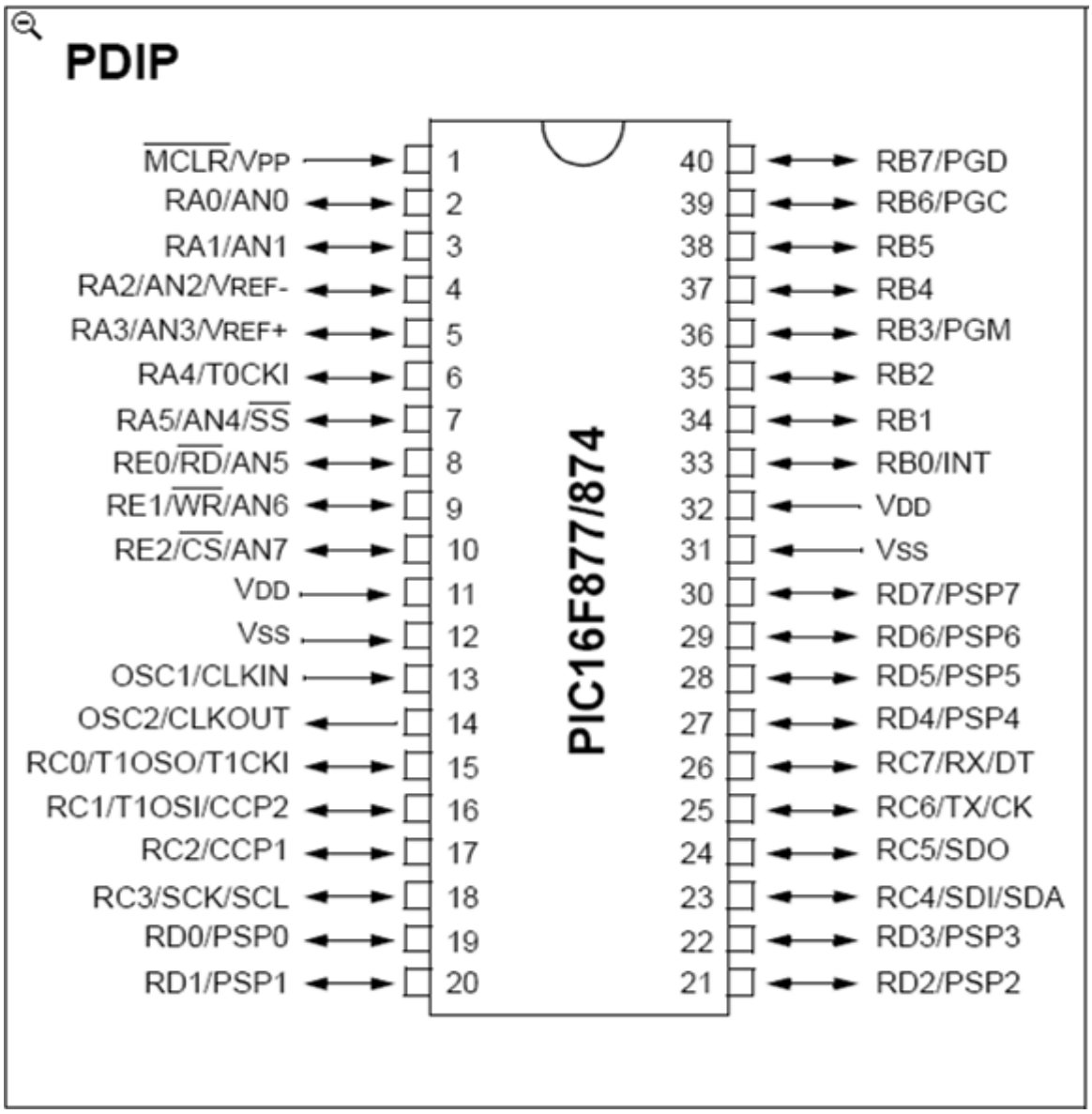
MICROCONTROLEURS

Famille Mid-Range
de Microchip

LE PIC 16F876/877

A. Oumnad



SOMMAIRE

I Introduction

Un microcontrôleur est un composant électronique *Autonome* doté :

- d'un microprocesseur,
- de la mémoire RAM,
- de la mémoire permanente,
- des interfaces d'E/S //, série (RS232,I2C, SPI …)
- des interfaces d'E/S analogique
- Des Timer pour gérer le temps
- D'autres module plus au moins sophistiqués selon la taille des µC

Il est généralement moins puissant qu'un microprocesseur en terme de rapidité ou de taille mémoire, il se contente le plus souvent d'un bus 8 ou 16 bits. Ceci en fait un composant très bon marché parfaitement Adapté pour piloter les applications embarquées dans de nombreux domaines d'application. Je pense qu'on ne se tromperait pas beaucoup si on affirme qu'aujourd'hui il y'a un microcontrôleur (± grand) dans chaque équipement électronique :

- Informatique (souris, modem …)
- Vidéo (Appareil photos numérique, caméra numérique …)
- Contrôle des processus industriels (régulation, pilotage)
- Appareil de mesure (affichage, calcul statistique, mémorisation)
- Automobile (ABS, injection, GPS, airbag)
- Multimédia (téléviseur,carte audio, carte vidéo, MP3, magnétoscope)
- Téléphones (fax, portable, modem)
- Electroménager (lave-vaisselle, lave-linge, four micro-onde)

Un microcontrôleur peut être programmé une fois pour toutes afin qu'il effectue une ou des tâches précises au sein d'un appareil électronique. Mais les µC récents peuvent être reprogrammés et ceci grâce à leur mémoire permanente de type FLASH (d'où le terme flasher quelque chose)

Plusieurs Constructeurs se partagent le marché des microcontrôleurs, citons INTEL, MOTOROLA, AMTEL, ZILOG, PHILIPS et enfin MICROCHIP avec ses PICs très populaires qui nous intéresse ici dans ce cours.

Les microcontrôleurs, quelque soit leurs constructeurs, ont des architecture très similaires et sont constitués de modules fondamentaux assurant les mêmes fonctions : UAL, Ports d'E/S, interfaces de communications série, Interfaces d'E/S analogiques, Timers et horloge temps réels …On peut dire que seul le langage de programmation (Assembleurs) constitue la différence majeure en deux microcontrôleur (similaires) venant de deux constructeurs différents.

Nous avons choisit dans ce cours d'apprendre les microcontrôleurs à travers une étude détaillée des microcontrôleur 16F87x (x=3, 4, 6, 7) qui constitue les éléments fondamentaux de la famille mid-range qui est la famille « moyenne puissance » de Microchip

I.1 Les PICs de Microchip

Les PICs sont des microcontrôleurs à architecture RISC (Reduce Instructions Construction Set), ou encore composant à jeu d'instructions réduit. L'avantage est que plus on réduit le nombre d'instructions, plus leur décodage sera rapide ce qui augmente la vitesse de fonctionnement du microcontrôleur.

La famille des PICs est subdivisée en 3 grandes familles : La famille **Base-Line**, qui utilise des mots d'instructions de 12 bits, la famille **Mid-Range**, qui utilise des mots de 14 bits (et dont font partie la 16F84 et 16F876), et la famille **High-End**, qui utilise des mots de 16 bits.

Les PICs sont des composants STATIQUES, Ils peuvent fonctionner avec des fréquences d'horloge allant du continu jusqu'à une fréquence max spécifique à chaque circuit. Un PIC16F876-04 peut fonctionner avec une horloge allant du continu jusqu'à 4 MHz.

Nous nous limiterons dans ce document à la famille Mid-Range et particulièrement au PIC 16F876/877, sachant que si on a tout assimilé, on pourra facilement passer à une autre famille, et même à un autre microcontrôleur.

PIC	FLASH	RAM	EEPROM	I/O	A/D	Port //	Port Série
16F870	2K	128	64	22	5	NON	USART
16F871	2K	128	64	33	8	PSP	USART
16F872	2K	128	64	22	5	NON	MSSP
16F873	4K	192	128	22	5	NON	USART/MSSP
16F874	4K	192	128	33	8	PSP	USART/MSSP
16F876	8K	368	256	22	5	NON	USART/MSSP
16F877	8K	368	256	33	8	PSP	USART/MSSP

Tableau I.1 : différents circuit de la famille 16F87X

Les éléments essentiels du PIC 16F876 sont :

- Une mémoire programme de type EEPROM flash de 8K mots de 14 bits,
- Une RAM donnée de 368 octets,
- Une mémoire EEPROM de 256 octets,
- Trois ports d'entrée sortie, A (6 bits), B (8 bits), C (8 bits),
- Convertisseur Analogiques numériques 10 bits à 5 canaux,
- USART, Port série universel, mode asynchrone (RS232) et mode synchrone
- SSP, Port série synchrone supportant I2C
- Trois TIMERS avec leurs Prescalers, TMR0, TMR1, TMR2
- Deux modules de comparaison et Capture CCP1 et CCP2
- Un chien de garde,
- 13 sources d'interruption,
- Générateur d'horloge, à quartz (jusqu' à 20 MHz) ou à Oscillateur RC
- Protection de code,
- Fonctionnement en mode sleep pour réduction de la consommation,
- Programmation par mode ICSP (*In Circuit Serial Programming*) 12V ou 5V,
- Possibilité aux applications utilisateur d'accéder à la mémoire programme,
- Tension de fonctionnement de 2 à 5V,
- Jeux de 35 instructions

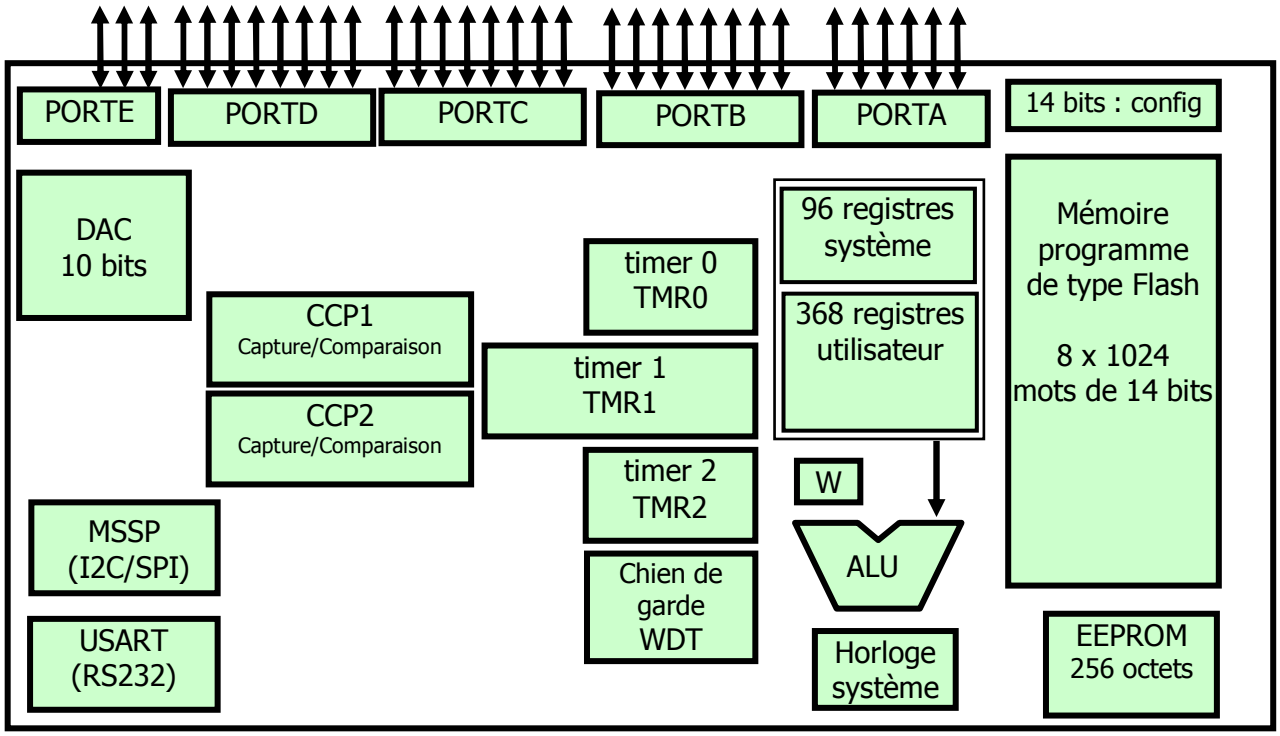


Fig. I.1 : Les éléments constitutifs du PIC 16F877

Le port D (8 bits) et le port E (3 bits) ne sont pas disponibles sur tous les processeurs. (Voir Tableau I.1)

II Les éléments de base du PIC 16F876/877

II.1 L'Horloge

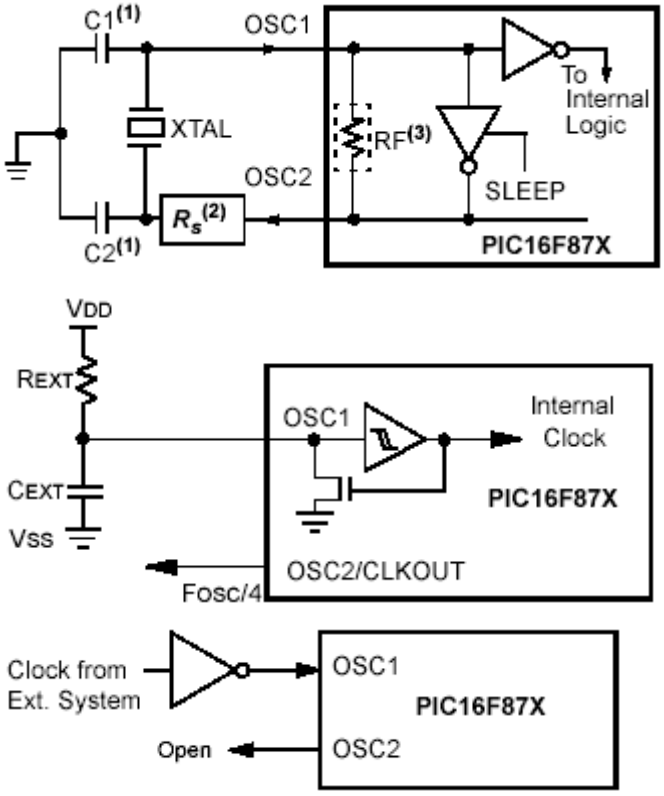
L'horloge peut être soit interne soit externe. L'horloge interne est constituée d'un oscillateur à quartz ou d'un oscillateur RC.

Avec l'oscillateur à Quartz, on peut avoir des fréquences allant jusqu'à 20 MHz selon le type de µC. Le filtre passe bas (Rs, C1, C2) limite les harmoniques dus à l'écrêtage et Réduit l'amplitude de l'oscillation, il n'est pas obligatoire.

Avec un oscillateur RC, la fréquence de l'oscillation est fixée par Vdd, Rext et Cext. Elle peut varier légèrement d'un circuit à l'autre.

Dans certains cas, une horloge externe au microcontrôleur peut être utilisée pour synchroniser le PIC sur un processus particulier.

Quelque soit l'oscillateur utilisé, l'horloge système dite aussi horloge instruction est obtenue en divisant la fréquence par 4. Dans la suite de ce document on utilisera le terme Fosc/4 pour désigner l'horloge système.



Avec un quartz de 4 MHz, on obtient une horloge instruction de 1 MHz, soit le temps pour exécuter une instruction de 1µs.

II.2 L'ALU et l'accumulateur W

L'ALU est une Unité Arithmétique et logique 8 Bits qui réalise les opérations arithmétiques et logique de base. L'accumulateur W est un registre de travail 8 bits, toutes les opérations à deux opérandes passe par lui. On peut avoir :

- Une instruction sur un seul opérande qui est en général un registre situé dans la RAM
- Une instruction sur 2 opérandes. Dans ce cas, l'un des deux opérandes est toujours l'accumulateur W, l'autre peut être soit un registre soit une constante.

Pour les instructions dont un des opérandes est un registre, le résultat peut être récupéré soit dans l'accumulateur, soit dans le registre lui-même.

II.3 Organisation de la mémoire RAM

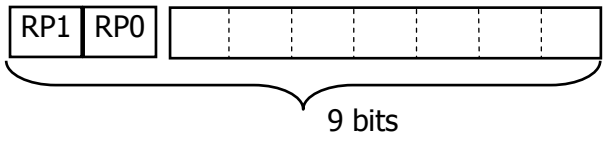
L'espace mémoire RAM adressable est de **512** positions de 1 octet chacune :

- 96 positions sont réservées au SFR (*Special Function Registers*) qui sont les registres de configuration du PIC.
- Les 416 positions restantes constituent les registres GPR (*General Propose Registers*) ou RAM utilisateur. Sur le 16F876 et 16F877, 3 blocs de 16 octets chacun ne sont pas implantés physiquement d'où une capacité de RAM utilisateur de 368 GPR.

Pour accéder à la RAM, on dispose de deux modes d'adressage :

II.3.1 Accès à la RAM par adressage DIRECT

Avec ce mode d'adressage, on précise dans l'instruction la valeur de l'adresse à laquelle on veut accéder. Par exemple, pour copier le contenu de l'accumulateur W dans la case mémoire d'adresse 50, on utilise l'instruction ***MOVWF 50***. Cette instruction sera codée sur 14 bits, la partie adresse est codée sur **7 bits** ce qui va poser quelques petits problèmes. En effet, 7 bits permettent d'adresser seulement 128 positions. Pour pouvoir adresser les 512 positions accessibles, il faut **9 bits** d'adresse. Pour avoir ces 9 bits, le PIC complète les 7 bits venant de l'instruction par deux bits situés dans le registre de configuration STATUS. Ces bits sont appelés RP0 et RP1 et doivent être positionnés correctement avant toute instruction qui accède à la RAM par l'adressage direct.



La RAM apparaît alors organisée en 4 banks de 128 octets chacun. L'adresse instruction permet d'adresser à l'intérieur d'un bank alors que les bits RP0 et RP1 du registre STATUS permettent de choisir un bank*. La* Figure II-1 *montre l'organisation de la RAM avec les zones allouée au SFR et aux GPR. Les zones hachurées ne sont pas implantées physiquement. Si on essaye d'y accéder, on est aiguillé automatiquement vers la zone [70_h,$7F_h$] appelée zone commune.*

Même si on précise une adresse supérieure à 127 (+ de 7 bits) dans une instruction, elle est tronquée à 7 bits puis complétée par les bits RP0 et RP1 pour former une adresse 9 bis. Par exemple, pour copier l'accumulateur W dans la case mémoire d'adresse $1EF_h$, il faut d'abord placer les bits RP0 et RP1 à 1 (bank 3), ensuite on utilise soit l'instruction **MOVWF 6Fh** soit l'instruction

MOVWF 1EFh, qui donne le même résultat. En effet, que l'on écrive 6Fh = 0110 1111 ou 1EFh = 0001 1110 1111, le PIC ne prend que 7 bits soit : 1101111 = 6Fh et complète avec les bits RP1,RP0 pour obtenir 11 1101111 = 1EFh

	Bank 0 (00)
00h	INDF
01h	TMR0
02h	PCL
03h	STATUS
04h	FSR
05h	PORTA
06h	PORTB
07h	PORTC
08h	PORTD (*)
09h	PORTE (*)
0Ah	PCLATH
0Bh	INTCON
0Ch	PIR1
0Dh	PIR2
0Eh	TMR1L
0Fh	TMR1H
10h	T1CON
11h	TMR2
12h	T2CON
13h	SSPBUF
14h	SSPCON
15h	CCPR1L
16h	CCPR1H
17h	CCP1CON
18h	RCSTA
19h	TXREG
1Ah	RCREG
1Bh	CCPR2L
1Ch	CCPR2H
1Dh	CCP2CON
1Eh	ADRESH
1Fh	ADCON0
20h 6Fh	
70h 7Fh	Zone commune

	Bank 1 (01)
80h	INDF
81h	OPTION_REG
82h	PCL
83h	STATUS
84h	FSR
85h	TRISA
86h	TRISB
87h	TRISC
88h	TRISD
89h	TRISE
8Ah	PCLATH
8Bh	INTCON
8Ch	PIE1
8Dh	PIE2
8Eh	PCON
8Fh	
90h	
91h	SSPCON2
92h	PR2
93h	SSPADD
94h	SSPSTAT
95h	
96h	
97h	
98h	TXSTA
99h	SPBRG
9Ah	
9Bh	
9Ch	
9Dh	
9Eh	ADRESL
9Fh	ADCON1
A0h EFh	
F0h FFh	

	Bank 2 (10)
100h	INDF
101h	TMR0
102h	PCL
103h	STATUS
104h	FSR
105h	
106h	PORTB
107h	
108h	
109h	
10Ah	PCLATH
10Bh	INTCON
10Ch	EEDATA
10Dh	EEADR
10Eh	EEDATH
10Fh	EEADRH
110h 16Fh	
170h 17Fh	

	Bank 3 (11)
180h	INDF
181h	OPTION_REG
182h	PCL
183h	STATUS
184h	FSR
185h	
186h	TRISB
187h	
188h	
189h	
18Ah	PCLATH
18Bh	INTCON
18Ch	EECON1
18Dh	EECON2
18Eh	
18Fh	
190h 1EFh	
1F0h 1FFh	

Figure II-1 : organisation de la RAM du 16F876/877

Nous allons anticiper un peu et présenter les instructions bcf et bsf et qui permettent de positionner un bit à 0 ou à 1

```
bcf     STATUS,RP0    ; place le bit RP0 à 0
bsf     STATUS,RP1    ; place le bit RP1  à 1
```

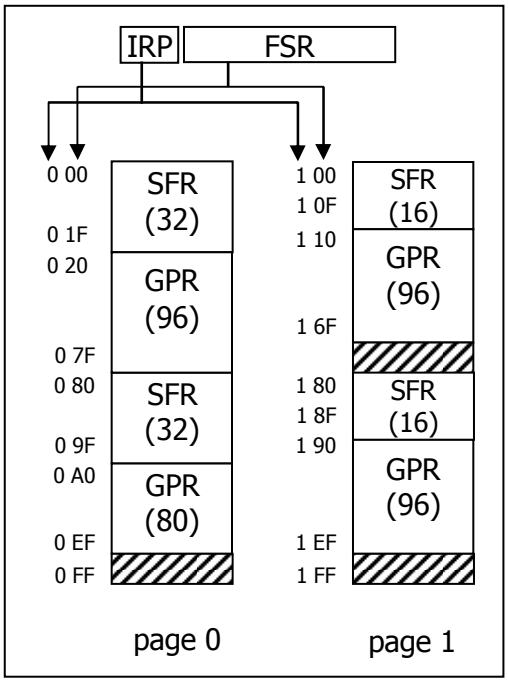
II.3.2 Accès à la RAM par l'adressage INDIRECT

Pour accéder à une position de la RAM en utilisant l'adressage indirect, on passe toujours par une position fictive appelée INDF *(Indirect File)*.
Exemple : l'instruction *CLRF INDF* signifie : mettre à zéro la case mémoire d'adresse INDF.
Mais quelle est l'adresse de cette position appelée INDF ? La réponse est :

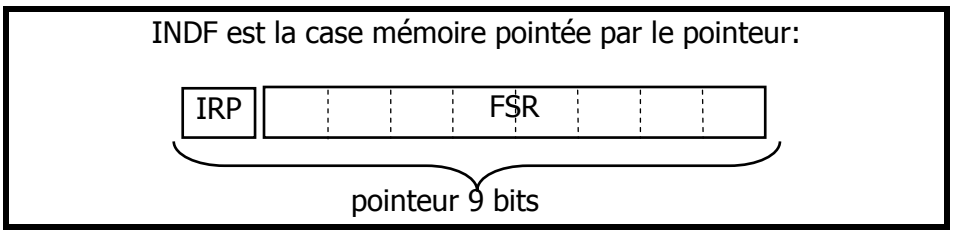
INDF est la case mémoire pointée par le pointeur IRP/FSR.

IRP est un bit qui se trouve dans STATUS et FSR est un registre accessible dans tous les bancs. On peut se demander pourquoi on ajoute le bit IRP. En effet, le registre de pointage FSR est un registre 8 bits, il peut donc adresser au maximum 256 positions mémoire (de 00h à FFh), c'est seulement la moitié de la RAM dont on dispose. Il nous manque un bit pour avoir les 9 bits nécessaires. On utilise le bit IRP qui se trouve dans le registre STATUS.

Exemle : Si on place 74h dans le registre FSR et on positionne le bit IRP à 1, alors, l'instruction *CLRF INDF* signifie : remettre à zéro la case mémoire d'adresse ***174h***.



Donc en résumé, chaque fois que le PIC rencontre le mot INDF dans un programme, il sait qu'il s'agit de la case mémoire dont l'adresse (9 bits) se trouve dans le registre FSR complété par le bit IRP du registre STATUS



IRP = 0 → 000h à 0FFh → page 0

IRP = 1 → 100h à 1FFh → page 1

II.4 Quelques registres de configuration et leurs bits

STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx
OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
PIE2	N.I.	Réservé	N.I.	EEIE	BCLIE	N.I.	N.I.	CCP2IE	-r-0 0--0
PIR2	N.I.	Réservé	N.I.	EEIF	BCLIF	N.I.	N.I.	CCP2IF	-r-0 0--0
EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
CCPxCON	—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0	--00 0000
T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000
SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000
SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF	0000 0000
CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000
ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0
ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000
TRISx									1111 1111

Tableau II.1 : détail des registres SFR et leurs états au démarrage

Exercice 1)

Dans quel bank se trouvent les cases mémoire d'adresse : 1A4h, B5h, 130h, 58h, 100, 200, 250, 300, 400

Exercice 2)

Combien de cases mémoires libres (GPR) y a-t-il dans la zone mémoire qui commence à la position *A0h* et se termine à *EAh*.

Exercice 3)

Quelle est l'adresse de la dernière position d'une zone mémoire de 40 cases qui commence à la position 190h.

Exercice 4)

Combien de cases mémoires libres (GPR) y a-t-il dans le bank1. Même question pour le bank2.

II.5 Les instructions du 16F876/877

- Tous les *PICs Mid-Range* ont un jeu de 35 instructions,
- Chaque instruction est codée sur un mot de 14 bits qui contient le code opération (OC) ainsi que l'opérande,
- Toutes les instructions sont exécutées en un cycle d'horloge, à part les instructions de saut qui sont exécutées en 2 cycles d'horloge. Sachant que l'horloge système est égale à fosc/4, si on utilise un quartz de 4MHz, on obtient une horloge fosc/4 = 1000000 cycles/seconde, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d' Instructions Par Seconde). Avec un quartz de 20MHz, on obtient une vitesse de traitement de 5 MIPS.

II.5.1 Les instructions « orientées Registre»

Ce sont des instructions qui manipulent un octet se trouvant dans la RAM. Ça peut être un registre de configuration SFR ou une case mémoire quelconque (Registre GPR)

II.5.2 Les instructions « orientées bits »

Ce sont des instructions destinées à manipuler directement un bit d'un registre que se soit un registre de configuration SFR ou une case mémoire quelconque (registre GPR). Tous les bits de la RAM peuvent être manipulé individuellement.

II.5.3 Les instructions opérant sur une constante

Ce sont les instructions entre l'accumulateur W est une constante K

II.5.4 Les instructions de saut et appel de procédures

Ce sont les instructions qui permettent de sauter à une autre position dans le programme et de continuer l'exécution du programme à partir de cette position.

