ISIMA 2ème année – Filières 1 et 2

Systèmes Embarqués

E. Mesnard

Sommaire

I Introduction sur les Systèmes Embarqués

II Les outils de développement

III Interaction avec l'environnement : les transfert de données

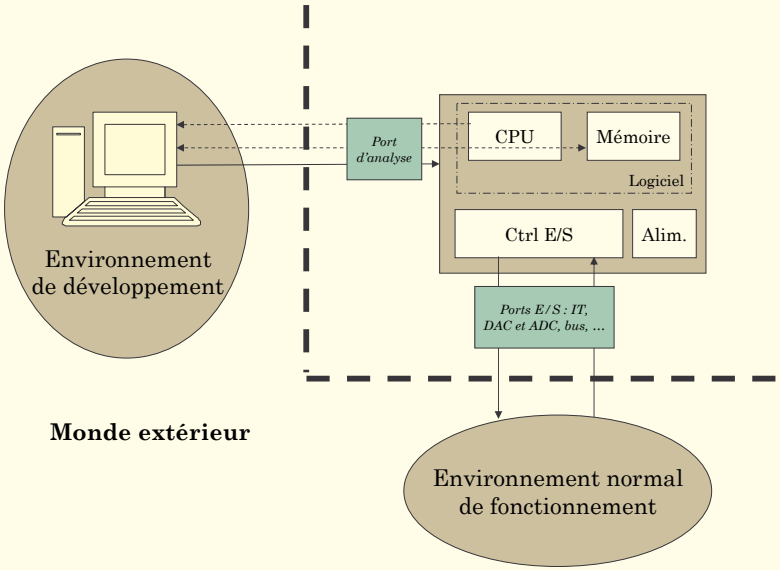
IV Interaction avec l'environnement : les interruptions

V Temps Réel

I-1 Définition(s) d'un Système Embarqué (SE)

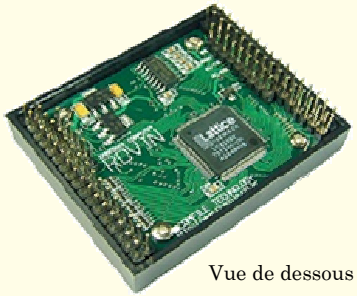
- Système électronique, piloté par un **logiciel**, qui est complètement intégré au système qu'il contrôle `[Wikipédia]`.
- Système électronique et **informatique** autonome, invisible en tant que tel, mais intégré dans un équipement doté d'une autre fonction, dédié à la résolution d'un problème spécifique.
- Système devant répondre de manière déterministe à des événements, avec la garantie de ne pas « crasher ».
- Le SE devient SETR (SE Temps Réel) quand :
  - l'information après acquisition et traitement est encore pertinente
  - pour une information arrivant de façon périodique, le temps d'acquisition/traitement doit être inférieur à la période de rafraîchissement de cette information.
- Le premier système « moderne » embarqué a été le système de guidage de la mission lunaire Apollo (AGC), développé au MIT.

I-2 Architecture typique des SE : vue générale



I-3 Le « processeur » ROVIN RV1000

Le ROVIN est développé par la société ComFile Technologies



Vue de dessous

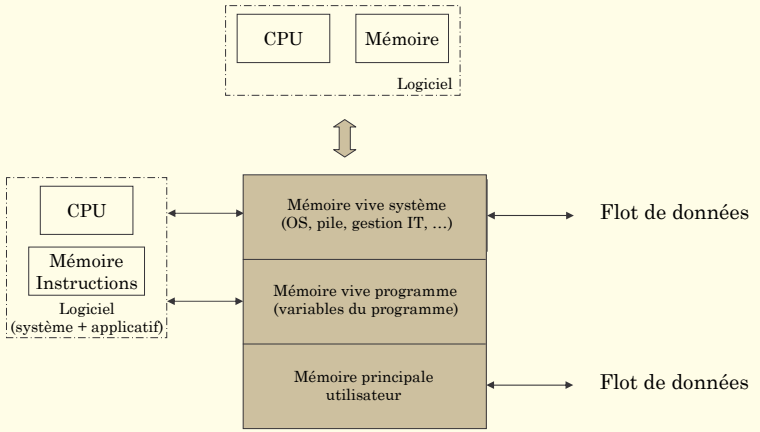
C'est un petit système embarqué à lui tout seul

Il est architecturé autour d'un processeur ARM 7 TDMI

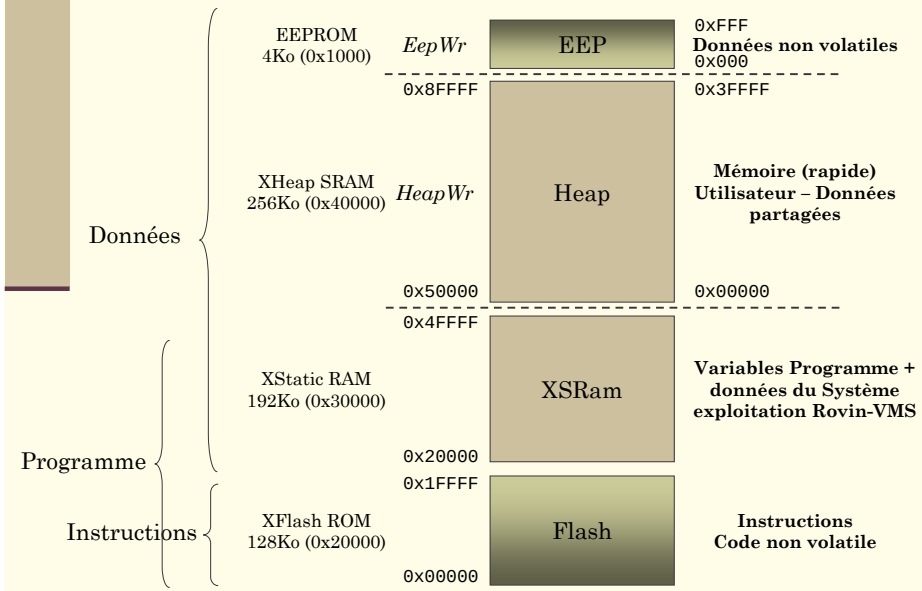
Afin de « jouer avec », il est installé sur une carte de test : la QuickStart Board

I-3 Architecture globale du ROVIN

- Processeur ARM7 TDMI (Advanced RISC Machines 7)
- Architecture de type « Harvard »
  Mémoires d'instructions (programme compilé) et mémoires de données séparées



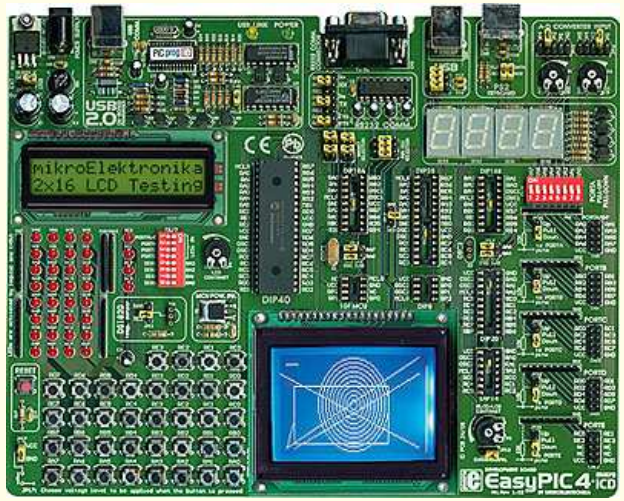
I-3 Architecture mémoire du ROVIN



I-4 Le processeur PIC et la carte EasyPIC

Le PIC est un micro contrôleur développé par la société MicroChip

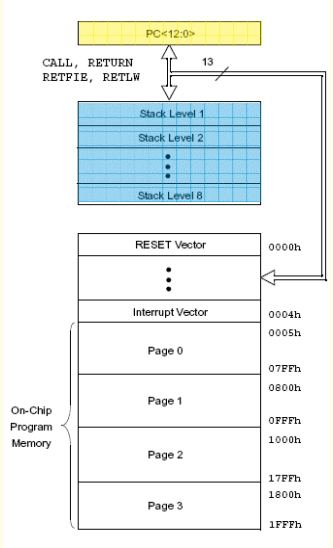
La carte de test EasyPIC, développée par mikroElectronika, contient un PIC16F877A



I-4 Architecture du PIC16F877 – particularités

- Processeur RISC, en architecture Harvard, en VLIW (Very Long Word Instructions)
- Core
  - Oscillateur interne et logique de démarrage
  - CPU (Central Processing Unit) + ALU (Arithmetic Logical Unit)
  - Gestion des interruptions
  - Unité de contrôle sous la forme d'un pipeline d'instructions
- Ports d'E/S
  - Broches d'usage générales
  - Port série (basique « Synchronous Serial Port » SSP, et maître « Master » MSSP)
  - Port parallèle (Parallel Slave Port (PSP)
- Convertisseurs
  - Conversion série / Parallèle USART (SCI)
  - 3 Convertisseurs Analogique / Numérique (résolution 8 bits)
  - 1 Convertisseur Analogique / Numérique (résolution 10 bits)
  - Capture, Comparaison et PWM (CCP)
- 3 timers
- Mémoire d'instructions EPROM 8K mots de 14 bits
- Mémoire de données RAM (bancs de registres) de 368 octets
- Mémoire de données EEPROM de 256 octets

I-4 Architecture mémoire de l'EasyPIC



- PC(12:0)

  Program Counter, pointeur d'instructions permettant d'atteindre les 8K
- Stack Level 1 à 8 :

  Pointeur de pile sur 8 niveaux pour les appels de fonctions
- On-Chip Program Memory Flash

  Mémoire d'instructions EPROM 8K mots de 14 bits
  - Accessible entre `0x0000` et `0x1FFF`
  - Vecteur RESET en `0x0000`
  - Vecteur Interruption en `0x0004`
- Data Memory :

  Mémoire de données RAM (bancs de registres) de 368 octets
- Data EEPROM :

  Mémoire de données EEPROM de 256 octets

I-4 Architecture mémoire de données « Data Memory » de l'EasyPIC

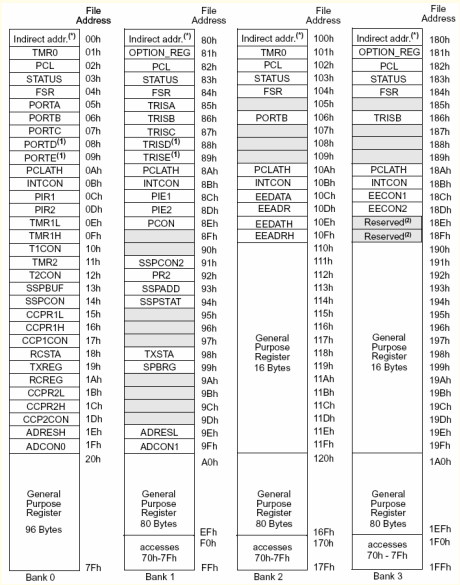
- La « data memory » est découpée en 4 banks.
- Une bank est sélectionnée par les bits RP1 (STATUS(6)) et RP0 (STATUS(5)).
- Les banks contiennent :
  - GPR (« General Purpose Registers ») :

    Les GPR contiennent les variables locales.

    Les noms donnés aux « Registers » sont proches de ceux utilisés dans le source, et au pire, ils sont du type STACK_N.
  - SFR (« Special Function Registers ») :

    Ce sont les registres systèmes du processeurs.

