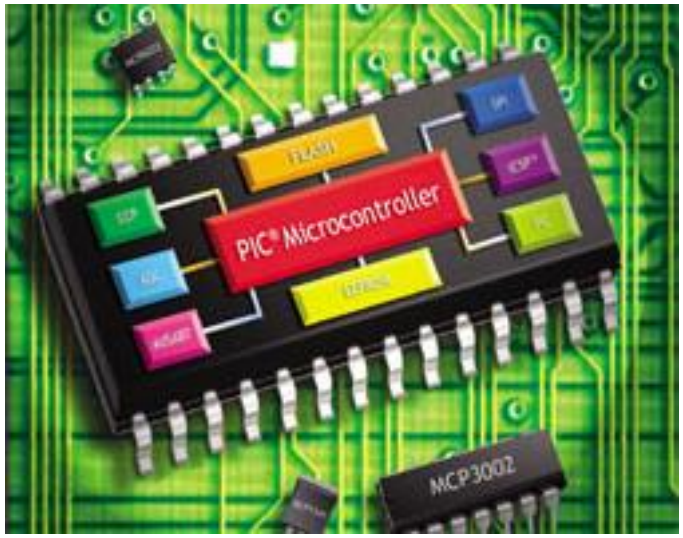
Programmation XC8
µC PIC 16F



Module - ERS2

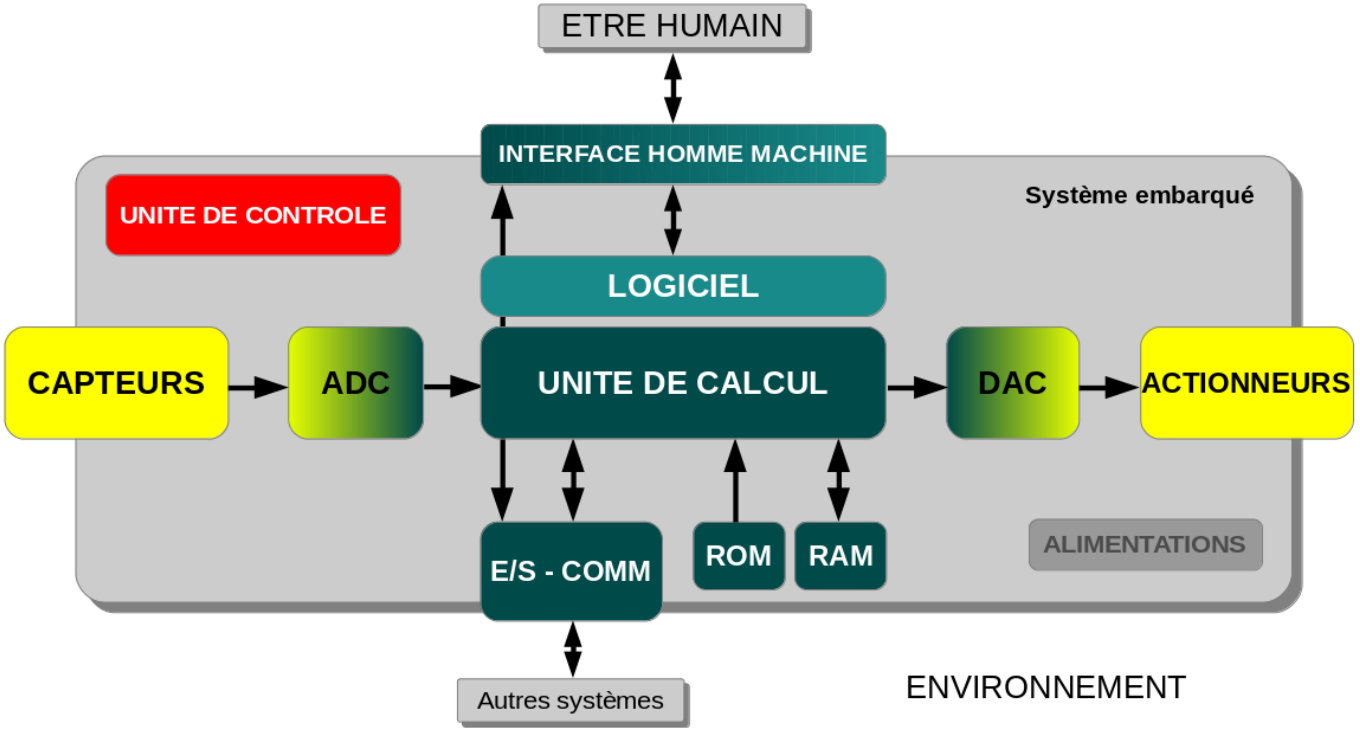
Frédéric Giamarchi
IUT de Nîmes – Centre Spatial Universitaire
Université de Montpellier

