Le Microcontrôleur PIC18F4520

PRESENTATION DE LA FAMILLE PIC18F

Les microcontrôleurs PICs sont des composants dits RISC (Reduced Instructions Set Computer), ou encore composants à jeu d'instructions réduit. La famille PIC18 présente des améliorations assez notables par rapport à la famille précédente (PIC16). Les changements majeurs sont apportés au cœur du microcontrôleur. Le jeu d'instructions a passé à 75 instructions, dont la plupart sont codées sur un seul mot de 16 bits. La mémoire de programme a subi une extension énorme avec un accès linéaire. La capacité de la mémoire de données est multipliée par huit par rapport à la famille PIC16. Le jeu d'instructions est augmenté de huit nouvelles instructions ; ces instructions activables dans l'option de configuration, sont spécialement conçues pour optimiser sa programmation en langages évolués (Basic, C, Pascal…). Les principales caractéristiques de cette famille sont :

- Technologie Nanowatt, garantit une consommation d'énergie très réduite,
- CPU 8 bits à architecture RISC avec une fréquence maximale de 40MHz,
- 75 instructions, extensibles à 83 (16 bits de large),
- Jusqu'à 2Moctets d'espace programme adressable,
- Mémoire RAM de 4Ko maximum, y compris la zone des SFR (Special Function Registers). La zone SFR comporte les registres de la CPU et des périphériques,
- Pile matérielle de 31 niveaux,
- La plupart des versions intègrent une EEPROM pour le stockage des données non-volatiles,
- Intègre un multiplieur 8x8 bits câblés,

La série PIC18 fait partie de la gamme haute performance (High-End) des architectures 8 bits de la société Microchip. Outre les périphériques standards, cette famille intègre des modules avancés tels que les modules CAN, USB, Ethernet, LCD...

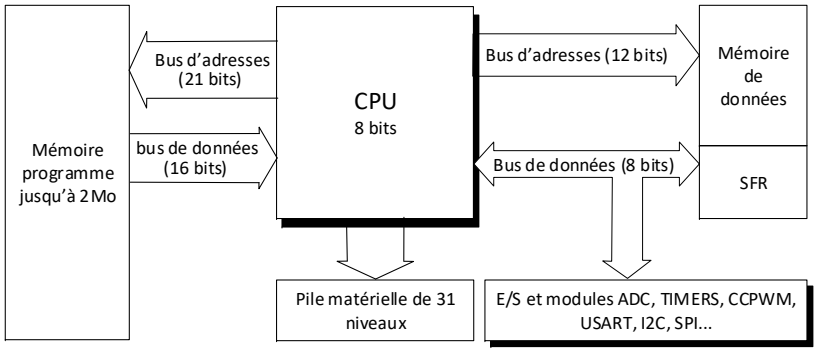


Figure 11 : Schéma bloc du microcontrôleur PIC18

La lettre « **F** » dans la référence du circuit, indique que le circuit contient une mémoire de programme **F**lash. D'autres versions repérées par la lettre **C**, contiennent un OTPROM (One Time PROM).

Registres du microcontrôleur PIC18F

Le microcontrôleur PIC18 comporte un ensemble de registres répartis en deux catégories : les registres rattachés à l'unité centrale et les registres associés aux périphériques. Tous les registres sont regroupés dans une zone nommée *Special Function Registers* (**SFR**) et font partie intégrante de la mémoire de données. Celle-ci est organisée en octets, par conséquent, tous les registres sont de huit bits. Quelques registres sont concaténés pour former des registres 12 bits, 16 bits ou plus. La **Erreur ! Source du renvoi introuvable.** présente les registres de la CPU. Les registres associés aux interruptions (non représentés sur la figure) font partie de l'unité centrale.

Compteur de programme PC

Le Compteur de Programme **PC**, contient l'adresse de la prochaine instruction à exécuter. Il est composé de trois registres de 8 bits (**PCL**, **PCH** et **PCU**) puisqu'il est codé en entier sur 21 bits. La mémoire de programme est organisée en octets, tandis que les instructions sont codées sur 16 bits (ou 32 bits pour quelques instructions). Le bit le plus faible du registre PCL est fixé à zéro (PC est incrémenté de 2), pour éviter un mauvais alignement des instructions.

Registres de gestion de la pile

Le pointeur de pile (nommé **STKPTR** pour le PIC18), est un registre de 5 bits, utilisé pour l'adressage d'une pile de 31 niveaux. Cette pile dispose de son propre

espace mémoire, destinée principalement pour la sauvegarde des adresses de retour des sous-programmes.

Les microcontrôleurs PIC18 disposent aussi, d'un registre *Top Of Stack* (**TOS**), composé de trois registres (**TOSL**, **TOSH**, **TOSU**). Ce registre peut être utilisé avec les instructions PUSH et POP pour une implémentation d'une pile logicielle.

Registre de travail WREG

Le *Work Register* **WREG**, dénommé **W** dans les instructions, est le seul registre à usage général dont dispose le microcontrôleur. Il s'agit d'un accumulateur dans la signification habituelle, où la plupart des opérations arithmétiques et logiques sont effectuées à l'aide de ce registre.

Le registre d'état STATUS

Le registre **STATUS** est lié étroitement à l'unité arithmétique et logique. Il renseigne à travers des bits indicateurs (Flags) de l'état de quelques opérations. La fonction de ces indicateurs est donnée comme suit :

			N	OV	Z	DC	C

Figure 12 : Registre STATUS du microcontrôleur PIC18

C : indicateur de retenue (carry), la polarité de ce bit est inversée en cas d'un emprunt.

DC : indicateur de retenue auxiliaire, la polarité de ce bit est inversée en cas d'un emprunt.