    Les SFR les plus utilisées sont dupliquées dans les autres bank, réduisant ainsi la taille de code et les temps d'accès.
- Les SFR les plus courantes sont :
  - PCL (Program Counter) : pointeur d'instructions (compteur programme)
  - STATUS : registre d'état
  - INTCON : configuration des interruptions
  - TRISA, TRISB,… : configuration des ports entrées/sorties
  - PORTA, … : valeurs des entrées/sorties
  - PIE1 : configuration individuelle des interruptions



Sommaire

I Introduction sur les Systèmes Embarqués

II Les outils de développement

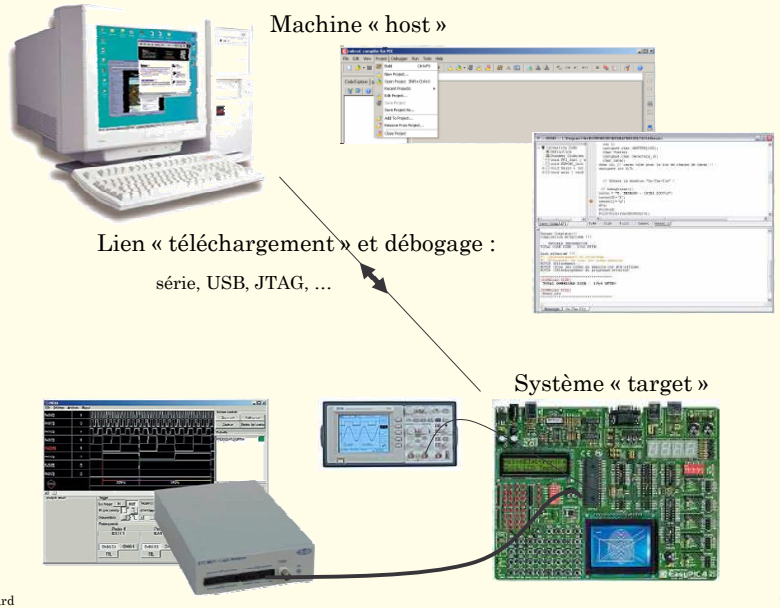
1 Les chaînes logicielles : Environnement de développement croisé
2 Environnement de développement ROVIN
3 Environnement de développement EasyPIC

III Interaction avec l'environnement : les transfert de données

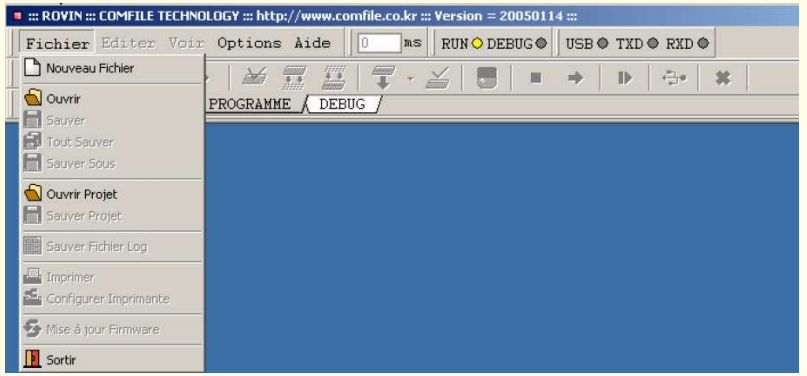
IV Interaction avec l'environnement : les interruptions

V Temps Réel

II-1 Les chaînes logicielles : Environnement de développement **croisé**

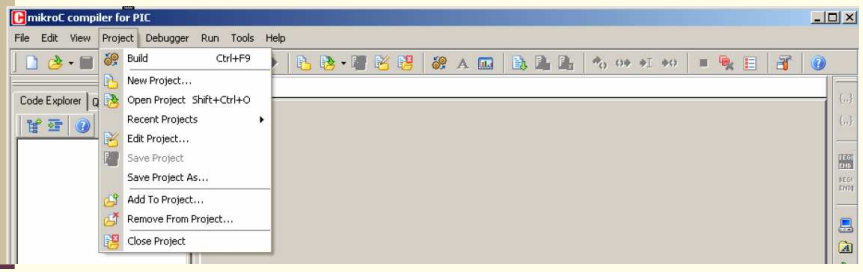


II-2 ROVIN : Outil de développement (ROVIN – IDE)



- Trois possibilités après avoir lancé ROVIN-IDE :
  - « Nouveau Fichier » : Créer un nouveau fichier C : implicitement, créer un nouveau **projet**
  - « Ouvrir » : Ouvrir un fichier C existant : modifier ou créer un **projet**
  - « Ouvrir Projet » : Ouvrir un projet existant (extension « .dwn »)

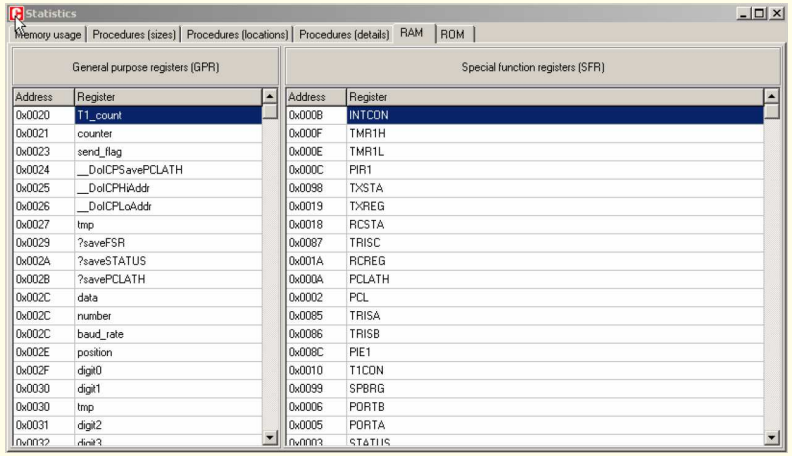
II-3 EasyPIC : Outil de développement (mikroC)



- Deux possibilités après avoir lancé mikroC :
  - « New Project… » : Créer un nouveau **projet** (extension « .ppc »)
  - « Open Project » : Ouvrir un projet existant : modifier un projet
- Démarche en salle de TP :
  - Configurer le projet (« New Project… » avec paramètres PIC16F877A)
  - Ecrire le code ou recopier un code source existant
  - Construire le projet. Cette phase de compilation (« Build ») permet la création du code assembleur (« .asm ») et du fichier de configuration (« .hex ») nécessaire à la programmation du microcontrôleur.
  - Programmer le microcontrôleur

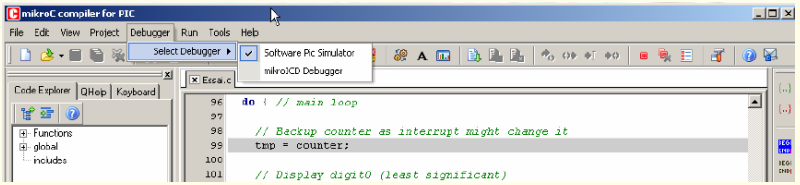
II-3 EasyPIC : statistiques occupation mémoire RAM

- Détail d'occupation de la mémoire des données « Data Memory » :
  - GPR (« General Purpose Registers »)
  - SFR (« Special Function Registers »)



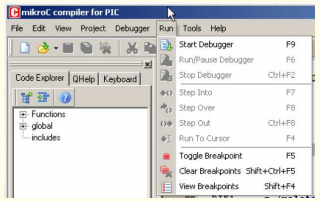
II-3 EasyPIC : Déboguage et mise au point avec MikroC

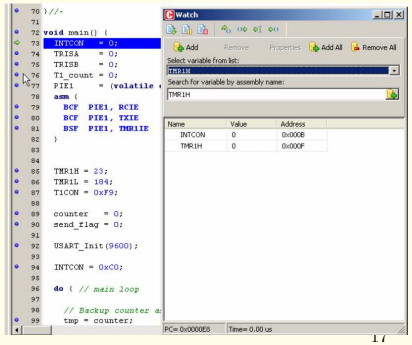
- 2 modes de déboguage (choix dans « Debugger – Select Debugger ») :
  - Simulateur « Software Pic Simulator » : pas d'implémentation sur le PIC
  - Emulateur « mikroICD Debugger »



Le démarrage du débogueur (« Start Debugger ») lance automatiquement :

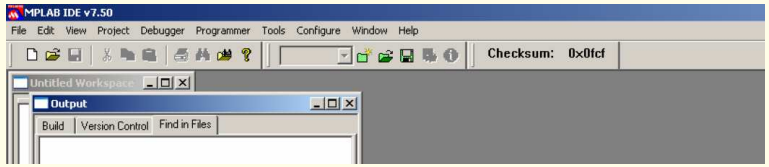
- Le mode pas à pas sur l'assembleur et sur le code C
- la fenêtre WATCH (où il est possible de modifier la valeur des variables)





II-3 EasyPIC : Déboguage et mise au point avec MPLAB

- Autre logiciel pour le déboguage : MPLAB (MicroChip)
  - Simulation à partir d'un fichier HEX (version gratuite)
  - Compilation et émulation (version payante)



- Principe d'utilisation de MPLAB
  - Importer le fichier Hex issu de la compilation (« File – Import »)
  - Sélectionner le simulateur (« Debugger – Select Tool – MPLAB SIM »)
  - Visualiser les grandeurs utiles (« View ») :
    - EEPROM,
    - Registres fichier,
    - pointeur de pile,
    - variables locales,
    - mémoire programme,
    - Registres spéciaux (SFR), …
- Intérêt : analyser le code **assembleur** !

II-3 EasyPIC : Format des instructions PIC16F877A

Pour analyser l'assembleur, il faut d'abord comprendre le format des instructions :

Byte-oriented file register operations

13 … 8	7	6 … 0
OPCODE	d	f (FILE #)

d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Bit-oriented file register operations

13 … 10	9 … 7	6 … 0
OPCODE	b (BIT #)	f (FILE #)

b = 3-bit bit address
f = 7-bit file register address

Literal and control operations

General

13 … 8	7 … 0
OPCODE	k (literal)

k = 8-bit immediate value

CALL and GOTO instructions only

13 … 11	10 … 0
OPCODE	k (literal)

k = 11-bit immediate value

- 2 registres manipulés :
  - W : « Working register » est l'accumulateur
  - f : « file register » adresse d'un registre dans le banc de registres (de 0x00 à 0x7F sur les 4 banks)
- 3 formats d'instructions :
  - Byte-oriented operations : manipulation d'octets à partir de registres
    source = f,
    destination = f ou W, selon d
  - Bit-oriented operations : instructions manipulant des bits
    b = numéro de bit concerné,
    f = registre manipulé
  - Literal and control operations : instructions en adressage immédiat et instructions de sauts.
    k est la valeur immédiate (« literal »)
- 3 modes d'adressage :
  - Adressage immédiat (instructions « literal »)
  - Adressage direct (instructions avec f)
  - Adressage indirect (instructions avec INDF)

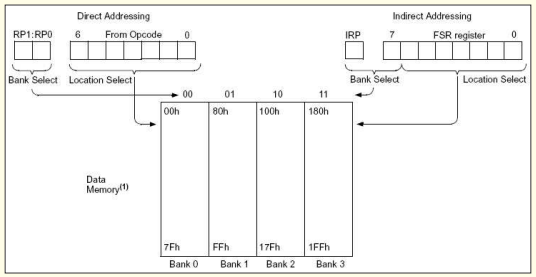
II-3 EasyPIC : Jeu d'instructions du PIC16F877A

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 1fff 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 1fff 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	$\overline{TO}$,$\overline{PD}$	
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	$\overline{TO}$,$\overline{PD}$	
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	

Soit seulement 35 instructions…

II-3 EasyPIC : modes d'adressage sur PIC16877A

- Immédiat (« Literal ») :
  - Movlw 0x22 : W ← 0x22
  - Movlw mavariable : W ← @mavariable (adresse de mavariable et pas la valeur)
- Direct :
  - Movwf mavariable : mavariable ← W
  - Movf 0x33,0 : W ← [0x33], mais … Movf 0x44,1 : 0x44 ← [0x44] (≠NOP : test flag Z)
- Indirect :
  - Ce mode d'adressage s'appuie sur 3 registres : INDF (0x00), SFR (0x04) ainsi que le bit IRP du registre STATUS (0x03)
  - INDF n'est pas un « vrai » registre ! Toute instruction qui utilise ce registre accèdera en fait au registre pointé par SFR, dans une bank de registres. IRP est utilisé en complément de FSR(7) pour sélectionner une des 4 banks.
  - Movf INDF,0 : W ← [[IRP,FSR]]



Sommaire

I Introduction sur les Systèmes Embarqués

II Les outils de développement

III Interaction avec l'environnement : les transfert de données

1 Les échanges analogiques : convertisseurs ADC et DAC
2 Les échanges numériques : la représentation des données
3 Les communications par bus OneWire
4 Les communications par bus I2C
5 Les communications par bus SPI

IV Interaction avec l'environnement : les interruptions

V Temps Réel

III-1 Les principaux moyens d'interactions du SE

- Ports d'E/S analogiques et numériques directs (sans protocole)
  - Données analogiques via des convertisseurs : ADC et DAC
  - Données numériques (bits, octets,…) directement sur les broches du composant
- Port UART (Universal Asynchronous Receiver Transmitter)
  - Interface de connexion série (données parallèles vers liaison série)
  - En émission, l'UART sérialise les données à l'aide d'un registre à décalage. Elles sont transmises bit après bit sur la liaison série (en bits/seconde).
  - En réception l'UART réalise l'opération contraire. Il réceptionne et stocke les données reçues en série, avant de les restituer sous forme parallèle.
  - Exemple : afficheur LCD, terminal PC « HyperTerminal », …
- Port OneWire
  - Port de communication simplissime (Dallas Semiconductor) : 1 seul fil !
  - Exemples : capteur de température, RAM 4Ko

III-1 Les principaux moyens d'interactions du SE

- Port I2C (Inter Ic Control)
  - Port de communication sur 2 fils (Philips)
  - Exemples : mémoire I2C, horloge temps réel, DAC et ADC
- Port SPI (Serial Peripheral Interface)
  - Port de communication série sur 4 fils (Motorola – FreeScale).
  - Exemple : mémoire SPI
- Port MicroWire
  - Port proche du SPI (trame étendue à plusieurs octets)
- Ports réseaux embarqués :
  - UDF
  - TCP/IP,…

- Convertisseur Analogique/Numérique (CAN ou ADC)

$$\text{Nombre} = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_0 2^0$$

MSb : a_{n-1} LSb : a_0

MSb (ou msb) « Most Significant bit » (≠MSB « Byte »)

LSb (ou lsb) : « Least Significant bit », quantum – variation analogique provoquant le « +1 » dans la représentation numérique.

