Ministère de l'enseignement supérieur et de la recherche scientifique
Institut Supérieur des Etudes Technologiques de Sousse

Département Génie Electrique
Spécialité : Automatisme Informatique Industrielle – 2ème Année

Support de cours

Microcontrôleur

PIC16F877

Ali HMIDENE

Agrégé en Génie Electrique

Technologue à l'ISET de Sousse

20010/2011

TABLE DES MATIERES

I - INTRODUCTION AU MICROPROCESSEUR 4

1. Structure de base d'un calculateur 4

2. Architecture d'un CPU 5
2.1. Unité arithmétique et logique 5
2.2. Les registres 6
2.3. L'unité de contrôle (UC) 6

3. Fonctionnement d'un microprocesseur 6
3.1. Jeu instructions 6
3.2. Cycle d'exécution d'une instruction 7
3.3. Les interruptions 9

4. Système à microprocesseur 9
4.1. Microprocesseur et Microcontrôleur 9
4.2. Architecture des microprocesseurs 9

II - LES MICROCONTROLEURS PIC16F876/877 11

1. Présentation 11

2. Caractéristiques du PIC16F877 11
2.1. Caractéristique de la CPU 11
2.2. Caractéristiques des périphériques 12
2.3. Brochage 12

3. Architecture interne du microcontrôleur PIC16F877 12

4. Traitement des Instructions 13

5. Organisation de la mémoire 14
5.1. Mémoire programme 14
5.2. Mémoire données 14

5.3. Registres Spéciaux (SFR) 16

6. Les Instructions 18
6.1. Les modes d'adressage 19

7. Les Interruptions 22

III - LES PORTS D'ENTREE/SORTIE 25

1. PORTA 25

2. PORTB 26

3. PORTC 26

4. PORTD et PORTE 26

IV - LES TIMERS 29

1. Introduction 29

2. Timer 0 29

3. Timer 1 31

4. Timer 2 32

V - CONVERTISSEUR ANALOGIQUE NUMERIQUE 36

1. Présentation 36
1.1. Registre ADCON1 37
1.2. Registre ADCON0 38

2. Conversion A/N 39
2.1. Temps d'acquisition 39
2.2. Temps de conversion 40

3. Etapes de programmation 41

VI - MODULE CAPTURE COMPARE PWM (CCP) 42

1. Présentation 42

2. Module CCP en mode capture 43

3. Module CCP en mode Compare 43

4. Module CCP en mode PWM 44

VII - L'INTERFACE SERIE 46

1. Présentation 46

2. Port série du microcontrôleur PIC 47
2.1. Registre SPBRG 47
2.2. Registre TXSTA 48
2.3. Registre RCSTA 48
2.4. Module d'émission 49
2.5. Module de réception 50

I - INTRODUCTION AU MICROPROCESSEUR

1. Structure de base d'un calculateur

D'un point de vue matériel, un système informatique minimal est constitué d'un processeur, des mémoires et d'un ensemble des entrées sorties.

- Le microprocesseur (MPU, *Micro-Processing Unit)* est un circuit intégré à très grande échelle d'intégration (VLSI) capable d'exécuter automatiquement des instructions (opérations élémentaires) qu'il ira chercher dans la mémoire du micro-ordinateur. Il remplit donc les fonctions d'une unité centrale de traitement (CPU, *Central Processing Unit)* en un seul boîtier.

Toutes les informations qu'utilise le microprocesseur sont stockées dans des mémoires, en particulier le programme, le fonctionnement du microprocesseur est entièrement conditionné par le contenu de celles-ci. Ces mémoires contiennent deux types d'informations : le programme (ensemble des ordres à exécuter) et les données nécessaires pour la réalisation d'une tâche précise.

- Les données ou OPERANDES proviennent, le plus souvent, d'un calcul effectué par le microprocesseur ou d'un périphérique d'entrée, (clavier, disque, ...) via une interface. Elles sont stockées dans des mémoires qui peuvent être lues et écrites appelées RAM (Random Acces Memory).

- Les programmes figés sont placés, de façon définitive, dans des mémoires qui ne pourront qu'être lues par le microprocesseur. Ce sont des mémoires non volatiles, de type ROM (Read Only Memory), PROM, EPROM, Flash ...

- Dans un système à microprocesseur, l'interface d'entrée-sortie permet d'assurer la liaison entre l'unité centrale, via le bus interne du micro-ordinateur et l'environnement extérieur (périphériques).

Le microcontrôleur échange les informations avec les composants qui lui sont associés (mémoire et périphériques I/O) au moyen d'un ensemble de connexions appelé *bus*. Un bus est un ensemble de fils qui assure la transmission du même type d'information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement à microprocesseur :

- le bus de données : bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement. Le nombre de lignes du bus de données définie la capacité de traitement du microprocesseur ; selon le microprocesseur la largeur du bus peut être de 8 bits, 16 bits, 32 bits, 64 bits ...

- le bus d'adresses: unidirectionnel qui permet la sélection de la case contenant l'information à traiter dans un *espace mémoire* (ou *espace adressable*). L'espace adressable peut avoir 2n emplacements, avec n est le nombre de lignes du bus d'adresses.

- le bus de commande: constitué par quelques conducteurs qui assurent la synchronisation du flux d'informations sur les bus de données et d'adresses.

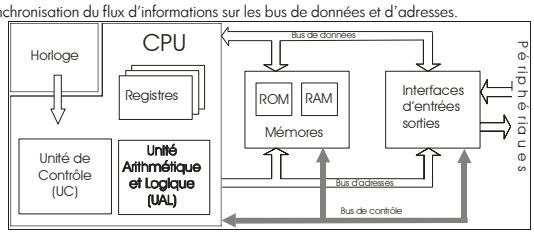


Figure I-1 : Schéma bloc d'un système à microprocesseur

2. Architecture d'un CPU

Le microprocesseur est constitué des unités fonctionnelles suivantes :

- d'unité Arithmétique et Logique (ALU),
- des registres,
- une unité de commande (CU).

2.1. Unité arithmétique et logique

Cet organe, interne au microprocesseur, permet la réalisation d'opérations arithmétiques (Addition, Soustraction) et logiques (AND, OR, XOR...). Outre les opérations arithmétiques et logiques, l'ALU réalise aussi les opérations de décalage et de rotation. Le registre d'état est lié étroitement à L'ALU, nous donne à travers ces inducteurs (Flags) des renseignements supplémentaires sur le résultat d'une opération (résultat nul, négatif, dépassement...). On peut citer par exemple les indicateurs de :

- retenue (carry : C)
- retenue intermédiaire (Auxiliary-Carry : AC)
- signe (Sign : S)
- débordement (overflow : OV ou V)
- zéro (Z)
- parité (Parity : P)

2.2. Les registres

Il existe deux types de registres : les registres d'usage général, et les registres d'adresses (pointeurs)

Les registres d'usage général (Registre de travail)

Ce sont des mémoires rapides, à l'intérieur du microprocesseur, qui permettent à l'UAL de manipuler des données à vitesse élevée. Ils sont connectés au bus de données interne du microprocesseur.

L'adresse d'un registre est associée à son nom (on donne généralement comme nom une lettre A, B, C...). L'accumulateur est un registre de travail qui sert à stocker un opérande au début d'une opération et/ou le résultat à la fin de l'opération.

Les registres d'adresses (pointeurs)

Ce sont des registres connectés sur le bus adresses.

On peut citer comme registre:

- Le compteur ordinal (pointeur de programme PC) : celui-ci contient toujours l'adresse de la prochaine instruction à exécuter.

- Le pointeur de pile (Stack Pointer SP), utilisé pour la sauvegarde de l'adresse de retour d'un sous-programme.

- Les registres pointeur de données ou d'index : utilisés pour l'adressage indirect de la mémoire.

2.3. L'unité de contrôle (UC)

Elle permet de séquencer le déroulement des instructions. Elle effectue la recherche en mémoire des instructions, décodage et exécuté l'instruction recherchée. Elle est composée essentiellement :

- d'un *registre d'instruction* (RI), recevant le code de l'instruction à exécuter.

- d'un *décodeur d'instruction*, permettant de déterminer le type de l'instruction à exécuter.

- d'un *Bloc logique de commande (ou séquenceur)* : Il organise l'exécution des instructions au rythme d'une horloge, et élabore tous les signaux de synchronisation internes ou externes du microprocesseur.

3. Fonctionnement d'un microprocesseur

3.1. Jeu instructions

Les microprocesseurs sont capables d'effectuer un certain nombre d'opérations élémentaires. Cet ensemble d'opérations élémentaires est appelé jeu d'instructions.

Une instruction au niveau machine doit fournir à l'unité centrale toutes les informations nécessaires pour déclencher une telle opération élémentaire. Elle comporte en général plusieurs champs ; le premier champ contient le code de l'opération (*Code-Op* ou *Op-Code* en anglais); les autres champs peuvent comporter des données ou l'identification des opérandes. Sur certaines machines les instructions ont toutes la même longueur, sur d'autres cette longueur peut varier avec le code opération ou le mode d'adressage.

Format d'une instruction :

Code de l'opération	opérande1	opérande2

Exemple : Additionner le contenu du registre A avec le contenu de la case mémoire dont l'adresse est mentionnée dans le champ 3

ADDITIONNER	Registre A	⦿ Case mémoire

3.2. Cycle d'exécution d'une instruction

Le traitement d'une instruction peut être décomposé en plusieurs phases. Celles-ci sont au nombre de trois :

- ✓ Recherche de l'instruction (Fetch)
- ✓ Décodage (decode)
- ✓ Exécution (execute)

3.2.1. Recherche d'une instruction (Fetch)

Le contenu de PC (compteur ordinal) est placé sur le bus des adresses (c'est l'unité de contrôle qui établit la connexion). L'unité de contrôle (UC) émet un ordre de lecture (READ=RD=1), au bout d'un certain temps (temps d'accès à la mémoire) le contenu de la case mémoire sélectionné est disponible sur le bus des données.L'unité de contrôle charge la donnée dans le registre d'instruction pour décodage.

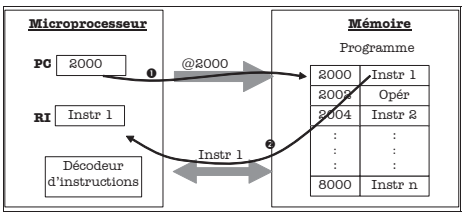


Figure I-2 : Recherche de l'instruction

3.2.2. Décodage

Le registre d'instruction (RI) contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction. L'unité de commande décode le code opératoire. Si l'instruction est suivie d'un opérande (une constante ou adresse d'une donnée), l'unité de contrôle récupère cet opérande, et positionne le compteur ordinal (PC) sur l'instruction suivante.

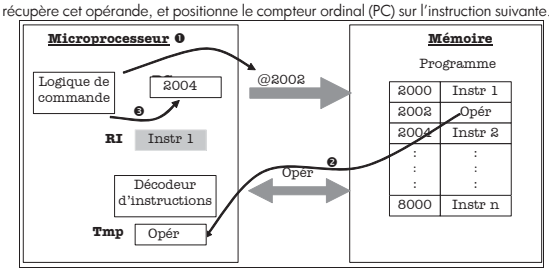


Figure I-3 : Récupération de l'opérande

3.2.3. Exécution

Le microprogramme réalisant l'instruction est exécuté. Les indicateurs sont positionnés (registre d'état).

L'exemple suivante montre l'exécution d'une addition d'un Opérande se trouvant dans un registre temporaire (Tmp) avec le contenu du registre A, le résultat est chargé dans le registre A.

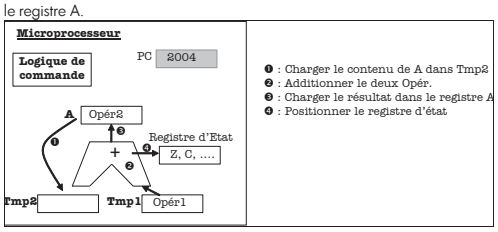


Figure I-4 : Exécution de l'instruction

3.3. Les interruptions

Le microprocesseur exécute les instructions de manière séquentielle. Souvent, il est nécessaire d'interrompre un programme pour exécuter une tâche supposée prioritaire. Cette tâche (appelée routine d'interruption) est déclenchée suite à un évènement extérieur.