Z : indicateur d'un résultat nul.

OV : indicateur de dépassement, ce bit est utilisé par les opérations arithmétiques signées.

N : indicateur de signe.

Registre de sélection de banque BSR

La mémoire de données est adressée sur 12 bits. Le registre **BSR** (**B**ank **S**elect **R**egister) contient les quatre bits de la partie haute de l'adresse <11 :8>. Il est utilisé avec la partie basse de l'instruction pour l'adressage direct de la mémoire de données.

Registres d'adressage indirect FSR

Les microcontrôleurs PIC18 disposent de trois registres d'accès indirect à la mémoire de données **FSR0**, **FSR1** et **FSR2** (**F**ile **S**elect **R**egister). Ces registres sont susceptibles de contenir l'adresse de la mémoire de données qui s'étend sur 12 bits.

Par conséquent, chacun d'eux est composé de deux registres de 8 bits **FSRnL** et **FSRnH**, où n indique le numéro du registre.

Registres du multiplieur PRODL et PRODH

Le PIC18 intègre une unité de multiplication 8x8 bits. Le résultat fourni est stocké dans une paire de registres de 8 bits, PRODL (Octet bas) et PRODH (Octet Haut).

Registres de contrôle et de configuration

Le microcontrôleur PIC18F4520 comporte un ensemble de registres de contrôle et de configuration. Nous citons à titre indicatif les registres RCON, WDTCON, les registres CONFIGx, les registres DEVID et les registres ID.

CARACTERISTIQUES DU PIC18F4520

Caractéristiques de la CPU

- CPU à architecture RISC (8 bits)
- 32 KOctets de mémoire flash intégré
- SRAM de 1536 Octets (6 bnaques)
- EEPROM de données de 256 Octets,
- Interruptions à 2 niveaux de priorité,
- Pile de 31 niveaux,
- Vitesse d'exécution jusqu'à 10MIPS,
- Instructions codées sur 16bits
- Multiplieur 8x8 bits
- Chien de garde (WatchDog),

Caractéristiques des périphériques

- 5 ports d'entrées/sorties bidirectionnels (A, B, C, D et E),
- 4 Timers
- Deux modules CCP « Capture, Compare et PWM »
- ADC à 13 entrées, avec une résolution de 10 bits,
- Deux comparateurs analogiques,
- Détection de niveau de tension (HLVD)

- Module série synchrone travaillant en mode SPI ou I2C,
- Module série asynchrone avancé,
- Possibilité de se connecter à un microprocesseur via un port parallèle de 8 bits avec signaux de contrôle $\overline{RD}$, $\overline{RW}$ et $\overline{CS}$.

HORLOGE DU MICROCONTROLEUR PIC18

Toutes les opérations de la CPU sont cadencées par une horloge à fréquence fixe. C'est l'une des caractéristiques principales qui fixe la vitesse d'exécution. Pour les microcontrôleurs PIC, la fréquence d'horloge principale est divisée en interne par quatre pour générer le cycle d'instruction. Les broches OSC1/RA7 et OSC2/RA6 sont réservées au circuit d'horloge.

Options de l'oscillateur

Le microcontrôleur PIC18F4520 peut opérer avec 10 sources d'horloges différentes. Les quatre bits : FOSC3, FOSC2, FOSC1 et FOSC0 du registre CONFIG1H permettent la sélection de l'un des modes d'horloges. Le Tableau 4 présente une description de différents modes.

Tableau 4 : Différents modes d'horloges

Mode	Description
LP	Résonateur à Quartz (ou Céramique), faible consommation jusqu'au 200 kHz (typiquement 32kHz)
XT	Résonateur à Quartz (ou Céramique) de 1 à 4Mhz
HS	Résonateur à Quartz (ou Céramique) à Haute Vitesse, de 4 à 25Mhz
HSPLL	Résonateur à Quartz (ou Céramique) à Haute Vitesse avec PLL (Multiplication de fréquence par 4). Dans ce cas la fréquence du quartz ne doit pas dépasser 10 MHz.
RC	Circuit RC externe, FOSC/4 est fournie sur la broche RA6
RCIO	Circuit RC externe, la broche OSC2/RA6 peut être utilisée en E/S (Entrée/Sortie TOR)
INTIO1	Oscillateur interne, FOSC/4 est fournie sur la broche RA6 et RA7 peut être utilisée en E/S.
INTIO2	Oscillateur interne, les broches RA6 et RA7 peuvent être utilisées en E/S.
EC	Horloge externe appliquée à l'entrée OSC1/RA7, FOSC/4 est fournie sur la broche OSC2/RA6
ECIO	Horloge externe appliquée à l'entrée OSC1, la broche OSC2/RA6 peut être utilisée en E/S.

CIRCUIT RESET

Les microcontrôleurs PIC18 disposent de huit sources de RESET distinctes :

- RESET à la mise sous tension appelée **POR** (Power On Reset)
- RESET par la mise à la masse, pendant le fonctionnement normal de la ligne $\overline{MCLR}$.
- RESET par la mise à la masse pendant le fonctionnement en mode sommeil de la ligne $\overline{MCLR}$.
- RESET par débordement du timer Watchdog
- RESET par la détection d'une chute de la tension d'alimentation **BOR** (Brown Out Reset)
- RESET logicielle : suite à l'exécution de l'instruction **RESET**.
- RESET suite à une opération d'empilement alors que la pile est pleine.
- RESET suite à une opération de dépilement alors que la pile est vide.

