại trừ một số hoạt động đặc biệt. Bảng 8-1 và 8-2 cho thấy tương tác của các module CCP. Phần sau sẽ mô tả hoạt động của module CCP1. CCP2 hoạt động giống CCP1, ngoại trừ vài chỗ được ghi chú.

TABLE 8-1: CCP MODE - TIMER RESOURCES REQUIRED

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

TABLE 8-2: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 time-base
Capture	Compare	The compare should be configured for the special event trigger, which clears TMR1
Compare	Compare	The compare(s) should be configured for the special event trigger, which clears TMR1
PWM	PWM	The PWMs will have the same frequency and update rate (TMR2 interrupt)
PWM	Capture	None
PWM	Compare	None

CCP1 module:

Thanh ghi CCP1 (CCPR1) gồm 2 thanh ghi 8 bit: CCPR1L (byte thấp) và CCPR1H (byte cao). Thanh ghi CCP1CON điều khiển hoạt động của CCP1. Bộ kích sự kiện đặc biệt được phát sinh bởi việc so sánh bằng và sẽ khởi động lại Timer1.

CCP2 module:

Thanh ghi CCP2 (CCPR2) gồm 2 thanh ghi 8 bit: CCPR2L (byte thấp) và CCPR2H (byte cao). Thanh ghi CCP2CON điều khiển hoạt động của CCP2. Bộ kích sự kiện đặc biệt được phát sinh bởi việc so sánh bằng và sẽ khởi động lại Timer1 và bắt đầu một quá trình biến đổi A/D (nếu module A/D được cho phép).

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:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
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 is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Dưới đây trình bày hoạt động của khối CCP1.

a. ***Chế độ bắt giữ ngõ vào***:

Trong chế độ bắt giữ ngõ vào, CCPR1H:CCPR1L bắt một giá trị 16 bit của thanh ghi TMR1 khi một sự kiện xảy ra ở chân RC2/CCP1. Sự kiện được định nghĩa là:

- một cạnh xuống, hoặc
- một cạnh lên, hoặc
- mỗi cạnh lên thứ 4, hoặc
- mỗ cạnh lên thứ 16.

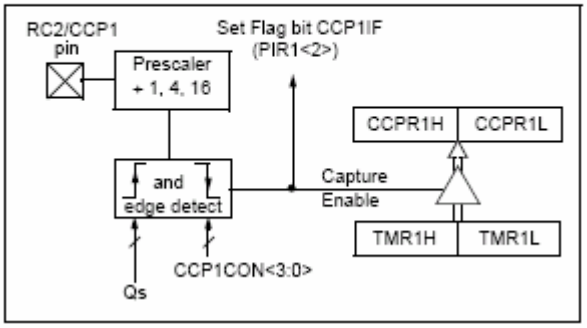
Loại sự kiện được chỉ định bởi các bit điều khiển CCP1M3:CCP1M0 (CCPxCON<3:0>). Khi bắt một sự kiện, bit cờ yêu cầu ngắt CCP1IF (PIR<2>) được bật lên 1. Cờ ngắt phải được xóa bằng phần mềm. Nếu xảy ra việc bắt sự kiện khác trước khi giá trị trong thanh ghi CCPR1 được đọc, giá trị bắt mới sẽ ghi đè lên giá trị cũ.

• ***Cấu hình chân CCP***:

Trong chế độ bắt sự kiện, chân RC2/CCP1 được cấu hình là một ngõ vào bằng cách đặt bit TRISC<2> lên 1.

Ghi chú: Nếu chân RC2/CCP1 được cấu hình là ngõ ra, thì việc ghi ra cổng có thể gây nên điều kiện bắt sự kiện.

FIGURE 8-1: CAPTURE MODE OPERATION BLOCK DIAGRAM



- ***Chọn chế độ hoạt động TIMER1:***

Timer1 phải đang chạy ở chế độ định thời, hay chế độ đếm đồng bộ cho module CCP để sử dụng tính năng bắt sự kiện. Trong chế độ đếm bất đồng bộ, hoạt động bắt sự kiện có thể không làm việc.

- ***Ngắt phần mềm***:

Khi chế độ bắt sự kiện thay đổi, có thể phát sinh một ngắt bắt sự kiện sai. Người dùng có thể giữ bit CCP1IE (PIE1<2>) bằng 0 để tránh ngắt sai và nên xóa bit cờ CCP1IF sau mỗi lần thay đổi chế độ hoạt động.

- ***Bộ định thang CCP (CCP Prescaler)***:

Có 4 cài đặt cho bộ định thang chỉ định bởi các bit CCP1M3:CCP1M0. Bất cứ khi nào tắt module CCP, hay module CCP không trong chế độ bắt sự kiện, thì bộ đếm định thang sẽ được xóa. RESET cũng sẽ xóa bộ đếm định thang.

Việc chuyển từ một bộ định thang bắt sự kiện này sang bộ khác có thể phát sinh một ngắt. Cũng vậy, bộ đếm định thang sẽ không bị xóa, do vậy mà việc bắt sự kiện đầu tiên có thể từ một bộ định thang khác 0. Ví dụ 8-1 cho thấy phương pháp đề xuất cho việc chuyển qua lại giữa các bộ định thang. Ví dụ này cũng xóa bộ đếm định thang và sẽ không phát sinh ngắt sai.

b. ***Chế độ so sánh***:

Trong chế độ so sánh, giá trị thanh ghi 16 bit CCPR1 luôn được so sánh với giá trị cặp thanh ghi TMR1. Khi bằng nhau, chân RC2/CCP1 sẽ được:

- lái lên mức cao,
- lái xuống mức thấp,
- duy trì không đổi.

Tác động trên một chân dựa trên giá trị của các bit điều khiển CCP1M3:CCP1M0 (CCP1CON<3:0>. Cùng lúc đó, bit cờ ngắt CCP1IF sẽ được bật lên 1.

- ***Cấu hình chân CCP***:

Người dùng phải cấu hình chân RC2/CCP1 là ngõ ra bằng cách xóa bit TRISC<2>.

Ghi chú: Việc xóa thanh ghi CCP1CON sẽ làm cho chốt ngõ ra so sánh mặc định xuống mức thấp. Đây không phải là chốt dữ liệu PORTC I/O.

- ***Chọn chế độ Timer1***:

Timer1 phải đang chạy ở chế độ định thời, hoặc chế độ đếm đồng bộ, nếu module CCP đang dùng tính năng so sánh. Trong chế độ đếm bất đồng bộ, hoạt động so sánh có thể không làm việc.

- ***Chế độ ngắt phần mềm***:

Khi chế độ ngắt phần mềm được chọn, chân CCP1 sẽ không bị ảnh hưởng. Đặt bit CCPIP lên 1 sẽ phát sinh ngắt CCP (nếu được cho phép).

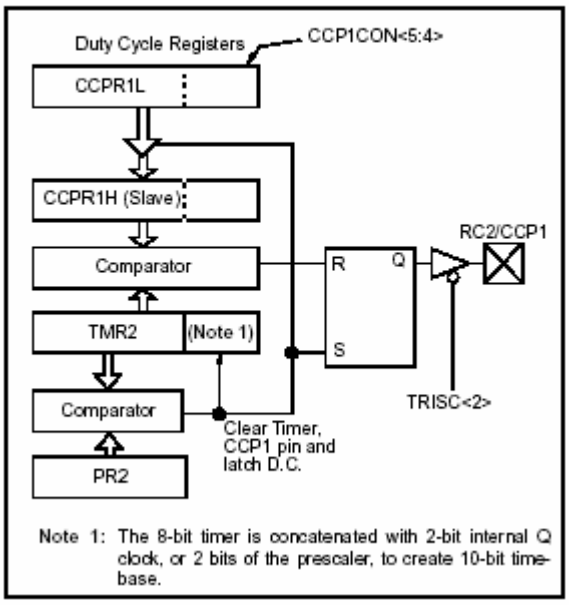
- ***Bộ kích sự kiện đặc biệt***:

Trong chế độ này, bộ kích phần cứng bên trong sẽ được phát sinh, dùng để kích hoạt một hoạt động.

c. ***Chế độ PWM***:

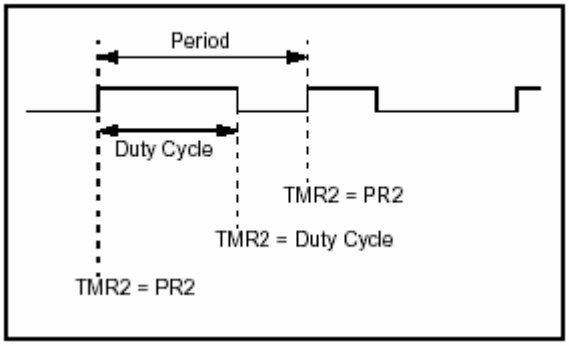
Trong chế độ PWM, chân CCPx được dùng để phát xung PWM với độ phân giải 10 bit. Do chân CCP1 là chân chốt dữ liệu đa năng của PORTC nên bit TRISC<2> phải được xóa để dùng như một ngõ ra. Hình sau trình bày hoạt động của bộ PWM.

FIGURE 8-3: SIMPLIFIED PWM BLOCK DIAGRAM



Tín hiệu PWM như sau:

FIGURE 8-4: PWM OUTPUT



Chu kỳ điều rộng xung (PWM) được xác định bởi việc ghi vào thanh ghi PR2 và được tính toán như biểu thức sau:

$$T_{PWM} = [(PR2)+1]*4*T_{OSC}*(TMR2)$$

Trong đó T_{OSC} là chu kỳ của tín hiệu dao động thạch anh, TMR2 là giá trị định thang của timer 2.

Khi TMR2 = PR2 thì:

- TMR2 được xóa.
- Chân CCP1 được set.

- Thời gian mức cao được chốt từ CCPR1L vào CCPR1H

Thời gian mức cao PWM được xác định bằng cách ghi vào thanh ghi CCPR1L và bit 4, 5 của thanh ghi CCP1CON để đạt độ phân giải 10 bit. Thanh ghi CCPR1L chứa 8 bit cao và CCP1CON<5:4> chứa 2 bit thấp. Thời gain mức cao được tính như sau:

$$T_{HIGH} = (CCPR1L:CCP1CON<5:4>)*T_{OSC}*TMR2$$

Để khởi tạo PWM ta thực hiện các bước như sau:

- Đặt chu kỳ PWM bằng cách ghi vào thanh ghi PR2.
- Đặt thời gian mức cao bằng cách ghi vào thanh ghi CCPR1L và CCP1CON<5:4>.
- Xoá bit TRISC<2>.
- Đặt giá trị cho TIMER2 và cho phép TIMER2 hoạt động bằng cách ghi vào thanh ghi T2CON.
- Định cấu hình cho khối CCP1 ở chế độ PWM.

TABLE 8-3: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFFh	0xFFh	0xFFh	0x3Fh	0x1Fh	0x17h
Maximum Resolution (bits)	10	10	10	8	7	5.5

TABLE 8-4: REGISTERS ASSOCIATED WITH CAPTURE, COMPARE, AND TIMER1

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	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by Capture and Timer1.
Note 1: The PSP is not implemented on the PIC16F873/876; always maintain these bits clear.

TABLE 8-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

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	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
92h	PR2	Timer2 Module's Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PWM and Timer2.
Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

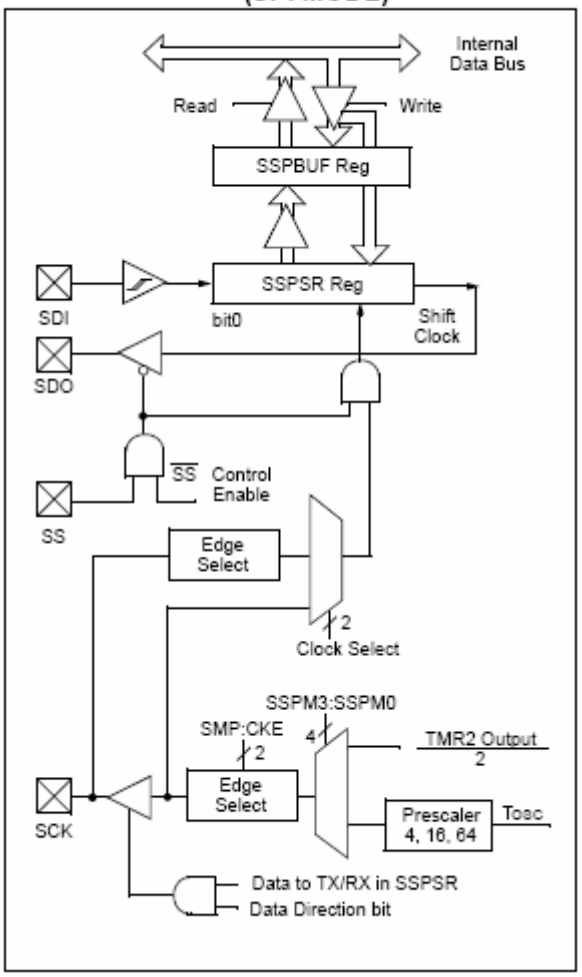
8. **Module Cổng Nối Tiếp Đồng Bộ Chủ (Mssp – Master Synchronous Serial Port)**:

Module MSSP là một giao tiếp nối tiếp, dùng để truyền thông với các ngoại vi hay vi điều khiển khác. Ngoại vi có thể là EEPROM nối tiếp, thanh ghi dịch, mạch lái hiển thị, bộ chuyển đổi A/D, ... Module MSSP có thể hoạt động trong 1 trong 2 chế độ:

- SPI (Serial Peripheral Interface)
- I2C (Inter-Integrated Circuit)

Hình 9-1 cho thấy sơ đồ hoạt động chế độ SPI, còn hình 9-5 và 9-9 là sơ đồ hoạt động các chế độ I2C.

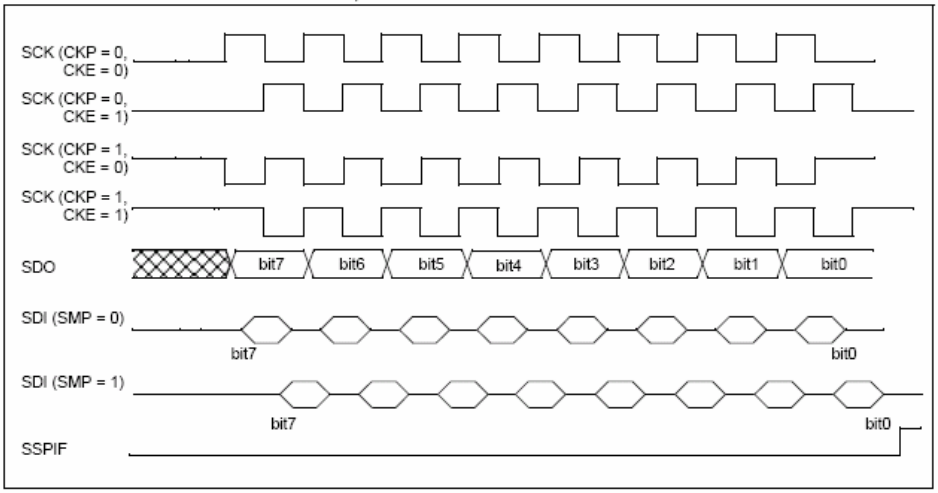
FIGURE 9-1: MSSP BLOCK DIAGRAM (SPI MODE)



a.***<u>Chế độ SPI</u>***:

Chế độ SPI cho phép 8 bit dữ liệu truyền và nhận đồng bộ cùng lúc. Cả 4 chế độ SPI đều được hỗ trợ. Chế độ này sử dụng 3 chân tín hiệu: SDO (serial data out), SDI (serial data in), SCK (serial clock). Ngoài ra, tín hiệu SS (slave select) được sử dụng khi hoạt động ở chế độ slave.

FIGURE 9-2: SPI MODE TIMING, MASTER MODE

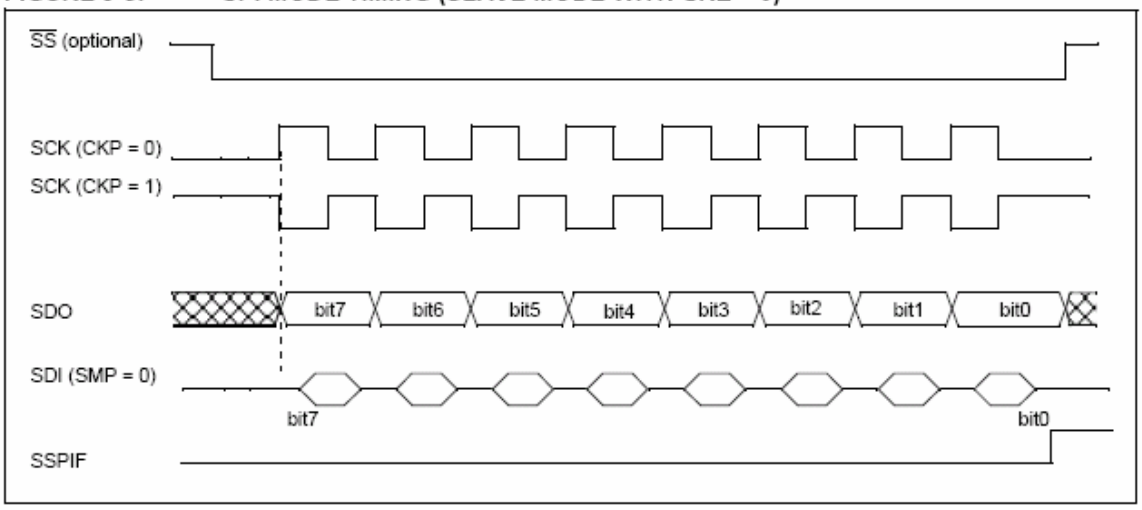


Khi khởi động SPI, một số tùy chọn cần phải được chỉ định qua các bit điều khiển SSPCON<5:0> và SSPSTAT<7:6>. Các bit này cho phép chỉ định:

- Chế độ master (SCK là ngõ ra xung nhịp)

- Chế độ slave (SCK là ngõ vào xung nhịp)
- Cực tính xung nhịp (trạng thái nghỉ của SCK)
- Pha lấy mẫu dữ liệu vào (giữa hay cuối thời gian dữ liệu ra)
- Cạnh xung nhịp (dữ liệu xuất vào lúc xuất hiện cạnh lên/ cạnh xuống của SCK)
- Tốc độ xung nhịp (chỉ trong chế độ master)
- Chế độ chọn slave (chỉ trong chế độ slave)

FIGURE 9-3: SPI MODE TIMING (SLAVE MODE WITH CKE = 0)



Hình 9-4 là sơ đồ khối module MSSP trong chế độ SPI.

Để cho phép cổng nối tiếp, bit SSPEN (SSPCON<5>) phải được đặt lên 1. Để reset hay cấu hình lại chế độ SPI, ta xóa bit SSPEN, và khởi tạo lại các thanh ghi SSPCON, rồi bật bit SSPEN lên 1; chân SDI, SDO, SCK và /SS sau đó sẽ trở thành những chân cổng nối tiếp. Để những chân này thực hiện chức năng port nối tiếp, vài chân phải được lập trình xác định các bit chiều dữ liệu thích hợp (trong thanh ghi TRIS); cụ thể là đối với:

- SDI: tự động được điều khiển bởi module SPI.
- SDO: phải xóa bit TRISC<5>
- SCK (chế độ master): phải xóa bit TRISC<3>
- SCK (chế độ slave): phải đặt bit TRISC<3> lên 1.
- /SS: phải đặt TRISA<5> lên 1 và thanh ghi ADCON1 (xem phần 11.0: module A/D) phải được đặt theo cách mà chân RA5 được cấu hình làm I/O số.

• ***Chế độ master***:

Master có thể khởi tạo việc truyền dữ liệu bất kỳ lúc nào vì nó điều khiển SCK. Master quyết định khi nào slave (Processor 2, hình 9-5) phát dữ liệu theo giao thức trong phần mềm.

Trong chế độ master, dữ liệu được thu/phát ngay sau khi ghi vào thanh ghi SSPBUF. Nếu như module SPI chỉ muốn nhận, thì ngõ ra SDO có thể không cho phép (lập trình như là ngõ vào). Thanh ghi SSPSR sẽ tiếp tục dịch vào tín hiệu có ở chân SDI với tốc độ xung nhịp đã được lập trình. Khi mỗi byte được nhận, nó sẽ được nạp vào thanh ghi SSPBUF như là một byte nhận bình thường (các ngắt và các bit trạng thái được đặt thích hợp). Điều này hữu dụng trong các ứng dụng bộ thu làm "giám sát hoạt động đường dây".

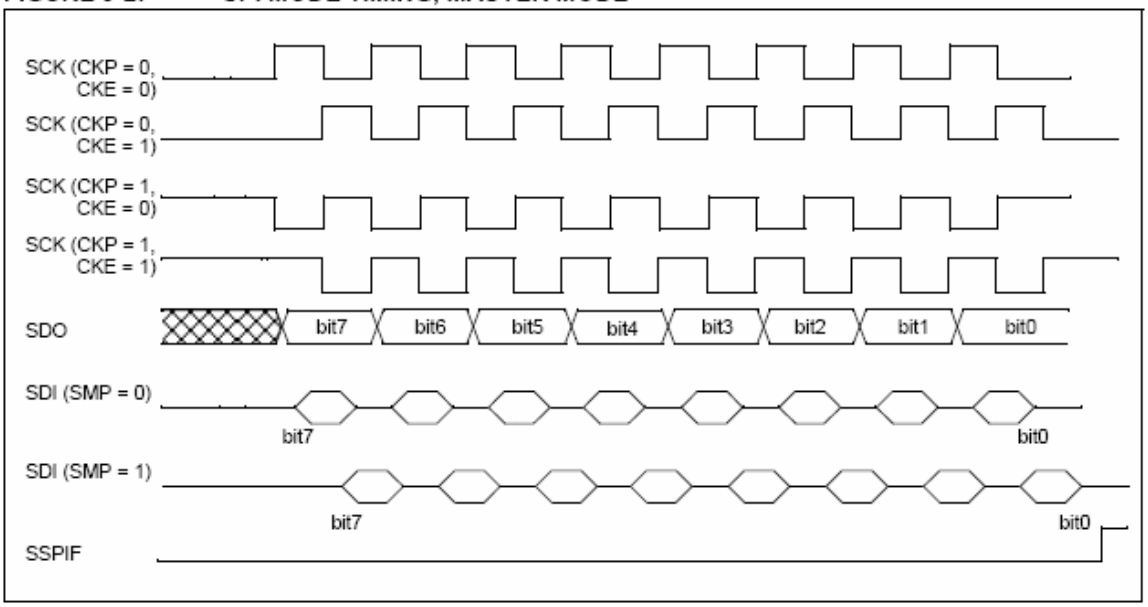
Cực tính xung nhịp được chọn bằng cách lập trình bit CKP (SSPCON<4>) có giá trị thích hợp. Kết quả ta sẽ được dạng sóng cho truyền thông SPI như hình 9-6, 9-8, 9-9, trong đó MSB sẽ được truyền trước tiên. Trong chế độ master, tốc độ bit cho SPI có thể lập trình được theo một trong những công thức sau:

- $F_{OSC}/4$ (hay T_{CY})
- $F_{OSC}/16$ (hay $4T_{CY}$)
- $F_{OSC}/64$ (hay $16T_{CY}$)
- Timer2 output/2

Tốc độ bit tối đa có thể đạt được là 5MHz (tại xung nhịp 20MHz)

Hình 9-6 là các dạng sóng chế độ master. Khi CKE = 1, dữ liệu SDO hợp lệ trước khi xuất hiện cạnh xung ở SCK. Sự thay đổi mẫu ngõ vào được thể hiện trên hình dựa trên trạng thái của bit SMP. Từ hình ta cũng thấy được thời gian khi mà dữ liệu nhận nạp vào SSPBUF.

FIGURE 9-2: SPI MODE TIMING, MASTER MODE



• ***<u>Chế độ slave</u>***:

Trong chế độ này, dữ liệu được truyền/nhận khi xung ngoài xuất hiện trên SCK. Khi bit sau cùng được chốt, cờ ngắt SSPIF (PIR<3>) bật lên 1.

Khi đang trong chế độ slave, xung nhịp ngoài được đưa vào qua chân SCK. Xung ngoài phải thỏa những điều kiện về điện ghi trong tài liệu kỹ thuật.

Trong chế độ SLEEP, slave có thể phát/thu dữ liệu. Khi một byte được nhận, thiết bị sẽ "thức dậy", thoát khỏi chế độ SLEEP.

Ghi chú 1: Khi module SPI ở chế độ slave với chân /SS tích cực (SSPCON<3:0> = 0100), module SPI sẽ khởi động lại nếu chân /SS nối với V_{DD}.

Ghi chú 2: Nếu SPI được dùng trong chế độ slave với CKE = 1, thì điều khiển chân /SS phải được cho phép.

FIGURE 9-3: SPI MODE TIMING (SLAVE MODE WITH CKE = 0)

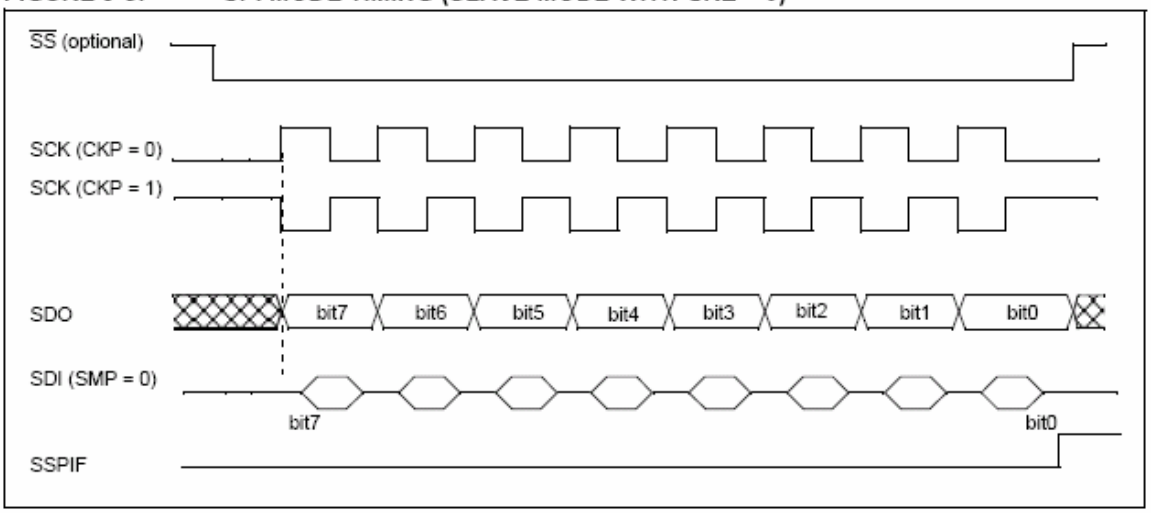


FIGURE 9-4: SPI MODE TIMING (SLAVE MODE WITH CKE = 1)

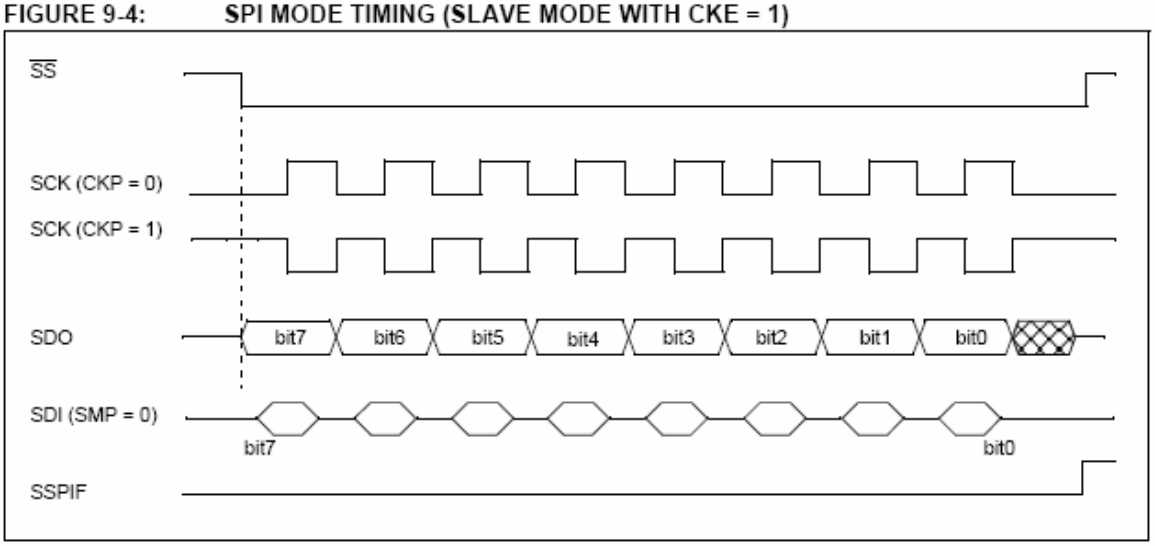


TABLE 9-1: REGISTERS ASSOCIATED WITH SPI OPERATION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on: MCLR, WDT
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the SSP in SPI mode.
Note 1: These bits are reserved on PCI16F873/876 devices; always maintain these bits clear.

b. ***Hoạt động I2C của MSSP***:

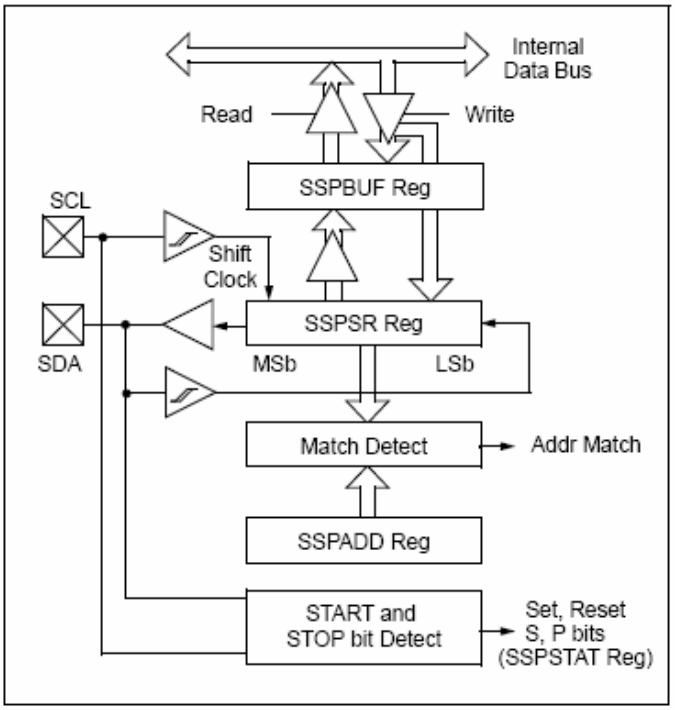
Module MSSP trong chế độ I2C thực hiện đầy đủ tất cả các chức năng master và slave và cung cấp các ngắt ở các bit START và STOP trong phần cứng để xác định bus rảnh (chứa năng multi-master). Module MSSP thực hiện các tính năng của chế độ chuẩn như là định địa chỉ 7 bit và 10 bit.

Hai chân dùng cho việc truyền dữ liệu nối tiếp là SCL (xung nhịp) và SDA (dữ liệu). Các chân này sẽ tự động được cấu hình khi cho phép chế độ I2C. Các chức năng module SSP được cho phép bằng cách đặt bit SSPEN (SSPCON<5>).

Module MSSP có 6 thanh ghi dành cho hoạt động I2C: SSPCON, SSPCON2, SSPSTAT, SSPBUF, SSPBUF, SSPSR (không truy xuất trực tiếp được), SSPADD.
Thanh ghi SSPCON cho phép điều khiển hoạt động I2C. Bốn bit chọn chế độ (SSPCON<3:0>) cho phép chọn:
- Chế độ I2C slave địa chỉ 7 bit
- Chế độ I2C slave địa chỉ 10 bit
- Chế độ I2C master, clock = OSC/4 (SSPAD +1)
- Các chế độ I2C firmware (phần dẻo) (tương thích với các sản phẩm cấp trung bình).

Trước khi chọn bất kỳ một chế độ I2C nào, các chân SCL và SDA phải được lập trình là ngõ vào bằng cách đặt các bit TRIS thích hợp. Việc chọn một chế độ I2C bằng cách đặt bit SSPEN sẽ cho phép chân SCL và SDA tương ứng làm đường truyền xung nhịp và dữ liệu trong chế độ I2C. Các điện trở kéo lên cần phải được nối bên ngoài vào các chân SCL và SDA.
Bit CKE (SSPSTAT<6:7>) đặt mức các chân SDA và SCL trong chế độ master hoặc slave. Khi CKE = 1, các mức phải thích ứng với đặc tính kỹ thuật của SMBus. Khi CKE = 0, các mức phải thích ứng với đặc tính kỹ thuật của I2C.
Thanh ghi SSPSTAT cho biết trạng thái của việc truyền dữ liệu. Thông tin này bao gồm phát hiện bit START (S) hay STOP (P), cho biết byte nhận được là dữ liệu hay địa chỉ, byte tiếp theo có phải là địa chỉ 10 bit hoàn chỉnh không, và việc truyền dữ liệu là đọc hay ghi.

FIGURE 9-5: I^2C SLAVE MODE BLOCK DIAGRAM



SSPBUF là thanh ghi mà dữ liệu truyền sẽ được ghi vào hay đọc từ đó ra. Thanh ghi SSPSR dịch dữ liệu vào/ra thiết bị. Trong hoạt động nhận, SSPBUF và SSPSR tạo nên một bộ đệm nhận kép, cho phép bắt đầu tiếp nhận byte tiếp theo trước khi đọc bit cuối

của dữ liệu nhận. Khi byte hoàn chỉnh nhận về, nó được truyền, nó được chuyển đến thanh ghi SSPBUF và bit cờ SSPIF bật lên. Nếu một byte hoàn chỉnh khác được nhận trước khi SSPBUF được đọc, thì bộ nhận sẽ bị tràn, bit SSPOV (SSPCON<6>) sẽ bật lên 1, và byte trong SSPSR sẽ bị mất.

Thanh ghi SSPADD giữ địa chỉ slave. Trong chế độ 10 bit, ta cần phải ghi vào byte cao địa chỉ (1111 0 A9 A8 0); tiếp theo cần phải nạp byte thấp (A7:A0).

• ***Chế độ slave***:

Trong chế độ slave, các chân SCL và SDA phải được cấu hình là ngõ vào. Module MSSP sẽ gối lên trạng thái ngõ vào bằng dữ liệu ra khi yêu cầu (bộ phát slave).

Khi đúng địa chỉ, hay dữ liệu truyền sau khi nhận được đúng địa chỉ, phần cứng sẽ tự động phát xung Acknowledge (/ACK), rồi sau đó nạp vào thanh ghi SSPBUF giá trị vừa nhận trong thanh ghi SSPSR.