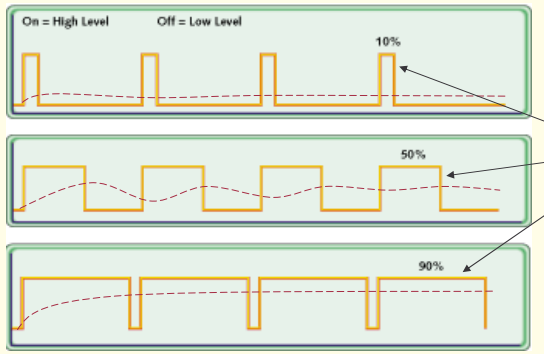
Exemple Rovin :

```
AdcSet(ADC0,ADCDF_128);
AdcOn();
while(1) {
    ADC_DATA = AdcRead(); // Lecture de la valeur analogique
    // Affichage de la mesure dans la fenetre 'ON-The-Fly' du mode Debug
    DebugClear();
    DebugPrint("Valeur analogique presente sur ADC0 : ");
    DebugDOUBLE((ADC_DATA*5)/pow(2,10),DEC); // petit calcul au vu de
    DebugPrint(" Vcc.");                     // la resolution choisie
}
```


- Convertisseur Numérique/Analogique (DAC)
  - PWM (« Pulse Width Modulation ») – MLI (Modulation de Largeur d'Impulsions)
  - Basé sur un compteur n bits ET sur un « Timer »
  - Rapport cyclique : « Duty Cycle »

Exemple Rovin :

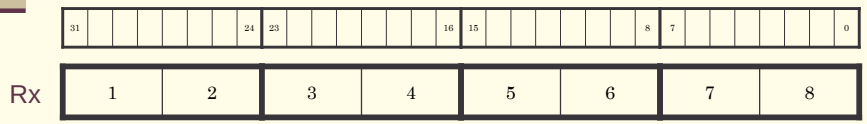
```
Pwm1_Set(PWM_16BIT_FAST,PWM_FREQ_1);
Pwm1_Duty(PWM_CHA, ((float)0xFFFF*10)/100); //Rapport 10%
Pwm1_Duty(PWM_CHB, ((float)0xFFFF*50)/100); //Rapport 50%
Pwm1_Duty(PWM_CHC, ((float)0xFFFF*90)/100); //Rapport 90%
Pwm1_AllOn(); // Active tous les canaux en même temps
```



« Duty Cycle » : ratio des cycles actifs sur la période

Valeur moyenne en sortie : Filtre passe-bas

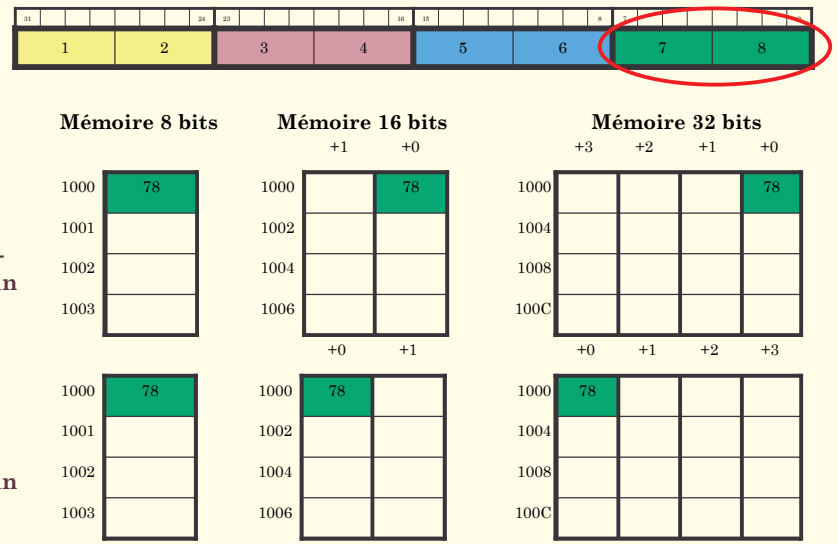
- La représentation des données se fait généralement sur plusieurs octets.
- Problématique : **ordre** ?
  - Ordre dans lequel ces octets sont organisés en mémoire
  - Ordre dans lequel ces octets sont émis lors d'une communication (échange)
- « Endianness » (boutisme)
  - Little-Endian (orientation petit-boutiste) : mot de poids faible en tête
  - Big-Endian (orientation gros-boutiste) : mot de poids fort en tête
- Exemple :
  - Registre d'un processeur 32 bits : Rx



- Comparaison de 3 « Move » sur ce registre :

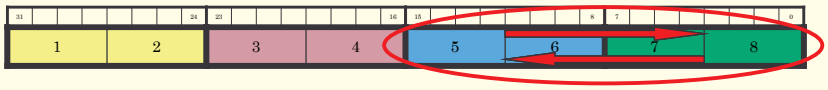
```
Move.B Rx, 0x1000;   // Byte, donc octet, soit 8 bits
Move.W Rx, 0x1000;   // Word, donc 2 octets, soit 16 bits
Move.L Rx, 0x1000;   // Long word, donc 32 bits
```

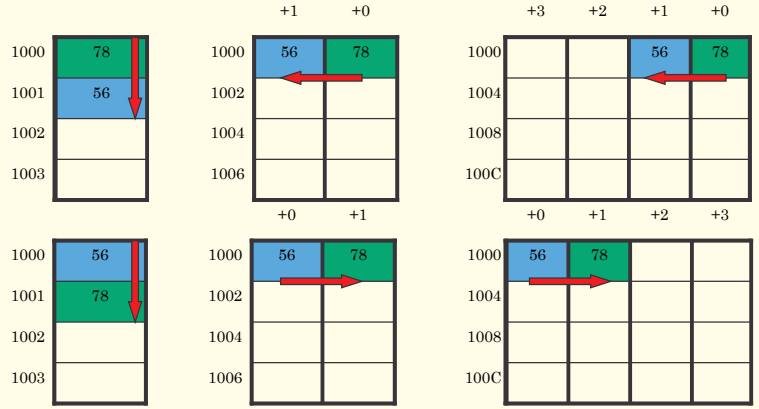

Move.B Rx, 0x1000



III-2 Les échanges numériques : Endianness lors de transferts 16 bits

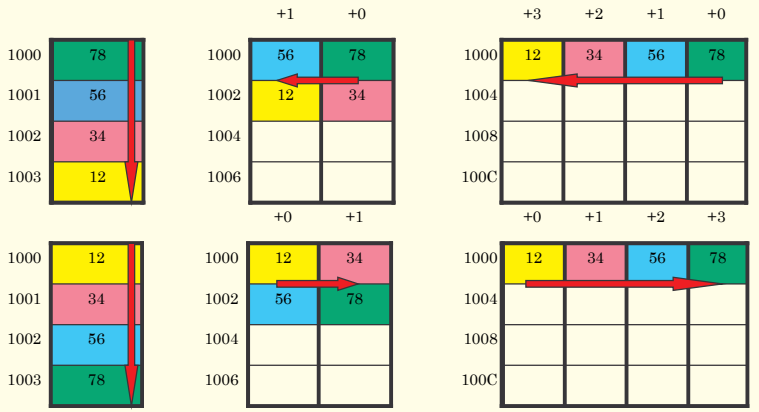
```
Move.W  Rx, 0x1000
```





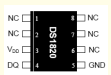
III-2 Les échanges numériques : Endianness lors de transferts 32 bits

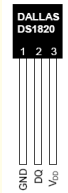
```
Move.L  Rx, 0x1000
```

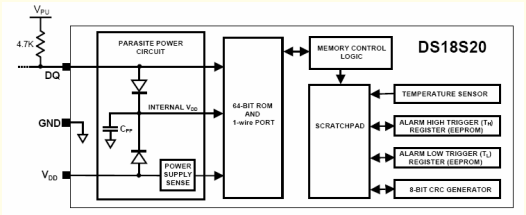


III-3 Les communications par bus OneWire

- Bus OneWire (« 1-Wire »), développé par Dallas Semiconductor
- Fonctionnement sur 1 fil (DQ)…
- Très lent : 16 kbps, mais éventuellement très long : jusqu'à 300 mètres !
- Composants OneWire orientés capteurs (température) et petites RAM ou ROM.
  - Exemple : thermomètre numérique DS1820







- Possibilité d'avoir plusieurs composants OneWire sur le même bus.
- Ni en série, ni en parallèle car il n'y a qu'un fil !
- Tous ces composants OneWire sont alors esclaves sur le bus.
- « ROM Code 64 bits » : 8-bit device type, 48-bit serial number, 8-bit CRC
  Numéro de série unique à chaque composant

III-3 Les communications par bus OneWire – Séquence de transaction

Séquence de transaction en 3 temps, généré par le maître du bus (processeur) :

- Etape 1 : Initialisation
  Série d'impulsions sur le bus :
  impulsion « Reset » émise par le maître suivie d'impulsions « Presence » du ou des esclaves
- Etape 2 : Adressage des composants : « Commande ROM »
  - SEARCH ROM [F0h] : pour identifier tous les ROM Codes (bit après bit, en lecture ET logique sur le bus, de la valeur directe suivi de la valeur complémentée)
  - READ ROM [33h] : lecture du ROM Code si un seul esclave
  - MATCH ROM [55h] : adressage à l'esclave dont le ROM Code est donné en paramètre.
  - SKIP ROM [CCh] : diffusion à tous les esclaves (aucun ROM Code passé en paramètre)
  - ALARM SEARCH [ECh] : même principe que SEARCH ROM, par contre, seuls les composants en alarme répondent.
- Etape 3 : Commande d'une fonction spécifique au composant (avec échange de données)
  Les fonctions disponibles ici dépendent du composant.
  Exemples de telles fonctions :
  - WRITE SCRATCHPAD [4Eh] : écriture dans un registre interne "Scratchpad"
  - READ SCRATCHPAD [BEh] : lecture du registre interne "Scratchpad"
  - WRITE MEMORY [0Fh] : écriture mémoire
  - READ MEMORY [F0h] : lecture mémoire
  - WRITE STATUS [55h] : écriture dans le registre d'état
  - READ STATUS [AAh] : lecture du registre d'état
  - …

III-3 EasyPIC : Les communications par bus OneWire

- Exemple : bibliothèque « OneWire » sur EasyPIC

- Ow_Reset :

```
unsigned short Ow_Reset(unsigned short *port, unsigned short pin);
```

  - Retourne 0 si un composant OneWire est présent, et 1 sinon
  - Les paramètres Port et Pin indiquent sur quelle broche le composant est connecté
  - Exemple : composant branché sur broche 5 du port PORTA :

```
Ow_Reset(&PORTA, 5);
```

- Ow_Read :

```
unsigned short Ow_Read(unsigned short *port, unsigned short pin);
```

  - Retourne un octet de données lu sur le bus OneWire

- Ow_Write :

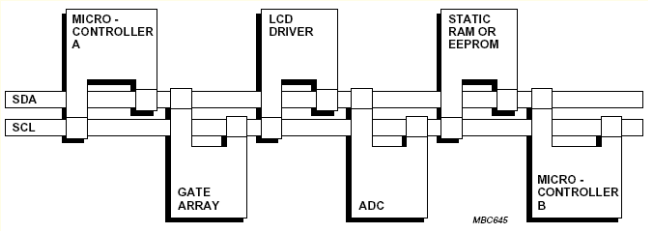
```
void Ow_Write(unsigned short *port, unsigned short pin, unsigned short par);
```

  - Ecrit un octet de données sur le bus OneWire
  - Exemple, envoi de la commande SKIP ROM :

```
Ow_Write(&PORTA, 5, 0xCC);
```