La mise à zéro de broche $\overline{MCLR}$ initialise les principaux registres, ce qui permet de démarrer la CPU à partir d'une adresse fixe. Le microcontrôleur PIC18F4520 intègre un bloc de réinitialisation, dont le rôle est de générer à la mise sous tension, une impulsion de largeur suffisante à la stabilisation de l'horloge de l'oscillateur. Cette structure décharge l'utilisateur du dimensionnement du circuit RC traditionnel. La broche $\overline{MCLR}$ peut être connectée à la tension d'alimentation par une résistance de 1 et 10 kΩ.

Power On Reset (POR)

Le circuit Power On Reset (POR), génère une transition positive dès la mise sous tension du microcontrôleur.

Power-up Timer (PWRT)

Le PWRT est un compteur 11 bits incrémenté par l'horloge INTRC. Ce qui donne un intervalle de temps de l'ordre de 65ms, pendant lequel le microcontrôleur est maintenu à l'état RESET.

Oscillator Start-Up Timer (OST)

L'Oscillateurr Start-Up Timer fournit un retard de 1024 cycle d'horloge supplémentaire afin de stabiliser l'oscillateur principal. Ce timer n'est actif que lorsque l'oscillateur principal est configuré en mode LP, XT ou HS.

PLL Lock Time-Out

Si la PLL est activée, un décalage d'environ 2 ms est ajouté pour garantir le verrouillage.

La Figure 13 représente le chronogramme type d'une séquence de RESET à la mise sous tension (POR).

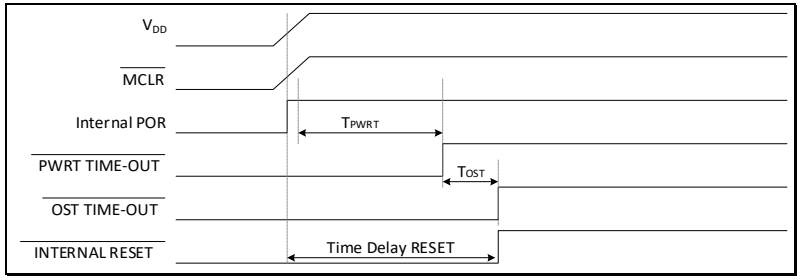


Figure 13 : Chronogramme type d'une séquence de RESET

ORGANISATION DE LA MEMOIRE

Les microcontrôleurs PIC18F4520 disposent de trois types de mémoires :

- Mémoire de programme (Flash)
- Mémoire de données (SRAM)
- EEPROM de données

La mémoire de données (dénommée **File Register** par Microchip) contient les registres des ressources internes du microcontrôleur et les variables du programme proprement dites. Cette mémoire fait appel un mécanisme de pagination qui complique un peu son accès.

Mémoire de programme

Les microcontrôleurs PIC18 disposent d'un compteur de programme (PC) de 21bits, ce qui leur permet d'adresser un espace maximal de 2Moctets (000000h à 1FFFFFH) sans aucune restriction. La mémoire flash intégrée dans la famille PIC18 ne dépasse pas 128ko, soit de 32Ko pour le PIC18F4520 ; l'espace restant est lu comme zéro. Les instructions sont codées sur 16 bits, alors que la mémoire programme est organisée en octets, ce qui explique la parité d'adressage de la mémoire ; par suite le bit LSB du compteur de programme (PC) est toujours nul.

Il est parfois nécessaire de stocker des données, tels que les messages d'invites, dans la mémoire de programme. Cette procédure est gérée dans la famille PIC16, par une méthode fastidieuse, qui fait appel à l'utilisation du compteur du programme.

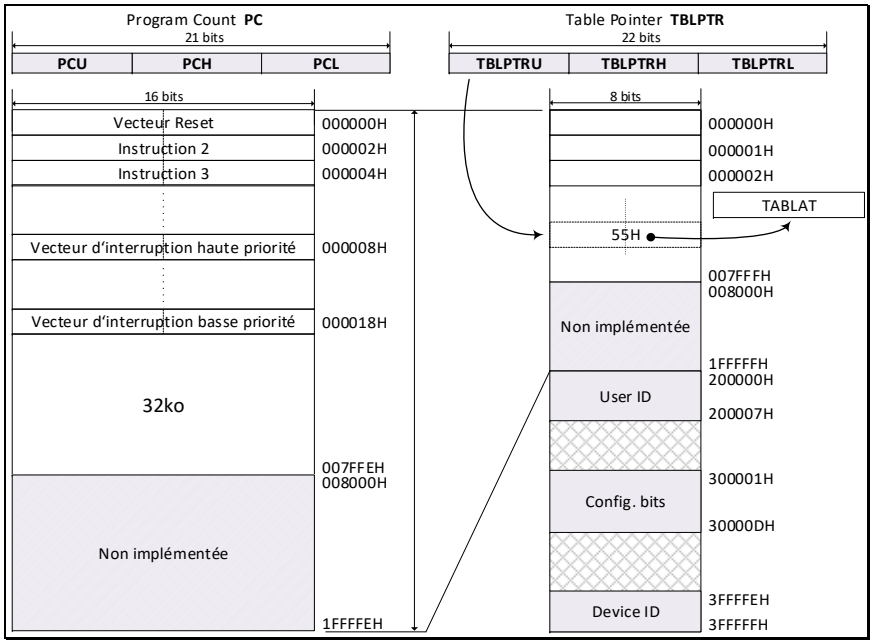


Figure 14 : organisation de la mémoire programme

Dans les PIC18, Microchip a introduit deux nouveaux registres **TBLPTR** (Table Pointer) et **TABLAT** (Table Latch). Ces deux registres sont utilisés par deux puissantes instructions **TBLRD*** et **TBLWT*** pour lire ou écrire des données dans la mémoire de programme en cours d'exécution. Dans ce cas le registre TBLPTR est utilisé comme pointeur d'adresse et le registre TABLAT comme registre de donnée. Le registre TBLPTR comporte 22 bits, il est possible donc d'accéder à des emplacements au-delà de 2Mo. Le 22ème bit permet l'accès aux registres « device ID », « user ID » et des registres « **CONFIGx** ». Etant donné qu'il est mappé dans la zone SFRs, le registre TBLPTR est composé de trois registres de 8 bits TBLPTRL, TBLPTRH et TBLPTRU.