II.5.5 Le jeu d'instructions

{W,F ? d} signifie que le résultat va soit dans W si d=0 ou w, soit dans F si d= 1 ou f

INSTRUCTIONS OPERANT SUR REGISTRE			**indicateurs**	**Cycles**
ADDWF	**F,d**	W+F → {W,F ? d}	C,DC,Z	1
ANDWF	**F,d**	W and F → {W,F ? d}	Z	1
CLRF	**F**	Clear F	Z	1
COMF	**F,d**	Complémente F → {W,F ? d}	Z	1
DECF	**F,d**	décrémente F → {W,F ? d}	Z	1
DECFSZ	**F,d**	décrémente F → {W,F ? d} skip if 0		1(2)
INCF	**F,d**	incrémente F → {W,F ? d}	Z	1
INCFSZ	**F,d**	incrémente F → {W,F ? d} skip if 0		1(2)
IORWF	**F,d**	W or F → {W,F ? d}	Z	1
MOVF	**F,d**	F → {W,F ? d}	Z	1
MOVWF	**F**	W → F		1
RLF	**F,d**	rotation à gauche de F a travers C → {W,F ? d}	C	1
RRF	**F,d**	rotation à droite de F a travers C → {W,F ? d}		1
SUBWF	**F,d**	F – W → {W,F ? d}	C,DC,Z	1
SWAPF	**F,d**	permute les 2 quartets de F → {W,F ? d}		1
XORWF	**F,d**	W xor F → {W,F ? d}	Z	1

INSTRUCTIONS OPERANT SUR BIT				
BCF	**F,b**	RAZ du bit b du registre F		1
BSF	**F,b**	RAU du bit b du registre F		1
BTFSC	**F,b**	teste le bit b de F, si 0 saute une instruction		1(2)
BTFSS	**F,b**	teste le bit b de F, si 1 saute une instruction		1(2)

INSTRUCTIONS OPERANT SUR CONSTANTE				
ADDLW	**K**	W + K → W	C,DC,Z	1
ANDLW	**K**	W and K → W	Z	1
IORLW	**K**	W or K → W	Z	1
MOVLW	**K**	K → W		1
SUBLW	**K**	K – W → W	C,DC,Z	1
XORLW	**K**	W xor K → W	Z	1

AUTRES INSTRUCTIONS				
CLRW		Clear W	Z	1
CLRWDT		Clear Watchdoc timer	TO', PD'	1
CALL	**L**	Branchement à un sous programme de label L		2
GOTO	**L**	branchement à la ligne de label L		2
NOP		No operation		1
RETURN		retourne d'un sous programme		2
RETFIE		Retour d'interruption		2
RETLW	**K**	retourne d'un sous programme avec K dans W		2
SLEEP		se met en mode standby	TO', PD'	1

On remarque que :

- Les instructions qui agissent sur un registre ou un bit d'un registre contiennent toutes la lettre F dans le nom de l'instruction. Ceci vient du fait que chez Microchip, la RAM est appelée *register File* (**F**ichier des registres).
- Les instructions qui agissent sur une constante contiennent toutes la lettre L, parce que chez Microchip, on appelle « *Literal* » ce genre d'adressage, chez d'autres constructeurs, on parle d'adressage immédiat

II.5.6 Les paramètres des instructions agissant sur registre

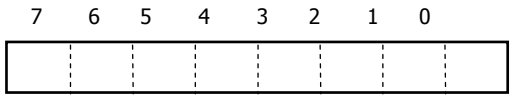
Pour les instructions qui agissent sur registre, le paramètre F représente l'adresse du registre considéré. Le paramètre d (destination) joue un rôle important, si on prend *d = 0 ou w*, le résultat de l'opération sera placé dans l'accumulateur W, si on prend *d = 1* ou *f*, le résultat de l'opération sera placé dans le registre précisé par F.

ADDWF 70h,1 ou ADDWF 70h,f
Signifie : additionner le contenu de W avec le contenu de la case mémoire d'adresse 70h et placer le résultat dans la case mémoire 70h

XORWF 35h,0 ou XORWF 35h,w
Signifie : faire un ou exclusif entre W et le contenu de la case mémoire d'adresse 35h et placer le résultat dans l'accumulateur W

II.5.7 Les paramètres des instructions agissant sur bit

Pour les instructions agissant sur un bit, le paramètre F indique le registre qui contient le bit à modifier et le paramètre b indique le numéro du bit à modifier; on compte à partir de zéro en commençant à droite



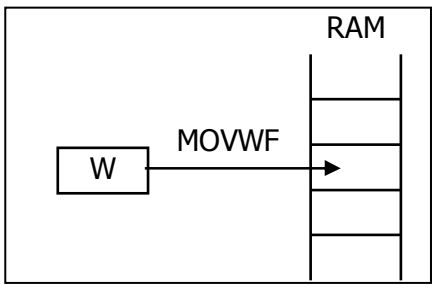
BSF STATUS,2 ; signifie : placer à 1 le bit 2 (3ème bit à partir de la droite) du registre STATUS

BCF 45h,6 ; signifie : placer à 0 le bit 6 (7ème bit à partir de la droite) du registre de la case mémoire d'adresse 45h

II.5.8 Les instructions MOVWF et MOVF

Ce sont les instructions les plus utilisées,

MOVWF permet de copier l'accumulateur W dans un registre (SFR ou GPR):



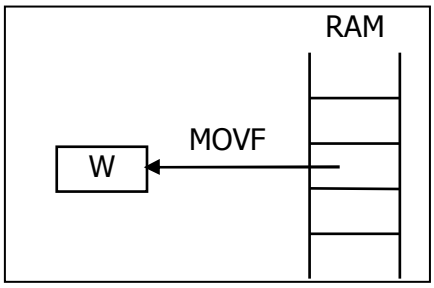
MOVWF STATUS ; signifie : Copier le contenu de W dans le registre STATUS

MOVWF 55h ; signifie : Copier le contenu de W dans la case mémoire d'adresse 55h

MOVF permet de copier le contenu d'un registre (SFR ou GPR) dans l'accumulateur W, le paramètre d doit être = 0

MOVF STATUS,0 ; Copier le contenu du registre STATUS dans l'accumulateur W

MOVF 35h,0 ; Copier le contenu de la case mémoire d'adresse 35h dans l'accumulateur W



Avec le paramètre *d=1*, l'instruction MOVF semble inutile car elle permet de copier un registre sur lui-même ce qui à priori ne sert à rien.

MOVF STATUS,1 ; Copier le contenu du registre STATUS dans lui même

En réalité cette instruction peut s'avérer utile car, comme elle positionne l'indicateur Z, elle permet de tester si le contenu d'un registre est égal à zéro

II.5.9 Les instructions btfss et btfsc

Ces instructions permettent de tester un bit et de sauter ou non une ligne de programme en fonction de la valeur du bit,

btfsc *F,b* : **bit test skip if clear** : teste le bit b du registre F et saute l'instruction suivante si le bit testé est nul

btfss *F,b* : **bit test skip if set** : teste le bit b du registre F et saute l'instruction suivante si le bit testé est égal à 1

exemple :

```
sublw   100          ; 100 – W → W
btfss   STATUS,Z     ; tester le bit Z du registre STATUS et sauter une ligne si Z=1
clrf    70h          ; après btfss, le programme continue ici si Z=0
cmpf    70h,f        ; après btfss, le programme continue ici si Z=1
suite du programme
suite du programme
…
```

II.5.10 Les instructions incfsz et decfsz

Ces instructions permette d'incrémenter ou de décrémenter un registre et de sauter si le résultat est nul

Incfsz *F,1* : **increment skip if Z** : incrémente le registre F et sauter une ligne si le résultat = 0. Le paramètre 1 indique que le résultat de l'incrémentation doit aller dans F.

deccfsz *F,1* : **decrement skip if Z** : décrémente le registre F et sauter une ligne si le résultat = 0. Le paramètre 1 indique que le résultat de la décrémentation doit aller dans F.

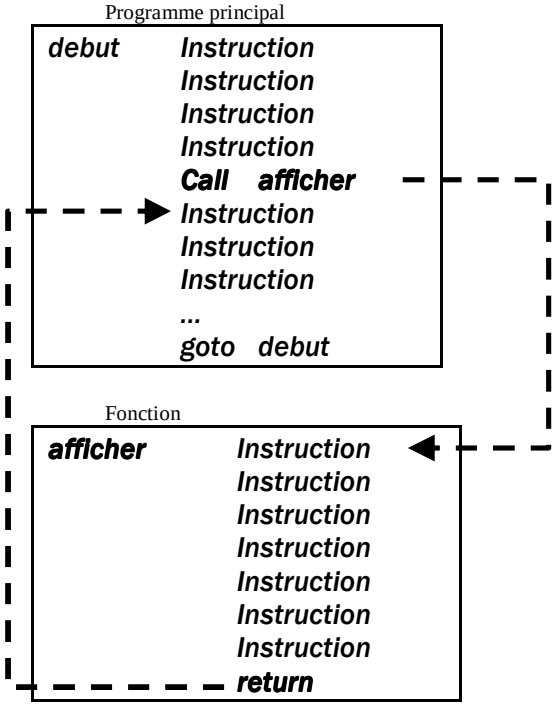
II.5.11 L'instruction goto

Permet de transférer l'exécution à une autre position du programme repérée par une ***étiquette*** (label)

```
                Instruction 1
                Instruction 2
                Goto      bonjour
                instruction 3
                instruction 4
                instruction 5
bonjour         instruction 6
                instruction 7
```

II.5.12 L'instruction call

L'instruction ***call*** permet d'appeler une fonction. Une fonction est un sous programme écrit à la suite du programme principal. Sa première ligne doit comporter une étiquette et elle doit se terminer par ***return***



La différence en ***call*** et ***goto*** est que, quant le processeur rencontre l'instruction ***call***, il sauvegarde l'adresse de la ligne suivante avant d'aller exécuter les instructions constituant la fonction. Comme ça, quand il rencontre l'instruction ***return***, il sait où il doit retourner pour continuer l'exécution du programme principal.

II.5.13 Les indicateur d'état (drapeaux)

Les bits Z, DC et C situés dans le registre STATUS sont des indicateurs qui permettent de savoir comment une instruction s'est terminée. Toutes les instructions n'agissent pas sur les indicateurs, voir liste des instructions ci-dessous.

Z : passe à 1 quand le résultat d'une instruction est nul
C : passe à 1 quand l'opération a généré une retenue
DC : passe à 1 quand les 4ème bits génère une retenue

Ces bits peuvent être utilisé très astucieusement par les instructions ***btfsc*** et ***btfss*** qui permettent de tester un bit et de réaliser un saut conditionnel. Nous aurons l'occasion d'en reparler dans la suite du cours.

STATUS	IRP	RP1	RP0			Z	DC	C

II.5.13.1 Les indicateurs, la soustraction et la comparaison

Les instructions SUBWF et SUBLW positionne les drapeaux Z et C. Remarquons seulement que la retenue B (Borrow) de la soustraction correspond à C,

F - W = 0 ==> Z=1 , C=1 , B=0 => pas de retenue de soustraction
F - W > 0 ==> Z=0 , C=1 , B=0 => pas de retenue de soustraction
F - W < 0 ==> Z=0 , C=0 , B=1 => il ya retenue de soustraction

Pour réaliser une comparaison entre F et W, on fait F – W et on observe Z et C

- Z=1 ==> égalité
- C=1 ==> F sup ou égal à W
- C=0 ==> F inférieur à W

II.6 Les directives de l'assembleur MPASM

Les directives de l'assembleur sont des instructions qu'on ajoute dans le programme et qui seront interprétées par l'assembleur MPASM. Ce ne sont pas des instructions destinées au PIC. Nous ne présentons ici que quelques directives, pour le reste, consulter la documentation de MPASM *"MPASM USER'S GUIDE".*

Remarque : Je vais présenter les directives en Majuscule, mais sachez que le compilateur MPASM accepte les instructions et les directives en minuscule et en majuscule. Attention ce n'est pas le cas pour les étiquettes et les noms de déclaration des constantes.

- **LIST** : permet de définir un certain nombre de paramètres comme le processeur utilisé (p), la base par défaut pour les nombres (r) ainsi que d'autres paramètres. Exemple :
  LIST p=16F876, r=dec

avec r=dec, les nombres sans spécification particulière seront considérés par l'assembleur comme des nombre décimaux, sinon voir tableau ci-contre

Base	Préfixe	Exemple (36)
Décimal	D'nnn' .nnn	D'36' .36
Hexadécimal	H'nn' 0xnn nnh	H'24' 0x24 24h
Binaire	B'….'	B'00100100'
Octal	O'nnn'	O'44'

- **INCLUDE** : permet d'insérer un fichier source. Par exemple le fichier p16f876.inc contient la définition d'un certain nombre de constante comme les noms des registres ainsi que les noms de certains bits;
  INCLUDE "p16f876.inc"

- **EQU** : permet de définir une constante ou une variable :
  XX EQU 0x20
  Chaque fois que le compilateur rencontrera XX, il la remplacera par 0x20. Ça peut être une constante s'il s'agit d'une instruction avec adressage *Literal*, ou d'une adresse s'il s'agit d'une instruction avec adressage direct.

```
MOVLW     XX    ; charger dans W la constante 0x20
MOVF      XX,w  ; charger dans W le contenu de la case d'adresse 0x20
```

- **CBLOCK/ENDC** : permet de définir un ensemble de constantes :

  L'ensemble des déclarations suivantes :

```
XX1   EQU   0x20
XX2   EQU   0x21
XX3   EQU   0x22
XX4   EQU   0x23
```

  peut être remplacé par

```
CBLOCK   0x20
 XX1, XX2, XX3, XX4
ENDC
```

- **ORG** : définit la position dans la mémoire programme à partir de laquelle seront inscrites les instructions suivantes.

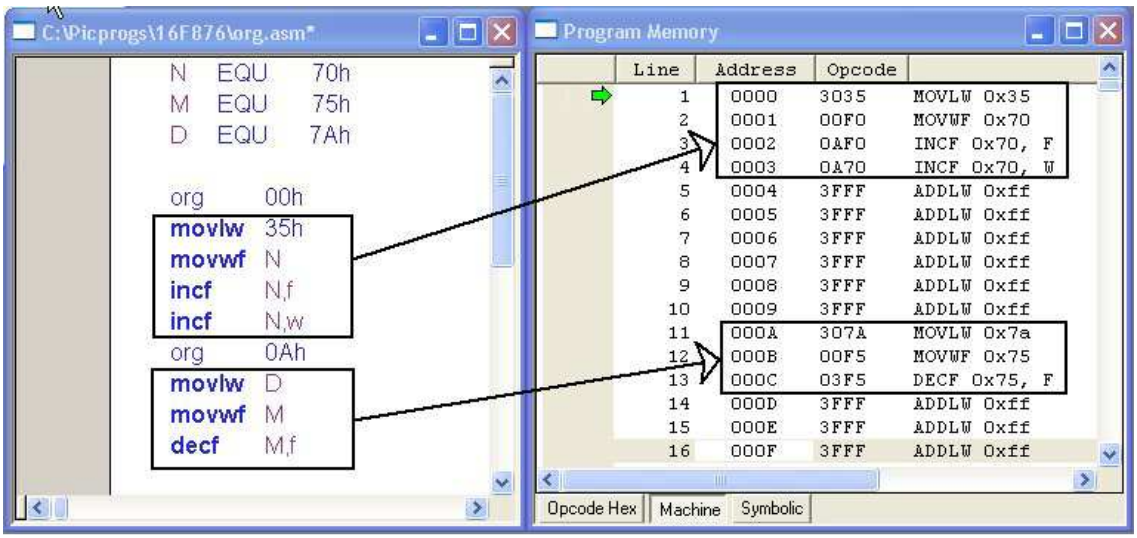


Fig. II.1 : Illustration de la directive ORG

- **#DEFINE** : fonctionne un peu comme la directive EQU tout en étant un peu plus générale, car elle permet d'affecter toute une chaîne à une abréviation

  #DEFINE XX 0x20 ; dans ce cas c'est équivalent à *XX EQU 0x20*

  #DEFINE LED PORTB,3 ; ici chaque fois que le compilateur rencontrera le mot *LED,* il le remplacera par *PORTB,*3

  BCF LED ; éteindre la LED branchée sur la broche 3 de PORTB

On peu ainsi affecter une abréviation à toute une instruction

#DEFINE eteindre bcf PORTC,5 ;affecte l'abréviation *eteindre* à l'instruction *bcf PORTC,5*

#DEFINE allumer bsf PORTC,5 ;affecte l'abréviation *allumer* à l'instruction *bsf PORTC,5*

allumer ; allume la LED branchée sur la broche 5 de PORTC
eteindre ; éteindre la LED branchée sur la broche 5 de PORTC

- **DE** : pour déclarer des donnés qui seront stockée dans l'EEPROM de donnée au moment de l'implantation du programme sur le PIC
  ORG 0x2100
  DE "Programmer un PIC, rien de plus simple", 70, 'Z'

- **DT** : pour déclarer un tableau RETLW
  DT 10,35h,'Hello' ; sera remplacée par la suite d'instructions :

  RETLW 10
  RETLW 35h
  RETLW 'H'
  RETLW 'e'
  RETLW 'l'
  RETLW 'l'
  RETLW 'o'

- **END** : indique la fin du programme

- **__CONFIG** : permet de définir les 14 bits (fusibles ou *switch*) de configuration qui seront copiés dans l'EEPROM de configuration (adresse 2007h) lors de l'implantation du programme dans le PIC.

CP1	CP0	DEBUG	—	WRT	CPD	LVP	BODEN	CP1	CP0	PWRTE	WDTE	F0SC1	F0SC0

CP1/CP0 : bits 13/12 ; Déterminent quelle zone de la mémoire programme sera protégée contre la lecture externe (via ICSP) ou l'écriture par programme conformément à l'état du bit 9 (WRT). On peut choisir de protéger la totalité de la mémoire ou seulement une partie. Les différentes zones pouvant être protégées sont les suivantes :
1 1 : Aucune protection (_CP_OFF)
1 0 : Protection de la zone 0x1F00 à 0x1FFF (_CP_UPPER_256)
0 1 : Protection de la zone 0x1000 à 0x1FFF (_CP_HALF)
0 0 : Protection de l'intégralité de la mémoire (_CP_ALL)

DEBUG : bit 11 : Debuggage sur circuit. Permet de dédicacer RB7 et RB6 à la communication avec un debugger.
1 : RB6 et RB7 sont des I/O ordinaires (_DEBUG_OFF)
0 : RB6 et RB7 sont utilisés pour le debuggage sur circuit (_DEBUG_ON)

WRT : bit 9 : Autorisation d'écriture en flash
1 : Le programme peut écrire dans les zones non protégées par les bits CP1/CP0 (_WRT_ENABLE_ON)
0 : Le programme ne peut pas écrire en mémoire flash (_WRT_ENABLE_OFF)

CPD : bit 8 : Protection en lecture de la mémoire EEPROM de données.
1 : mémoire EEPROM non protégée (_CPD_OFF)
0 : mémoire EEPROM protégée contre la lecture externe via ICSP (_CPD_ON)

LVP : bit 7 : Utilisation de la pin RB3/PGM comme broche de programmation 5V
1 : La pin RB3 permet la programmation du circuit sous tension de 5V (_LVP_ON)
0 : La pin RB3 est utilisée comme I/O standard (_LVP_OFF)

BODEN : bit 6 : provoque le reset du PIC en cas de chute de tension (surveillance de la tension d'alimentation)
1 : En service (_BODEN_ON)
0 : hors service (_BODEN_OFF)

PWRTE : bit 3 : Délai de démarrage à la mise en service. Attention, est automatiquement mis en service si le bit BODEN est positionné.
1 : délai hors service (sauf si BODEN = 1) (_PWRTE_OFF)
0 : délai en service (_PWRTE_ON)

WDTE : bit 2 : Validation du Watchdog timer
1 : WDT en service (_WDT_ON)
0 : WDT hors service (_WDT_OFF)

FOSC1/FOSC0 : bits 1/0 : sélection du type d'oscillateur
11 : Oscillateur de type RC (_RC_OSC) (3K < R < 100k, C > 20 pF)
10 : Oscillateur haute vitesse (_HS_OSC) (4 Mhz à 20 Mhz)
01 : Oscillateur basse vitesse (_XT_OSC) (200 kHz à 4 Mhz)
00 : Oscillateur faible consommation (_LP_OSC) (32 k à 200 kHz)

Voici 3 exemples d'utilisation :

__CONFIG B'11111100111001'
__CONFIG H'3F39'
__CONFIG _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_ON & _CPD_OFF & _LVP_OFF & _BODEN_OFF & _PWRTE_OFF & _WDT_OFF & _XT_OSC

Remarque :
Attention, les microcontrôleurs avec l'extension « A » comme 16F876A ou 16F877A ont une disposition des bits de configuration légèrement différente, consultez le datasheet.

III Les outils de développement

L'outil de développement principal est l'environnement de développement intégré MPLAB fournit gratuitement par Microchip

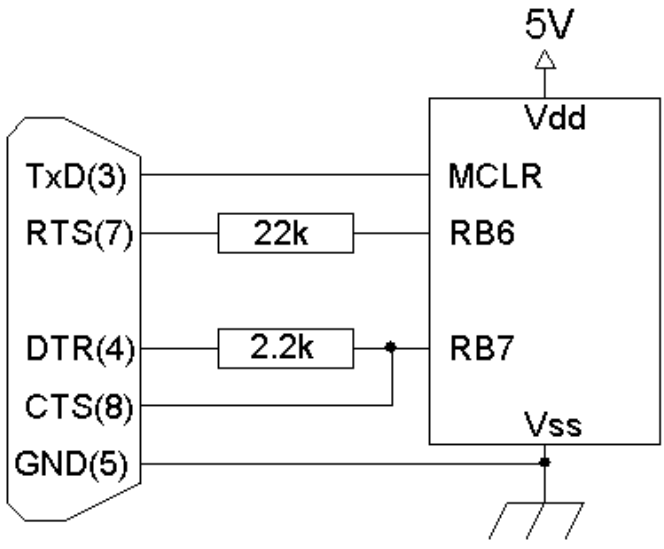
III.1 Procédure de travail

Les étapes nécessaires permettant de voir un programme s'exécuter sur un PIC sont :

- Ecrire un programme en langage assembleur dans un fichier texte et le sauvegarder avec l'extension **.asm**
- Compiler ce programme avec l'assembleur MPASM fourni par Microchip. Le résultat est un fichier exécutable avec l'extension **.hex** contenant une suite d'instruction compréhensible par le pic.
- Transplanter le fichier **.hex** dans la mémoire programme du PIC (mémoire flash) à l'aide d'un programmateur adéquat. On peut utiliser les programmateurs de Microchip ou tout autre programmateur acheté ou réalisé par soit même.
- Mettre le PIC dans son montage final, mettre sous tension et admirer le travail.

Microchip propose gratuitement l'outil de développement **MPLAB** qui regroupe l'éditeur de texte, le compilateur MPASM, un outil de simulation et le logiciel de programmation. Le programmateur lui-même, n'est malheureusement pas gratuit.

Pour ce qui nous concerne, nous utiliseront MPLAB pour écrire, compiler et éventuellement simuler nos programmes, ensuite, pour implanter les programmes dans la mémoire flash du PIC, nous utiliserons un programmateur fait maison que nous piloteront par le logiciel ICPROG, les deux sont disponibles gratuitement sur le Web.



Programmeur à 2 résistances

Internet est riche en informations sur ce sujet. Vous trouverez une rubrique sur les programmateurs de PIC sur mon site http://z.oumnad.123.fr

III.2 L'environnement de développement MPLAB

MPLAB-IDE peut être téléchargé sur le site Web http://www.microchip.com

Après l'installation, lancer MPLAB et faire les config ci-dessous :

- Configure → Select Device → PIC16F876 ou PIC16F877

Nous allons réaliser un tout petit programme sans grand intérêt pour voir la procédure de fonctionnement (avec MPLAB 6.30)

- Ouvrir une nouvelle fenêtre (de l'éditeur) pour commencer à écrire un programme : **file → new** ou cliquez sur l'icône feuille blanche
- Taper le petit programme ci-dessous dans la fenêtre qui vient de s'ouvrir. Ce programme incrémente sans fin la position mémoire (RAM) 70_H

```
loop    incf 70h,1
        goto looop
        end
```

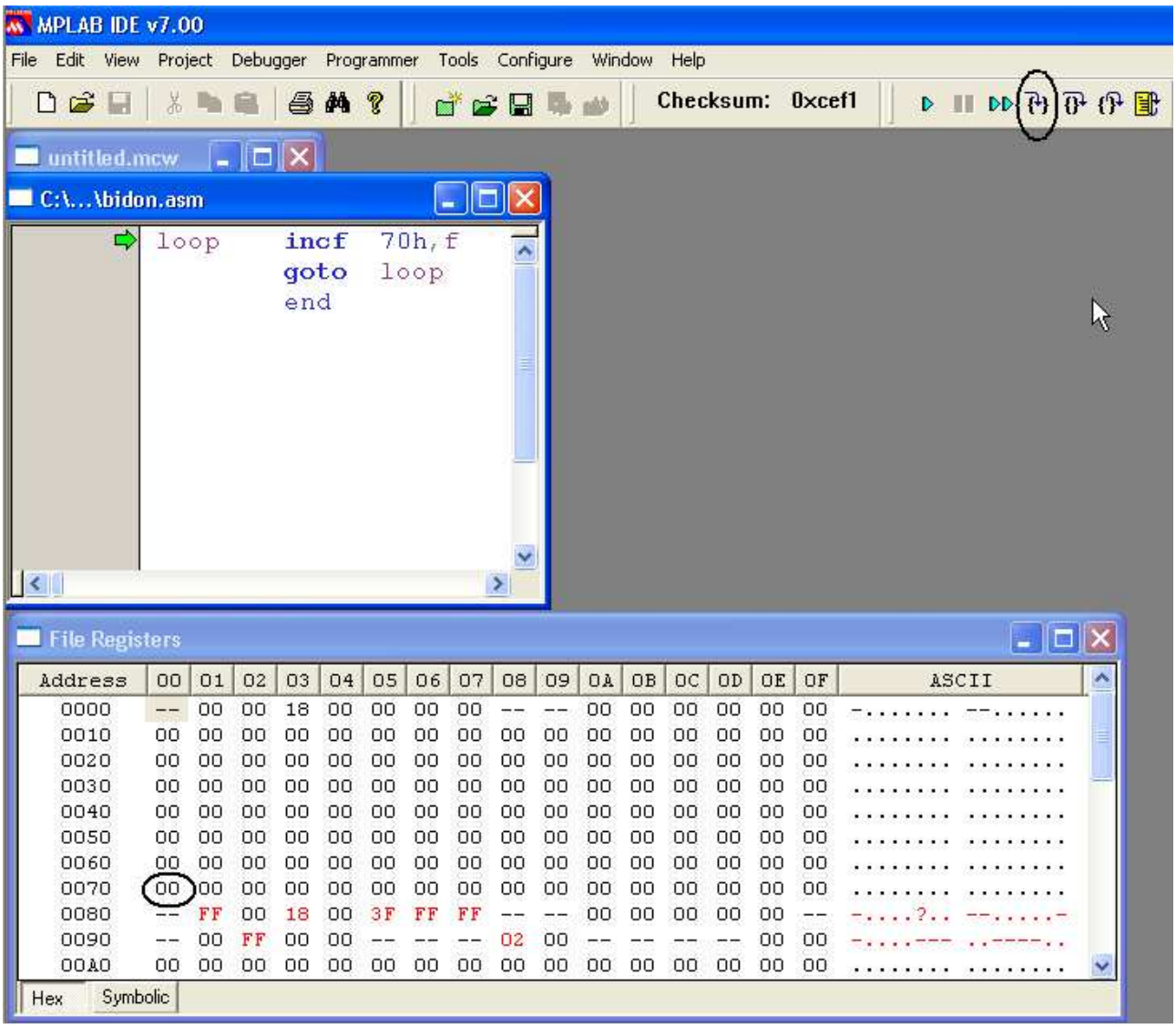
- Sauvegarder (file → save) ce programme dans la directory de votre chois sous le nom *bidon.asm*
- Lancer la compilation du programme à l'aide de la commande ***project → Quikbuild***
  Apparemment il y a un problème, le compilateur nous dit qu'il y une erreur à la ligne 2

  Error[113] C:\...\BIDON.ASM 2 : Symbol not previously defined (looop)

  *E*videmment, le label loop définit dans la ligne précédente prend seulement deux o. Corrigez et recommencez. Cette fois ça a l'air d'aller. On peut vérifier que le compilateur a crée le fichier *bidon.hex* dans la même directory où se trouve *bidon.asm*. Les fichiers *bidon.cod, bidon.err et bidon.lst* ne nous servent à rien pour l'instant on peut les détruire.