Une fois que l'interruption à été servie (exécutée) le programme principal reprend son exécution à partir de l'endroit où il a été interrompu.

Le microprocesseur possède des broches spécialisées pour les interruptions. Si un composant extérieur au microprocesseur veut l'interrompre, il lui suffit de changer l'état d'une de ces broches (0 ⇨1 ou 1⇨0 suivant l'état les broches). Le sous-programme d'interruption est placé en mémoire par le concepteur à une adresse connue par microprocesseur.

4. Système à microprocesseur

4.1. Microprocesseur et Microcontrôleur

Au début de la commercialisation des microprocesseurs, un système minimum était obligatoirement constitué de plusieurs circuits intégrés.

Les microcontrôleurs sont apparus ensuite, lorsque l'intégration des composants a fait des progrès considérables. Actuellement, dans un circuit intégré, un microcontrôleur regroupe l'équivalent d'un système à microprocesseur dans un même boîtier. Un système à microprocesseur peut être constitué de nombreux composants (mémoires, etc.) ; il est facilement extensible. Les constituants d'un microcontrôleur sont figés.

Les microcontrôleurs sont plutôt dédiés aux applications qui ne nécessitent pas une grande quantité de calculs complexes, mais qui demandent beaucoup de manipulations d'entrées/sorties. C'est le cas de contrôle de processus.

Les systèmes à microprocesseur sont plutôt réservés pour les applications demandant beaucoup de traitement de l'information et assez peu de gestion d'entrées / sorties. Les ordinateurs sont réalisés avec des systèmes à microprocesseur.

4.2. Architecture des microprocesseurs

Pour l'organisation des différentes unités, il existe deux architectures :

- l'architecture Von Neuman (du nom d'un des savants qui a contribué à la mise au point d'un des tout premiers ordinateurs). La mémoire programme, la mémoire données et les périphériques d'entrées/sorties partagent le même bus s'adresses et de données.

- l'architecture Harvard, sépare systématiquement la mémoire de programme de la mémoire des données : l'adressage de ces mémoires est indépendant. Ce type d'architecture est utilisé sur des microcontrôleurs qui ont connu un développement important ces dernières années.

L'architecture généralement utilisée par les microprocesseurs est la structure Von Neuman (exemples : la famille Motorola 68XXX, la famille Intel 80x86). L'architecture Harvard est plutôt utilisée dans des microprocesseurs spécialisés pour des applications temps réels, tel que les DSP, les microcontrôleurs PIC, les AVR...

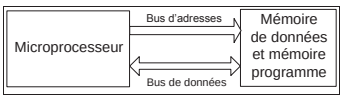


Figure I-5 : Architecture Von Neuman

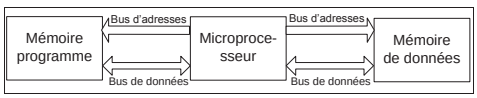


Figure I-6 : Architecture Harvard

II - LES MICROCONTROLEURS PIC16F876/877

1. Présentation

Les microcontrôleurs PICs sont des composants dits RISC (Reduce Instructions Set Computing), ou encore composants à jeu d'instructions réduit. Tous les PICs Mid-Range ont un jeu de 35 instructions, stockent chaque instruction dans un seul mot de programme, et exécutent chaque instruction (sauf les sauts) en un cycle machine. On atteint donc des très grandes vitesses.

L'horloge fournie au PIC est divisée par 4. C'est cette base de temps qui donne le temps d'un cycle. Si on utilise par exemple un quartz de 4MHz, on obtient donc 1000000 de cycles/seconde ; or, comme le PIC exécute pratiquement une instruction par cycle, hormis les sauts, cela nous donne une puissance de l'ordre de 1MIPS (1 Million d'Instructions Par Seconde).

- Les microcontrôleurs PICs sont subdivisés en 3 grandes familles :
- La famille Base-Line, utilise des mots d'instructions de 12 bits,
- la famille Mid-Range, utilise des mots de 14 bits « PIC16 »,
- la famille High-End, utilise des mots de 16 bits. « PIC17 et PIC18 »

Nous allons dans ce document étudier le microcontrôleur PIC16F877, qui n'est rien d'autre qu'un PIC16F84 amélioré.

2. Caractéristiques du PIC16F877

2.1. Caractéristique de la CPU

⇨ CPU à architecture RISC (8 bits)
⇨ Mémoire programme de 8 Kmots de 14 bits (Flash),
⇨ Mémoire donnée de 368 Octets,
⇨ EEPROM donnée de 256 Octets,
⇨ 14 sources d'interruptions,
⇨ 5 ports d'entrées/sorties bidirectionnels,
⇨ Pile de 8 niveaux,
⇨ Oscillateurs de type RC ou Quartz (Fmax = 20MHz),
⇨ Programmation en mode série de la mémoire Flash à travers les broches RB6/PGC et RB7/PGD,
⇨ Chien de garde (WatchDog),

2.2. Caractéristiques des périphériques

- ⇨ Timer0 : Timer/Compteur 8 bits avec un prédiviseur 8 bits
- ⇨ Timer1 : Timer/Compteur 16 bits avec un prédivision de 1, 2, 4, ou 8 ; il peut être incrémenté en mode veille (Sleep), via une horloge externe,
- ⇨ Timer2 : Timer 8 bits avec deux diviseurs (pré et post diviseur)
- ⇨ Deux modules « Capture, Compare et PWM »
- ⇨ Module capture 16 bits avec une résolution max. 12,5 ns,
- ⇨ Module Compare 16 bits avec une résolution max. 200 ns,
- ⇨ Module PWM avec une résolution max. 10 bits,
- ⇨ ADC multi-canal (8 voies) avec une conversion sur 10 bits,
- ⇨ Synchronous Serial Port (SSP) travaillant en mode SPI et en mode I2C (mode maitre/escalve),
- ⇨ Universel Synchronous Asynchronous Receiver Transmitter (USART/SCI) avec un 9ème bit pour la détection d'adresse.
- ⇨ Parallel Slave Port de 8 bits avec signaux de contrôle externes RD\, RW\ et CS\.

2.3. Brochage

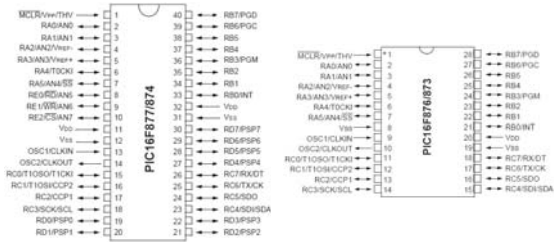


Figure II-1 : Brochages des microcontrôleurs PIC

3. Architecture interne du microcontrôleur PIC16F877

Les microcontrôleurs la famille PIC16 se base sur une architecture RISC, cette architecture permet une accessibilité séparée de la mémoire programme et celle des données avec un nombre d'instructions très réduit (35 insts.). Le compteur de

programme des microcontrôleurs Mid-range est de 13 bits ; ils peuvent donc adresser 8Kmots au maximum.

La pile du PIC est constituée d'une mémoire circulaire (LIFO) de 8 niveaux, par conséquent, il n'est pas possible de faire plus de 8 empilements successifs ; de plus elle n'est pas accessible à l'utilisateur.

La Figure II-2, montre la structure interne des microcontrôleurs PIC16.

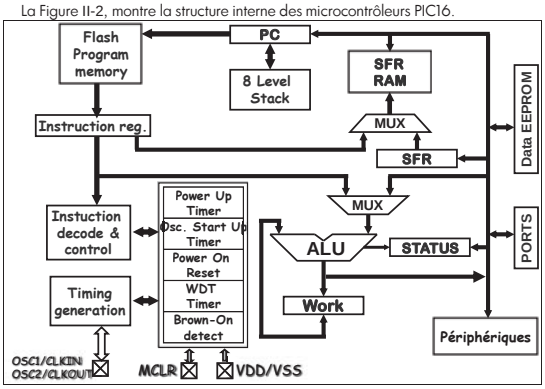


Figure II-2 : Architecture interne

4. Traitement des instructions

L'UAL reçoit les arguments (opérandes) de l'unique registre de travail (Work) et la mémoire de données (File). Le résultat de traitement pourra être stocké, selon la valeur du bit d, dans registre W (d = 0) ou dans le mémoire File (d = 1). Il est a noté que le mémoire File comprend les registres du CPU (STATUS, FSR, PCLATH, PCL), les registres des périphériques I/O et la mémoire de données utilisateur.

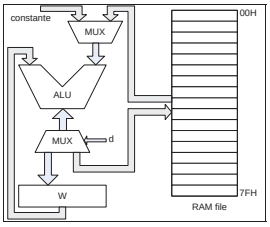


Figure II-3

Avec cette structure le microcontrôleur PIC utilise trois modes d'adressage :

- *Adressage immédiat (Litéral)* : ce mode ne fait pas intervenir le mémoire de données (File). Exemple : ADDLW Const ; W ← W + Const.
- *Adressage direct* : Ce mode fait intervenir l'accumulateur W et la mémoire File. Le résultat pourra être chargé selon la valeur du bit d, dans W ou File.

Exemple : ADDWF F,d

Si d = 0, alors W ← W + (F)

Si d = 1, alors (F) ← W + (F)

- Adressage indirect utilisant le registre FSR pour pointer la donnée.

5. Organisation de la mémoire

5.1. Mémoire programme

La mémoire programme est constituée de 8 Kmots de 14 bits. Le vecteur Reset occupe l'adresse 0000H, alors que le vecteur d'interruption occupe l'adresse 0004H. La mémoire programme est divisée en 4 pages (2 Kmots par page).

Microchip a réservé une zone de pile de 8 niveaux pour l'empilement des adresses de retour des sous programmes.

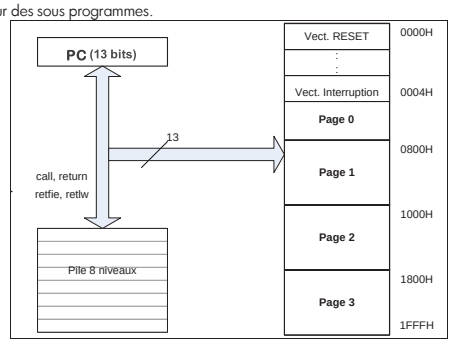


Figure II-4 : Adressage de la mémoire programme

5.2. Mémoire données

La mémoire de données (File Registers) compte au total 512 octets, reparties sur 4 banques (0, 1, 2 et 3); dans chacune des banques vous trouvez des cases mémoires spéciales appelées «SPECIAL FUNCTION REGISTRES (SFR) », et des cases mémoires à usage général appelées GENERAL PURPOSE REGISTER (GPR).

La zone des registres SFR comporte les registres du CPU et celles des périphériques.

La partie restante GPR, constitue la mémoire de stockage des données utilisateur. Dans le cas des PIC16F876/877, elle compte au total 368 octets, reparties de la manière suivante :

96 octets (80 octets dans la banque 0 et 16 Octets zone commune)
80 octets dans la banque 1,
96 octets dans la banque 2,
96 octets dans la banque 3,
L´accès à la zone commune n'impose pas la sélection de la banque.

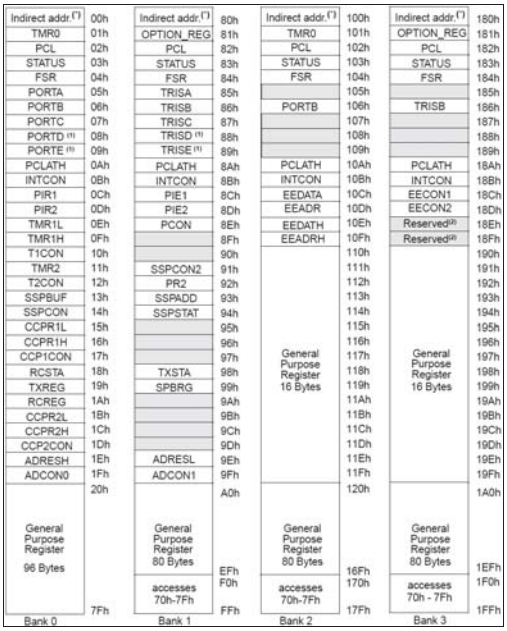


Figure II-5 : Mémoire de données

Deux modes d'adressage sont utilisés pour l'accès à la mémoire File :

- adressage direct : les 7 bits (b_6 .. b_0) de la partie basse de l'adresse font partie du code de l'instruction ; la partie haute de l'adresse est constituée des bits RP1 et RP0 du registre STATUS.