Có những điều kiện làm cho module MSSP không phát xung /ACK:

a) Bit báo bộ đệm tràn BF (SSPSTAT<0>) bật lên 1 trước khi truyền.

b) Bit tràn SSPOV (SSPCON<6>) bật lên 1 trước khi truyền.

Nếu bit BF bật lên 1, thì giá trị thanh ghi SSPSR sẽ không được nạp vào SSPBUF, nhưng bit SSPIF và SSPOV được đặt lên 1. Bảng 9-2 cho thấy những gì xảy ra khi nhận byte dữ liệu. Những ô màu xám cho thấy điều kiện mà phần mềm của người dùng xóa điều kiện tràn không đúng. Bit cờ BF được xóa khi đọc thanh ghi SSPBUF, còn SSPOV được xóa bằng phần mềm.

TABLE 9-2: DATA TRANSFER RECEIVED BYTE ACTIONS

Status Bits as Data Transfer is Received		SSPSR → SSPBUF	Generate $\overline{ACK}$ Pulse	Set bit SSPIF (SSP Interrupt occurs if enabled)
BF	SSPOV			
0	0	Yes	Yes	Yes
1	0	No	No	Yes
1	1	No	No	Yes
0	1	Yes	No	Yes

Note: Shaded cells show the conditions where the user software did not properly clear the overflow condition.

- Định địa chỉ:

Một khi module MSSP được cho phép, nó chờ điều kiện START xảy ra. Theo sau điều kiện START, 8 bit được dịch vào thanh ghi SSPSR. Tất cả các biến đến điều được lấy tại cạnh lên của SCL. Giá trị của thanh ghi SSPSR<7:1> được so sánh với giá trị trong thanh ghi SSPADD. Địa chỉ được so sánh tại cạnh xuống của xung nhịp thứ 8. Nếu đúng địa chỉ, các bit BF và SSPOV = 0, thì sẽ xảy ra các sự kiện sau:

a) Giá trị thanh ghi SSPSR nạp vào SSPBUF tại cạnh xuống của xung SCL thứ 8.

b) Bit báo báo bộ đệm đầy, BF, bật lên 1 tại xuống của xung SCL thứ 8.

c) Một xung /ACK được phát ra.

d) Cờ ngắt SSP, SSPIF (PIR1<3>), bật lên tại cạnh xuống của xung SCL thứ 9.

Trong chế độ địa chỉ 10 bit, slave cần phải nhận 2 byte địa chỉ. 5 bit cao của byte địa chỉ

9. **Module Truyền/Nhận Nối Tiếp Đồng Bộ/Bất Đồng Bộ Định Được Địa Chỉ (USART)**:

Module USART là một trong 2 module vào ra nối tiếp. USART còn được xem như là một cổng giao tiếp dữ liệu nối tiếp hoặc SCI. USART cũng có thể được định dạng như

một hệ thống bất đồng bộ kép dùng để truyền thông với các thiết bị ngoại vi như cổng CRT hoặc là máy tính cá nhân hoặc có thể dịnh dạng như một hệ thống đồng bộ đơn để giao tiếp dữ liệu với các thiết bị như các mạch ADC, DAC, ROM nối tiếp ... Các dạng cuấu hình của module USART:

- Bất đồng bộ kép (2 chiều).
- Đồng bộ chủ đơn (Một chiều).
- Đồng bộ tớ đơn (Một chiều).

Bit SPEN (RCSTA<7>) và bit TRISC<7:6> phải được set để định cấu hình cho chân RC6/TX/CK và chân RC7/RX/DT như một USART. Module USART cũng có khả năng truyền thông đa xử lý dùng 9 bit để phân biệt địa chỉ. Hoạt động của module USART được quản lý bởi 2 thanh ghi:

REGISTER 10-1: 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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 10-2: 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)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

a. ***Phát tốc độ baud USART (BRG)***:

BRG hỗ trợ cả chế độ USART đồng bộ và chế độ USART bất đồng bộ. PIC sẽ dành 8 bit để phát tốc độ baud. Thanh ghi SPBRG điều khiển chu kỳ của timer 8 bit hoạt động tự do. Trong chế độ bất đồng bộ, bit BRGH được sử dụng để điều khiển tốc độ baud. Công thức tính tốc độ baud như bảng sau:

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)

Bảng sau trình bày các tốc độ baub tương ứng với tần số xung nhịp.

TABLE 10-3: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

TABLE 10-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

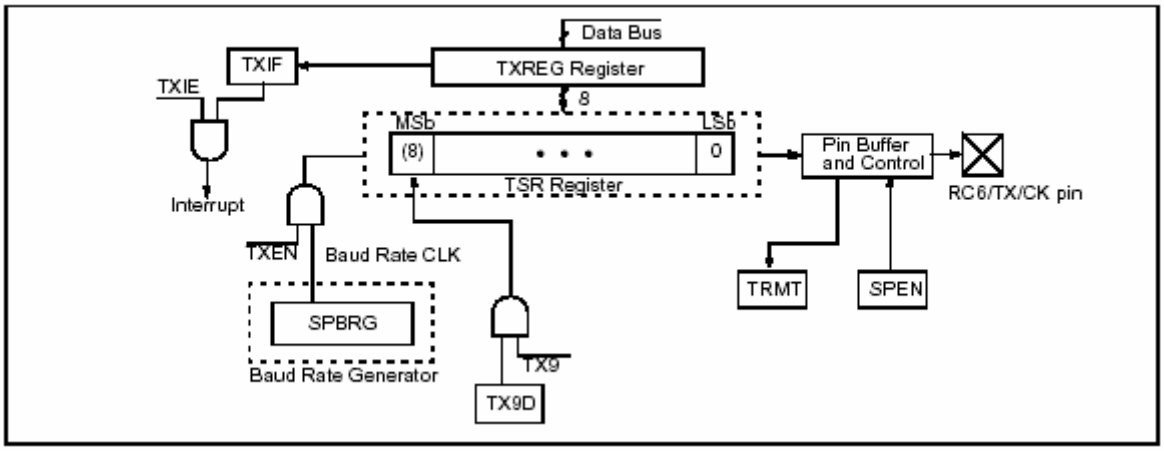
BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

b. ***<u>Chế độ USART bất đồng bộ</u>***:

Trong chế độ bất đồng bộ, USART sử dụng định dạng dữ liệu theo kiểu 1 start bit, 8 hoặc 9 bit data và 1 bit stop. Sơ đồ khối của bộ truyền dữ liệu như sau:

FIGURE 10-1: USART TRANSMIT BLOCK DIAGRAM



Các bước để cài đặt truyền bất đồng bộ:

- Đặt giá trị ban đầu cho thanh ghi SPBRG tương ứng với tốc độ baud.
- Cho phép cổng nối tiếp bất đồng bộ bằng cách xoá bit SYNC và đặt bit SPEN.
- Nếu yêu cầu dùng ngắt thì đặt bit TXIE để cho phép ngắt.
- Nếu tuyền 9 bit thì đặt bit truyền TX9.
- Cho phép truyền bằng cách đặt bit TXEN và TXIF.
- Nếu chọn chế độ truyền 9 bit thì bit thứ 9 sẽ được đưa vào bit TX9D.
- Load data vào thanh ghi TXREG.
- Nếu dùng ngắt thì phải set bit GIE và PEIE trong thanh ghi INTCON.

FIGURE 10-2: ASYNCHRONOUS MASTER TRANSMISSION

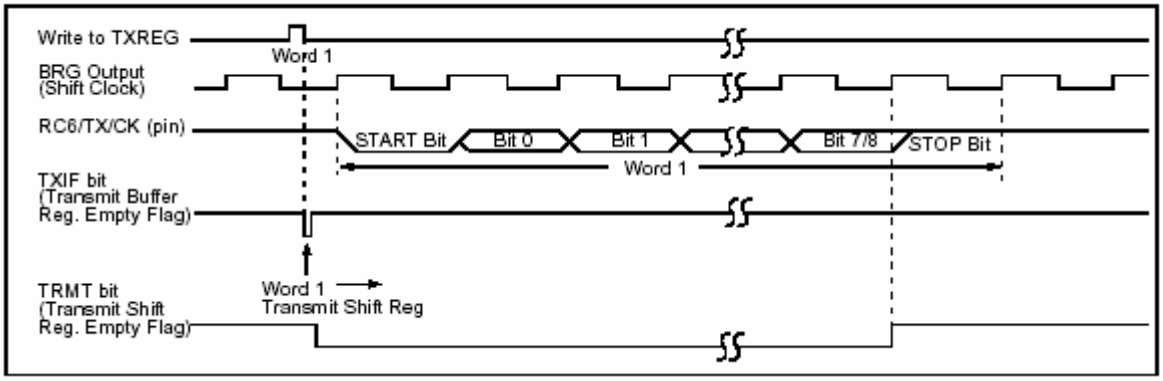


FIGURE 10-3: ASYNCHRONOUS MASTER TRANSMISSION (BACK TO BACK)

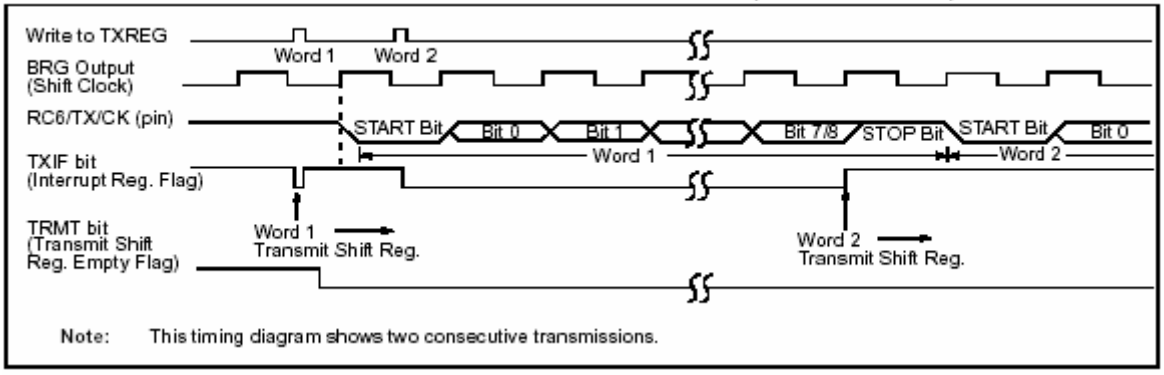


TABLE 10-5: REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

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

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.
Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

Quá trình nhận dữ liệu được thực hiện thông qua chân RC7. Khối nhận dữ liệu USART như hình:

FIGURE 10-4: USART RECEIVE BLOCK DIAGRAM

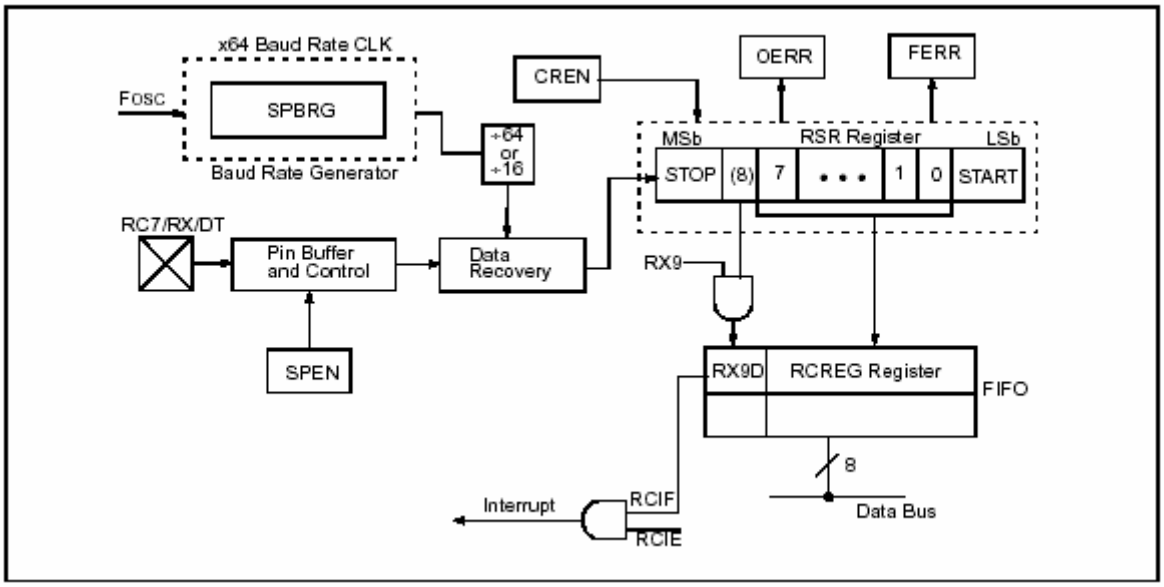
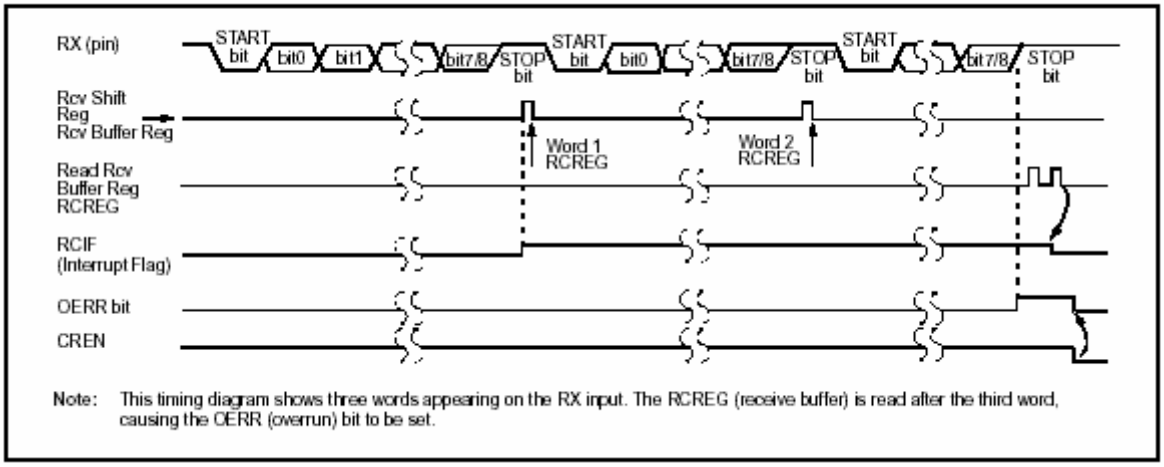


FIGURE 10-5: ASYNCHRONOUS RECEPTION



Note: This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.

Các bước để thực hiện việc nhận dữ liệu USART.

- Đặt giá trị ban đầu cho thanh ghi SPBRG phù hợp vớt tốc độ baud. Nếu yêu cầu tốc độ baud cao, set bit BRGH.
- Cho phép cổng nối tiếp bất đồng bộ bằng các xoá bit SYNC và set bit SPEN.
- Nếu yêu cầu ngắt thì set bit RCIE.
- Nếu yêu cầu nhận 9 bit thì set bit RX9.

- Cho phép nhận dữ liệu bằng cách set bit CREN.
- Bit cờ RCIF sẽ bật lên 1 khi nhận xong dữ liệu và một ngắt sẽ báo nếu bit RCIE được set.
- Đọc thanh ghi RCSTA để lấy bit thứ 9 và xác định lỗi nếu có trong quá trình nhận.
- Lấy 8 bit dữ liệu bằng cách đọc thanh ghi RCREG.
- Nếu có lỗi xảy ra, xoá lỗi bằng cách xoá bit CREN.
- Nếu dùng ngắt, đặt bit GIE và PEIE trong thanh ghi INTCON.

Nhóm các thanh ghi quản lý quá trình nhận.

TABLE 10-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

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

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

Khi cài đặt chế độ nhận sử dụng 9 bit để xác định địa chỉ trong chế độ nhiều MCU, ta thực hiện các bước sau:

- Đặt giá trị ban đầu cho thanh ghi SPBRG phù hợp với tốc độ baud. Nếu chọn tốc độ cao thì đặt bit BRGH.
- Cho phép cổng nối tiếp bằng cách xoá bit SYNC và đặt bit SPEN.
- Nếu chọn ngắt thì đặt bit RCIE.
- Đặt bit RX9 để cho phép nhận 9 bit.
- Đặt ADDEN để cho phép xác định địa chỉ.
- Cho phép nhận bằng cách đặt bit CREN.
- Bit cờ RCIF sẽ bật lên khi nhận xong và ngắt sẽ báo nếu bit RCIE được set.
- Đọc thanh ghi RCSTA để lấy bit thứ 9 và xác định lỗi trong quá trình nhận nếu có.
- Lấy 8 bit data bằng cách đọc thanh ghi RCREG để xác định địa chỉ của MCU.
- Nếu trong quá trình nhận có lỗi, xoá lỗi bằng cách xoá bit CREN.
- Nếu đúng địa chỉ của MCU, xoá bit ADDEN để cho phép byte dữ liệu và byte địa chỉ được đọc vào vủng đệm dữ liệu nhận và ngắt CPU.

Sơ đồ tiến trình nhận dữ liệu nhiều MCU và giản đồ tín hiệu.

FIGURE 10-7: ASYNCHRONOUS RECEPTION WITH ADDRESS DETECT

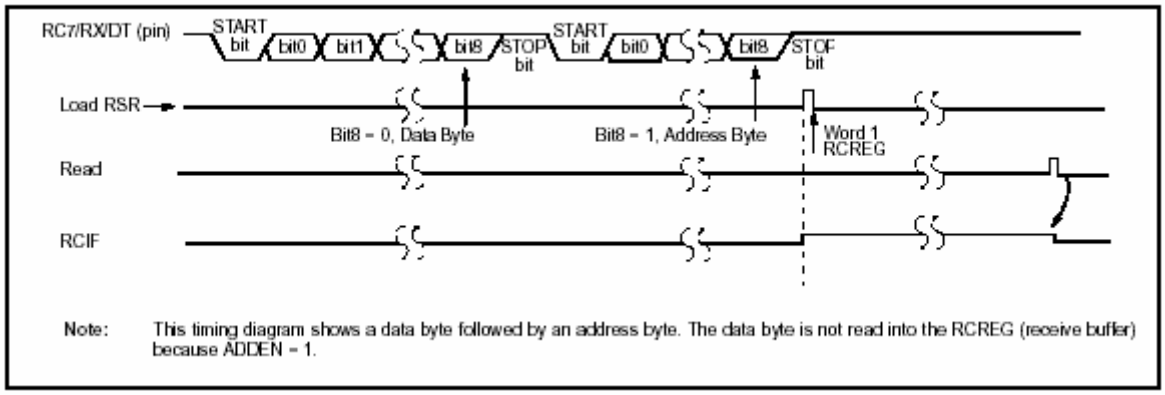


FIGURE 10-6: USART RECEIVE BLOCK DIAGRAM

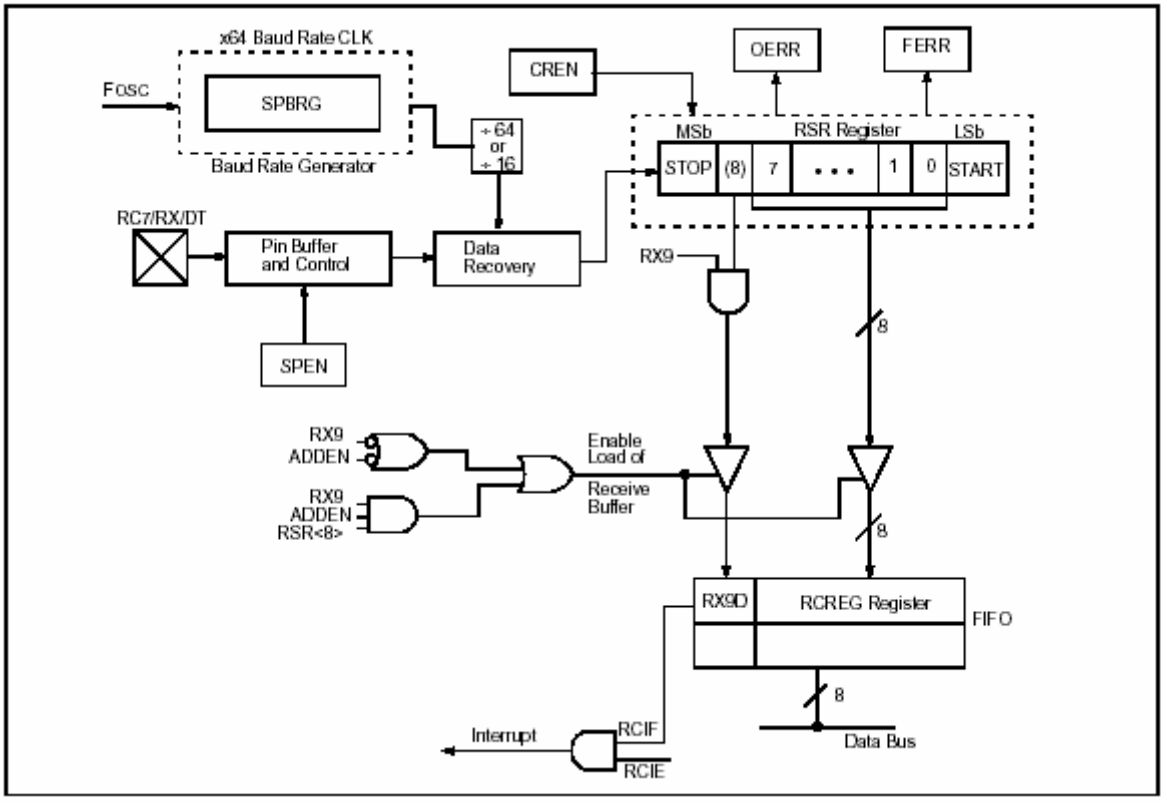


FIGURE 10-8: ASYNCHRONOUS RECEPTION WITH ADDRESS BYTE FIRST

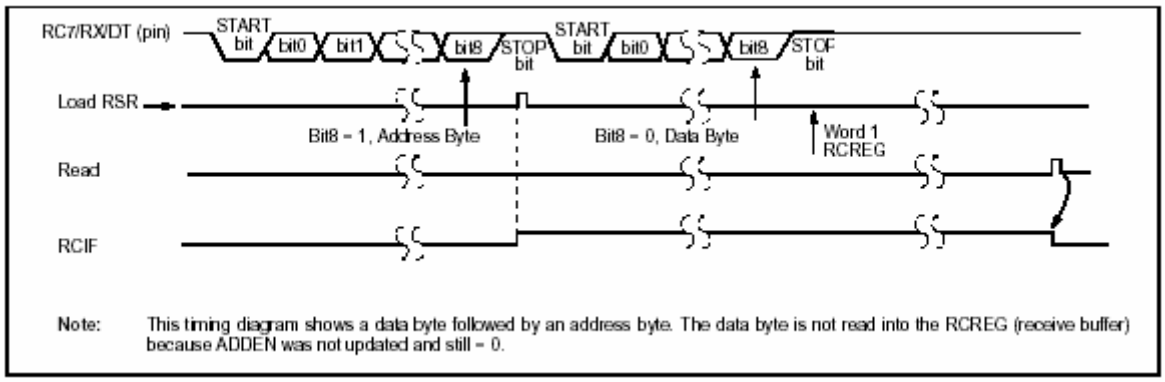


TABLE 10-7: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

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	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
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

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

c. ***USART ở chế độ đồng bộ chủ***:

Trong chế độ đồng bộ chủ, dữ liệu truyền nhận không diễn ra đồng thời. Chế độ đồng bộ được chọn thông qua bit SYNC (TXSTA<4>). Bit SPEN (RCSTA<7>) được đặt lên để cấu hình cho chân RC6/TX/CK và chân RC7/RX/DT thành đường clock và data. Chế độ đồng bộ chủ được cho phép thông qua việc cài đặt bit CSRC (TXSTA<7>).

Sơ đồ khối của module truyền như hình 10 -6. Để thực hiện truyền USART ở chế độ chủ, ta thực hiện các bước như sau:

- Đặt giá trị ban đầu cho thanh ghi SPBRG cho phù hợp với tốc độ baud.
- Cho phép cổng nối tiếp bằng cách đặt bit SYNC, SPEN và CSRC.
- Nếu sử dụng ngắt thì đặt bit TXIE.
- Nếu chọn truyền 9 bit thì đặt bit TX9.
- Cho phép truyền bằng cách đặt bit TXEN.
- Nếu chọn chế độ truyền 9 bit thì bit thức 9 sẽ được load vào bit TX9D.
- Bắt đầu truyền bằng cách load dữ liệu vào thanh ghi TXREG.
- Nếu sử dụng ngắt thì đặt bit GIE và bit PEIE trong thanh ghi INTCON.

Nhóm các thanh ghi trong chế độ truyền đồng bộ chủ.

TABLE 10-8: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER TRANSMISSION

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

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for synchronous master transmission.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

FIGURE 10-9: SYNCHRONOUS TRANSMISSION

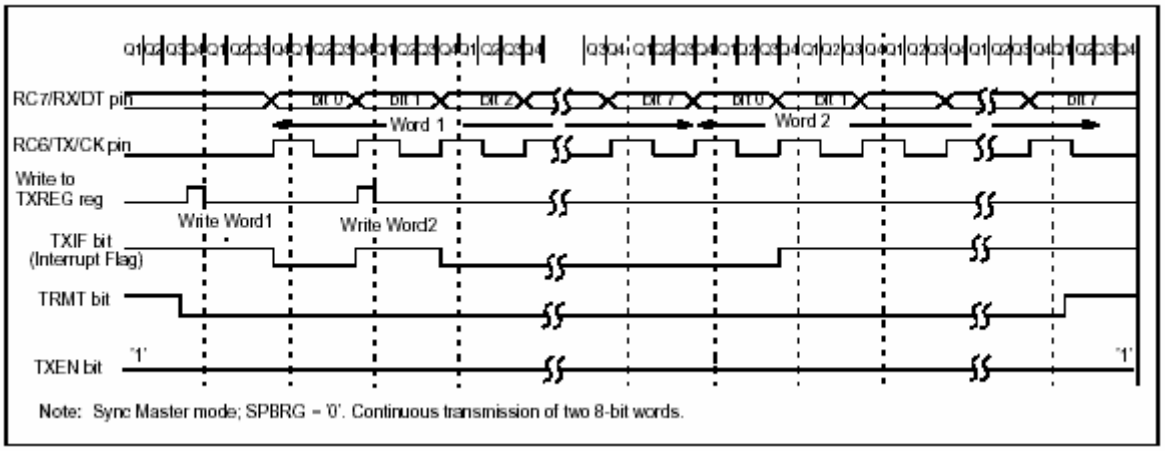
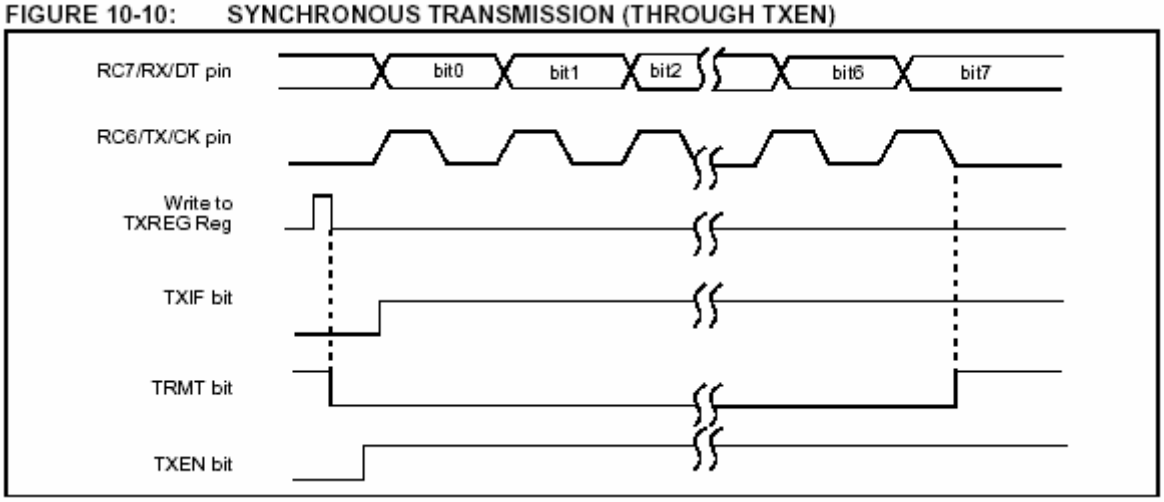


FIGURE 10-10: SYNCHRONOUS TRANSMISSION (THROUGH TXEN)



Nhận dữ liệu đồng bộ ở chế độ chủ được cho phép thông qua bit SREN (RCSTA<5>) hoặc bit CREN (RCSTA<4>). Dữ liệu được lấy mẫu trên chân RC7 ngay tại cạch xuống của clock. Nếu bit SREN được đặt thì chỉ có một từ đơn được nhận. Nếu bit CREN được set thì dữ liệu được nhận cho tới khi bit này bị xoá. Nếu cả 2 bit SREN và CREN cùng được set thì bit CREN sẽ được ưu tiên. Các bước nhận dữ liệu trong mode master:

- Đặt giá trị ban đầu cho thanh ghi SPBRG phù hợp với tốc độ baud.
- Cho phép cổng nối tiếp bằng cách đặt bit SYNC, SPEN và CSRC.
- Xoá bit SREN và CREN.
- Nếu dùng ngắt thì đặt bit RCIE.
- Nếu chọn chế độ nhận dữ liệu 9 bit thì set bit RX9.
- Nếu chỉ nhận 1 từ, set bit SREN, nếu nhận nhiều từ thì set bit CREN.
- Bit cờ ngắt RCIF sẽ bật khi nhận xong và sẽ báo ngắt nếu bit RCIE đã được set.
- Đọc thanh ghi RCSTA để lấy bit thứ 9 để xác định lỗi trong quá trình nhận nếu có.
- Nếu có lỗi trong quá trình nhận, xoá lỗi bằng cách xoá bit CREN.
- Set bit GIE và PEIE trong thanh ghi INTCON để cho phép ngắt.

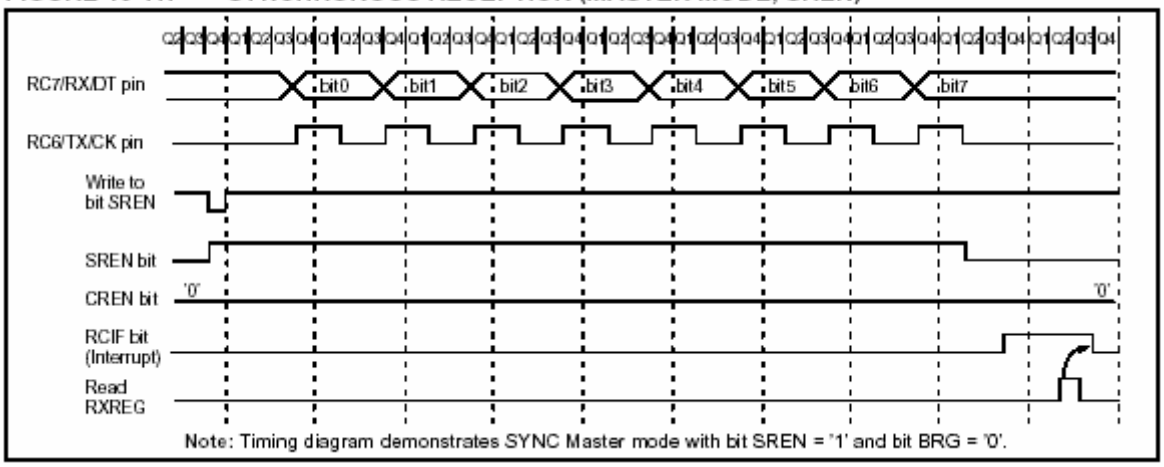
Nhóm các thanh ghi dùng trong chế độ nhận dữ liệu nối tiếp đồng bộ.

TABLE 10-9: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER RECEPTION

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

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for synchronous master reception.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

FIGURE 10-11: SYNCHRONOUS RECEPTION (MASTER MODE, SREN)



d. ***<u>USART trong chế độ đồng bộ tớ</u>***:

Chế động đồng bộ tớ khác so với chế độ đồng bộ chủ ở chỗ xung nhịp được đưa vào từ ngoài thông qua chân RC6 (thay vì được cung cấp từ bên trong ở chế độ chủ). Vì vậy mà MCU có thể truyền nhận dữ liệu trong chế độ "ngủ". Chế độ đồng bộ tớ được xác định bởi việc xoá bit CSRC (TXSTA<7>).

Hoạt động của USART ở mode master và mode slave là như nhau ngoại trừ ở chế độ "ngủ". Nếu có 2 word được ghi vào thanh ghi TXREG và sau đó lệnh SLEEP được thực hiện thì các vấn đề sau sẽ xảy ra:

- Word đầu tiên sẽ lập tức được chuyển vào thanh ghi TSR và truyền.
- Word thứ 2 vẫn ở trên thanh ghi TXREG.
- Bit cờ TXIF sẽ không được set.
- Khi word thứ nhất được shift khỏi thanh ghi TSR, thanh ghi TXREG sẽ chuyển wodr thứ 2 vào thanh ghi TSR và bit cớ TXIF bây giờ sẽ được set.
- Nếu bit cho phép TXIE được set, thì ngắt sẽ "đánh thức" MCU từ chế độ "ngủ" và nếu ngắt toàn cục được cho phép, thì chương trình sẽ rẽ nhánh đến vector ngắt (0004h).

Khi cài đặt quá trình truyền nhận đồng bộ ở chế độ tớ, ta thực hiện các bước sau:

- Cho phép cổng nối tiếp bằng cách đặt các bit SYNC, SPEN và xoá bit CSRC.
- Xoá bit CREN và SREN.
- Nếu cho phép ngắt thì set bit TXIE.
- Nếu chọn quá trình truyền 9 bit thì đặt bit TX9.

- Cho phép truyền bằng cách đặt bit TXEN.
- Nếu chọn truyền 9 bit thì bit thứ 9 sẽ được load vào bit TX9D.
- Bắt đầu quá trình truyền bằng cách load data vào thanh ghi TXREG.
- Đặt bit GIE và PEIE trong thanh ghi INTCON để dùng ngắt.

Nhóm các thanh ghi phục vụ quá trình truyền:

TABLE 10-10: REGISTERS ASSOCIATED WITH SYNCHRONOUS SLAVE TRANSMISSION

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	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
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

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for synchronous slave transmission.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

Quá trình nhận dữ liệu đồng bộ ở chế độ tớ và chế độ chủ là như nhau ngaọi trừ chế độ "ngủ". Bit SREN không cần quan tâm trong chế độ tớ. Nếu quá trình nhận được cho phép bằng cách set bit CREN trước lệnh SLEEP thì 1 word sẽ được nhận trong khi MCU ở chế đổ "ngủ". Khi nhận xong 1 word, thanh RSR sẽ chuyển data vào thanh ghi RCREG và set bit RCIE, ngắt sẽ báo và "đánh thức" MCU từ chế độ "ngủ". Nếu ngắt toàn cục được cho phép, chương trình sẽ rẽ nhánh tới vector ngắt (0004h).