- Nous pouvons maintenant exécuter notre programme en simulation pour voir s'il réalise bien la tache demandée :
  Pour faire apparaître la barre d'outil du simulateur :
  Debugger → Select tool → MPLAB SIM
- Ouvrez la fenêtre qui visualise la mémoire RAM : view → *FileRegisters*. et repérer la case mémoire 70h
- Exécuter maintenant le programme PAS à PAS en cliquant à chaque fois sur le bouton Step Into **{↴}** en observant la case mémoire 70h . (on dirait que ça marche).
- On peut aussi exécuter en continu en cliquant sur le bouton animate ⏩ , pour arrêter, il faut cliquer sur le bouton halt ⏸

Pour plus de détail, consulter le manuel d'utilisation de MPLAB

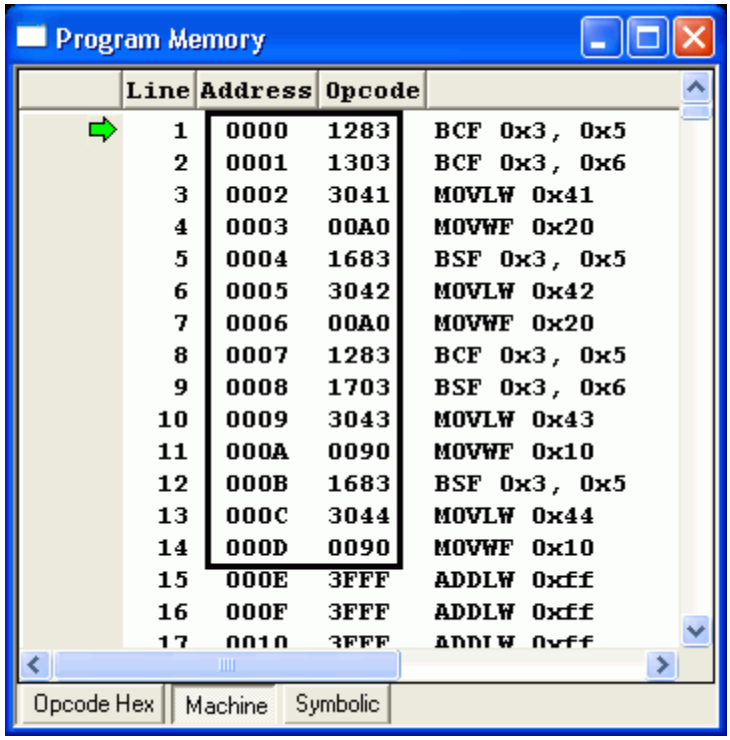


Modifiez maintenant la case 70h par la case 190h, compilez, simulez et tirer les conclusions

III.3 Structure générale d'un programme

```
;*****************************************************************
; Ce programme écrit les lettres A, B, C, D dans les positions suivantes de la RAM
; A  =>  20h                B => A0h              C => 110h            D => 190h
;*****************************************************************

        list p=16f877 , r=dec
        include <p16f877.inc>
        __CONFIG  _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_OFF &
                  _LVP_OFF & _BODEN_OFF & _PWRTE_OFF & _WDT_OFF & _XT_OSC

        bcf             STATUS,RP0      ;bank 0
        bcf             STATUS,RP1      ;bank 0
        movlw           'A'
        movwf           20h

        bsf             STATUS,RP0      ;bank 1
        movlw           'B'
        movwf           0xA0

        bcf             STATUS,RP0      ;bank 2
        bsf             STATUS,RP1      ;bank 2
        movlw           'C'
        movwf           110h

        bsf             STATUS,RP0      ;bank 3
        movlw           'D'
        movwf           190h

        end
```

Compilons le programme avec MPLAB *(project→quikbuild)* et visualisons la mémoire programme *(view→program memory)* , on obtient la fenêtre ci-dessus.



Observations :

- Comme il n'y a pas d'interruption, on ne s'est pas soucié de l'implantation du programme dans la mémoire programme (directive ORG). on remarque sur la fenêtre que l'implantation du programme commence à l'adresse 0.

- La colonne de droite de cette fenêtre obtenue par cross-assemblage du code machine n'utilise pas les noms des registres mais leurs adresses. Si nous avons pu utiliser les instruction comme : bcf STATUS,RP0, au lieu de bcf 0x3,0x5 c'est parce que le fichier *p16f876.inc* que nous avons inclus contient les déclarations *STATUS EQU 0x3* et *RP0 EQU 0x5*

- On remarque aussi que l'instruction *movwf 190h* a été remplacée par *movwf 0x10*, c'est tout à fait normal car on ne retient que 7 bits de l'adresse 190h = 0001 1001 0000 ce qui donne 001 0000 = 10h. Rassurons nous, c'est quand même l'adresse 190h qui sera adressée l'adresse 001 0000 sera complétée par RP1,RP0=11 (bank 3) et on obtient 11001 0000 = 190h

III.4 Quelques exemples

Exercice 5) : Accès à la RAM par l'adressage direct

Donner le programme qui copie :

35 dans la position 20h, 'A' dans la position A0h
-5 dans la position 110h, 35h dans la position 190h

Exercice 6) : Accès à la RAM par l'adressage indirect

Donner le programme qui copie l'alphabet majuscule dans la RAM à partir de la position 190h

Exercice 7) : Soustraction

Donner le programme qui :

a) soustrait la constante 33 de l'accumulateur W (W-33)

b) qui soustrait le contenu de la case mémoire 70h de l'accumulateur W avec le résultat dans W. (W – [70h] → W)

Exercice 8) : (comp1.asm)

Comparer les contenus des cases mémoire 6Fh et EFh, s'il son égaux mettre à zéro tous les bits de la case 16Fh sinon mettre à 1 tous les bits de la case 1EFh

Exercice 9) : (comp2.asm)

Comparer les contenus des cases mémoire 6Fh et EFh,

- si [6Fh] = [EFh] copier la lettre E dans la case mémoire 16Fh
- si [6Fh] > [EFh] copier la lettre S dans la case mémoire 16Fh
- si [6Fh] < [EFh] copier la lettre I dans la case mémoire 16Fh

III.4.1 Boucles de temporisation

Il arrive souvent qu'on désire introduire des temporisations pendant l'exécution d'un programme. Le PIC dispose de 3 *timers* permettant de gérer le temps avec précision. Nous étudierons ces modules plus tard, pour l'instant regardons comment on peut réaliser des temporisations à l'aide de simples boucles. L'idée est d'initialiser une variable à une valeur donnée et ensuite la décrémenter en boucle jusqu'à ce qu'elle atteint 0. Connaissant le temps d'exécution de chaque instruction, on peut calculer le temps que mettra le processeur terminer la boucle de décrémentation

III.4.1.1 Temporisation avec une boucle

Examinons l'exemple ci-dessous, on met une valeur N1 dans la case mémoire 70h et on la décrémente jusqu'à 0

```
        movlw   4
        movwf   70h
ici     decfsz  70h,f
        goto    ici
```

- Les instructions *movlw* et *movwf* prennent 1 cycle chacune
- L'instruction *decfsz* prend un cycle si elle ne saute pas et 2 cycles quand elle saute
- L'instruction *goto* prend 2 cycles
- chaque passage dans la boucle prend (1+2) cycle sauf le dernier qui prend 2 cycle

$$T = 2 + (W-1)*3 + 2 = 3*W + 1 \text{ cycles}$$

La valeur max que l'on peut donner à W est = 0 = 256, ce qui donne une temporisation max de 769 cycles.

Avec un quartz = fosc = 4 Mhz, 1 cycle = fosc/4 = 1 µs, ce qui donne une temporisation max de 769 µs

Pour faciliter l'utilisation de cette temporisation on va l'utiliser comme une fonction que l'on appellera ***tempo1.*** On note AN1 la case mémoire qui sert de compteur, il faut la déclarer au début avec la directive ***EQU***

```
tempo1:     ; il faut placer une valeur dans W  avant d'appeler cette fonction
        movwf   AN1
t1:     decfsz  AN1,f
        goto    t1
        return
```

Cette fonction doit être placée après le programme principal, et on l'appelle avec l'instruction ***call tempo1***

Pour le calcul il faut rajouter 2 cycles pour l'instruction ***call*** et 2 cycles pour l'instruction ***return***, ce qui donne

$$T1 = 3\ W + 5 \text{ cycles}$$

Le maximum est obtenu pour N1 =256, soit T1max = 773 µs = 0.77 ms

III.4.1.2 Temporisation avec 2 boucles imbriquées

La boucle intérieure (N1) se fait toujours 256 fois. La boucle extérieure se fait N2 fois. C'est N2 qui constituera le paramètre de la fonction, Il faut le placer dans W avant de l'appeler.

```
tempo2 :    ; il faut définir W  dans le programme principal
            movwf     AN2
t2 :        decfsz    AN1,f
            goto      t2
            decfsz    AN2,f
            goto      t2
            return
```

T2 = 2 + 2 + [W(t1+3)-1] + 2 avec t1=256*3-1=767 on obtient :

T2 = 770 W +5 cycles

Le maximum est obtenu pour N2 =256, soit T2max = 197125 µs = 0.197 s

III.4.1.3 Temporisation avec 3 boucles imbriquées

Les boucles intérieures (N1 et N2) se font toujours 256 fois. La boucle extérieure se fait N3 fois. C'est N3 qui constituera le paramètre de la fonction, Il faut le placer dans W avant de l'appeler.

```
tempo3 :    ;Il faut définir W dans le programme principal
            movwf     AN3
t3 :        decfsz    AN1,f
            goto      t3
            decfsz    AN2,f
            goto      t3
            decfsz    AN3,f
            goto      t3
            return
```

T3 = 2 + 2 + W(t2+3)-1 + 2 avec t2=256*(767+3)-1 on obtient :

T3 = 197122 W +5 cycles

Le maximum est obtenu pour N3 =256, soit T3max = 50463237 µs = 50.46 s

Remarque : La précision de ces fonctions peut être améliorée en y insérant des instructions ***nop***, dans ce cas il faut revoir les formules. On verra plus tard comment on peut faire des temporisations à l'aide des timers.

IV Les ports d'E/S

Le PIC 16F877 dispose de 33 broches d'entrée sortie regroupés dans 5 ports PORTA, PORTB, PORTC, PORTD et PORTE. Chaque broche d'un port peut être configurée soit en entrée soit en sortie à l'aide des registres de direction TRISA, TRISB, TRISC et TRISD et TRISE:

Bit *k* de TRISx = *0* ➔ broche *k* de PORTx = *SORTIE*
Bit *k* de TRISx = *1* ➔ broche *k* de PORTx = *ENTRÉE*

Certains ports ont quelques particularités que nous allons détailler ci-desous,

IV.1 Le port d' E/S PORTA

Le port A désigné par PORTA est un port de 6 bits (RA0 à RA5). RA6 et RA7 ne sont pas accessibles.

La configuration de direction se fait à l'aide du registre TRISA, positionner un bit de TRISA à 1 configure la broche correspondante de PORTA en entré et inversement. Au départ toutes les broches sont configurées en entrée

IV.1.1 La broche RA4

En entrée, la broche RA4 peut être utilisée soit comme E/S numérique normale, soit comme entrée horloge pour le Timer TMR0

En sortie, RA4 est une E/S à drain ouvert, pour l'utiliser comme sortie logique, il faut ajouter une résistance de pull-up externe. Le schéma (Fig. IV.1) illustre (pour les non électroniciens) le principe d'une sortie drain ouvert (ou collecteur ouvert) : si RA4 est positionnée à 0, l'interrupteur est fermé, la sortie est reliée à la masse, c'est un niveau bas. Si RA4 est placée à 1, l'interrupteur est ouvert, la sortie serait déconnectée s'il n'y avait pas la résistance externe qui la place au niveau haut.

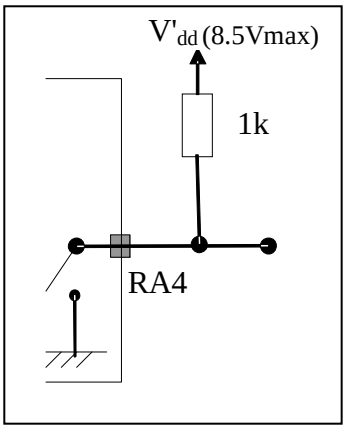


Fig. IV.1 : résistance de pull-up

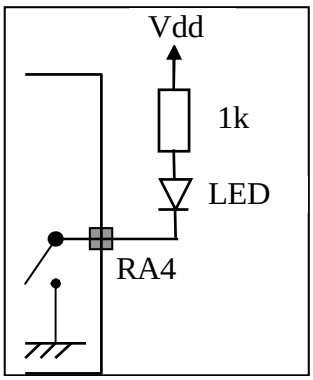


Fig. IV.2 : LED sur RA4

Si on veut utiliser RA4 pour allumer une LED, on peut utiliser le schéma de *Fig. IV.2*. Il faut juste remarquer que la logique est inversée, si on envoie 0 sur RA4, l'interrupteur se ferme et la LED s'allume. Si on envoie 1, l'interrupteur s'ouvre et la LED s'éteint.

Le schéma illustré sur *Fig. IV.3* peut aussi être utilisé. La logique n'est pas inversée mais il demande une précaution particulière. Il ne faut pas positionner la sortie RA4 à l'aide d'une instruction qui réalise une opération sur l'état actuel du port; genre IORWF, ANDWF, XORWF ou COMF, ADDWF, INCF … Ces instructions réalisent une lecture-écriture en commencent par lire l'état du port pour ensuite faire une opération dessus. Or, (sur *Fig. IV.3*) si la sortie était au niveau haut, l'interrupteur est ouvert, la LED est allumée et elle impose une tension de l'ordre de 1.5V qui sera considérée (à tort) comme un niveau bas lors de la lecture du port par les instructions précitées. La solution est d'utiliser des instructions qui positionnent le PORT sans tenir compte de son état courant comme MOVWF, BSF ou BCF

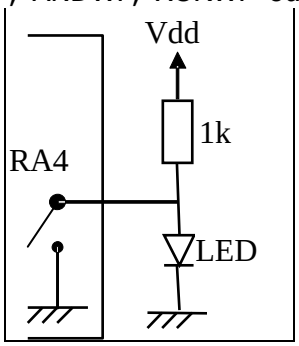


Fig. IV.3 : LED sur RA4

IV.1.2 Les autres broches de PORTA

Les autres broches (RA0, RA1, RA2, RA3 et RA5) peuvent être utilisées soit comme E/S numériques soit comme entrées analogiques. Au RESET, ces E/S sont configurées en entrées **analogiques**.

- Pour les utiliser en SORTIES NUMERIQUES, il suffit de les configurer en sortie à l'aide du registre de direction TRISA.
- Pour les utiliser en ENTRÉES NUMERIQUES, il faut les configurer en entrée à l'aide de TRISA, en plus il faut les configurer en numérique en plaçant la valeur '00000110' dans le registre ADCON1,
- Pour les utiliser en ENTRÉE ANALOGIQUE, voir chapitre sur le convertisseur Analogique numérique.

Quelque soit le mode (Analogique ou Numérique), il faut utiliser le registre TRISA pour configurer la direction des E/S :

- Bit i de TRISA = 0 → bit i de PORTA configuré en **sortie**
- Bit i de TRISA = 1 → bit i de PORTA configuré en **entrée**

IV.2 Le port d' E/S PORTB

- Le port B désigné par PORTB est un port bidirectionnel de 8 bits (RB0 à RB7). Toutes les broches sont compatibles TTL.
- La configuration de direction se fait à l'aide du registre TRISB, positionner un bit de TRISB à 1 configure la broche correspondante de PORTB en entré et inversement. Au départ toutes les broches sont configurées en entrée.
- En entrée, la ligne RB0 appelée aussi INT peut déclencher l'interruption externe INT.
- En entrée, une quelconque des lignes RB4 à RB7 peut déclencher l'interruption RBI. Nous reviendrons là-dessus dans le paragraphe réservé aux interruptions.

IV.3 Le port d' E/S PORTC

- Le port C désigné par PORTC est un port bidirectionnel de 8 bits (RC0 à RC7). Toutes les broches sont compatibles TTL.
- La configuration de direction se fait à l'aide du registre TRISC, positionner un bit de TRISC à 1 configure la broche correspondante de PORTC en entré et inversement. Au départ toutes les broches sont configurées en entrée.
- Toutes les broches du port C peuvent être utilisées soit comme E/S normales soit comme broches d'accès à différents modules comme le timer 1, les modules de comparaison et de capture CCP1/2, le timer 2, le port I2C ou le port série, ceci sera précisé au moment de l'étude de chacun de ces périphériques.
- Pour l'utilisation d'une broche du port C comme E/S normale, il faut s'assurer qu'elle n'a pas été affectée à un de ces modules. Par exemple, si TIMER1 est validé, il peut utiliser les broches RC0 et RC1 selon sa configuration.

IV.4 Le port d' E/S PORTD

- Le port D désigné par PORTD est un port bidirectionnel de 8 bits (RD0 à RD7). Toutes les broches sont compatibles TTL et ont la fonction trigger de Schmitt en entrée.
- Chaque broche et configurable en entrée ou en sortie à l'aide du registre TRISD. Pour configurer une broche en entrée, on positionne le bit correspondant dans TRISD à 1 et inversement.
- PORTD peut être utilisé dans un mode particulier appelé *parallel slave port*, pour cela il faut placer le bit *PSPMODE* (bit 4) de TRISE à 1. Dans ce cas les 3 bits de PORTE deviennent les entrées de control de ce port (RE, WE et CS)

Pour utiliser PORTD en mode normal, il faut placer le bit PSPMODE de TRISE à 0

IV.5 Le port d' E/S PORTE

- PORTE contient seulement 3 bits RE0, RE1 et RE2. Les 3 sont configurables en entrée ou en sortie à l'aide des bits 0, 1 ou 2 du registre TRISE.
- Les 3 bits de PORTE peuvent être utilisés soit comme E/S numérique soit comme entrées analogiques du CAN. La configuration se fait à l'aide du registre ADCON1.
- Si le bit PSPMODE de TRISE est placé à 1, Les trois bits de PORTE deviennent les entrées de control du PORTD qui (dans ce cas) fonctionne en mode *parallel Slave mode*
- A la mise sous tension (RESET), les 3 broches de PORTE sont configurées comme entrées analogiques.

Pour utiliser les broches de PORTE en E/S numériques normales : - Placer 06h dans ADCON1 - Placer le bit *PSPMODE* de TRISE à 0

IV.6 Situation au démarrage

Au démarrage :

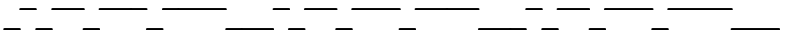
- Tous les ports sont configurés en ***entrée***
- PORTD et PORTE sont configurés en ***mode normal***
- PORTA et PORTE sont configurés en ***analogique***

Exercice 10) : Clignoter une LED

Donner le programme qui fait clignoter une LED branchée sur RA3 avec une temporisation voisine de 0.5s. Sachant que le PIC est doté d'un quartz de 4 MHz, la temporisation sera réalisée à l'aide de boucles imbriquées

Exercice 11) :

On dispose d'un PIC cadencé par un quartz de 4MHz. Donner le Programme qui utilise les instructions *bsf*, *bcf* et *nop* pour générer le signal suivant sur la sortie RB0



Exercice 12) :

Programme qui surveille l'état de l'entrée RA1 :

- Si RA1 = 0 ➔ faire RA3 = 1, PORTB = 00001111
- Si RA1 = 1 ➔ faire RA3 = 0, PORTB = 11110000

Exercice 13) :

Programme qui :

- Recopie W sur PORTB
- Génère une impulsion *(STROBE)* ⎍ de largeur 3 cycles machines sur RC4
- Attend une impulsion *(ACK)* ⏋ sur RA2

Exercice 14) : compteur d'impulsions

Programme qui :

- Allume la LED branchée sur RB3
- Compte 150 impulsions ⎍ sur l'entrée RA4 (la case mémoire 70h servira de compteur)
- Eteint la LED branchée sur RB3

V Les mémoires permanentes

Le PIC 16F877 dispose de 2 mémoires permanentes. La mémoire EEPROM PROGRAMME de capacité 8k mots de 14 bits et la mémoire EEPROM DE DONNÉES de capacité 256 octets. On dispose de deux méthodes pour écrire dans ces mémoires.

La première consiste à flasher le PIC avec l'exécutable (.hex). Les instructions sont flashés dans la EEPROM programme et les donnée sont flashés dans la EEPROM de données. Le flashage se fait à l'aide d'un programmeur et le soft qui va avec. (voir les directive DE et ORG dans ce cours et d'autres directives comme DA, DW … dans le manuel d'utilisation de MPASM)

La deuxième méthode consiste à accéder aux mémoires EEPROM à partir du programme durant la phase d'exécution de ce dernier. C'est ce que nous allons détailler dans les paragraphes ci-dessous.

V.1 La mémoire EEPROM de données

Le PIC 16F876/877 dispose de 256 octets de mémoire EEPROM de donnée. Son implantation physique commence à la position d'adresse absolue 2100h. Mais pour y accéder à partir des programmes utilisateur on utilise l'adressage relatif par rapport à la première position. La première position aura l'adresse 0, la deuxième aura l'adresse 1. . . et la dernière aura l'adresse 255.

Pour accéder à la EEPROM, on utilise 4 registres particuliers :

- EEADR : registre d'adresse (relative) (bank 2)
- EEDATA : registre de donnée (bank 2)
- EECON1 : registre de control (bank 3)
- EECON2 : 2ème registre de control (bank 3)

Le registre EECON1 :

EEPGD	—	—	—	WRERR	WREN	WR	RD

EEPGD : Accès à la mémoire EEPROP ou à la mémoire Programme
0 : EEPROM de données
1 : Mémoire programme (flash)

WRERR : Erreur d'écriture (indicateur)
0 : Pas d'erreur
1 : Une erreur s'est produite

WREN : Validation de l'écriture dans l'EEPROM
0 : Ecriture interdite
1 : Ecriture autorisée

WR : Write Enable. Ce bit doit être mis à 1 pour démarrer l'écriture d'un octet. Il est remis à zéro automatiquement à la fin de l'écriture. Ce bit ne peut pas être mis à zéro par une instruction.

RD : Read Enable. Ce bit doit être mis à 1 pour démarrer la lecture d'un octet. Il est remis à zéro automatiquement à la fin de la lecture. Ce bit ne peut pas être mis à zéro par une instruction

V.1.1 Procédure de lecture dans l'EEPROM

Pour lire le contenu d'une position de la mémoire EEPROM, on place l'adresse dans le registre EEADR, on lance l'opération de lecture à l'aide du bit RD du registre EECON1, la donnée désirée est tout de suite disponible dans le registre EEDATA :

1) Mettre le bit EEPGD à 0 pour pointer sur l'EEPROM de donnée
2) Placer l'adresse relative de la position à lire dans EEADR
3) Mettre le bit RD à 1 pour démarrer la lecture. Ce bit revient à 0 automatiquement tout de suite après le transfert de la donnée vers EEDATA, (moins d'un cycle)
4) Traiter la donnée disponible dans EEDATA
5) Recommencer au point 2 si on a d'autres données à lire,

V.1.2 Procédure d'écriture dans l'EEPROM

Pour écrire une donnée dans une position de la mémoire EEPROM, on place l'adresse dans le registre EEADR, la donnée dans le registre EEDATA, on lance l'opération d'écriture à l'aide du bit WR de EECON1 et du registre EECON2. La donnée présente dans EEDATA est alors copiée dans la EEPROM mais cette opération prend 10 ms. A la fin de l'écriture le bit WR revient à zéro automatiquement, le drapeau EEIF est levé ce qui peut déclencher l'interruption EEI si elle a été validée auparavant :

1) Interdire les interruptions (si elles ont été validées avant : bit INTCON.GIE)
2) Mettre le bit EEPGD à 0 pour pointer sur l'EEPROM de donnée
3) Positionner le bit WREN pour valider l'écriture dans l'EEPROM
4) Placer l'adresse relative de la position à écrire dans EEADR
5) Placer la donnée à écrire dans le registre EEDATA
6) - Ecrire 55h dans EECON2 (commande de process hardwares)
   - Ecrire AAh dans EECON2 (commandes de process hardwares)
   - Positionner le bit WR pour démarrer l'opération d'écriture, et attendre qu'il revienne à 0
7) Recommencer au point (4) si on a d'autres données à écrire,

Remarque : Les bits WREN et WR ne peuvent être positionné dans la même instruction. Le bit WR ne peut être positionné que si le bit WREN a été positionné avant.

Exercice 15)

Programme qui écrit l'alphabet majuscule dans la mémoire EEPROM de données à partir de la position 20h.

Exercice 16) EEPROM-D vers RAM

Programme qui utilise la directive DE pour écrire la chaîne "BONJOUR CHER AMI" dans la EEPROM de données à partir de la position 2140h. Le programme doit ensuite lire ces caractères (1 par 1) dans la EEPROM et les copier dans la RAM à partir de la position 110h

V.2 La mémoire Programme ou mémoire flash

Cette mémoire de 8 x 1024 mots de 14 bits sert à stocker le programme, mais elle est accessible par programme et peut donc être utilisée comme une extension de la mémoire EEPROM de données. Elle est non volatile (flash) et reprogrammable à souhait. Chaque position de 14 bits contient une instruction. L'emplacement du programme peut se situer à n'importe quel endroit de la mémoire. Cependant il faut savoir que suite à un RESET ou lors de la mise sous tension, le PIC commence l'exécution à l'adresse 0000_H. De plus, lorsqu'il y a une interruption, le PIC va à l'adresse 0004_H. Il est donc nécessaire de bien organiser le programme si celui-ci utilise des interruptions.

Le programme exécutable par le PIC est implanté dans la mémoire flash à l'aide d'un programmateur (hard+soft) sur lequel nous reviendrons dans la suite de ce document.

L'accès à la mémoire flash par les instructions du programme se fait de la même façon que l'accès à l'EEPROM de données, à l'exception des points suivant :

- Le bit *EEPGD* doit être placé à 1
- Le registre *EEADR* (8 bits) seul ne suffit pas à adresser les 8k de mémoire programme, on lui accole le registre *EEADRH* dans lequel il faut écrire la partie haute de l'adresse. On obtient ainsi les 13 bits nécessaires pour adresser 8K.
- Le registre *EEDATA* (8 bits) seul ne suffit pas pour contenir les 14 bits contenus dans une position de la mémoire programme. On lui accole le registre *EEDATH* qui contiendra les 6 bits supérieurs.
- Il faut insérer deux instructions NOP dans les cycles de lecture/écriture. Pendant la phase d'écriture, le processeur arrête l'exécution des instructions. Il n'est donc pas nécessaire d'attendre la fin de l'écriture pour continuer car ceci se fait automatiquement, à la fin de l'écriture, l'exécution reprend à l'instruction qui suit le 2ème NOP,

V.2.1 Procédure de lecture dans la mémoire programme

1) Mettre le bit EEPGD à 1 pour pointer sur la mémoire programme
2) Placer l'adresse de la position à lire dans EEADRH:EEADR
3) Mettre le bit RD à 1 pour démarrer la lecture
4) Attendre 2 cycles machine (2 instructions NOP)
5) Lire le contenu des registres EEDATH:EEDATA
6) Recommencer au point 2) si on a d'autres données à lire

V.2.2 Procédure d'écriture dan la mémoire programme

1) Interdire les interruptions (si elles ont été validées avant)
2) Mettre le bit EEPGD à 1 pour pointer sur la mémoire programme
3) Positionner le bit WREN pour valider l'écriture dans la mémoire programme
4) Placer l'adresse de la position à écrire dans EEADRH:EEADR
5) Placer la donnée à écrire dans le registre EEDATH:EEDATA
6) - Ecrire 55h dans EECON2 (commande de process hardwares)
   - Ecrire AAh dans EECON2 (commandes de process hardwares)
   - Positionner le bit WR pour démarrer l'opération d'écriture
7) Exécuter 2 instructions NOP
8) Recommencer au point 3 si on a d'autres données à écrire

Remarque : après le point (6), ça ne sert à rien d'attendre que WR passe à 0 pour détecter la fin de l'écriture. Le processeur arrête l'exécution du programme pendant la phase d'écriture dans la mémoire programme. La boucle de scrutation ne sera exécutée qu'après la fin de l'écriture ce qui la rend tout à fait inutile. Il suffit de mettre deux instructions NOP avant de continuer.

Exercice 17) Mem-Prog vers RAM

Programme qui lit 20 positions de la mémoire programme débutant à la position 12FAh et les copie dans la RAM à partir de la position 110h. ***Attention*** *:*

- *Le contenu d'une position mémoire programme permet de remplir 2 positions de la RAM.*
- *Le débordement de EEADR n'affecte pas EEADRH*

VI Les interruptions

Une interruption provoque l'arrêt du programme principal pour aller exécuter une procédure d'interruption. A la fin de cette procédure, le microcontrôleur reprend le programme principal à l'endroit où il l'a laissé. A chaque interruption sont associés deux bits, un bit de validation et un drapeau. Le premier permet d'autoriser ou non l'interruption, le second permet au programmeur de savoir de quelle interruption il s'agit.