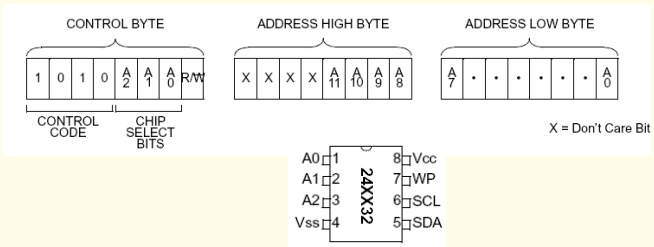

III-4 Les communications par bus I2C

- Bus I2C « 2 wired » ou « 2-wire » (Philips Semiconductors), sur 2 fils
  - SDA : Data, ligne de données E/S entre maître et esclaves
  - SCL : Clock, ligne d'horloge, du maître vers les esclaves
- Bus liaison série synchrone, débit : 400 Kb/s (lent), sur une distance de quelques centimètres (plutôt court)
- Bus multi-maître
- Esclaves de toute nature : esclaves tels que, eeprom, horloge temps réel, ports d'entrées / sorties, afficheurs,…
- Protocole très simple, mais en « half duplex » (émission et réception non simultanées)
- Dédié pour établir une communication inter-composant ou inter-carte



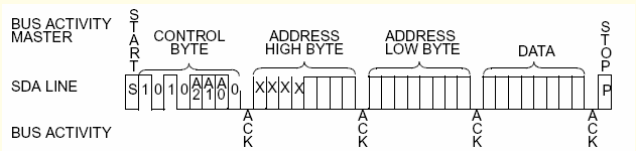
III-4 Les communications par bus I2C – principe d'adressage

Commande d'adressage d'une puce :



A(2:0) : bits de sélection – jusqu'à 8 puces sur le même bus

Exemple : écriture d'un octet sur le bus



III-4 ROVIN : Création du bus I2C et signaux de contrôle

- I2cCreate : déclaration et création du port I2C

```
I2cCreate (unsigned char pI2Cid,      // numéro de port I2C à créer (0 à 15)
    unsigned char pUSEport,           // port d'E/S du composant à utiliser
    unsigned char pSDApin,            // broche du port utilisée en signal SDA
    unsigned char pSCLpin,            // broche du port utilisée en signal SCL
    unsigned char pOUTmode,           // Endianess : I2C_MSBFIRST ou LSB
    unsigned int pSLOWtime);          // Vitesse de communication
```

Exemple :

```
I2cCreate(      2,PA,     // Création du port I2C numéro 2, sur port PA
                5,        // SDA est la broche 5 du port PA
                4,        // SCL est la broche 4 du port PA
                I2C_MSBFIRST // Transfert Big-Endian
                0);       // Transfert en vitesse I2C maximale
```

- I2cStart : créer une condition START sur le port I2C (à utiliser avant chaque début de paquet)

```
I2cStart(unsigned char pI2Cid)
```

- I2cStop : créer une condition STOP sur le port I2C (à utiliser à la fin de chaque paquet)

```
I2cStop(unsigned char pI2Cid)
```

- I2cAck : retourne l'acquittement (True=ACK_reçu) ou non (False=TimeOut) en provenance du composant esclave adressé sur le port I2C.

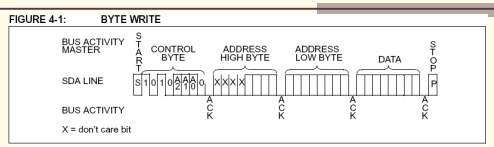
```
unsigned char I2cAck(unsigned char pI2Cid, unsigned int pTimeOut)
```

III-4 ROVIN : Utilisation du bus I2C – lecture et écriture de données

- I2cOut : écriture d'un octet pData sur le port I2C
  I2cOut(unsigned char pI2Cid, unsigned char pData)
- I2cBitOut : permet d'envoyer de 1 à 64 bits sur le bus I2C
  I2cBitOut(unsigned char pI2Cid, unsigned char pBITsize, unsigned char pData)
  Exemple : envoi de 10 bits sur le port numéro 2 :
  `I2cBitOut(2,10,0b1010110011);`
- I2cIn : lecture et retour d'un octet en provenance du port I2C
  I2cIn(unsigned char pI2Cid)
- I2cBitIn : lecture de 1 à 64 bits depuis le bus I2C
  I2cBitIn(unsigned char pI2Cid, unsigned char pBITsize)
  Exemple : lecture de 64 bits
  ```
  unsigned long I2CData;
  I2CData = I2cBitIn(2,64);
  ```

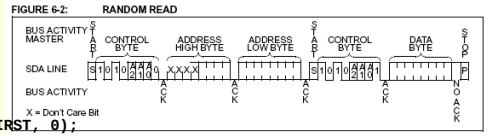

III-4 ROVIN : Exemple d'écriture dans une mémoire I2C 24C32

Port I2C numéro 0
SCL = PA.4 et SDA = PA.5




```
void main(void)
{
   unsigned int Adresse;
   unsigned short int Mot;

   // Création du port I2C
   I2cCreate(0, PA,         // Création port I2C n° 0, en utilisant la sortie XPPI-PA
             5,             // SDA sera la broche PA.5
             4,             // SCL sera la broche PA.4
             I2C_MSBFIRST, // Big-Endian : Sortie du MSB en premier
             0);            // Vitesse de communication I2C maximale


   // Cycle d'écriture « Byte Write » dans la mémoire EEprom 24LC32
   // -------------------------------------------------------------

   // Pour ce test, on désire écrire  0x0F à l'adresse 0x02F1
   Adresse=0x02F1;
   Mot=0x0F;

   I2cStart(0);                 // Etape 1 : START

   I2cOut(0,0b10100000);        // Etape 2 : CONTROL BYTE
   if(!I2cAck(0,0xff)) error(); // Attente de ACK avec Time Out de 32 ms

   I2cOut(0, Adresse >>8);      // Etape 3 : ADRESS HIGH BYTE
   if(!I2cAck(0,0xff)) error();

   I2cOut(0, Adresse);          // Etape 4 : ADRESS LOW BYTE
   if(!I2cAck(0,0xff)) error();

   // A cet instant, la mémoire est adressée. Il ne reste plus qu'a envoyer
   // la donnée pour finaliser l'écriture

   I2cOut(0,Mot);               // Etape 5 : DATA
   if(!I2cAck(0,0xff)) error();

   I2cStop(0);                  // Etape 6 : STOP
}
```

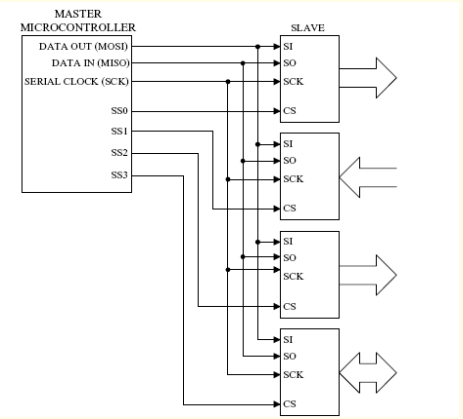

III-4 ROVIN : Exemple de lecture dans une mémoire I2C 24C32




```
void main(void)
{
   unsigned int Adresse;
   unsigned short int Mot;

   // Création du port I2C
   I2cCreate(0, PA, 5, 4, I2C_MSBFIRST, 0);

   // Cycle de lecture « Random Read » dans la mémoire EEprom 24LC32
   // --------------------------------------------------------------

   // Pour ce test, on désire lire un mot à l'adresse 0x02F1
   Adresse=0x02F1;
   Mot=0x00;

   I2cStart(0);                 // Etape 1 : START

   I2cOut(0,0b10100000);        // Etape 2 : CONTROL BYTE , comme un Write (bit0 =0)
   if(!I2cAck(0,0xff)) error(); // Attente de ACK avec Time Out de 32 ms

   I2cOut(0, Adresse >>8);      // Etape 3 : ADRESS HIGH BYTE
   if(!I2cAck(0,0xff)) error();

   I2cOut(0, Adresse);          // Etape 4 : ADRESS LOW BYTE
   if(!I2cAck(0,0xff)) error();

   // A cet instant, la mémoire est adressée. Il ne reste plus qu'à lire la donnée

   I2cStart(0);                 // Etape 5 : START

   I2cOut(0,0b10100001);        // Etape 6 : CONTROL BYTE, avec un Read (bit0 =1)
   if(!I2cAck(0,0xff)) error();

   Mot=I2cIn(0);                // Etape 7 : DATA
   I2cAck(0,0);                 // Etape 7 bis : NO ACK !

   I2cStop(0);                  // Etape 8 : STOP
}
```


III-5 Les communications par bus SPI

- Bus SPI (Motorola – Freescale), sur 4 fils
  - SCLK : Clock, ligne d'horloge, du maître vers les esclaves (jusqu'à 500Khz)
  - MOSI : Master data Output, Slave data Input, envoi des données
  - MISO : Master data Input, Slave data Output, lecture des données
  - SS : Slave Select : 1 signal par esclave (servant d'adresse)
- Bus liaison série synchrone, débit : 10 Mb/s (rapide), sur une distance de quelques centimètres (plutôt court)
- Bus à accès maître / esclave, en mode « full duplex » (émission / réception simultanée)
- Protocole simple, sans contrôle de flux, ni confirmation de réception des données.



III-5 EasyPIC : Utilisation du bus SPI – lecture et écriture de données

- Spi_Read

  unsigned short Spi_Read(unsigned short buffer);

  Fournit l'horloge par l'envoi des données du buffer, et reçoit les données correspondante à la fin du transfert.
- Spi_Write

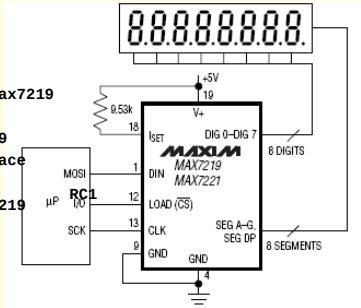
  void Spi_Write(unsigned short data);

  Provoque la transmission de l'octet de données data.
- Exemple :

  Contrôleur afficheurs 8 segments Max7219

  Broche CS (Chip Select) du max7219 sur broche RC1 (bit 1 de PORTC) du PIC16F877A

```
unsigned short i;
void main() {
  Spi_Init();
  TRISC &= 0xFD;
  max7219_init1();          // Initialize  max7219
  for (i = 1; i <= 8u; i++) {
    PORTC &= 0xFD;          // Select max7219
    Spi_Write(i);           // Send digit place
    Spi_Write(8 - i);       // Send digit
    PORTC |= 2;             // Deselect max7219
  }
}
```



Sommaire

I Introduction sur les Systèmes Embarqués

II Les outils de développement

III Interaction avec l'environnement : les transfert de données

IV Interaction avec l'environnement : les interruptions

1 Principe de fonctionnement des interruptions
2 Les IT selon ROVIN
3 Exemples ROVIN : IT externe, comparateur, capture, timer, alarme et RS232
4 Les IT selon EasyPIC
5 Exemples EasyPIC : IT timer

V Temps Réel

IV-1 Principe d'utilisation des interruptions

- Les interruptions (IT) sont généralement liées à des événements.
- Les événements peuvent survenir à tout instant : ils sont totalement asynchrones (non liés à l'horloge interne).
- Les IT interrompent le déroulement du programme (fil d'exécution) en cours d'exécution…
- … uniquement si le processeur l'autorise.