Các bước để thực hiện nhận dữ liệu nối tiếp đồng bộ ở mode slave:

- Cho phép cổng nối tiếp bằng cách đặt các bit SYNC, SPEN và xoá bit CSRC.
- Nếu cho phép ngắt thì set bit RCIE.
- Nếu chọn quá trình truyền 9 bit thì đặt bit RX9.
- Cho phép truyền bằng cách đặt bit CREN.
- Bit cờ RCIF sẽ bật lên khi nhận xong và ngắt sẽ báo nếu bit RCIE đã được set.
- Đọc thanh ghi RCSTA và lấy bit thứ 9 để kiểm tra lỗi trong quá trình nhận.
- Lấy 8 bit dữ liệu bằng cách đọc thanh ghi RCREG.
- Nếu có lỗi trong quá trình nhận, xoá lỗi bằng cách xoá bit CREN.
- Set các bit GIE, PEIE của thanh ghi INTCON để dùng ngắt.

Nhóm các thanh ghi phục vụ nhận dữ liệu nối tiếp đồng bộ trong chế độ tớ:

TABLE 10-11: REGISTERS ASSOCIATED WITH SYNCHRONOUS SLAVE RECEPTION

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	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
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

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for synchronous slave reception.
Note 1: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices, always maintain these bits clear.

10. **Module chuyển đổi A/D**:

Module chuyển đổi A/D có 5 ngõ vào đối với chip 28 chân và 8 ngõ vào đối với những loại chip khác.

Ngõ vào tương tự nạp một tụ lấy mẫu và giữ. Ngõ ra của tụ này là ngõ vào của bộ biến đổi. Sau đó, bộ biến đổi phát ra kết quả số của tín hiệu tương tự trên qua việc xấp xỉ liên tiếp. Việc chuyển đổi A/D tín hiệu vào tương tự cho kết quả là một số 10 bit tương ứng. Module A/D có ngõ vào tham chiếu điện áp cao và thấp có thể chọn được bằng phần mềm một sự tổ hợp nào đó của VDD, VSS, RA2, hay RA3.

Bộ chuyển đổi A/D có một tính năng độc đáo là có thể hoạt động khi thiết bị đang trong chế độ SLEEP (ngủ). Trong trường hợp này, xung nhịp A/D phải được lấy từ bộ giao động RC bên trong.

Module A/D có 4 thanh ghi:

- ADRESH (A/D Result High Register): thanh ghi kết quả A/D cao.
- ADRESL (A/D Result Low Register): thanh ghi kết quả A/D thấp.
- ADCON0 (A/D Control Register0): thanh ghi điều khiển A/D 0.
- ADCON1 (A/D Control Register1): thanh ghi điều khiển A/D 1.

Thanh ghi ADCON0 (xem Register 11-1) điều khiển hoạt động của module A/D.

Thanh ghi ADCON1 (xem Register 11-2) định cấu hình chức năng của các chân port. Các chân port có thể cấu hình là những ngõ vào tương tự (RA3 cũng có thể là tham chiếu điện áp), hay là I/O số.

REGISTER 11-1: ADCON0 REGISTER (ADDRESS: 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented**: Read as '0'

bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 11-2: ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

Note 1: These channels are not available on PIC16F873/876 devices.
2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Các thanh ghi ADRESH:ADRESL chứa 10 bit kết quả biến đổi A/D. Khi biến đổi xong, kết quả được nạp cặp thanh ghi này, bit GO/DONE (ADCON0<2>) được xóa và cờ ngắt ADIF bật lên. Hình 11-1 là sơ đồ khối của module A/D.

Sau khi định cấu hình module A/D như mong muốn, kênh được chọn phải được yêu cầun trước khi quá trình chuyển đổi bắt đầu. Các kênh vào tương tự phải có những bit TRIS được chọn làm ngõ vào tương ứng.

Để xác định thời gian lấy mẫu, xem phần 11.1. Sau khoảng thời gian thu thập này, quá trình chuyển đổi A/D mới có thể bắt đầu.

Để thực hiện một quá trình chuyển đổi A/D, ta thực hiện các bước sau:

a. Định cấu hình module A/D:

- Định cấu hình các chân tương tự/tham chiếu điện áp và I/O số (ADCON1)
- Chọn kênh A/D (ADCON0)
- Chọn xung nhịp cho hoạt động biến đổi A/D (ADCON0)
- Bật module A/D (ADCON0)

b. Định cấu hình ngắt A/D (nếu muốn):

- Xóa bit ADIF

- Đặt bit ADIE
- Đặt bit PEIE
- Đặt bit GIE

c. Chờ thời gian thu thập cần thiết.

d. Bắt đầu chuyển đổi:

- Đặt bit DO/DONE (ADCON0)

e. Chờ biến đổi xong, bằng cách:

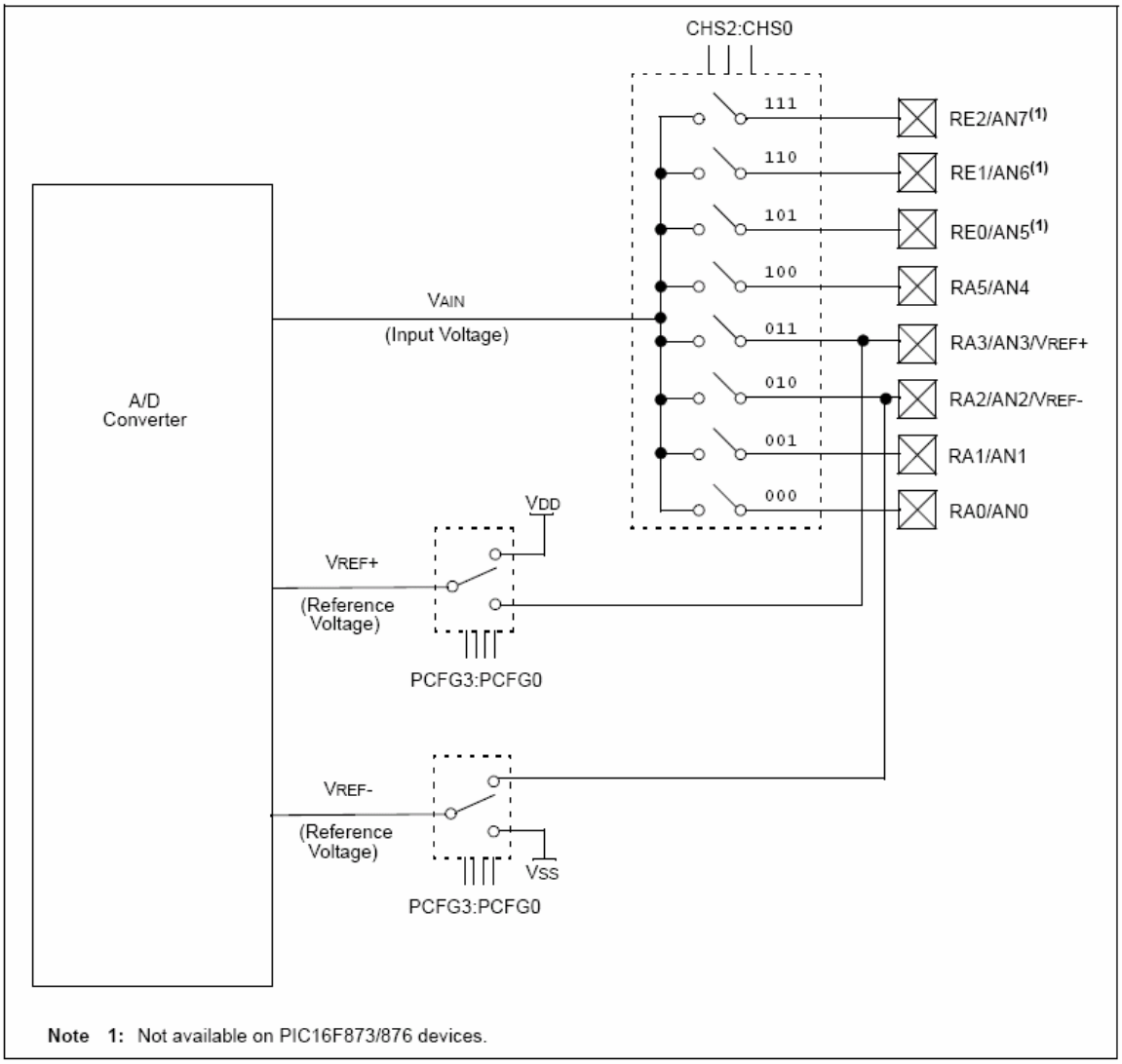
- Hỏi vòng bit GO/DONE xem xóa chưa (với các ngắt đã được cho phép); hay
- Chờ ngắt A/D

f. Đọc cặp thanh ghi kết quả ADRESH:ADRESL, xóa bit ADIF nếu cần

g. Để thực hiện quá trình chuyển đổi tiếp theo, trở về bước 1 hoặc bước 2 nếu cần.

Thời gian chuyển đổi A/D trên bit định nghĩa là T_{AD}. Cần phải chờ ít nhất $2T_{AD}$ trước khi bắt đầu quá trình thu thập số liệu tiếp theo.

FIGURE 11-1: A/D BLOCK DIAGRAM



Note 1: Not available on PIC16F873/876 devices.

11. **Các Đặc Tính Quan Trọng Của PIC**:

Tất cả các MCU PIC16F87x đều có các tính năng mở rộng để tăng tối đa khả năng của hệ thống, giảm đến mức tối thiểu giá thành cũng như các ngoại vi hỗ trợ, tiêu tốn năng lượng ít và bảo mật mã nguồn.

a. ***Các bit cấu hình***:

Các bit cấu hình có thể được dùng để thay đổi cấu hình của MCU. Các bit này được định địa chỉ tại 2007h trong bộ nhớ chương trình. Địa chỉ này nằm ngoài vùng nhớ user và chỉ có thể truy xuất được trong quá trình lập trình.

Chi tiết về các bit cấu hình như sau:

REGISTER 12-1: CONFIGURATION WORD (ADDRESS 2007h)(1)

CP1	CP0	DEBUG	—	WRT	CPD	LVP	BODEN	CP1	CP0	$\overline{\text{PWRTE}}$	WDTE	F0SC1	F0SC
bit13													b

bit 13-12, bit 5-4 — CP1:CP0: FLASH Program Memory Code Protection bits(2)
11 = Code protection off
10 = 1F00h to 1FFFh code protected (PIC16F877, 876)
10 = 0F00h to 0FFFh code protected (PIC16F874, 873)
01 = 1000h to 1FFFh code protected (PIC16F877, 876)
01 = 0800h to 0FFFh code protected (PIC16F874, 873)
00 = 0000h to 1FFFh code protected (PIC16F877, 876)
00 = 0000h to 0FFFh code protected (PIC16F874, 873)

bit 11 — DEBUG: In-Circuit Debugger Mode
1 = In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins
0 = In-Circuit Debugger enabled, RB6 and RB7 are dedicated to the debugger.

bit 10 — Unimplemented: Read as '1'

bit 9 — WRT: FLASH Program Memory Write Enable
1 = Unprotected program memory may be written to by EECON control
0 = Unprotected program memory may not be written to by EECON control

bit 8 — CPD: Data EE Memory Code Protection
1 = Code protection off
0 = Data EEPROM memory code protected

bit 7 — LVP: Low Voltage In-Circuit Serial Programming Enable bit
1 = RB3/PGM pin has PGM function, low voltage programming enabled
0 = RB3 is digital I/O, HV on $\overline{\text{MCLR}}$ must be used for programming

bit 6 — BODEN: Brown-out Reset Enable bit(3)
1 = BOR enabled
0 = BOR disabled

bit 3 — $\overline{\text{PWRTE}}$: Power-up Timer Enable bit(3)
1 = PWRT disabled
0 = PWRT enabled

bit 2 — WDTE: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1-0 — FOSC1:FOSC0: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Note 1: The erased (unprogrammed) value of the configuration word is 3FFFh.
2: All of the CP1:CP0 pairs have to be given the same value to enable the code protection scheme listed.
3: Enabling Brown-out Reset automatically enables Power-up Timer (PWRT), regardless of the value of bit $\overline{\text{PWRTE}}$. Ensure the Power-up Timer is enabled any time Brown-out Reset is enabled.

b. ***Cấu hình xung nhịp***:

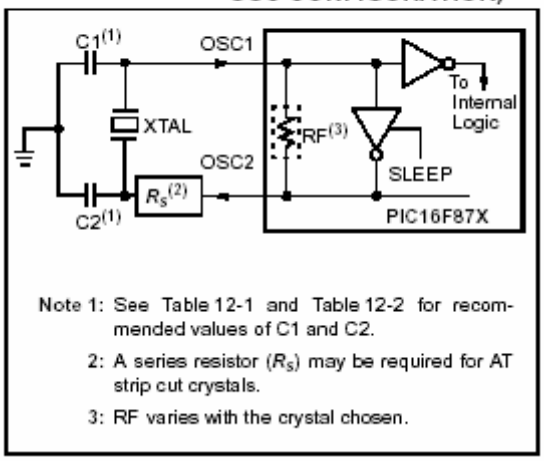
PIC16F87x có thể hoạt động với 4 loại xung nhịp khác nhau và được chọn lựa bởi người sử dụng thông qua 2 bit cấu hình:

- LP : Low Power Crytal (Thạch anh công suất thấp)
- XT: Crytal/ Resonator (Thạch anh/ mạch cộng hưởng dao động).
- HS: High speed crystal/ resonator.
- RC: Resistor/ Capacitor

• ***Mạch dao động thạch anh***:

Sơ đồ cấp xung nhịp cho PIC dùng mạch dao động thạch anh:

FIGURE 12-1: CRYSTAL/CERAMIC RESONATOR OPERATION (HS, XT OR LP OSC CONFIGURATION)



Note 1: See Table 12-1 and Table 12-2 for recommended values of C1 and C2.

2: A series resistor (R_S) may be required for AT strip cut crystals.

3: RF varies with the crystal chosen.

Thông số tụ điện và thạch anh như bảng sau:

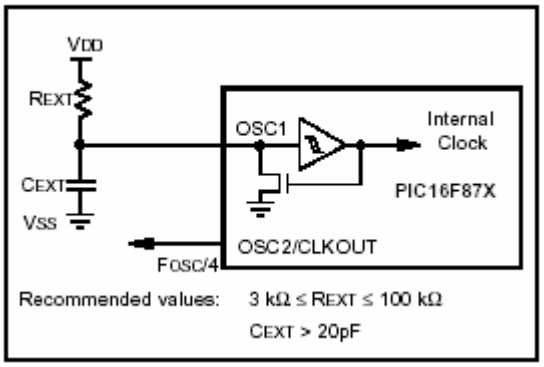
TABLE 12-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF
These values are for design guidance only. See notes following this table.			
Crystals Used			
32 kHz	Epson C-001R32.768K-A	± 20 PPM	
200 kHz	STD XTL 200.000KHz	± 20 PPM	
1 MHz	ECS ECS-10-13-1	± 50 PPM	
4 MHz	ECS ECS-40-20-1	± 50 PPM	
8 MHz	EPSON CA-301 8.000M-C	± 30 PPM	
20 MHz	EPSON CA-301 20.000M-C	± 30 PPM	

• ***Mạch dao động RC***:

Sơ đồ mạch như sau:

FIGURE 12-3: RC OSCILLATOR MODE



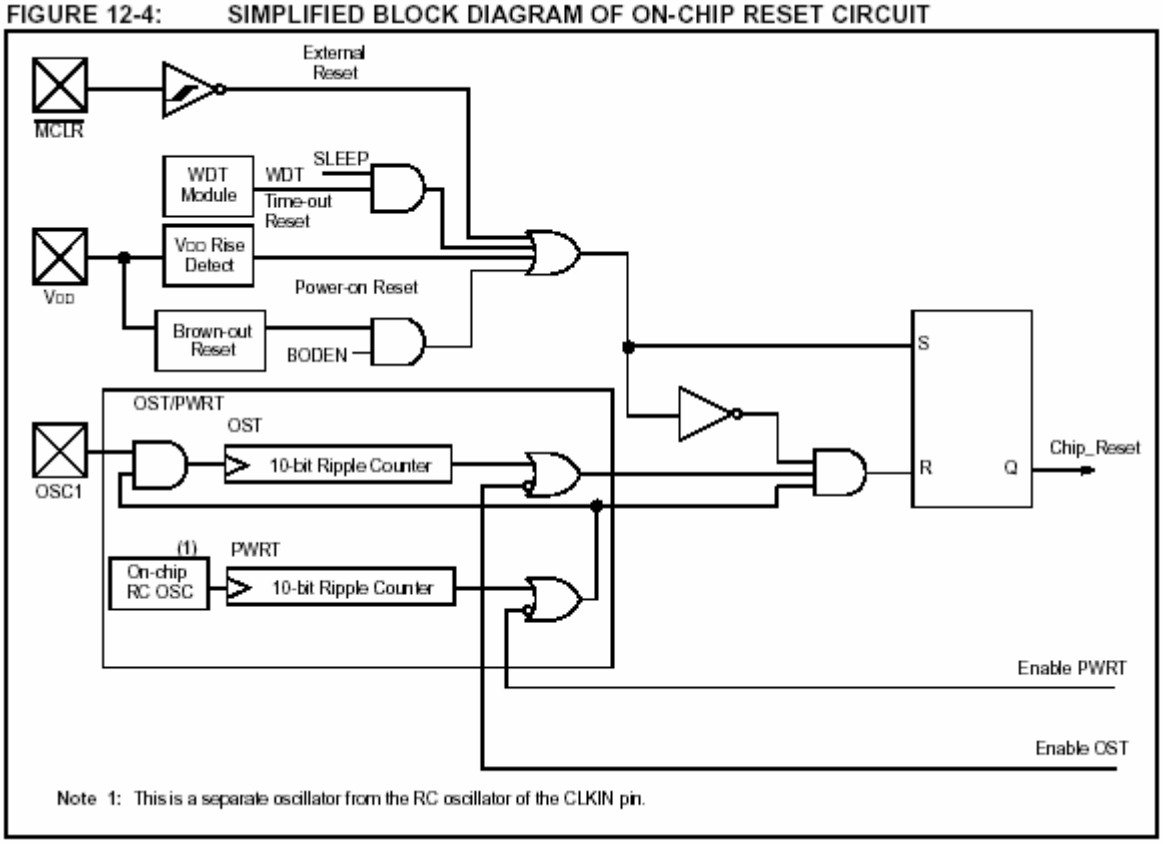
c. ***RESET***:

RESET của PIC16F87x khác so với các loại khác, MCU PIC16F87x gồm các reset sau:

- Reset khi bật nguồn (Power On Reset).
- $\overline{MCLR}$: Reset ở chế độ bình thường.
- $\overline{MCLR}$: Reset ở chế độ "ngủ".
- WDT reset: Reset ở chế độ bình thường.
- WDT Wake-up: Reset ở chế độ "ngủ".
- Brown-out reset (BOR)

Sơ đồ của mạch reset trong MCU:

FIGURE 12-4: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



Khi reset, sẽ có một số thanh gi bị thay đổi. Bảng sau tóm tắt các trạng thái của thanh ghi sau reset.

TABLE 12-5: RESET CONDITION FOR SPECIAL REGISTERS

Condition	Program Counter	STATUS Register	PCON Register
Power-on Reset	000h	0001 1xxx	---- --0x
$\overline{\text{MCLR}}$ Reset during normal operation	000h	000u uuuu	---- --uu
$\overline{\text{MCLR}}$ Reset during SLEEP	000h	0001 0uuu	---- --uu
WDT Reset	000h	0000 1uuu	---- --uu
WDT Wake-up	PC + 1	uuu0 0uuu	---- --uu
Brown-out Reset	000h	0001 1uuu	---- --u0
Interrupt wake-up from SLEEP	PC + 1(1)	uuu1 0uuu	---- --uu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0'

Note 1: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

TABLE 12-6: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets, WDT Reset	Wake-up via WDT or Interrupt
W	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	873	874	876	877	N/A	N/A	N/A
TMR0	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	873	874	876	877	0000h	0000h	PC + 1(2)
STATUS	873	874	876	877	0001 1xxx	000q quuu(3)	uuuq quuu(3)
FSR	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA	873	874	876	877	--0x 0000	--0u 0000	--uu uuuu
PORTB	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTC	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTD	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTE	873	874	876	877	---- -xxx	---- -uuu	---- -uuu
PCLATH	873	874	876	877	---0 0000	---0 0000	---u uuuu
INTCON	873	874	876	877	0000 000x	0000 000u	uuuu uuuu(1)
PIR1	873	874	876	877	r000 0000	r000 0000	ruuu uuuu(1)
	873	874	876	877	0000 0000	0000 0000	uuuu uuuu(1)
PIR2	873	874	876	877	-r-0 0--0	-r-0 0--0	-r-u u--u(1)
TMR1L	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR1H	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	873	874	876	877	--00 0000	--uu uuuu	--uu uuuu
TMR2	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
T2CON	873	874	876	877	-000 0000	-000 0000	-uuu uuuu
SSPBUF	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
SSPCON	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
CCPR1L	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR1H	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP1CON	873	874	876	877	--00 0000	--00 0000	--uu uuuu
RCSTA	873	874	876	877	0000 000x	0000 000x	uuuu uuuu
TXREG	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
RCREG	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
CCPR2L	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR2H	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP2CON	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
ADRESH	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON0	873	874	876	877	0000 00-0	0000 00-0	uuuu uu-u
OPTION_REG	873	874	876	877	1111 1111	1111 1111	uuuu uuuu
TRISA	873	874	876	877	--11 1111	--11 1111	--uu uuuu
TRISB	873	874	876	877	1111 1111	1111 1111	uuuu uuuu
TRISC	873	874	876	877	1111 1111	1111 1111	uuuu uuuu
TRISD	873	874	876	877	1111 1111	1111 1111	uuuu uuuu
TRISE	873	874	876	877	0000 -111	0000 -111	uuuu -uuu
PIE1	873	874	876	877	r000 0000	r000 0000	ruuu uuuu
	873	874	876	877	0000 0000	0000 0000	uuuu uuuu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition, r = reserved, maintain clear

Note 1: One or more bits in INTCON, PIR1 and/or PIR2 will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

3: See Table 12-5 for RESET value for specific condition.

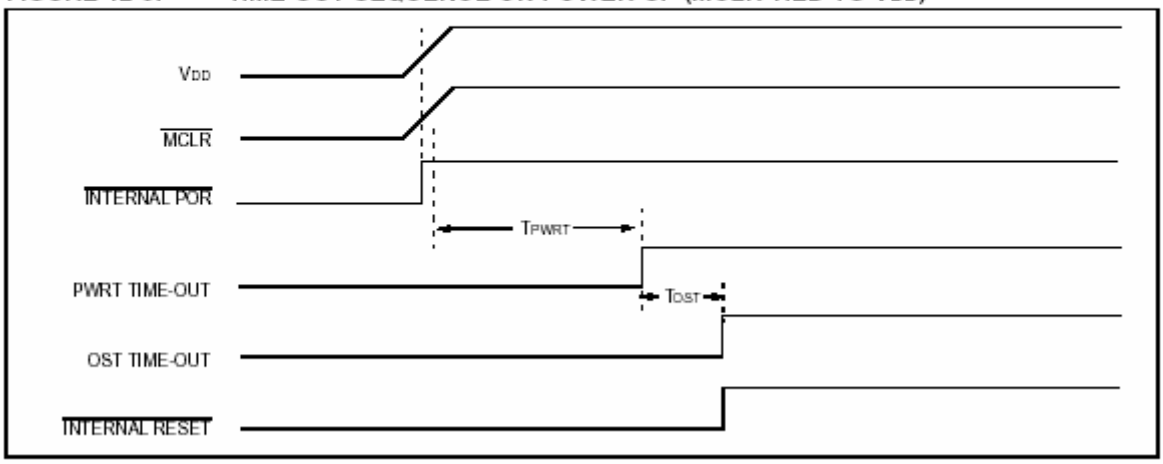
TABLE 12-6: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Devices				Power-on Reset, Brown-out Reset	$\overline{MCLR}$ Resets, WDT Reset	Wake-up via WDT or Interrupt
PIE2	873	874	876	877	-r-0 0--0	-r-0 0--0	-r-u u--u
PCON	873	874	876	877	---- --qq	---- --uu	---- --uu
PR2	873	874	876	877	1111 1111	1111 1111	1111 1111
SSPADD	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
SSPSTAT	873	874	876	877	--00 0000	--00 0000	--uu uuuu
TXSTA	873	874	876	877	0000 -010	0000 -010	uuuu -uuu
SPBRG	873	874	876	877	0000 0000	0000 0000	uuuu uuuu
ADRESL	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON1	873	874	876	877	0--- 0000	0--- 0000	u--- uuuu
EEDATA	873	874	876	877	0--- 0000	0--- 0000	u--- uuuu
EEADR	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATH	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADRH	873	874	876	877	xxxx xxxx	uuuu uuuu	uuuu uuuu
EECON1	873	874	876	877	x--- x000	u--- u000	u--- uuuu
EECON2	873	874	876	877	---- ----	---- ----	---- ----

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition, r = reserved, maintain clear

Note 1: One or more bits in INTCON, PIR1 and/or PIR2 will be affected (to cause wake-up).
2: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).
3: See Table 12-5 for RESET value for specific condition.

FIGURE 12-5: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ TIED TO V_{DD})

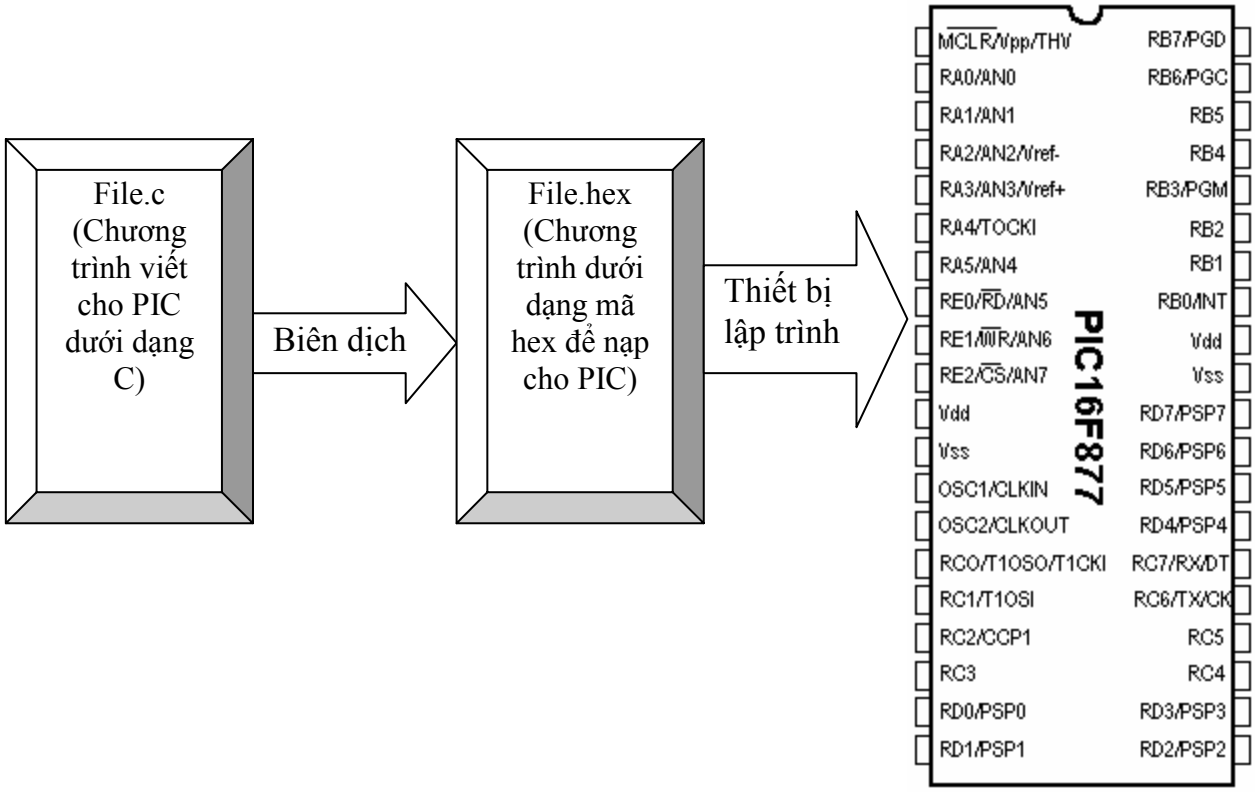


Chương II: LẬP TRÌNH CHO PIC DÙNG C COMPILER

I. GIỚI THIỆU PIC C COMPILER:

1. Giới Thiệu PIC C Compiler:

PIC C compiler là ngôn ngữ lập trình cấp cao cho PIC được viết trên nền C. chương trình viết trên PIC C tuân thủ theo cấu trúc của ngôn ngữ lập trình C. Trình biên dịch của PIC C compiler sẽ chuyển chương trình theo chuẩn của C thành dạng chương trình theo mã Hexa (file.hex) để nạp vào bộ nhớ của PIC. Quá trình chuyển đổi được minh hoạ như hình 2.1.



Hình 2.1 Quá trình lập trình, biên dịch và nạp cho PIC

PIC C compiler gồm có 3 phần riêng biệt là PCB, PCM và PCH. PCB dùng cho họ MCU với bộ lệnh 12 bit, PCM dùng cho họ MCU với bộ lệnh 14 bit và PCH dùng cho họ MCU với bộ lệnh 16 và 18 bit. Mỗi phần khác nhau trong PIC C compiler chỉ dùng được cho họ MCU tương ứng mà không cho phép dùng chung (Ví dụ không thể dùng PCM hoặc PCH cho họ MCU 12 bit được mà chỉ có thể dùng PCB cho MCU 12 bit).

2. Cài Đặt Và Sử Dụng PIC C Compiler:

a. *Cài đặt PIC C compiler*:

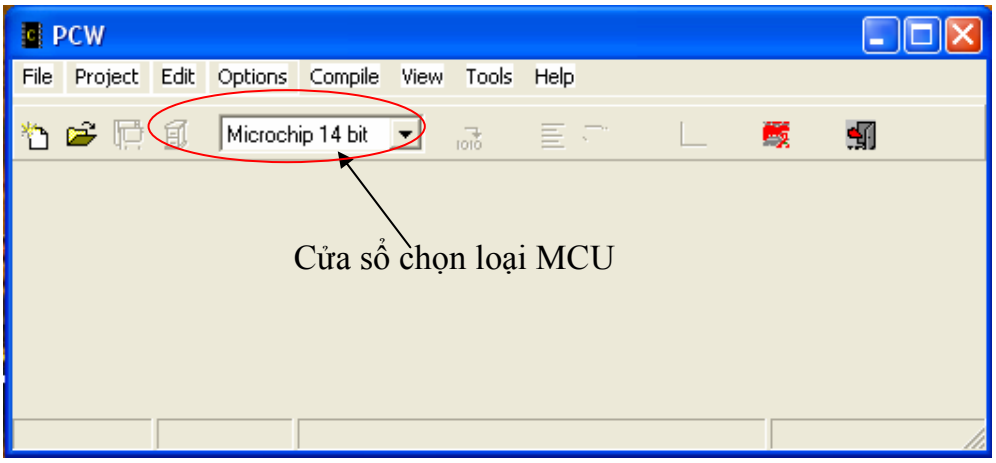
Để cài đặt PIC C compiler, bạn phải có đĩa CD chứa software PCW. Phần mềm này có thể download trên mạng ở địa chỉ . Khi có đĩa CD software, việc cài đặt PIC C compiler được thực hiện theo các bước sau:

- Từ Start menu -> chọn run -> chọn browse -> chọn thư mục PCW -> chọn setupPCW -> click OK. Khi đó xuất hiện cửa sổ welcome.
- Trên cửa sổ Welcome, click chuột vào nút Next, sau khi click Next, cửa sổ Software License Agreement sẽ xuất hiện, click nút nhấn Yes.
- Trong của sổ Readme information, click nút nhấn Next.

- Sau khi click Next trong cửa sổ Readme information, của sổ Choose Destination Location sẽ xuất hiện. Thư mục mặc nhiên để cài đặt PIC C compiler là c:\Program files\PICC. Ta có thể thay đổi thư mục cài đặt PCW bằng cách chọn nút Browse và chỉ đường dẫn tới thư mục hoặc ổ đĩa cần cài đặt, nếu muốn để ở thư mục mặc nhiên, click nút nhấn Next để tiếp tục cài đặt.
- Trong cửa sổ Select Program Folder, click nút nhấn Next.
- Click nút nhấn Next tong cửa sổ Start Copying Files sau đó chờ cho quá trình setup thực hiện.
- Trong cửa sổ Select Files .crg, nhập vào tên file pcb.crg, pcm.crg hoặc pch.crg nếu muốn dùng PIC C compiler cho MCU 12 bit, MCU 14 bit hay MCU 16, 18 bit sau đó click nút OK.
- Click nút Finish để hoàn tất việc cài đặt.