Sur le 16F876/877, l'es interruptions sont classées en deux catégories, les interruptions primaires et les interruptions périphériques. Elles sont gérées par les registres :

INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIE1 (bk1)	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIR1 (bk0)	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE2 (bk0)	-	-	-	EEIE	BCLIE	-	-	CCP2IE
PIR2 (bk1)	-	-	-	EEIF	BCLIF	-	-	CCP2IF
OPTION_REG(bk1)		INTEDG						

- Toutes les interruptions peuvent être validées/interdites par le bit INTCON.GIE
- Toutes les interruptions périphériques peuvent être validées/interdites par le bit INTCON.PEIE
- Chaque interruption peut être validée/interdite par son bit de validation individuel

En résumé, pour valider une interruption périphérique (par exemple), il faut positionner 3 bits, GIE, PEIE et le bit individuel de l'interruption.

VI.1 Déroulement d'une interruption

Lorsque l'événement déclencheur d'une interruption intervient, alors son drapeau est positionné à 1 (levé). Si l'interruption a été validée (bits de validations = 1), elle est alors déclenchée : le programme arrête ce qu'il est en train de faire et va exécuter la procédure d'interruption qui se trouve à l'adresse 4 en exécutant les étapes suivantes :

- l'adresse contenue dans le PC (Program Counter) est sauvegardée dans la pile, puis remplacée par la valeur 0004 (adresse de la routine d'interruption).
- Le bit GIE est placé "0" pour inhiber toutes les interruptions (afin que le PIC ne soit pas dérangé pendant l'exécution de la procédure d'interruption).
- A la fin de la procédure d'interruption (instruction RETFIE) :
- le bit GIE est remis à 1 (autorisant ainsi un autre événement)
- le contenu du PC est rechargé à partir de la pile ce qui permet au programme de reprendre là où il s'est arrêté

Deux remarques importantes sont à faire :

Le drapeau reste à l'état haut même après le traitement de l'interruption. Par conséquent, il faut toujours le remettre à "0" à la fin de la routine d'interruption sinon l'interruption sera déclenchée de nouveau juste après l'instruction RETFIE

Seul le PC est empilé automatiquement. Si cela est nécessaire, les registres W et STATUS doivent être sauvegardés en RAM puis restaurés à la fin de la routine pour que le microcontrôleur puisse reprendre le programme principal dans les mêmes conditions où il l'a laissé.

VI.2 Les sources d'interruption

Interruption : Source d'interruption	Validation	Flag	PEIE
T0I : Débordement Timer 0	INTCON,T0IE	INTCON,T0IF	non
INT : Front sur RB0/INT	INTCON,INTE	INTCON,INTF	non
RBI : Front sur RB4-RB7	INTCON,RBIE	INTCON,RBIF	non
ADI : Fin de conversion A/N	PIE1,ADIE	PIR1,ADIF	oui
RCI : Un Octet est reçu sur l'USART	PIE1,RCIE	PIR1,RCIF	oui
TXI : Fin transmission d'un octet sur l'USART	PIE1,TXIE	PIR1,TXIF	oui
SSPI : Caractère émis/reçu sur port série synchrone	PIE1,SSPIE	PIR1,SSPIF	oui
TMR1I : Débordement de Timer 1	PIE1,TMR1IE	PIR1,TMR1IF	oui
TMR2I :Timer 2 a atteint la valeur programmée	PIE1,TMR2IE	PIR1,TMR2IF	oui
PSPI : Lecture/écriture terminée sur Port parallèle (16F877)	PIE1,PSPIE	PIR1,PSPIF	oui
CCP1I : Capture/comparaison de TMR1 avec module CCP1	PIE1,CCP1IE	PIR1,CCP1IF	oui
CCP2I : Capture/comparaison de TMR1 avec module CCP2	PIE2,CCP2IE	PIR2,CCP2IF	oui
EEI : Fin d'écriture en EEPROM	PIE2,EEIE	PIR2,EEIF	oui
BCLI : Collision sur bus SSP en mode I2C	PIE2,BCLIE	PIR2.BCLIF	oui

VI.3 L'interruption INT (Entrée RB0 du port B)

Cette interruption est provoquée par un changement d'état sur l'entrée RB0 du port B quand elle est programmée en entrée. En plus de son bit de validation INTE et son drapeau INTF, elle est gérée aussi par le bit INTEDG (OPTION_REG) qui détermine le front sur lequel l'interruption se déclenche, 1=montant, 0=descendant

VI.4 L'interruption RBI (RB4 A RB7 du port B)

Cette interruption est provoquée par un changement d'état sur l'une des entrées RB4 à RB7 du port B, Le front n'a pas d'importance. Les bits associés sont RBIE (validation) et RBIF (drapeau).

ATTENTION : Le drapeau RBIF ne peut être remis à zéro sans la lecture préalable de PORTB (*MOVF PORTB,w*). Si on ne veut pas modifier le contenu de W, on peut copier PORTB sur lui-même *(MOVF PORTB,f)*.

VI.5 Les autres interruptions

Les autres interruptions seront abordées au moment de l'étude des modules qui les déclenchent.

Exercice 18) (int.asm)

Programme qui utilise l'interruption INT comme suit :
Chaque fois que l'entrée RB0 passe de 1 à 0, la LED branchée sur RB1 clignote 5 fois au rythme de la 0.6 seconde. Le PIC est cadencé par un quartz de 8 MHz

Exercice 19) (div-freg-rbi.asm)

Ecrire un programme qui réalise une division de fréquence par 5. Le signal d'entrée est appliqué sur l'entrée RB4. Le signal de sortie est généré sur la sortie RB1. On utilisera l'interruption RBI pour détecter les transitions du signal d'entrée.

VII Les Timers

VII.1 Le Timer TMR0

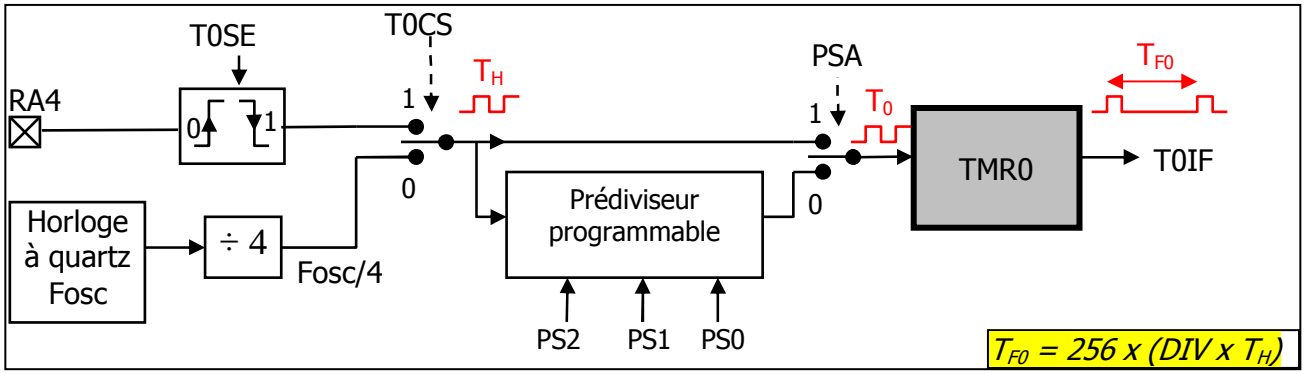
C'est un compteur 8 bits ayant les caractéristiques suivantes :

- Il est incrémenté en permanence soit par l'horloge interne Fosc/4 (mode timer) soit par une horloge externe appliquée à la broche RA4 du port A (mode compteur). Le choix de l'horloge se fait à l'aide du bit T0CS du registre OPTION_REG
  - T0CS = 0 → horloge interne
  - T0CS = 1 → horloge externe appliquée à RA4
- Dans le cas de l'horloge externe, Le bit T0SE du registre OPTION_REG permet de choisir le front sur lequel le TIMER s'incrémente.
  - T0SE = 0 → incrémentation sur fronts montants
  - T0SE = 1 → incrémentation sur fronts descendants
- Quelque soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport DIV est fixés par les bits PS0, PS1 et PS2 du registre OPTION_REG (tableau ci-contre). L'affectation ou non du prédiviseur se fait à l'aide du bit PSA du registre OPTION_REG
  - PSA = 0 → on utilise le prédiviseur
  - PSA = 1 → pas de prédiviseur (affecté au chien de garde)
- Le contenu du timer TMR0 est accessible par le registre qui porte le même nom. Il peut être lu ou écrit à n'importe quel moment. Après une écriture, le timer ne comte pas pendant deux cycles machine.
- **Au débordement de TMR0 (FF → 00), le drapeau INTCON.T0IF est placé à 1. Ceci peut déclencher l'interruption T0I si celle-ci est validée**

PS2	PS1	PS0	Div
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

OPTION_REG

RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0



En résumé, chaque fois que le compteur complète un tour, le drapeau T0IF se lève. Si on note T_H la période de l'horloge source, T_0 l'horloge de TMR0 et T_{F0} le temps qui sépare 2 levés de drapeau successifs :

- Sans prédiviseur : $T_{F0} = 256\ T_0 = 256\ T_H$
- Avec prédiviseur : $T_{F0} = 256\ T_0 = 256 \times (DIV \times T_H)$
- Avec prédiviseur et compteur N dans le programme on obtient : $NxT_{F0} = N \times 256 \times (DIV \times T_H)$

Exercice 20) Clignoter LED / TMR0

PIC doté d'un quartz de 4 MHz. Programme qui fait clignoter une LED branchée sur RB0, delay voisin de 0.5s à l'aide de TMR0

a) Par scrutation du drapeau T0IF (pas d'interruption)
b) En utilisant l'interruption T0I

VII.2 Le Watchdog Timer WDT (Chien de garde)

C'est un compteur 8 bits incrémenté en permanence (même si le µC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (WDT TimeOut), deux situations sont possibles :

- Si le µC est en fonctionnement normal, le WDT time-out provoque un RESET. Ceci permet d'éviter de rester *planté* en cas de blocage du microcontrôleur par un processus indésirable non contrôlé
- Si le µC est en mode SLEEP, le WDT time-out provoque un WAKE-UP, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode SLEEP. Cette situation est souvent exploitée pour réaliser des temporisations

L'horloge du WDT a une période voisine de 70 µs ce donne un Time-Out toutes les 18 ms. Il est cependant possible d'augmenter cette durée en faisant passer le signal Time-Out dans un prédiviseur programmable (partagé avec le timer TMR0). L'affectation se fait à l'aide du bit PSA du registre OPTION_REG

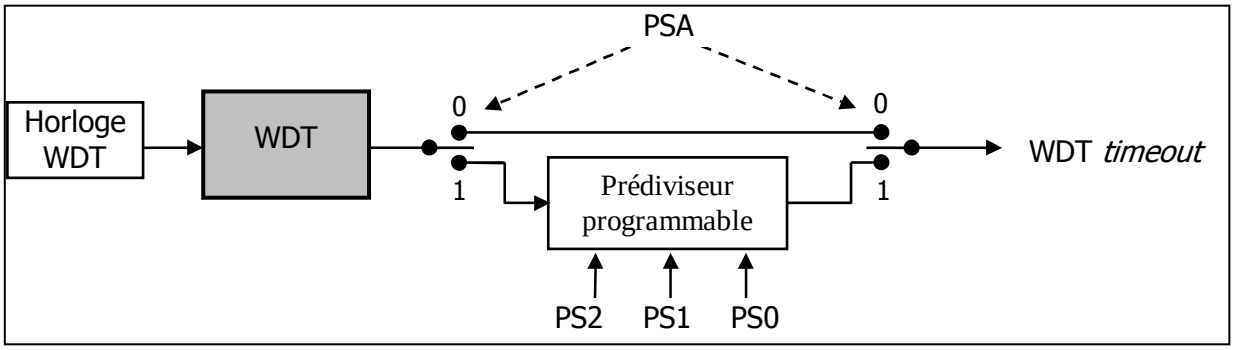
- PSA = 1 → on utilise le prédiviseur
- PSA = 0 → pas de prédiviseur (affecté à TMR0)

Le rapport du prédiviseur est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG (voir tableau ci-contre)

PS2	PS1	PS0	Div
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

L'utilisation du WDT doit se faire avec précaution pour éviter la réinitialisation (inattendue) répétée du programme. Pour éviter un WDT timeOut lors de l'exécution d'un programme, on a deux possibilités :

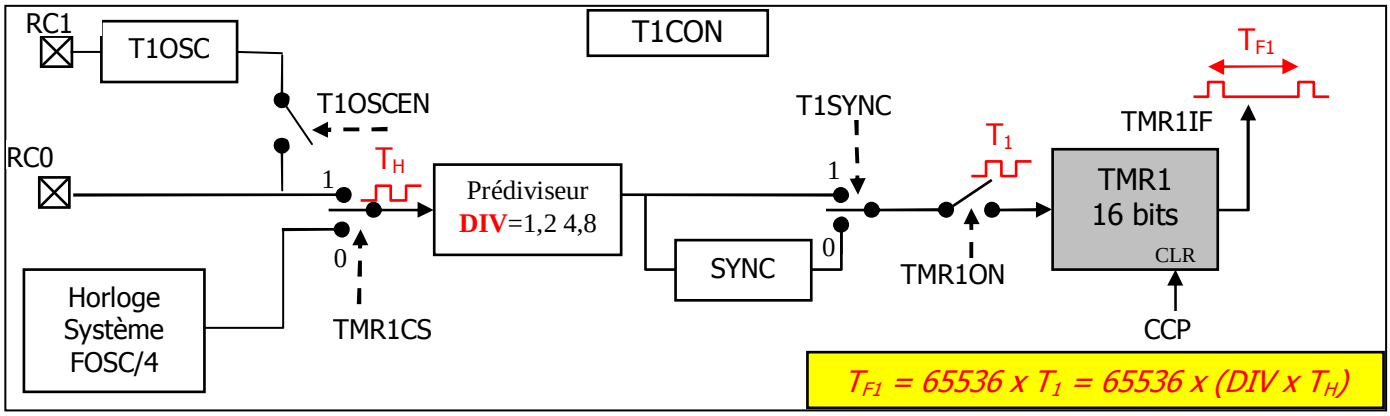
- Inhiber le WDT d'une façon permanente en mettant à 0 le bit WDTE dans l'EEPROM de configuration
- Remettre le WDT à 0 périodiquement dans le programme à l'aide de l'instruction CLRWDT pour éviter qu'il ne déborde



VII.3 Le Timer TMR1

TMR1 est un Timer/Compteur 16 bits accessible en lecture/écriture par l'intermédiaire des registres 8 bits TMR1H (bank0) et TMR1L (bank0) qui constituent sa partie haute et sa partie basse.

On le configure à l'aide du registre T1CON (bank0)



- TMR1 peut fonctionner dans 3 modes différents :
  - Timer Synchrone (horloge interne)
  - Compteur Synchrone (horloge externe)
  - Compteur Asynchrone (horloge externe)

  Le bit de contrôle TMR1CS détermine le fonctionnement en Timer ou en Compteur et le bit T1SYNC détermine le mode de fonctionnement en synchrone ou en asynchrone

- TMR1 peut être arrêté/démarré à l'aide du bit TMR1ON
- TMR1 peut être RAZ à l'aide du module de capture/comparaison CCP
- TMR1 peut être précédé d'un prédiviseur (Prescaler) qui peut diviser la fréquence par 1, 2, 4 ou 8 selon la valeur des bits T1CKPS1 et T1CKPS0
- A son débordement (FFFFh → 0000h) le drapeau PIR1.TMR1IF (bank0) est positionné ce qui peut déclencher l'interruption périphérique TMR1I si elle est validée par son bit de validation PIE1.TMR1IE (bank1).

VII.3.1 Le mode Timer

Dans ce mode, TMR1 est incrémenté par l'horloge système Fosc/4 éventuellement prédivisée. Le bit de synchronisation n'a pas d'effet car l'horloge Fosc/4 est toujours synchronisée sur l'horloge système.

VII.3.2 Le mode Compteur

Dans ce mode, TMR1 est incrémenté à chaque front montant de l'horloge externe T1CKI (RC0) ou l'horloge dédiée générée par l'oscillateur T1OSC à condition de positionner le bit T1OSCEN à 1 et de brancher un quartz entre les broche RC0 et RC1.

En mode compteur, RC0 et RC1 sont automatiquement configurées en entrée, on n'a pas besoin de configurer les bits TRISC,0 et TRISC,1

- En fonctionnement **Synchrone**, l'horloge externe (éventuellement prédivisée) n'incrémente pas directement le timer mais elle est synchronisée sur l'horloge système ce qui peut entraîner un délai de l'ordre de 1 cycle machine. Dans cette configuration

- En fonctionnement **Asynchrone**, l'horloge externe (éventuellement prédivisée) incrémente le timer indépendamment de l'horloge système.

En mode Compteur Asynchrone, on ne peut pas utiliser les modules CCP1 et CCP2 pour faire des captures ou des comparaisons sur TMR1

VII.3.3 Le registre de control de T1CON

—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

T1CKPS1,T1CKPS0 : Control du prescaler
00 : division par 1
01 : division par 2
10 : division par 4
11 : division par 8

T1OSCEN : Validation de l'Oscillateur associé à TMR1
0 : Oscillateur arrêté
1 : Oscillateur activé

T1SYNC : Synchronisation de l'horloge externe (ignoré en mode timer)
0 : Synchronisation
1 : pas de synchronisation

TMR1CS : Choix de l'horloge du Timer
0 : horloge système (Fosc/4) : mode timer
1 : Horloge externe : mode compteur

TMR1ON : Démarrer arrêter le timer
0 : Timer stoppé
1 : Timer en fonctionnement

En résumé :
Le temps qui sépare 2 levés de drapeau TMR1IF est : $T_{F1} = 65536 \times T_1 = 65536 \times (DIV \times T_H)$
Si on ajoute un compteur *N* dans le programme on obtient un temps = $N \times T_{F1}$

Exercice 21) Clignoter LED / scrutation de TMR1

Clignoter une LED reliée à RE0. La temporisation voisine de 0.5s sera réalisée à l'aide de TMR1 par scrutation du drapeau TMR1IF

Exercice 22) Clignoter LED / interruption TMR1

Clignoter une LED reliée à RD0. La temporisation voisine de 0.5s sera réalisée à l'aide de TMR1 est son interruption TMR1I

VII.4 Les module de Comparaison/Capture CCP1 et CCP2

Chacun des modules CCP1 et CCP2 permet :

- Soit de CAPTURER en un seul coup le contenu du double registre TMR1
- Soit de COMPARER en permanence son contenu avec un registre 16 bits et de déclencher un événement au moment de l'égalité.

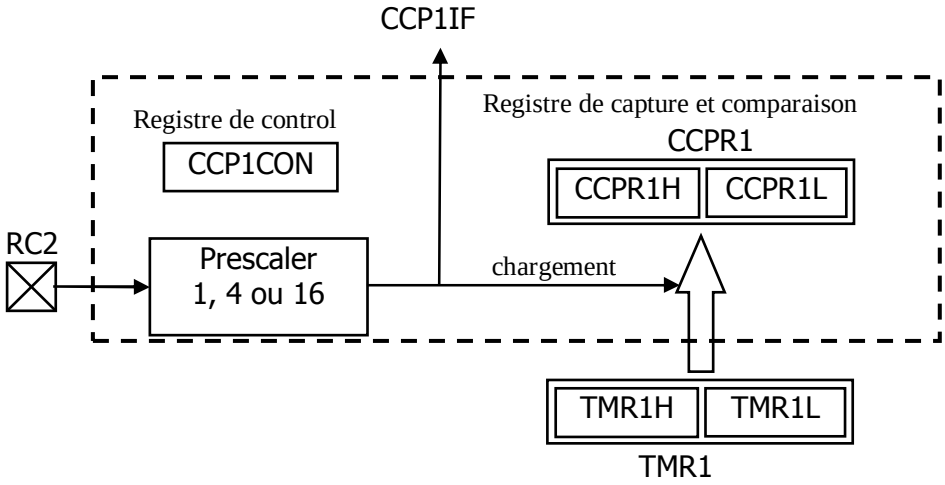
Ces modules ne fonctionnent pas si TMR1 est configuré en mode Compteur non synchronisé

VII.4.1 Le module CCP1

Ce module est constitué de :

- Un registre 16 bits CCPR1 utilisé pour la capture ou la comparaison de TMR1. Il est accessible par sa partie basse CCP1L et sa partie haute CCP1H
- Un registre de contrôle 8 bits CCP1CON.
- Un prédiviseur permettant de filtrer les événements déclencheurs de capture venant de la broche RC2

VII.4.1.1 Le mode Capture



Dans ce mode le contenu de TMR1 est copié dans CCPR1 chaque fois qu'un événement intervient sur la broche RC2. Le choix de l'événement déclencheur se fait en programmant le prescaler à l'aide des bits 0 à 3 du registre de contrôle CCP1CON. On a le choix parmi les événements suivants :

- A chaque front descendant
- A chaque front montant (prescaler 1:1)
- A chaque 4ème front montant (prescaler 1:4)
- A chaque 16ème front montant (prescaler 1:16)

A la fin de la capture, le drapeau CCP1IF est positionné, l'interruption périphérique associée est déclenchée si elle a été validée.

Plusieurs aspects sont à noter comme :

- Si on veut déclencher la capture par un signal externe, celui-ci doit être appliqué sur la proche RC2 qui doit être configurée en entrée par le bit TRISC,2

- Si on veut déclencher la capture par programme en changeant la valeur du bit RC2, celui-ci doit être configuré en sortie par le bit TRISC,2
- Lors de la modification du mode de capture, une interruption indésirable peut intervenir. L'utilisateur doit veiller à masquer l'interruption CCP1I avant de procéder à cette modification et de baisser le drapeau CCP1IF après la modification.
- En mode Sleep, le prescaler et le drapeau CCP1IF restent opérationnels. le positionnement du drapeau peut déclencher un Wake-up si le bit de validation CCP1IE a été validé auparavant

VII.4.1.2 Le registre de configuration CCP1CON

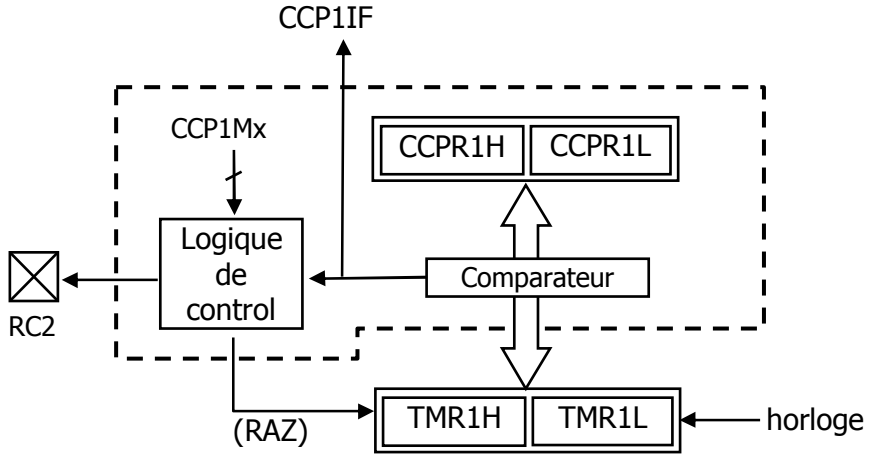
—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

DC1B1, DC1B0 : utilisés en PWM mode
Ce sont les 2 bits de poids faible des 10 bits MWM duty cycle. Les 8 autres bits sont dans le registre CCPR1L

CCP1M3 à CCP1M0 : mode de fonctionnement du module CCP1
0000 : Module arrêté (reset module)
0100 : Capture sur chaque front descendant
0101 : Capture sur chaque front montant
0110 : Capture tous les 4 fronts montants
0111 : capture tous les 16 fronts montants
1000 : Mode comparaison (drapeau CCP1IF + broche RC2 $0 \rightarrow 1$)
1001 : Mode comparaison (drapeau CCP1IF + broche RC2 $1 \rightarrow 0$)
1010 : Mode comparaison (drapeau CCP1IF seulement)
1011 : Mode comparaison (drapeau CCP1IF + RAZ TMR1)
10xx : PWM mode (modulation de largeur d'impulsion), ce mode ne sera pas traité dan ce cours

VII.4.1.3 Le mode Comparaison

Dans ce mode le registre CCPR1 est comparé en permanence à TMR1. Quand l'égalité intervient, le drapeau CCP1IF passe à 1 et différentes actions sont accomplies selon le mode défini par les bits de configuration CCP1M3:CCP1M0



- **mode 1000 :**
Au moment de l'égalité, le drapeau CCP1IF est le bit RC2 passent à 1. C'est à l'utilisateur de les remettre à 0 pour une prochaine utilisation. RC2 doit être configuré en sortie.

- **mode 1001 :**
  Au moment de l'égalité, le drapeau CCP1IF passe à 1 et le bit RC2 passe à 0. C'est à l'utilisateur de les remettre à leur état d'origine pour une prochaine utilisation. RC2 doit être configuré en sortie.

- **mode 1010 :**
  A l'égalité le drapeau CCP1IF passe à 1. La broche RC2 n'est pas utilisée.

- **mode 1011 :**
  A l'égalité, le drapeau CCP1IF passe à 1 et le timer TMR1 est remis à 0

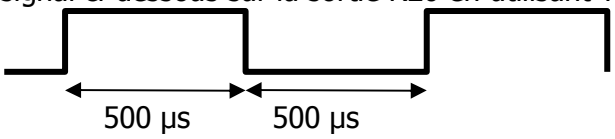
VII.4.2 Le module CCP2

Le module CCP2 est identique au module CCP1, il suffit d'interchanger 1 et 2 et de constater les points suivants :

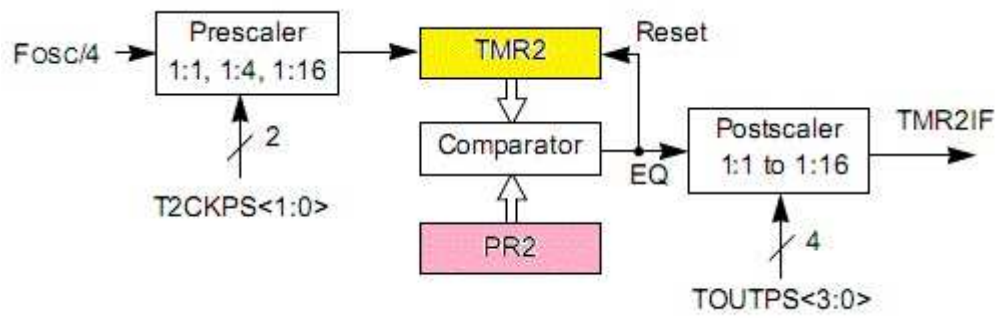
- CCP2 est associé à la broche RC1
- Il est géré par le registre de control CCP2CON
- Son registre de capture/comparaison est CCPR2 = CCPR2H:CCPR2L
- En mode comparaison 1011, A l'égalité :
  - le drapeau CCP2IF passe à 1
  - RAZ de TMR1
  - envoi d'un GO vers le convertisseur analogique numérique

Exercice 23) générer signal TMR1/CCP1

Programme qui génère le signal ci-dessous sur la sortie RE0 en utilisant TMR1 associé à CCP1



VII.5 Le Timer TMR2



TMR2 est un timer 8 bits accessible en lecture écriture constitué de :

- un registre de control T2CON (bank0)
- un prédiviseur (1,4,16)
- un registre de période PR2 (bank1) accessible en lecture/écriture
- un comparateur,
- un postdiviseur (1 à 16)

- TMR2 est incrémenté par l'horloge interne Fosc/4. Il commence à 0 et quant il atteint la valeur du registre PR2, le comparateur génère un signal qui :
  - Remet TMR2 à 0
  - incrémente le postscaler qui fonctionne comme un diviseur de fréquence

  Comme le comptage commence à 0, si PR2=N, alors le comparateur annonce une égalité tous les N+1 coups d'horloge
- Au débordement du postscaler, le drapeau PIR1.TMR2IF est positionné, l'interruption correspondante et déclenchée si elle est validée
- TMR2 est remis à zéro à chaque RESET
- Le prescaler et le postscaler sont initialisés à chaque écriture dans TMR2 ou dans T2CON et au RESET du processeur
- Le fonctionnement de TMR2 est configuré à l'aide du registre de control T2CON :

En résumé, Si on note :

DIV1 : rapport du prédiviseur

DIV2 : Rapport du postdiviseur

P : Valeur placée dans le registre PR2

Tsy : période de l'horloge système,

la périodicité du drapeau TMR2IF est donnée par :

$$TF2 = DIV1 \times (P+1) \times DIV2 \times Tsy .$$

Le registre T2CON:

—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

TOUTPS3:TOUTPS0 : ratio du postscaler
0000 : division par 1
0001 : division par 2
. . .
1111 : division par 16

TMR2ON : démarrer arrêter TMR2
0 : TMR2 off
1 : TMR2 on

T2CKPS1,T2CKPS0 : ratio du prescaler
00 : prédiviseur par 1
01 : prédiviseur par 4
1x : prédiviseur par 16

REMARQUE :

Dans le cas de la scrutation du drapeau TMR2IF par les 2 instructions suivantes :

```
ici     btfss   PIR1,TMR2IF  ; attendre drapeau
        goto    ici
```

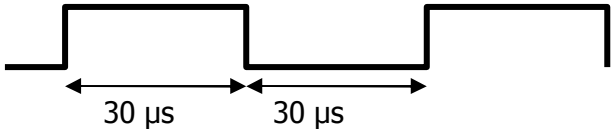
Les temporisations obtenues peuvent être erronées à cause de l'instruction goto qui prend deux cycles pour s'exécuter ce qui peut retarder la détection du drapeau.
La solution est d'utiliser l'interruption liée au drapeau TMR2IF, dans ce cas, les temporisations obtenues sont exactes

Exercice 24)

a) Si le PIC est cadencé par un quartz de 8 MHz, quelle la temporisation maximale que l'on peut obtenir à l'aide du timer TMR2
b) Que devient cette valeur si on ajoute un compteur N dans le programme
c) Quelle est la valeur max si le compteur N utilise une seule case mémoire

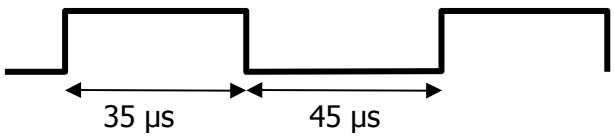
Exercice 25)

On dispose d'un PIC doté d'un quartz de 4 MHz. Donner le programme qui génère le signal ci-dessous sur la sortie RB2. On écrira deux versions du programme :
a) On scrute nous même le drapeau TMR2IF à l'aide de l'instruction ***btfss***,
b) On exploite l'interruption TMR2I.