Dans ce mode d'adressage, tout accès à la mémoire est précédé par la sélection d'une banque selon le tableau suivant :

RP1 : RP0	Banque	Zone d'adressage
0 0	Banque 0	00H .. 7FH
0 1	Banque 1	80H .. FFH
1 0	Banque 2	100H .. 17FH
1 1	Banque 3	180H .. 1FFH

- adressage indirect : la partie basse de l'adresse est fournie par le registre FSR (File Select Register). La sélection d'une banque est réalisée dans ce cas par le 8 bits du registre FSR et le bit IRP du registre STATUS.

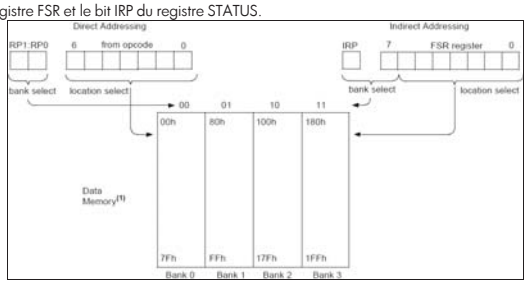


Figure II-6 : Adressage de la RAM file

5.3. Registres Spéciaux (SFR)

Les SFR « Special Function Registers » sont implémentés en RAM statique (File) et occupent les parties basses des banques. Ces registres sont classés en deux catégories ; certains liés au CPU et les autres pour le contrôle des périphériques. Nous allons décrire dans cette partie les registres attachés au CPU.

5.3.1. Le registre STATUS

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO\	PD\	DC	Z	C
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : IRP : sélection des banques en adressage indirect.
IRP = 0 : banque 0, 1
IRP = 1 : banque 2, 3
Bits 6 et 5 : RP1 et RP0 : sélection des banques en adressage direct
Bit 4 : TO\ : Time Out bit (bit en lecture seulement).
TO\ = 1 : Après une mise sous tension, après l'instruction CLRWDT ou bien après l'instruction SLEEP.
TO\ = 0 : Signifie qu'un Time Out du timer de watchdog est survenu.
Bit 3 : PD\ = Power Down bit (bit en lecture seulement).
PD\ = 1 : Après une mise sous tension ou bien après une RAZ du Watchdog.
PD\ = 0 : Après l'instruction SLEEP.
Bit 2 : Z = Zero bit.
Z = 1 : Le résultat d'une opération arithmétique ou logique est nul.
Z = 0 : Le résultat d'une opération arithmétique ou logique est différent de zéro. Ce bit est positionné aussi par l'instruction « movf »
Bit 1 : DC = Digit Carry/$\overline{borrow}$ bit (affecté par ADDWF, ADDLW, SUBWF et SUBLW). Ce bit est inversé en cas d'un emprunt.
DC = 1 : Une retenue sur le 4ème bit des poids faible est survenue.
DC = 0 : Pas de retenue sur le 4ème bit des poids faible.
Bit 0 : C = Carry/$\overline{borrow}$ bit (affecté par ADDWF, ADDLW, SUBWF et SUBLW). Ce bit est inversé en cas d'un emprunt.
C = 1 : Une retenue sur le bit MSB est survenue.
C = 0 : Pas de retenue sur le bit MSB.

5.3.2. Le registre PCON

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-1
						POR\	BOR\
b7	b6	b5	b4	b3	b2	b1	b0

Bit 1 : POR\ : Power-on Reset Status bit.
POR\ = 1 : Pas de Power-on Reset.
POR\ = 0 : Power-on Reset est survenue (ce bit doit être mis à 1 après un POR).
Bit 0 : BOR\ : Brown-out Reset Status bit.
BOR\ = 1 : Pas de Brown-out Reset Une.
BOR\ = 0 : Brown-out Reset est survenue (ce bit doit être mis à 1 après un BOR).
Associés au bit TO\ du registre STATUS, ces permettent de déterminer la source de RESET.

5.3.3. Le Compteur de Programme (PC : PCLATH et PCL)

Le compteur de programme PC a une largeur de 13 bits (8Kmots). La partie basse de l'adresse se trouve dans le registre PCL, alors que la partie haute (b_{12} .. b_8) se trouve dans le registre PCH. Ce dernier n'est pas accessible, mais il peut être modifié indirectement à travers le registre PCLATH.

Deux cas possibles permettent de modifier le contenu du registre PC :

- Instruction de saut calculé en cours d'exécution du programme ; cette instruction permet de parcourir un tableau constant en ajoutant à PCL le contenu du registre W. Dans ce cas le contenu de PCLATH est chargé automatiquement dans PCH.

- Instructions de branchement (call et goto). Le 11 bits d'adresse sont fournit par l'instruction de branchement. Le deux bits restants (b_{12} et b_{11}) seront chargé depuis le registre PCLATH.

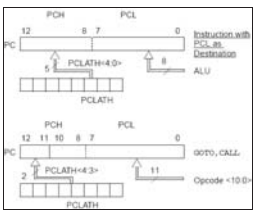


Figure II-7 : Registre PCLATH

6. Les Instructions

La famille Mid-Range, possède 35 instructions codées sur des mots de 14 bits. Ces instructions comportent le code opératoire « OPCODE » pour spécifier le type de l'instruction et un ou deux opérandes. Microchip a regroupée ses instructions en trois catégories (voir Figure II-8) :

- les instructions orientée octet (byte),
- les instructions orientée bit,
- les instructions à adressage immédiat et de contrôle,

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	TO,PD	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	TO,PD	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Figure II-8 : Table des instructions

6.1. Les modes d'adressage

Les instructions utilisent différentes méthodes pour accéder aux informations qu'elles manipulent. Ces méthodes s'appellent les « modes d'adressage ».

6.1.1. Adressage inhérent ou implicite

Les instructions agissent sur les registres internes du CPU. Les instructions comportent donc un code opératoire (Mnémmonique) sans opérande.

Syntaxe : MNEMONIQUE

```
Exemple :       clrw   : remettre le registre W à 0
                clrwdt : remettre le compteur WatchDog à 0
                sleep  : entrer en mode veille
```

6.1.2. Adressage immédiat ou Littéral

Dans ce mode d'adressage, l'instruction comporte le code opératoire suivi de la donnée à manipuler. L'adressage immédiat ne s'applique qu'au registre W ;

Syntaxe : MNEMONIQUE Constante

Format de l'instruction

Instruction

Code-Op	Constante k (8bits)

b_{13} b_7 b_6 b_0

```
Exemples :    movlw  20       ; w ← 20
              addlw  0x50     ; w ← w + 50
              sublw   10      ; w ← 10 - w
```

6.1.3. Adressage direct

L'instruction comporte le code opératoire suivi de deux opérandes : l'adresse de la donnée à manipuler et le bit de destination.

Syntaxe : MNEMONIQUE f, d

f : désigne l'adresse de la case mémoire « file register » codée sur 7 bits (soit 128 octets). L'adresse effective (réelle) est complétée par les bits RP1 et RP0 du registre STATUS.

d : désigne le bit de destination

si d = 0 : le résultat est chargé dans W

si d = 1 : le résultat est chargé dans la mémoire file.

Format de l'instruction

Instruction

Code-op	d	f

b_{13} b_8 b_7 b_6 b_0

Exemple :

```
VAR1 EQU  0x20 ; VAR1 est située dans la banque 0 à l'adresse 0x20
VAR2 EQU  0x20 ; VAR2 est située dans la banque 2 à l'adresse 0x20
;-----------------------------------------------------------------
; Pour accéder à la variable VAR1 il faut sélectionner la banque 0
     bcf  STATUS, RP1     ;     mettre RP1 à 0
     bcf  STATUS, RP0     ;     mettre RP0 à 0
     addwf    VAR1, 0     ;     W← W+ (VAR1)
; Pour accéder à la variable VAR2 il faut sélectionner la banque 2
     bsf  STATUS, RP1     ;     mettre RP1 à 1
     bcf  STATUS, RP0     ;     mettre RP0 à 0
     addwf    VAR2, 1     ;     (VAR2)← W+ (VAR1)
;-----------------------------------------------------------------
```

6.1.4. Adressage indirect

Ce mode d'adressage utilise le registre virtuel INDF. Les instructions qui utilisent le registre INDF (INDirect File), accèdent réellement à la case mémoire pointée par le

registre FSR (File Select Register). Les 9 bits d'adresse sont obtenus en concaténant les 8 bits du registre FSR avec le bit IRP du registre STATUS.

Syntaxe : MNEMONIQUE INDF, d

L'exemple suivant initialise à zéro les cases mémoires de l'adresse 20H à 3FH.

```
     movlw     0x20      ; Initialise le pointeur
     bcf       STATUS,7  ;  9ème bit de l'adresse à 0
     movwf     FSR       ;
Next clrf      INDF      ; remet à 0 la case mémoire pointée
                         ; par FSR
     incf      FSR,F     ; incrémente le pointeur
     btfss     FSR,6     ; fin?
     goto      Next      ; remettre à zéro la case suivante
                         ; continue
```

6.1.5. Instructions de manipulation des bits

Ces instructions font partie de l'adressage direct. L'adresse comporte le code opératoire suivi de l'adresse de la case mémoire contenant le bit à manipuler ainsi que le numéro du bit.

Syntaxe : MNEMONIQUE f, b

f : l'adresse de la case mémoire

b : le numéro du bit, b est compris entre 0 et 7.

Format de l'instruction

Instruction

Code-op	b	f

b_{13} b_{10} b_9 b_7 b_6 b_0

```
Exemple : bcf    PORTB, 2 ;    Mettre le bit 2 du PORTB à 0
          bsf    PORTC, 7 ;    Mettre le bit 7 du PORTC à 1
```

6.1.6. Instructions de branchement

Les instructions *call* et *goto*, utilisent un adressage absolu sur 11 bits. Ces instructions ne permettent pas de réaliser des sauts au delà de 2 kmots. Pour les sauts inter-pages (au-delà de 2kmots), il faut positionner correctement les bits 4 et 3 du registre PCLATH avant d'utiliser l'instruction call ou goto.

Syntaxe : MNEMONIQUE k (adresse sur 11 bits)

Format de l'instruction

Instruction

Code-Op	k (Adresse sur 11bits)

b_{13} b_{11} b_{10} b_0

Exemple :

```
; saut dans la même page
```



```
     call  TEMPO     ; appel du sous programme TEMPO
     goto  FIN       ; branchement à l'étiquette FIN
; saut inter-pages, appel de la procédure CONV se trouvant dans la
4ème page. (page 3)
     bsf   PCLATH, 4  ; sélection de la page 3
     bsf   PCLATH, 3  ;
     call CONV        ; appel du sp. CONV
; l'assembleur de Microchip, fournit un moyen plus simple dans
; le cas des sauts inter-pages.
     movlw  high(CONV)     ; charger dans W la partie haute de
                           ; l'adresse de CONV.
     movwf  PCLATH         ; mettre à jour le registre PCLATH
     call   CONV           ; appel du sp. CONV
```

7. Les Interruptions

Le microcontrôleur PIC16F877 dispose 14 sources d'interruptions. Chaque interruption a un bit d'autorisation (Enable) et un bit indicateur (Flag). Dans le tableau suivant, nous avons énuméré les différentes sources d'interruptions ainsi que les registres associés.