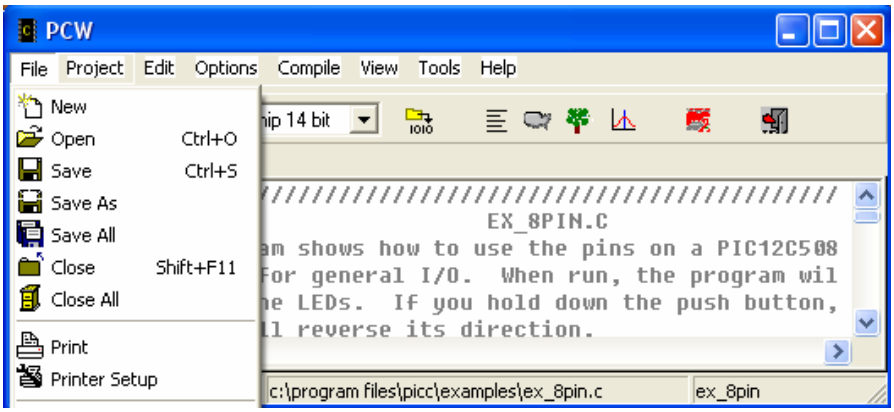
b. ***Sử dụng PIC C compiler***:

Sau khi cài đặt xong PIC C compiler, trên Desktop của window sẽ xuất hiện biểu tượng của PIC C compiler. Double Click vào biểu tượng của PIC C compiler để chạy chương trình khi đó cửa sổ chương trình của PIC C compiler sẽ xuất hiện như sau:



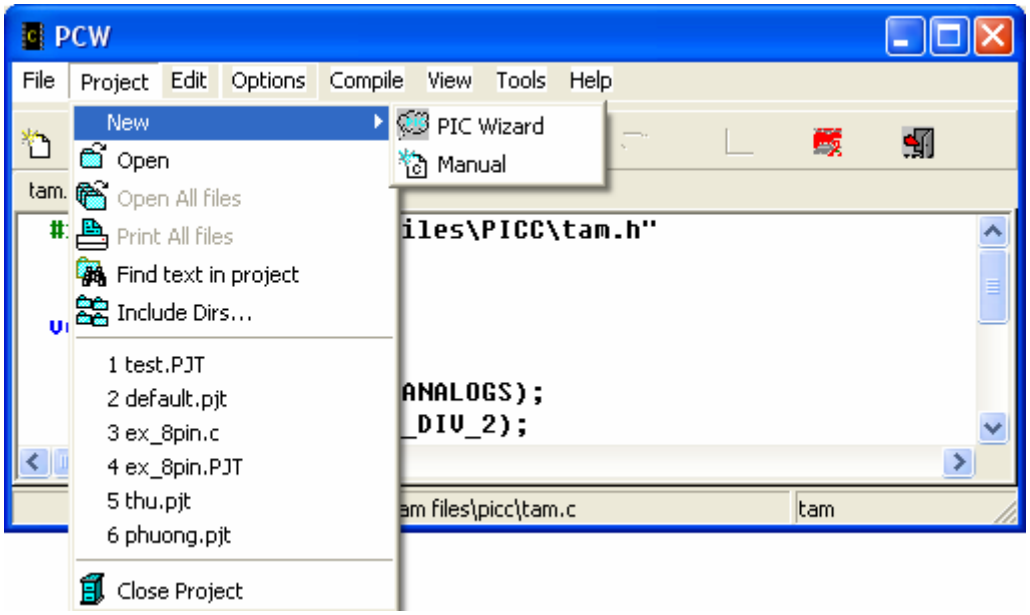
Trong cửa sổ chương trình cửa PIC C compiler gồm có các thực đơn (Menu): File, Project, Edit, Options, Compile, View, Tools và Help. Chi tiết về các thực đơn như sau:

- ***File (tệp)***: File là thực đơn quản lý tệp gồm các thực đơn như hình

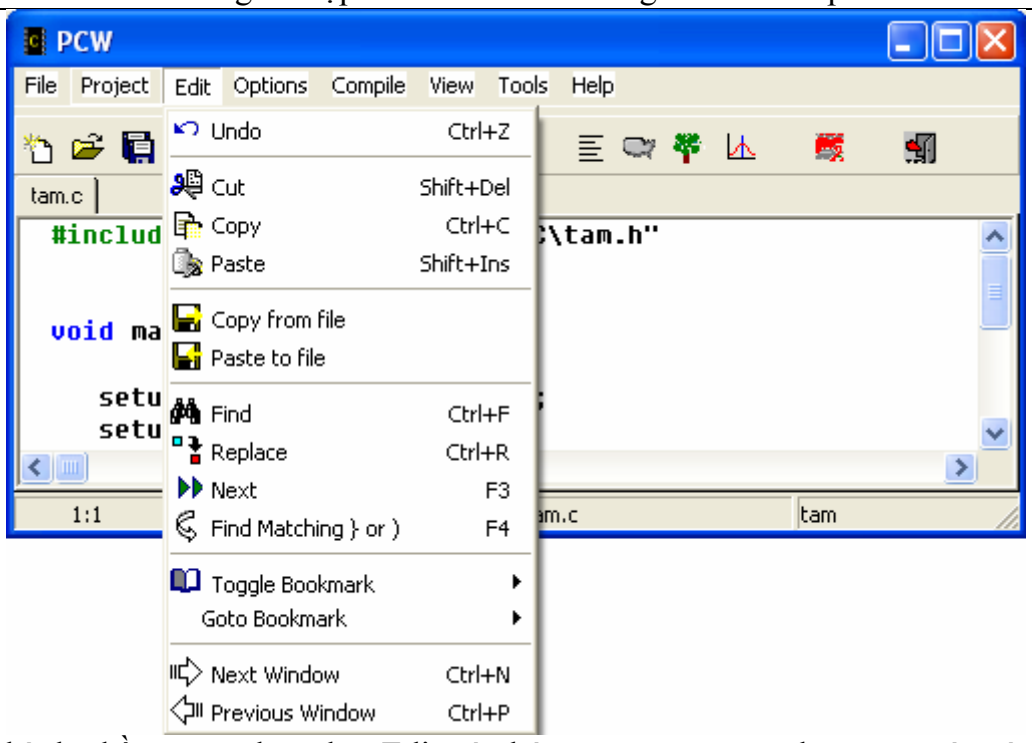


+ New: Tạo file.c mới
+ Open: Mở một file.c đã có, được lưu trữ trong đĩa.
+ Save: Lưu file.c vào đĩa.
+ Save As: Lưu trữ file.c vào đĩa cứng với tên khác.
+ Save All: Lưu trữ tất cả các file được mở vào đĩa.
+ Close: Đóng file hiện hành.
+ Close All: Đóng tất cả các file.
+ Print: In file hiện hành.

- ***Project (Dự án)***: Là thực đơn quản lý dự án (một chương trình ứng dụng). Thực đơn Project gồm các thực đơn như hình

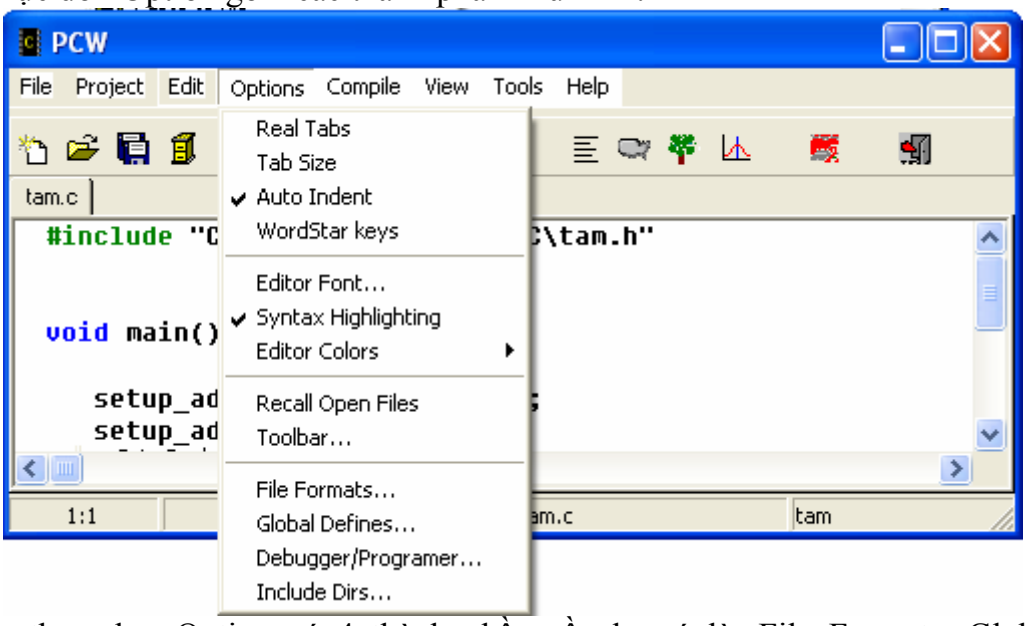


+ New: Tạo một dự án mới. Dự án mới có thể được tạo một cách thủ công hoặc tạo tự động thông qua PIC Wizard. Nếu chọn phương thức thủ công thì chỉ có file.pjt được tạo để giữ thông tin cơ bản của dự án và một file.c mặc định trước hoặc một file.c rỗng được tạo để soạn thảo chương trình. Nếu tạo dự án thông qua PIC Wizard, thì người sử dụng có thể xác định tham số của dự án và khi hoàn tất thì các file.c, file.h và file.pjt được tạo. Mã nguồn chuẩn và các hằng số được sinh ra dựa trên tham số của dự án. Việc chọn lựa các tham số cho dự án mới được thực hiện trên mẫu được PIC C compiler đề nghị, trong mẫu gồm các chọn lựa như đặc tính của đường vào ra theo chuẩn RS232, I^2C, chọn lựa timer, chọn lựa ADC, sử dụng ngắt, các driver cần thiết và tên của tất cả các chân của MCU. Sau khi hoàn tất việc chọn lựa các tham số cho dự án thì file.c và file.h sẽ tạo ra với #defines, #include và một số lệnh ban đầu cần thiết cho dự án. Đây là cách nhanh nhất để tạo một dựa án mới.
+ Open: Mở một file.pjt đã có trong đĩa.
+ Open All: Mở một file.pjt và tất cả các file dùng trong dự án.
+ Find text in project: Tìm kiếm một từ hay một ký tự trong dự án.
+ Include Dirs…: Cho phép xác định các thư mục được dùng để tìm kiếm các file include cho dự án. Thông tin này được lưu vào file.pjt.
+ Close Project: Đóng tất cả các file trong dự án.
- ***Edit***: Thực đơn Edit gồn các thành phần như hình.



Các thành phần trong thực đơn Edit có chức năng tương tự như trong các trình ứng dụng trên môi trường window quen thuộc như word, excel …

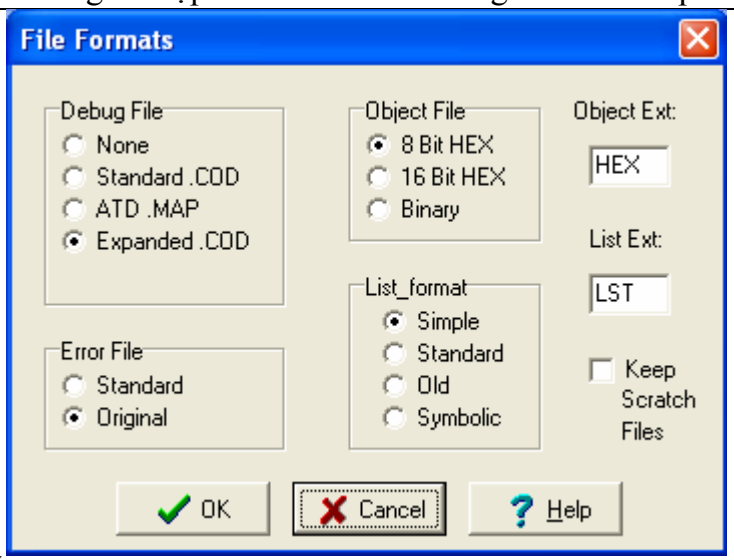
- ***Option***: Thực đơn Option gồm các thành phần như hình.



Trong thực đơn Option có 4 thành phần cần lưu ý là: File Formats, Global Defines, Debugger/Programer và Include Dirs. Các thành phần khác thì tương tự như các trình ứng dụng quen thuộc.

+ File Format: Cho phép chọn lựa kiểu định dạng của file xuất. Khi chọn Option->File Format, cửa sổ File Format sẽ xuất hiện. Trong cửa sổ File Format có các chọn lựa để chọn kiểu định dạng cho file xuất ra sau khi biên dịch.

Cửa sổ File Format có dạng như sau:



Debug File: File gỡ rối chương trình chạt trên MPLAB. Chọn Standard.COD nếu muốn chạy gỡ rối chương trình, chọn None nếu không cần chạy gỡ rối.

Error File: Xuất ra file lỗi khi chương trình có lỗi trong quá trình biên dịch. Chọn Standard cho các MCU chuẩn hiện hành của Microchip, chọn Original cho các MCU thế hệ trước của Microchip.

List Format:Chọn Simple cho định dạng cơ bản với mã C và ASM. Chọn Standard để định dạng chuẩn MPASM với mã máy. Chọn Old cho định dạng MPASM thế hệ trước. Chọn Symbolic để định dạng gồm mã C trong ASSEMBLY.

Object File: Chọn kiểu cho file.hex, Chọn 8 bit HEX cho file hex intel 8 bit và chọn 16 HEX cho file hex intel 16 bit.

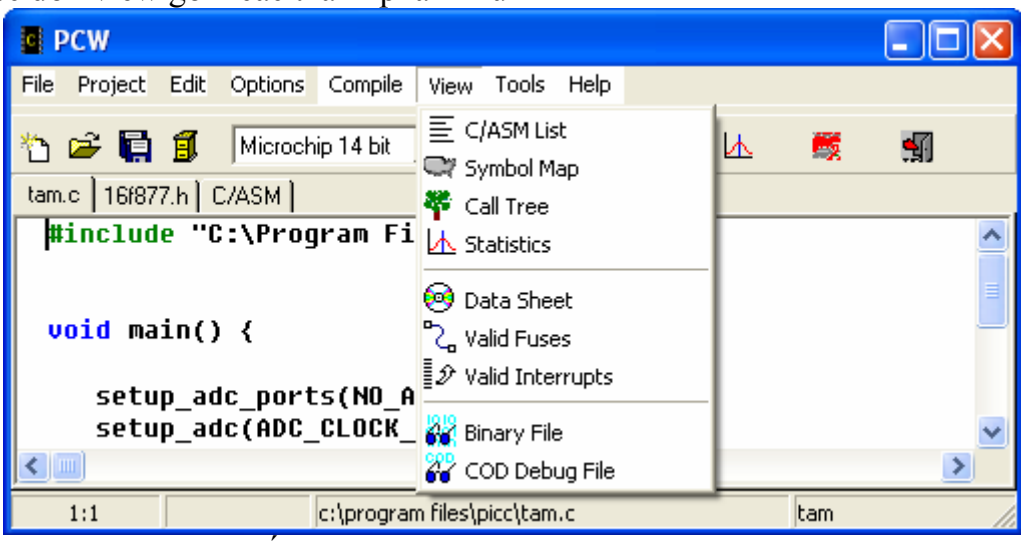
Sau khi đã chọn lựa kiểu định dạng file xuất ra sau khi biên dịch, click OK.

+ Global Defines: Cho phép đặt #define để sử dụng cho biên dịch chương trình. Điều này tương tự như việc khai báo #define ở đầu chương trình.

+ Debug/Programer: Cho phép xác định thiết bị lập trình được sử dụng khi chọn lựa công cụ lập trình cho chip.

+ Include Dirs: Tương tự như trong thực đơn Project.

- ***Compiler***: Biên dịch dự án hiện hành.
- ***View***: Thực đơn view gồm các thành phần như hình



+ C/ASM List: Mở file.lst ở chế độ chỉ đọc, file này phải được biên dịch trước từ file.c. Khi được mở, file này sẽ trình bày theo dạng vừa có mã C vừa có mã Assembly.

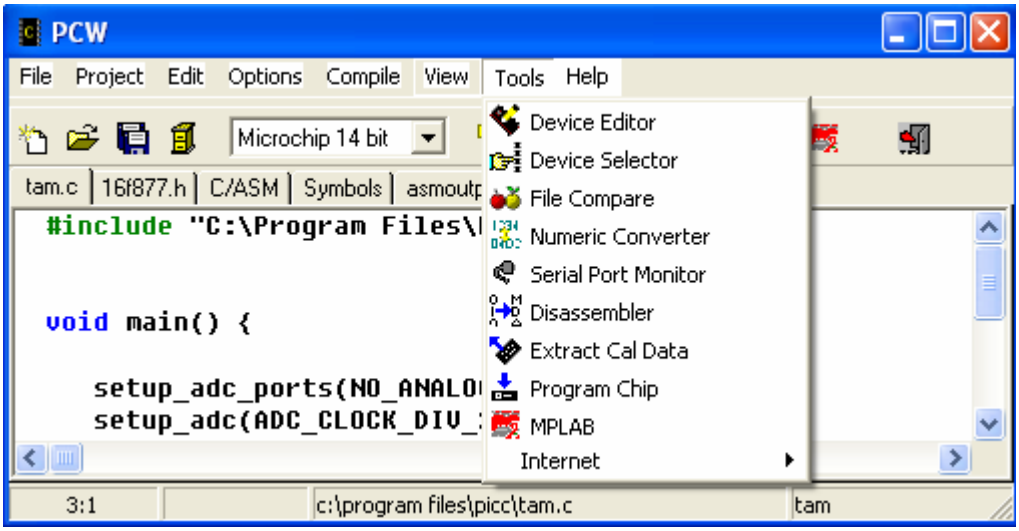
Ví dụ File.lst

……delay_ms(3);

0F2: MOVLW 05
0F3: MOVWF 08
0F4: DESCZ 08,F
0F5: GOTO 0F4
…….while input(pin_0));
0F6: BSF 0B,3

+ Symbol Map: Mở file dạng mã ASM ở chế độ chỉ đọc. File này phải được biên dịch từ file.c. File này cho biết địa chỉ của các thanh ghi sử dụng trong chương trình.
+ Binary file: Mở file nhị phân ở chế độ chỉ đọc, File này được hiển thị ở mã HEX và mã ASCII.

- ***Tool***: Thực đơn Tool quản lý một số công cụ đặc biệt. Các thành phần trong thực đơn tool như hình.



Trong thực đơn tool chỉ có một công cụ khá đặc biệt mà người sử dụng MCU cần lưu ý là công cụ disassembler, công cụ này cho phép dịch ngược file.bin hoặc file.hex thành file theo kiểu mã ASM.

- ***Help***: thực đơn trợ giúp, trong thực đơn này chứa phần hướng dẫn sử dụng PIC C compiler dưới dạng HYML.

c. ***Lập Trình Cho MCU Của Microchip Dùng PIC C Compiler***:

Các bước để lập trình cho MCU PIC dùng PIC C compiler:

- Chạy PIC C Compiler bằng cách double click vào biểu tượng của phần mềm.
- Trên Menu Bar của phần mềm, chọn Project -> New -> PIC Wizard để tạo dự án mới hoặc chọn Project –> Open để mở dự án trong đã lưu trong đĩa.
- Nếu là dự án mới thì sau khi chọn PIC Wizard, đặt tên cho dự án và click SAVE.
- Sau khi click SAVE, của sổ cho phép chọn thông số cho dự án theo mẫu hiện ra, chọn các thông số cần thiết cho dự án và click OK.
- Sau khi click OK, cửa sổ soạn thảo chương trình theo mã C xuất hiện, viết mã theo giải thuật để thực hiện dự án. Chọn File save all để lưu trữ các file trong dự án vào đĩa cứng.
- Sau khi viết mã xong, chọn Compiler -> compiler để biên dịch chương trình thành file.hex. Nếu chương trình không có lỗi thì file.hex được tạo ra còn ngược lại thì sửa lỗi chươn gtrình rồi biên dịch lại.
- Sau khi tạo được file.hex, dùng chương trình PIC downloader để nạp chương trình vào bộ nhớ FLASH của MCU.

3. **Viết Chương Trình (Mã Nguồn) Cho PIC Trên PIC C Compiler**:

Chương trình được viết trên PIC C compiler gồm 4 phần tử chính, Trong mỗi phần tử sẽ bao gồm nhiều chi tiết để tạo nên chương trình. Cấu trúc chương trình như sau:

- ***Phần ghi chú***: Ở phần ghi chú, người lập trình sẽ ghi những chú thích cần thiết cho chương trình. Phần chú thích được bắt đầu từ dấu // hoặc /* cho tới cuối hàng. Khi biên dịch, trình biên

dịch sẽ bỏ qua phần ghi chú. Phần ghi chú có thể xuất hiện bất cứ chỗ nào trong chương trình thậm chí có thể đặt ngay sau hàng mã lệnh để chú thích cho hàng lệnh.
Ví dụ:
// Đây là phần ghi chú của chương trình
/* những ghi chú này sẽ không ảnh hưởng gì tới chương trình khi biên dịch.
- ***Chỉ định các tiền xử lý***: Phần này sẽ chỉ định các tiền xử lý được sử dụng khi biên dịch. Các tiền xử lý được bắt đầu bằng dấu #.
Ví dụ: khai báo các tiền xử lý, chi tiết về từng tiền xử lý sẽ được trình bày chi tiết sau.
include // Chỉ định tiền xử lý include
device
…
- ***Định nghĩa các dữ liệu***: Đây là phần khai báo hằng, khai báo biến và kiểu dữ liệu sử dụng trong chương trình.
Ví dụ:
int a,b,c,d; // Khai báo biến a,b,c,d kiểu nguyên
- ***Định nghĩa các hàm***: Định nghĩa các hàm (Function) được dùng để thực hiện giải thuật của chương trình. Hàm có cấu trúc như sau:

Tên hàm (Các đối số của hàm)
{
Các phát biểu
}

Trong đó Tên hàm được đặt tuỳ ý của người viết chương trình. Các đối số của hàm là các thông số dùng để trao đổi dữ liệu cũa hàm, đối số có thể là rỗng nếu hàm không trao đổi dữ liệu hoặc có thể có nhiều đối số, các đối số phân cách nhau bằng dấu ‘,’
Ví dụ:
```
 void lcd_putc(char c )  // Định nghĩa hàm
{
...
}
```
Dưới đây trình bày một chương trình mẫu để minh họa cấu trúc của chương trình.
```
// Phần khai báo chỉ định tiền xử lý
#if defined(__PCB__) // Khai báo tiền
#include <16c56.h>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_A3, rcv=PIN_A2)  // Jumpers: 11 to 17, 12 to 18
#elif defined(__PCM__)
#include <16c74.h>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)  // Jumpers: 8 to 11, 7 to 12
#elif defined(__PCH__)
#include <18c452.h>
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)  // Jumpers: 8 to 11, 7 to 12
#endif
#include <ltc1298.c>
// Kết thúc phần khai báo tiền chỉ định tiền xử lý.
```

```
// Phần khai báo biến, hằng và kiểu dữ liệu
int a,b,c,d;
char *h;
struct data_record {
byte a [2];
byte b : 2; /*2 bits */
byte c : 3; /*3 bits*/
int d;}
// Kết thúc khai báo hằng, biến và kiểu dữ liệu
// Bắt đầu phần định nghĩa các hàm.
void display_data( long int data )                  // Khai báo hàm hiển thị data
        {                                           // Từ khóa bắt đầu chương trình
           char volt_string[6];                     // Khai báo biến
           convert_to_volts( data, volt_string ); // Phát biểu
           printf(volt_string);                     // Phát biểu
           printf(" (%4lX)",data);                  // Phát biểu
        }                                           // từ khoá kết thúc hàm
 main()                                             // Định nghĩa hàm
        {                                           // Từ khóa bắt đầu chương trình
          long int value;                           // Khai báo biến
          adc_init();                               // Phát biểu gọi hàm
          printf("Sampling:\r\n");                  // Phát biểu
          do                                        // Phát biểu
                {                                   // Từ khóa bắt đầu nhóm phát biểu
                   delay_ms(1000);          // Phát biểu
                   value = read_analog(0);          // Phát biểu gọi hàm
                   printf("\n\rCh0: ");             // Phát biểu
                   display_data( value );           // Phát biểu gọi hàm
                   value = read_analog(1);          // Phát biểu gọi hàm
                   printf("   Ch1: ");              // Phát biểu
                   display_data( value );           // Phát biểu gọi hàm
                 }
        while (TRUE);
        }
```

a. ***Các lệnh tiền xử lý***:

Tiền xử lý gồm 55 lệnh chi tiết như sau:

#DEFINE ID STRING.
#ELSE.
#ENDIF.
#ERROR.
#IF expr.
#IFDEF id.
#INCLUDE "FILENAME" .
#INCLUDE <FILENAME> .
#LIST.
#NOLIST.
#PRAGMA cmd.
#UNDEF id.
#INLINE

#INT_DEFAULT
#INT_GLOBAL
#INT_xxx
#SEPARATE
__DATE__
__DEVICE__
__FILE__
__LINE__
__PCB __
__PCM__
__PCH__
__TIME__
#DEVICE CHIP
#ID NUMBER
#ID "filename"
#ID CHECKSUM
#FUSES options
#SERIALIZE
#TYPE type=type
#USE DELAY CLOCK
#USE FAST_IO
#USE FIXED_IO
#USE I2C
#USE RS232
#USE STANDARD_IO
#ASM
#BIT id=const.const
#BIT id=id.const
#BYTE id=const
#BYTE id=id
#LOCATE id=const
#ENDASM
#RESERVE
#ROM
#ZERO_RAM
#BUILD
#FILL_ROM
#CASE
#OPT n
#PRIORITY
#ORG
#IGNORE_WARNINGS

- ***#ASM và #ENDASM***: Đây là cặp lệnh đi kèm với nhau để cho phép chèn đoạn mã dạng assembly vào chương trình.

Cú pháp của cặp lệnh này như sau:

```
#asm
        mã assembly
# endasm
```

Ví dụ sau minh họa cách sử dụng cặp lệnh trên vào chương trình.

```
int find_parity (int data)
{
int count;
#asm                    //Khai báo bắt đầu đoạn mã assembly
        movlw 0x8
        movwf count
        movlw 0
loop:   xorwf data,w
        rrf data,f
        decfsz count,f
        goto loop
        movwf _return_
#endasm                 // khai báo kết thúc đoạn mã assembly
}
```

- ***#BIT***:

+ *Cú pháp*:

#bit id = x.y

id là tên hợp lệ trong C, x là hằng số hoặc biến trong C, y là hằng số từ 0 -7

+ *Mục đích*:

Tạo một biến mới (1 bit) trong C và đặt nó vào trong bộ nhớ tại byte x, bit y. Lệnh này thường được sử dụng để truy xuất trực tiếp từ C tới một bit trong thanh ghi chức năng đặc biệt.

Ví dụ: ví dụ dưới dây minh hoạ cách sửû dụng lệnh #bit.

```
#bit T0IF = 0xb.2
...
T0IF = 0; // Clear Timer 0 interrupt flag
int result;
#bit result_odd = result.0
...
if (result_odd)
```

- ***# BUILD***:

+ *Cú pháp*:

#build(***segment*** =***address***)

#build(***segment*** = ***address***,***segment*** = ***address***)

#build(***segment*** =***start***:***end***)

#build(***segment*** = ***start***: ***end***, ***segment*** = ***start***: ***end***)

Trong đó: segment là một đoạn bộ nhớ tiếp theo mà đã được ấn định vị trí (Bộ nhớ Reset, ngắt). Address là địa chỉ của bộ nhớ ROM trong MCU, start và end được sử dụng để xác định địa chỉ đấu và cuối của vùng nhớ được dùng.

+ *Mục đích*:

Đối với các MCU PIC18XXX dùng bộ nhớ ngoài hoặc các MCU PIC 18XXX không có bộ nhớ ROM bên trong, lệnh này cho phép biên dịch trực tiếp để sử dụng bộ nhớ ROM.

Ví dụ: ví dụ dưới dây minh hoạ cách sử dụng lệnh #build.

```
#build(memory=0x20000:0x2FFFF)      //ấn định không gian nhớ.
#build(reset=0x200,interrupt=0x208)     //ấn định vị trí đầu của các vector ngắt và reset
#build(reset=0x200:0x207, interrupt=0x208:0x2ff)   //ấn định giới hạn không gian của các
                                                    // vector ngắt và reset
```

- ***#BYTE***:

+ *Cú pháp*:

#byte id = x

Trong đó: id là tên hợp lệ trong C, x là một biến trong C hoặc là hằng số.

+ *Mục đích*: Nếu id đã được xác định như một biến hợp lệ trong C thì lệnh này sẽ đặt biến C vào trong bộ nhớ tại địa chỉ x, biến C này không được thay đổi khác với định nghĩa ban đầu. Nếu id là biến chứ biết thì một biến C mới sẽ được tạo và đặt vào bộ nhớ tạo địa chỉ x.

Ví dụ: ví dụ dưới dây minh hoạ cách sử dụng lệnh #byte.

```
#byte status = 3
#byte b_port = 6
struct {
short int r_w;
short int c_d;
int unused : 2;
int data : 4; } a_port;
#byte a_port = 5
...
a_port.c_d = 1;
```

- ***#DEFINE***:

+ *Cú pháp*:

#define id text hoặc #define id(x,y...) text

Trong đó: id là một định nghĩa tiền xử lý. text là đoạn văn bản bất kỳ. x,y là một định nghĩa tiền xử lý nội đặt cách nhau bởi dấu phẩy

+ *Mục đích*:

Định nghĩa một hằng hay một tham số thường sử dụng trong chương trình

Ví dụ: ví dụ dưới dây minh hoạ cách sủ dụng #define.

```
#define  BITS  8
a=a+BITS;                  //tương tự như   a=a+8;
#define hi(x)  (x<<4)
a=hi(a);                    //tương tự như   a=(a<<4);
```

- ***# DEVICE***:

+ *Cú pháp*:

#device chip options

Trong đó: chip là tên bộ vi xử lý (ví dụ như: 16f877), Options là các toán tử tiêu chuẩn được quy định trong mỗi chip. Các option bao gồm:

- *=5: Sử dụng con trỏ 5 bit (Cho tất cả các MCU)
- *=8: Sử dụng con trỏ 8 bit (Cho MCU 14 và 16 bit)
- *=16: Sử dụng con trỏ 16 bit (Cho MCU 14 bit)
- ADC=xd9: x là số bit trả về sau khi gọi hàm read_adc()
- ICD=TRUE: Tạo ra mã tương ứng Microchips ICD debugging hardware.

+ *Mục đích*:

Định nghĩa chip sử dụng. Tất cả các chương trình đều phải sử dụng khai báo này nhằm định nghĩa chân trên chip và một số định nghĩa khác của chip.

Ví dụ: ví dụ dưới dây minh hoạ cách sử dụng #device.

```
#device PIC16C74
#device PIC16C67 *=16
#device *=16  ICD=TRUE
#device PIC16F877 *=16 ADC=10
```

- ***__DEVICE__***:

+ *Cú pháp*:

__device__

+ *Mục đích*:

Định nghĩa này được dùng để nhận dạng số cơ bản của MCU đang dùng (từ # divice). Thông thường thì số cơ bản là số nằm ngay sau phần chữ trong mã số của MCU. Ví dụ như MCU đang dùng là loại PIC16C71 thì số cơ bản là 71.

Ví dụ: ví dụ dưới dây minh hoạ cách sử dụng __device__.

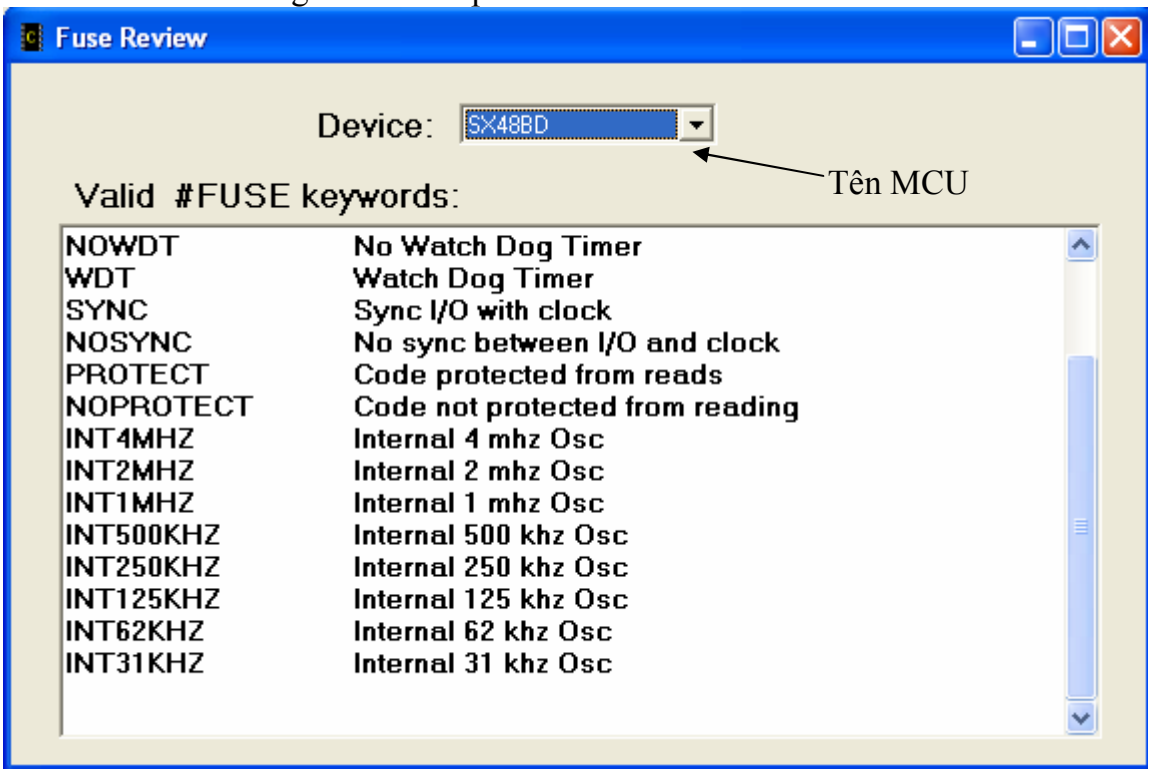
```
#if __device__==71
SETUP_ADC_PORTS( ALL_DIGITAL );
#endif
```

- ***#FUSE***:

+ *Cú pháp*:

#fuse option

Trong đó option thay đổi tuỳ thuộc vào thiết bị. Danh sách các option hợp lệ được đặt tại đầu của file.h trong phần chú thích của mỗi thiết bị. Các option được diễn giải chi tiết trong thực đơn View-> Fuse valid trong PIC C compiler. Cửa sổ FUSE Review như sau:



+ *Mục đích*:

Định nghĩa này được dùng để chỉ ra fuses nào được đặt vào MCU khi lập trình cho nó. Chỉ dẫn này không ảnh hưởng gì đến quá trình biên dịch như nó được xuất ra file output.

Ví dụ:

```
#fuses HS,NOWDT
```

- ***#ID***:

+ *Cú pháp*:

#ID number 16

#ID number, number, number, number

#ID "filename"

#ID CHECKSUM

Trong đó Number16, 4 lần lượt là các số 16 bit và 4 bit, filename là tên file có thực trong bộ nhớ máy tính, checksum à một loại từ khóa.

+ *Mục đích*:

Định nghĩa này được dùng để chỉ ra từ id nào được lập trình vào MCU. Chỉ dẫn này không ảnh hưởng gì đến quá trình biên dịch như nó được xuất ra file output.

Ví dụ:

#id 0x1234
#id "serial.num"
#id CHECKSUM

- ***#IF, #ELSE, #ELIF, #ENDIF***:

+ *Cú pháp*:

#if *biểu thức*
phát biểu
#elif *biểu thức*
phát biểu
#else
phát biểu
#endif

+ *Mục đích*:

Kiểm tra điều kiện của biểu thức, nếu điều kiện là đúng thì thực hiện phát biểu ở hàng ngay sau đó còn ngược lại nếu biểu thức là sai thì thực hiện phát biểu ngay sau #else.