Exercice 26)

c) Essayer de trouver une opération logique simple permettant de faire basculer le contenu d'un registre entre deux valeur N1 et N2
d) Utiliser cette astuce pour obtenir un programme qui génère le signal ci-dessous sur la sortie RB0. Le programme doit utiliser l'interruption TMR2I associée à TMR2. Le PIC est doté d'un quartz de 4 MHz.

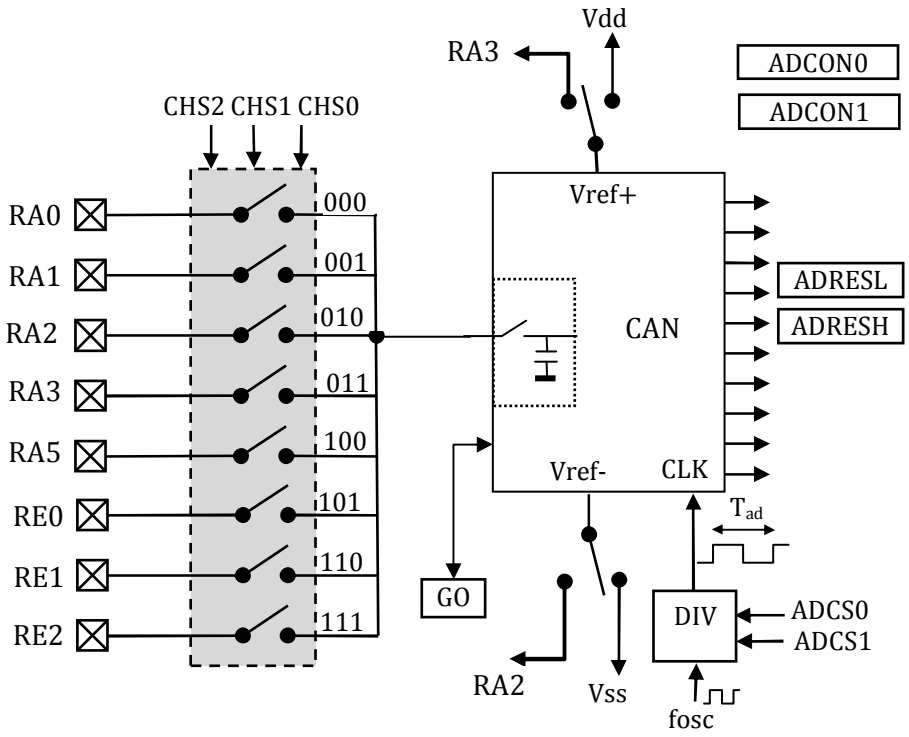


Exercice 27)

On dispose d'un PIC doté d'un quartz de 4 MHz. Donner le programme qui fait clignoter une LED reliée à RE0. La temporisation voisine de 0.5s est réalisée à l'aide de TMR2 :

1) Par scrutation du drapeau TMR2IF
2) En utilisant l'interruption TMR2I

VIII Le module de conversion A/N



Ce module est constitué d'un convertisseur Analogique Numérique 10 bits dont l'entrée analogique peut être connectée sur l'une des 8 (5 pour 16F876) entrées analogiques externes. On dit qu'on a un CAN à 8 canaux. Les entrées analogiques doivent être configurées en entrée à l'aide des registres TRISA et/ou TRISE. L'échantillonneur bloqueur est intégré, il est constitué d'un interrupteur d'échantillonnage et d'une capacité de blocage de 120 pF.

Les tensions de références permettant de fixer la dynamique du convertisseur. Elles peuvent être choisies parmi Vdd, Vss, Vr+ ou Vr-

Le control du module se fait par les deux registres ADCON0 et ADCON1

ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON

ADCS1:ADCS0 : Choix de l'horloge de conversion donc du temps de conversion (voir paragraphe VIII.2)
00 : Fosc/2
01 : Fosc/8
10 : Fosc/32
11 : Oscillateur RC dédié au CAN (fonctionne seulement pour FOSC < 1MHz)

CHS2:CHS0 : choix de l'entrée analogique
000 = channel 0, (RA0)
001 = channel 1, (RA1)
010 = channel 2, (RA2)
011 = channel 3, (RA3)
100 = channel 4, (RA5)
101 = channel 5, (RE0)
110 = channel 6, (RE1)
111 = channel 7, (RE2)

GO/DONE : Une conversion démarre quand on place ce bit à 1. A la fin de la conversion, il est remis automatiquement à zéro. Ce bit peut aussi être positionné automatiquement par le module CCP2.

ADON : Ce bit permet de mettre le module AN en service

ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0

ADFM : justification à droite ou à gauche du résultat dans les registre ADRESH et ADRESL

	ADRESH	**ADRESL**
1 : justifié à droite	**000000XX**	**XXXXXXXX**
0 : justifié à gauche	**XXXXXXXX**	**XX000000**

PCFG3:PCFG0 : configuration des E/S et des tensions de références. Les 5 broches de PORTA et les 3 de PORTE peuvent être configurés soit en E/S digitales, soit en entrées analogiques. RA2 et RA3 peuvent aussi être configurées en entrée de référence.

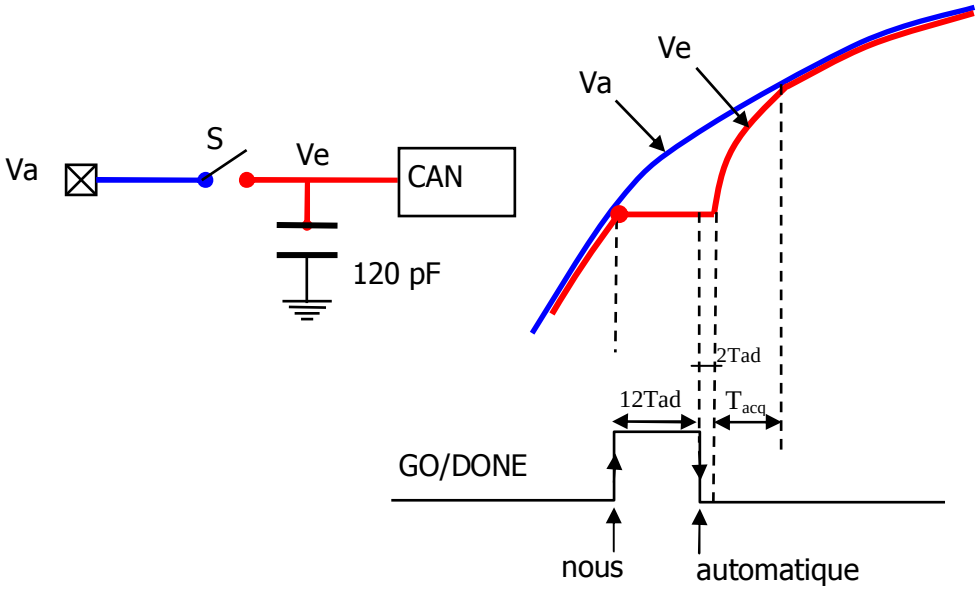
PCFG3: PCFG0	RE2	RE1	RE0	RA5	RA3	RA2	RA1	RA0	V_{REF+}	V_{REF-}	A/R/N
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0/0
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	7/1/0
0010	N	N	N	A	A	A	A	A	V_{DD}	V_{SS}	5/0/3
0011	N	N	N	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/1/3
0100	N	N	N	N	A	N	A	A	V_{DD}	V_{SS}	3/0/5
0101	N	N	N	N	V_{REF+}	N	A	A	RA3	V_{SS}	2/1/5
011x	N	N	N	N	N	N	N	N	V_{DD}	V_{SS}	0/0/8
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	6/2/0
1001	N	N	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0/2
1010	N	N	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	5/1/2
1011	N	N	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	4/2/2
1100	N	N	N	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	3/2/3
1101	N	N	N	N	V_{REF+}	V_{REF-}	A	A	RA3	RA2	2/2/4
1110	N	N	N	N	N	N	N	A	V_{DD}	V_{SS}	1/0/7
1111	N	N	N	N	V_{REF+}	V_{REF-}	N	A	RA3	RA2	1/2/5

VIII.1 Déroulement d'une Conversion

Le PIC dispose d'un échantillonneur bloqueur intégré constitué d'un interrupteur S, d'une capacité de maintien C=120 pF et d'un convertisseur Analogique numérique 10 bits. Pendant la conversion, la tension Ve à l'entrée du convertisseur A/N doit être maintenue constante.

Au départ il faut commencer par faire l'acquisition du signal en fermant l'interrupteur S, ceci se fait à l'aide du registre ADCON0, soit au moment de la validation du module par le bit ADON soit après un changement de canal si ADON est déjà positionné.

Après la fin de l'acquisition, on peut démarrer une conversion en positionnant le bit GO_DONE, l'interrupteur S s'ouvre pour assurer le blocage de la tension. La conversion commence, elle est réalisée en 12 T_{AD}, à la fin, le bit GO_DONE repasse à 0, le drapeau ADIF passe à 1 et le résultat est chargé dans les registres ADRESL et ADRESH. Le module met 2 T_{AD} supplémentaires pour fermer l'interrupteur S ce qui démarre une nouvelle phase d'acquisition pendant laquelle la tension Ve rejoint la tension analogique d'entrée Va. Le temps d'acquisition dépend de la constante de temps RC, R étant la somme des résistances entre le module de conversion et la source de la tension analogique. Après la fin de l'acquisition, on peut démarrer une nouvelle conversion et ainsi de suite



VIII.2 Temps de conversion

Le temps de conversion est égal à 12 T_{AD}
T_{AD} est le temps de conversion d'un bit, il dépend de la fréquence du quartz et du prédiviseur (div) choisi : T_{AD} = div x 1/fosc. Le choix de div doit être ajusté pour que T_{AD} soit ≥ à 1,6 µs

	Div	20Mhz	5Mhz	4Mhz	2Mhz	1Mhz
00	2	0,1 µs	0,4 µs	0,5 µs	1 µs	2 µs
01	8	0,4 µs	1,6 µs	2 µs	4 µs	8 µs
10	32	1,6 µs	6,4 µs	8 µs	16 µs	32 µs
11	RC	Non utilisable				≈4 µs

Tableau VIII.1 : Temps de conversion d'un bit TAD (les cases grisées sont hors plage d'utilisation)

Avec un quartz de 4 MHz, il faut choisir div=8 ce qui donne T_{AD} = 2 µs
soit un temps de conversion : T_{CONV} = 24 µs

VIII.3 Temps d'acquisition

Temps d'acquisition = T_{ACQ} = Tc + CT +2 µs

Tc : temps de charge du condenseur = (Ric+Rss+Rs) C Ln(2047)
Ric = Résistance d'interconnexions, elle est inférieure à 1k
Rss = Résistance du l'interrupteur S (Sampling switch), elle dépend de la tension d'alimentation Vdd. Elle est égale à 7kΩ pour Vdd=5V
Rs : Résistance interne de la source du signal analogique. Microchip recommande de ne pas dépasser 10 kΩ
C : Capacité de blocage = 120 pF

CT : Coefficient de température = (Tp -25°C) 0.05 µs/°C
Tp = Température Processeur, voisine de 45°C en temps normal

Exemple :
Ric = 1k, Rss = 7k, Rs = 2k, Tp = 45 °C :
Tc = 10k x 120pF x Ln(2047) = 9 µs
CT = 20 x 0.05 µs = 1 µs

T_{ACQ} = 2 + 9 + 1 µs = 12 µs

VIII.4 Fréquence d'échantillonnage

Si on veut échantillonner un signal variable, La période d'échantillonnage Te doit être supérieur ou égale à $T_{emin} = T_{CONV} + 2\ Tad + T_{ACQ}$

Avec les exemple précités, on aura la péride d'échantillonnage min $T_{emin} = (12+2)x2 + 12 = 40\ \mu s$
La fréquence d'échantillonnage max est donc $femax = 1/T_{emin} = 25\ kHz$
Si on tient compte de la règle de Shannon (fe > 2 fmax), on constate que l'on peut échantillonner des signaux dont la fréquence ne dépasse pas 12 KHz.

VIII.5 Valeur numérique obtenue

Quelle est la relation entre la tension analogique convertie et le nombre N recueilli dans le registre ADRES ? Si on note :

Q = pas de quantification = $(V_{ref+} - V_{ref-})/1024$
Va = tension analogique à convertir
N = valeur numérique obtenue,

N = valeur entière de $(Va - V_{ref-}) / Q$

Avec Vref- = masse, on obtient N = int (Va / Q)

exemple :

Vref+ = Vdd = 5V, Vref- = 0, Vin = 4 V
Q = 5V/1024 = 0,0048828125 V
N = 4V / 0,0048828125 = 819

VIII.6 Programmation

1) Si des entrée de PORTE sont utilisées, le configurer en mode normal à l'aide du bit PSPMODE
2) Configurer les E/S en Analogique/Numérique/Référence (ADCON1)
3) Configurer les entrées analogiques en entrées (TRISA, TRISE)
4) Définir l'horloge de conversion à l'aide du diviseur DIV dans ADCON0
5) Choisir le canal à convertir et valider le module (ADCON0)
6) Attendre le temps d'acquisition (12 µs)
7) Lancer la conversion, GO = 1 (ADCON0)
8) Attendre fin de conversion, GO = 0 ou drapeau ADIF=1
9) Traiter le résultat
10) Si l'on désire prendre d'autres mesures, recommencer au point 7 en faisant attention aux timings

Exercice 28)

PIC doté d'un quartz de 4 MHz. Programme qui converti la tension appliquée à RA0 et recopie le résultat dans la RAM à la position 70h et 71h

Exercice 29)

Sur un PIC doté d'un quartz de 4 MHz, Donner le programme qui fait l'acquisition de 40 échantillons du signal appliqué sur RA0, et recopie les résultats dans la RAM à partir de la position 190h. L'échantillonnage se fera à la vitesse la plus rapide possible

Exercice 30)

Refaire l'exercice précédent avec une fréquence d'échantillonnage fe = 8000Hz. Utiliser le timer 2 pour ajuster la période d'échantillonnage.

IX L'USART

L'USART (Universal Synchronous Asynchronous Receiver Transmitter) est l'un des deux modules de communication série dont dispose le PIC 16F876/877. L'USART peut être configuré comme système de communication asynchrone full duplex ou comme système synchrone half duplex (non étudié).

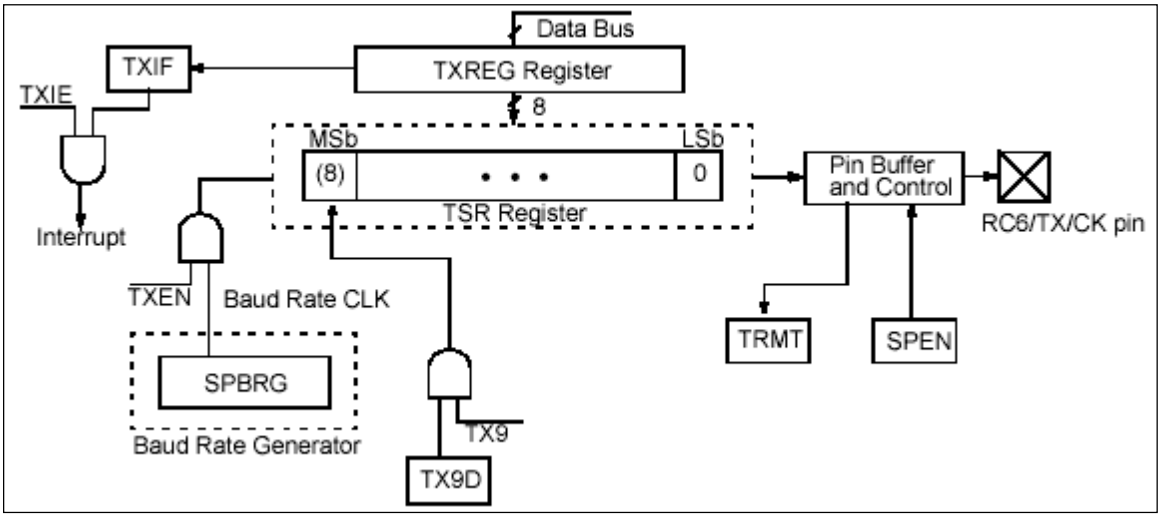
La communication se fait sur les deux broches RC6/TX et RC7/RX qui doivent être configurés toutes les deux en ENTREE par TRISC. (oui, j'ai bien dit toutes les deux)

IX.1 Mode Asynchrone

Si on place le bit SYNC du registre TXSTAT à 0, l'USART fonctionne dans le mode asynchrone standard, 10 (ou 11) bits sont transmis ou reçus dans l'ordre ci-dessous :
- 1 bit de START (toujours 0)
- 8 ou 9 bits de donnée (LSB d'abord)
- 1 bits de STOP (toujours 1)

- La transmission se fait sur la broche RC6/TX et la réception sur la broche RC7/RX
- La configuration et le control du port se fait par les registres TXSTA et RCSTA
- La vitesse de communication est fixée par le registre SPBRG et le bit TXSTA.BRGH
- La parité n'est pas gérée d'une façon matérielle, elle peut être gérée par soft si son utilisation est nécessaire.
- L'accès au port en lecture ou écriture se fait par les registres tampon RCREG et TXREG. La transmission et la réception se font par deux registres à décalage, un pour la transmission (TSR) et un pour la réception (RSR). L'accès au registres tampon peut se faire alors que les registre à décalage sont en train de transmettre/recevoir une donnée.
- Les drapeaux PIR1.RCIF et PIR1.TXIF sont très utiles pour gérer la lecture/écriture dans le port. RCIF est positionné quand le port a terminé une réception et TXIF est positionné quand le buffer de transmission TXREG est "vide".

IX.2 Le port en transmission



Le contrôle de la transmission se fait par le registre TXSTA

CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D

CSRC : non utilisé en mode asynchrone

TX9 et TX9D : Pour utiliser le mode 9 bits il faut positionner le bit TX9. Le 9ème bit doit être écrit dans TX9D avant d'écrire les 8 bits de données dans TXREG
TXEN : permet de valider ou interdire la transmission
SYNC : 0 → mode asynchrone, 1 → mode synchrone
BRGH : sélectionne le mode haut débit du générateur de baud rate
TRMT : Indicateur de l'activité du registre à décalage de transmission TSR
1 → TSR libre, 0 → TSR en activité

Déroulement de la transmission

- Quand on écrit un octet D dans le registre TXREG, le drapeau PIR1.TXIF passe à 0, ensuite, deux situations sont possibles :
- Le registre de transmission TSR n'est pas occupé, alors la donnée D est transférée immédiatement dans TSR qui commence sa transmission bit bar bit. Le drapeau TXIF repasse à 1 pour nous dire que nous pouvons de nouveau écrire dans TXREG.
- Le registre de transmission TSR est occupé à transmettre un octet qui lui été donné auparavant. La donnée D attend dans TXREG, et le drapeau TXIF reste à 0 jusqu'à ce que TSR termine de transmettre l'octet précédent. La donnée D est alors transférée dans TSR qui commence sa transmission bit bar bit. Le drapeau TXIF repasse à 1 pour nous dire que nous pouvons de nouveau écrire dans TXREG.

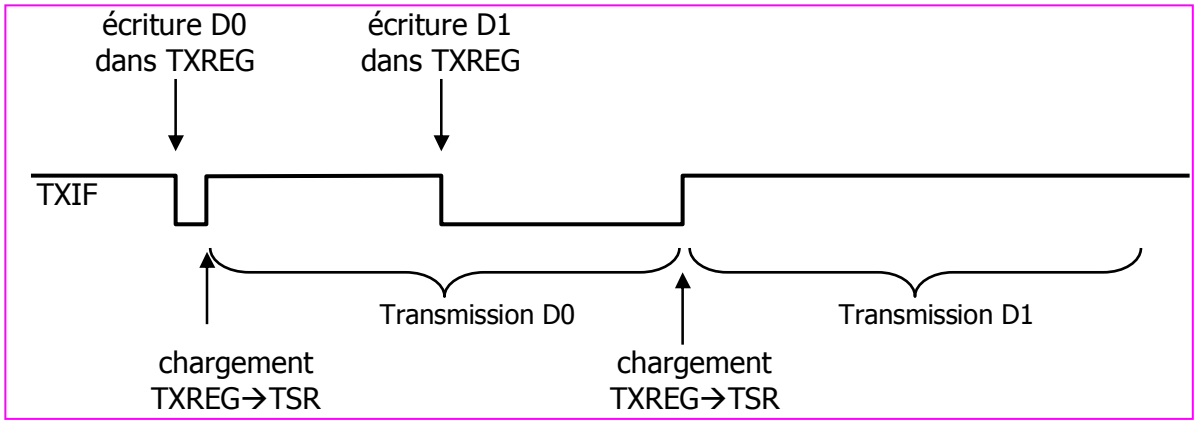


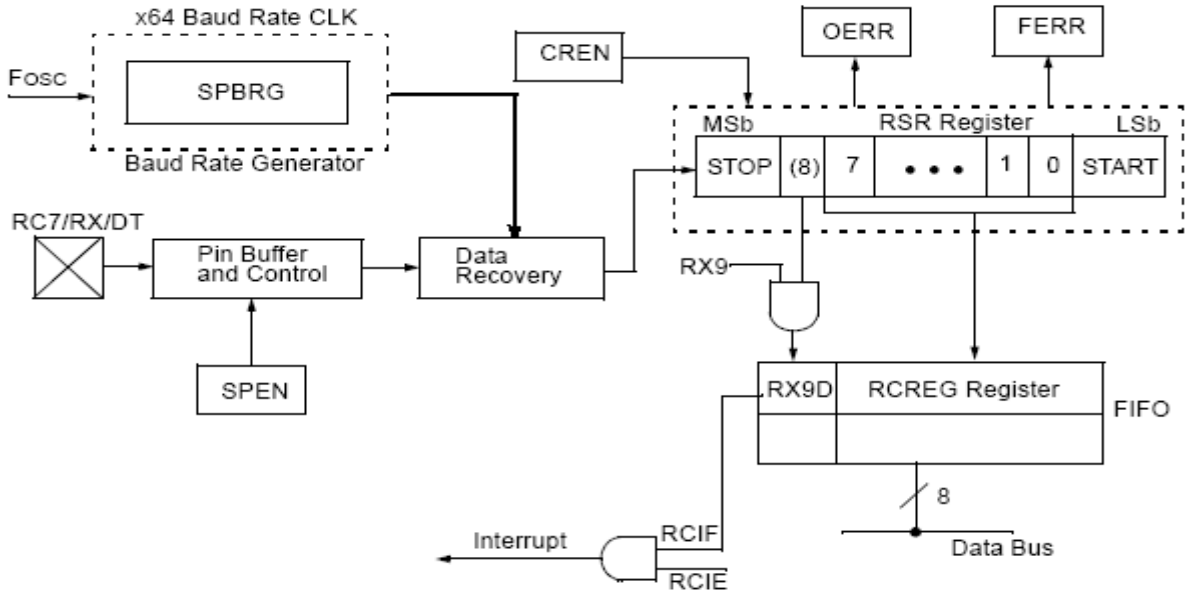
Figure IX-1 : illustration de la transmission de deux octet successifs D_0 et D_1

- A la mise sous tension, tous les drapeaux sont à zéro y compris TXIF. Dès qu'on valide la transmission en positionnant le bit TXEN, le drapeau TXIF passe automatiquement à 1, sauf si on écrit d'abord un octet dans TXREG et on valide ensuite le bit TXEN, dans ce cas, dès la validation de TXEN, le contenu de TXREG est transféré dans TSR qui commence sa transmission et le drapeau TXIF passe à 1.
- Comme on vient de le voir, le drapeau TXIF est géré automatiquement, on ne peut pas le modifier directement par programme.

IX.2.1 Les étapes de transmission (sans interruption, mode 8 bits)

1) S'assurer que l'interruption TXI n'est pas validée
2) Configurer la broche TX/RC6 en entrée
3) Configurer le registre TXSTA (mode 8 bits, valider transmission, asynchrone, BRGH)
4) Initialiser le registre SPBRG pour définir la vitesse de transmission
5) Valider le port avec le bit RCSTA.SPEN
6) Vérifier que le drapeau PIR1.TXIF=1 c.à.d TXREG est vide
7) Placer la donnée à transmettre dans le registre TXREG
8) recommencer au point 6) tant qu'on a des données à transmettre

IX.3 Le port en réception



- La réception est validée par le bit CREN
- La réception d'un octet démarre à la réception du START bit qui commence toujours par une transition 1 → 0
- A la réception du stop bit le contenu du registre à décalage de réception RSR est recopié dans le registre tampon de réception RCREG. Le drapeau RCIF (PIR1.5) est positionné, l'interruption associée est déclenchée si elle est validée. Le drapeau RCIF est remis à zéro automatiquement au moment de la lecture dans RCREG.
- Le registre RCREG est un registre double (FIFO à 2 positions). On peut donc avoir 2 octets en attente dans ce registre et être en train de recevoir un 3ème dans le registre à décalage RSR. A la fin de la réception du 3ème octet, si RCREG est toujours plein, alors le dernier octet reçu est perdu et le bit OERR (Overrun ERRor bit) est positionné ce qui provoque l'arrêt des transfert du registre à décalage RSR vers le buffer RCREG. Pour reprendre la réception il faut réinitialiser le module de réception en mettant à 0 puis à 1 le bit CREN (↺).
- Le drapeau PIR1.RCIF ne passe à 0 que quand la pile RCREG est vide
- Si on reçoit un 0 à la position du STOP bit qui doit être toujours à 1, alors le bit FERR (Framing ERRor) est positionné. Ce bit ainsi que le 9ème bit (si utilisé) sont *bufferisés*, Il doivent être lu dans le registre RCSTA avant la lecture du registre RCREG.