Source d'interruption	Flag	Registre	Enable bit	Registre	Int. en PICC
Timer 0	T0IF	INTCON	T0IE	INTCON	*int_rtcc*
Pin RB0 / INT	INTF	INTCON	INTE	INTCON	*int_ext*
Ch. RB4/RB7	RBIF	INTCON	RBIE	INTCON	*int_rb*
Convert. A/D	ADIF	PIR1	ADIE	PIE1	*int_ad*
Rx USART	RCIF	PIR1	RCIE	PIE1	*int_rda*
Tx USART	TXIF	PIR1	TXIE	PIE1	*int_tbe*
Port série SSP	SSPIF	PIR1	SSPIE	PIE1	*int_ssp*
Module CCP1	CCP1IF	PIR1	CCP1IE	PIE1	*int_ccp1*
Module CCP2	CCP2IF	PIR2	CCP2IE	PIE2	*int_ccp2*
Timer 1	TMR1IF	PIR1	TMR1IE	PIE1	*int_timer1*
Timer 2	TMR2IF	PIR1	TMR2IE	PIE1	*int_timer2*
EEPROM	EEIF	PIR2	EEIE	PIE2	*int_eeprom*
SSP mode I2C	BCLIF	PIR2	BCLIE	PIE2	*int_i2c*
Port parallèle	PSPIF	PIR1	PSPIE	PIE1	*int_psp*

Les bits indicateurs ainsi que les bits de validation des interruptions Timer0, RB0/INT et RB, se trouvent dans le registre INTCON. Alors que les bits indicateur et de validation des interruptions associées aux périphériques se trouvent dans les registres PIR1, PIR2, PIE1 et PIE2 (voir annexe A).

En plus des bits de validation spécifiques; le bit GIE (Global Interrupt Enable) du registre INTCON sert à masquer toutes les interruptions. Vous remarquer aussi que pour les interruptions périphériques s'ajoute un autre bit de validation PEIE (PEripheral

Interrupt Enable) ; ce bit sert à inhiber les interruptions périphériques, tout en conservant les interruptions Timer0, INT et RB4/RB7.

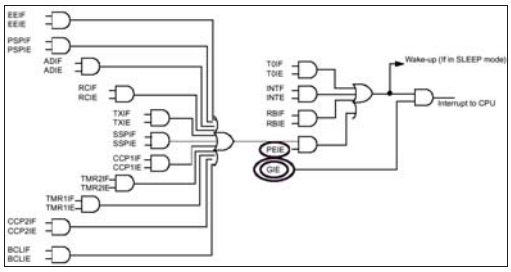


Figure II-9 : Logigramme des interruptions

Pour mettre une interruption en service, il faut :

- mettre à 1 le bit de validation spécifique (T0IE = 1, dans le cas de Timer0),
- mettre à 1 le bit PEIE, s'il s'agit d'une interruption périphérique,
- mettre à 1 le bit GIE, validation globale

Une fois le microcontrôleur a reconnu l'interruption, il remet à 0 le bit GIE, pour bloquer les interruptions, sauvegarde dans la pile l'adresse de retour et charge la valeur 0004H dans le compteur de programme PC. Il incombe au service d'interruption :

- d'identifier la source d'interruption en consultant les indicateurs (flags),
- de remettre à 0 le flag qui a provoqué l'interruption.
- à la fin de la routine d'interruption l'instruction *retfie*, positionne à nouveau le bit GIE.

Exemple de routine d'interruption en C (CCS compiler):

```
#int_ad             // directive de l'interruption A/D
void Isr_ad()       // routine d'interruption
{
     x = ADRESH ;
}
```

7.1.1. Registre INTCON

Ce registre contient les flags et les bits de validation des interruptions de débordement du Timer0, de RB Port (RB4 /RB7) et l'interruption externe RB0/INT.
Le bit GIE (Global Interrupt Enable), sert à masquer toutes les interruptions, si GIE vaut 0, aucune interruption n'est autorisée ; alors que le bit PEIE (Peripheral Interrupt Enable) masque les interruptions associées aux périphériques.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : GIE : Global Interrupt Enable.
GIE = 1 : validation de toutes les interruptions non masquées
GIE = 0 : désactive toutes les interruptions
Bit 6 : PEIE : Peripheral Interrupt Enable
PEIE = 1 : validation de toutes les interruptions périphériques non masquées
PEIE = 0 : désactive toutes les interruptions périphériques
bit 5 : T0IE : Timer TMR0 Overflow Interrup Enable bit.
T0IE = 1 : autorise l'interruption TIMER0
T0IE = 0 : désactive l'interruption TIMER0
Bit 4 : INTE : RB0/INT External Interrup Enable bit.
INTE = 1 : autorise les interruptions externes sur la broche RB0
INTE = 0 : désactive les interruptions externes sur la broche RB0
Bit 3 : RBIE : RB Port Change Interrup Enable bit.
RBIE = 1 : autorise les interruptions par changement d'état sur les broches RB4 à RB7 du PORTB
RBIE = 0 : désactive les interruptions RB4/RB7
Bit 2 : T0IF : Timer TMR0 Overflow Interrup Flag bit.
T0IF = 1 : Le Timer a débordé. Ce flag doit être remis à zéro par soft.
T0IF = 0 : Le Timer n'a pas débordé
Bit 1 : INTF = RB0/INT External Interrup Flag bit
INTF = 1 : Une interruption sur la broche RB0 est survenue
INTF = 0 : Pas d'interruption sur la broche RB0
Bit 0 : RBIF : RB Port Change Interrup Flag bit
RBIF = 1 : au moins une entrée du port B (de RB4 à RB7) a changé d'état.
RBIF = 0 : Aucune entrée de RB4 à RB7 n'a changé d'état.

III - LES PORTS D'ENTREE/SORTIE

Le microcontrôleur PIC16F877 est équipé de 33 lignes d'entrée/sortie réparties sur 5 ports parallèles bidirectionnels.

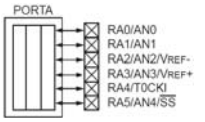
6 lignes sur le PORTA : RA0 .. RA5
8 lignes sur le PORTB : RB0 .. RB7
8 lignes sur le PORTC : RC0 .. RC7
8 lignes sur le PORTD : RD0 .. RD7
3 lignes sur le PORTE : RE0 .. RE2

Chaque port a un registre de direction des données TRIS (TRansfert Input Set) ; les registres TRISA, TRISB, TRISC, TRISD et TRISE correspondent respectivement aux PORTA, PORTB, PORTC, PORTD et PORTE. La mise à 1 ou à 0 d'un bit du registre TRISx, configure la broche correspondante du PORTx en entrée ou en sortie.

La plupart des broches des PORTs sont partagées avec des périphériques. En général si un périphérique est utilisé, les broches correspondantes ne peuvent pas être utilisées comme broches d'entrée/sortie.

Après un RESET, tous les ports sont configurés en entées.

1. PORTA



La broche RA4 est multiplexé avec l'entrée d'horloge externe du timer0 (RA4/T0CKI). Cette broche est une entrée de type trigger de schmitt, à drain ouvert. Les autres broches sont multiplexées, avec les entrées du convertisseur analogique-numérique (AN0 .. AN4) .

Après un reset les broches du PORTA sont configurée en entées analogiques. Si vous voulez utiliser ses broches en tant qu'entrées/sorties TOR, il faut charger le registre ADCON1 par la valeur Binaire '0000011x'.

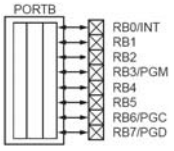
L'exemple suivant configure les trois broches du PORTA RA0, RA1 et RA2 en entrée et les autres en sortie :

```
bcf     STATUS, RP1    ;
bcf     STATUS, RP0    ; Sélection de la Banque 1
movlw   B'00000110'    ; Configure toutes les broches
movwf   ADCON1         ; en entrées numériques
movlw   0x07           ; RA<0:2> en entrées et RA<3:7>
movwf   TRISA          ; en sortie
```


En langage C

```
ADCON1 = 0x6 ;
TRISA = 0x07 ;
```

2. PORTB



Toutes les broches du PORTB possèdent des résistances de tirage (pull-ups). Ces résistances sont mises en œuvre par la mise à 0 du bit RBPU\ du registre OPTION_REG ; elles sont automatiquement désactivées quand le port est configuré en sortie.

Les broches RB7/PGD, RB6/PGC et RB3/PGM sont utilisé pour la programmation du circuit.

Les broches RB4:RB7 peuvent affecter le flag RBIF en cas de changement d'état de l'une des 4 broches, et éventuellement générer une interruption si, celle-ci est autorisée. Il faut bien noter, que seules les broches qui sont configurées en entrées peuvent déclencher l'interruption.

La broche RB0, représente l'interruption externe du PIC (RB0/INT). Cette interruption peut être générée, sur front montant ou descendant selon l'état du bit INTEDG du registre OPTION_REG. Ce registre sera détaillé dans le chapitre TIMERS.

3. PORTC

Le PORTC est multiplexé avec plusieurs périphériques ; on se limite ici à donner les fonctions des différentes broches.

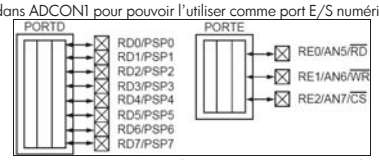
RC0/T1OSO/T1CKI : Input/output port pin or Timer1 oscillator output/Timer1 clock input
RC1/T1OSI/CCP2 : Input/output port pin or Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output
RC2/CCP1 : Input/output port pin or Capture1 input/Compare1 output/PWM1output
RC3/SCK/SCL : RC3 can also be the synchronous serial clock for both SPI and I2C modes.
RC4/SDI/SDA : RC4 can also be the SPI Data In (SPI mode) or data I/O (I2C mode).
RC5/SDO : Input/output port pin or Synchronous Serial Port data output (SPI mode)
RC6/TX/CK : Input/output port pin or Asynchronous Transmit or Synchronous Clock
RC7/RX/DT : Input/output port pin or Asynchronous Receive or Synchronous Data

4. PORTD et PORTE

En plus de leur utilisation comme PORTS E/S; les ports D et E, permettent au microcontrôleur de travailler en mode PSP (Parallel Slave Port) c'est-à-dire, qu'il peut être interfacé avec un autre microprocesseur. Dans ce cas le PORTD représente le bus

de données et le PORTE les signaux de contrôle (RD\, WR\ et CS\). Le registre TRISE dispose de certains bits supplémentaires pour la gestion du mode PSP.

Le PORTE peut être aussi, configuré en mode analogique pour former avec le PORTA les 8 entrées du convertisseur analogique numérique. Par défaut, le PORTE est configuré comme port analogique, et donc, comme pour le PORTA, vous devrez placer la valeur "0000011x" dans ADCON1 pour pouvoir l'utiliser comme port E/S numérique.



Exemple : Ecrire un programme en langage C permettant d'incrémenter le contenu du PORTC à chaque appui sur le bouton BP connecté à la ligne RB0.

1ère Solution

```
#include <16F877.h>
#use delay(clock = 4000000)
#byte     PORTB     = 0x 06
#byte     TRISB     = 0x 86
#byte     PORTC     = 0x 07
#byte     TRISC     = 0x 87
#bit      BP        = PORTB.0
void main()
{    TRISC = 0x00 ;
     TRISB = 0xFF ;
     PORTC = 0x00 ;
     while(1)
     { if(BP == 1)
          PORTC++ ;
     }
}
```

Bien que cette solution semble logique, néanmoins elle présente deux défauts :

- l'action humaine est plus lente que celle du microcontrôleur, d'où l'incrémentation du PORTC avec un nombre de pas aléatoire et non pas de 1.
- Problème du rebondissement du bouton poussoir.

2ème Solution

Utilisation de la ligne RB0 comme entrée d'interruption (INT). Chaque appui sur BP entraine le positionnement du bit INTF du registre INTCON. Le problème revient donc, à détecter le positionnement du bit INTF au lieu de tester l'état de la ligne RB0 (pour savoir l'état du BP). Deux méthodes peuvent être employées : la méthode scrutation (polling) ou la méthode interruption.