Ví dụ:

#if MAX_VALUE > 255
value;
#else
int value;
#endif

- ***#INCLUDE***:

+ *Cú pháp*:

#include <***filename***> hoặc #include "***filename***"

Trong đó filename là tên file hợp lệ trong PC.

+ *Mục đích*:

Bộ tiền xử lý sẽ sử dụng thông tin cần thiết được chỉ ra trong filename trong quá trình biên dịch để thực thi lệnh trong chương trình chính. Tên file nếu đặt trong dấu " " sẽ được tìm kiếm trước tiên, nếu đặt trong dấu <> sẽ được tìm sau cùng. Nếu đường dẫn không chỉ rõ, trình biên dịch sẽ thực hiện tìm kiếm trong thư mục chứa project đang thực hiện. Khai báo #include <> được sử dụng hầu hết trong các chương trình để khai báo thiết bị (IC) đang sử dụng cũng như cần kế thừa kết quả một chương trình đã có trước đó.

Ví dụ:

#include <16C54.H>
#include<C:\INCLUDES\COMLIB\MYRS232.C>

- ***#INLINE***:

+ *Cú pháp*:

#inline

+ *Mục đích*:

Thông báo cho trình biên dịch thực thi tức thời hàm sau khai báo #inline. Tức là sẽ copy nguyên bản code đặt vào bất cứ nơi nào mà hàm được gọi. Điều này sẽ tăng tốc độ xử lý và khoảng trống trong vùng nhớ stack. Nếu không có chỉ thị này, trình biên dịch sẽ lựa chọn thời điểm tốt nhất để thực thi procedures INLINE.

Ví dụ:

#inline
swapbyte(int &a, int &b)

```
{
        int t;
        t=a;
        a=b;
        b=t;
}
```

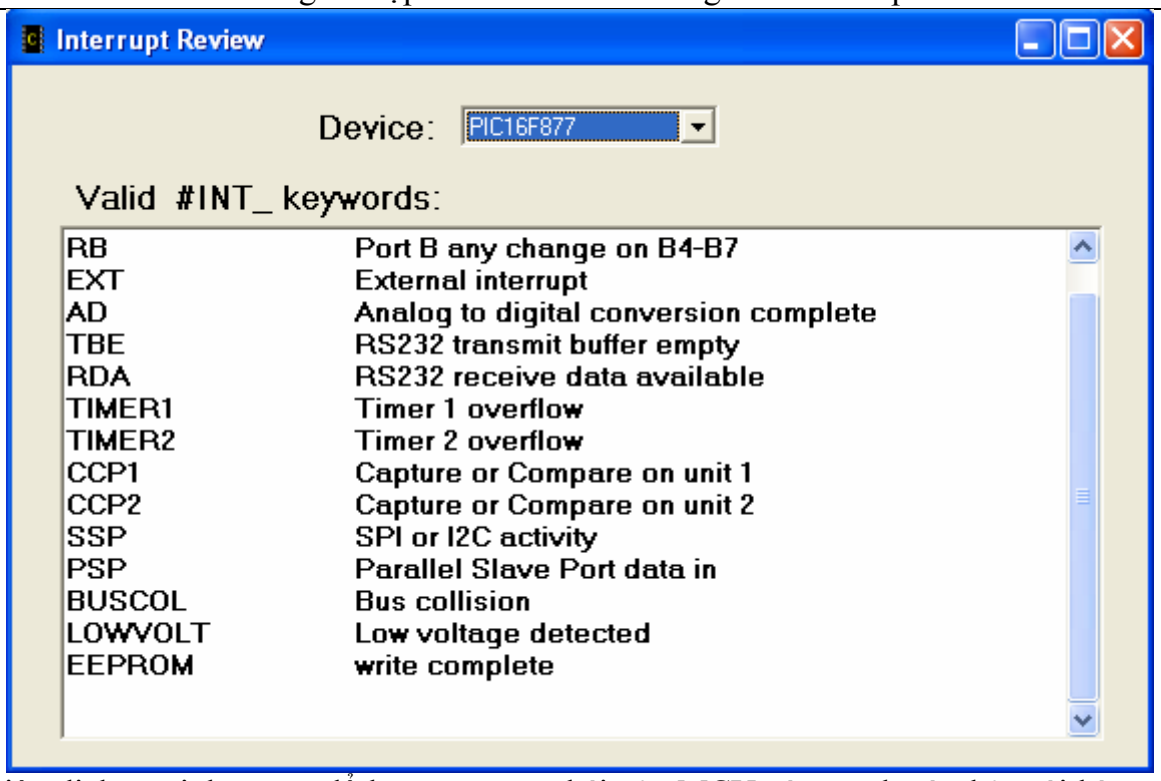
- ***#INT_xxxx***:

+ *Cú pháp*:

#INT_AD	Hoàn tất chuyển đổi ADC
#INT_ADOF	Hết thời gian chuyển đổi ADC
#INT_BUSCOL	Đụng Bus
#INT_BUTTON	Nút nhấn
#INT_CCP1	CCP1
#INT_CCP2	CCP2
#INT_COMP	Phát hiện so sánh
#INT_EEPROM	Ghi EEPROM hoàn tất
#INT_EXT	Ngắt ngoài
#INT_EXT1	Ngắt ngoài 1
#INT_EXT2	Ngắt ngoài 2
#INT_I2C I2C	Ngắt(chỉ trong 14000)
#INT_LCD	LCD tích cực
#INT_LOWVOLT	Phát hiện điện áp thấp
#INT_PSP	Nhận dữ liệu từ cổng song song
#INT_RB Port B	Thay đổi của bit B4-B7
#INT_RC Port C	Thay đổi của bit C4-C7
#INT_RDA RS232	Đang nhận dữ liệu qua cổng nối tiếp
#INT_RTCC	Tràn timer 0
#INT_SSP	SPI hoặc I2C đang làm việc
#INT_TBE RS232	Bộ đệm truyền rỗng
#INT_TIMER0	Timer 0 tràn
#INT_TIMER1	Timer 1 tràn
#INT_TIMER2	Timer 2 tràn
#INT_TIMER3	Timer 3 tràn

+ *Mục đích*:

INT_xxxx được dùng để chỉ dẫn cho biết hàm đi kém là một hàm phục vụ ngắt. Ha2m phục vụ ngắt có thể là một hàm không có tham số. Không phải các chỉ dẫn trên dùng được cho tất cả các MCU mà có thể có một số MCU chỉ sử dụng một số chỉ dẫn mà thôi. Chi tiết về các chỉ dẫn trong các MCU có thể xem trong thực đơn view -> Valid ints. Cửa sổ interrupt review như sau:



Trình biên dịch sẽ sinh ra mã để lưu trữ trạng thái của MCU vào stack và nhảy tới hàm phục vụ ngắt khi phát hiện thấy ngắt tương ứng. Sau khi thực hiện xong chương trình phục vụ ngắt, các trạng thái ban đầu của CPU được lưu trữ trong stack sẽ được lấy lại và đồng thời xoá bỏ cờ ngắt. Chương trình ứng dụng phải gọi hàm ENABLE_INTERRUPTS(INT_xxxx) cho phép các ngắt làm việc.

Ví dụ:

```
#int_ad
adc_handler()
{
adc_active=FALSE;
}
#int_rtcc noclear
isr()
{
...
}
```

- ***INT_DEFAULT***:

+ *Cú pháp*:

```
#int_default
```

+ *Mục đích*:

Hàm theo sau chỉ thị này sẽ được gọi nếu có xung kích ngắt và sẽ không có cờ ngắt nào được set. Nếu có cờ ngắt nhưng không có một trigger nào cả thì hàm #INT_DEFAULT sẽ được khởi gọi

Ví dụ:

```
#int_default
default_isr()
{
    printf("Unexplained interrupt\r\n");
}
```

- ***#INT_GLOBAL***:

+ *Cú pháp*:

```
#int_global
```

+ *Mục đích*:

Hàm sau chỉ thị này sẽ được gọi khi có tín hiệu ngắt được gởi đi. Thông thường, không nên sử dụng. Nếu sử dụng, trình biên dịch không tạo code khởi động cũng như code xoá và save trên thanh ghi

Ví dụ:

```
#int_global
isr()
{
    #asm
    bsf   isr_flag
    retfie
    #endasm
}
```

- ***#LOCATE***:

+ *Cú pháp* :

```
#locate id=x
```

Trong đó: id là tên biến trong C, x là địa chỉ vùng nhớ.

+ *Mục đích*:

Hoạt động của #LOCATE tương tự như #BYTE tuy nhiên #LOACTE sẽ không cho xử dụng vùng nhớ này vao các mục đích khác.

Ví dụ:

```
float x;
#locate x=0x50   // Biến x là biến kiểu thực và được đặt tại địa chỉ 50 – 53 và C sẽ không
                 //dùng vùng nhớ này cho các biến khác.
```

- ***#ORG***:

+ *Cú pháp*:

```
#org start, end
Hoặc
#org segment
Hoặc
#org start, end {}
Hoặc
#org start, end auto=0
#org start,end DEFAULT
Hoặc
#org DEFAULT
```

Trong đó: Start là địa chỉ đầu của vùng nhớ được chỉ định, end là địa chỉ cuối cùng của vùng nhớ được chỉ định, segment là địa chỉ đầu của vùng nhớ ROM ngay sau #ORG trước

+ *Mục đích*:

Tiền xử lý này sẽ đặt hàm theo sau nó vào vùng nhớ ROM xác định

Ví dụ:

```
#ORG 0x1E00, 0x1FFF
MyFunc()      // Hàm MyFunc() sẽ được đặt trong ROM tại vùng nhớ 0x1E00 - 0x1FFF
{
….
}
#ORG 0x1E00
Anotherfunc()
```

```
{
…
}
#ORG 0x800, 0x820 {}
#ORG 0x1C00, 0x1C0F
CHAR CONST ID[10}= {"123456789"};
#ORG 0x1F00, 0x1FF0
Void loader()
{
…
}
```

- ***__PCB__***:

+ *Cú pháp*:

__pcb__

+ *Mục đích*:

Tiền xử lý này được định nghĩa để xác định trình biện dịch pcb

Ví dụ:

```
#ifdef __pcb__
#device PIC16c54
#endif
```

- ***__PCM__***:

+ *Cú pháp*:

__pcm__

+ *Mục đích*:

Tiền xử lý này được định nghĩa để xác định trình biện dịch pcm

Ví dụ:

```
#ifdef __pcm__
#device PIC16F877
#endif
```

- ***__PCH__***:

+ *Cú pháp*:

__pch__

+ *Mục đích*:

Tiền xử lý này được định nghĩa để xác định trình biện dịch pch

Ví dụ:

```
#ifdef __pch__
#device PIC18c452
#endif
```

- ***#USE DELAY***:

+ *Cú pháp*:

#use delay (***clock=speed***)

hoặc

#use delay(***clock=speed***, ***restart_wdt***)

Trong đó speed là tốc độ xung nhịp của thạch anh, là hằng số nằm trong khoảng từ 0 – 100000000 (Từ 0 – 100 MHz)

+ *Mục đích*:

Đây là hàm thư viện của PIC C Compiler chứa các hàm delay_ms(), delay_us(). Khai báo tiền xử lý này sẽ cho phép sử dụng các hàm trên trong chương trình.

Ví dụ:

#use delay (clock=20000000)
#use delay (clock=32000, RESTART_WDT)

- ***# USE FAST_I/O***:

+ *Cú pháp*:

#use fast_io (***port***)

trong đó port là các ký tự A – G.

+ *Mục đích*:

Đây là chỉ định tiền xử lý để định dạng vào ra cho các cổng xuất nhập dữ liệu. Khi dùng chỉ định tiền xử lý này người lập trình phải chắc chắn thanh ghi đã được set đúng thông qua lệnh set_tris_X().

Ví dụ:

#use fast_io(A)

- ***# USE FIXED_I/O***:

+ *Cú pháp*:

#use fixed_io (***port_outputs=pin***, ***pin***?)

trong đó port là các ký tự A – G, pin là chân tương ứng trong port

+ *Mục đích*:

Đây là chỉ định tiền xử lý để định dạng vào ra cho các bit trong cổng xuất nhập dữ liệu.

Ví dụ:

#use fixed_io(a_outputs=PIN_A2, PIN_A3) // Chân A2 và A3 của port A là chân xuất

…

#use fixed_io(a_inputs=PIN_A1, PIN_A5) // Chân A1 và A5 của port A là chân nhập

- ***#USE I2C***:

+ *Cú pháp*:

#use i2c (***options***)

Trong đó options là các phần như sau và được phân cách bởi dấu ‘,’.

MASTER : Đặt chế độ chủ
SLAVE : Đặt chế độ tớ
SCL=pin : Xác định địa chỉ SCL
SDA=pin : Xác định địa chỉ SDA
ADDRESS=nn :Xác định địa chỉ ở chế độ tớ
FAST : Sử dụng đặc tính nhanh của I2C
SLOW : Sử dụng đặc tính chậm của I2C
RESTART_WDT Khởi động lại WDT trong khi chờ đọc I2C
FORCE_HW Sử dụng các hàm phần cứng của I2C.
NOFLOAT_HIGH : Không cho phép thả nổi tín hiệu
SMBUS : Dùng bus tương tự như bus I2C.

+ *Mục đích*:

Đây là thư viện chứa các hàm sử dụng trong chế độ I2C như: I2C_START, I2C_STOP, I2C_READ, I2C_WRITE and I2C_POLL. Khai báo tiền xử lý này để dùng các hàm thư viện của nó.

Ví dụ:

#use I2C(master, sda=PIN_B0, scl=PIN_B1)
#use I2C(slave,sda=PIN_C4,scl=PIN_C3, address=0xa0,FORCE_HW)

- ***#USE RS232***:

+ *Cú pháp*:

#use rs232 (***options***)

Trong đó options là các phần như sau và được phân cách bởi dấu ‘,’.

STREAM=id : Nhận dạng nhóm cổng RS232

BAUD=x : Đặt tốc độ Baud
XMIT=pin: Đặt chân truyền
RCV=pin : Đặt chân nhận
FORCE_SW :sinh mã truyền nối tiếp khi chân UART được xác định.
BRGH1OK :Cho phép huỷ tốc độ baud khi có vấn đề về tốc độ baud trong chip
DEBUGGER :Xác định đường truyền/ nhận dữ liệu thông qua bộ CCS IDC . Mạc nhiên là chân B3, dùng XMIT= và RCV = để thay đổi chân, cả 2 đường tuyền / nhận có thể sử dụng trên 1 chân.
RESTART_WDT : Khởi động lại Watch Dog timer.
INVERT : Đổi cực tính của chân nối tiếp
PARITY=X : Sốbit chẵn lẻ, X là N, E, O
BITS =X : Số bit data, x có giá trị từ 5 - 9
FLOAT_HIGH : Cổng RS232 sẽ ở mức cao khi không có dữ liệu.
RS232_ERRORS: Bit bào lỗi RS232
LONG_DATA : Dữ liệu nhận về từ RS232 sẽ ở dạng int16
DISABLE_INTS: Xoá ngắt RS232
+ *Mục đích*:
Định cấu hình RS232.
Ví dụ:
#use rs232(baud=9600, xmit=PIN_A2,rcv=PIN_A3) // Cổng RS232 với tốc độ buad là 9600, Chân A2 là chân truyền, chân A3 là chân nhận.

- ***# USE STANDARD_IO***:

+ *Cú pháp*:
#USE STANDARD_IO (***port***)
Trong đó port là A - G
+ *Mục đích*:
Dùng để định cấu hình vào ra cho các cổng giao tiếp dữ liệu

- ***# ZERO_RAM***:

+ *Cú pháp*:
#zero_ram
+ *Mục đích*:
Reset tất cả các thanh ghi trước khi thực hiện chương trình.
Ví dụ:

```
#zero_ram
void main()
{
…
}
```

b. ***Định nghĩa các kiểu dữ liệu***:
Kiểu dữ liệu được dùng để khai báo biến, hằng. Các biến được khai báo như sau:
Kiểu dữ liệu biến1, biến2, …;
Ví dụ: int a,b,c;

- ***Các kiểu dữ liệu đơn giản***:

Các kiểu dữ liệu đơn giản dùng trong PIC C compiler tương tự như trong C chuẩn, gồm các kiểu như trong bảng sau:

Kiểu dữ liệu	Mô tả
int1	Định nghĩa một dữ liệu 1 bit (kiểu nguyên)
int8	Định nghĩa một dữ liệu 8 bit (kiểu nguyên)
int16	Định nghĩa một dữ liệu 16 bit (kiểu nguyên)

int32	Định nghĩa một dữ liệu 32 bit (kiểu nguyên)
char	Định nghĩa một dữ liệu kiểu ký tự 8 bit
float	Định nghĩa một dữ liệu 32 bit dạng dấu chấm động (kiểu thực)
short	Mặc nhiên là int1
int	Mặc nhiên là int8
long	Mặc nhiên là int16
void	Chỉ một kiểu dữ liệu không xác định
static	Định nghĩa biến tĩnh toàn cục và có giá trị ban đầu bằng 0. Khi khai báo biến này thì bộ nhơ sẽ dành một vùng nhớ tuỳ theo kiểu biến để lưu trữ và vùng nhớ này được giữ cho dù biến đó không được sử dụng.
auto	Định nghĩa một biến kiểu động, biến này chỉ tồn tại khi hàm sử dụng nó hoạt động, vùng nhớ chứa biến này sẽ được trả lại khi hàm thực hiện xong.
double	Dự trữ một word nhớ nhưng không hỗ trợ kiểu dữ liệu
extern	Kiểu dữ liệu mỡ rộng
register	Kiểu thanh ghi

- ***Dữ liệu kiểu liệt kê***:

Dữ liệu kiểu liệt kê là loại dữ liệu mở rộng để dinh nghĩa thêm kiểu dữ liệu. Khai báo kiểu liệt kê được thực hiện như sau:

enum tên biến {liệt kê các giá trị}

Ví dụ:

```
enum boolean {true,false};
boolean j; // biến j là biến kiểu boolean sẽ có các giá trị là true hay false.
```

- ***Dữ liệu kiểu cấu trúc***:

Dữ liệu kiểu cấu trúc là một dạng dữ liệu phức tạp được định nghĩa để mở rộng thêm các kiểu dữ liệu sử dụng trong chương trình. Định nghĩa kiểu cấu trúc như sau:

```
Struct tên kiểu {kiểu dữ liệu tên biến : miền giá trị}
```

Ví dụ:

```
struct port_b_layout {int data : 4; int rw : 1; int cd : 1; int enable : 1; int reset : 1; };
struct port_b_layout port_b;
#byte port_b = 6
struct port_b_layout const INIT_1 = {0, 1,1,1,1};
struct port_b_layout const INIT_2 = {3, 1,1,1,0};
struct port_b_layout const INIT_3 = {0, 0,0,0,0};
struct port_b_layout const FOR_SEND = {0,0,0,0,0}; // All outputs
struct port_b_layout const FOR_READ = {15,0,0,0,0}; // Data is an input

main() {
        int x;
        set_tris_b((int)FOR_SEND);
        port_b = INIT_1;
        delay_us(25);
        port_b = INIT_2;
        port_b = INIT_3;
        set_tris_b((int)FOR_READ);
        port_b.rw=0;
        port_b.cd=1;
        port_b.enable=0;
```

```
        x = port_b.data;
        port_b.enable=0
    }
```

- ***Định nghĩa kiểu***:

Cho phép định nghĩa thêm các kiểu dữ liệu mới từ các kiểu dữ liệu chuẩn của PIC C compiler. Việc định nghĩa thêm kiểu dữ liệu theo quy định như sau:

typedef tên kiểu dữ liệu chuẩn tên kiểu dữ liệu mới

Ví dụ:

```
typedef  int  byte;  // Định nghĩa kiểu dữ liệu có tên byte
typedef  short  bit; // Định nghĩa kiểu dữ liệu có tên bit
bit e,f;  // Khai báo biến kiểu bit
byte k = 5;   // khai báo biến kiểu byte
byte const   WEEKS = 52; // khai báo hằng kiểu byte
byte const FACTORS [4] ={8, 16, 64, 128}; // khai báo mảng tĩnh kiểu byte
```

c. ***Định nghĩa các hàm***:

Các hàm được định nghĩa gồm các phát biểu để thực hiện các giải thuật phục vụ cho dự án. Cấu trúc của hàm như sau:

```
Từ khoá của hàm    tên hàm  (các biến mà hàm sử dụng)
Khai báo các biến cần thiết và các kiểu dữ liệu cho hàm
{
        Các phát biểu trong hàm
}
```

+ Từ khoá của hàm gồm các thành phần: void, #separate , #inline, #int_…hoặc có thể bỏ trống.
+ Tên hàm được đặt tùy ý
+ Các biến sử dụng trong hàm sẽ được phân cách nhau bởi dấu ‘,’. Nếu không sử dụng biết trong hàm thì các biến trong hàm bỏ trống.

Ví dụ:

```
void lcd_putc(char c )   // Định nghĩa hàm
{
        ...
}
        lcd_putc ("Hi There.");   // gọi hàm
```

d. ***Các phát biểu điều kiện và vòng lặp***:

+ *Phát biểu điều kiện if - else*:

Phát biểu điều kiện if – else được dùng để rẽ nhánh chương trình, phát biểu if – else có dạng như sau:

```
If (điều kiện)
{
        Các lệnh trong chương trình
}
else
{
        Các lệnh trong chương trình
}
```

Cắp từ khóa {} được dùng nếu các lệnh trong chương trình gồm nhiều lệnh. Trong trường hợp chỉ có một lệnh thì không cần dùng cắp từ khoá {}.

Ví dụ:

```
if (x==25)
```

```
            x=1;
        else
            x=x+1;
```

+ *Vòng lặp WHILE*:

Vòng lặp while được dùng để lặp chương trình. Cấu trúc của vòng lặp while như sau:

```
        while   (biểu thức điều kiện)
        {
            Các lệnh trong chương trình
        }
```

Hoạt động của vòng lặp while là sẽ thực hiện các lệnh trong cặp từ khoá {} khi mà biểu thức điều kiện là đúng.

Ví dụ:

```
        while(get_rtcc()!=0)
        {
            putc('n');
            getc();
            …
        }
```

+ *Vòng lặp do – while*:

Vòng lặp do – while được sử dụng tương tự như vòng lặp while tuy nhiên, vòng lặp while kiểm tra điều kiện trước khi thực hiện các lệnh còn vòng lặp do – while sẽ kiểm tra điều kiện sau khi thực hiện các lệnh. Cấu trúc của vòng lặp do – while như sau:

```
        do
        {
            Các lệnh trong chương trình
        }
        while (biểu thức điều kiện);
```

Ví dụ:

```
        do
        {
            putc(getc());
            get_rtcc();
            …
        }
        while(c!=0);
```

+ *Vòng lặp for*:

Vòng lặp for được dùng để lặp lại chương trình theo một biến đếm. Cấu trúc của vòng lặp for như sau:

```
        For (biểu thức 1; biểu thức 2; biểu thức 3)
        {
            Các lệnh trong chương trình
        }
```

Trong đó biểu thức 1 là giá trị khởi đầu của biến đếm, biểu thức 2 là giá trị cuối của biến đếm, biểu thức 3 là biểu thức đếm.

Ví dụ:

```
        For (I=1;I<=100; I++)
            {
                b = I +100;
```

```
                printf('%u\r\n', I);
        }
```

+ *Phát biểu SWITCH – CASE*:

Phát biểu switch – case được dùng để rẽ nhánh chương trình. Cấu trúc của phát biểu này như sau:

Switch (biểu thứcđiều kiện)
{
case điều kiện 1:
Các lệnh trong chương trình;
Break;
Case điều kiện 2:
Các lệnh trong chương trình;
Break;
...
default:
Các lệnh trong chương trình;
Break;
}

Ví dụ:

```
Switch (cmd)
{
        case 0:
                {
                printf("cmd 0");
                break;
                }
        case 1:
                {
                printf("cmd 1");
                break;
                }
        default:
                {
                 printf("bad cmd");
                break;
                }
}
```

+ *Phát biểu return*:

Phát biểu return được dùng để trả về trị của hàm. Phát biể return như sau:

Return (giá trị);

+ *Phát biểu goto*:

Phát biểu goto được dùng để nhảy tới một nhãn trong chương trình. Phát biểu goto có dạng như sau:

goto nhãn;

+ *Phát biểu break*:

Phát biểu break dùng để kết thúc một nhóm lệnh trong cặp từ khóa {}. Phát biểu break như sau:

break;

+ *Phát biểu continue*:

Phát biểu continue dùng để tiếp tục thực hiện một nhóm lệnh trong cặp từ khóa {}. Phát biểu continue như sau:

continue;

e. ***Các phép toán trong PIC C compiler***:

Trogn PIC C compiler sử dụng các ký hiệu như bảng sau để đặc trưng cho các phép toán.

Ký hiệu	Phép toán
Các phép toán số học	
+	Thực hiện phép tính cộng
+=	Thực hiện phép cộng dồn. Ví dụ: x+=y tương tự như x = x + y
-	Thực hiện phép tính trừ
-=	Thực hiện phép trừ dồn. Ví dụ:x-=y tương tự như x = x – y
*	Thực hiện phép tính nhân
*=	Thực hiện phép nhân dồn. Ví dụ:x*=y tương tự như x = x * y
/	Thực hiện phép tính chia
/=	Thực hiện phép chia dồn. Ví dụ:x/=y tương tự như x = x / y
++	Thực hiện việc tăng một giá trị. Ví dụ: i++ tương tự như i = i + 1
--	Thực hiện việc giảm một giá trị. Ví dụ: i-- tương tự như i = i – 1
=	Phép gán. Ví dụ: x = 1
%	Lấy trị tuyệt đối
Các phép toán so sánh	
==	Phép so sánh bằng
!=	Phép so sánh khác
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
Các phép toán nhị phân	
&&	Phép toán AND
||	Phép toán OR
>>	Dịch phải
<<	Dịch trái
!	Đảo bit
~	Lấy bù 1
&	AND từng bit
|	OR từng bit
sizeof	Lấy kích thước của byte

II. **CÁC HÀM THƯ VIỆN CỦA PIC C COMPILER**:

PIC C compiler xây dựng sẵn 108 hàm chia thành 11 nhóm và được chứa trong các file.h. Khi trong chương trình muốn gọi các hàm này thì ở đầu chương trình phải khai báo tiền xử lý #use hoặc #include để trình biên dịch tìm các hàm tương ứng để đưa vào dự an. Nhóm hàm như bảng sau:

Các hàm phục vụ cổng RS232		
getc() putc() fgetc() gets() puts() fgets()	fputc() fputs() printf() kbhit() fprintf() set_uart_speed()	perror() assert() getchar() putchar() setup_uart()
Các hàm phục vụ cổng giao tiếp dữ liệu nối tiếp đồng bộ SPI		

setup_spi() spi_read()	spi_write() spi_data_is_in()	
Nhóm các hàm phụ vụ cổng vào ra số		
output_low() output_high() output_float() output_bit()	input() output_X() output_toggle() input_state()	input_X() port_b_pullups() set_tris_X()
Các hàm toán học chuẩn của C		
abs() acos() asin() atan() ceil() cos() exp() floor() labs()	sinh() log() log10() pow() sin() cosh() tanh() fabs() fmod()	atan2() frexp() ldexp() modf() sqrt() tan() div() ldiv()
Hàm phục vụ điện áp tham chiếu cho		
setup_vref()		
Hàm phục vụ chuyển đổi ADC		
setup_adc_ports() setup_adc()	set_adc_channel() read_adc()	
Các hàm phục vụ timer		
setup_timer_X() set_timer_X()	get_timer_X() setup_counters()	setup_wdt() restart_wdt()
Các hàm quản lý bộ nhớ		
memset() memcpy() offsetof() offsetofbit()	malloc() calloc() free() realloc()	memmove() memcmp() memchr()
Các hàm phục vụ CCP		
setup_ccpX() set_pwmX_duty()	setup_power_pwm() setup_power_pwm_pins()	set_power_pwmx_duty() set_power_pwm_override()
Các hàm ký tự chuẩn của C		
atoi() atoi32() atol() atof() tolower() toupper() isalnum() isalpha() isamoung() isdigit() islower() isspace() isupper()	isxdigit() strlen() strcpy() strncpy() strcmp() stricmp() strncmp() strcat() strstr() strchr() strrchr() isgraph() iscntrl()	strtok() strspn() strcspn() strpbrk() strlwr() sprintf() isprint() strtod() strtol() strtoul() strncat() strcoll(), strxfrm()
Các hàm quản lý bộ nhớn EEPROM		

read_eeprom() write_eeprom() read_program_eeprom() write_program_eeprom()	read_calibration() write_program_memory() read_program_memory()	write_external_memory() erase_program_memory() setup_external_memory()
Các hàm đặc biệt của C		
rand()	srand()	
Các hàm delay		
delay_us()	delay_ms()	delay_cycles()
Hàm phục vụ analog compare		
setup_comparator()		

Chi tiết của một số hàm thông dụng trong các nhóm và chỉ định tiền xử lý tương ứng được trình bày trong các mục sau:

1. **Các Hàm Toán Học Chuẩn Của C**:

a. ***ABS()***:

Hàm này được dùng để tính giá trị tuyệt đối của một số.

+ Cú pháp : value = abs(x)
+ Tham số : x là một số có dấu 8, 16, hoặc 32 bit với kiểu dữ liệu integer hay float.
+ Trị trả về : cùng kiểu với tham số truyền.
+ Yêu cầu : cần khai báo #include <stdlib.h>

Ví dụ:

```
signed int target,actual;
...
error = abs(target-actual);
```

b. ***SIN(), COS(),TAN(), ASIN(), ACOS(), ATAN()***:

Các hàm lượng giác thuận và nghịch dùng để tính sin, cos, tg của một góc và tính lượng giác ngược arcsin, arccos, arctg của một số.

+ Cú pháp: val = sin (rad)
  val = cos (rad)
  val = tan (rad)
  rad = asin (val)
  rad = acos (val)
  rad = atan (val)
+ Tham số : rad là một số thực biểu thị giá trị góc tính bằng radian(-2π đến 2π).
  val là một số thực có giá trị từ -1.0 đến 1.0
+ Trị trả về : rad là một số thực biểu thị giá trị góc tính bằng radian(-2π đến 2π).
  val là một số thực có giá trị từ -1.0 đến 1.0
+ Yêu cầu : khai báo #include<math.h >.

Ví dụ:

```
float phase;   // Phát sóng sin
for(phase=0; phase<2*3.141596; phase+=0.01)
set_analog_voltage( sin(phase)+1 );
```

c. ***CEIL()***:

Hàm này được dùng để làm tròn số theo hướng tăng.

+ Cú pháp : result = ceil (value)
+ Tham số : value là một số thực
+ Trị trả về : số thực
+ Yêu cầu : #include <math.h>

Ví dụ:

x = ceil(12.60) khi đó x = 13.

d. ***EXP()***:

Hàm này được dùng để tích tính hàm ex. Ví dụ exp(1) is 2.7182818.

+ Cú pháp : result = exp (value)
+ Tham số : value là một số thực
+ Trị trả về : một số thực
+ Yêu cầu : khai báo #include <math.h>

Ví dụ:

```
// Tính x luỹ thừa y
x_power_y = exp( y * log(x) );
```

e. ***FLOOR()***:

Hàm này được dùng để làm tròn theo hướng giảm. Ví dụ Floor(12.67) is 12.00.

+ Cú pháp : result = floor (value)
+ Tham số : value là một số thực
+ Trị trả về : số thực
+ Yêu cầu : khai báo #include<math.h>.

Ví dụ:

```
// Tìm phần lẻ ô1
frac = value - floor(value);
```

f. ***LABS()***:

Hàm này được dùng để xác định giá trị tuyệt đối của một số long integer.

+ Cú pháp : result = labs (value)
+ Tham số : value là một số 16 bit long int
+ Trị trả về : một 16 bit signed long int
+ Yêu cầu : khai báo #include <stdlib.h>.

Ví dụ:

```
if( labs( target_value - actual_value ) > 500 )
        printf("Error is over 500 points\r\n");
```

g. ***LOG()***:

Hàm này được dùng để tính logarit cơ số tự nhiên ln.

+ Cú pháp : result = log (value)
+ Tham số : value là tham số kiểu float
+ Trị trả về : kiểu float
+ Yêu cầu : khai báo #include <math.h>.

Ví dụ:

```
lnx = log(x);
```

h. ***LOG10()***:

Hàm này được dùng để tính logarit cơ số 10 log.

+ Cú pháp : result = log10 (value)
+ Tham số : value là tham số kiểu float
+ Trị trả về : kiểu float
+ Yêu cầu : khai báo #include <math.h>.

Ví dụ:

```
logx = log10(x);
```

i. ***POW()***:

Hàm này được dùng để tính luỹ thừa n của một số

+ Cú pháp : f = pow (x,n)
+ Tham số : x, n là số thực

+ Trị trả về : số thực
+ Yêu cầu : #include <math.h>

Ví dụ:

```
F = pow(2,3); // Tính 2^3 , F có giá trị bằng 8
```

j. ***SQRT()***:

Hàm này được dùng để tính căn bậc 2 của một số không âm.

+ Cú pháp : result = sqrt (value)
+ Tham số : value là kiểu float
+ Trị trả về : kiểu float
+ Yêu cầu : #include <math.h>

Ví dụ:

```
distance = sqrt( sqr(x1-x2) + sqr(y1-y2) );
```

2. **Nhóm Các Hàm Ký Tự Chuẩn Của C**:

a. ***ATOI(), ATOL(), ATOI32()***:

Hàm này được dùng để chuyển một chuỗi thành một số nguyên . Cho phép tham số gồm decimal và hexadecimal.

+ Cú pháp : ivalue = atoi(string)
  hoặc
  lvalue = atol(string)
  hoặc
  i32value = atoi32(string)
+ Tham số : string là một chuỗi ký tự
+ Trị trả về : ivalue là một số nguyên 8 bit.
  lvalue là một số nguyên kiểu int 16 bit.
  i32value là kiểu số nguyên 32 bit.
+ Yêu cầu : #include <stdlib.h>

Ví dụ:

```
char string[10];
int x;
strcpy(string,"123");
x = atoi(string);               // x sẽ là 123
```

b. ***ATOF()***:

Hàm này được dùng để chuyển một chuỗi thành một số có kiểu dấu chấm phẩy động.