- Utilité de ce mécanisme d'IT :
  - « classique » : échange d'informations avec l'extérieur (Utilisateur, Périphériques)
  - « avancée » : couche basse du système d'exploitation (changement de tâche dans un mode de fonctionnement multi-tâches)

- Les interruptions permettent de libérer le processeur en attendant qu'un périphérique ne se signale
  - Il faut une ligne d'interruption par périphérique
  - Ces lignes sont connectées sur un **contrôleur d'interruption**

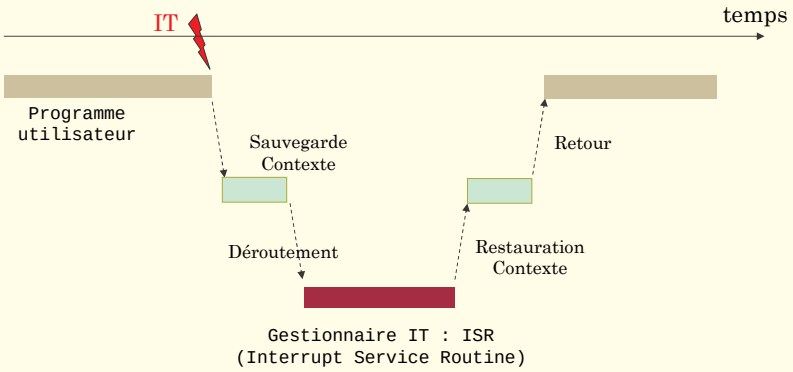
IV-1 Le contrôleur d'IT

- Rôles du contrôleur d'IT :
  - Vectorisation des interruptions :
    Le processeur retrouve alors le périphérique qui a généré l'interruption
  - Gestion des masques :
    IT masquables (activables) ou non,
    (NMI - No Maskable Interrupt, IRQ - Interrupt Request)
  - Eventuellement, Gestion des priorités :
    En cas de conflit sur les IT…
  - Eventuellement, Gestion des files de mémorisation :
    - Une File FIFO partagée entre toutes les IT,
    - Plusieurs files : une file dédiée à chaque type d'IT
    - …
- Propriété essentielle des IT : Transparence
  - Le programme interrompu ne doit pas se rendre compte qu'il n'a pas eu le CPU pendant un certains laps de temps
  - L'IT doit intégralement être traitée « dans son dos »

IV-1 Mise en œuvre de la transparence

« Transparence » du déroutement

mécanismes de sauvegarde et de restauration de *contexte* dans la zone mémoire temporaire (pile)



Contexte = { Registres « systèmes » ; Registres « utilisateurs » (uniquement ceux manipulés dans l'ISR)

IV-2 ROVIN : La gestion des IT internes et externes

- Concept unifié :
  IT (interne ou externe) = Message
- Un message signale l'arrivée d'un événement (« Event »).
- Fonctions EventOn() et EventOff() :
  activer/désactiver tous les événements, donc, toutes les IT.
- File d'attente de messages de type FIFO :

IV-2 ROVIN : La gestion des IT internes et externes

- GetMsg() : récupérer un message de la file d'attente
- Fonctions peu utilisées :
  CheckMsg() pour consulter le message sans le retirer
  ClearAllMsg() pour vider toute la file.
- Concept unifié :
  Une **seule** ISR, nommée IRQ_EVENT_(), pour gérer TOUS les messages (donc, toutes les interruptions).
- Donc, un unique vecteur d'interruption ! (le pointeur sur la fonction IRQ_EVENT_)…
  … et, dans l'ISR, test de type choix (« switch »)… pour connaître la source de l'interruption.

IV-2 ROVIN : Les messages associés aux sources d'interruption

IT externes

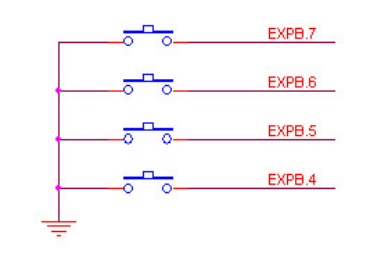
```
MSG_EXINT0      : APPARITION EVENEMENT EXINT0.
MSG_EXINT1      : APPARITION EVENEMENT EXINT1.
MSG_EXINT2      : APPARITION EVENEMENT EXINT2.
MSG_EXINT3      : APPARITION EVENEMENT EXINT3.
MSG_EXINT4      : APPARITION EVENEMENT EXINT4.
MSG_EXINT5      : APPARITION EVENEMENT EXINT5.
MSG_EXINT6      : APPARITION EVENEMENT EXINT6.
MSG_EXINT7      : APPARITION EVENEMENT EXINT7.
```

IT internes

```
MSG_NULL        : MESSAGE NUL.
MSG_ALARM       : APPARITION EVENEMENT ALARM.
MSG_VMTIMER     : APPARITION EVENEMENT VMTIMER.
MSG_ANCOMP      : APPARITION EVENEMENT ANCOMP.
MSG_UART0RX     : UART0 RECEIVE EVENT OCCURENCE.
MSG_UART1RX     : UART1 RECEIVE EVENT OCCURRENCE.
MSG_RTC          : RTC EVENT OCCURRENCE.
MSG_TIMER0      : TIMER0 EVENT OCCURRENCE.
MSG_TIMER1      : TIMER1 EVENT OCCURRENCE.
MSG_TIMER2      : TIMER2 EVENT OCCURRENCE.
MSG_TIMER3      : TIMER3 EVENT OCCURRENCE.
MSG_COUNTER0    : COUNTER0 EVENT OCCURRENCE.
MSG_COUNTER1    : COUNTER1 EVENT OCCURRENCE.
MSG_COUNTER2    : COUNTER2 EVENT OCCURRENCE.
MSG_CAPTURE0    : CAPTURE0 EVENT OCCURRENCE.
MSG_CAPTURE1    : CAPTURE1 EVENT OCCURRENCE.
```

IV-3 ROVIN – Exemple 1 : Interruptions externes MSG_EXINTi

- Pour tester ces IT externes, exemple avec 4 entrées EXINT.4 à EXINT.7 (broches EXPB4 à EXPB7).
- Sur ces broches, 4 boutons poussoirs sont connectés : BP4 … BP7



Front montant ou descendant

Niveau bas

Front descendant

Front montant

- Chacune des entrées est, sur cet exemple, configurée pour un mode de fonctionnement différent afin de pouvoir générer des ITs liées à des actions différentes.
- Affichage du message dans la fenêtre « On-The-Fly ».

IV-3 ROVIN – Source de l'exemple IT externes (page 1/2)

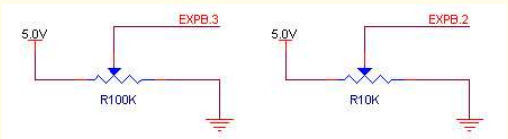
```
void PPI_Init(void)    // Initialisation des ports PA, PB et PC en entrées, et PD en sortie.
{
  PPI_SetMode(PA,0xff);   PPI_Out(PA,0xff);
  PPI_SetMode(PB,0xff);   PPI_Out(PB,0xff);
  PPI_SetMode(PC,0xff);   PPI_Out(PC,0xff);
  PPI_SetMode(PD,0x00);   PPI_Out(PD,0xff);
}

void EXPORT_Init(void) // Initialisation des ports EXPA, EXPB et EXPD en entrées
{
  PortSetMode(EXPA,0xff); PortSetMode(EXPB,0xff); PortSetMode(EXPD,0xff);
}

void IRQ_EVENT_(void)  // « Interrupt Service Routine » pour tous les événements
{
  unsigned char iMSG;  // Message d'IT : iMSG

  iMSG=GetMsg();       // Retrait d'un message de la file

  switch(iMSG)
  {
    case MSG_EXINT4 :  // Front montant
         DebugPrint("'EXINT4' détecté !  -> Front montant (relâchement de la touche).\n");
         break;
    case MSG_EXINT5 :  // Front descendant
         DebugPrint("'EXINT5' détecté !  -> Front descendant (Appui sur la touche).\n");
         break;
    case MSG_EXINT6 :  // Niveau bas
         DebugPrint("'EXINT6' détecté !  -> Niveau bas (maintien appuyé de la touche).\n");
         ExintOn(6);   // Réactivation de l'autorisation d'IT car fonctionnement sur niveau
         break;        // et non pas sur un front
    case MSG_EXINT7 :  // Front, montant ou descendant
         DebugPrint("'EXINT7' détecté !  -> Changement d'état (appui ou relâchement touche).\n");
         break;
  }
}
```


IV-3 ROVIN – Source de l'exemple IT externes (page 2/2)

```
void main(void)             // PROGRAMME PRINCIPAL
{
  PPI_Init();               // Initialisation des ports PA, PB, PC et PD du ROVIN
  EXPORT_Init();            // Initialisation des ports EXPA, EXPB et EXPD du ROVIN


  // Initialisation des entrées d'interruption EXINT

  PortSetMode(EXPB,0xf0);   // Les broches EXINT[7..4] sont en entrée.
  PortOut(EXPB,0xff);       // Pull UP sur les entrées


  // Configuration des IT externes

  ExintSet(7,EXINT_CHANGE); // IT sur changement d'état
  ExintOn(7);               // Autorisation individuelle de l'IT EXINT.7

  ExintSet(6,EXINT_LOW);    // IT sur niveau bas
  ExintOn(6);

  ExintSet(5,EXINT_LEDGE);  // IT sur front descendant
  ExintOn(5);

  ExintSet(4,EXINT_HEDGE);  // IT sur front montant
  ExintOn(4);



  EventOn();                // Autorisation globale de la génération des événements (IT) !

  while(1);                 // Attente d'une IT
}
```


IV-3 ROVIN – Exemple 2 : Comparateur interne ANCOMP

- Le module ROVIN est doté d'un comparateur analogique (ANCOMP).
- Principe : 2 entrées analogiques utilisées…
  - La première entrée sert à définir une tension analogique de « référence ».
  - L'entrée 1 est comparée en permanence à la tension analogique de l'entrée 2.
  - IT si le comparateur du ROVIN détecte que les deux tensions sont **identiques**.
- Pour tester le comparateur, deux potentiomètres sur les entrées 2 et 3 du port externe B (EXPB) :



- Ecriture d'un code source :
  - Configuration du comparateur
  - Autorisation de l'IT
  - Détection de la génération de l'IT (événement MSG_ANCOMP)
  - Affichage d'un message dans la fenêtre « On-The-Fly » du PC.

IV-3 ROVIN – Source de l'exemple IT interne ANCOMP

```
void PPI_Init(void)    // Initialisation des ports PA, PB et PC en entrée, et PD en sortie.
{
  PPI_SetMode(PA,0xff);   PPI_Out(PA,0xff);
  PPI_SetMode(PB,0xff);   PPI_Out(PB,0xff);
  PPI_SetMode(PC,0xff);   PPI_Out(PC,0xff);
  PPI_SetMode(PD,0x00);   PPI_Out(PD,0xff);
}

void EXPORT_Init(void) // Initialisation des ports EXPA, EXPB et EXPD en entrée
{
  PortSetMode(EXPA,0xff); PortSetMode(EXPB,0xff); PortSetMode(EXPD,0xff);
}

void IRQ_EVENT_(void)  // « Interrupt Service Routine » pour tous les événements
{
  switch(GetMsg()) {   // Récupération d'un message
    case MSG_ANCOMP :  // Test si IT en provenance du comparateur analogique
                       // Oui -> Affiche message dans la fenêtre "On-The-Fly"
        DebugPrint("Les 2 tensions sont identiques !\n");
        AncompOn();    // Autorise à nouveau la génération d'IT par le comparateur pour qu'en
                       // cas d'équilibre parfait, le message s'affiche en permanence.
        break;
  }
}

void main(void)        // PROGRAMME PRINCIPAL
{
  PPI_Init();          // Initialisation des ports PA, PB, PC et PD du ROVIN
  EXPORT_Init();       // Initialisation des ports EXPA, EXPB et EXPD du ROVIN
  AncompSet(ANCOMP_TOGGLE); // Configure le comparateur analogique
  AncompOn();          // Autorisation individuelle de la génération d'IT par ce dernier !
  RtcOn();             // Active l'horloge RTC car le comparateur dépend de cette dernière.
  EventOn();           // Autorisation globale de la génération des événements (IT) !
  while(1);            // Attente d'une IT
}
```


IV-3 ROVIN – Exemple 3 : COUNTER

- Un Counter est un outil de comptage de fronts (montants ou descendants) sur une entrée (généralement, une broche externe du processeur).
- Les fronts peuvent survenir irrégulièrement dans le temps
- Le ROVIN dispose de 3 Counters (0, 1 et 2).
- 3 fonctions indispensables pour faire fonctionner un Counter :
  - Counter_Set (numéro du counter, mode counter)
    configuration avec plusieurs modes possibles :
    - comptage des fronts descendants ou montants (COUNTER_LOWEDGE et COUNTER_HIGHEDGE)
    - choix d'émission d'une IT à chaque incrément ou non (COUNTER_1EVENT et COUNTER_XEVENT)
  - Counter_Count(numéro du counter, valeur comptage)
    la valeur de comptage indique combien il faut compter avant d'émettre une IT
  - Counter_On(numéro du counter)
    autorisation de l'émission de l'IT associée au counter passé en paramètre.

- Exemple :
  - Bouton poussoir 0 sur l'entrée Counter0 (EXPD.6)
  - Ce bouton devra être sollicité 3 fois de suite pour qu'une IT soit générée.
  - L'image de la valeur binaire du compteur est présentée sur les Leds du port PD.