Méthode scrutation

Dans cette méthode, on teste continuellement le bit INTF, lorsqu'il passe à 1, on incrémente le PORTC. Il ne faut pas oublier de remettre ce bit à zéro.

```
#include <16F877.h>
#use delay(clock = 4000000)
#byte      PORTB      = 0x 06
#byte      TRISB      = 0x 86
#byte      PORTC      = 0x 07
#byte      TRISC      = 0x 87
#bit       INTF       = INTCON.1
void main()
{    TRISC = 0x00 ;
     TRISB = 0xFF.
     PORTC = 0x00 ;
     while(1)
     { if(INTF == 1)
          {PORTC++ ;
           INTF = 0 ;
           delay_ms(10) ; // cette attente permet de dépasser
                          // le temps de rebondissement.
          }
     }
}
```

Méthode Interruption

Dans cette méthode le positionnement du bit INTF, entraine automatiquement l'exécution d'une fonction spéciale appelée « routine d'interruption ». Cette fonction est repérée dans le compilateur C de CCS par la directive **#int_ext**. Il faut noter qu'on doit mettre à 1 les bits INTE et GIE pour activer l'interruption externe RB0/INT.

```
#include <16F877.h>
#use delay(clock = 4000000)
#byte      PORTB      = 0x 06
#byte      TRISB      = 0x 86
#byte      PORTC      = 0x 07
#byte      TRISC      = 0x 87
#bit GIE         = INTCON.7
#bit INTE        = INTCON.4
#int_ext
void isr_int_Ext()
{
     PORTC++ ;
     delay_ms(10) ;
}
void main()
{    TRISC = 0x00 ;
     TRISB = 0xFF ;
     PORTC = 0x00 ;
     INTE = 1 ; // valider l'int. Externe RB0/INT
     GIE  = 1 ; // validation globale
     while(1) ;
}
```


IV - LES TIMERS

1. Introduction

Les Timers/compteurs sont des périphériques de gestion de temps. Ils permettent des réaliser les fonctions suivantes :

- comptage des évènements
- synchronisation des signaux
- fixer le débit d'une liaison série synchrone ou asynchrone
- génération des événements périodiques (échantillonnage des signaux analogiques, rafraichissement des afficheurs multiplexés ...)
- génération des signaux périodiques (carré, MLI ...)
- mesure de temps...

1.1.1. Fonctionnement du Timer

Les timers sont des compteurs formés généralement d'un pré-diviseur suivi d'un registre compteur de 8 ou 16 bits. L'entrée d'horloge peut être interne *(mode timer)* ou externe *(mode compteur d'événements)*. Lorsque le registre compteur atteint sa valeur maximale et repasse à 0, un bit indicateur (*flag*) sera positionné et une interruption pourra être générée, informant ainsi la CPU du débordement du timer. Il faut bien noter que le programmeur devra remettre à zéro cet indicateur après chaque débordement.
Le microcontrôleur PIC16F877 dispose de trois timers appelés Timer0, Timer1 et Timer2.

2. Timer 0

Ce Timer est implanté dans tous les versions des microcontrôleurs Microchip, son ancienne appellation était RTC (Real Time Clock). Il est formé d'un pré-diviseur programmable (Programmable Prescaler) suivi d'un registre compteur 8 bits (TMR0).

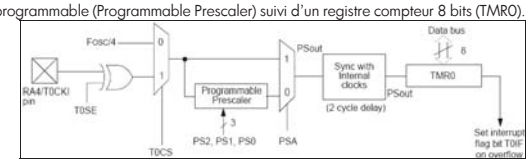


Figure IV-1 : Schéma bloc du Timer 0

Le bit T0CS permet de choisir l'horloge, interne (Fosc/4) ou externe RA4/T0CKI. Dans ce dernier cas l'incrémentation du timer 0 peut se faire soit sur front montant ou descendant suivant la valeur du bit T0SE.

Le prédiviseur est partagé entre le Wtachdog et le Timer 0. Le bit PSA permet donc d'allouer le prédivisieur au Wtachdog ou bien au Timer 0. Les bits PS2, PS1 et PS0 servent à fixer valeur de pré division.

Quand le contenu du TMR0 passe de FF à 00 le bit T0IF du registre INTCON passe à 1 pour signaler un débordement, si le bit T0IE est à 1 alors une interruption timer 0 est déclenchée. Tous les bits de configuration du Timer0 sont implantés dans le registre OPTION_REG.

Registre OPTION_REG

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU\	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : RBPU\ : PORTB Pull Up Enable bit.
RBPU\ = 1 : Pull up désactivé sur le PORTB.
RBPU\ = 0 : Pull up activé

Bit 6 : INTEDG : Interrupt Edge Select bit
INTEDG = 1 : déclenchement de l'interruption RB0 sur front montant.
INTEDG = 0 : déclenchement de l'interruption RB0 sur front descendant.

Bit 5 : T0CS : TMR0 Clock Source Select bit
T0CS = 1 : fonctionnement en mode compteur (horloge externe)
T0CS = 0 : fonctionnement en mode Timer (horloge interne Fosc/4)

bit 4 : T0SE TMR0 Source Edge Select bit.
T0SE = 1 : incrémentation sur transition positive de l'horloge externe
T0SE = 0 : incrémentation sur transition négative de l'horloge externe

Bit 3 : PSA : Prescaler Assignment bit.
PSA = 1 : Prédiviseur assigné au Watchdog (WDT)
PSA = 0 : Prédiviseur assigné au Timer0

Bit 2..0 : PS2, PS1, PS0 : Prescaler Rate Select bits.

PS2	PS1	PS0	Prédiv Timer0	Prédiv. WDT
0	0	0	2	1
0	0	1	4	2
0	1	0	8	4
0	1	1	16	8
1	0	0	32	16
1	0	1	64	32
1	1	0	128	64
1	1	1	256	128

3. Timer 1

La sructure du timer 1 est semblable à celle du Timer 0, cependant il présente des différences notables :

- il est formé d'une paire de registres de 8 bits TMR1H et TMR1L montés en cascade formant ainsi un registre compteur 16 bits.

- Le prédiviseur programmable permet la division de fréquence par 1, 2, 4 ou 8.

- Le timer 1 peut, tout comme le timer 0, fonctionner en mode Timer ou en mode compteur avec la possibilité de connecter un Quartz entre les broches RC0 et RC1.

- Fonctionnement en mode synchrone ou asynchrone. Le mode asynchrone permet au timer 1 de compter en mode veille (Sleep).

- Possibilité de bloquer le comptage.

- Tous les bits de configuration associés au timer1 se trouvent dans le registre T1CON.

Le bit indicateur de débordement *TMR1IF* et le bit de validation de l'interruption *TMR1IE* se trouvent respectivement dans les registres PIR1 et PIE1.

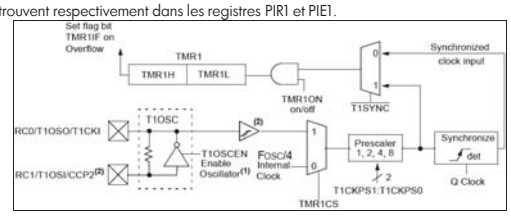


Figure IV-2: schéma bloc du Timer 1

Registre T1CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC\	TMR1CS	TMR1ON
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7..6 : Non implementés.

Bit 5..4 : T1CKPS1:T1CKPS0: Timer1 Input Clock Prescale Select bits

T1CKPS1	T1CKPS0	Prédivision
0	0	1
0	1	2
1	0	4
1	1	8

Bit 3 : T1OSCEN: Timer1 Oscillator Enable Control bit
T1OSCEN = 1 : oscillateur autorisé

T1OSCEN = 0 : oscillateur stoppé

bit 2 : T1SYNC\ : Timer1 External Clock Input Synchronization Control bit
T1SYNC\ = 1 : Pas de synchronisation de l'horloge externe
T1SYNC\ = 0 : Synchronisation de l'horloge externe

Bit 1 : TMR1CS: Timer1 Clock Source Select bit
TMR1CS = 1 : horloge externe
TMR1CS = 0 : horloge interne

Bit 0 : TMR1ON: Timer1 On bit
TMR1ON = 1 : Timer 1 débloqué
TMR1ON = 0 : Timer 1 bloqué

4. Timer 2

Le timer 2 comporte un registre compteur 8 bits (TMR2) avec un prédivideur et un postdiviseur. Ce timer admet uniquement une horloge interne (Fosc/4). Le prédiviseur peut être paramétré par l'une de trois valeurs : 1, 4 ou 16 ; tant disque le postdiviseur permet des divisions de 1 à 16 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ou 16.

Le principe de fonctionnement du timer 2 est différent à ces précédents. Le débordement du compteur n'aura pas lieu lorsque celui-ci dépasse sa valeur maximale (FFH), mais lorsqu'il dépasse une valeur prédéfinie mémorisée dans le registre PR2. La période totale de débordement est donnée par la formule suivante : $T = T_{osc} \times 4 \times \text{Prédiv} \times (PR2 + 1) \times Postdiv$.

Chaque débordement entraine le positionnement de l'indicateur *TMR2IF* et pourra générer une interruption si le bit *TMR2IE* est positionné.

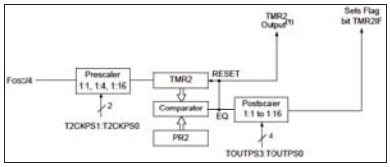


Figure IV-3 : schéma bloc du Timer 2

Le registre T2CON permet la configuration du timer 2.

Registre T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS 2	TOUTPS 1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : Non implementé .

Bit 6..3 : TOUTPS3..TOUTPS0: Timer2 Output Postscale Select bits

TOUTPS3	TOUTPS 2	TOUTPS 1	TOUTPS0	Prédivision
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		:		:
		:		:
1	1	1	1	16

Bit 2 : TMR2ON: Timer2 On bit
TMR2ON = 1 : Timer 2 ON (mise en service)
TMR2ON = 0 : Timer 2 OFF (mise hors service)

bit 1..0 : T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits

T2CKPS1	T2CKPS0	Prédivision
0	0	1
0	1	4
1	x	16

Exemple : On veut incrémenter le contenu du PORTD à chaque demi-seconde. Le microcontrôleur PIC6F877 est piloté par un quartz de 4 Mhz.

Utilisation de la fonction delay

```
#include <16F877.h>
#use delay(clock = 4000000)
#fuses XT, NOWDT, PUT, NOLVP
#byte     PORTD     = 0x 08
#byte     TRISD     = 0x 88
void main()
{     TRISD = 0x00 ;
      PORTD = 0x00 ;
      while(1)
      {     PORTC++ ;
            delay_ms(500) ;
      }
}
```

L'utilisation de la fonction « delay » est simple à mettre en œuvre, cependant l'ajout d'autres tâches au programme affecte la précision.

Utilisation des Timers

L'utilisation des timers permet de libérer le CPU pour réaliser d'autres tâches, en plus de la précision de temps requise.
On dispose de trois timers, on devra donc chercher lequel convient mieux !
Dans cet exemple on doit utiliser le mode timer, donc :

$T_h = 4 \cdot T_{osc} = \frac{4}{f_{osc}} = \frac{4}{4 \cdot 10^6} = 10^{-6} s = 1\mu s$

Timer 1

Le timer1 a un registre de 16bits, il est possible donc qu'on atteint la valeur voulue (500ms).

$T1_{\max} = T_h \cdot \text{Prédiv}_{\max} \cdot TMR1_{\max} = 1\mu s \cdot 8 \cdot 65536 = 524288\mu s \approx 524ms$. Comment peut-on ajuster la période de débordement à 500ms?

Soit T1 la période de débordement (ici 500ms), T1 = T_h*N où N est le nombre d'impulsions comptées par le timer1.