La Figure 14 montre, qu'au niveau des instructions, la mémoire est considérée comme étant une mémoire formée des mots de 16 bits (double octets) ; le compteur de programme s'incrémente toujours de deux unités. Le vecteur RESET occupe l'adresse 00000H. Les vecteurs d'interruptions démarrent à l'adresse 00008H ou 00018H, selon type de priorité (haute priorité ou basse priorité). Lorsque la mémoire est pointée par le registre **TBLPTR** « Table Pointer » (illustré dans la partie gauche de la Figure 14), la mémoire est considérée comme une succession des cases de 8

bits. Le registre TBLPTR permet l'accès octet par octet à la totalité de la mémoire de 000000H à 3FFFFFH.

Mémoire de données

La mémoire de données (dénommé **F**ile Registers) contient les registres **SFR** (Special Function Registers), et les cases mémoires à usage général appelées **GPR** (General Purpose Register). La zone des registres **SFR** comporte les registres de la CPU et celles des périphériques, tandis que la partie restante **GPR**, constitue la mémoire de stockage des données utilisateur.

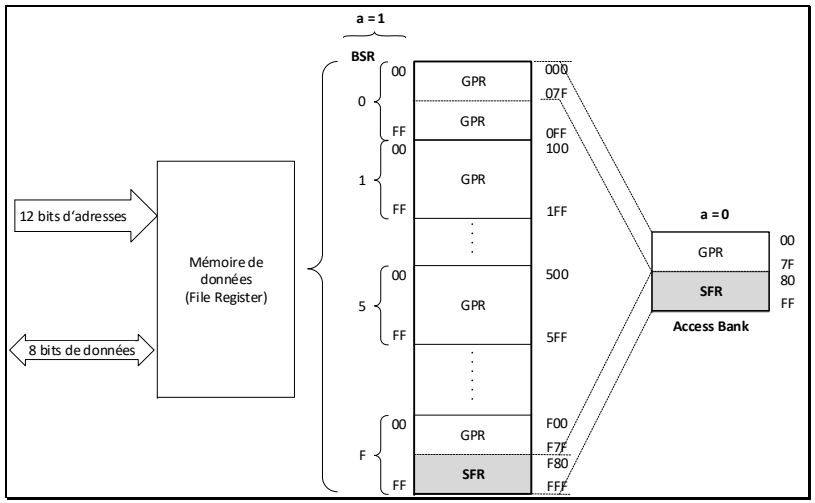


Figure 15 : Mémoire de données

Elle utilise un mécanisme de pagination (nommé ***bank*** par Microchip) plus ou moins hérité des familles précédentes, mais avec une organisation beaucoup plus souple.

La mémoire « File Register » est organisée en octets, elle comporte 16 banques de 256 octets chacun, soit un espace maximal de 4Ko (soit 1536 octets de GPR pour le microcontrôleur PIC18F4520). La sélection des banques est obtenue par les 4 bits du registre **BSR** (Bank Select Register).

Il est important de savoir que tous les registres **SFR** sont regroupés dans la partie haute de la banque 15 (de F80H à FFFH), mais il est possible d'accéder à ces registres à partir de la banque 0 en utilisant le mode « ***Access bank*** ».

Trois modes possibles sont utilisés pour l'accès à la mémoire de données :

- « *Access Bank* », la banque 0 devient une combinaison de 128 octets de la partie basse de la mémoire (00H à 7FH) et 128 octets de la partie haute de la banque 15 (registres **SFR**).
- « *banked* », l'adresse d'une case mémoire est la concaténation de 4 bits du registre **BSR** et les 8 bits provenant de l'instruction, pour former une adresse de 12 bits. Il est possible en utilisant l'instruction **movff** d'accéder de manière linéaire à la totalité de la mémoire.

Accès indirect en utilisant l'un des registres **FSR0**, **FSR1** ou **FSR2**. Chaque registre est formé de deux registres de 8 bits (FSR0H, FSR0L pour FSR0 par exemple). L'accès à la totalité de la mémoire se fait d'une manière linéaire et sans aucune contrainte de pagination.

Ces différentes méthodes d'accès à la mémoire seront étudiées en détail dans la section « modes d'adressages ».

Registres SFR du microcontrôleur PIC18F4520

SFR (Special Function Registres) est l'ensemble de tous les registres utilisés par la CPU et par les périphériques. Ces registres sont implémentés dans la RAM à partir de l'adresse F80H jusqu'à FFFH.