IV-3 ROVIN – Source de l'exemple IT interne COUNTER

```
int NB_IT_Counter;     // Nombre d'interruptions comptabilisées

void PPI_Init(void)    // Initialisation des ports PA, PB et PC en entrée, et PD en sortie.
{
  PPI_SetMode(PA,0xff);   PPI_Out(PA,0xff);
  PPI_SetMode(PB,0xff);   PPI_Out(PB,0xff);
  PPI_SetMode(PC,0xff);   PPI_Out(PC,0xff);
  PPI_SetMode(PD,0x00);   PPI_Out(PD,0xff);
}

void IRQ_EVENT_(void)  // « Interrupt Service Routine » pour tous les événements
{
  switch(GetMsg()) {   // Récupération d'un message
     case MSG_COUNTER0 :
         NB_IT_Counter ++;             // Incrémentation de la valeur du counter
         PPI_Out(PD,~NB_IT_Counter);   // Affichage de la valeur binaire du compteur
         break;                        // sur les LED du port PD
  }
}

void main(void)        // PROGRAMME PRINCIPAL
{
  NB_IT_Counter = 0;
  PPI_Init();          // Initialisation des ports PA, PB, PC et PD du ROVIN

  // Configuration du counter 0 pour s'incrémenter sur chaque front descendant
  // donc, a chaque appui sur le bouton poussoir associé
  CounterSet(0, COUNTER_XEVENT|COUNTER_LOWEDGE);

  CounterCount(0, 3);  // Emission d'une IT après 3 appuis sur le bouton

  CounterOn(0);        // Autorisation individuelle de la génération d'IT par le Counter0

  EventOn();           // Autorisation globale de la génération des événements (IT) !
  while(1);            // Attente d'une IT
}
```


IV-3 ROVIN – Source de l'exemple IT interne COUNTER – variante

- Fonction CounterRead() : donne le nombre restant avant la prochaine IT

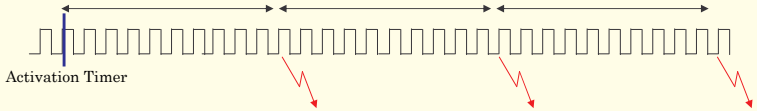
```
void PPI_Init(void)    // Initialisation des ports PA, PB et PC en entrée, et PD en sortie.
{
  PPI_SetMode(PA,0xff);   PPI_Out(PA,0xff);
  PPI_SetMode(PB,0xff);   PPI_Out(PB,0xff);
  PPI_SetMode(PC,0xff);   PPI_Out(PC,0xff);
  PPI_SetMode(PD,0x00);   PPI_Out(PD,0xff);
}

void main(void)        // PROGRAMME PRINCIPAL
{
  PPI_Init();          // Initialisation des ports PA, PB, PC et PD du ROVIN

  // Configuration du counter 2 pour s'incrémenter sur chaque front montant
  // et affichage, ** sans émission d'IT **, du compte à rebours à partir de 255.

  CounterSet(2, COUNTER_XEVENT|COUNTER_HIGHEDGE);

  CounterCount(2, 0xFF);  // 0xFF = 255...

  CounterOn(2);        // Autorisation individuelle de la génération d'IT par le Counter2
                       // nécessaire pour valider le compteur, même si l'IT ne sera pas émise...

  EventOff();          // PAS d'autorisation globale de la génération des IT
                       // donc, pas de fonction IRQ_EVT !
                       // L'ISR est inutile puisque la file ne se remplira pas : seul
                       // le mécanisme de comptage est utilisé ici.
  while(1) {
     // Affichage sur les LED du nombre de sollicitations nécessaires sur le bouton
     // avant d'arriver au bout du comptage et re-démarrer un compte à rebours.
     PPI_Out(PD, ~CounterRead(2));
  }
}
```

IV-3 ROVIN – Exemple 4 : TIMER

- Définition du Timer :
  compteur particulier, capable de compter les fronts d'horloge

  Attention, certains processeurs appellent parfois « Timer » un « Counter » (pour le comptage de fronts sur broches – c'est le cas de l'EasyPIC, qui unifie le concept –)

- Emission d'une IT lorsque le Timer atteint une valeur prédéfinie.
- Utilité : réveil programmé (périodique), watchdog, …



Emission des IT

IV-3 ROVIN – Exemple 4 : TIMER

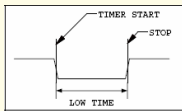
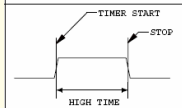
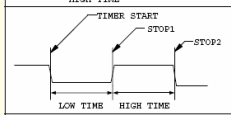
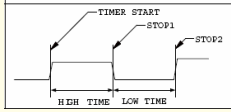
- Les Timers de ROVIN :
  Uniquement du comptage de **fronts montants de l'horloge**.
  - 4 Timers classiques Ti (T0, T1, T2, T3)
  - un Timer spécifique VmTimer :
    sur ce timer, il est possible de connaître la durée restante
    (fonction VmTimer_GetTime).
- 3 fonctions indispensables pour faire fonctionner un Timer :
  - Timer_SetTime(numéro du timer, valeur timer)
    configuration avec des pas multiples de 0.01 seconde
  - Timer_SourceOn()
    activation de la source du timer : CE=1 sur les 4 compteurs associés aux Timers Ti
  - Timer_On(numéro du timer)
    autorisation de l'émission de l'IT associée au Timer passé en paramètre.

IV-3 ROVIN – Exercice IT interne TIMER – Clignotement de 4 LEDs

```
void PPI_Init(void)    // Initialisation des ports PA, PB et PC en entrée, et PD en sortie.
{
  PPI_SetMode(PA,0xff);   PPI_Out(PA,0xff);
  PPI_SetMode(PB,0xff);   PPI_Out(PB,0xff);
  PPI_SetMode(PC,0xff);   PPI_Out(PC,0xff);
  PPI_SetMode(PD,0x00);   PPI_Out(PD,0xff);
}

void IRQ_EVENT_(void)  // « Interrupt Service Routine » pour tous les événements
{
  switch(GetMsg()) {   // Récupération d'un message
    case MSG_TIMER0 :
         PPI_BitOut(PD,0,~PPI_BitIn(PD,0)); break; // Changement état de LED 0 sur port D
    case MSG_TIMER1 :
         PPI_BitOut(PD,1,~PPI_BitIn(PD,1)); break; // Changement état de LED 1 sur port D
    case MSG_TIMER2 :
         PPI_BitOut(PD,2,~PPI_BitIn(PD,2)); break; // Changement état de LED 2 sur port D
    case MSG_TIMER3 :
         PPI_BitOut(PD,3,~PPI_BitIn(PD,3)); break; // Changement état de LED 3 sur port D
  }
}

void main(void)        // PROGRAMME PRINCIPAL
                       // Clignotement de LED a différentes fréquences
{
  PPI_Init();          // Initialisation des ports PA, PB, PC et PD du ROVIN
  Timer_SetTime(T0, 1);  Timer_On(T0);    // Initialisation des durées, en centième de
  Timer_SetTime(T1, 2);  Timer_On(T1);    // seconde, et autorisations individuelles
  Timer_SetTime(T2, 4);  Timer_On(T2);    // d'émission des IT pour chacun des
  Timer_SetTime(T3, 8);  Timer_On(T3);    // quatre Timers

  Timer_SourceOn();    // Lancement de tous les Timers (en validant le CE sur le compteur)

  EventOn();           // Autorisation globale de la génération des événements (IT) !
  while(1);            // Attente d'une IT
}
```


IV-3 ROVIN – Exemple 5 : IT interne de CAPTURE

- Capture = chronomètre ; mesurer un intervalle de temps entre 2 impulsions.
- Le ROVIN est doté de 2 entrées de captures indépendantes sur 16 bits.
- 4 modes de fonctionnement :

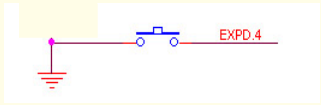
	CAPTURE_LOW Mesure la durée lorsque le signal est au niveau « BAS ».
	CAPTURE_HIGH Mesure la durée lorsque le signal est au niveau « HAUT ».
	CAPTURE_LOWCYCLE Mesure la durée lorsque le signal est au niveau « BAS », puis mesure la durée lorsque le signal est au niveau « HAUT ».
	CAPTURE_HIGHCYCLE Mesure la durée lorsque le signal est au niveau « HAUT », puis mesure la durée lorsque le signal est au niveau « BAS ».

NB : IT émise à la fin du cycle…

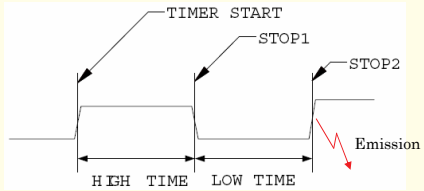
- Un diviseur d'horloge peut également être utilisé pour pouvoir mesurer des durées max. de 0,1 µs à 3,5 s.

IV-3 ROVIN – Exemple 5 : IT interne de CAPTURE

- Pour tester, une entrée de capture : capture 0 sur broche EXPD.4



- Par exemple, mesure d'un cycle High_Cycle :



- Appui sur le bouton-poussoir n° 0 pour placer l'entrée capture 0 au niveau 0 (car, présence de Pull-Up).
- Début de la première mesure au relâchement du bouton (Tmax : 1s)
- qui prend fin au début de la seconde mesure d'appui/relâchement
- Le dernier relâchement provoque l'IT

IV-3 ROVIN – Source de l'exemple IT interne CAPTURE

```
#define _capTIME_ ((double)1/18432000)*1024    // Unité de mesure = précision du chronomètre
unsigned short  CP_LOW_TIME, CP_HIGH_TIME;     // Variables globales pour la mesure

void IRQ_EVENT_(void)   // « Interrupt Service Routine » pour tous les événements
{
  switch(GetMsg()) {    // Récupération d'un message
    case MSG_CAPTURE0 : // Test si IT en provenance du module de capture n° 0

         CaptureRead(0, &CP_LOW_TIME, &CP_HIGH_TIME); // Lecture et sauvegarde

         DebugClear();  // Effacement de la fenêtre d'affichage
         DebugPrint("Durée niveau HAUT = ");
         DebugDOUBLE(_capTIME_ * CP_HIGH_TIME, DEC);
         DebugPrint(" sec.\n");

         DebugPrint("Durée niveau BAS = ");
         DebugDOUBLE(_capTIME_ * CP_LOW_TIME, DEC);
         DebugPrint(" sec.\n");

         CaptureOn(0);  // Réactive la capture
         break;
}

void main(void)         // PROGRAMME PRINCIPAL
{
  PPI_Init();           // Initialisation des ports PA, PB, PC et PD du ROVIN
  EXPORT_Init();        // Initialisation des ports EXPA, EXPB et EXPD du ROVIN

  CaptureSet(0, CAPTURE_HIGHCYCLE|CAPTURE_DIV1024);
                        // Configure l'entrée de capture en mode "HIGHCYCLE" afin de capturer
                        // la durée d'une impulsion haute suivie d'une impulsion basse
                        // Introduit un facteur de division de 1024

  CaptureOn(0);         // Autorisation individuelle de la génération d'IT par la capture
  EventOn();            // Autorisation globale de la génération des événements (IT) !
  while(1);             // Attente d'une IT
}
```

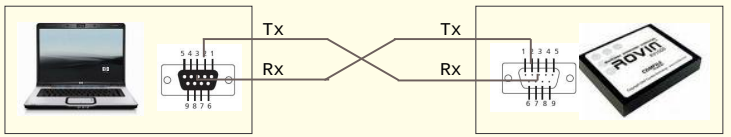

IV-3 ROVIN – Exemple 6 : IT interne d'ALARM de RTC

- RTC (« Real Time Clock ») : circuit capable de fournir à tout moment la date du jour et l'heure courante.
- La fonction RtcSetDate() permet de mettre l'horloge à l'heure, si elle ne l'est pas déjà. (RtcGetDate pour connaître la date et l'heure).
- La RTC est capable d'émettre une IT toutes les secondes : RtcEventOn()
- ou bien, elle émet une IT au moment demandé (capacité ALARM, afin de programmer une alarme, pour se réveiller à la fin des cours, par exemple).
- Fonction AlarmSetTime() pour configurer l'heure du réveil.

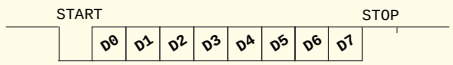
```
void IRQ_EVENT_(void)      // « Interrupt Service Routine » pour tous les événements
{
  switch(GetMsg()) {       // Récupération d'un message
    case MSG_ALARM :
         DebugPrint("\n Ca y est, mon gars... fin des souffrances ! \n ");
         break;
    }
}

void main(void)            // PROGRAMME PRINCIPAL
{
  AlarmSetTime(17,30,00);  // Réveil tous les jours à 17h30
  RtcOn();                 // Il faut quand même que l'horloge fonctionne
  AlarmOn();               // Autorisation individuelle de la génération d'IT par l'alarme
  EventOn();               // Autorisation globale de la génération des événements (IT) !
  while(1);                // Attente d'une IT
}
```


IV-3 ROVIN – Exemple 7 : Liaison série UART (type RS232)

- UART : « Universal Asynchronous Receiver Transmitter »
  - USART (« Synchronous or Asynchronous »)
  - Liaison série asynchrone (RS232,…)
  - Sérialisation des données fournies en parallèle, via des registres à décalage (Shift Registers)
  - Mode « full duplex » (émission / réception simultanée) : Rx et Tx



  - « Baud Rate » : débit de transmission (9600, 19200, 38400, 57600, 115200 Bps)
  - Trame UART :
    - 1 bit de départ (Start = niveau logique 0),
    - Bits de données (5 à 9, avec classiquement 8 bits),
    - Bit de parité (facultatif, pour le contrôle de flux),
    - 1 (ou 2) bit d'arrêt (Stop = niveau logique 1)
- UART0 et UART1 sur ROVIN :
  - 8 bits de données,
  - sans parité,
  - avec 1 bit d'arrêt,
  - et un débit d'au maximum 115200 Bps.