Le contrôle de la réception se fait par le registre RCSTA

SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

SPEN : Validation du port série (1 = validé, 0 = Inhibé)
RX9 : Validation du mode 9 bits (1 → mode 9 bits, 0 → mode 8 bits)
SREN : validation de réception d'u seul octet (non utilisé en mode asynchrone)
CREN : Validation du mode réception continue (1 → validé, 0 → inhibé)
ADDEN : validation du mode détection d'adresse en mode 9 bits, utilisé en mode multiprocesseurs (1 → validé, 0 : Inhibé)
FERR : Erreur de synchronisation, lecture seule. (Voir au dessus)
OERR : Erreur débordement du buffer de réception, lecture seule. (Voir au dessus)
RX9D : En mode 9 bits, le 9ème bit est reçu ici

IX.3.1 Les étapes de réception (sans interruption, mode 8 bits)

1) S'assurer que l'interruption RCI n'est pas validée
2) Configurer la broche RX/RC7 en entrée
3) Initialiser le registre SPBRG pour définir la vitesse de communication
4) Configurer le registre TXSTA (asynchrone, BRGH)
5) Configurer le registre RCSTA (validation port, mode 8 bits, valider réception continue)
6) Attendre que drapeau RCIF passe à 1 ce qui annonce la fin de réception d'un octet
7) Lire l'octet reçu dans le registre RCREG
8) recommencer au point 6) tant qu'on a des données à recevoir

IX.4 La vitesse de communication

La vitesse de communication est déterminée par le générateur de rythme BRG (Baud Rate Generator) qui est dédié au port série. La vitesse de communication est définie à l'aide du registre de control SPBRG et du bit BRGH (TXSTA.2) qui quadruple la vitesse quand il est positionné. Il préférable d'utiliser le mode haute vitesse (BRGH=1) car permet d'obtenir une meilleur précision sur la fréquence.

$$vitesse = \frac{4^{BRGH} \times Fosc}{64 \times (SPBRG + 1)} \ baud$$

$$SPBRG = \frac{4^{BRGH} \times Fosc}{64 \times vitesse} - 1$$

FOSC	1000000			4000000			8000000			20000000		
	SPBRG	Baude	err%	SPBRG	Baude	err%	SPBRG	Baude	err%	SPBRG	Baude	err%
2400	**6**	2232	7	25	2404	0,17	**51**	2404	0,17	**129**	2404	0,17
4800	**2**	5208	8,5	12	4808	0,17	**25**	4808	0,17	**64**	4808	0,17
9600	**1**	7813	18,61	6	8929	6,99	**12**	9615	0,16	**32**	9470	1,35
19200	**0**	15625	18,62	2	20833	8,51	**6**	17857	6,99	**15**	19531	1,72

*Tableau IX.1 : vitesse de transmission **BRGH = 0***

FOSC	1000000			4000000			8000000			20000000		
	SPBRG	Baude	err%	SPBRG	Baude	err%	SPBRG	Baude	err%	SPBRG	Baude	err%
2400	**25**	2404	0,17	**103**	2404	0,17	**207**	2404	0,17			
4800	**12**	4808	0,17	**51**	4808	0,17	**103**	4808	0,17			
9600	**6**	8929	6,99	**25**	9615	0,16	**51**	9615	0,16	**129**	9615	0,16
19200	**2**	20833	8,51	**12**	19231	0,16	**25**	19231	0,16	**64**	19231	0,16
28800	**1**	31250	8,51	**8**	27778	3,55	**16**	29412	2,13	**42**	29070	0,94
33600	**1**	31250	6,99	**6**	35714	6,29	**14**	33333	0,79	**36**	33784	0,55

*Tableau IX.2 : vitesse de transmission **BRGH = 1***

Exercice 31)

On dispose d'un PIC doté d'un quartz de 4MHz.
Donner le programme qui transmet l'alphabet sur le port série à 19200 baud avec un délai voisin de ½ s entre les caractères.

Exercice 32)

On dispose d'un PIC doté d'un quartz de 10MHz.
Donner le programme qui lit 40 caractères sur le port série (9600 baud) et les stocke dans la RAM à partir de la position 190h.

X Le module MSSP (Master Synchronous Serial Port)

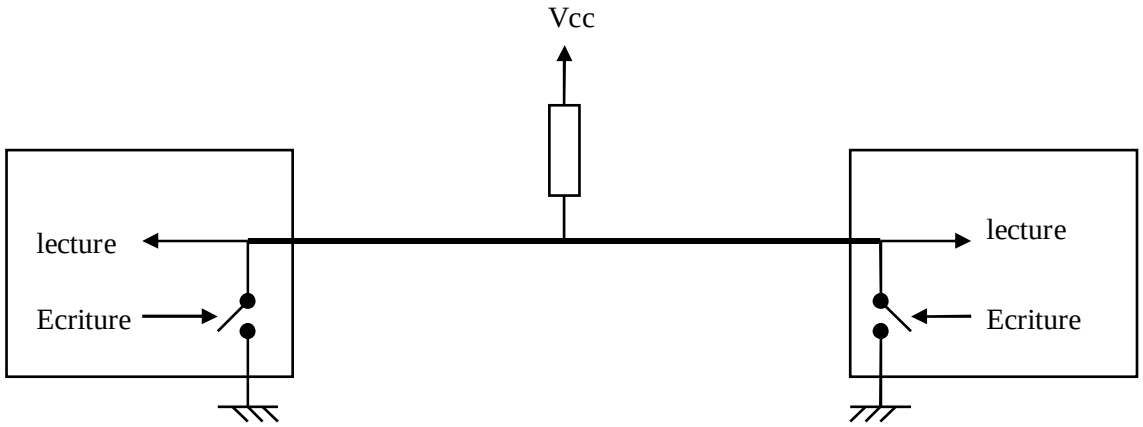
Le MSSP est une des deux modules de communication série du PIC 16F876/877. Il permet d'échanger des données en mode synchrone avec d'autres circuits qui peuvent être des microcontrôleurs, des mémoires EEPROM série, des convertisseurs A/N, des modules d'affichage . . . Il peut fonctionner selon deux modes : le mode SPI (Serial Peripheral Interface) et le mode I2C (Inter-Integrated Circuit)

X.1 Introduction au bus I2C

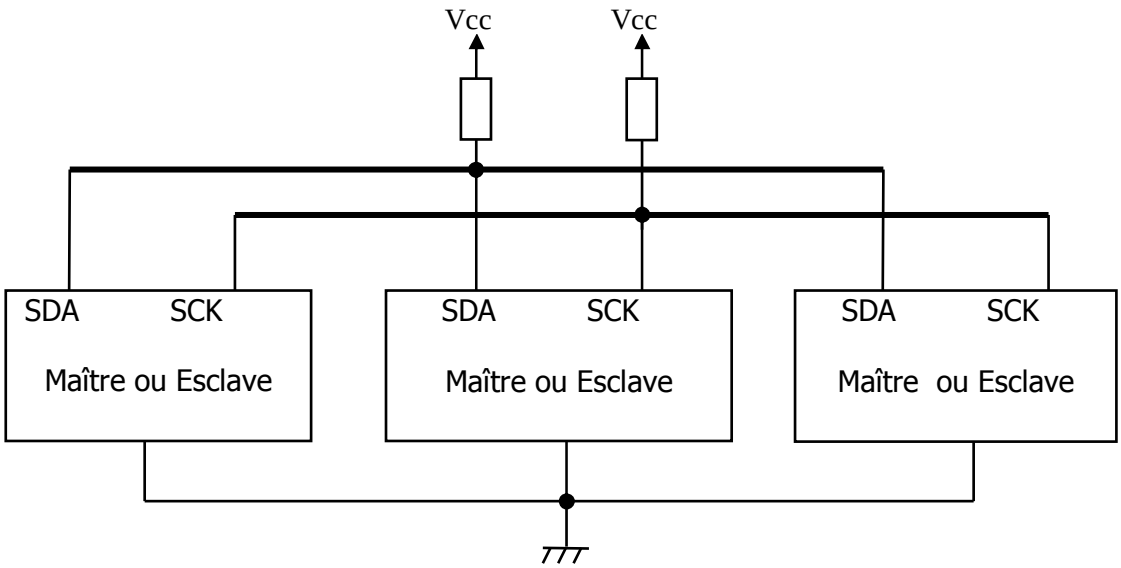
Avant de parler du module MSSP en mode I2C du PIC, introduisons très brièvement Le standard I2C

Le bus I²C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement 3 fils : Un signal de donnée (SDA), un signal d'horloge (SCL), et un signal de référence électrique (Masse).

Comme les lignes SDA et SCK sont utilisées dans les deux sens par les deux circuits qui communiquent, on peut avoir un circuit qui place la ligne à 1 (Vcc) et l'autre qui la place à 0 (masse) ce qui correspond à un court circuit qui peut détruire les deux composants. Pour éviter ce problème, les E/S SDA et SCK fonctionnent en mode collecteur ouvert (ou drain ouvert) de sorte qu'un circuit ne peut imposer que le niveau bas ou ouvrir la ligne, le niveau haut est obtenu par une résistance de tirage externe. Ainsi une ligne est à 0 quand un des deux circuits impose le 0. Elle passe à 1 quand les deux circuits imposent le 1 (circuit ouvert). Le protocole I2C jongle avec cette situation pour organiser l'échange des données entre les deux composants.



Un bus I2C peut être relié à plusieurs circuits, mais pendant une communication, un des circuits est le maître, c'est lui génère l'horloge et initie les séquences de transmission, l'autre est l'esclave, il subit l'horloge du maître sur la ligne SCK mais il peut tout de même recevoir et émettre des données sur la ligne SDA. Chaque esclave a une adresse, au début d'une séquence de communication, le maître qui initie la séquence envoie l'adresse du slave avec lequel il désire communiquer, celui-ci reconnaît son adresse et répond, les autres slaves (s'il y en a) restent muets.

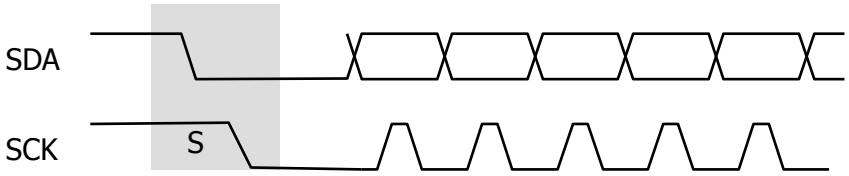


Certains circuits sont fabriqués pour être des masters, d'autres des slaves et d'autres peuvent être soit l'un soit l'autre.

Pour prendre le contrôle du bus, il faut que celui-ci soit au repos (SDA et SCL à '1'). Lorsqu'un circuit prend le contrôle du bus, il en devient le maître. C'est lui qui génère le signal d'horloge et c'est lui qui initie les séquences d'échange.

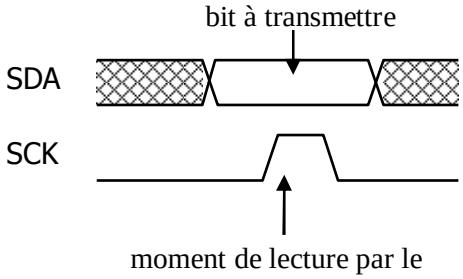
X.1.1 start condition

Au début d'une séquence d'échange, le master génère un start condition (S) avant de commencer l'échange de données. Au repos, les lignes SCL et SDA sont à l'état haut (relâchées). Pour générer un start, le master place d'abord la ligne SDA à 0, ensuite il place SCK à 0.



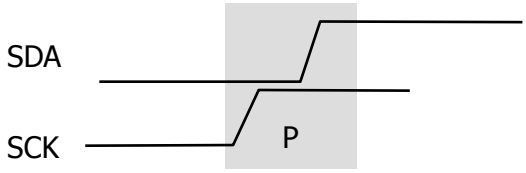
X.1.2 Transmission d'un bit

On place le bit à transmettre sur la ligne SDA ensuite on envoie une impulsion d'horloge sur la ligne SCK. C'est cette impulsion qui informe le slave qu'il doit lire la donnée sur SDA



X.1.3 Stop condition

A la fin d'une séquence d'échange, le master génère un stop condition (P) après lequel le bus est de nouveau libre. Pour cela, à partir de la situation SDA=0, SCK=0, le master commence par placer SCK à 1 et place ensuite SDA à 1.

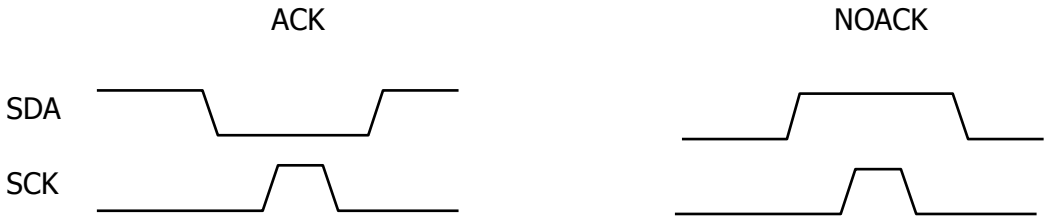


X.1.4 Remarque sur le Start et le Stop condition

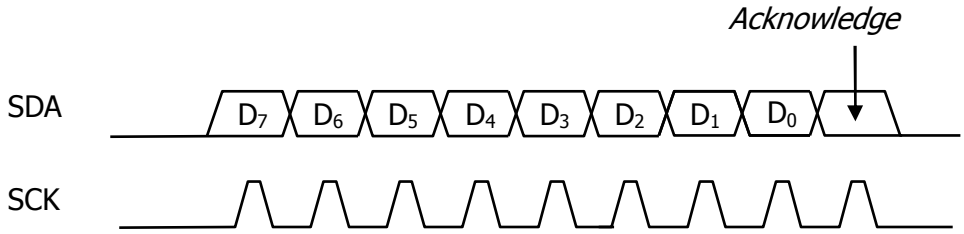
Une séquence de transmission peut contenir plusieurs *Starts conditions* avant de rencontrer un *Stop Condition*. On parle de *repeated Start condition*. Un Stop condition est toujours synonyme de FIN de transmission.

X.1.5 L' acknowledge

L' *acknowledge* est l'accusé de réception. Il est placé par le circuit qui reçoit sur la ligne SDA juste après la réception du 8ème bit. C'est l'émetteur qui le lit de la même façon qu'on lit un bit ordinaire. SDA=0 → *acknowledge* positif (ACK), SDA=1 → *acknowledge* négatif (NOACK),

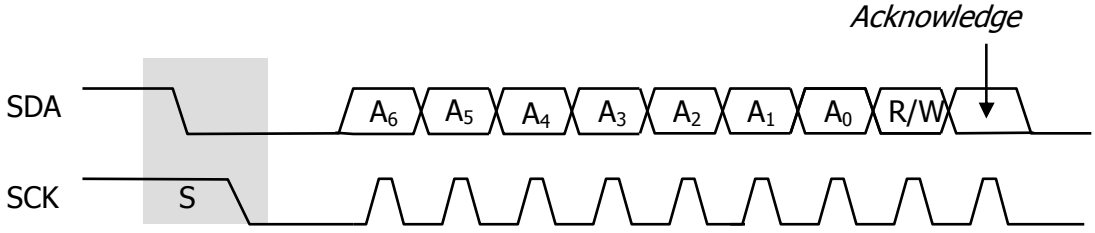


En conclusion, l'échange d'un octet nécessite 8 + 1 = 9 impulsions d'horloge sur la pin SCL.



X.1.6 L'adresse et le bit R/W

Comme on peut brancher plusieurs composant sur un bus I2C, il est nécessaire de définir une adresse unique pour chacun. Elle est codée sur 7 bits A_6 A_5 A_4 A_3 A_2 A_1 A_0. Le master qui démarre une séquence d'échange envoie l'adresse du slave juste après le *start condition*. Comme il y a seulement 7 bits, le master envoie à la 8ème position le bit R/W pour indiquer au slave s'il désire une émission (R/W=0) ou une réception (R/W=1).



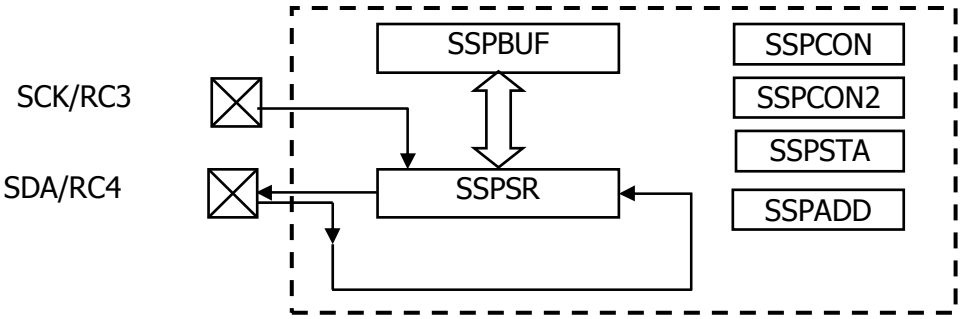
Le standard I2C supporte maintenant l'adressage sur 10 bits. Comme avec 7 bits on peut adresser jusqu'à 128 composants, cela nous suffira largement.

X.2 Le module MSSP en mode I2C

Le module MSSP du PIC peut être configuré en master ou en slave. Il utilise les broches RC3/SCL (Horloge) et RC4/SDA (données). Ces broches doivent être configurées en ENTREE à l'aide du registre TRISC et doivent être munie de résistances de pull-up externes nécessaire au fonctionnement I2C.

Les fréquences d'horloges supportées sont 100 kHz, 400 kHz et 1 MHz

L'accès au module en lecture et écriture se fait à l'aide du registre tampon (buffer) SSPBUFF. La transmission et la réception se fait à l'aide du registre à décalage SSPSR auquel nous n'avons pas directement accès



X.2.1 Transmission d'un octet

Pour transmettre un octet, il suffit de le copier dans le registre SSPBUF, et le module MSSP s'occupe du reste. Au moment de l'écriture dans SSPBUF, le bit BF passe à 1 et la transmission commence. A la fin de la transmission, le bit SSPSTAT.BF repasse à 0 et le drapeau d'interruption PIR1.SSPIF passe à 1.

Le bit BF apparaît donc comme un bit très important, c'est lui qui nous permet de savoir si le registre SSPBUF est libre ou non

Si on tante d'écrire dans SSPBUF alors que BF=1, le bit SSPCON.WCOL passe à 1 pour indiquer une collision et l'écriture n'a pas lieu.

X.2.2 Réception d'un octet

A la fin de la réception d'un octet, celui-ci est transféré dans SSPBUF, l'indicateur SSPSTAT.BF et le drapeau d'interruption PIR1.SSPIF passent à 1. BF repasse automatiquement à 0 au moment de la lecture de SSPBUF alors que SSPIF soit être remis à 0 par soft.

Si le PIC termine la réception d'un octet avant que l'octet précédent qui se trouve dans SSPBUF n'ait été lu, on a un Overflow qui sera signalé par le drapeau SSPOV. Le transfert n'a pas lieu, l'octet arrivé est perdu.

X.2.3 Les registres de configuration

Le control du module se fait à l'aide des registres :

SSPCON : registre de control (BANK0)
SSPCON2 : registre de control (BANK1)
SSPSTAT : registre d'état (BANK1)
SSPADD : registre d'adresse (BANK1)

X.2.3.1 Le registre SSPSTAT :

SSPSTAT	SMP	CKE	D/A	P	S	R_W	UA	BF

SMP : Control de slew rate : 1 pour 100 kHz et 1 MHz, 0 pour 400 kHz
CKE : Control des niveaux de tension à l'entrée, 0 : I2C, 1 : SMBUS
D/A : Indicateur d'état, 0 : dernier octet émis/reçu = adresse, 1 : dernier octet émis/reçu = donnée
P : Indicateur de Stop Condition. Ce bit passe à 1 quand on détecte un Stop Condition. Il est automatiquement remis à 0 quand un autre événement et reçu.
S : Indicateur de Start Condition. Ce bit passe à 1 quand on détecte un Start Condition. Il est automatiquement remis à 0 quand un autre événement et reçu
R_W : Indicateur de lecture écriture
Mode master : 0 : pas de transmission en cours, 1 : transmission en cours
Mode slave : image du bit R/W qui constitue le bit 0 du premier octet suivant le start condition. 0 : écriture, 1 : lecture
UA : Indicateur utilisé en mode esclave avec adressage 10 bits pour nous informer qu'il faut placer la $2^{ème}$ partie de l'adresse dans le registre adresse. Ce dernier ayant seulement 8 bits, l'adresse est traitée en 2 parties. Ce bit est remis automatiquement à zéro après l'écriture dans SSPADD.
BF : Indicateur sur l'état du registre SSPBUF.

X.2.3.2 Le registre SSPCON :

SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

WCOL : Collision d'écriture. En mode I2C master, ce bit passe à 1 si on tente d'écrire dans SSPBUF à mauvais moment de la séquence de transmission (BF=1). En mode I2C slave, il est positionné si on tente d'écrire dans SSBUF avant la fin de la transmission de l'octet précédant. Ce bit doit être remis à zéro par programme.

SSPOV : Overflow, cet indicateur d'erreur est positionné quand une nouvelle donnée est reçue alors que le registre SSBUF n'est pas encore lu. Dans ce cas la nouvelle donnée est perdue. Ce bit doit être remis à zéro par programme.

SSPEN : Validation du module MSSP (1 = validé)

CKP : Utilisé en mode I2C slave pour générer des pauses en inhibant l'horloge.
0 : horloge forcée à zéro, 1 : horloge validée

SSPM3:SSPM0 : mode de fonctionnement du module
0110 : I2C Slave, adresse 7 bits
0111 : I2C Slave, adresse 10 bits
1000 : I2C master, fréquence = F_{OSC} / (4 * (SSPADD+1))
1011 : I2C Slave forcé à l'état de repos
1110 : I2C master, adresse 7 bits, interruption sur START et STOP
1110 : I2C master, adresse 10 bits, interruption sur START et STOP

X.2.3.3 Le registre SSPCON2 :

SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN

Le bit 7 concerne le mode slave, les autres bits concernent le mode master :

GCEN : *General call enable bit* : Si ce bit est validé, le PIC répond à son adresse et à l'adresse générale 0000h

ACKSTAT : *Acknowledge Status* bit (mode master) : ce bit est l'image de l'accusé de réception envoyé par le slave. 0=accusé positif, 1=accusé négatif

ACKDT : *Acknowledge Data* bit (master mode) : Valeur du bit envoyé au slave en fin de réception. Ce bit est envoyé au moment ou on positionne le bit ACKEN.

ACKEN : *Acknowledge Sequence Enable bit* (master mode) : quand il est placé à un, ce bit démarre une séquence accusé de réception qui consiste à placer les broches SDA et SCL dans le mode correspondant au bit ACKDT

RCEN : Receive Enable bit : (master, réception) Quand on le place à 1, ce bit démarre la réception d'un octet. Il revient automatiquement à 0 après la transmission du 8ème bit. La réception ne démarre que si l'horloge n'est pas *stretchée* (forcée à 0: pause) par le slave.

PEN : STOP Condition Enable bit , démarre une séquence stoP_condition sur les broches SDA et SCL. Ce bit est remis à 0 automatiquement après la séquence du STOP

RSEN : Repeated START Condition Enable bit : démarre une séquence Repeated Start condition. Il est remis à 0 automatiquement après la séquence du R-START

SEN : START Condition Enable bit : démarre une séquence Start condition. Il est remis à 0 automatiquement après la séquence du START

X.2.3.4 Le registre SSPADD :

En mode I2C **master**, ce registre permet de déterminer la fréquence de communication

$$F = \frac{Fosc}{4 \times (SSPADD + 1)} \; Hz$$

Les fréquences possibles sont : 100 kHz, 400 kHz et 1 MHz

$$SSPADD = \frac{Fosc}{4 \times F} - 1 \; Hz$$

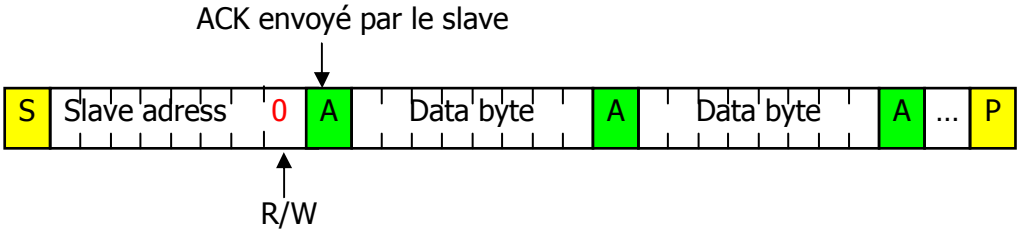
En mode I2C **slave**, ce registre doit contenir l'adresse du slave. Le bit 0 est réservé, il doit être toujours placé à zéro, l'adresse doit être écrite à partir du bit 1. En cas d'adresse 7 bits, cela ne pose pas de problème. En cas d'adresse 10 bits, l'adresse est écrite en 2 temps, On commence par écrire 1 1 1 1 0 A_9 A_8 0 dans SSPADD, en attend l'indicateur UA, ensuite on écrit A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0

X.2.4 MSCP en mode I2C Master

Une séquence de communication est toujours initiée par le master qui commence par envoyer un **S**tart_bit (SSPCON2.**S**EN), suivi de l'adresse du slave. L'adresse sera codée seulement sur 7 bits, Le LSB qu'on appelle bit R/W permet de préciser le sens de l'échange qui va suivre, R/W=0 pour transmission, R/W=1 pour réception. On envoie ou on reçoit les données et on termine la communication par un sto**P**_bit (SSPCON2.**P**EN)

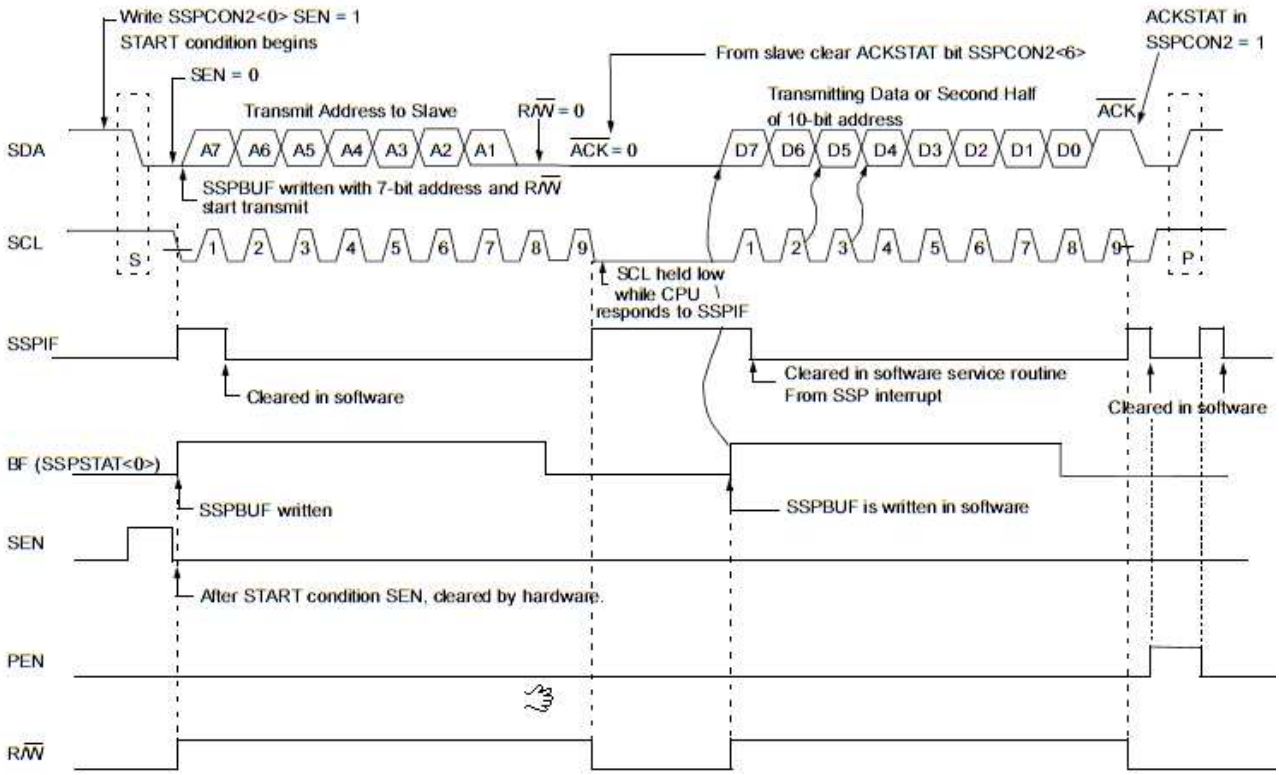
Si l'on désire changer le sens de l'échange avant l'envoi du stoP, ce qui évite l'éventualité de perdre la ligne au profit d'un autre master, il faut envoyer un RepeatStart condition suivi de l'adresse accompagnée d'un R/W différent