FFFh	TOSU	FDFh	INDF2	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2	FBCh	CCPR2H	F9Ch	
FFBh	PCLATU	FDBh	PLUSW2	FBBh	CCPR2L	F9Bh	OSCTUNE
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	
FF9h	PCL	FD9h	FSR2L	FB9h		F99h	
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	PWM1CON	F97h	
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS	F96h	TRISE
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD
FF4h	PRODH	FD4h		FB4h	CMCON	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	
FEFh	INDF0	FCFh	TMR1H	FAFh	SPBRG	F8Fh	
FEEh	POSTINC0	FCEh	TMR1L	FAEh	RCREG	F8Eh	
FEDh	POSTDEC0	FCDh	T1CON	FADh	TXREG	F8Dh	LATE
FECh	PREINC0	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD

FEBh	PLUSW0	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh		F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	
FE7h	INDF1	FC7h	SSPSTAT	FA7h	EECON2	F87h	
FE6h	POSTINC1	FC6h	SSPCON1	FA6h	EECON1	F86h	
FE5h	POSTDEC1	FC5h	SSPCON2	FA5h		F85h	
FE4h	PREINC1	FC4h	ADRESH	FA4h		F84h	PORTE
FE3h	PLUSW1	FC3h	ADRESL	FA3h		F83h	PORTD
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA

Figure 16 : Registres spéciaux (SFR) du PIC18f4520

Nous donnons à Figure 16 la liste de tous les registres avec leurs emplacements mémoire. Vous remarquez que le registre WREG fait partie des SFR. Les cases grisées sont non implémentées et les cases hachurées représentent des registres virtuels (qui n'existent pas physiquement). En mode Access Bank cette zone est translatée à l'adresse 080H.

LES PORTS D'ENTREES-SORTIES

Le microcontrôleur PIC18F4520 dispose de 36 broches d'entrées/sorties (I/O pour Input/Output) réparties sur cinq ports parallèles bidirectionnels. C'est à travers ces ports que le microcontrôleur interagit avec le monde extérieur.

La plupart des lignes (broches) sont partagées (multiplexées) avec d'autres périphériques. Lorsqu'un module périphérique est activé, les broches correspondantes sont lui assignées.

Les ports sont représentés par des lettres alphabétiques. Le Tableau 5 illustre la désignation des ports et des broches correspondantes.

Tableau 5 : les ports d'entrées/sorties

Ports	broche 7	broche 6	broche 5	broche 4	broche 3	broche 2	broche 1	broche 0
PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
PORTE	-	-	-	-	RE3	RE2	RE1	RE0

- ***RA7** et **RA6**, sont généralement utilisées par l'oscillateur principal.*

- *RE3 est disponible en entrée logique, que lorsque la fonctionnalité de $\overline{MCLR}$ est désactivée.*

Chaque broche peut être configurée soit en entrée soit en sortie, et chaque port est contrôlé par trois registres :

- **PORTx** : c'est le registre de donnée utilisé pour lire ou modifier l'état des lignes correspondantes (x ∈{A, B, C, D, E}).
- **TRISx** (TRansfert Input Set) : registre de direction des données ; il permet de définir individuellement le sens de chaque ligne du port en entrée ou en sortie. La mise à 1 ou à 0 d'un bit du registre TRISx, configure la broche correspondante du PORTx en entrée ou en sortie.
- **LATx** : vous pouvez le considérez comme image du PORTx. Il est très utile pour les opérations «read-modify-write», c-à-d, lorsqu'on veut modifier l'état de quelques lignes et non pas la totalité du port. Il est vivement conseillé lors d'une opération de modification des bits d'utiliser le registre LATx au lieu du PORTx correspondant.

```
bsf  LATB, 0    ; mieux que bsf  PORTB, 0
```

La plupart des broches des ports sont partagées avec d'autres périphériques. En général, si une broche est utilisée par un périphérique, celle-ci ne peut pas être utilisée comme broche d'entrée/sortie. Après un RESET, tous les ports sont configurés en entrées logiques, excepté les proches dédiées au convertisseur analogique numérique.

Note : *Le convertisseur analogique numérique admet 13 entrées réparties sur les ports A, B et E ; les broches correspondantes sont configurées par défaut en entrées analogiques. Si vous voulez les utiliser en entrées/sorties numériques ; il suffit d'écrire la valeur 0x0F dans le registre ADCON1 et 0x07 dans le registre CMCON.*

PORTA

Le PORTA est un port de 8 bits, mais du fait du partage de certaines de ces lignes avec d'autres ressources indispensables, le nombre des lignes disponibles peuvent se trouver réellement réduites. Les lignes RA6 et RA7 du port PORTA sont multiplexées avec les lignes de l'horloge principale OSCI et OSCO, donc elles sont disponibles que dans le cas où le microcontrôleur fonctionnement avec une horloge interne. La broche RA4 est multiplexé avec l'entrée d'horloge externe du timer0 (T0CKI) ou la sortie du comparateur analogique. Les autres broches sont multiplexées avec les entrées du convertisseur analogique-numérique ou le comparateur analogique (AN0 .. AN5).

La broche RA5 peut être utilisée par le module SPI ou par le module de la détection de la variation de tension HLVD.

La broche RA2 peut être utilisée comme sortie de référence (CVref), le bit CVROE du registre CVRCON permet de configurer la ligne RA2 pour la génération d'une tension de référence.

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

```
ADCON1  = 0x0F  ;
CMCON   = 0x07  ;
TRISA   = 0xF8  ;
```

PORTB

Toutes les broches du PORTB possèdent des résistances de tirage (pull-up). Ces résistances sont mises en œuvre par la mise à 0 du bit $\overline{RBPU}$ du registre INTCON2 ; elles sont automatiquement désactivées pour les broches configurées en sortie.

Les broches RB7/PGD, RB6/PGC et RB5/PGM sont utilisées pour la programmation du circuit.

La broche RB3 peut être utilisée par le module CCP2 ou l'entrée analogique AN9.

Les broches RB0, RB1 et RB2 peuvent être utilisées comme entrées d'interruptions externes ou entrées analogiques.

Le changement d'état de l'une de quatre broches RB4 à RB7 peut éventuellement générer une interruption. Il faut bien noter, que seules les broches qui sont configurées en entrées peuvent déclencher l'interruption.