+ Cú pháp : result = atof(string)
+ Tham số : string là một chuỗi ký tự.
+ Trị trả về : một số thực dấu chấm phẩy động 32 bit.
+ Yêu cầu : #include <stdlib.h>

Ví dụ:

```
char string [10];
float x;
strcpy (string, "123.456");
x = atof(string);               // x sẽ là 123.45a2
```

c. ***TOLOWER(), TOUPPER()***:

Các hàm này được sử dụng để thay đổi case của các ký tự. TOLOWER(X) sẽ chuyển 'a'..'z' của X thành 'A'..'Z'. TOUPPER(X) sẽ thực hiện chức năng ngược lại, tức chuyển 'A'..'Z' của X thành 'a'..'z'

+ Cú pháp : result = tolower (cvalue)
  result = toupper (cvalue)
+ Tham số : cvalue là một ký tự

+ Trị trả về : một ký tự 8 bit
+ Yêu cầu : không

Ví dụ:

```
switch( toupper(getc()) )
{
case 'R' : read_cmd();  break;
case 'W' : write_cmd(); break;
case 'Q' : done=TRUE;   break;
}
```

d. ***ISALNUM(), ISALPHA(), ISDIGIT(), ILOWER(), ISSPACE(), ISUPPER(), ISXDIGIT()***:

Các hàm này được sử dụng để test ký tự nằm trong khoảng cho phép theo sau:

```
isalnum(x)    X is 0..9, 'A'..'Z',  or 'a'..'z'
isalpha(x)    X is 'A'..'Z' or 'a'..'z'
isdigit(x)    X is '0'..'9'
islower(x)    X is 'a'..'z'
isupper(x)    X is 'A'..'Z
isspace(x)    X is a space
isxdigit(x)   X is '0'..'9', 'A'..'F', or 'a'..'f'
```

+ Cú pháp:
```
value = isalnum(datac)
value = isalpha(datac)
value = isdigit(datac)
value = islower(datac)
value = isspace(datac)
value = isupper(datac)
value = isxdigit(datac)
```
+ Tham số :datac là một ký tự 8 bit
+ Trị trả về :0 (or FALSE) nếu datac không thuộc một số tiêu chuẩn quy định, 1 (or TRUE) nếu ngược lại.
+ Yêu cầu :ctype.h

Ví dụ:

```
char id[20];
  ...
if(isalpha(id[0]))
{
valid_id=TRUE;
for(i=1;i<strlen(id);i++)
   valid_id=valid_id&& isalnum(id[i]);
}
else
valid_id=FALSE;
```

e. ***ISAMOUNG()***:

Hàm này được dùng để trả về TRUE nếu ký tự là một ký tự trong chuỗi hằng (tìm ký tự trong một chuỗi).

+ Cú pháp : result = isamoung (value, cstring)
+ Tham số : value là kiểu ký tự
  cstring là chuỗi hằng
+ Trị trả về : 0 (or FALSE) nếu value không phải là cstring
  1 (or TRUE) nếu value là cstring
+ Yêu cầu : không

Ví dụ:

```
char x;
...
if( isamoung( x, "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ") )
printf("The character is valid");
Example Files:      ctype.h
```

3. **Các Hàm Xử Lý Chuỗi Chẩn**:

a. ***STRCAT()***:

Hàm này được dùng để ghép 2 chuỗi.

+ Cú pháp: ptr=strcat(s1, s2)
+ Tham số: s1 and s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng. Cần lưu ý rằng s1 and s2 không phải là một chuỗi hằng (ví dụ "hi").
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

Ví dụ:

```
char string1[10], string2[10];
strcpy(string1,"hi ");      // string1 là “hi”
strcpy(string2,"there");  // string2 là “there”
strcat(string1,string2);   // string1 là “hi there”
printf("Length is %u\r\n", strlen(string1) ); // Gửi qua RS 232 giá trị là 8
```

b. ***STRCHR()***:

Hàm này được dùng để tìm ký tự trong chuỗi và trả về vị trí của ký tự trong chuỗi.

+ Cú pháp: ptr=strchr(s1, c) // Tìm ký tự c trong chuỗi s1, trả về vị trí của c trong s1
+ Tham số: s1 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng, c là ký tự kiểu char.
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

c. ***STRRCHR()***:

Hàm này có chức năng và hoạt động tương tự như hàm strchr() nhưng tìm theo chiều ngược lại.

+ Cú pháp: ptr=strrchr(s1, c) // Tìm ký tự c trong chuỗi s1, trả về vị trí của c trong s1
+ Tham số: s1 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng, c là ký tự kiểu char).
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

d. ***STRCMP()***:

Hàm này được dùng để so sánh 2 chuỗi.

+ Cú pháp: cresult =strcmp(s1, s2) // So sánh chuỗi s1 và s2
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: result là: -1 (less than), 0 (equal) or 1 (greater than)
+ Yêu cầu: #include <string.h>

e. ***STRNCMP()***:

Hàm này được dùng để so sánh chuỗi s1 với s2 (n bytes).

+ Cú pháp: iresult=strncmp(s1, s2, n) // so sánh chuỗi s1 với s2 (n bytes)
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng), n là số byte.
+ Trị trả về: iresult là số nguyên 8 bit kiểu int
+ Yêu cầu: #include <string.h>

f. ***STRICMP()***:

Hàm này được dùng để so sánh chuỗi s1 với s2 (không biết s1 và s2).

+ Cú pháp: iresult=stricmp(s1, s2) // so sánh trong trường hợp không biết s1 và s2
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: iresult là số nguyên 8 bit kiểu int
+ Yêu cầu: #include <string.h>

g. ***STRNCPY()***:

Hàm này được dùng để copy n ký tự từ s2 và gán vào s1.
+ Cú pháp: ptr=strncpy(s1, s2, n)
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng), n là số ký tự .
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

h. ***STRCSPN()***:

Hàm này được dùng để đếm số ký tự đầu của chuỗi s1 không nằm trong chuỗi s2..
+ Cú pháp: iresult=strcspn(s1, s2)
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: iresult là số nguyên 8 bit kiểu int
+ Yêu cầu: #include <string.h>

i. ***STRSPN()***:

Hàm này được dùng để đếm số ký tự đầu của chuỗi s1 nằm trong chuỗi s2..
+ Cú pháp: iresult=strspn(s1, s2)
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: iresult là số nguyên 8 bit kiểu int
+ Yêu cầu: #include <string.h>

j. ***STRLEN()***:

Hàm này được dùng để tính chiều dài của chuỗi s1.
+ Cú pháp: iresult=strlen(s1)
+ Tham số: s1 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: iresult là số nguyên 8 bit kiểu int
+ Yêu cầu: #include <string.h>

k. ***STRLWR()***:

Hàm này được dùng để chuyển các ký tự trong chuỗi s1 thành chữ thường.
+ Cú pháp: ptr =strlwr(s1)
+ Tham số: s1 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

l. ***STRPBRK()***:

Hàm này được dùng để tìm vị trí của ký tự trong chuỗi s1 và là ký tự đầu trong chuỗi s2.
+ Cú pháp: ptr =strpbrk(s1, s2)
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

m. ***STRSTR()***:

Hàm này được dùng để tìm vị trí của chuỗi s2 trong chuỗi s1.
+ Cú pháp: ptr =strstr(s1, s2)
+ Tham số: s1, s2 là pointer trỏ tới vùng nhớ của mảng ký tự (hoặc tên một mảng).
+ Trị trả về: ptr là biến con trỏ của s1
+ Yêu cầu: #include <string.h>

n. ***STRCPY()***:

Hàm này được dùng để Copy một một hằng hoặc một chuỗi từ src vào một chuỗi dest.

+ Cú pháp: strcpy (dest, src)
+ Tham số: dest là pointer trỏ đến vùng nhớ của dãy ký tự.
src có thể là chuỗi hằng hoặc là một pointer
+ Trị trả về: không
+ Yêu cầu: không
Ví dụ:

```
char string[10], string2[10];
…
strcpy (string, "Hi There");
strcpy(string2,string);
```

o. ***STRTOK()***:

Hàm này được dùng để Tìm mã kế tiếp ở trong phạm vi s1 bằng ký tự được tách ra từ chuỗi s2 và trả con trỏ về lại chỗ đó. Lần gọi đầu tiên bắt đầu tại vị trí đầu của s1 và tìm kiếm cho đến khi thấy được ký tự NOT chứa trong s2, và trả vềký tự NULL nếu không tìm thấy. Nếu không tìm thấy, nó bắt đầu với mã đầu tiên (trả lại giá trị cũ). Lệnh này (tiếp tục tìm kiếm) lúc đó tiếp tục tìm kiếm ký tự đó chứa trong s2. Nếu không tìm thấy, từ mã hiện tại cho đến mã kết thúc của s1,tiếptục tìm kiếm cho đến hết mã vạch sẽ trả về giá trị NULL. Nếu 1 được tìm thấy nó sẽ được viết đè lên bởi \0, để kết thúc mã. Lệnh này sẽ lưu lại biến con trỏ sau một ký tự từ lần kiếi tiếp theo sẽ được bắt đầu. Mỗi lần gọi tiếp theo với 0 là đối số, bắt đầu tìm kiếm với giá trị con trỏ đã được lưu trước đó.

+Cú pháp : ptr = strtok(s1, s2)
+ Tham số : s1, s2 là biến con trỏ chỉ đến ký tự của mảng (hoặc tên của mảng). Chú ý rằng s1, s2 có thể không phải là hằng số (ở mức cao). S1 có thể bằng 0 để báo là hành động vẫn được thực hiện.
+ Trị trả về: ptr trỏ tới ký tự cần trong s1 hoặc nó bằng 0.
+ Lợi ích: Tất cả các thiết bị.
+ Yêu cầu: #include <string.h>
Ví dụ:

```
char string[30], term[3], *ptr;
strcpy(string,"one,two,three;");
strcpy(term,",;");
ptr = strtok(string, term);
while(ptr!=0)
    {
        puts(ptr);
        ptr = strtok(0, term);
    }
        // Prints:
          one
          two
          three
```

4. **Nhóm Hàm Quản Lý Bộ Nhớ Của Của C**:

a. ***MEMSET()***:

Hàm này được dùng để copy n byte của value vào bộ nhớ đích.
+ Cú pháp: memset (destination, value, n)
+ Tham số: destination là một con trỏ; value, n số nguyên 8bit.
+ Trị trả về: không
+ Yêu cầu : không
Ví dụ:

```
memset(arrayA, 0, sizeof(arrayA));
memset(arrayB, '?', sizeof(arrayB));
memset(&structA, 0xFF, sizeof (structA));
```

b. ***MEMCPY(), MEMMOVE()***:

Hàm này được dùng để copy n byte từ source đến destination trong RAM.

+ Cú pháp: memcpy (*destination*, *source*, *n*)
memmove(*destination*, *source*, *n*)
+ Tham số: destination là một biến con trỏ trong bộ nhớ, source là biến con trỏ trong vùng nhơ', n là số byte cần chuyển
+ Trị trả về: không
+ Yêu cầu: không

Ví dụ:

```
memcpy(&structA,&structB,sizeof (structA));
memcpy(arrayA,arrayB,sizeof (arrayA));
memcpy(&structA, &databyte, 1);
```

5. **Lệnh ANALOG COMPARE (SETUP_COMPARATOR())**:

Hàm này được dùng để khởi tạo khối analog comparator. Hằng số trên có 4 giá trị của đầu vào C1-, C1+, C2-, C2+

+ Cú pháp: setup_comparator (mode)
+ Tham số: mode là hằng số. Các hằng số hợp lệ được chứa trong các file có đuôi .h và có thể như sau:

```
A0_A3_A1_A2
A0_A2_A1_A2
NC_NC_A1_A2
NC_NC_NC_NC
A0_VR_A1_VR
A3_VR_A2_VR
A0_A2_A1_A2_OUT_ON_A3_A4
A3_A2_A1_A2
```

+ Trị trả về: không định nghĩa
+ Lợi ích: lệnh này chỉ dùng cho các thiết bị có analog comparator.
+ Yêu cầu: Hằng số được định nghĩa trong file có tên devices.h

Ví dụ:

```
// Sets up two independent comparators (C1 and C2),
// C1 uses A0 and A3 as inputs (- and +), and C2
// uses A1 and A2 as inputs
setup_comparator(A0_A3_A1_A2);
```

6. **Lệnh SETUP_VREF()**:

Hàm này được dùng để thiết lập điện áp tham chiếu cho tín hiệu analog và/hoặc cho xuất ra tín hiệu qua chân A2.

+ Cú pháp: setup_vref (mode | value)
+ Tham số: mode có thể là 1 trong các hằng số sau:

```
FALSE           (off)
VREF_LOW        for VDD*VALUE/24
VREF_HIGH       for VDD*VALUE/32 + VDD/4
Giá trị là số nguyên 0-15 bit
```

+ Trị trả về : Không xác định
+ Lợi ích : lệnh này chỉ sử dụng với thiết bị có phần VREF.
+ Yêu cầu : Hằng số được định nghĩa trong file có tên devices.h

Ví dụ:

```
setup_vref (VREF_HIGH | 6); // At VDD=5, the voltage is 2.19V
```

7. **Nhóm Hàm Quản Lý ROM Nội**:

a. ***READ_EEPROM()***:

Hàm này được dùng để đọc byte từ địa chỉ dữ liệu EEPROM xác định. Địa chỉ bắt đầu ở 0 và phạm vi phụ thuộc vào chức năng.

+ Cú pháp :value = read_eeprom (address)
+ Tham số :Địa chỉ là một số nguyên 8 bit.
+ Trị trả về :là một số nguyên 8 bit.
+ Lợi ích :Lệnh này chỉ có tác dụng với các thiết bị có gắn phần EEPROMS.
+ Đòi hỏi :Không.

Ví dụ:

```
#define LAST_VOLUME  10
volume = read_EEPROM (LAST_VOLUME);
```

b. ***WRITE_EEPROM()***:

Hàm này được dùng để ghi byte vào địa chỉ dữ liệu EEPROM xác định. Lệnh này có thể chiếm vài phần nghìn giây của chu kỳ máy. Lệnh này chỉ làm việc với các thiết bị có cài sẵn chương trình EEPROM bên trong nó. Lệnh này cũng có tác dụng với các thiết bị có phần EEPROM ngoài hoặc với bộ phận EEPROM riêng biệt (line the 12CE671), xem thêm EX_EXTEE.c with CE51X.c, CE61X.c or CE67X.c.

+ Cú pháp :write_eeprom (address, value)
+ Tham số :địa chỉ là một số nguyên 8 bit, phạm vi phụ thuộc vào thiết bị, giá trị là một số nguyên 8 bit.
+ Trị trả về :Không rõ.
+ Lợi ích :Lệnh này chỉ thực hiện với các thiết bị có phần mềm hỗ trợ trên chip.
+ Yêu cầu :Không.

Ví dụ:

```
#define LAST_VOLUME  10    // Location in EEPROM
volume++;
write_eeprom(LAST_VOLUME,volume);
```

c. ***READ_PROGRAM_EEPROM()***:

Hàm này được dùng để đọc từ những vùng chương trình EEPROM riêng biệt.

+ Cú pháp : write_program_eeprom (address, data)
+ Tham số :địa chỉ là 16 bit cho vùng PCM và là 32 bit cho vùng PCH, dữ liệu là 16 bit cho vùng PCM và 8 bit cho vùng PCH.
+ Trị trả về :Tuỳ dữ liệu trong EEPROM
+ Lợi ích : Chỉ có các thiết bị cho phép đọc từ bộ nhớ chương trình.
+ Yêu cầu :Không.

Ví dụ:

```
write_program_eeprom(0,0x2800);
//disables program
```

d. ***READ_CALIBRATION()***:

Hàm này có thể đọc được kích cỡ của tham số n vào khoảng 14000, đó là độ lớn của bộ nhớ.

+ Cú pháp : value = read_calibration (n)
+ Tham số : n là kích thước bộ nhớ, và bộ nhớ bắt đầu là 0.
+ Trị trả về : Là byte 8 bit.
+ Lợi ích : Lệnh này chỉ dùng cho PIC 14000.
+ Yêu cầu : Không.

Ví dụ: fin = read_calibration(16);

8. **Nhóm Hàm Vào/Ra Số**:

a. ***OUTPUT_LOW()***:

Hàm này dùng để reset các chân của MCU về mức logic 0

+ Cú pháp: output_low (pin)

+ Tham số: các chân (pin) phải được định nghĩa trong file tiêu đề devices .h. Giá trị hoạt động là bit địa chỉ. Ví dụ, cần port A (byte 5) bit 3 tương ứng với giá trị 5*8+3 or 43 thì sẽ định nghĩa như sau: #define PIN_A3 43

+ Trị trả về: không

+ Yêu cầu: Chân IC phải được định nghĩa trong tập tin tiêu đề devices .h

b. ***OUTPUT_HIGH()***:

Hàm này dùng để set các chân của MCU lên mức logic 1.

+ Cú pháp: output_high(pin)

+ Tham số: các chân (pin) phải được định nghĩa trong file tiêu đề devices .h. Giá trị hoạt động là bit địa chỉ. Ví dụ, cần port A (byte 5) bit 3 tương ứng với giá trị 5*8+3 or 43 thì sẽ định nghĩa như sau: #define PIN_A3 43

+ Trị trả về: không

+ Yêu cầu: Chân IC phải được định nghĩa trong tập tin tiêu đề devices .h

Ví dụ:

```
#include <16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)  // Jumpers: 8 to 11, 7 to 1
void main()
{
        printf("Press any key to begin\n\r");
        getc();
        printf("1 khz signal activated\n\r");
        while (TRUE)
                {
                output_high(PIN_B0);
                delay_us(500);
                output_low(PIN_B0);
                delay_us(500);
                }

}
```

c. ***OUTPUT_FLOAT()***:

Hàm này dùng để Set mode input cho các chân. Hàm này cho phép pin ở mức cao với mục đích như là một cực collector thu hở.

+ Cú pháp: output_float(pin)

+ Tham số: các chân (pin) phải được định nghĩa trong file tiêu đề devices .h. Giá trị hoạt động là bit địa chỉ. Ví dụ, cần port A (byte 5) bit 3 tương ứng với giá trị 5*8+3 or 43 thì sẽ định nghĩa như sau: #define PIN_A3 43

+ Trị trả về: không

+ Yêu cầu: Chân IC phải được định nghĩa trong tập tin tiêu đề devices .h

Ví dụ:

```
if( (data & 0x80)==0 )
    output_low(pin_A0);
else
    output_float(pin_A0);
```

d. ***OUTPUT_BIT()***:

Hàm này được dùng để xuất giá trị 0 hoặc 1 trên các ngõ ra.

+ Cú pháp: output_bit (pin, value)
+ Tham số: tương tự output_low(pin) và output_high(pin)
+ Trị trả về: không
+ Yêu cầu: chân IC phải được định nghĩa trên các tập tin tiêu đề devices .h

Ví dụ:

```
output_bit( PIN_B0, 0);                  // Giống như output_low(pin_B0);
output_bit( PIN_B0,input( PIN_B1 ) );    // Đặt chân B0 giống chân  B1
output_bit( PIN_B0,shift_left(&data,1,input(PIN_B1)));
```

e. ***INPUT()***:

Hàm này trả về trạng thái các chân. Phương thức I/O phụ thuộc vào USE *_IO điều khiển trước đó.

+ Cú pháp: value = input (pin)
+ Tham số: các chân của device phải được khai báo trong tập tin tiêu đề device.h và sẽ được đọc sau lệnh input(). Giá trị là các bit địa chỉ.
+ Trị trả về: 0 (or FALSE) nếu giá trị trên chân IC là 0
  1 (or TRUE) nếu giá trị trên chân IC là 1

Ví dụ:

```
while ( !input(PIN_B1) );   // waits for B1 to go high
if( input(PIN_A0) )
printf("A0 is now high\r\n");
```

f. ***OUTPUT_X()***:

Hàm này được dùng để xuất một byte ra cổng. Cổng điều khiển phụ thuộc vào khai báo điều khiển #USE *_IO trước đó

+ Cú pháp : output_a (value)
  output_b (value)
  output_c (value)
  output_d (value)
  output_e (value)
+ Tham số : value là một số nguyên 8 bit
+ Trị trả về : không
+ Yêu cầu : không

Ví dụ:

```
OUTPUT_B(0xf0);  // Xuất 11110000b ra port B
```

g. ***INPUT_X()***:

Hàm này được dùng để nhập một byte từ Port. Port điều khiển phụ thuộc vào khai báo điều khiển #USE*_IO trước đó.

+ Cú pháp: value = input_a()
  value = input_b()
  value = input_c()
  value = input_d()
  value = input_e()

+ Tham số: không
+ Trị trả về: số nguyên 8 bit.
+ Yêu cầu : không
Ví dụ:

```
data = input_b();
```

h. ***PORT_B_PULL-UPS()***:
Hàm này được dùng để Sets cổng vào B pullup. TRUE sẽ hoạt động, và FALSE sẽ ngừng.
+ Cú pháp: port_b_pull-ups (value)
+ Tham số: value là biến bool logic (TRUE hoặc FALSE)
+ Trị trả về: không
Lưu ý: Chỉ có trên các thiết bị 14 và 16 bit (PCM and PCH). (Note: use SETUP_COUNTERS trong PCB phần).
Ví dụ:

```
port_b_pullups(FALSE);
```

i. ***SET_TRIS_X()***:
Những hàm này định nghĩa chân I/O cho một port. Điều này chỉ thực hiện được khi sử dụng FAST_IO và khi cổng xuất nhập được truy cập. Mỗi bit đại diện cho một chân. số 1 chỉ chân nhập và 0 chỉ chân xuất
+ Cú pháp: set_tris_a (value)
set_tris_b (value)
set_tris_c (value)
set_tris_d (value)
set_tris_e (value)
+ Tham số: value là số nguyên 8 bit với mỗi bit đại diện cho một cổng xuất nhập.
+ Yêu cầu: không
Ví dụ:

```
SET_TRIS_B( 0x0F );      // B7,B6,B5,B4  là xuất,  B3,B2,B1,B0 là nhập
```

9. **Nhóm Lệnh Trên Bit, Byte**:
a. ***SHIFT_RIGHT(), SHIFT_LEFT()***:
Các hàm này được dùng để dịch phải (trái) một bit vào một mảng hay một cấu trúc. Địa chỉ phải là một định nghĩa mảng hoặc địa chỉ trỏ tới cấu trúc (such as &data). Bit 0 của byte thấp trong RAM sẽ được xử lý bàng LSB.
+ Cú pháp: shift_right (address, bytes, value)
+ Tham số: address địa chỉ trỏ tới vùng nhớ, bytes là số byte thao tác, value là 0 hoặc 1 được dịch vào.
+ Trị trả về: 0 or 1 đối với bit bị dịch ra
+ Yêu cầu : không
Ví dụ:

```
// Đọc 16 bit từ chân A1, mỗi khi chân A2 chuyển từ 0 sang 1 (cạnh lên)
struct {
        byte time;
        byte command : 4;
        byte source  : 4;
        } msg;
for(i=0; i<=16; ++i)
{
        while(!input(PIN_A2));
        shift_right(&msg,3,input(PIN_A1));
        while (input(PIN_A2)) ;
```

```
    }       // Dịch 8 bit ra chân A0, bắt đầu từ bit LSB.
    for(i=0;i<8;++i)
        output_bit(PIN_A0,shift_right(&data,1,0));
```

b. ***ROTATE_RIGHT()***:

Hàm này được dùng để quay phải một bit của một mảng hay cấu trúc. . Địa chỉ phải là một định nghĩa mảng hoặc địa chỉ trỏ tới cấu trúc (such as &data). Bit 0 của byte thấp trong RAM sẽ được xử lý bàng LSB.

+ Cú pháp: rotate_right (address, bytes)
+ Tham số: address địa chỉ vùng nhớ, bytes là số byte thao tác.
+ Trị trả về: không
+ Yêu cầu : không

Ví dụ:

```
    struct
    {
        int cell_1 : 4;
        int cell_2 : 4;
        int cell_3 : 4;
        int cell_4 : 4;
    } cells;
    rotate_right( &cells, 2);
    rotate_right( &cells, 2);
    rotate_right( &cells, 2);
    rotate_right( &cells, 2);      // cell_1->4, 2->1, 3->2 and 4-> 3
```

c. ***ROTATE_LEFT()***:

Hàm này được dùng để quay trái một bit.

+ Cú pháp: rotate_left (address, bytes)
+ Tham số: address là địa chỉ con trỏ trong vùng nhớ, byte là số byte thao tác
+ Trị trả về: không
+ Yêu cầu : không

Ví dụ:

```
    x = 0x86;
    rotate_left( &x, 1);            // x = 0x0d
```

d. ***BIT_CLEAR()***:

Hàm này được dùng để xoá bit của một biến (0-7, 0-15 or 0-31). Hàm này tương đương: var &= ~(1<<bit);

+ Cú pháp: bit_clear(var,bit)
+ Tham số: var có thể là biến hay giá trị 8,16 hoặc 32 bit, bit là một số 0-31 thể hiện số bit, 0 là LSB.
+ Trị trả về: không
+ Yêu cầu : None

Ví dụ:

```
    int x;
    x=5;
    bit_clear(x,2);         // x is now 1
    bit_clear(*11,7);       // A crude way to disable ints
```

e. ***BIT_SET()***:

Hàm này được dùng để Sets bit cho một biến. Hàm này tương đương: var |= (1<<bit);

+ Cú pháp: bit_set(var,bit)

Tham số: var là biến 8,16 or 32 bit, bit là một số từ 0-31 thể hiện số bit, 0 là bit có trọng số thấp nhất.

+ Trị trả về: không
+ Yêu cầu : không

Ví dụ:

```
int x;
x=5;
bit_set(x,3);  // x is now 13
bit_set(*6,1);          // A crude way to set pin B1 high
```

f. ***BIT_TEST()***:

Hàm này được dùng để test bit (0-7,0-15 or 0-31) trong một biến.

+ Cú pháp: value = bit_test (var,bit)

Tham số: var là biến 8,16 or 32 bit, bit là một số từ 0-31 thể hiện số bit, 0 là bit có trọng số thấp nhất.

+ Trị trả về: 0 or 1
+ Yêu cầu : không.

Ví dụ:

```
if( bit_test(x,3) || !bit_test (x,1) )
{
      //Nếu bit 3 là 1 hoặc bit 1 là 0
}
if(data!=0)
for(i=31;!bit_test(data,i);i--) ;
```

g. ***SWAP()***:

Hàm này được dùng để tráo đổi bit thấp thành cao của một byte. Hàm này tương đương với:byte = (byte << 4) | (byte >> 4);

+ Cú pháp: swap (lvalue)
+ Tham số: lvalue là biến một byte
+ Trị trả về: không – CẢNH BÁO: hàm này không cho kết quả
+ Trị trảvề : không

Ví dụ:

```
x=0x45;
swap(x);              //x now is 0x54
```

h. ***MAKE8()***:

Hàm này được dùng để trích một byte từ một biến.

+ Cú pháp:: i8 = MAKE8(var,offset)
+ Tham số: var là số nguyên 16 hoặc 32 bit.
offset is a byte offset of 0,1,2 or 3.
+ Trị trả về: số nguyên 8 bit

Ví dụ:

```
int32 x;
int y;
y = make8(x,3);       // Gets MSB of x
```

i. ***MAKE16()***:

Hàm này được dùng để tạo một số 16 bit từ hai số 8 bit.

+ Cú pháp:: i16 = MAKE16(varhigh, varlow)
+ Tham số: varhigh, varlow là hai số 8 bit.
+ Trị trả về: một số nguyên 16 bit
+ Trị trả về: không

Ví dụ:

```
long x;
int hi,lo;
x = make16(hi,lo);
```

j. ***MAKE32()***:

Hàm này được dùng để tạo một số 32 bit từ số 8 and 16 bit.

+ Cú pháp:: i32 = MAKE32(var1, var2, var3, var4)
+ Tham số: var1-4 là số 8 or 16 bit . var2-4 là các tuỳ chọn.
+ Trị trảvề: một số 32 bit
+ Yêu cầu: Nothing

Ví dụ:

```
int32 x;
int y;
long z;
x = make32(1,2,3,4);  // x is 0x01020304
y=0x12;
z=0x4321;
x = make32(y,z);  // x is 0x00124321
x = make32(y,y,z);  // x is 0x12124321
```

10.**Nhóm Hàm Phục Vụ Delay**:

a. ***DELAY_CYCLES()***:

Hàm này được dùng để thực hiện delay một số xung lệnh (instruction clock) định trước. Một xung lệnh bằng 4 xung dao động

+ Cú pháp:: delay_cycles(count)
+ Tham số: count – hằng số 0~255
+ Trị trả về: không
+ Yêu cầu: không

Ví dụ: delay_cycles(25); //Với xung clock dao động 20MHz, chương trình sẽ delay 5us
//25 x (4/20000000) = 5us

b. ***DELAY_MS()***:

Hàm này được dùng để thực hiện delay một thời gian định trước. Thới gian tính bằng milisecond. Hàm này sẽ thực hiện một số lệnh nhằm delay 1 thời gian yêu cầu. Hàm này không sử dụng bất kỳ timer nào. Nếu sử dụng ngắt (interupt), thời gian thực hiện các lệnh trong khi ngắt không được tính vào thới gian delay.

+ Cú pháp: delay_ms(time)
+ Tham số: time - 0~255 nếu time là một biến số, 0~65535 nếu time là hằng số
+ Trị trả về: không
+ Yêu cầu: #uses delay

c. ***DELAY_US()***:

Hàm này được dùng để thực hiện delay một thời gian định trước. Thới gian tính bằng microsecond. Shorter delays will be INLINE code and longer delays and variable delays are calls to a function. Hàm này sẽ thực hiện một số lệnh nhằm delay 1 thời gian yêu cầu. Hàm này không sử dụng bất kỳ timer nào. Nếu sử dụng ngắt (interupt), thời gian thực hiện các lệnh trong khi ngắt không được tính vào thới gian delay.

+ Cú pháp: delay_us(time)
+ Tham số: time - 0~255 nếu time là một biến số, 0~65535 nếu time là hằng số
+ Trị trả về: không
+ Yêu cầu: #uses delay

11.**Nhóm Hàm Timer và Counter**:

a. ***RESTART_WDT()***:

Hàm này được dùng để Khởi động lại watchdog timer.

+ Cú pháp:: restart_wdt()
+ Tham số: không
+ Trị trả về: không
+ Yêu cầu: #fuses

b. ***SETUP_WDT()***:

Hàm này được dùng để định chế độ hoạt động cho watchdog timer.

+ Cú pháp: setup_wdt(mode)
+ Tham số: Mode có thể là một trong các hằng số sau
  - WDT_18MS
  - WDT_36MS
  - WDT_72MS
  - WDT_144MS
  - WDT_288MS
  - WDT_576MS
  - WDT_1152MS
  - WDT_2304MS
+ Trị trả về: không
+ Yêu cầu: #fuses

các hằng số phải được định nghĩa trong device file PIC16F876.h

c. ***SETUP_COUNTRES()***:

Hàm này được dùng để Set up RTCC hay WDT

+ Cú pháp: setup_counters(rtcc_state, ps_state)
+ Tham số: rtcc_state – hằng số, được định nghĩa như sau
  - RTCC_INTERNAL
  - RTCC_EXT_L_TO_H 32
  - RTCC_EXT_H_TO_L 48

  ps_state– hằng số, được định nghĩa như sau

RTCC_DIV_2	0
RTCC_DIV_4	1
RTCC_DIV_8	2
RTCC_DIV_16	3
RTCC_DIV_32	4
RTCC_DIV_64	5
RTCC_DIV_1286	
RTCC_DIV_2567	
RTCC_8_BIT	0
WDT_18MS	8
WDT_36MS	9
WDT_72MS	10
WDT_144MS	11
WDT_288MS	12
WDT_576MS	13
WDT_1152MS	14
WDT_2304MS	15

+ Trị trả về: không trả về
+ Yêu cầu: các hằng số phải được định nghĩa trong device file .h

d. ***GET_RTCC(), GET_TIMER0()***:

Hàm này được dùng để trả về giá trị biến đếm của real time clock/counter. Khi giá trị timer vượt quá 255, value được đặt trở lại 0 và đếm tiếp tục (…, 254, 255, 0, 1, 2, …)

+ Cú pháp:: value = get_rtcc()
value = get_timer0()
+ Tham số: không
+ Trị trả về: 8 bit int 0~255
+ Yêu cầu: không

e. ***SET_RTCC(), SET_TIMER0()***:

Hàm này được dùng để đặt giá trị ban đầu cho real time clock/counter. Tất cả các biến đều đếm tăng. Khi giá trị timer vượt quá 255, value được đặt trở lại 0 và đếm tiếp tục (…, 254, 255, 0, 1, 2, …)

+ Cú pháp: set_rtcc()
set_timer0()
+ Tham số: 8 bit, value = 0~255
+ Trị trả về: không
+ Yêu cầu: không

f. ***SETUP_TIMER1()***:

Hàm này được dùng để khởi động timer 1. Sau đó timer 1 có thể được ghi hay đọc dùng lệnh set_timer1() hay get_timer1(). Timer 1 là 16 bit timer. Với xung clock là 20MHz, timer 1 tăng 1 đơn vị sau mỗi 1,6us (khi ta dùng chế độ T1_DIV_BY_8) và tràn sau 104,8576ms.

+ Cú pháp: setup_timer_1(mode)
+ Tham số: mode - tham số như sau
T1_DISABLED : tắt timer1
T1_INTERNAL : xung clock của timer1 bằng ¼ xung clock nội của IC (OSC/4)
T1_EXTERNAL : lấy xung ngoài qua chân C0 (không ở chế độ đồng bộ)
T1_EXTERNAL_SYNC : lấy xung ngoài qua chân C0 ở chế độ động bo (khi ở chế độ này nếu CPU ở chế độ SLEEP
T1_CLK_OUT : enable xung clock ra
T1_DIV_BY_1 :65536-(samplingtime(s)/(4/20000000)) timemax=13.1ms
T1_DIV_BY_2 :65536-(samplingtime(s)/(8/20000000)) timemax=26.2ms
T1_DIV_BY_4 : 65536-(samplingtime (s)/(16/20000000))timemax=52.4ms
T1_DIV_BY_8 :65536-(samplingtime(s)/(32/20000000))timemax=104.8ms
+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file .h

g. ***GET_TIMER1()***:

Hàm này được dùng để trả về giá trị biến đếm của real time clock/counter. Khi giá trị timer vượt quá 65535, value được đặt trở lại 0 và đếm tiếp tục (…, 65534, 65535, 0, 1, 2, …)

+ Cú pháp:: value = get_timer1()
+ Tham số: không
+ Trị trả về: 16 bit int 0~65535
+ Yêu cầu: không

h. ***SET_TIMER1()***:

Hàm này được dùng để đặt giá trị ban đầu cho real time clock/counter. Tất cả các biến đều đếm tăng. Khi giá trị timer vượt quá 65535, value được đặt trở lại 0 và đếm tiếp tục (…, 65534, 65535, 0, 1, 2, …)

+ Cú pháp:: set_timer1()
+ Tham số: 16 bit, value = 0~65535
+ Trị trả về: không

+ Yêu cầu: không

i. ***SETUP_TIMER2()***:

Hàm này được dùng để khởi động timer 2. mode qui định số chia xung clock. Sau đó timer 2 có thể được ghi hay đọc dùng lệnh set_timer2() hay get_timer2(). Timer 1 là 8 bit counter/timer.