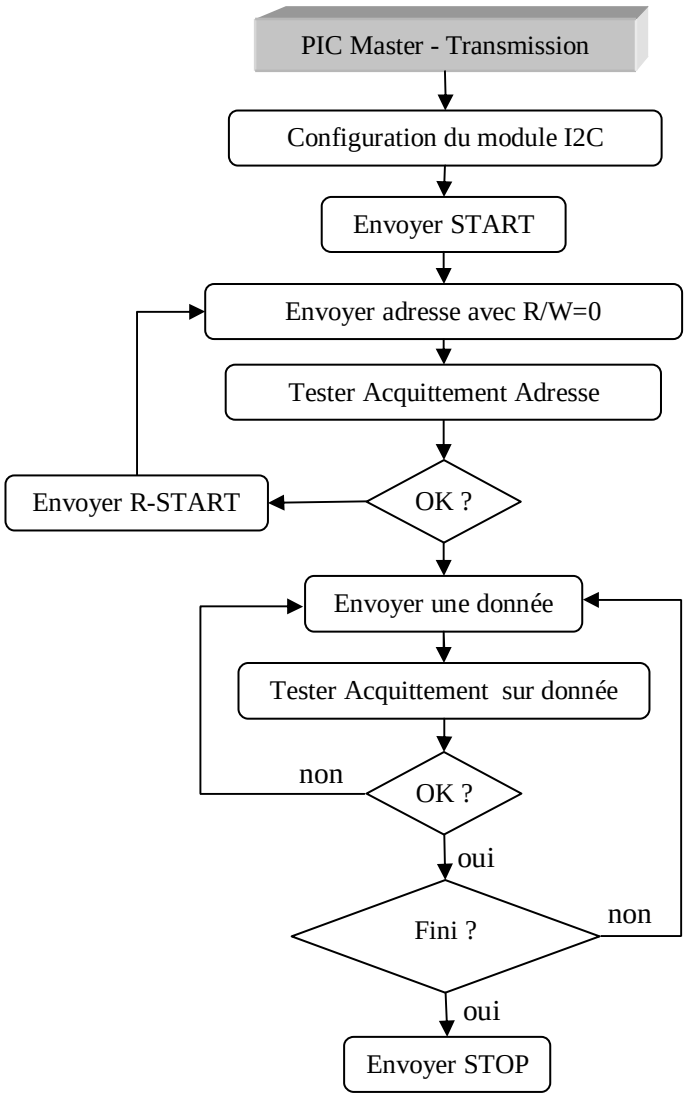
X.2.4.1 Master en transmission :



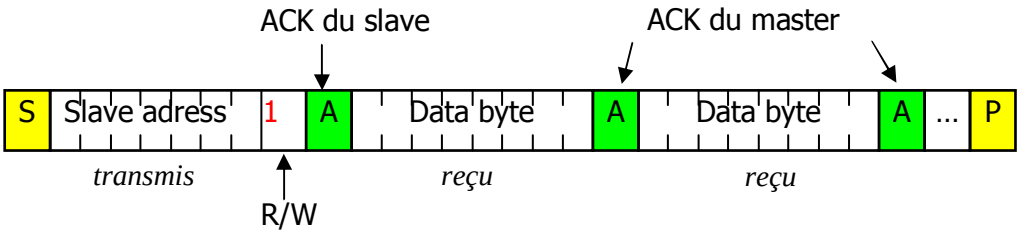
- On positionne le bit SSPCON2.SEN sur le master, celui-ci envoie le START bit sur la ligne SDA puis r.à.z automatiquement le bit SEN et positionne le drapeau d'interruption PIR1.SSPIF, ce dernier doit être r.a.z par soft. Le slave voit le START, positionne son bit S et se prépare à recevoir l'adresse,

- On copie l'adresse avec R/W=0 dans SSPBUF, les bits SSPSTAT.BF et SSPSTAT.R_W passent à 1, la transmission démarre, après le 8ème coup d'horloge, SSPSTAT.BF repasse à zéro, après le 9ème R_W repasse à 0, et le drapeau d'interruption SSPIF passe à 1 et l'horloge est forcée à 0 (pause) jusqu'`a ce qu'un nouvel octet est copié dans SSPBUF. Le drapeau SSPIF doit être raz par soft pour qu'il puisse servir dans la suite.
- L'acquittement sur l'adresse est renvoyé par le slave pendant le 9ème coup d'horloge sur la ligne SDA : 0 (ACK) s'il a lu l'octet correctement, 1 (NoACK) s'il n'a pas pu lire l'octet correctement. Ce bit recopié par le master dans le bit SSPCON2.ACKSTAT. Le programmeur doit vérifier l'état de ce bit pour décider de la suite des événements,
- Si acquittement de l'adresse est négatif on envoie un R-START et on renvoie ládresse
- Si acquittement de l'adresse est positif, on envoie un octet de donnée en l'écrivant dans SSPBUF. Tout se passe comme lorsqu'on a envoyé l'adresse. Les bit BF et R_W passent à 1 au début de l'émission et repassent à 0 à la fin. Le drapeau SSPIF passe à 1. Le slave renvoie l'acquittement pendant le 9ème bit qui sera copié dans le bit ACKSTAT.
- Si l'acquittement sur la donnée est négatif, le plus simple est de retransmettre la donnée
- Si l'acquittement sur la donnée est positif :
  - Si on a encore des données à transmettre, on démarre une nouvelle émission en écrivant dans SSPBUF.
  - Si on n'a plus de données à transmettre, on envoie un stoP_bit en positionnant le bit SSPCON2.PEN qui est remis à zéro automatiquement après la transmission d'un Stop bit sur la ligne SDA.
- Quand le slave voie le stoP_bit, il r.a.z le bit S et positionne le bit P puis réinitialise son électronique pour se préparer à la réception d'un nouveau Start_bit.

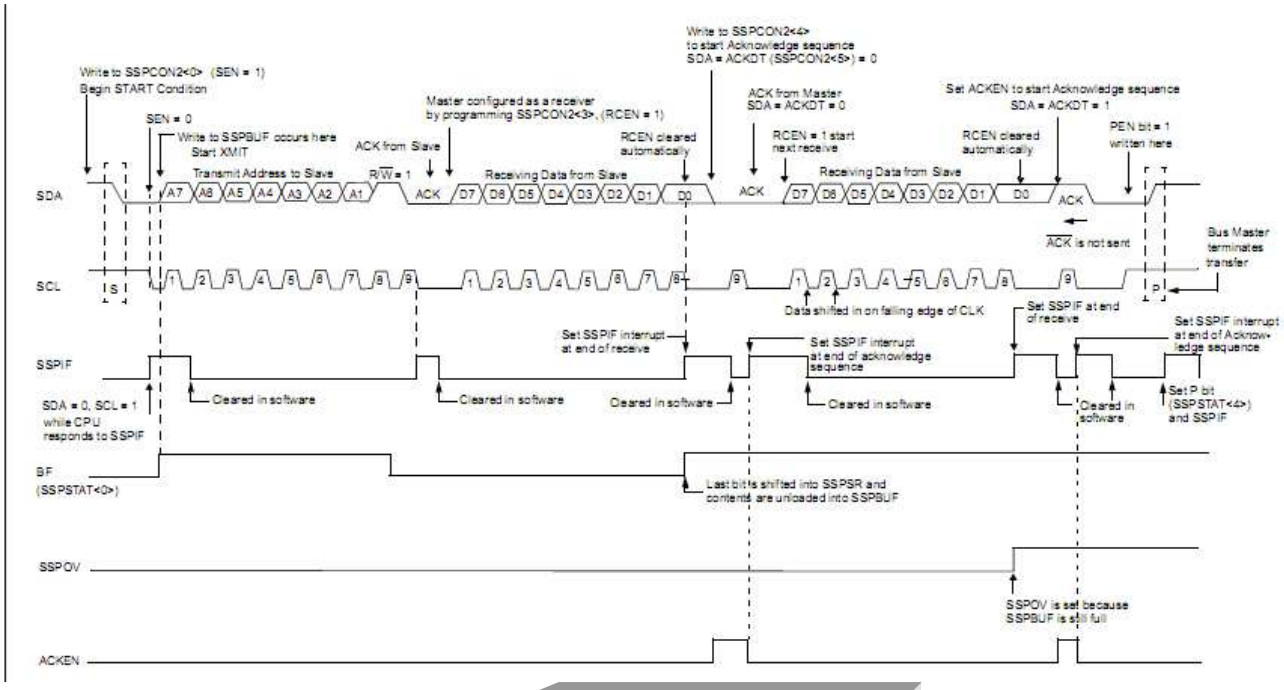

Write SSPCON2<0> SEN = 1
START condition begins
SEN = 0
Transmit Address to Slave
R/W = 0
From slave clear ACKSTAT bit SSPCON2<6>
Transmitting Data or Second Half of 10-bit address
ACKSTAT in SSPCON2 = 1
ACK
SDA
A7 A6 A5 A4 A3 A2 A1
ACK = 0
D7 D6 D5 D4 D3 D2 D1 D0
SSPBUF written with 7-bit address and R/W start transmit
SCL
S
1 2 3 4 5 6 7 8 9
SCL held low while CPU responds to SSPIF
P
SSPIF
Cleared in software
Cleared in software service routine From SSP interrupt
Cleared in software
BF (SSPSTAT<0>)
SSPBUF written
SSPBUF is written in software
SEN
After START condition SEN, cleared by hardware.
PEN
R/W

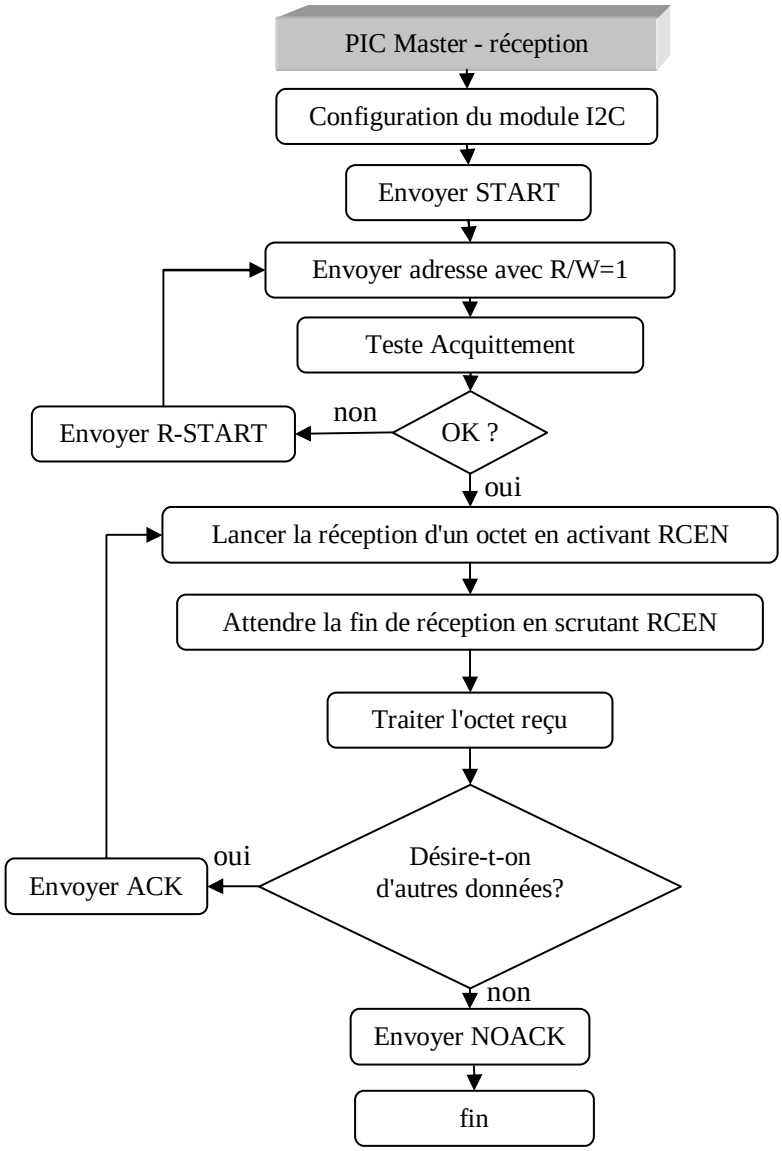

PIC Master - Transmission
Configuration du module I2C
Envoyer START
Envoyer adresse avec R/W=0
Tester Acquittement Adresse
OK ?
Envoyer R-START
Envoyer une donnée
Tester Acquittement sur donnée
non
OK ?
oui
Fini ?
non
oui
Envoyer STOP

X.2.4.2 Master en réception :



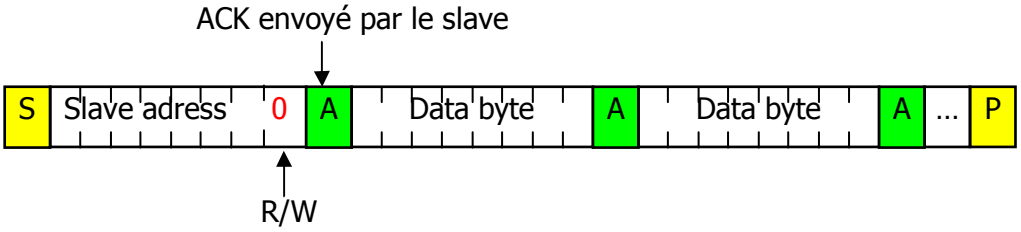
- On positionne le bit SSPCON2.SEN sur le master, celui-ci envoie le Start_bit sur la ligne SDA et r.à.z automatiquement le bit SEN et positionne le drapeau d'interruption PIR1.SSPIF, ce dernier doit être r.a.z par soft. Le slave voit le Start_bit, il place le bit SSPSTAT.S à 1 et se prépare à recevoir l'adresse,
- On copie l'adresse avec R/W=1 dans SSPBUF, les bits SSPSTAT.BF et SSPSTAT.R_W passent à 1, la transmission démarre, après le 8ème coup d'horloge, SSPSTAT.BF repasse à zéro, après le 9ème R_W repasse à 0, et le drapeau d'interruption SSPIF passe à 1, il doit être raz par soft pour qu'il puisse servir dans la suite.
- Pendant le 9ème coup d'horloge, le slave renvoie un ACK/NOACK selon sa situation et force l'horloge à zéro (pause) pour empêcher le master de lire avant que la donnée ne soit préparée sur le bus. On appelle ça : *clock stretching*.
- Le programme du slave prépare la donnée sur le bus puis libère l'horloge à l'aide du bit CKP
- Le programme du master doit tester l'acknowledge dans SSPCON2.ACKSTAT, si = 0(OK), il positionne le bit SSPCON2.RCEN pour demander le début d'une réception.
- Le module I2C master voyant le RCEN=1 essaye de commencer une réception par l'envoi d'une rafale de 8 coups d'horloge. Quand il place l'horloge à 1 (première impulsion), celle-ci peut ne pas passer à 1 car il se peut qu'elle soit forcée à 0 par le slave. Dans ce cas le master attend que l'horloge passe à 1 pour continuer la rafale de coup d'horloge
- En résumé, dès que les deux conditions {RCEN=1 et horloge libre} sont vérifiées, le master envoie une rafale de 8 coups d'horloge qui s'accompagne de la réception de 8 bits de données.
  - A la fin de la réception, le module I2C master ramène le bit SSPCON2.RCEN à 0, transfert l'octet reçu dans SSPBUF, l'indicateur BF passe à 1 ainsi que le drapeau d'interruption PIR1.SSPIF. Ensuite Il force l'horloge à 0 (pause) pour donner le temps au programme d'envoyer l' *Acknowledge,*
- Le programme utilisateur du master vide le buffer SSPBUF, BF repasse à 0. Ensuite il envoie l' *Acknowledge* en plaçant ACK/NOACK (0/1) dans la bit SSPCON2.ACKDT et en validant le bit SSPCON2.ACKEN. Le bit ACKDT est placé sur la ligne SDA, et ACKEN est r.a.z. automatiquement. L'horloge est libérée juste le temps d'un coup d'horloge (le 9ème) pour que le slave puisse prendre l' *Acknowledge* en compte. Elle est remise en pause après. Le drapeau d'interruption SSPIF est aussi positionné après l'envoi du ACK
- Si on a envoyé ACK, ça veut dire qu'on veut recevoir d'autres données, alors notre programme peut initier la réception d'un nouvel octet en validant SSPCON2.RCEN.
- A la réception du ACK, le slave met l'horloge en mode pose pour donner le temps à son programme de placer une donnée dans le registre SSPBUF et de libérer ensuite l'horloge à l'aide du bit CKP.
- Si on a envoyé NOACK, (on ne veut plus de donnée), Le slave qui le reçoit réinitialise sa logique comme s'il a reçu un STOP.


Write to SSPCON2<0> (SEN = 1)
Begin START Condition
SEN = 0
Write to SSPBUF occurs here
Start XMIT
Transmit Address to Slave
ACK from Slave
R/W = 1
Master configured as a receiver
by programming SSPCON2<3>, (RCEN = 1)
Receiving Data from Slave
RCEN cleared
automatically
Write to SSPCON2<4>
to start Acknowledge sequence
SDA = ACKDT (SSPCON2<5>) = 0
ACK from Master
SDA = ACKDT = 0
RCEN = 1 start
next receive
Receiving Data from Slave
RCEN cleared
automatically
Set ACKEN to start Acknowledge sequence
SDA = ACKDT = 1
PEN bit = 1
written here
ACK is not sent
Bus Master
terminates
transfer
SDA
SCL
SSPIF
BF
(SSPSTAT<0>)
SSPOV
ACKEN
Data shifted in on falling edge of CLK
Set SSPIF interrupt
at end of receive
Set SSPIF interrupt
at end of acknowledge
sequence
Set SSPIF at end
of receive
Set SSPIF interrupt
at end of Acknowledge sequence
Cleared in software
SDA = 0, SCL = 1
while CPU
responds to SSPIF
Set P bit
(SSPSTAT<4>)
and SSPIF
Last bit is shifted into SSPSR and
contents are unloaded into SSPBUF
SSPOV is set because
SSPBUF is still full


PIC Master - réception
Configuration du module I2C
Envoyer START
Envoyer adresse avec R/W=1
Teste Acquittement
non
OK ?
Envoyer R-START
oui
Lancer la réception d'un octet en activant RCEN
Attendre la fin de réception en scrutant RCEN
Traiter l'octet reçu
Désire-t-on
d'autres données?
oui
Envoyer ACK
non
Envoyer NOACK
fin

X.2.5 MSCP en mode I2C Slave

X.2.5.1 Slave en Réception de données (venant du master)

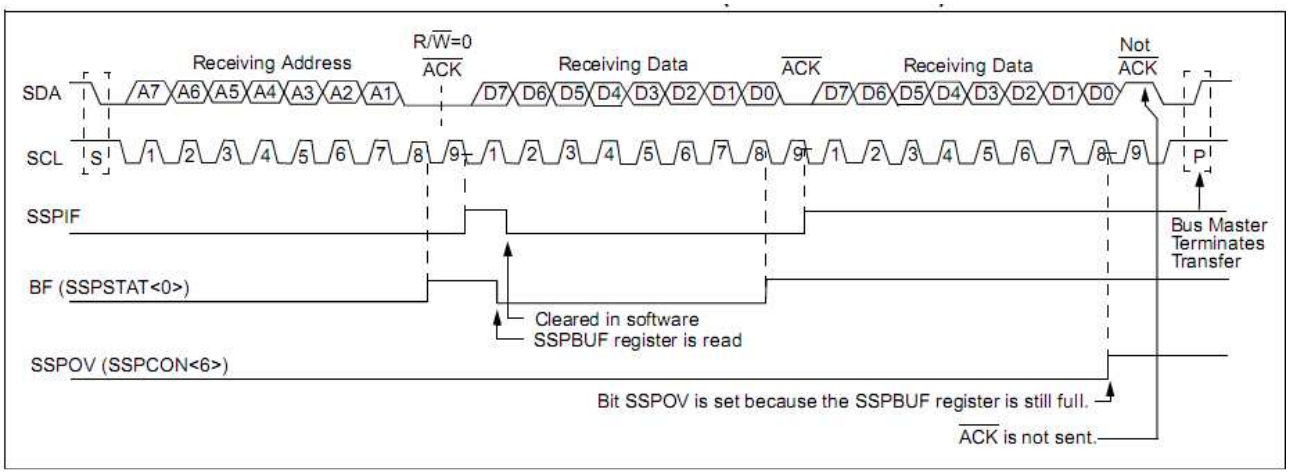


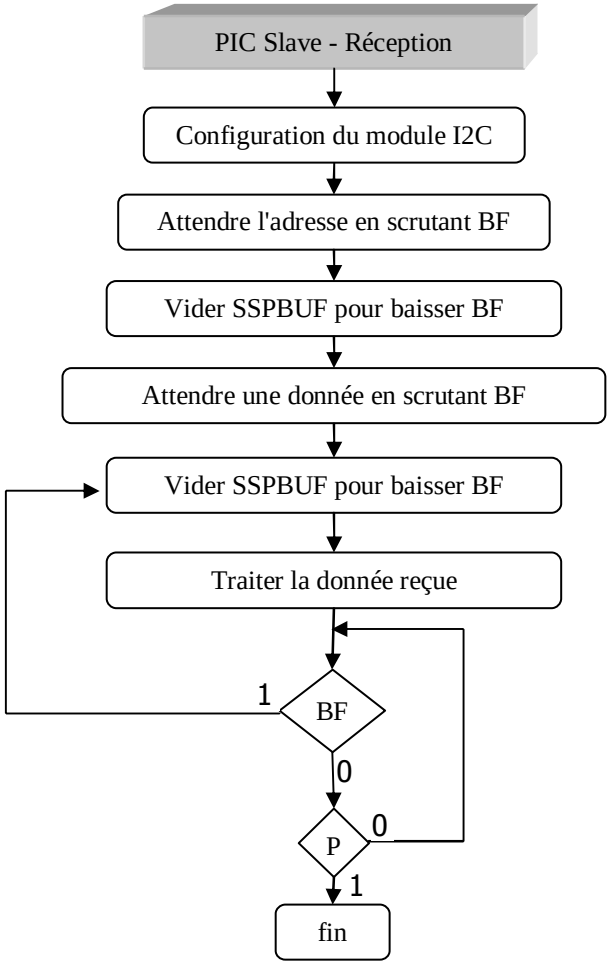
- Le PIC détecte le START sur la ligne SDA, il positionne le bit SSPSTAT.S et se prépare à recevoir l'adresse,
- Le PIC détecte la fin de réception de l'adresse, il vérifie si elle correspond à son adresse.
- Si c'est l'adresse d'un autre slave, il ne réagit pas, il se met à attendre le stoP_bit,
- Si l'adresse reçue correspond à celle de notre PIC :
  - Le LSB R/W est recopié dans SSPSTAT.R_W
  - Le bit SSPSTAT.D_A passe à 0 ⇒ c'est une adresse qui est arrivée
  - la suite dépend de l'état de BF et SSPOV (voir tableau ci-dessus pour plus de détail) :
  - Si BF=0 et SSPOV=0 (tout est OK)
    - SSPSR (l'adresse) est transféré dans SSPBUF → BF passe à 1
    - Un ACK (0) est placé sur la ligne SDA pendant le 9ème coup d'horloge. C'est le PIC qui le fait, notre programme ne s'en occupe pas
    - Le drapeau PIR1.SSPIF passe à 1 ce qui peut déclencher une interruption. Il doit être r.a.z par soft.
  - Si BF=1 (Buffer SSPBUF plein)
    - L'adresse ne sera pas transférée dans SSPBUF
    - SSPOV sera positionné
    - L'ACK ne sera pas transmis (NoAck)
    - Le drapeau PIR1.SSPIF passe à 1 ce qui peut déclencher une interruption, Il doit être r.a.z par soft
  - Si BF=0 et SSPOV=1 (Buffer SSPBUF vide mais il y'a eu un overflow avant)
    - SSPSR (l'adresse) est transférée dans SSPBUF
    - SSPOV reste positionné
    - L'ACK ne sera pas transmis (NoAck)
    - Le drapeau PIR1.SSPIF passe à 1 ce qui peut déclencher une interruption

Status Bits as Data Transfer is Received		SSPSR → SSPBUF	Generate $\overline{\text{ACK}}$ 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

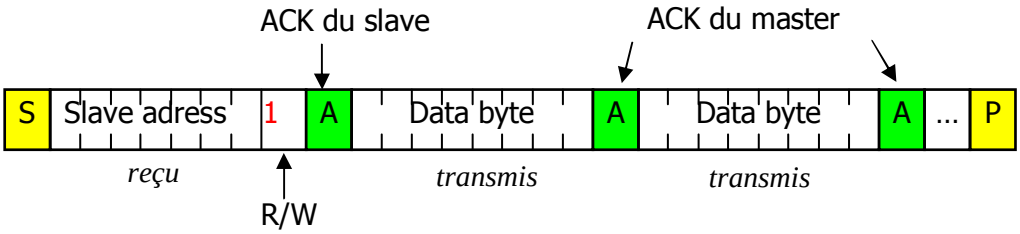
Note: Shaded cells show the conditions where the user software did not properly clear the overflow condition.