PORTC

Le PORTC est multiplexé avec plusieurs périphériques ; nous présentons de façon globale les fonctions des différentes broches.

RC0/T1OSO/T13CKI : sortie de l'oscillateur du TIMER1 ou entrée d'horloge du TIMER1 ou TIMER3

RC1/T1OSI/CCP2 : entrée de l'oscillateur du TIMER1 ou Capture2/Compare2/PWM2

RC2/CCP1/P1A : Capture1/Compare1/PWM1

RC3/SCK/SCL : horloge en modes SPI ou I2C.

RC4/SDI/SDA : entrée de données en mode SPI ou données bidirectionnelles en mode I2C.

RC5/SDO : sortie de données en mode SPI

RC6/TX/CK : donnée de transmission série asynchrone ou horloge en mode Synchrone

RC7/RX/DT : donnée de réception série asynchrone ou donnée bidirectionnelle en mode Synchrone

PORTD et PORTE

En plus de leur utilisation comme ports d'entrées/sorties, les ports D et E, permettent au microcontrôleur de travailler en mode **PSP** (*Parallel Slave Port*), ce mode lui permet d'être interfacé avec un autre microprocesseur. Dans ce cas le PORTD représente le bus de données et le PORTE les signaux de contrôle ($\overline{RD}$, $\overline{WR}$ et $\overline{CS}$). Le registre TRISE dispose de certains bits supplémentaires pour la gestion du mode PSP. Les lignes RD5, RD6 et RD7 sont partagées avec le module ECCP1.

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 comme dans le cas du PORTA, vous devez placer la valeur "00001111" dans le registre ADCON1 pour pouvoir l'utiliser comme port E/S numérique. Cette remarque concerne aussi le PORTB.

Outils de développement

Pour mettre en œuvre la programmation des microcontrôleurs, nous avons besoin des outils de développement et de test. Ces outils peuvent être classés en deux catégories : logiciels et matériels. Les outils logiciels incluent les éditeurs, assembleurs, compilateurs, simulateurs, débogueurs et programmes de communication. Les outils matériels comprennent, des cartes de démonstration, des analyseurs logiques, des émulateurs, des oscilloscopes et des sondes logiques.

Microchip a fourni aux utilisateurs de ses microcontrôleurs l'environnement de développement intégré MPLAB X IDE (*Integrated Development Environment*). MPLAB X est un logiciel puissant, extensible et configurable. Il incorpore des outils puissants de développement, de simulation, de débogage et de programmation des microcontrôleurs sur site. MPLAB X est disponible gratuitement sur le site de Microchip.

COMPILATEUR MPLAB XC8

A nos jours, les microcontrôleurs sont bien adaptés à la programmation en langage haut niveau tel que le langage C. Le langage C est conçu au début des années 70 pour fournir un grand nombre des caractéristiques des langages de bas niveau, convenant au développement des drivers et des systèmes d'exploitation. Ces caractéristiques de bas niveau font du langage C le choix idéal pour la programmation des microcontrôleurs. Plusieurs compilateurs C sont disponibles pour la programmation des microcontrôleurs PIC.

MPLAB XC8 de Microchip, est un compilateur C puissant développé pour les microcontrôleurs 8 bits, PIC10/12/16/18. Il est fourni en trois versions : licence PRO, licence d'essai en 60 jours et version gratuite.

Un fichier d'en-tête est généralement inclus dans chaque fichier source C. Le fichier <xc.h> est un fichier d'en-tête générique qui inclura d'autres fichiers d'en-tête spécifiques à l'architecture utilisée lorsque vous compilez votre projet. L'inclusion de ce fichier permettra l'accès aux SFR via des variables spéciales, ainsi que des macros

qui permettent un accès spécial à la mémoire ou l'inclusion d'instructions spéciales, comme CLRWDT.

Bits de configuration

Les bits de configuration sont définis par le biais de la directive `#pragma config`. Les paramètres et les états associés à chaque circuit peuvent être déterminés à partir du fichier ***pic18_chipinfo.html*** du dossier ***docs***.

Sous MPLAB X IDE, Il est possible aussi d'utiliser la commande **Set configuration bits** du menu **Production** (**Erreur ! Source du renvoi introuvable.**).

La définition des bits de configuration aura la forme suivante :

```
#pragma config  paramètre = état/valeur
#pragma config registre = valeur
```

Exemple

```
#pragma  config  OSC = HS, WDT = OFF, LVP = OFF, PWRT = ON, DEBUG = ON
```

Type de données

Type entier

Le compilateur MPLAB XC8 prend en charge le type entier avec des tailles de 1, 2, 3 et 4 octets ainsi qu'un type à bit unique. Le Tableau 6 montre le type de données, leur taille et le type arithmétique correspondants. Les types par défaut sont représentés en gras.

Tableau 6 : types des entiers

Type	Taille	Type Arithmétique	Norme MISRA
__bit	1 bits	Entier non signé	
signed char	8 bits	Entier signé	int8_t
unsigned char	8 bits	Entier non signé	uint8_t
signed short	16 bits	Entier signé	int16_t
unsigned short	16 bits	Entier non signé	uint16_t
signed int	16 bits	Entier signé	Int16_t
unsigned int	16 bits	Entier non signé	uint16_t
__int24	24 bits	Entier signé	int24_t
__uint24	24 bits	Entier non signé	uint24_t
signed long	32 bits	Entier signé	int32_t
unsigned long	32 bits	Entier non signé	uint32_t
__Int64	64 bits	Entier signé	int64
unsigned __Int64	64 bits	Entier non signé	uint64

L'utilisation de la norme MISRA nécessite l'inclusion du fichier <*stdint.h*>. Par défaut tous les types sont signés sauf **char** et **__bit**.