+ Cú pháp: setup_timer_2(mode,period,postscale)
+ Tham số: mode - tham số như sau
  T2_DISABLED : tắt timer2
  T2_DIV_BY_1 :
  T2_DIV_BY_4 :
  T2_DIV_BY_16 :
  Period – 0~255 qui định khi giá trị clock được reset
  Postscale – 1~16 qui định số lần reset timer trước khi ngắt (interupt)
+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file PIC16F876.h

Ví dụ: Hàm này có thể dùng để đặt tần số cho PWM và được tính như sau

$$fre = [(\underbrace{OSC}_{oscillator\ fre.} / 4) / \underbrace{timemode}_{1\ or\ 4\ or\ 16}] / \underbrace{255}_{tick\ /\ rollover}$$

$\underbrace{\qquad\qquad}_{instruction\ fre.}$

SETUP_TIMER_2(T2_DIV_BY_1,255,1):PWM frequency=[(20MHz/4)/1]/255= 19,61kHz
SETUP_TIMER_2(T2_DIV_BY_4,255,1): PWM frequency = [(20MHz/4)/4]/255 = 4,90kHz
SETUP_TIMER_2(T2_DIV_BY_16,255,1): PWM frequency = [(20MHz/16)/1]/255 = 1,23kHz

j. ***GET_TIMER2()***:

Hàm này được dùng để trả về giá trị biến đếm của real time clock/counter. Khi giá trị timer vượt quá 255, value được đặt trở lại 0 và đếm tiếp tục (…, 254, 255, 0, 1, 2, …)

Cú pháp: value = get_timer2()
Tham số: không
Trị trả về: 8 bit int 0~255
Yêu cầu: không

k. ***SET_TIMER2()***:

Hàm này được dùng để đặt giá trị ban đầu cho real time clock/counter. Tất cả các biến đều đếm tăng. Khi giá trị timer vượt quá 255, value được đặt trở lại 0 và đếm tiếp tục (…, 254, 255, 0, 1, 2, …)

+ Cú pháp: set_timer2(value)
+ Tham số: 8 bit, value = 0~255
+ Trị trả về: không
+ Yêu cầu: không

12.**Nhóm Hàm Quản Lý ADC**:

a. ***SETUP_ADC()***:

Hàm này được dùng để định cấu hình cho bộ biến đổi A/D

+ Cú pháp: setup_adc(mode)
+ Tham số: mode – mode chuyển đổi Analog ra Digital bao gồm

ADC_OFF: tắt chức năng sử dụng A/D
ADC_CLOCK_INTERNAL: thời gian lấy mẫu bằng clock, clock là thời gian clock trong IC
ADC_CLOCK_DIV_2: thời gian lấy mẫu bằng clock/2
ADC_CLOCK_DIV_8: thời gian lấy mẫu bằng clock/8
ADC_CLOCK_DIV_32: thời gian lấy mẫu bằng clock/32

+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file .h

b. ***SETUP_ADC_PORT()***:

Hàm này được dùng để xác định cổng dùng để nhận tín hiệu analog

+ Cú pháp: setup_adc_ports(value)
+ Tham số: value – hằng số được định nghĩa như sau

NO_ANALOGS : không sử dụng cổng analog
ALL_ANALOG : RA0 RA1 RA2 RA3 RA5 RE0 RE1 RE2 Ref=Vdd
ANALOG_RA3_REF : RA0 RA1 RA2 RA5 RE0 RE1 RE2 Ref=RA3
A_ANALOG : RA0 RA1 RA2 RA3 RA5 Ref=Vdd
A_ANALOG_RA3_REF : RA0 RA1 RA2 RA5 Ref=RA3
RA0_RA1_RA3_ANALOG : RA0 RA1 RA3 Ref=Vdd
RA0_RA1_ANALOG_RA3_REF : RA0 RA1 Ref=RA3
ANALOG_RA3_RA2_REF : RA0 RA1 RA5 RE0 RE1 RE2 Ref=RA2,RA3
ANALOG_NOT_RE1_RE2 : RA0 RA1 RA2 RA3 RA5 RE0 Ref=Vdd
ANALOG_NOT_RE1_RE2_REF_RA3 : RA0 RA1 RA2 RA5 RE0 Ref=RA3
ANALOG_NOT_RE1_RE2_REF_RA3_RA2 : RA0 RA1 RA5 RE0 Ref=RA2,RA3
A_ANALOG_RA3_RA2_REF : RA0 RA1 RA5 Ref=RA2,RA3
RA0_RA1_ANALOG_RA3_RA2_REF : RA0 RA1 Ref=RA2,RA3
RA0_ANALOG : RA0
RA0_ANALOG_RA3_RA2_REF : RA0 Ref=RA2,RA3

+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file PIC16F876.h

Ví dụ:

setup_adc_ports(ALL_ANALOG) : dùng tất cả các pins để nhận tín hiệu analog.

setup_adc_ports(RA0_RA1_RA3_ANALOG) : dùng pin A0, A1 và A3 để nhận tín hiệu analog. Điện áp nguồn cấp cho IC được dùng làm điện áp chuẩn.

setup_adc_ports(A0_RA1_ANALOGRA3_REF) : dùng pin A0 và A1 để nhận tín hiệu analog. Điện áp cấp vào pin A3 được dùng làm điện áp chuẩn.

c. ***SETUP_ADC_CHANNEL()***:

Hàm này được dùng để xác định pin để đọc giá trị Analog bằng lệnh READ_ADC()

+ Cú pháp:: setup_adc_channel(chan)
+ Tham số: chân : 0~7 – chọn pin để lấy tín hiệu Analog bao gồm
1 : pin A0
2 : pin A1
3 : pin A2
4 : pin A5
5 : pin E0
6 : pin E1
7 : pin E2
+ Trị trả về: không
+ Yêu cầu: không

d. ***READ_ADC()***:

Hàm này được dùng để đọc giá trị Digital từ bộ biến đổi A/D. Các hàm setup_adc(), setup_adc_ports() và set_adc_channel() phải được dùng trước khi dùng hàm read_adc(). Đối với PIC16F877, bộ A/D mặc định là 8 bit. Để sử dụng A/D 10 bit ta phải dùng thêm lệnh #device PIC16F877 *=16 ADC=10 ngay từ đầu chương trình.

+ Cú pháp: value = read_adc()
+ Tham số: không
+ Trị trả về: 8 hoặc 10 bit
value = 0~255 nếu dùng A/D 8 bit (int)
value = 0~1023 nếu dùng A/D 10 bit (long int)
+ Yêu cầu: không

e. ***SETUP_VREF()***:

Hàm này được dùng để thiết lập điện áp chuẩn bên trong MCU cho bộ analog compare hoặc cho ngõ ra ở chân A2

+ Cú pháp: setup_vref(mode/value)
+ Tham số: mode gồm một trong các hằng số sau
  FALSE (off)
  VREF_LOW for VDD*VALUE/24
  VREF_HIGH for VDD*VALUE/32 + VDD/4
  any may be or'ed with VREF_A2.
  value is an int 0-15.
+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file PIC16F87x .h

13.***Nhóm Hàm Quản Lý Truyền Thông RS-232***:

a. ***GETC(), GETCH(), GETCHAR()***:

Hàm này được dùng để đợi nhận 1 ký tự từ pin RS232 RCV. Nếu không muốn đợi ký tự gởi về, dùng lệnh kbhit().

+ Cú pháp:: ch = getc()
  ch = getch()
  ch = getchar()
+ Tham số: không
+ Trị trả về: ký tự 8 bit
+ Yêu cầu: #use rs232

Ví dụ:

```
#include <16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
char answer;
void main()
{
  printf("Continue (Y,N)?");
  do
  {
    answer=getch();
  } while(answer!='Y' && answer!='N');
}
```

b. ***GETS()***:

Hàm này được dùng để đọc các ký tự (dùng GETC()) trong chuỗi cho đến khi gặp lệnh RETURN (giá trị 13).

+ Cú pháp:: gets(char *string)
+ Tham số: string là con trỏ (pointer) chỉ đến dãy kí tự
+ Trị trả về: không
+ Yêu cầu: #use rs232

Ví dụ:

```
#include <16f877.h>
#include <string.h>
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
char string[30];
void main()
{
  printf("Input string: ");
```

```
  gets(string);
  printf("\n\r");
  printf(string);
}
```

c. ***PUTC(), PUTCHAR()***:

Hàm này được dùng để gởi một ký tự thông qua pin RS232 XMIT. Phải dùng #USE RS232 trước khi thực hiện lệnh này để xác định tốc độ (baud rate) và pin truyền.

+ Cú pháp: putc(cdata)
  putchar(cdata)
+ Tham số: cdata là ký tự 8 bit
+ Trị trả về: không
+ Yêu cầu: #use rs232
Ví dụ:

```
#include <16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)

int  i;
char string[10];
void main()
{
  strcpy(string,"Hello !");                  //copy “Hello !” to string
  for(i=0; i<10; i++) putc(string[i]);       //put each charater of string onto screen
}
```

d. ***PUTS()***:

Hàm này được dùng để gởi mỗi ký tự trong chuỗi đến pin RS232 dùng PUTC(). Sau khi chuỗi được gởi đi thì RETURN (13) và LINE-FEED (10) được gởi đi. Lệnh printf() thường dùng hơn lệnh puts().

+ Cú pháp: puts(string)
+ Tham số: string là chuỗi hằng (constant string) hay dãy ký tự (character array)
+ Trị trả về: không
+ Yêu cầu: #use rs232
Ví dụ: Dùng PUTS()

```
#include <16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
void main()
{
  puts(" ------------ ");
  puts(" |   Hello   | ");
  puts(" ------------ ");
}
```

Dùng PRINTF()

```
#include <16f877.h>
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
void main()
{
  printf(" ------------ \n\r");
  printf(" |   Hello   | \n\r");
```

```
      printf(" ---------------");
    }
```

e. ***KBHIT()***:

Hàm này được dùng để báo đã nhận được bit start.

+ Cú pháp:: value = kbhit()
+ Tham số: không
+ Trị trả về: 0 (hay FALSE) nếu getc() cần phải đợi để nhận 1 ký tự từ bàn phím
  1 (hay TRUE) nếu đã có 1 ký tự sẵn sàng để nhận bằng getc().
+ Yêu cầu: #use rs232

f. ***PRINTF()***:

Hàm này được dùng để xuất một chuỗi theo chuẩn RS232 hoặc theo một hàm xác định. Dữ liệu được định dạng phù hợp với đối số của chuỗi. Các định dạng dữ liệu như sau:

C	Kiểu ký tự
S	Chuỗi hoặc ký tự
U	Số nguyên không dấu
x	Hex int (xuất chữ thường)
X	Hex int (xuất chư hoa)
D	số nguyên có dấu
e	Số thực định dạng kiểu số mũ
f	Kiểu dấu chấm động
Lx	Hex long int (chữ thường)
LX	Hex long int (chữ hoa)
Iu	số thập phân không dấu
Id	Số thập phân có dấu.
%	Dấu %

+ Cú pháp: printf(string)
  printf(cstring, values...)
  printf(fname, cstring, values...)
+ Tham số: String là một chuỗi hằng Hoặc một mảng ký tự không xác định. Values là danh sách các biến phân cách nhau bởi dấu ‘,’ , fname is là tên hàm dùng để xuất dữ liệu (mặc nhiên là putc()).
+ Trị trả về: không
+ Yêu cầu: #use rs232

g. ***SET_UART_SPEED()***:

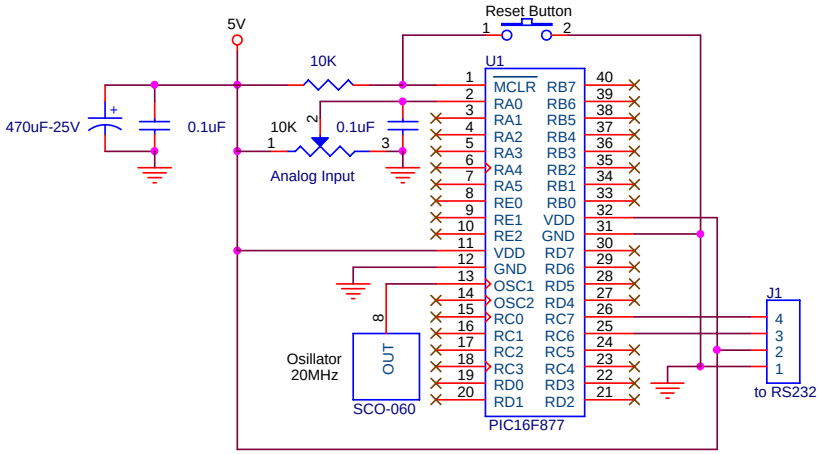
Hàm này được dùng để đặt tốc độ truyền dữ liệu thông qua cổng RS232.

+ Cú pháp:: set_uart_speed(baud)
+ Tham số: baud là hằng số tốc độ truyền (bit/giây) từ 100 đến 115200.
+ Trị trả về: không
+ Yêu cầu: #use rs232

Ví dụ:

```
// Set baud rate based on setting of pins B0 and B1
switch(input_b() & 3)
{
  case 0 : set_uart_speed(2400);   break;
  case 1 : set_uart_speed(4800);   break;
  case 2 : set_uart_speed(9600);   break;
  case 3 : set_uart_speed(19200);  break;
}
```

Ví dụ sử dụng đường truyền RS232 để lấy dữ liệu từ ADC
Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau




```
//file name: using_rs232.c
//using RS232 to get value from A/D converter
//pins connections
//    A0: Analog input (from 10K variable resistor)

#include <16f877.h>
#device  PIC16F877 *=16 ADC=10                    //using 10 bit A/D converter
#use delay(clock=20000000)                       //we're using a 20 MHz crystal
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
int16 value;
void AD_Init()                                   //initialize A/D converter
{
  setup_adc_ports(RA0_RA1_RA3_ANALOG);           //set analog input ports: A0,A1,A3
  setup_adc(ADC_CLOCK_INTERNAL);                 //using internal clock
  set_adc_channel(0);                            //input Analog at pin A0
  delay_us(10);                                  //sample hold time
}
void main()
{
  AD_Init();                                     //initialize A/D converter
  while(1)
  {
    output_high(PIN_C0);                         //motor direction
    output_high(PIN_C3);                         //brake
    value=read_adc();                            //for changing motor speed
    printf("A/D value %lu\r", value);
  }
}
```

14. **<u>Nhóm Hàm Quản Lý Truyền Thông I2C</u>**:

a. ***<u>#USE I2C()</u>***:

Thư viện I2C gồm các hàm dùng cho I2C bus. #USE I2C dùng với các lệnh I2C_START, I2C_STOP, I2C_READ, I2C_WRITE and I2C_POLL. Các hàm phần mềm được tạo ra trừ khi dùng lệnh FORCE_HW.

+ Cú pháp: #use i2c(mode,SDA=pin,SCL=pin[options])
+ Tham số: mode: master/slave - đặt master/slave mode

SCL=pin – chỉ định pin SCL (pin là bit address)
SDA=pin – chỉ định pin SDA
options như sau
ADDRESS=nn : chỉ định địa chỉ slave mode
FAST : sử dụng fast I2C specification
SLOW : sử dụng slow I2C specification
RESTART_WDT : khởi động lại WDT trong khi chờ đọc I2C_READ
FORCE_HW : sử dụng chức năng I2C phần cứng (hardware I2C functions)

b. ***I2C_START()***:

Hàm này được dùng để Khởi động start bit (bit khởi động) ở I2C master mode. Sau khi khởi động start bit, xung clock ở mức thấp chờ đến khi lệnh I2C_WRITE() được thực hiện. Chú ý I2C protocol phụ thuộc vào thiết bị slave.

+ Cú pháp:: i2c_start()
+ Tham số: không
+ Trị trả về: không
+ Yêu cầu: #use i2c

Ví dụ:
```
i2c_start();
i2c_write(0xa0);          //Device address
i2c_write(address);  //Data to device
i2c_start();          //Restart
i2c_write(0xa1);          //to change data direction
data=i2c_read(0);    //Now read from slave
i2c_stop();
```

c. ***I2C_STOP()***:

Hàm này được sử dụng để tắt sử dụng I2C ở master mode.

+ Cú pháp: i2c_stop()
+ Tham số: không
+ Trị trả về: không
+ Yêu cầu: #use i2c

Ví dụ:
```
i2c_start();          //Start condition
i2c_write(0xa0);          //Device address
i2c_write(5);          //Device command
i2c_write(12);          //Device data
i2c_stop();          //Stop condition
```

d. ***I2C_POLL()***:

Hàm này được dùng để hỏi vòng I2C, hàm này chỉ được dùng khi SSP được dùng. Hàm này trả về giá trị TRUE nếu nhận được giá trị ở bộ đệm. Khi hàm này lên TRUE, nếu dùng hàm I2C_READ thì ta được giá trị đọc về.

+ Cú pháp: i2c_poll()
+ Tham số: không
+ Trị trả về: 1 (TRUE) hay 0 (FALL)
+ Yêu cầu: #use i2c

Ví dụ:
```
i2c_start();                    //Start condition
i2c_write(0xc1);                //Device address/Read
count=0;
while(count!=4)
{
  while(!i2c_poll()) ;
  buffer[count++]= i2c_read();     //Read Next
```

```
        }
        i2c_stop();                 // Stop condition
```

e. ***I2C_READ(), I2CREAD(ACK)***:

Hàm này được dùng để Đọc một byte qua cổng I2C Ở thiết bị master: lệnh này tạo xung clock và ở thíet bị claver, lệnh này chờ đọc xung clock. There is no timeout for the slave, use I2C_POLL to prevent a lockup. Use RESTART_WDT in the #USE I2C to strobe the watch-dog timer in the slave mode while waiting.

Cú pháp: i2c_stop()
i2c_stop(ack)

Tham số: tùy chọn, mặc định là 1
ack = 0: không kiểm tra trạng thái thu gởi tín hiệu (ack: acknowlegde)
ack = 1: kiểm tra trạng thái thu gởi tín hiệu

Trị trả về: 8 bit int

Yêu cầu: #use i2c

Ví dụ:
```
i2c_start();
i2c_write(0xa1);
data1 = i2c_read();
data2 = i2c_read();
i2c_stop();
```

f. ***I2C_WRITE()***:

Hàm này được dùng để Gửi từng byte thông qua giao diện I2C. Ở chế độ chủ sẽ phát ra xung Clock với dữ liệu và ở chế độ Slave sẽ chờ xung Clock từ con chủ truyền về. Không tự động đếm ngoài là điều kiện của lệnh này. Lệnh này sẽ trả về bit ACK. Phát LSB trước khi truyền khi đã xác định hướng truyền của dữ liệu truyền (0 cho master sang slave). Chú ý chuẩn giao tiếp I2C phụ thuộc vào thiết bị slave.

+ Cú pháp: i2c_write(data)
+ Tham số: data: 8 bit int
+ Trị trả về: Lệnh này trả về bit ACK
ack = 0: không kiểm tra trạng thái thu gởi tín hiệu (ack: acknowlegde)
ack = 1: kiểm tra trạng thái thu gởi tín hiệu
+ Yêu cầu: #use i2c

Ví dụ:
```
long cmd;
...
i2c_start();                //Start condition
i2c_write(0xa0);            //Device address
i2c_write(cmd);             //Low byte of command
i2c_write(cmd>>8);          //High byte of command
i2c_stop();                 //Stop condition
```

15. **Nhóm Hàm Quản Lý Vào / Ra (SPI)**:

a. ***SETUP_SPI()***:

Hàm này được dùng để khởi gán giá trị ban đầu cho các port giao tiếp với phương thức nối tiếp (Serial Port Interface (SPI)).

+ Cú pháp: setup_spi (mode)
+ Tham số: modes có thể là:

- SPI_MASTER, SPI_SLAVE, SPI_SS_DISABLED
- SPI_L_TO_H, SPI_H_TO_L
- SPI_CLK_DIV_4, SPI_CLK_DIV_16,
- SPI_CLK_DIV_64, SPI_CLK_T2

- Constants from each group may be or'ed together with |.

+ Trị trả về: Không
+ Yêu cầu: Constants phải được khai báo trong tập tin tiêu đề devices .h

Ví dụ:

```
setup_spi(spi_master |spi_l_to_h | spi_clk_div_16 );
```

b. ***SPI_READ()***:

Hàm này được dùng để Trả về giá trị được đọc bởi SPI. Nếu giá trị đó phù hợp với lệnh SPI_READ thì dữ liệu phát xung clock ngoài và dữ liệu nhận lại khi trả về. Nếu không có dữ liệu lúc nó đọc, SPI_READ sẽ đợi dữ liệu.

Trả về giá trị được đọc bởi SPI. Nếu giá trị được truyền tới SPI_READ, thì dữ liệu sẽ đếm xung ngoài, và sau đó dữ liệu sẽ được trả lại khi kết thúc. Nếu chưa có dữ liệu (tín hiệu) thì SPI_read sẽ chờ dữ liệu(tín hiệu).

Nếu có xung rồi thì thực hiện SPI_WRITE(data) tiếp theo SPI_READ() hoặc thực hiện SPI_READ(data). Cả hai hành động đó đều giống nhau và sẽ tạo ra xung đếm. Nếu không có dữ liệu để phát đi thì chỉ cần thực hiện SPI_READ(0) để tạo xung.

Nếu có thiết bị khác cung cấp xung thì khi gọi SPI_READ() phải đợi xung và dữ liệu hoặc sử dụng SPI_DATA_IS_IN() để xác định nếu dữ liệu đã sẵng sàng.

+ Cú pháp: value = spi_read (data)
+ Tham số:: value = spi_read (data)
+ Tham số: dữ liệu tuỳ chọn và là số nguyên 8 bit.
+ Trị trả về: là số nguyên 8 bit.
+ Lợi ích: Lệnh này chỉ sử dụng với thiết bị có phần SPI

Yêu cầu: Không

Ví dụ :

```
in_data = spi_read(out_data);
```

c. ***SPI_WRITE()***:

Hàm này được dùng để gởi một byte (8 bit) đến SPI . Hàm này sẽ ghi giá trị lên SPI.

+ Cú pháp: SPI_WRITE (value)
+ Tham số: value là số nguyên 8 bit
+ Trị trả về: không
+ Yêu cầu : không.

Ví dụ:

```
spi_write( data_out );
data_in = spi_read();
```

d. ***SPI_DATA_IS_IN()***:

Hàm này được dùng để trả về TRUE nếu dữ liệu đã được SPI nhận.

+ Cú pháp: result = spi_data_is_in()
+ Tham số: không
+ Trị trả về: 0 (FALSE) or 1 (TRUE)
+ Yêu cầu : không

Ví dụ:

```
while( !spi_data_is_in() && input(PIN_B2) ) ;
        if( spi_data_is_in() )
        data = spi_read();
```

16. **Nhóm Hàm Quản Lý Xuất Nhập Song Song**:

a. ***SETUP_PSP()***:

Hàm này được dùng để khởi gán port giao tiếp song song(Parallel Slave Port (PSP)). Hàm SET_TRIS_E(value) có thể được sử dụng để set dữ liệu trực tiếp. Dữ liệu có thể đọc hoặc ghi bằng việc sử dụng biến PSP_DATA.

+ Cú pháp: setup_psp (mode)
+ Tham số: mode có thể là:
PSP_ENABLED
PSP_DISABLED
+ Trị trả về: không
+ Yêu cầu: các hằng phải được định nghĩa trong các tập tin tiêu đề devices .h.
Ví dụ:

```
setup_psp(PSP_ENABLED);
```

b. ***PSP_OUTPUT_FULL(), PSP_INPUT_FULL(), PSP_OVERFLOW()***:
Hàm này được dùng để những hàm này kiểm tra cổng song song Slave (Parallel Slave Port (PSP)) để xác định điều kiện và trả về TRUE or FALSE.
+ Cú pháp: result = psp_output_full()
result = psp_input_full()
result = psp_overflow()
+ Tham số: không
+ Trị trả về: 0 (FALSE) or 1 (TRUE)
Yêu cầu: không
Ví dụ:

```
while (psp_output_full()) ;
        psp_data = command;
while(!psp_input_full()) ;
        if ( psp_overflow() )
                error = TRUE;
        else
                data = psp_data;
```

17. **Nhóm Các Hàm Điều Khiển MCU**:
a. ***SLEEP()***:
Hàm này được dùng để duy trì trạng thái "ngủ" của chíp cho đến khi nhận được tác động từ bên ngoài
+ Cú pháp : sleep()
+ Tham số : không
+ Trị trả về : không
+ Yêu cầu : không
Ví dụ:

```
SLEEP();
```

b. ***RESET_CPU()***:
Hàm này được dùng để reset MCU
+ Cú pháp: reset_cpu()
+ Tham số: không
+ Trị trả về: không
+ Yêu cầu : không
Ví dụ:

```
if(checksum!=0)
reset_cpu();
```

c. ***RESTART_CAUSE()***:
Hàm này được dùng để trả về nguyên nhân reset MCU lần cuối cùng.
+ Cú pháp: value = restart_cause()
+ Tham số: không

+ Trị trả về: giá trị chỉ ra nguyên nhân gây reset cuối cùng trong bộ vi xử lý. Giá trị này tuỳ thuộc vào mỗi loại chip cụ thể. Có thể tham khảo trên file device .h để biết được các giá trị đặt biệt này. Ví dụ: WDT_FROM_SLEEP WDT_TIMEOUT, MCLR_FROM_SLEEP and NORMAL_POWER_UP.
+ Yêu cầu: hằng phải được khai báo trong các tập tin tiêu đề devices .h

Examples:

```
switch ( restart_cause() )
{
        case WDT_FROM_SLEEP:
        case WDT_TIMEOUT:
                handle_error();
}
```

d. ***READ_BANK()***:

Hàm này được dùng để đọc một byte dữ liệu từ vùng RAM .
+ Cú pháp: value = read_bank (bank, offset)
+ Tham số: bank là RAM trong IC tuỳ thuộc vào loại IC (đối với 16f877 có 3 bank), offset là bước nhảy khi thực thi (bắt đầu là 0)
+ Trị trả về: số nguyê 8 bit
+ Yêu cầu không

Ví dụ:

```
// See write_bank example to see
// how we got the data
// Moves data from buffer to LCD
i=0;
do
 {
        c =read_bank(1,i++);
        if(c!=0x13)
        lcd_putc(c);
}
while (c!=0x13);
```

e. ***WRITE_BANK()***:

Hàm này được dùng để ghi một byte lên RAM
+ Cú pháp: : write_bank (bank, offset, value)
+ Tham số: tương tự với read_bank, value là dữ liệu 8 bit
+ Trị trả về: không
+ Yêu cầu : không

Ví dụ:

```
i=0;     // Uses bank 1 as a RS232 buffer
do
{
        c=getc();
        write_bank(1,i++,c);
}
while (c!=0x13);
```

18. **Nhóm Hàm Quản Lý CCP**:

Nhóm này bao gồm các hàm:

setup_ccpX()
set_pwmX_duty()

Capture Mode: *Khi các chân RC1/CCP2 và RC2/CCP1 xảy ra các sự kiện sau:*

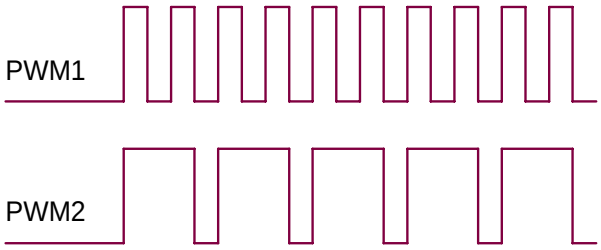
- Every falling edge : Nhận xung cạnh xuống
- Every rising edge : Nhận xung cạnh lên
- Every 4th rising edge : Nhận xung cạnh lên sau 4 xung
- Every 16th rising edge : Nhận xung cạnh lên sau 16 xung.

Thì các giá trị CCP1,CCP2 tương ứng sẽ mang giá trị của TIMER1 tại thời điểm đó.

Compare Mode: *Khi giá trị đếm của TIMER1 bằng với giá trị CCP1/CCP2 thì xảy ra các sự kiện:*

Ngõ ra RC1/RC2 ở mức cao
Ngõ ra RC1/RC2 ở mức thấp
Xảy ra ngắt
Reset lại Timer1.

PWM Mode (Pulse Width Modulation): chế độ phát xung



a. ***SETUP_PWM1_DUTY(),SETUP_PWM1_DUTY()***:

Hàm này được dùng để xác định % thời gian trong 1 chu kỳ, PWM ở mức cao

+ Cú pháp: set_pwm1_duty(value)
set_pwm2_duty(value)
+ Tham số: value có thể là biến hay hằng số với 8 hay 16 bit
+ Trị trả về: không
+ Yêu cầu: không
Ví dụ: set_pwm1_duty(512): đặt 50% mức cao (50% duty)

b. ***SETUP_CCP1(),SETUP_CCP2()***:

Hàm này được dùng để Khởi động CCP. Bộ đếm CCP có thể được thực hiện thông qua việc sử dụng CCP_1 và CCP_2. CCP hoạt động ở 3 mode. Ở capture mode, CCP copy giá trị đếm timer 1 vào CCP_x khi cổng vào nhận xung. Ở compare mode, CCP thực hiện 1 tác vụ chỉ định trước khi timer 1 và CCP_x bằng nhau. Ở chế độ PWM, CCP tạo một xung vuông.

+ Cú pháp: setup_ccp1(mode)
setup_ccp2(mode)
+ Tham số: mode là hằng số như sau

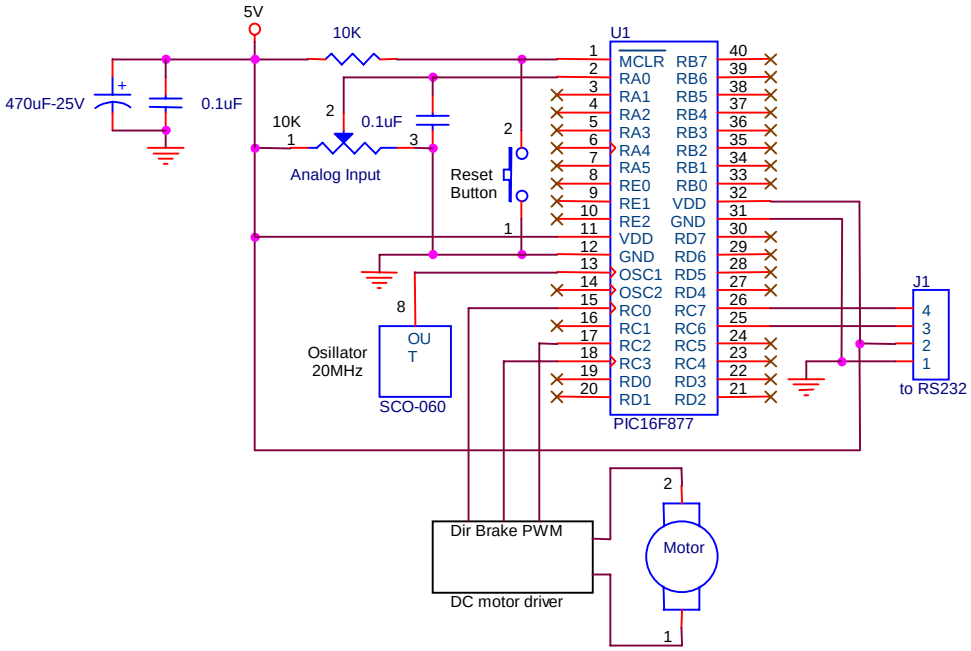
```
long CCP_1;
    #byte  CCP_1    =          0x15
    #byte  CCP_1_LOW=          0x15
    #byte  CCP_1_HIGH=         0x16
long CCP_2;
    #byte  CCP_2    =          0x1B
    #byte  CCP_2_LOW=          0x1B
    #byte  CCP_2_HIGH=         0x1C
// Tắt CCP
    CCP_OFF;
// Đặt CCP ở chế độ capture
```

```
        CCP_CAPTURE_FE;              // Nhận cạnh xuống của xung
        CCP_CAPTURE_RE;              // Nhận cạnh lên của xung
        CCP_CAPTURE_DIV_4;           // Nhận xung sau mỗi 4 xung vào
        CCP_CAPTURE_DIV_16;          // Nhận xung sau mỗi 16 xung vào
    // Đặt CCP ở chế độ compare
        CCP_COMPARE_SET_ON_MATC;     // Output high on compare
        CP_COMPARE_CLR_ON_MATCH;     //Output low on compare
        CP_COMPARE_INT;      //      Interrupt on compare
        CCP_COMPARE_RESET_TIMER  // Reset timer on compare
    // Đặt CCP ở chế độ PWM
        CCP_PWM                      // Mở PWM
        CCP_PWM_PLUS_1
        CCP_PWM_PLUS_2
        CCP_PWM_PLUS_3
```

+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file PIC16F87x.h

Ví dụ sử dụng PWM:

Sơ đồ mạch điều khiển động cơ DC sử dụng PIC16F877 và chương trình ví dụ như sau




```
//file name: using_PWM.c
//using PWM to control DC motor, motor speed is controlled using Analog input at A0
//pins connections
//A0: Analog input (from 10K variable resistor)
//C0: motor direction   C3: motor brake   C2: PWM
#include <16f877.h>
#device  PIC16F877 *=16 ADC=10                  //using 10 bit A/D converter
#use delay(clock=20000000)                      //we're using a 20 MHz crystal
int16 value;
void AD_Init()                                  //initialize A/D converter
{
  setup_adc_ports(RA0_RA1_RA3_ANALOG);          //set analog input ports:   A0,A1,A3
```

```
  setup_adc(ADC_CLOCK_INTERNAL);          //using internal clock
  set_adc_channel(0);                     //input Analog at pin A0
  delay_us(10);                           //sample hold time
}
void main()
{
  AD_Init();                              //initialize A/D converter
  SETUP_CCP1(CCP_PWM);                    //set CCP1(RC2) to PWM mode
  SETUP_TIMER_2(T2_DIV_BY_1,255,1);       //set the PWM frequency to
[(20MHz/4)/1]/255=16.9KHz
  while(1)
  {
    output_high(PIN_C0);                  //motor direction
    output_high(PIN_C3);                  //brake
    value=read_adc();                     //for changing motor speed
    set_PWM1_duty(value);                 //output PWM to motor driver
  }
}
```

19. **Nhóm Hàm Quản Lý Ngắt**:

a. ***EXT_INT_EDGE()***:

Hàm này được dùng để qui định thời điểm ngắt tác động: cạnh lên hay xuống.