IV-3 ROVIN – Exemple 7 : Liaison série UART

- Fonctions minimales à employer pour **transmettre** (émettre) :
  - UartSetBaud() : classiquement, de 9600 Bps à 115200 Bps.
  - UartTxOn() et UartTxOff : activation et désactivation de la broche Tx de l'UART
  - Ou bien : UartOn() et UartOff() : activation et désactivation de l'UART en Rx et Tx
  - UartWrite() : écriture d'un octet sur la broche Tx
- Fonctions minimales à employer pour **recevoir** :
  - UartSetBaud(), UartOn() et UartOff() : comme pour la transmission
  - UartRxOn() et UartRxOff : activation et désactivation de la broche Rx de l'UART
  - UartSetPacketSize() : configuration de la taille du paquet
    - Le paquet est à destination du buffer de réception de l'UART.
    - Unité : octet
  - UartEventOn() et EventOn() : autorisation des IT en provenance de l'UART.
    - Dès qu'un paquet est reçu, une IT (MSG_UARTiRX) est émise.
  - UartRead() et UartRxBufRead() : lecture d'un paquet du buffer de réception.
- Exemple : ré-émission sur Tx d'un octet lu sur buffer de Rx de l'UART n° 0

IV-3 ROVIN – Exemple 7 : Liaison série UART

```
#define Taille_Paquet 1    // Taille du paquet, en octet

void IRQ_EVENT_(void)      // « Interrupt Service Routine » pour tous les événements
{
  unsigned char Octet_Recu;// Paquet de 1 octet.

  switch(GetMsg()) {       // Récupération d'un message
    case MSG_UART0RX :
         DebugPrint(" Paquet recu...");
         Octet_Recu = UartRead(0);     // Lecture d'un octet à partir du buffer de Rx...
                                       // ou bien : UartRxBufRead(0,&Octet_Recu);
         UartWrite(0,Octet_Recu);      // ...suivi de sa ré-écriture sur Tx
         break;
    }
}

void main(void)            // PROGRAMME PRINCIPAL
{
  PPI_Init();              // Initialisation des ports
  EXPORT_Init();

  UartSetBaud(0,115200);   // UART n°0 en 115200 Bps avec paquet de 1 octet
  UartSetPacketSize(0,Taille_Paquet);

  UartOn(0)                // Activation de l'UART

  UartEventOn(0);          // Autorisation individuelle de la génération d'IT par l'UART0
  EventOn();               // Autorisation globale de la génération des événements (IT) !
  while(1);                // Attente d'une IT
}
```


IV-3 ROVIN – Exemple 7 : Liaison série UART

- Variante : lecture de paquets de 2 octets

```
#define Taille_Paquet 2    // Taille du paquet, en octet

void IRQ_EVENT_(void)      // « Interrupt Service Routine » pour tous les événements
{
  unsigned char Paquet[Taille_Paquet]; // Déclaration du paquet = buffer

  switch(GetMsg()) {       // Récupération d'un message
    case MSG_UART0RX :
         DebugPrint(" Paquet recu...");
         Paquet[0] = UartRead(0);  // Récupération du paquet à partir du buffer de Rx...
         Paquet[1] = UartRead(0);  // (second octet)
         UartWrite(0,Paquet[0]);   // ...suivi de sa ré-écriture sur Tx
         UartWrite(0,Paquet[1]);   // (second octet)
         // OU Mieux, a la place de ces 4 lignes :
         //       UartRxBufRead(0,Paquet);            // Récupération du paquet
         //       UartBufOut(0,Paquet,Taille_Paquet); // Emission du paquet
         break;
    }
}

void main(void)            // PROGRAMME PRINCIPAL
{
  PPI_Init();              // Initialisation des ports
  EXPORT_Init();

  UartSetBaud(0,115200);   // UART n°0 en 115200 Bps avec paquet de 2 octets
  UartSetPacketSize(0,Taille_Paquet);

  UartOn(0)                // Activation de l'UART

  UartEventOn(0);          // Autorisation individuelle de la génération d'IT par l'UART0
  EventOn();               // Autorisation globale de la génération des événements (IT) !
  while(1);                // Attente d'une IT
}
```


Sommaire

I Introduction sur les Systèmes Embarqués

II Les outils de développement

III Interaction avec l'environnement : les transfert de données

IV Interaction avec l'environnement : les interruptions

1 Principe de fonctionnement des interruptions
2 Les IT selon ROVIN
3 Exemples ROVIN : IT externe, comparateur, capture, timer, alarme et RS232
4 Les IT selon EasyPIC
5 Exemples EasyPIC : IT timer

V Temps Réel

IV-4 Les interruptions sur EasyPic

- 14 sources d'interruption :
  - Interruptions primaires : Timer0, Pin RB0 et Ch. RB4
  - Interruptions secondaires (ou périphériques) : Timer1 et 2, UART, port série, …
- 3 niveaux d'autorisation d'interruption (« Interrupt Enable ») :
  - Autorisation globale :
    - bit GIE (« Global Interrupt Enable ») dans INTCON
  - Autorisation périphériques :
    - bit PEIE (« Peripheric Interrupt Enable ») dans INTCON
  - Autorisations individuelles :
    - Interruptions primaires : dans registre INTCON
    - Interruptions périphériques : dans registres PIE1 et PIE2 (en bank 1)
- Concept unifié :
  - Une seule ISR (comme pour ROVIN) pour gérer toutes les interruptions.
  - Un unique vecteur d'interruption : pointeur (situé à l'adresse 0x04) sur la fonction C nommée « **`interrupt`** » (nom réservé à l'ISR)
  - Flag : bit qui se trouve positionné lors d'une demande d'interruption. La lecture de ce flag permet à l'ISR de déterminer la source de l'IT.
- Une interruption ne peut pas être interrompue par une autre interruption.
- Contexte minimaliste :
  - Contexte = PC (pointeur instruction),
  - sauvegardé dans une pile de taille 8.
  - Prévoir une sauvegarde logicielle des autres registres : W et STATUS

IV-4 EasyPic – Les sources d'interruption

	Déclencheur	Flag	Registre	Adr	PEIE	Enable	Registre	Adr
Primaires	Timer 0	T0IF	INTCON	0x0B	NON	T0IE	INTCON	0x0B
	Pin RB0 / INT	INTF	INTCON	0x0B	NON	INTE	INTCON	0x0B
	Ch. RB4/RB7	RBIF	INTCON	0x0B	NON	RBIE	INTCON	0x0B
Secondaires	Convert. A/D	ADIF	PIR1	0x0C	OUI	ADIE	PIE1	0x8C
	Rx USART	RCIF	PIR1	0x0C	OUI	RCIE	PIE1	0x8C
	Tx USART	TXIF	PIR1	0x0C	OUI	TXIE	PIE1	0x8C
	Port série SSP	SSPIF	PIR1	0x0C	OUI	SSPIE	PIE1	0x8C
	Module CCP1	CCP1IF	PIR1	0x0C	OUI	CCP1IE	PIE1	0x8C
	Module CCP2	CCP2IF	PIR2	0x0D	OUI	CCP2IE	PIE2	0x8D
	Timer 1	TMR1IF	PIR1	0x0C	OUI	TMR1IE	PIE1	0x8C
	Timer 2	TMR2IF	PIR1	0x0C	OUI	TMR2IE	PIE1	0x8C
	EEPROM	EEIF	PIR2	0x0D	OUI	EEIE	PIE2	0x8D
	SSP mode I2C	BCLIF	PIR2	0x0D	OUI	BCLIE	PIE2	0x8D
	Port parallèle	PSPIF	PIR1	0x0C	OUI	PSPIE	PIE1	0x8C

Registres concernés — Registres concernés

Bits indicateurs de présence d'IT — Bits d'autorisation individuelle

IV-4 EasyPic – Le registre INTCON

- Registre INTCON (« Interrupt Control »): dédié au contrôle des interruptions
- Présent à l'adresse 0x0B sur les 4 banks

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

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

- GIE (Global Interrupt Enable) : bit pour valider ou invalider globalement les interruptions
- PEIE : bit pour valider ou invalider les interruptions périphériques.
- TMR0IE : bit pour valider ou invalider l'interruption liée au timer0
- INTE : bit pour valider ou invalider l'interruption liée à un changement de niveau sur la broche RBO
- RBIE : bit pour valider ou invalider l'interruption liée à un changement de niveau sur les broches RB4 à RB7
- TMR0IF, INTF et RBIF : flags signalant les interruptions TMR0, INT et RB

IV-4 EasyPic – Principe d'installation d'une IT

- Mise en service des interruptions primaires :
  - Valider le bit individuel de l'interruption (registre INTCON),
  - Valider le bit global GIE (registre INTCON)
- Mise en service des interruptions périphériques :
  - Valider le bit individuel de l'interruption (registre PIE1 ou PIE2)
  - Valider le bit du groupe périphérique PEIE (registre INTCON)
  - Valider le bit global GIE (registre INTCON)

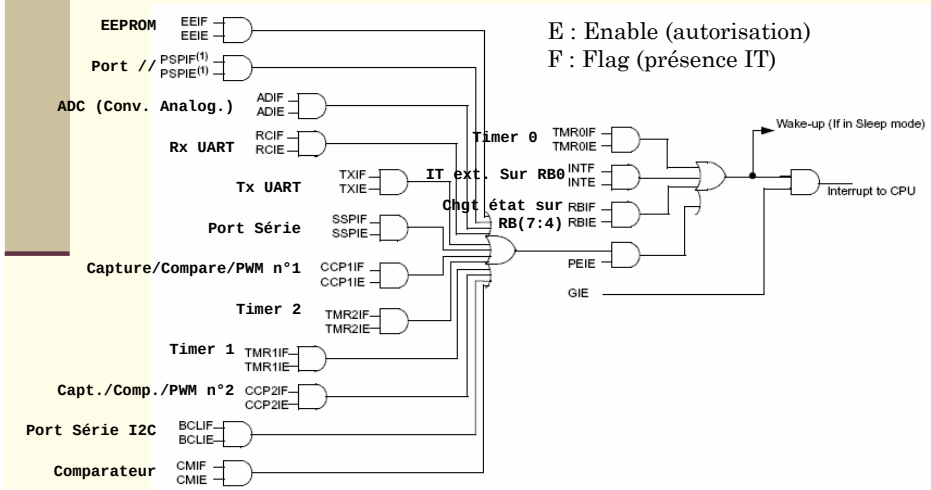
PIE1 REGISTER (ADDRESS 8Ch)

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

PIE2 REGISTER (ADDRESS 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

IV-4 EasyPic – Logique de contrôle des IT



IV-4 EasyPic – Principe de détection de la source d'une IT

- Lorsqu'une IT survient, déroutement en 0x04 (vecteur ISR)
- En C, cette ISR est la fonction nommée « interrupt() ».
- L'ISR doit avant tout traitement détecter l'origine de la source d'interruption
  - Vérification pour une interruption primaire sur les 3 flags TMR0IF, INTF et RBIF du registre INTCON
  - Vérification pour une interruption périphérique dans les registres PIR1 et PIR2

PIR1 REGISTER (ADDRESS 0Ch)

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

PIR2 REGISTER (ADDRESS 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	Reserved	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

- A la fin de l'ISR, ne pas oublier d'effacer le flag qui a provoqué l'IT !