Type booleen

Le compilateur prend en charge le type `_Bool` ; une variable de ce type peut prendre les valeurs 0 (`false`) et 1 (`true`). Le fichier entête **`stdbool.h`** définit les macros `true` et `false` qui peuvent être utilisés avec le type `_Bool`. La macro `Bool` est substituée au type `_Bool`.

Exemple

```
#include <stdbool.h>
Bool  flag ;
```

Type float

Le compilateur MPLAB XC8 prend en charge les types à virgule flottante IEEE 754 32 bits et la forme tronquée 24 bits. Les tailles à virgule flottante de 32 bits seront automatiquement définies lorsque vous sélectionnez la conformité C99. Si le type à virgule flottante 24 bits est explicitement sélectionné avec l'option de compilation -*fshort-float*, le compilateur utilisera les bibliothèques C90. Le Tableau 7 montre les types de données et leurs tailles.

Tableau 7 : Type réel

Type	Taille en bits
`float`	24 / 32
`double`	24 / 32
`long double`	Même taille que double

Conversion de type (Casting)

Les expressions peuvent être explicitement converties à l'aide de l'opérateur de conversion "**`(type)`**". Dans tous les cas, la conversion d'un type à un autre doit être effectuée avec prudence et uniquement lorsque cela est absolument nécessaire.

Considérons l'exemple suivant :

```
uint32_t  k ;
uint16_t i ;
i = k ;
```

Dans cet exemple, un type *long* est assigné à un type *int*, le compilateur effectue implicitement la conversion de type.

Dans l'exemple suivant la conversion de type explicite est nécessaire.


```
int16_t  i, j;
float  fl;
fl = i/j;
```

Attention, seulement la partie entière du rapport est convertie en réel.

Si `i = 7` et `j = 2, fl =3.0`. Vous devez écrire `fl = (float)i/j ;`

Déclaration des registres SFRs

Le fichier d'en-tête spécifique du processeur est un fichier C qui contient des déclarations avec les qualificatifs `extern  volatile` pour les registres SFR. Nous donnons à titre d'exemple la déclaration du PORTD contenu dans le fichier en-tête pic18F4520.h

```
extern volatile unsigned char    PORTD   __at(0xF83);
#ifndef _LIB_BUILD
asm("PORTD equ 0F83h");
#endif
// bitfield definitions
typedef union {
    struct {
        unsigned RD0                    :1;
        unsigned RD1                    :1;
        unsigned RD2                    :1;
        unsigned RD3                    :1;
        unsigned RD4                    :1;
        unsigned RD5                    :1;
        unsigned RD6                    :1;
        unsigned RD7                    :1;
    };
    struct {
        unsigned PSP0                   :1;
        unsigned PSP1                   :1;
        unsigned PSP2                   :1;
        unsigned PSP3                   :1;
        unsigned PSP4                   :1;
        unsigned PSP5                   :1;
        unsigned PSP6                   :1;
        unsigned PSP7                   :1;
    };
    struct {
```

```
        unsigned                        :5;
        unsigned P1B                    :1;
        unsigned P1C                    :1;
        unsigned P1D                    :1;
    };
    struct {
        unsigned                        :7;
        unsigned SS2                    :1;
    };
} PORTDbits_t;
extern volatile PORTDbits_t   PORTDbits   __at(0xF83);
```

Les instructions suivantes illustres des exemples d'accès au registre du PORTD :

```
PORTD = 0x55 ;                  // PORTD ← 0x55
PORTDbits.RD2 = 1               // met le bit 2 du PORTD à 1
#define LED PORTDbits.RD2
LED = 1 ;                       // identique à PORTDbits.RD2 = 1
```

Fonction delay

Le compilateur MPLAB XC8 fournit un ensemble de pseudo-fonctions de retard. Ces fonctions créent des retards multiples du cycle d'instruction. Il est nécessaire donc de définir la fréquence d'horloge utilisée à travers le paramètre `_XTAL_FREQ`

Exemple

```
#define _XTATL_FREQ 8000000     // fréquence du quartz 8 MHz
```

`_delay(x)` : crée une attente de **x** cycles d'instruction ; le paramètre **x** est de type `unsigned long`. La valeur de **x** ne doit pas dépasser **50463240**.

`__delay_us(x)` : attente en microseconde

`__delay_ms(x)` : attente en milliseconde.

EXERCICES CORRIGES

Exercice 1

Ecrire un programme permettant de faire clignoter la LED connectée à la broche RC2 au rythme de 500ms allumée et 500ms éteinte.

Solution

Pour modifier l'état du bit RC2 sans modifier les autres bits, vous pouvez utiliser l'opérateur de masque OU exclusif (^) ou l'opérateur NON logique (!).

`PORTC = PORTC ^ 0x 04 ;` ou `PORTCbits.RC2 = !PORTCbits.RC2 ;`

```
#include <xc.h>
#define _XTAL_FREQ 8000000
#pragma  config  OSC = HS, WDT = OFF, LVP = OFF, PWRT = ON, DEBUG = ON
void main(){
     TRISC = 0xFB ;                             // RC2 en sortie
     while(1) {
        PORTCbits.RC2 = ! PORTCbits.RC2 ;       // Inverser RC2
        __delay_ms(500) ;
}
```

Exercice 2

Ecrire un programme permettant d'allumer la LED connectée à la broche RC2 selon la table de vérité suivante. Les interrupteurs de commande S1 et S2 sont connectés aux lignes RB0 et RB1.