N = Prédiv*M, avec M est le nombre d'impulsions comptées par le registre TMR1(TMR1H :TMR1L). N = T1/T_h = 500000µs/1µs = 500000 impulsions. Pour Prédiv = 8, M = 500000/8 = 62500. Or le timer1 déborde quand TMR1(TMR1H :TMR1L) dépasse 0xFFFF (65536) ; pour que le timer 1 déborde après 62500 imputions, il faut l'initialiser par la valeur 65536 – M = 65536-62500=3036. On devra charger cette valeur dans les deux registres TMR1H et TMR1L : TMR1H = 3036/256 = 11 et TMR1L = 3036-11*256 = 220

Cherchons la valeur à donner au registre T1CON, on a utilisé le timer1 en mode timer (horloge interne) avec une prédivision de 8 ; donc T1CON = 0b00110001 = 0x31

Passons Maintenant au programme ; On peut utiliser la méthode scrutation ou la méthode interruption.

- la méthode scrutation consiste à tester continuellement le bit indicateur TMR1IF pour détecter le débordement.
- La méthode interruption consiste à dérouter le CPU vers le sous programme d'interruption « routine d'interruption » à chaque débordement du timer (bit TMR1TF = 1).

Méthode scrutation

```
#include <16F877.h>
#use delay(clock = 4000000)
#fuses XT, NOWDT, PUT, NOLVP
#include <REG16F.h>

void main()
{     TRISD = 0x00 ;
      PORTD = 0x00 ;
      T1CON = 0x31 ;
      while(1)
      {     if(TMR1IF == 1)
            {   TMR1L = 220 ;
                TMR1H = 11 ;
                PORTD++ ;
                TMR1IF = 0 ;
            }
      }
}
```

Méthode interruption

```
#include <16F877.h>
#use delay(clock = 4000000)
#fuses XT, NOWDT, PUT, NOLVP
#include <REG16F.h>
#int_timer1
void isr_tmr1()
{
      TMR1L = 220 ;
      TMR1H = 11 ;
      PORTD++ ;
}
void main()
{     TRISD = 0x00 ;
      PORTD = 0x00 ;
      T1CON = 0x31 ;
      TMR1IE = 1;PEIE =1 ;GIE =1;
      while(1) ;
}
```

Timer 0

Cherchons la période maximale de débordement : $T0_{\max} = T_h \cdot \text{Prédiv}_{\max} \cdot TMR0_{\max} = 1\mu s \cdot 256 \cdot 256 = 65536\mu s \approx 65ms$. La période de débordement maximale est égale à 65ms, or nous voulons une période de débordement de 500ms. Pour arriver à cette valeur,

on peut utiliser une variable compteur k, tel que T = k*T0 ; T : le prériode d'incrémentation du PORTD (500ms) et T0 : la période de débordement du timer 0. Nous devons choisir la valeur de T0 de telle sorte que T soit un multiple de T0 (k : entier).

On peut prendre T0 = 50ms, d'où k = 500ms/50ms = 10.

T0 = T_h*Prédiv*M, avec M : le nombre d'impulsions comptées par le registre TMR0. Soit N = T0/T_h = Prédiv*M = 50000µs/1µs = 50000.

Prenons Prédiv = 256, alors M = 50000/256 = 195 impulsions.

La valeur de préchargement du registre TMR0 = 256-195 = 61.

La valeur du registre OPTION_REG = 0b00000111 = 0x07

Méthode scrutation

```
#include <16F877.h>
#use delay(clock = 4000000)
#fuses XT, NOWDT, PUT, NOLVP
#include <REG16F.h>
int Count = 10 ;
void main()
{     TRISD = 0x00 ;
      PORTD = 0x00 ;
      OPTION_REG = 0x07 ;
      while(1)
      {     if(T0IF == 1)
            {   TMR0 = 61 ;
                T0IF = 0 ;
              Count--;
                if(Count == 0)
{     PORTD++ ;
                    Count = 10 ;
                }
            }
      }
}
```

Méthode interruption

```
#include <16F877.h>
#use delay(clock = 4000000)
#fuses XT, NOWDT, PUT, NOLVP
#include <REG16F.h>
int Count = 10 ;
#int_timer0
void isr_tmr0()
{     TMR0 = 61 ;
      Count--;
      if(Count == 0)
      {     PORTD++ ;
            Count = 10 ;
      }
}
void main()
{     TRISD = 0x00 ;
      PORTD = 0x00 ;
      OPTION_REG = 0x07 ;
      T0IE = 1; GIE = 1 ;
      while(1) ;
}
```

V - CONVERTISSEUR ANALOGIQUE NUMERIQUE

1. Présentation

Le convertisseur analogique numérique des microcontrôleurs PIC fonctionne sur le principe des approximations successives avec une résolution de 10 bits. Le microcontrôleur PIC16F876 dispose de 5 canaux d'entrées analogiques dénommés AN0 à AN4. Ces entées sont reparties sur les lignes RA0, RA1, RA2, RA3 et RA5 du PORTA. Le microcontrôleur PIC16F877, quant à lui, dispose en plus du PIC16F876 de trois autres entrées analogiques (AN5 à AN7) multiplexés avec les trois broches du PORTE (RE0 à RE2).

Les tensions de référence V_{ref+} et V_{ref-} du module ADC peuvent être choisies de manière logicielle par combinaison des lignes VDD, VSS, RA2 ou RA3.

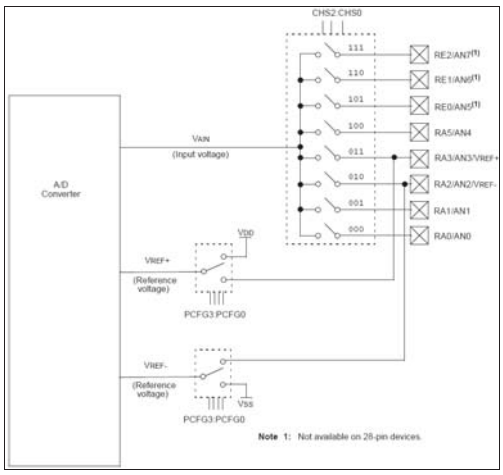


Figure V-1 : schéma bloc du Convertisseur A/N

Le module ADC dispose de trois registres :

⇨ ADRESL : A/D RESult Low register

⇨ ADRESH : A/D RESult High register

⇨ ADCON0 : A/D CONtrol register 0

⇨ ADCON1 : A/D CONtrol register 1

1.1. Registre ADCON1

Ce registre permet essentiellement de définir la fonction de chaque broche du port ; configurer les broches du port en entrées analogiques (RA2 et RA3 peuvent être utilisées pour les tensions de références) ou entrées/sorties numériques.

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : ADFM : A/D Result format select. Définie la façon de charger le résultat de conversion dans les registres ADRESL et ADRESH (figure 2).

ADFM = 1 : Justifié à droite. ADRESH ne contient que les 2 bits MSB du résultat. Les 6 MSB bits de ce registre sont lus comme des "0".

ADFM = 0 : Justifié à gauche. ADRESL ne contient que les 2 bits LSB du résultat. Les 6 LSB bits de ce registre sont lus comme des "0".

Bits 6 .. 4 : Non implémentés

bits 3..0 : PCFG3..PCFG0 : A/D Port ConFiGuration control bits.

PCFG 3 .. 0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0
0000	A	A	A	A	A	A	A	A
0001	A	A	A	A	Vref+	A	A	A
0010	D	D	D	A	A	A	A	A
0011	D	D	D	A	Vref+	A	A	A
0100	D	D	D	D	A	D	A	A
0101	D	D	D	D	Vref+	D	A	A
0110	D	D	D	D	D	D	D	D
0111	D	D	D	D	D	D	D	D
1000	A	A	A	A	Vref+	Vref-	A	A
1001	D	D	A	A	A	A	A	A
1010	D	D	A	A	Vref+	A	A	A
1011	D	D	A	A	Vref+	Vref-	A	A
1100	D	D	D	A	Vref+	Vref-	A	A
1101	D	D	D	D	Vref+	Vref-	A	A
1110	D	D	D	D	D	D	D	A
1111	D	D	D	D	Vref+	Vref-	D	A

A : Analogique D : Digital

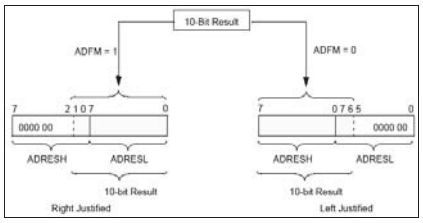
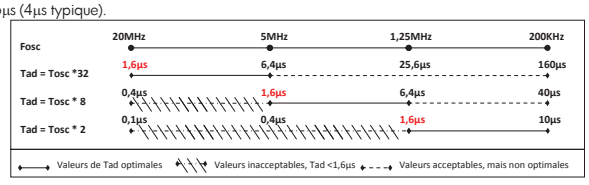


Figure V-2 : Format de résultat de conversion A/N

1.2. Registre ADCON0

On contrôle à travers se registre toutes les opérations du module ADC :

- La fréquence d'horloge du convertisseur A/D, celle-ci est dérivée de la fréquence principale du microcontrôleur Fosc. Pour des raisons électroniques, la période d'horloge de l'ADC, Tad ne doit pas descendre 1,6µs. Donc en fonction des fréquences utilisées pour le quartz du PIC, il faudra choisir le diviseur le plus approprié. Pour cela Microchip a divisé la plage de fréquence en 3 intervalles, afin de garantir un temps Tad ≥ 1,6µs. Dans le cas d'une horloge à base du circuit RC, Tad devrait être entre 2 et 6µs (4µs typique).



- La sélection de l'entrée à convertir.

- le début de conversion est assuré par la mise à 1 du bit GO/$\overline{DONE}$ du registre ADCON0 ou par le module CCP en mode compare. La fin de conversion est signalée par le retour à zéro du bit GO/$\overline{DONE}$ ou la mise à 1 du bit ADIF du registre PIR1.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	-	ADON
b7	b6	b5	b4	b3	b2	b1	b0

Bits 7..6 : ADCS1 ..ADCS0 : A/D Conversion Clock Select bits.

ADCS1	ADCS0	Diviseur	Fréquence maximale du Quartz
0	0	Fosc/2	1,25Mhz
0	1	Fosc/8	5Mhz
1	0	Fosc/32	20Mhz
1	1	-	FRC(horloge dérivée d'un oscillateur RC)

Bit 5..3 : CHS2..CHS0 : Analog Channel Select bits.

CHS2	CHS1	CHS0	Canal	Broche
0	0	0	0	RA0/AN0
0	0	1	1	RA1/AN1
0	1	0	2	RA2/AN2
0	1	1	3	RA3/AN3
1	0	0	4	RA5/AN4
1	0	1	5	RE0/AN5
1	1	0	6	RE1/AN6
1	1	1	7	RE2/AN7

bit 2 : GO/$\overline{DONE}$: A/D Conversion Status bit.

GO/$\overline{DONE}$ = 1 : la conversion progresse (la mise à 1 de ce bit démarre la conversion)

GO/$\overline{DONE}$ = 0 : Conversion terminée (ce bit est remis automatiquement à 0 à la fin de conversion)

Bit 1 : Non implémenté.

Bit 0 : ADON : A/D On bit

ADON = 1 : Convertisseur A/D en service

ADON = 0 : Converitisseur A/D hors service

2. Conversion A/N

La conversion d'un signal analogique en équivalent numérique passe par deux phases :

- l'échantillonnage blocage (sample and hold). Cette opération consiste à connecter l'entrée à convertir à un condensateur interne, qui va se charger à travers une résistance interne jusqu'à la tension appliquée.

- Une fois le condensateur est chargé, déconnecté la tension appliquée, pour procéder à la phase ce conversion.

2.1. Temps d'acquisition

C'est le temps nécessaire pour que le condensateur interne atteigne une tension proche de la tension à convertir. Cette charge s'effectue à travers une résistance interne

et la résistance de la source connectée à l'entrée de l'ADC. Ce temps est augmenté du temps de réaction des circuits et d'un temps qui dépend de la variation de la température.

Tacq = Tamp + Tc + Tcoff

Tamp : temps de réaction des circuits

Tc : temps de charge du condensateur

Tcoff : temps qui dépend du coefficient de température.

Le temps de réaction est fixé à 2µs ; de même Tcoff est limité à une valeur maximale de 1,25µs.

La résistance interne totale (composée de RIC et RSS) varie de 6Kohms sous 6V pour arriver à 12Kohms sous 3V, en passant par 8Kohms sous 5V. De plus Microchip recommande que la résistance de la source doit être inférieure à 10 Kohms.