IV-5 EasyPic – Exemple 1 : module TIMER0 (caractéristiques)

- Caractéristiques du module TIMER0 :
  - Timer ou Counter sur 8 bits
  - Prédiviseur programmable logiciellement sur 8 bits (afin d'éviter des interruptions trop fréquentes)
  - Fonctionnement sur horloge interne ou externe
  - Choix du sens des fronts pour l'horloge externe
  - IT produit lors de la transition 0xFF à 0x00 (flag TMR0IF dans le registre INTCON lors du débordement)
- 3 registres sont manipulés pour utiliser le TIMER0 :
  - **TMR0** (adresse 0x01 en bank 0) : valeur du timer.
    Attention, au démarrage, ce registre n'est pas initialisé
    Possibilité de lire ou d'écrire dans TMR0 (valeur limitée à 255)
  - **INTCON** (adresse 0x0B sur les 4 banks) : Autorisation et Flag de présence de l'interruption TMR0.
    Bits concernés :
    - GIE et TMR0IE : Autorisations globale et individuelle
    - TMR0IF : Flag de présence d'IT
  - **OPTION_REG** (adresse 0x81 en bank 0) : paramètres de configuration.
    Bits concernés (Cf. détails page suivante) :
    - T0CS (Timer Counter Selection),
    - T0SE (Timer Source Edge),
    - PSA (PreScaler Assignment),
    - PS(2:0) (PreScale ratio)

IV-5 EasyPic – Exemple 1 : module TIMER0 (options)

Configurations possibles via le registre OPTION_REG :

- Choix du mode de fonctionnement :
  - T0CS = 0 : fonctionnement en mode timer (mesure de temps basé sur les cycles horloge interne)
  - T0CS = 1 : fonctionnement en mode compteur (nombre d'impulsions sur la broche RA4 du portA)
- Choix de l'orientation des fronts :
  - T0SE = 0 : en mode compteur, comptage sur front montant de RA4
  - T0SE = 1 : en mode compteur, comptage sur front descendant de RA4
- Choix d'utiliser un prédiviseur (entre 2 et 256) :
  - PSA=0 : activation de la prédivision
  - PS(2:0) : valeur de division :

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

IV-5 EasyPic – Exemple 1 : source d'utilisation du TIMER0

```
unsigned int COUNT;    // Variable globale pour un comptage

void interrupt() {
  COUNT++;             // Incrementer la valeur a chaque interruption
  TMR0 = 56;           // Valeur provoquant une IT dans (256-56) = 200 unités de temps
  INTCON = 0b00100000; // TMR0IE=1 (bit 5) et TMR0IF=0 (bit 2)
                       // qui peut s'ecrire aussi : INTCON.TMR0IE=1; INTCON.TMR0IF=0;
}

void main()            // PROGRAMME PRINCIPAL
{
  // Initialisation des LED
  TRISB = 0x00;
  PORTB = 0xF0;

  // Initialisation du Timer 0
  OPTION_REG = 0b10000100; // Configuration choisie :
                           //    OPTION_REG.T0CS=0; mode timer
                           //    OPTION_REG.PSA=0; avec prédiviseur sur TMR0
                           //    OPTION_REG.PS2=1;
                           //    OPTION_REG.PS1=0; avec division de 32
                           //    OPTION_REG.PS0=0;

  TMR0  = 56;              // Initialisation provoquant une IT dans (256-56) = 200 unités de temps

  INTCON = 0b10100000;     // INTCON.TMR0IE=1 : autorisation de l'IT TMRO
                           // INTCON.GIE=1 : autorisation globale de toutes les IT

  COUNT = 0;               // Initialisation de la valeur de comptage

  while(1){
    if (COUNT == 400) {
      PORTB = ~PORTB;      // Clignotement des LED tous les 400 coups...
      COUNT = 0;
    }
  }
}
```

OPTION_REG REGISTER (ADDRESS 81h, 181h)

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

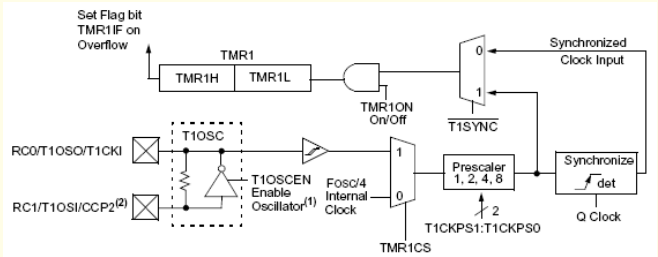
IV-5 EasyPic – Exemple 2 : module TIMER1 (caractéristiques)

- Caractéristiques du module TIMER1 :
  - Timer ou Counter sur 16 bits (contre 8 bits pour TIMER0)
  - Prédiviseur programmable limité à une division maximale par 8
  - Fonctionnement sur horloge interne ou externe
  - Pas le choix du sens des fronts pour l'horloge externe
  - IT produit lors de la transition 0xFFFF à 0x0000 (flag TMR1IF dans le registre PIR1 lors du débordement)

- 6 registres sont manipulés pour utiliser le TIMER1 :
  - **TMR1H et TMR1L** (adresses 0x0E et 0x0F en bank 0) : valeur du timer sur deux octets.
  - **INTCON** (0x0B sur les 4 banks) : Autorisations d'interruption
    - GIE : Autorisation globale
    - PEIE : Autorisation secondaire (périphérique)
  - **PIR1** (0x0C) :
    - TMR1IF, Flag de présence de l'interruption TMR1.
  - **PIE1** (0x8C) :
    - TMR1IE, Autorisation individuelle de l'interruption TMR1.
  - **T1CON** (0x10) : paramètres de configuration.
    - Voir page suivante…

IV-5 EasyPic – Exemple 2 : module TIMER1 (caractéristiques)

- **T1CON** (0x10) : paramètres de configuration.
  - T1OSCEN (Timer 1 Oscillator Enable) : choix de l'horloge (externe sur RC0 ou oscillateur)
  - TMR1CS (Timer1 Clock Source Select) : choix de l'horloge
    - 1 = horloge externe, sur front montant
    - 0 = horloge interne (FOSC/4)
  - T1CKPS(1:0) : Prédiviseur (00 ⇒ /1, 01 ⇒ /2, 10 ⇒ /4 et 11 ⇒ /8),
  - T1SYNC (Timer 1 External Clock Input Synchronization Control) : choix de la synchronisation avec l'horloge externe
  - TIMR1ON : marche/arrêt du timer 1.



IV-5 EasyPic – Exemple 2 : source d'utilisation du TIMER1

```
unsigned int COUNT;    // Variable globale pour un comptage

void interrupt() {
  COUNT++;             // Incrementer la valeur a chaque interruption
  PIR1.TMR1IF = 0;     // Effacement du flag d'IT TMR1IF
}

void main()            // PROGRAMME PRINCIPAL
{
  // Initialisation des LED con
  TRISB = 0x00;        // PORTB
  PORTB = 0xF0;        // Extin

  // Initialisation du Timer 1
  T1CON = 0b00000001;  // Configuration choisie :
                       //    T1CON.TMR1ON=1; activation timer 1
                       //    T1CON.TMR1CS=0; fonctionnant sur horloge interne
                       //    T1CON.T1CKPS1=0; et  T1CON.T1CKPS0=0; sans division

  TMR1L = 0xCD;        // Valeur d'initialisation du compteur : 0xABCD
  TMR1H = 0xAB;        // NB : seule la 1ere IT se fera au bout de ce temps
                       // Les autres auront lieu toutes les 0xFFFF, car pas de mise a jour
                       // de TMR1L et TMR1H dans l'ISR interrupt()

  PIE1.TMR1IE = 1;     // Autorisation de l'IT TMR1IF
  INTCON = 0b11000000; // INTCON.PIE=1 : autorisation des IT secondaires périphériques
                       // INTCON.GIE=1 : autorisation globale de toutes les IT

  COUNT = 0;           // Initialisation de la valeur de comptage

  while(1){
    if (COUNT == 0x7FFF) {
      PORTB = ~PORTB;  // Clignotement regulier des LED
      COUNT = 0;
    }
  }
}
```

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7							bit 0

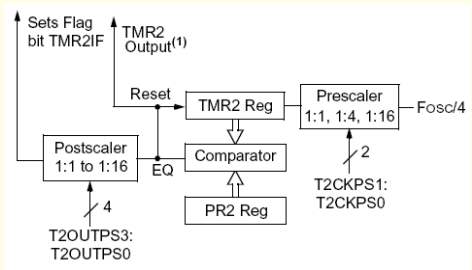
IV-5 EasyPic – Exemple 3 : module TIMER2 (caractéristiques)

- Caractéristiques du module TIMER2 :
  - Mode timer (avec horloge interne) uniquement
  - Comptage sur 8 bits, initialisés au démarrage
  - Possibilité d'un prédiviseur (1, 4 ou 16) et surtout d'un postdiviseur (entre 1 et 16) ⇒ grande gamme de diviseurs effectifs
  - IT produit lors de la transition 0xFF à 0x00 (flag TMR2IF dans le registre PIR1 lors du débordement)

- 6 registres sont manipulés pour utiliser le TIMER1 :
  - **TMR2** (0x11 en bank 0) : valeur du timer
  - **INTCON** (0x0B sur les 4 banks) : Autorisations d'interruption
    - GIE : Autorisation globale
    - PEIE : Autorisation secondaire (périphérique)
  - **PIR1** (0x0C) :
    - TMR2IF, Flag de présence de l'interruption TMR2.
  - **PIE1** (0x8C) :
    - TMR2IE, Autorisation individuelle de l'interruption TMR2.
  - **PR2** (0x92) : Période du timer
  - **T2CON** (0x12) : paramètres de configuration.
    - Voir page suivante…

IV-5 EasyPic – Exemple 3 : module TIMER2 (caractéristiques)

- **T2CON** (0x12) : paramètres de configuration.
  - T2CKPS(1:0) : Prédiviseur (00 ⇒ /1, 01 ⇒ /4, 1x ⇒ /16),
  - T2OUTPS(3:0) : Postdiviseur (0000 ⇒ /1, 0001 ⇒ /2, 0010 ⇒ /3, …1111 ⇒ /16),
  - TIMR2ON : marche/arrêt du timer 2.
- Fonctionnement :
  - Prédiviseur d'horloge re-divise FOSC/4 selon la configuration de T2CKPS(1:0)
  - Chaque cycle de cette horloge divisée provoque l'incrémentation de TMR2
  - Si le contenu de TMR2 égale le contenu de PR2, remise à 0 de TMR2 et envoi de ce signal EQ sur le postdiviseur.
  - Postdiviseur divise EQ selon la configuration de T2OUTPS(3:0)
  - A chaque modulo, positionnement du flag TMR2IF et génération de l'interruption.



IV-5 EasyPic – Exemple 3 : source d'utilisation du TIMER2

```
unsigned int COUNT;    // Variable globale pour un comptage

void interrupt() {
  if (PIR1.TMR2IF) {   // Pour une fois, verification de la source d'IT...
    COUNT++;           // Incrementer la variable bidon a chaque interruption
    PIR1.TMR2IF=0;     // Effacement du flag d'IT TMR2IF, ou plus FUN : PIR1 &= ~(1<<TMR2IF);
  }
}

void main()            // PROGRAMME PRINCIPAL
{
  // Initialisation des LED con
  TRISB = 0x00;        // PORTB
  PORTB = 0xF0;        // Extin

  // Initialisation du Timer 1
  T2CON = 0b01111111;  // Configuration choisie : division par 256
                       //    T2CON.TMR2ON=1; activation timer 2
                       //    T2CON.TOUTPS(3:0)=0b1111; Postdiviseur de 16
                       //    T2CON.T2CKPS(1:0)=0b11; Prediviseur de 16

  TMR2 = 0x00;         // Valeur d'initialisation du compteur : 0

  PIE1.TMR2IE = 1;     // Autorisation de l'IT TMR1IF, ou pour les amateurs : PIE1 |= (1<<TMR2IE);
  INTCON = 0b11000000; // INTCON.PIE=1 : autorisation des IT secondaires
                       // INTCON.GIE=1 : autorisation globale de toutes les IT

  COUNT = 0;           // Initialisation de la valeur de comptage

  while(1){
    if (COUNT == 3906) {
      PORTB = ~PORTB;  // Clignotement des LED toutes les secondes avec Quartz a 8 Mz
      COUNT = 0;       // car FOSC/4 = 2 MHz; puis /256 => 7812Hz, puis /3906 => 2Hz
    }
  }
}
```

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

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

IV-5 EasyPic – Protocole pour choisir un TIMER

- Besoin d'un temps compris entre 256 et 65536 cycles ?
  - valeur multiple de 256 (par exemple 512) ⇒ timer 0, 1 ou 2.
  - non divisibles par des puissances de 2 ⇒ timer 1 ou 2.
- Besoin de temps allant jusqu'à 524288 cycles ?
  - Timer 1 avec prédiviseur mais perte de précision
- Besoin de temps quelconques, mais de grande précision ?
  - Utilisation du timer1 avec un quartz
  - Utilisation du timer2 avec des valeurs idéales pour le prédiviseur et le postdiviseur
- Besoin de comptage d'événements sur une broche ?
  - Timer 0 sur front montant ou descendant
  - Timer 1 sur front montant uniquement
  - Timer 2 impossible