S2	S1	LED
0	0	LED éteinte
0	1	LED allumée
1	0	LED clignote à une fréquence de 1 Hz
1	1	LED clignote à une fréquence de 0,5 Hz

Solution

Il est possible d'utiliser les structures conditionnelles **`if`** ou bien **`switch`**.

```
#include <xc.h>
#define _XTAL_FREQ 8000000
#pragma  config  OSC = HS, WDT = OFF, LVP = OFF, PWRT = ON
#define S1 PORTBbits.RB0
#define S2 PORTBbits.RB1
#define LED PORTCbits.RC2
void main(){
  ADCON1 = 0x0F ;       // configuration des PORTA, B et E en numérique
  TRISB = 0xFF ;        // PORTB en entrée
  TRISC = 0xFB ;        // RC2 en Sortie
  PORTC & = 0xFB ;      // LED initialement éteinte
  while(1)
  {     if((S2 == 0)&&(S1 == 0))
                LED = 0 ;
```

```
        if((S2 == 0)&&(S1 == 1))
                LED = 1 ;
        if((S2 == 1)&&(S1 == 0)){
                LED = !LED ;
                __delay_ms(1000) ;
        }
        if((S2 == 1)&&(S1 == 1)){
                LED = !LED ;
                __delay_ms(2000) ;
        }
 }
}
```

Exercice 3

1. Traduire en langage C l'algorithme suivant :

```
Début
   PORTC ← 0x81 ;
           Répéter toujours
           PORTC ← (PORTC << 1)|(PORTC >> 7));
           Attente_1000ms;
   Fin répéter
Fin
```

2. Dresser dans un tableau, les états possibles du PORTC
3. Quelle fonction réalise l'instruction suivante :

```
PORTC ← (PORTC << 1)|(PORTC >> 7));
```

Solution

1. Traduction de l'algorithme en langage C

```
void main(){
     PORTC = 0x81 ;
     while(1) {
        PORTC = (PORTC << 1)|(PORTC >> 7) ;
        __delay_ms(1000) ;
    }
}
```

2. Etats possibles du PORTC

RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	0

0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1

3. Rotation à gauche du PORTC

Exercice 4

Soit le schéma de la Figure 17 :

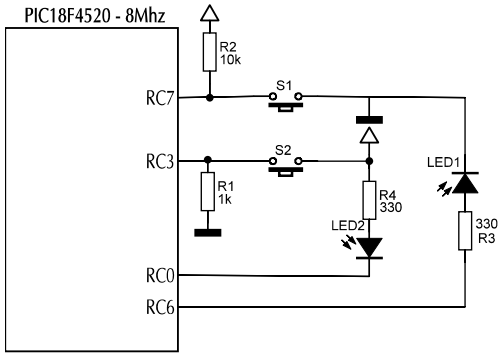
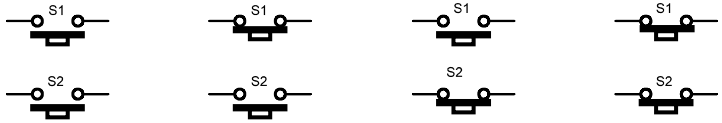


Figure 17

1. Quel est le rôle des résistances R1et R2 ?
2. Justifier les valeurs des résistances insérées en série avec les LEDs.
3. Donner le mot de commande à charger dans le registre TRISC. Les broches non utilisées seront considérées comme des entrées.
4. Pour lire l'état des boutons S1 et S2, nous vous proposons l'instruction suivante :

```
BPs = PORTC & 0x88 ;
```

Donner pour les cas suivants les valeurs possibles de BPs en Hexadécimal.



5. En utilisant l'instruction à choix multiples `switch … case` , écrire un programme en langage XC8 permettant d'allumer une LED lorsque le bouton correspondant est enfoncé.

Solution

1. **R1** : Résistance de tirage vers le bas (pull-down)

   R2 : Résistance de tirage vers le haut (pull-up).

2. Une LED conduit lorsqu'elle est polarisée en direct. Un courant de 10 mA ou plus est nécessaire pour avoir une luminosité acceptable. La tension de seuil de la LED lorsqu'elle est polarisée en direct peut aller d'environ 1,6 V à plus de 2,2 V, tandis que la tension à la sortie d'une broche V_{OH} (Output High Voltage) est environ 5V. Une résistance de limitation du courant donc est nécessaire. En appliquant la loi des mailles : $V_{OH} = R \cdot I + V_o$

V_{OH} : Niveau haut de la tension de sortie de la broche du microcontrôleur.

V_o : Tension de seuil de la LED.

D'où $R = \frac{V_{OH} - V_o}{I} = \frac{5-1{,}6}{10 \cdot 10^{-3}} = 340\Omega$

3. `TRISC = 0b10111110 = 0xBE`
4. `a) BPs = 0x80  b) BPs = 0x00  c) BPs = 0x88 d) BPs = 0x08`
5. Programme de commande des LEDs

```
#include <xc.h>
#define _XTAL_FREQ 8000000
#pragma  config  OSC = HS, WDT = OFF, LVP = OFF, PWRT = ON
char BPs ;
#define LED1  PORTCbits.RC6
#define LED2  PORTCbits.RC0
void main(){
   TRISC = 0xBE ;
   while(1) {
      BPs = PORTC & 0x88 ;
      switch(BPs) {
        case    0x80 : {LED1 = 0 ; LED2 = 1 ;} break ;
        case    0x00 : {LED1 = 1 ; LED2 = 1 ;} break ;
        case    0x88 : {LED1 = 0 ; LED2 = 0 ;} break ;
        case    0x08 : {LED1 = 1 ; LED2 = 0 ;} break ;
      }
   }
}
```