Tc = CHOLD (RIC + RSS + RS) ln(1/2047)

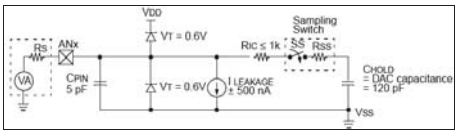


Figure V-3 : schéma de modélisation d'une entrée analogique

Dans les conditions normale et pour une tension de 5V, Tc = 16,47µs.

Le temps d'acquisition Tacq = 19,72µs ≈ 20µs.

2.2. Temps de conversion

C'est le temps nécessaire pour convertir le signal analogique en équivalent numérique. Il dépend de l'horloge interne Tad (Typiquement 1,6µs).

Le microcontrôleur PIC nécessite un temps Tad avant le démarrage effectif de la conversion, et un temps supplémentaire Tad à la fin de la conversion. Soit au total 12 Tad. Donc dans les meilleurs conditions Tconv = 12*1,6µs = 19,2µs.

Il faut bien note qu'un temps équivalent à 2 * Tad est nécessaire avant de pouvoir effectuer une nouvelle conversion.

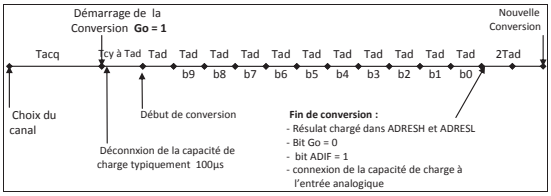


Figure V-4 : Cycles de conversion A/N

3. Etapes de programmation

1) Configurez ADCON1 en fonction des pins utilisées en mode analogique, ainsi que les registres TRISA et TRISE si nécessaire.

2) Validez, si souhaité, l'interruption du convertisseur (ADIE)

3) Paramétrez sur ADCON0 le diviseur utilisé

4) Choisissez le canal en cours de digitalisation sur ADCON0

5) Positionnez, si ce n'est pas déjà fait, le bit ADON du registre ADCON0

6) Attendez le temps Tacq (typiquement 19,7µs sous 5V)

7) Démarrez la conversion en positionnant le bit GO du registre ADCON0

8) Attendez la fin de la conversion (test sur le GO/$\overline{DONE}$ ou ADIF)

9) Lisez les registres ADRESH et si nécessaire ADRESL

10) Attendez un temps équivalent à 2Tad (typiquement 3,2µs)

11) Recommencez au point 7 (ou 4 si vous changez le canal).

VI - MODULE CAPTURE COMPARE PWM (CCP)

1. Présentation

Le module CCP est spécialement utilisé pour la commande des machines électriques (génération des signaux PWM, mesure de vitesse, génération des signaux logiques ...)

Le microcontrôleur PIC16F876/877 dispose de deux modules CCP identiques CCP1 et CCP2. Ces modules sont liés au timer1 pour le mode Capture, Compare et au timer2 pour la mode PWM. Puisque ces modules sont identiques, on utilise dans ce qui suit le suffixe « x », pour désigner le module 1 ou 2.

Chaque module est constitué de deux registres CCPRxH et CCPRxL, pour former un registre de 16 bits et d'un registre de configuration CCPxCON.

Registre CCPxCON (CCP1CON ou CCP2CON)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
b7	b6	b5	b4	b3	b2	b1	b0

bits 7..6 : Non implémenté.

bits 5..4 : CCPxX et CCPxY : PWM Least Significant bits.
Bits non utilisés en modes Capture et Compare.
Ce sont les 2 bits LSB pour le Duty cycle (Rapport Cyclique) en mode PWM. Les 8 bits MSB se trouvent dans le registre CCPRxL.

bits 3..0 : CCPxM3 et CCPxM0 : CCPx Mode Select bits.
0 0 0 0 : Module CCP à l'arrêt.
0 1 0 0 : Mode Capture à chaque front descendant.
0 1 0 1 : Mode Capture à chaque front montant.
0 1 1 0 : Mode Capture tous les 4 fronts montants.
0 1 1 1 : Mode Capture tous les 16 fronts montants.
1 0 0 0 : Mode Compare, broche de sortie mise à 1 et Flag CCP1IF = 1 à l'égalité.
1 0 0 1 : Mode Compare, à l'égalité le Flag CCP1IF est positionné et broche de sortie CCPx est mise à 0.
1 0 1 0 : Mode Compare, à l'égalité le Flag CCP1IF est positionné, une interruption pourra être générée, la broche CCPx est inaffectée.
1 0 1 1 : Mode Compare. Evénement spécial généré (CCP1IF = 1, la broche CCPx inaffectée) le module CCP1 remet TMR1 à 0 ; le CCP2 remet TMR1 à 0 et démarre la conversion A/N si le module ADC est en service.
1 1 x x : Mode PWM.

2. Module CCP en mode capture

En mode capture, le contenu du registre TMR1 (TMR1H : TMR1L) est capturé par les registres CCPRx (CCPRxH :CCPRxL) lorsque un événement extérieur apparaît sur la broche RC2/CCP1 ou RC1/CCP2. Cet événement est programmable par les 4 bits CCPxM du registre CCPxCON. La capture peut avoir lieu à chaque front descendant, à chaque front montant, tous les 4 ou 16 fronts montants. Quand la capture aura lieu, le flag CCPxIF sera positionné et une interruption pourra être générer si le bit de validation CCPxIE du registre PIEx est mis à 1. Si une nouvelle capture survient alors que la valeur dans CCPRx n'a pas été lue, l'ancienne valeur sera écrasée.

Il est donc impératif de configurer la broche CCPx en entrée via le registre TRISC avant de pouvoir utiliser le module CCPx en mode capture.

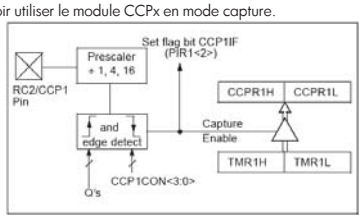


Figure VI-1 : Schéma bloc du mode CAPTURE

3. Module CCP en mode Compare

En mode compare, les 16 bits du registre CCPRx (CCPRxH : CCPRxL) sont constamment comparés avec le contenu du registre TMR1 (TMR1H : TMR1L). L'égalité met à 1 le flag CCPxIF et entraine l'une des actions suivantes :

- Mise à 1 de la sortie RC2/CCP1 ou RC1/CCP2
- Mise à 0 de la sortie RC2/CCP1 ou RC1/CCP2
- Sortie reste inchangée, avec la possibilité de générer une interruption
- Evénement Spécial, ce mode ne change pas la sortie, mais il remet à 0 le contenu du registre TMR1. Le module CCP2 lance automatiquement la conversion A/N (GO = 1) si le module ADC est en service.

Les broches RC2 ou RC1 doivent être configurées en sortie via le registre TRISC.

En mode Special Event (Evénement Spécial), la remise à zéro du registre TMR1 ne positionne pas le flag TMR1IF.

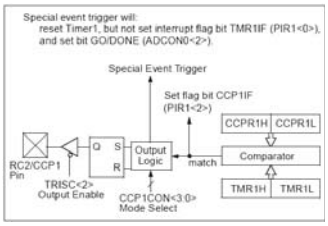
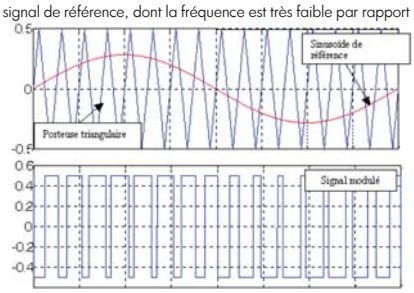


Figure VI-2 : schéma bloc du mode COMPARE

4. Module CCP en mode PWM

La modulation MLI (Modulation en Largeur d'Impulsion), PWM (Pulse Width Modution en anglais) est utilisée dans la variation de la vitesse des moteurs électriques. Le signal PWM est obtenu par modulation d'un signal triangulaire ou dents de scie avec un signal de référence, dont la fréquence est très faible par rapport à la porteuse.

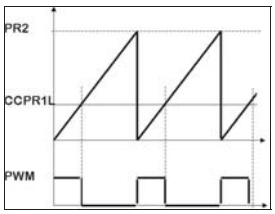


Un signal PWM est caractérise par sa période (T) et son rapport cyclique (Th).

Au niveau des microcontrôleurs PIC, la période de la porteuse est fixée par le timer2. T = Th * Prédiv * (PR2+1) et le rapport cyclique par le registre CCPRxL.

Le signal de sortie PWM passe à 1 lors du débordement du registre TMR2 et repasse à 0 lorsque le contenu du registre TMR2 coïncide avec CCPRxH. Le contenu de registre CCRxL est chargé dans CCPxH au début de chaque période. Cela implique qu'un changement du rapport cyclique ne prend effet qu'au début de la prochaine période. Il

faut bien noter que la valeur écrite dans CCPRxL ne doit pas dépasser le contenu du registre PR2.



Pour atteindre une résolution de 10 bits Microchip a compété le registre CCPxH par les deux bits CCPxX et CCPXY du registre CCPxCON.

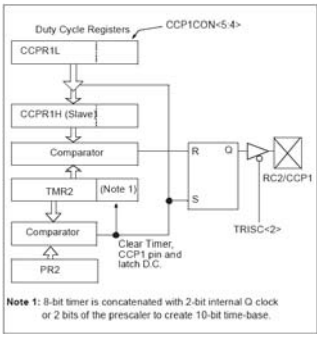


Figure VI-3 : schéma bloc du mode PWM

VII - L'INTERFACE SERIE

1. Présentation

Ce type d'interface permet au microcontrôleur de communiquer avec d'autres systèmes à base de microprocesseur. Les données envoyées ou reçues se présentes sous la forme d'une succession temporelle (sur un seul bit) de valeurs binaires images d'un mot. Il y a deux types de liaison série : synchrone et asynchrone. Nous traitons dans ce chapitre la liaison série asynchrone.

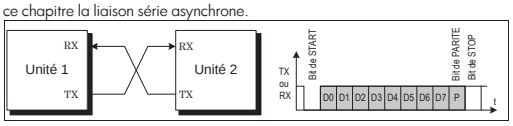


Figure VII-1 : Protocole série asynchrone

La liaison série asynchrone ne possède pas un signal d'horloge de synchronisation. Les unités en liaison possèdent chacune une horloge interne cadencée à la même fréquence. Lorsqu'une unité veut émettre un mot binaire, elle génère un front descendant sur sa ligne émettrice. A la fin de l'émission de ce mot, la ligne repasse au niveau haut. La donnée à transmettre peut contenir un bit supplémentaire appelé "bit de parité" servant à la correction d'erreurs.

Paramètres entrant en jeu pour la norme RS232 :

- Longueur des mots : 7 bits (ex : caractère ASCII) ou 8 bits

- La vitesse de transmission : elle est définie en bits par seconde ou bauds. Elle peut prendre des valeurs allant de 110 à 115 200 bps.

- Parité : le mot transmis peut être suivi ou non d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission.

- Bit de start : la ligne au repos est à l'état logique 1 pour indiquer qu'un mot va être transmis la ligne passe à l'état bas avant de commencer le transfert. Ce bit permet de synchroniser l'horloge du récepteur.

- Bit de stop : après la transmission, la ligne est positionnée au repos pendant 1, 2 ou 1,5 périodes d'horloge selon le nombre de bits de stop.

- Niveau de tension : Un "0" logique est matérialisé par une tension comprise entre 3 et 25V, un "1" par une tension comprise entre -25 et -3 V. Des circuits spécialisés comme le MAX 232 réalise la conversion à partir de niveau TTL.

2. Port série du microcontrôleur PIC

Le port série asynchrone du microcontrôleur présente les caractéristiques suivantes :

- La Port série du microcontrôleur PIC est appelé USART, qui peut fonctionner soit en mode synchrone ou asynchrone, le mode asynchrone peut travailler en mode adressable.
- Fonctionnement en mode full duplex (émission et réception des données en même temps)
- Transmission sur 8 ou 9 bits.
- Réception en mode Adressable, ceci permet au microcontrôleur PIC de fonctionner en mode réseau (utilisation de la norme RS485)
- Pas de bit de parité (le 9ème bit peut être utilisé de manière logicielle pour la détection de parité)
- Un seul bit de stop.