+ Cú pháp:: ext_int_edge(source,edge)
+ Tham số: source: giá trị mặc định là 0 cho PIC16F877
  edge: H_TO_L cạnh xuống $5V \rightarrow 0V$
  L_TO_H cạnh lên $0V \rightarrow 5V$
+ Trị trả về: không
+ Yêu cầu: các hằng số phải được định nghĩa trong device file PIC16F87x.h

b. ***#INT_xxx***:

Khai báo khởi tạo hàm ngắt. Hàm ngắt có thể không có bất kỳ tham số nào. Trình biên dịch tạo code để nhảy đến hàm ngắt khi lệnh ngắt thực hiện. Trình biên dịch cũng tạo nên code để lưu trữ trạng thái của CPU và xóa cờ ngắt. Dùng lệnh NOCLEAR sau #INT_xxx để không xóa cờ ngắt này. Trong chương trình , phải dùng lệnh ENABLE_INTERRUPTS(INT_xxxx) cùng với lệnh ENABLE_INTERRUPTS(GLOBAL) để khởi tạo ngắt.

+ Cú pháp: #INT_AD Kết thúc biến đổi A/D
  #INT_BUSCOL Xung đột bus
  #INT_CCP1 Capture or Compare on unit 1
  #INT_CCP2 Capture or Compare on unit 2
  #INT_EEPROM Kết thúc viết vào EEPROM
  #INT_EXT Ngắt ngoài
  #INT_LOWVOLT Low voltage detected
  #INT_PSP Parallel Slave Port data in
  #INT_RB Port B any change on B4-B7
  #INT_RDA RS232 receive data available
  #INT_RTCC Timer 0 (RTCC) overflow
  #INT_SSP SPI or I2C activity
  #INT_TBE RS232 transmit buffer empty
  #INT_TIMER1 Timer 1 overflow
  #INT_TIMER2 Timer 2 overflow

c. ***DISABLE_INTERRUPTS()***:

Hàm này được dùng để Tắt ngắt tại mức quy định bởi level. Mức toàn cục (GLOBAL level) không tắt bất kỳ các ngắt trong chương trình nhưng ngăn cản bất kỳ ngắt nào trong chương trình, đã được khởi tạo trước đó. Các mức ngắt hợp lệ giống như được dùng trong #int_xxx. GLOBAL level không tắt các ngắt gioao diện (peripheral interrupt). Chú ý không cần thiết tắt ngắt bên trong một ngắt khác vì các ngắt náy đã được tự động tắt.

+ Cú pháp:: disable_interrupts(level)
+ Tham số: một trong các hằng số sau
  - GLOBAL
  - INT_RTCC
  - INT_RB
  - INT_EXT
  - INT_AD
  - INT_TBE
  - INT_RDA
  - INT_TIMER1
  - INT_TIMER2
  - INT_CCP1
  - INT_CCP2
  - INT_SSP
  - INT_PSP
  - INT_BUSCOL
  - INT_LOWVOLT
  - INT_EEPROM
+ Trị trả về: không
+ Yêu cầu: phải dùng với #int_xxx

d. ***ENABLE_INTERRUPTS()***:

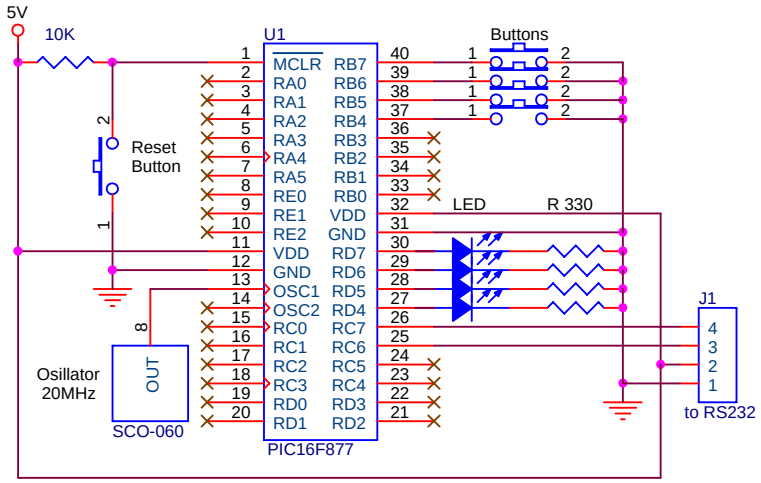
Hàm này được dùng để Khởi tạo ngắt tại mức quy định bởi level. Một thủ tục ngắt (interrupt procedure) cần được định nghĩa. Mức toàn cục (GLOBAL level) không khởi tạo bất kỳ ngắt chỉ định nào mà chỉ khởi tạo các biến ngắt đươc đã khởi tạo trước đó.

+ Cú pháp:: enable_interrupts(level)
+ Tham số: level - một trong các hằng số sau
  - GLOBAL
  - INT_RTCC
  - INT_RB
  - INT_EXT
  - INT_AD
  - INT_TBE
  - INT_RDA
  - INT_TIMER1
  - INT_TIMER2
  - INT_CCP1
  - INT_CCP2
  - INT_SSP
  - INT_PSP
  - INT_BUSCOL
  - INT_LOWVOLT
  - INT_EEPROM
+ Trị trả về: không
+ Yêu cầu: phải dùng với #int_xxx

e. ***Một số ví dụ sử dụng ngắt***:

Ví dụ 1: vi dụ sử dụng enable_interrupts.

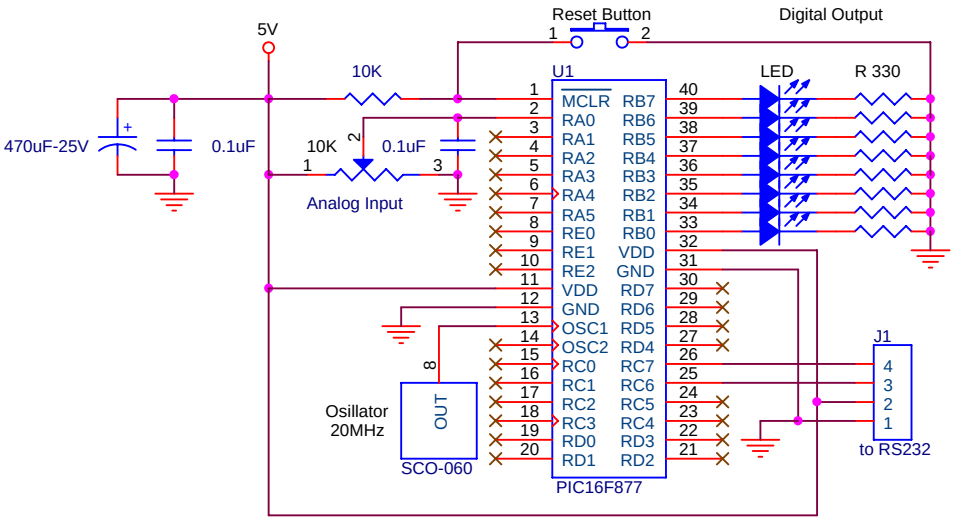
Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau




```
//file name: using_interrupt_rb.c
//using enable_interrupt(int_rb)
//pins connections
//        B4-7: buttons
//        D4-7: LEDs
#include <16F877.h>
#device  PIC16F877 *=16 ADC=10
#use delay(clock=20000000)
#byte portb = 0x06
#byte portd = 0x08
#INT_RB
rb_isr()
{
  portd=portb;
}
void main()
{
  set_tris_b(0xF0);
  set_tris_d(0x00);
  enable_interrupts(INT_RB);
  enable_interrupts(GLOBAL);
  while(1)
  {
  }
}
```

Ví dụ 2: Ví dụ sử dụng ngắt ADC.

Sơ đồ mạch biến đổi A/D sử dụng PIC16F877 và chương trình ví dụ như sau




```
//file name: using_adc.c
//using A/D converter
//pin conections
//    A0: Analog input
//    port B: Digital output (LED)
#include <16f877.h>
#use delay(clock=20000000)                    //use a 20 MHz crystal
#byte portb=0x06
#INT_AD
void int_ad()
{
  //Làm chương trình nào đó
}
void main()
{
  set_tris_b(0x00);                           //set register for I/O port B
  setup_adc_ports(RA0_RA1_RA3_ANALOG);
  setup_adc(ADC_CLOCK_INTERNAL);
  set_adc_channel(0);                         //analog input to pin A0
  while(1)
  {
    delay_us(10);             //sample hold time
    portb=read_adc();         //output portb digital value
   }
}
```

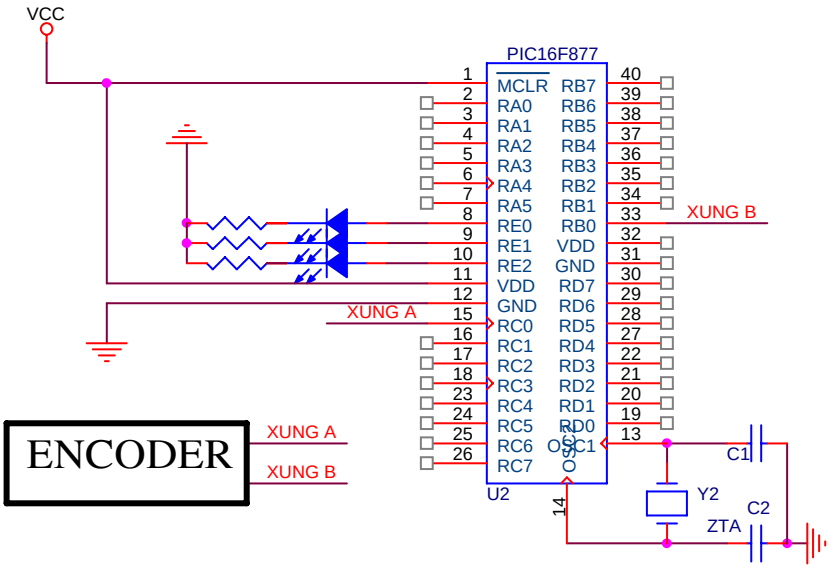
Ví dụ 3: Sử dụng ngắt timer 1

Chương trình ví dụ như sau

```
//file name: using_interrupt_ext.c
//using external interrupts to get motor speed with encoder
//pins connections
//    B0: from encoder
#include <16F877.h>
#use delay(clock=20000000)
#include <4bit_7seg_display.c>
int16 count;
```

```
float speed;
#INT_TIMER1
void Sampling_Time()
{
  speed=(float)count*3;                    //rpm Encoder Resolution: 2000
  count=0;
  set_timer1(59286);                       //10ms Prescaler:8
}
#INT_EXT
void HS_Input()
{
  count++;
}
main()
{
  count=0;
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
  ext_int_edge(L_TO_H);
  enable_interrupts(INT_TIMER1);
  enable_interrupts(INT_EXT);
  enable_interrupts(GLOBAL);
  set_timer1(59286);
  while(1)
  {
    BIN2BCD((int16)speed,0);
  }
}
```

Ví dụ 4: Sử dụng ngắt timer1 và ngắt ngoài.
Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau

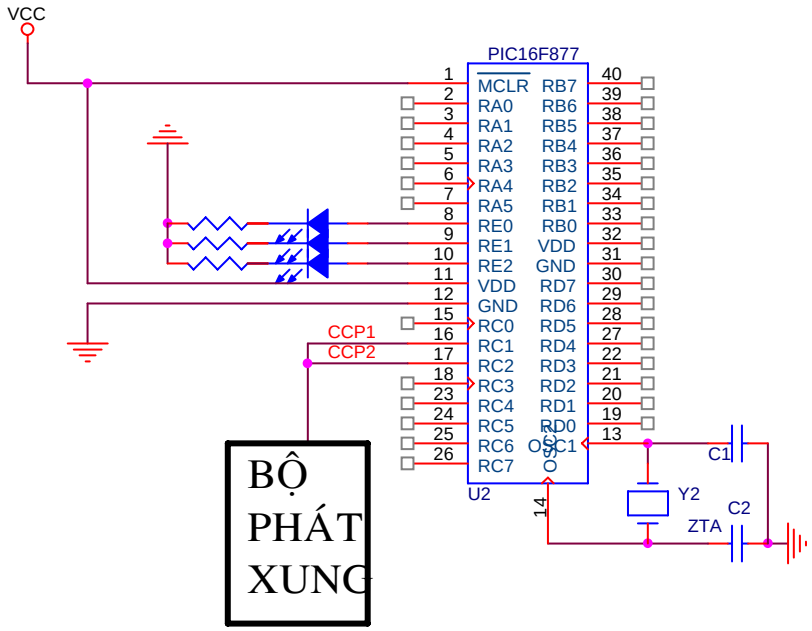


//Chương trình dùng để hiện thị số vòng quay của Encoder, khi quay thuận số vòng quay tăng, khi quay nghịch số vòng quay giảm.
//This program is ued for display the number of pulses of encoder

```
//When encoder rotates forward, the number will increase, otherwise decrease
// pusle A is connectted to pin C0;
// pulse B is connectted to pin B0
#include <16f877.h>
#device PIC16F877 *=16 ADC=10 //khai bao ADC=10 de doc ve duoc so 10 bit
#include <4bit_7seg_display_new.c>
//the function BIN2BCD(int32 Num, char DecimalPoint)
//is used to display LED 7 seg
#use delay(clock=20000000)
BOOLEAN forward=TRUE;
int16 count;
//external interrupt
#INT_EXT
void ext_isr()
{
        if (input(PIN_C0)) forward=TRUE;
        else forward=FALSE;
}
//Timer1 interrupt
#INT_TIMER1
void timer1_isr()
{
        count++;
}
void main()
{
        int16 temp1,countbackward,temp;
        temp=0;
        set_tris_b(0x01);
        ext_int_edge(0,L_TO_H);
        setup_timer_1(T1_EXTERNAL);
        enable_interrupts(INT_EXT);
        enable_interrupts(INT_TIMER1);
        enable_interrupts(GLOBAL);
        while (TRUE)
        {
                set_timer1(temp);
                while (forward)
                {
                        temp=get_timer1();
                        BIN2BCD(temp,0);
                }
                set_timer1(0);
                while (!forward)
                {
                        countbackward=get_timer1();
                        temp1=temp-countbackward;
                        BIN2BCD(temp1,0);
                }
```

```
                temp=temp1;
        }
}
```

Ví dụ 5: Sử dụng ngắt CCP.
Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau:

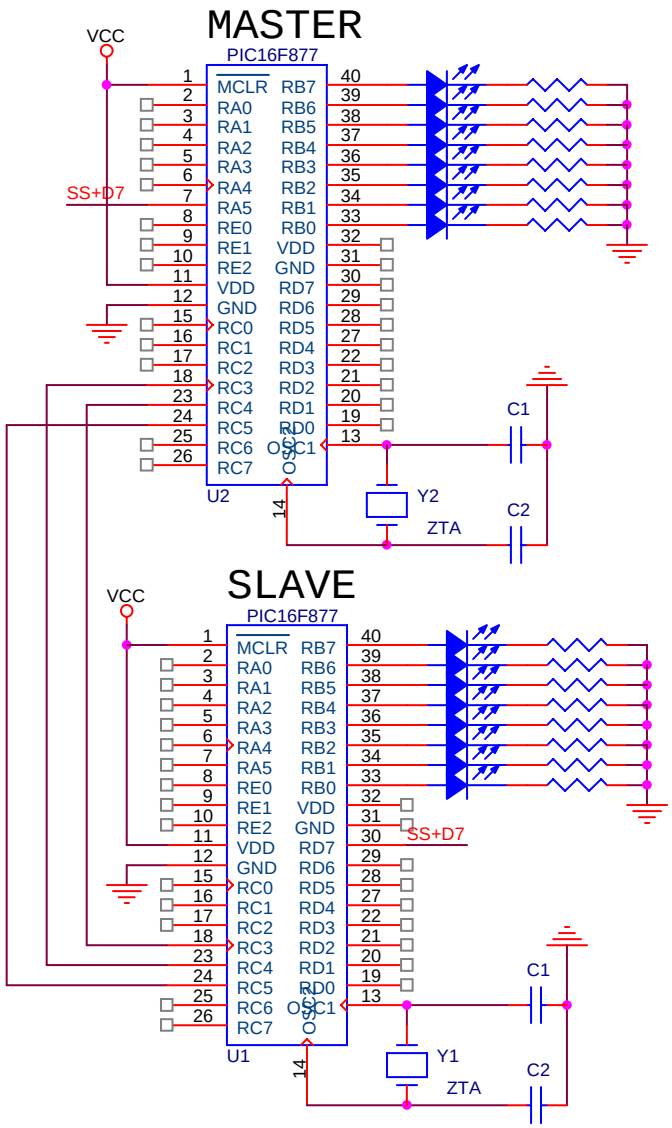



```
//Chương trình dùng để đo độ rộng của xung.
#include <16f877.h>
#include <stdlib.h>
#include <math.h>
#include <4bit_7seg_display.c>
#use delay(clock = 20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6, rcv=PIN_C7)
// Connect a pulse generator to pin 3 (C2) and pin 2 (C1)
int32 rise,fall,pulse_width;
int32 count;
int32 tam;
#INT_CCP1
void ccp1_isr()
        {
        rise = CCP_1;
        count=0;
        }
#INT_CCP2
void ccp2_isr()
        {
   fall = CCP_2;
                        if (count==0)
                        {
                                pulse_width = fall - rise;
                                tam=(2*pulse_width/10);//change into milisecond
                        }
```

```
                        else
                                tam=((13107*count)+(fall*2/10)-(rise*2/10))/1;
                                printf("%lu",tam);
                                BIN2BCD(tam/10000,0);
        }
#INT_TIMER1
void timer1_isr()
        {
        count++;
        }
void main()
        {
        int16 tam;
        setup_ccp1(CCP_CAPTURE_RE);    // Configure CCP1 to capture rise
        setup_ccp2(CCP_CAPTURE_FE);    // Configure CCP2 to capture fall
        setup_timer_1(T1_INTERNAL);//|T1_DIV_BY_8);    // Start timer 1
        enable_interrupts(INT_CCP1);   // Setup interrupt on rising edge
        enable_interrupts(INT_CCP2);   // Setup interrupt on falling edge
        enable_interrupts(INT_TIMER1); // Setup interrupt Timer1
        enable_interrupts(GLOBAL);
        BIN2BCD(0,0);
        while(TRUE)
        {
        }
}
```

Ví dụ 6: Sử dụng ngắt SPI.

Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau



CHƯƠNG TRÌNH MASTER:

```
// I2C Communications
// Master Controller
#include <16F877.h>
#device  PIC16F877 *=16 ADC=10
#include <math.h>
#include <stdlib.h>
#include <4bit_7seg_display.c>
#fuses hs, nowdt, noprotect, put, nolvp, brownout
#use delay(clock=20000000) //20mhz
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
void WriteTo(int16 Num)
{
        int16 tmp;
        tmp=Num;
        spi_write(tmp);          // Low byte of command
        delay_us(100);
        spi_write(tmp>>8);// High byte of command
}
void main()
```

```
{
int32 data_out=0;
int32 data_in=0;
  while(true)
  {
  setup_spi(spi_master |spi_l_to_h |spi_clk_div_16 );
  WriteTo(data_out);
  BIN2BCD(data_out,0);
  data_out++;
  delay_ms(200);
  if (data_out==9999) data_out=0;
  }
}
```

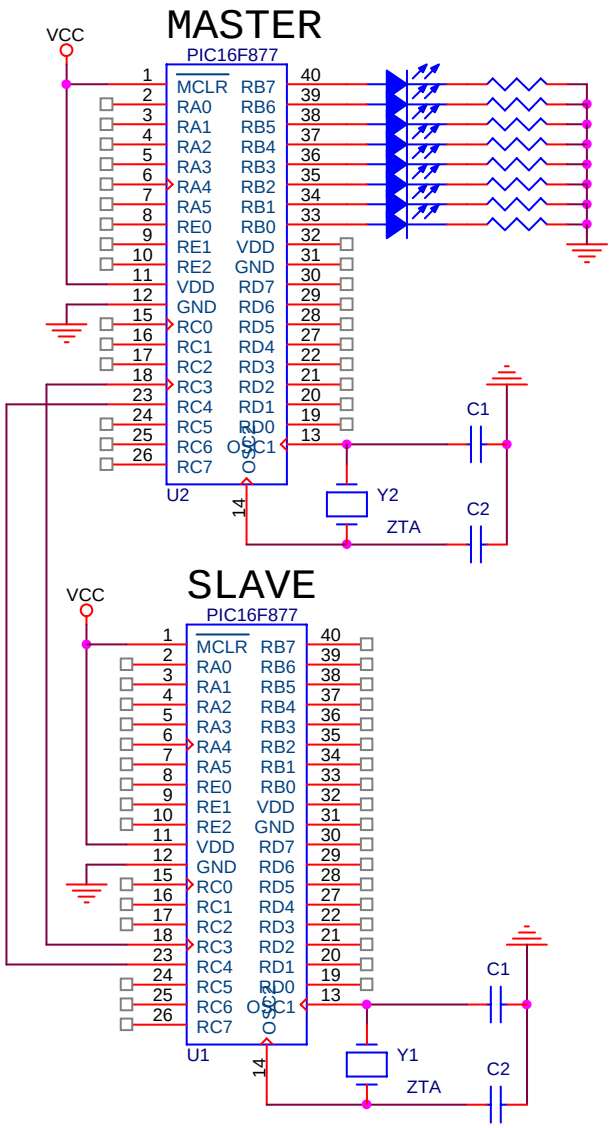
CHƯƠNG TRÌNH SLAVE:

```
// Slave Controller: Address: 0xa0
#include <16F877.h>
#device  PIC16F877 *=16 ADC=10
#include <stdlib.h>
#include <4bit_7seg_display.c>
#include <math.h>
#fuses hs, nowdt, noprotect, put, nolvp, brownout
#use delay(clock=20000000)
int16 data_in=0;
int data_in_high,data_in_low;
int1 ok=0;
#INT_SSP
void isr_ssp()
{
  if (!ok)
  {
    data_in_low = spi_read();
    ok=1;
  }
  else
  {
    data_in_high=spi_read();
    data_in=make16(data_in_high,data_in_low);
    ok=0;
    BIN2BCD(data_in,0);
  }
}
void main()
  {
  setup_spi(spi_slave |spi_l_to_h |spi_clk_div_16 );
  enable_interrupts(INT_SSP);
  enable_interrupts(GLOBAL);
  while(true)
    {
```

```
    }
}
```

Ví dụ 7: Sử dụng ngắt I2C

Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau



CHƯƠNG TRÌNH MASTER:

```
// I2C Communications
// Master Controller
#include <16F877.h>
#device  PIC16F877 *=16 ADC=10
#include <math.h>
#include <stdlib.h>
#include <4bit_7seg_display_new.c>
#fuses hs,nowdt,noprotect,put,nolvp,brownout
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7)
#use I2C(master, sda=PIN_C4, scl=PIN_C3,NOFORCE_SW)
void WriteTo(int16 Num)
```

```
{
        int16 tmp;
        tmp=Num;
   i2c_write(tmp);       // Low byte of command
   delay_us(100);
   i2c_write(tmp>>8);// High byte of command
}
void main()
{
   char s[10];
   int16 tam;
   BIN2BCD(0,0);
   set_tris_c(0x80);
   i2c_start();
   while(1)
   {
                gets(s);
                tam=atoi32(s);
                delay_ms(100);
                i2c_write(0xa0);//dia chi slave
                WriteTo(tam);
                delay_ms(100);
                BIN2BCD(tam,0);
        }
        i2c_stop();
}
```

CHƯƠNG TRÌNH SLAVE:

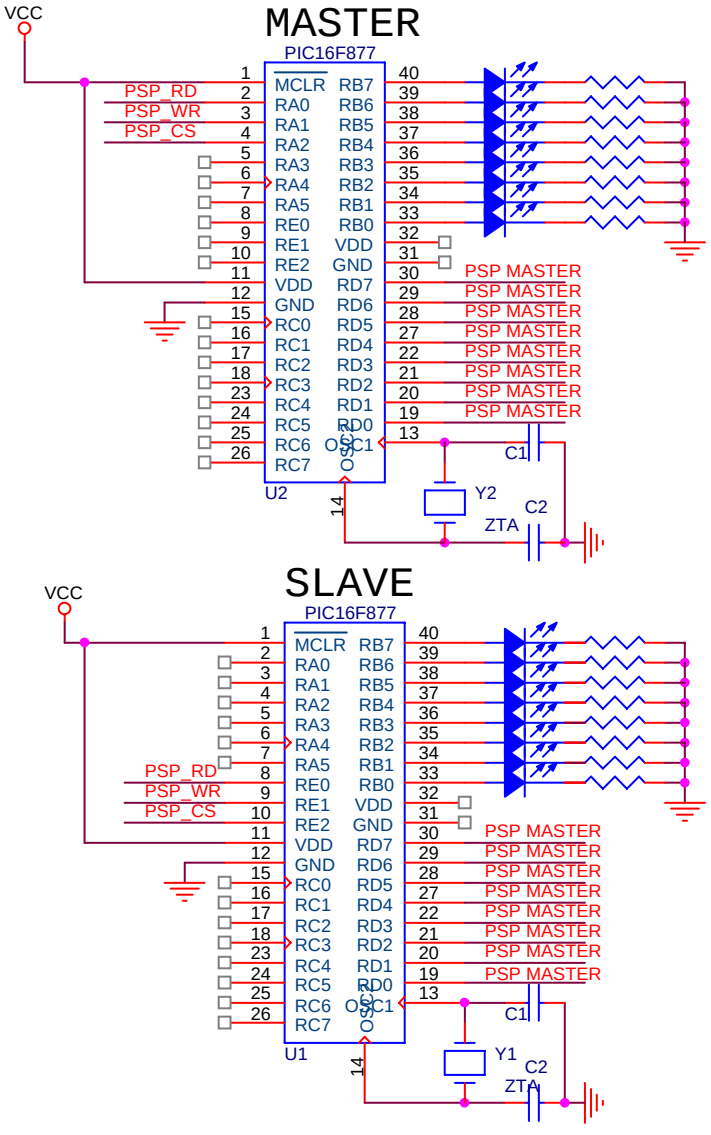
```
// Slave Controller: Address: 0xa0
// Slave receives 2 bytes
#include <16F877.h>
#device  PIC16F877 *=16 ADC=10
#include <stdlib.h>
#include <4bit_7seg_display_new.c>
#fuses hs, nowdt, noprotect, put, nolvp, brownout
#use delay(clock=20000000)
#use I2C(slave,sda=PIN_C4,scl=PIN_C3,address=0xa0,NOFORCE_SW)
int tam[3];
int i=0;
#INT_SSP
void ssp_isr()
{
        if (i2c_poll()==TRUE)
   {
        tam[i++]=i2c_read();
                if (i==3) i=0;
   }
}
void main()
{
```

```
        int16 dem=0;
        tam[1]=0;
        tam[2]=0;
        enable_interrupts(GLOBAL);
        enable_interrupts(INT_SSP);
        BIN2BCD(0,0);
        while (1)
        {
                if (i==0)
                {
                        dem=make16(tam[2],tam[1]);
                        BIN2BCD(dem,0);
                        delay_ms(10);
                }
        }
}
```

Chương trình giao tiếp với keypad và máy tính được viết ở thư viện của phòng.

Ví dụ 8: Sử dụng ngắt PSP

Sơ đồ mạch dùng PIC16F877 và chương trình ví dụ như sau



CHƯƠNG TRÌNH MASTER:

```
#include <16F877.h>
#use delay(clock=20000000)
#byte portb=0x06
#byte portd = 0x08
#define psp_rd pin_a0
#define psp_wr pin_a1
#define psp_cs pin_a2
void main ()
{
  set_tris_a(0x00);
  set_tris_b(0x00);
  portb = 0x00;
  output_high(psp_cs);
  output_high(psp_rd);
  output_high(psp_wr);
while(TRUE)
      {
        portb++;
        output_low(psp_cs);
        delay_us(10);
        output_low(psp_wr);
        PSP_DATA=portb;
        delay_us(50);
        output_high(psp_wr);
        delay_us(20);
        output_high(psp_cs);
        delay_ms(700);
     }
}
```

CHƯƠNG TRÌNH SLAVE:

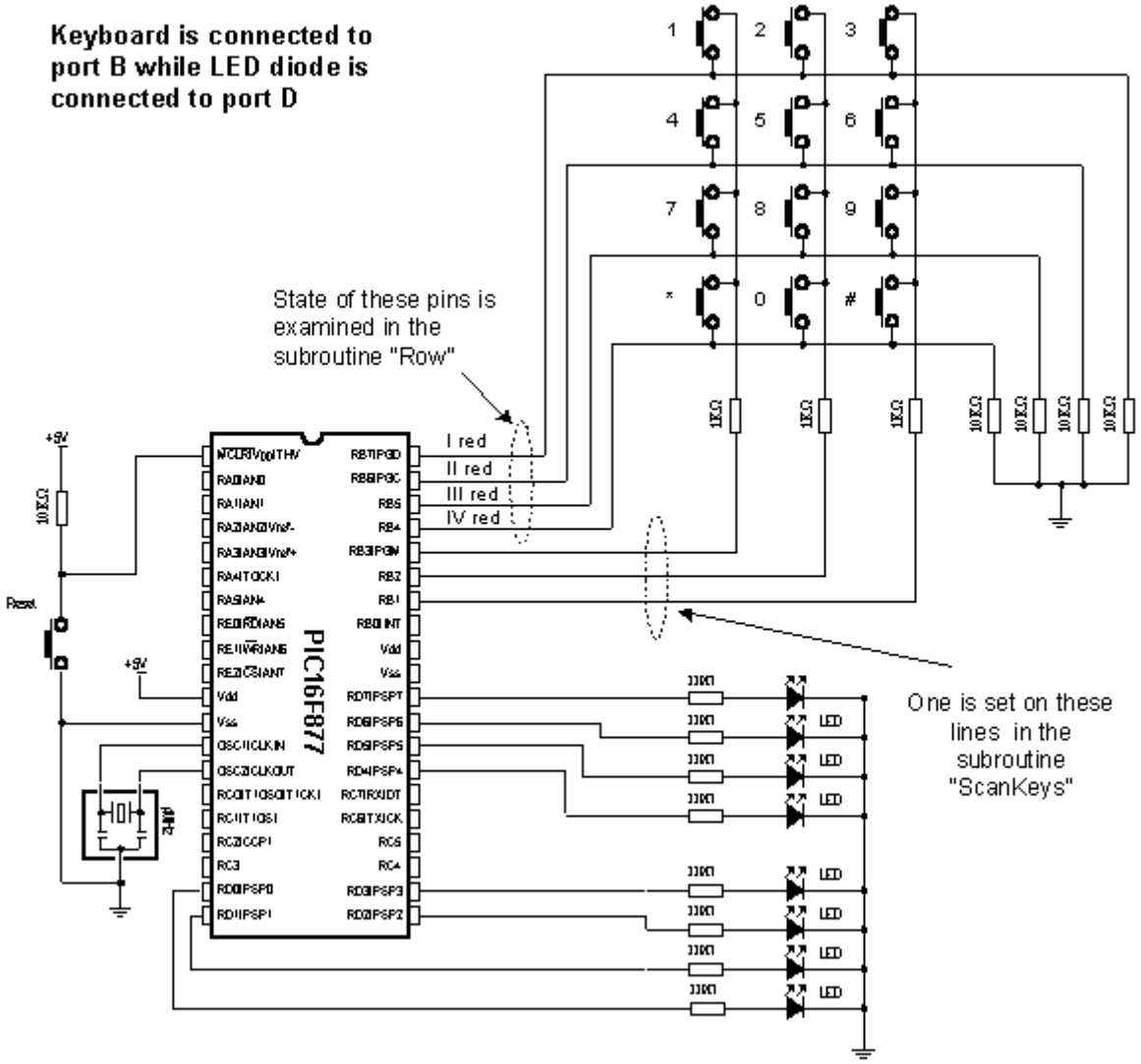
```
#include <16F877.h>
#device  PIC16F877 *=16 ADC=10
#use delay(clock=20000000)
#byte  portb=0x06
#INT_PSP
void psp_isr()
{
  portb = PSP_DATA;
  delay_ms(100);
}
void main()
 {
  set_tris_b(0x00);
  //set_tris_d(0xff);
  portb=0x00;
  setup_psp(PSP_ENABLED);
  //enable_interrupts(global);
  //enable_interrupts(INT_PSP);
```

```
  while (TRUE)
  {
     while(!psp_input_full());
      portb = PSP_DATA;
     delay_ms(100);
  }
}
```

Ví dụ: Dọc bàn phím hex

Đọc bàn phím 12 phím như sơ đồ và xuất giá trị ra led theo yêu cầu sau:

+ phím 0: tất cả các led sáng.	+ phím 6: led ở chân RD2&RD1 sáng.
+ phím 1: led ở chân RD0 sáng.	+ phím 7: led ở chân RD2&RD1&RD0 sáng.
+ phím 2: led ở chân RD1 sáng.	+ phím 8: led ở chân RD3 sáng.
+ phím 3: led ở chân RD1&RD0 sáng.	+ phím 9: led ở chân RD3&RD0 sáng.
+ phím 4: led ở chân RD2 sáng.	+ phím *: Các led ở chân RD0 tới RD3 sáng.
+ phím 5: led ở chân RD2&RD0 sáng.	+ phím #: Các led ở chân RD4 tới RD7 sáng.



Chương Trình:

```
#include "C:\Program Files\PICC\quetphim.h"
#include <KBD.C> // Khai báo thư viện đọc phàn phím.
#int_PSP
PSP_isr()  // Khai báo hàm ngắt ở cổng vào ra song song.
{
```

```
}
void main()  // Bắt đầu chương trình chính.
{
  char k;
  port_b_pullups(TRUE);
  kbd_init(); // Khoi tao gia tri ban dau cho ham doc ban phim
  enable_interrupts(INT_PSP);
  enable_interrupts(global);
  while(1)
  {
    k=kbd_getc();  // Goi ham doc ban phim, day la ham duoc dinh nghia trong file kbd.c
    switch(k)    // Xuất led ở port D.
      {
        case '0': set_tris_d(0xff);
        case '1': set_tris_d(0x01);
        case '2': set_tris_d(0x02);
        case '3': set_tris_d(0x03);
        case '4': set_tris_d(0x04);
        case '5': set_tris_d(0x05);
        case '6': set_tris_d(0x06);
        case '7': set_tris_d(0x07);
        case '8': set_tris_d(0x08);
        case '9': set_tris_d(0x09);
        case '*': set_tris_d(0x0f);
        case ‘#’: set_tris_d(0xf0);
      }
  }
}
```