- Dans le cas *"tout est OK"*, notre programme doit :
    - Vider SSPBUF pour que BF passe à 0 et qu'on soit prêt pour la suite.
    - On vérifie le bit R_W, on trouve 0, on sait qu'on va recevoir un octet
- Le PIC attend la réception du 8ème bit, vérifie BF et SSPOV (on suppose que tout est OK), il transfert l'octet arrivé dans SSPBUF, BF et D_A passent à 1.
- Le PIC envoie l'acquittement pendant le 9ème bit et positionne SSPIF à 1.
- Notre programme vide SSPBUF, BF repasse à 0
- Maintenant on peut soit recevoir, une autre donnée, soit un stoP_condition, soit un Repeat_Start_Condition. Il faut donc avoir l'œil, sur les indicateurs, BF, SSPIF, P et S pour décider de la suite des opérations





X.2.5.2 Slave en transmission de données (vers le master)

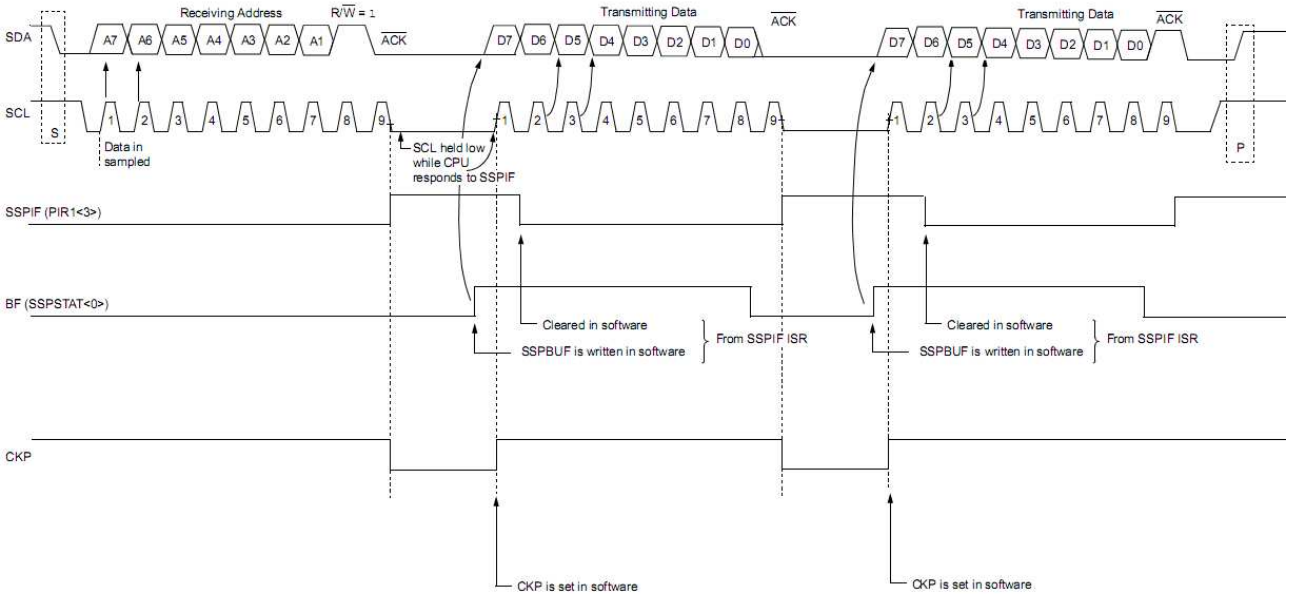


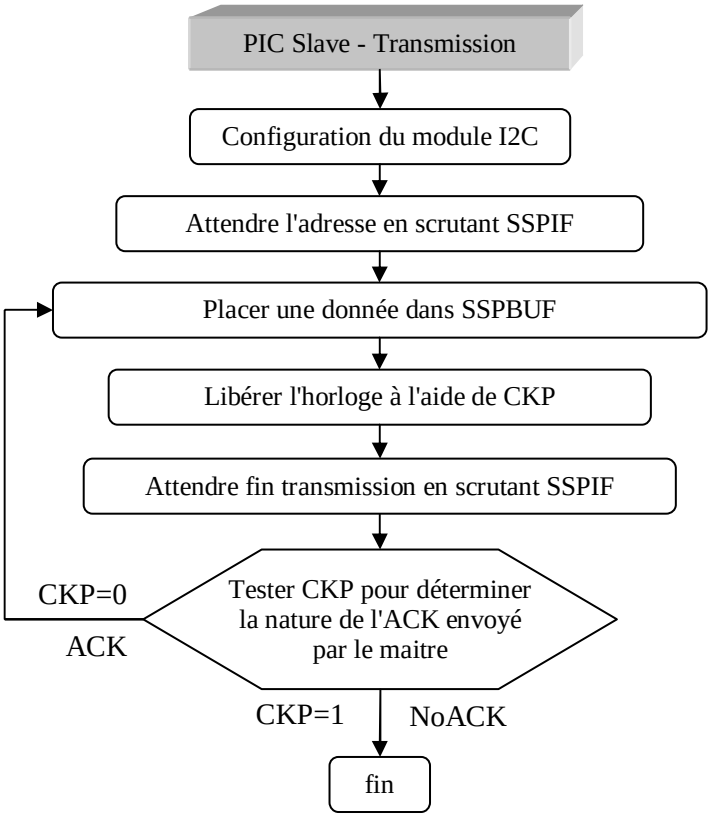
Remarquons que le slave en transmission ne fait que préparer la donnée à transmettre dans son buffer SSPBUF. C'est le master qui décide du début de l'échange puisque c'est lui qui gère l'horloge. Toutefois, le slave peut bloquer un échange en bloquant l'horloge à 0. *Clock Stretching*.

Le drapeau BF passe à 1 quand le programme utilisateur écrit un octet dans SSPBUF et revient à 0 après la transmission du 8 bit de cet octet. Quand au début de l'échange, on reçoit l'adresse, celle-ci est transférée dans SSPBUF et BF reste à 0 dans ce cas.

- Le PIC détecte le Start_bit sur la ligne SDA, positionne le bit SSPSTAT.S et se prépare à recevoir l'adresse,
- Le PIC détecte la fin de réception de l'adresse, il vérifie si elle correspond à son adresse.
- Si c'est l'adresse d'un autre slave, il ne réagit pas, il se met à attendre le stoP_condition,
- Si l'adresse reçue correspond à notre PIC. Le module I2C Slave recopie le bit R/W (1 dans ce cas) dans le bit R_W du registre SSPSTAT , puis vérifie les bits BF et SSPOV pour savoir si le contenu précèdent du registre SSPBUF à été lu ou non
  - Supposons que tout est OK, BF=0 et SSPOV=0
    - Il recopie SSPSR (l'adresse) dans SSPBUF → **BF ne passe pas à 1**
    - Il place un ACK (0) sur la ligne SDA pendant le 9ème coup d'hologe. C'est le PIC qui le fait, notre programme ne s'en occupe pas
    - Il place le bit CKP à 0 ce qui force l'horloge à 0.(Mode pause : Clock Stretching)
    - Le drapeau PIR1.SSPIF passe à 1 ce qui peut déclencher une interruption, il doit être r.a.z par soft
    - Le bit SSPSTAT.D_A passe à 0 ⇒ c'est une adresse qui est arrivée
- Notre programme détecte que SSPIF est passé à 1 :
  - Si on vérifie le bit R_W, on trouve 1, on sait qu'on va envoyer un octet
  - On n'a pas besoin de lire SSPBUF car BF est égal à 0
  - On copie un octet de donnée dans SSPBUF. L'indicateur SSPSTAT.BF passe à 1
  - On replace le bit CKP à 1 pour libérer l'horloge. C'est le moyen d'informer le master que la donnée est prête.
  - Le master détecte, la libération de l'horloge, envoie une rafale de coup d'horloge, l'octet est transmis. L'indicateur SSPSTAT.BF repasse à 0 après le 8ème
  - Maintenant le master peut renvoyer soit un ACK, soit NoACK :
    - Si c'est un ACK, le PIC (slave) positionne le drapeau SSPIF et r.a.z le bit CKP pour mettre l'horloge en pause. Ceci donne au programme du slave le temps de préparer une nouvelle donnée dans SSPBUF et libérer l'horloge ensuite. Si le master envoie un STOP après le ACK, le slave ne le détecte pas.

- Si c'est un NOACK, le PIC (slave) positionne le drapeau SSPIF, le bit CKP reste à 1 et le bit R_W passe à 0. L'électronique du slave est réinitialisée, il est prêt à recevoir un nouveau START. Normalement, le master n'a pas besoin d'envoyer un STOP dans ce cas. S'il le fait quand même, le slave le détecte et positionne l'indicateur P.

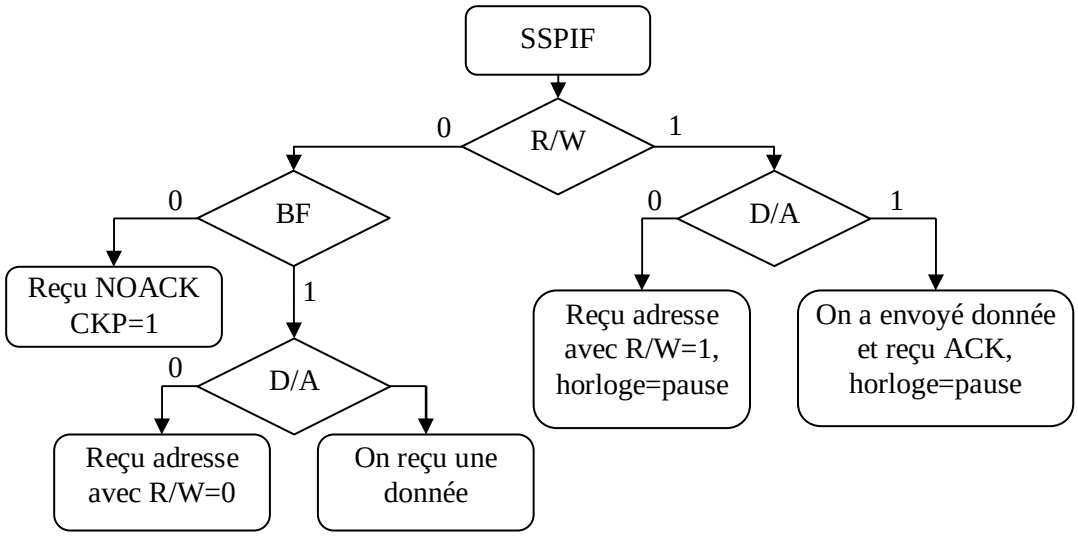




X.2.5.3 Utilisation des interruptions

Voici les 5 événements qui déclenchent l'interruption SSPI sur un PIC Slave.

Evénement	R/W	D/A	BF	CKP	Action
On a reçu une adresse avec R/W=0	0	0	1	NU	Il faut vider SSPBUF pour baisser le drapeau BF et attendre une donnée
On a reçu une donnée	0	1	1	NU	Il faut vider SSPBUF et attendre un événement (qui peut être un STOP)
On a reçu une adresse avec R/W=1. L'horloge est forcée en pause	1	0	0	0	Il faut placer une donnée dans SSPBUF et libérer l'horloge en positionnant le bit CKP
On a fini la transmission d'un octet et reçu un ACK. L'horloge est forcée en pause	1	1	0	0	Il faut placer une donnée dans SSPBUF et libérer l'horloge en positionnant le bit CKP
On dirait qu'on a reçu une donnée mais le buffer reste vide et l'horloge n'est pas pausée: anormal. C'est un NOACK qui a été reçu	0	1	0	1	La séquence d'échange est finie



X.3 Le module MSSP en mode SPI

Ce mode (Serial Peripheral Interface) permet de transmettre et recevoir simultanément des mots de 8bits. Il utilise les broches :
RC4/SDI : Serial Data Input (doit être configurée en entrée)
RC5/SDO : Serial Data Output (doit être configurée en sortie)
RC3/SCK : Serial Clock (doit être configurée en sortie)
RA4/SS : Slave Select : Utilisée optionnellement en mode slave soit comme entrée de synchronisation soit comme Chip Select en mode esclave multiples. Elle doit être configurée en E/S numérique à l'aide du registre ADCON1 et comme entrée à l'aide du registre TRISA

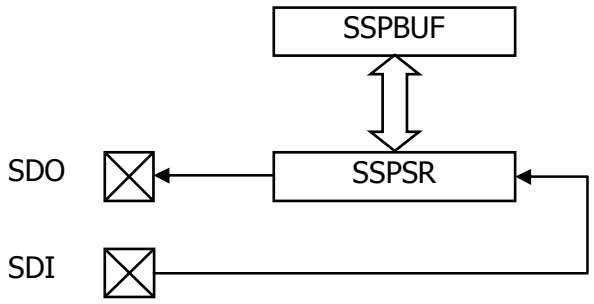
Le contrôle du port se fait par les bits 0 à 5 du registre SSPCON et les bits 6 et 7 du registre SSPSTAT.

Deux modes de fonctionnement sont possibles :

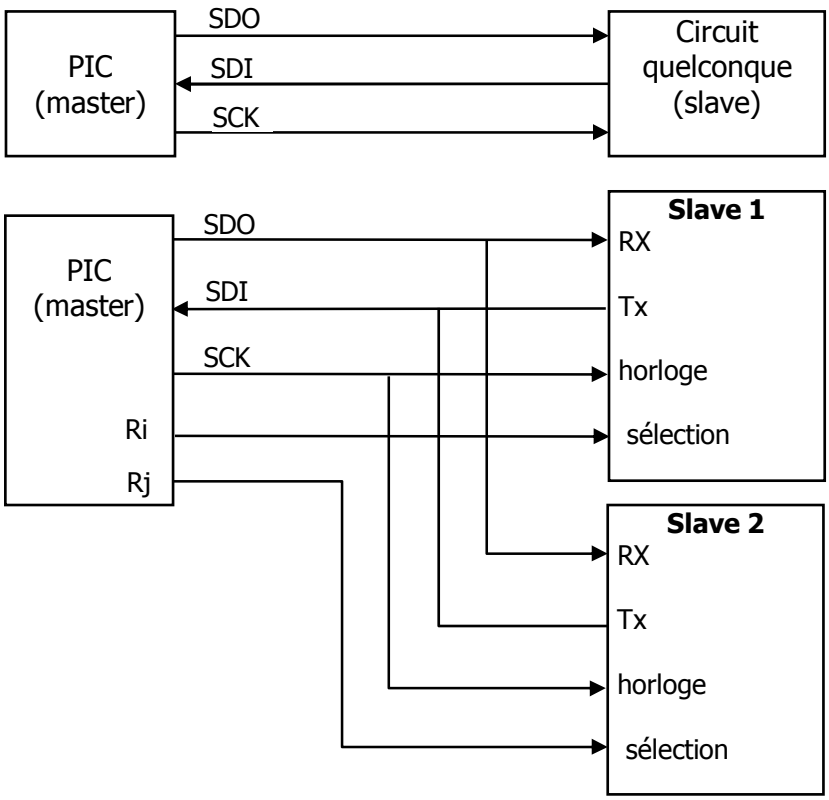
Mode **master** : C'est le PIC qui génère et gère l'horloge de synchronisation SCK. C'est donc lui qui décide du moment de démarrage d'une transmission réception.

Mode **slave** : L'horloge est générée par le circuit (master) avec lequel on communique. Notre Pic démarre une émission/réception au moment ou il commence à recevoir l'horloge venant du maître.

L'émission et la réception se font simultanément par le même registre à décalage SSPSR. Un cycle d'émission/réception commence au moment où on écrit un octet dans le registre SSPBUF, cet octet est copié par le PIC dans le registre à décalage SSPSR auquel il applique ensuite une rafale de 8 coups d'horloges. L'octet est alors transmis bit par bit sur la sortie SDO et il est remplacé par les bits qui arrivent sur l'entrée SDI. A la fin, le contenu de SSPSR (c.a.d l'octet reçu) est copié dans SSPBUF. Cette action est annoncée par la mise à 1 du bit BF et du drapeau d'interruption SSPIF. Le bit BF est un bit en lecture seule, la seule façon de le remettre à zéro est de lire le contenu du registre SSPBUF. Le drapeau SSPIF doit être remis à zéro à l'aide de l'instruction BCF par exemple.



X.3.1 Le mode SPI master



Le PIC fonctionne en master, il génère et gère l'horloge de synchronisation. C'est lui qui décide du moment où démarrer une émission/réception.
Le PIC peut communiquer avec plusieurs slaves, dans ce cas il faut désigner le slave avec lequel on désire communiquer, il faut que les slaves disposent d'une entrée de sélection CS qu'on peut contrôler avec une sortie quelconque d'un port d'E/S. Dans le cas ou les slaves sont des PIC, l'entrée de sélection est RA4/SS

Le cycle émission/réception d'un octet démarre au moment où on écrit dans le registre SSPBUF. A la fin du cycle, les bits SSPSTAT.BF et PIR1.SSPIF sont positionnés pour nous informer qu'un octet a été transmis et que l'octet reçu attend d'être lu dans le registre SSPBUF. Ces deux bits doivent être remis à zéro par nos soins, il faut juste remarquer que BF est un bit en lecture seule, pour le remettre à zéro il faut lire le registre SSPBUF

Le registre SSPCON en mode master :

		SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

SSPEN : Validation du module MSSP (1 = validé)

CKP : Clock Polarity : définit l'état de repos de l'horloge (idle state)
0 : Bas au repos, 1 : Haut au repos

SSPM3:SSPM0 : Ces bits définissent le mode de fonctionnement du module
0000 : SPI Master : débit = Fosc/4
0001 : SPI Master : débit = Fosc/16
0010 : SPI Master : débit = Fosc/64
0011 : SPI Master : débit = Ftmr2/2

Le registre SSPSTAT en mode master :

SMP	CKE						BF

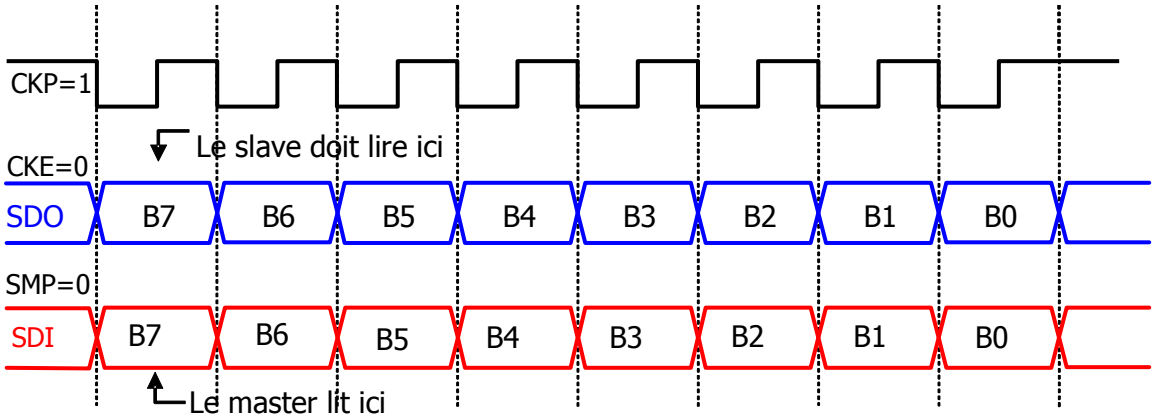
Seuls les bits utilisés par le mode SPI master sont représentés ici.

SMP : Sample bit : front d'horloge SCK sur lequel on lit le bit présent sur l'entrée SDI
0 : milieu de période d'horloge
1 : fin de la période d'horloge

CKE : Clock Edge Select : front de l'horloge SCK sur lequel on place un bit sur la sortie SDO. Ceci va dépendre du bit CKP. Si on désigne le niveau actif de l'horloge comme le niveau opposé du niveau de repos défini par CKP :
0 : Les bits sont transmis au moment où l'horloge passe au niveau actif
1 : Les bits sont transmis au moment où l'horloge passe au niveau de repos

BF : Indicateur qui indique quand il passe à 1 que le registre SSPBUF contient l'octet qui vient d'être reçu. Ce bit est remis automatiquement à zéro au moment de la lecture du registre SSPBUF.

Le dessin ci-dessous montre la configuration la plus fréquemment utilisée. Le Master écrit au début de la période d'horloge et lit aux milieux. Le Slave en fait de même.



X.3.1.1 Récapitulation mode master

En résumé, les étapes à suivre pour utiliser MSSP en mode SPI master sont :

Configurer la direction des lignes RC4/SDI, RC5/SDO et RC3/SCK
Définir la polarité de l'horloge par le bit SSPCON.CKP
Définir la fréquence de l'horloge à l'aide des bits SSPCON.SSPMx
Définir le moment de lecture des bits entrant à l'aide du bit SSPSTAT.SMP
Définir le moment d''ecriture des bits sortants à l'aide du bit SSPSTAT.CKE
Valider le port par le bit SSPCON.SSPEN

Une fois cette initialisation réalisée, l'émission/réception se fait comme suit :

- Copier un octet dans SSPBUF pour démarrer l'émission/réception
- attendre la fin en surveillant SSPSTAT.BF ou PIR1.SSPIF
- La lecture de SSPBUF est la seule façon de r.à.z le bit SSPSTAT.BF
- Si on l'utilise, le drapeau PIR1.SSPIF doit être remis à 0

X.3.2 Le mode SPI slave

En mode slave, le module MSSP fonctionne d'une façon similaire au mode master à part qu'ici, ce n'est pas notre PIC qui génère l'horloge et décide des moments de communication. Il se contente de réaliser une émission/réception chaque fois qu'il reçoit une rafale d'horloge sur la broche SCK qui est devenue une entrée.

Le registre SSPSTAT en mode slave :

SMP	CKE						BF

SMP : doit être mis à 0 : la lecture sur SDI doit toujours se faire au milieu du cycle horloge.

CKE : Clock Edge Select : ce bit fonctionne comme en mode master
0 : Les bits sont transmis au moment où l'horloge passe au niveau actif
1 : Les bits sont transmis au moment où l'horloge passe au niveau de repos. Dans ce cas le mode *Sélection de slave* soit être obligatoirement utilisé

BF : Indicateur qui indique quand il passe à 1 que le registre SSPBUF contient l'octet qui vient d'être reçu. Ce bit est remis automatiquement à zéro au moment de la lecture du registre SSPBUF.

Le registre SSPCON en mode slave :

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

WCOL : Write Collision Detect bit : Cet indicateur d'erreur est positionné quand on écrit une valeur dans SSBUF alors que la valeur précédente n'est pas encore transmise. Il doit être remis à zéro par programme

SSPOV : Receive Overflow Indicator bit : Cet indicateur d'erreur est positionné quand une nouvelle donnée est reçue alors que le registre SSBUF n'est pas encore lu. Dans ce cas la nouvelle donnée est perdue. Ce bit doit être remis à zéro par programme

SSPEN : Validation du module SSP (1 = validé)

SKP : Clock Polarity : définit l'état de repos de l'horloge (idle state)
0 : Bas au repos, 1 : Haut au repos (doit coïncider avec le choix du master)

SSPM3:SSPM0 : Ces bits définissent le mode de fonctionnement du module
0100 : SPI Slave : mode Slave Select, l'entrée SS est validée, il faut la placer à 0 quand on veut communiquer avec ce slave (chip select)
0100 : SPI Slave : l'entrée SS est inhibée (fonctionne en E/S normale)

Dans le cas ou l'entrée SS est utilisée, elle doit être configurée en entrée par TRISA.5 et elle doit être configurée comme entrée numérique à l'aide du registre ADCON1 car elle est multiplexée avec une entrée analogique du module CAN.

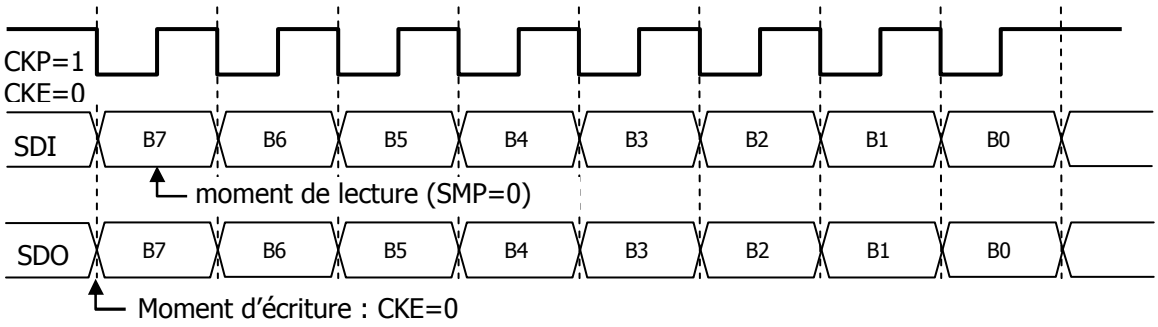
X.3.2.1 Les chronogrammes de fonctionnement :

Un cycle d'émission/réception démarre au moment du démarrage de l'horloge externe.

Si on ne désire réaliser que des réceptions, on n'a pas besoin d'écrire à chaque fois dans SSPBUF du moment que ce n'est pas nous qui démarrons le cycle Tx/Rx

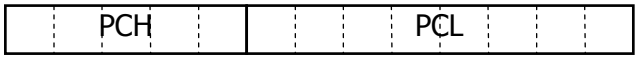
Si on ne désire réaliser que des transmissions, il va de soi que nous devons écrire les données à transmettre dans SSPBUF, mais il va falloir aussi lire ce registre à la fin de chaque cycle Tx/Rx pour baisser l'indicateur BF, sinon une erreur d' Overflow sera déclenchée (SSPOV)

Le chronogramme ci-dessous illustre la configuration la plus courante, CKP=1, CKE=0, CMP=0(obligatoire). Les bits sont transmis au début de la période d'horloge et lu au milieu de celle ci. Il n'est pas nécessaire d'utiliser l'entrée SS dans ce cas.



XI Annexe : Gestion du Programme Counter,

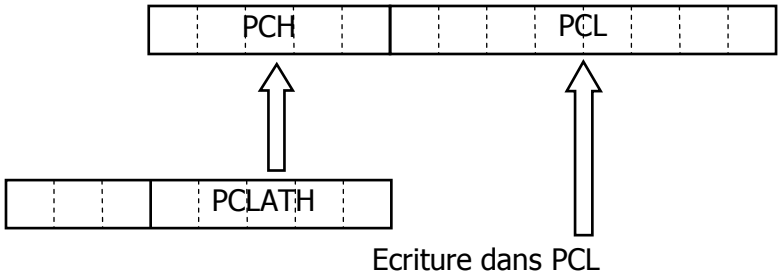
Le PC est un registre de 13 bits, il peut donc adresser 8 k de mémoire programme.



PCL (8 bits) est la partie basse de PC, il est accessible en lecture écriture
PCH (5 bits) est la partie haute de PC, il n'est pas accessible directement. On peut toutefois le modifier indirectement à l'aide du registre PCLATH qui est une registre SFR où seuls 5 bits sont utilisé.

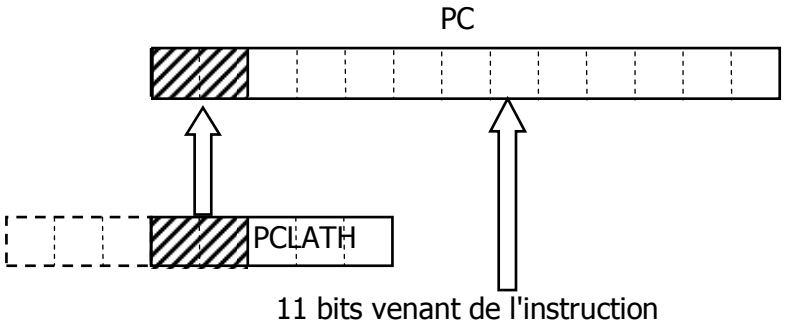
XI.1 GOTO calculé : modification de la valeur de PCL

Quand on écrit quelque chose dans PCL, alors le contenu de PCLATH est automatiquement copié dans PCH. Donc, avant d'écrire dans PCL s'assurer que la valeur contenue dans PCLATH est correcte.



XI.2 Instruction de branchement

Un branchement consiste à modifier la valeur de PC. Mais dans les instructions de branchement, seulement 11 bits sont utilisés pour coder l'adresse de destination, ce qui donne au maximum un intervalle de 2 k alors que nous avons une mémoire programme de 8k. Pour remédier à ce problème, les deux bits manquants pour compléter les 13 bits de PC sont pris dans PCLATH :



La mémoire programme apparaît donc comme organisée en 4 page de 2 k chacune. Lors d'un saut, l'adresse instruction précise la destination à l'intérieur d'une page et les deux bits 3 et 4 de PCLATH désignent la page

Attention, en assembleur on ne précise pas les adresses mais on utilise des étiquettes qui correspondent à des adresses complètes (13 bits). Cela ne change rient car seuls 11 bits de l'étiquette seront envoyés dans le PC et c'est à nous de gérer les deux bits de PCLATH pour compléter l'adresse de destination.

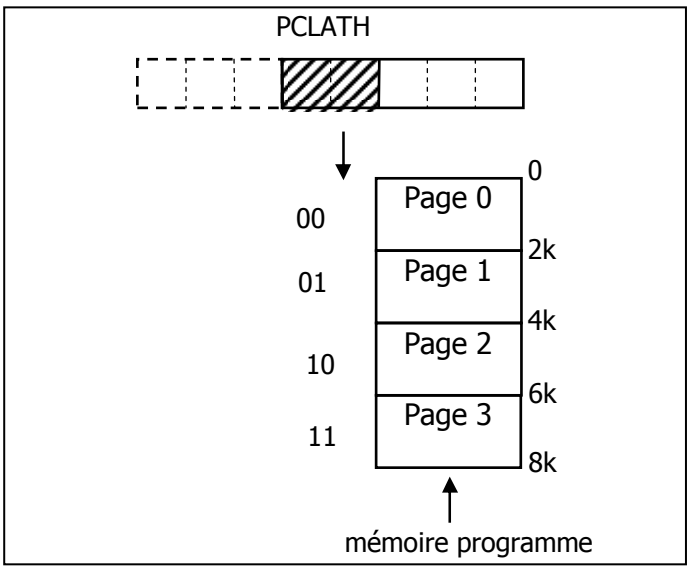


Fig. XI.1 : gestion de la mémoire programme lors d'une instruction de saut

XII Références

1) http://www.multimania.com/pmorphe/bienvenuechez_atm.htm
2) La programmation des PICs, Première partie – PIC16F84, Rev 5, Bigonff
3) La programmation des PICs, Secode partie-PIC16F876/877, Rev 7, Bigonff
4) PIC 16F87X Data Sheet (DS30292C) , Microchip
5) PICmicro™ Mid-Range MCU Family Reference Manual (DS33023A), Microchip
6) La page web de Jacques Weiss sur les PIC
7) La page web d'Offset sur les PIC