Le port série USART du microcontrôleur PIC16F87x possède 5 registres :

TXSTA : Registre de contrôle et d'état en mode transmission
RCSTA : Registre de contrôle et d'état en mode réception
SPBRG : USART Baud Rate Generator (permettant de fixer le débit)
TXREG : buffer d'émission,
RCREG : buffer de réception,

Le module USART utilise les broches RC6 pour la transmission TX, et RC7 pour la réception RX. Donc pour que le module soit fonctionnel, il faut configurer la broche RC6 en sortie et RC7 en entrée.

2.1. Registre SPBRG

Le Baud Rate Generator est un registre 8 bits qui contient le facteur de division de l'horloge interne (N), afin de fixer le débit de la liaison. Le facteur de division est calculé par une relation simple selon la valeur du bit BRGH.

BRGH = 0 (basse vitesse)	*BRGH = 1 (haute vitesse)*
$débit = \frac{F_{osc}}{64 \times (N+1)}$	$débit = \frac{F_{osc}}{16 \times (N+1)}$
$N = \frac{F_{osc}}{64 \times débit} - 1$	$N = \frac{F_{osc}}{16 \times débit} - 1$

Le nombre N est le nombre entier, arrondi de la valeur trouvée par les équations ci dessus. Il est recommandé d'utiliser si possible les vitesses hautes (BRGH=1), même pour des vitesses faibles, car dans ce cas on minimise l'erreur, en obtenant un nombre N plus grand.

2.2. Registre TXSTA

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	–	BRGH	TRMT	TX9D
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : CSRC : (Clock Source Select bit)
En mode Asynchrone : non utilise
En mode Synchrone
CSRC = 0 : Mode Esclave : horloge à partir d'une source externe
CSRC = 1 : Mode Maitre : horloge générée en interne par BRG
Bits 6 : TX9 : Autorisation du 9ème bit
TX9 = 0 : Emission sur 8 bits
TX9 = 1 : Emission sur 9 bits
Bit 5 : TXEN: Transmit Enable bit
TXEN = 0 : Emission non autorisé
TXEN = 1 : Emission autorisé.
Bit 4 : SYNC: USART Mode Select bit
SYNC = 0 : Mode Asynchrone
SYNC = 1 : Mode synchrone.
Bit 3 : Non implémenté
Bit 2 : BRGH: High Baud Rate Select bit
En mode Asynchrone
BRGH = 1 : haute vitesse.
BRGH = 0 : basse vitesse
Bit 1 : TRMT: Transmit Shift Register Status bit
TRMT = 1 : registre de sérialisation TSR vide.
TRMT = 0 : registre de sérialisation TSR plein.
Bit 0 : TX9D: 9th bit of transmit data.
Ce bit peut être utilisé comme bit de parité

2.3. Registre RCSTA

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RTX9D
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : SPEN : Serial Port Enable bit
SPEN = 0 : Port série désactivé
SPEN = 1 : Port série valide, les broches RC6 et RC7 sont utilisées par le port série.
Bits 6 : RX9 : Autorisation du 9ème bit
TX9 = 0 : Réception sur 8 bits
TX9 = 1 : Réception sur 9 bits
Bit 5 : SREN: Single Receive Enable bit

Non utilisé en mode Asynchrone
Mode Synchrone – Maitre
SREN = 0 : Réception d'un seul octet est désactivé
SREN = 1 : Réception d'un seul octet est activé.
Ce bit est mis à 0 lorsque la réception est terminée.
Bit 4 : CREN: Continuous Receive Enable bit
CREN = 0 : désactive la réception en continu.
CREN = 1 : autorise la réception en continu.
En mode synchrone, CREN écrase SREN.
Bit 3 : ADDEN: Address Detect Enable bit
En mode Asynchrone 9 bits (RX9 = 1)
ADDEN = 1 : valide la détection d'adresse. Charge le registre RCREG quand le 9ème bit reçu vaut 1.
ADDEN = 0 : désactive la détection d'adresse. Tout octet reçu sera transférer dans RCREG.
Bit 2 : FERR: Framing Error bit
FERR = 1 : Erreur de cadrage est détectée (bit de STOP lu comme 0).
FERR = 0 : Pas d'erreur de cadrage
Bit 1 : OERR: Overrun Error bit
OERR = 1 : Erreur de débordement (remis à 0 par la mise à 0 du bit RCEN).
OERR = 0 : Pas d'erreur de débordement.
Bit 0 : RX9D: 9th bit of received data.
Ce bit peut être utilisé comme bit de parité

2.4. Module d'émission

L'émission est autorisée par la mise à 1 du bit TXEN du registre TXSTA. La donnée à transmettre est mise dans le registre TXREG. Ce registre prévient qu'il est vide en mettant le flag TXIF à 1 (bit 4 de PIR1) ; il passe à 0 dés que l'on charge un octet dans le registre TXREG ; et repasse à 1 quand le registre est vidé par transfert de son contenu dans le registre à décalage TSR (Trensmit shift Register). Si on charge alors un 2ème octet dans le registre TXREG le flag TXIF va passer à 0 et y reste tant que le registre TSR n'aura pas complètement sérialisé l'octet précédent. Dés que le bit STOP de l'octet précédent a été transmis, le contenu du registre TXREG est transféré dans TSR et le flag TXIF repasse à 1 signalant ainsi que le registre de transmission TXREG est vide et peut donc recevoir un nouvel octet à transmettre.

Le bit TRMT du TXSTA informe sur l'état du registre TSR. Quand le registre TSR n'a pas fini la sérialisation, TRMT=0. Ce flag repasse à 1 quand le registre est vide, c'est à dire quand le stop a été émis.

Le flag TXIF permet aussi de générer une interruption, à condition qu'elle soit autorisée par mise à 1 du bit TXIE du registre PIE1. Il ne faut pas oublier mettre aussi à 1 les bits de validation PEIE et GIE du registre INTCON.

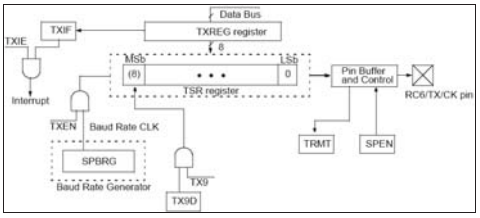


Figure VII-2 : schéma bloc du module de transmission

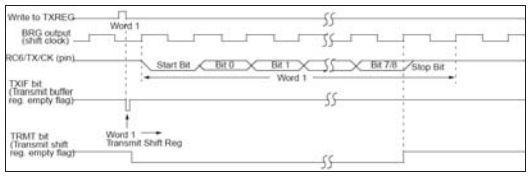


Figure VII-3 : Chronogramme de transmission

2.4.1. Procédure d'émission

- Initialiser le registre SPBRG par la vitesse désirée et mettre le bit BRGH à 1 si la haute vitesse a été choisie.
- Activer le module série SPEN = 1 et Autoriser le mode Asynchrone SYN = 0.
- Eventuellement faire TX9 = 1 si une émission sur 9 bits est désirée.
- Autoriser l'émission TXEN = 1.
- Vérifier si le buffer d'émission est vide (TXIF = 1)
- Si une transmission 9 bits a été choisie, mettre le 9ème bit dans TX9D.
- Mettre l'octet à transmettre dans TXREG.

2.5. Module de réception

La réception est autorisée par la mise à 1 du bit CREN du registre RCSTA.

La donnée reçue est mise dans le registre RCREG. Ce registre prévient qu'il est plein en mettant le flag RCIF à 1 (bit 5 de PIR1). Une interruption de réception pourra être générée si les bits RCIE, PEIE et GIE sont mis à 1 ; la lecture du registre RCREG remet automatiquement le flag RCIF à 0.

Il est possible de recevoir deux octets dans le FIFO RCREG. Si un 3ème octet survient alors que le registre RCREG n'a pas été vidé, une erreur OVERRUN se produit ; elle est signalée par le passage à 1 du bit OERR du RCSTA. L'octet dans le registre de désérialisation RSR (Receive Shift Register) est alors perdu.

De même, une erreur de format peut être détectée par la mise à un du bit FERR du registre RCSTA. La remise à 0 des bits d'erreur, se fait indirectement par la mise à 0 du bit CREN.

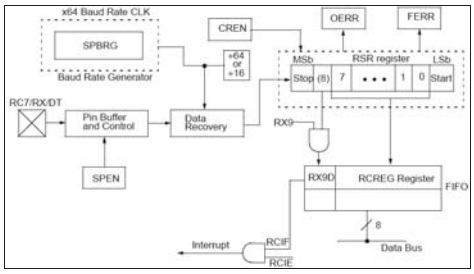


Figure VII-4 : Schéma bloc en mode réception

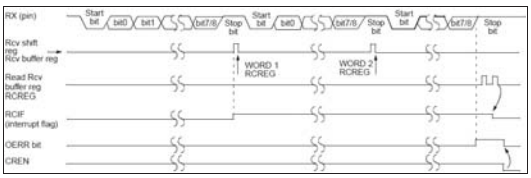


Figure VII-5 : Chronogramme de réception

2.5.1. Procédure de réception

- Initialiser SPBRG par la vitesse désirée et mettre le bit BRGH à 1 si la haute vitesse a été choisi.
- Autoriser le mode Asynchrone en mettant le bit SYN à 0 et SPEN à 1.
- si une réception par interruption est désirée, mettre alors le bit RCIE à 1.
- Eventuellement mettre RX9 à 1 si une réception sur 9 bits est désirée.
- Autoriser la réception par la mise à 1 du bit RCEN.
- Test du Flag RCIF (ou attente d'une interruption) pour savoir si un octet a été reçu.
- Lire éventuellement le 9ème bit de donnée dans RCSTA, et vérifiez les bits FERR et OERR pour déterminer les erreurs éventuelles.
- Lecture du registre RCREG pour récupérer l'octet reçu.
- Si une erreur est survenue, remettre les bits d'erreur à 0 en mettant le bit CREN à 0 puis CREN=1.

Bibliographie

[1] CHRISTIAN TAVERNIER. *Les microcontrôleurs PIC Recueil d'applications*. Dunod , Paris, 2005.
[2] CHRISTIAN TAVERNIER. *Programmation en C des PIC*. Dunod , Paris, 2005
[3] CHRISTIAN TAVERNIER. *Les microcontrôleurs PIC Description et mise en œuvre*. Dunod, Paris, 2000
[4] CHRISTIAN TAVERNIER. *Les MICROCONTROLEURS PIC 10, 12, 16 Description et mise en œuvre*. Dunod , Paris, 2007
[5] Bert VAN DAM. 50 nouvelles applications des microcontrôleurs PIC. Elector, 2010
[6] Datasheet PIC16F87X , ©1999 Microchip Technology Inc. DS30292B
[7] H. Lilen. *Microprocesseurs – Du CISC au RISC*. Dunod, Paris, 1995.

Webographie

[1] BIGONOFF (2002). La gomme MID-RANGE par l'étude des 16F87X (16F876-16F877), Consulté le 6 Nov. 2003.
http://www.abcelectronique.com/bigonoff/
[2] Datasheet PIC16F87X , ©1999 Microchip Technology Inc. DS30292B. Consulté 10 Sep. 2003.
http://www.microchip.com
[3] Philippe LETENNEUR (2003). Les microcontrôleurs PIC16F87X. Consulté en 2006
ftp://www.etab.ac-caen.fr/discip/crgelec/.../PIC16F87X_finale.pdf
[4] D. MENESPLIER (2001). Microcontrôleurs PIC16F876 et 16F877. Consulté le 21 Oct. 2011.
daniel.menesplier.free.fr/Doc/PIC%2016F877.pdf
[5] Compilateur C pour PIC (PCWH). Custom Computer Services, inc.
http://www.ccsinfo.com/