Objectifs

- Découvrir les systèmes embarqués
- Utiliser un composant de la famille PIC16F
- Associer électronique et programmation
- Apprendre à lire la documentation d'un µC
- Apprendre à programmer en C
- Réaliser un mini projet avec un µC

Les systèmes embarqués

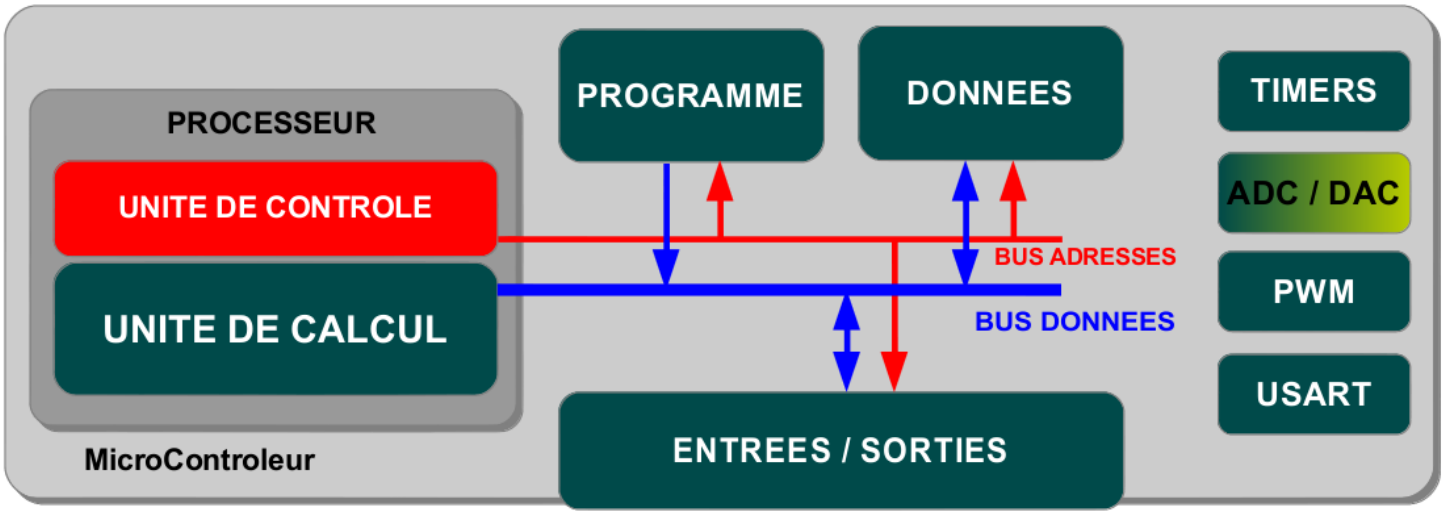
- Système matériel + logiciel, autonome



- API, Électroménager, Missiles, Satellite, Robots, Drones, Automobile, Smartphone, Box…

PIC 16F – un microcontrôleur d'entrée de gamme

- Peu de composants externes
- Ressources internes identiques aux PIC 18F



- Parfait pour débuter

PIC 16F – Architecture interne

- **Les mémoires :**

Flash (ROM Read Only Memory)

Mémoire qui contient le programme.

RAM (Random Access Memory)

Mémoire pour les données de calcul.

EEPROM (Elec. Erasable Prog. Read Only Memory)

Mémoire lente des données, même après coupure de l'alim.

- **Le microprocesseur (CPU) :**

ALU, PC, Registre, Pile, Horloge

- **Les Ports d'entrées/sorties :**

PORTA, PORTB (2x8 lignes E/S)

- **Les Ressources internes :**

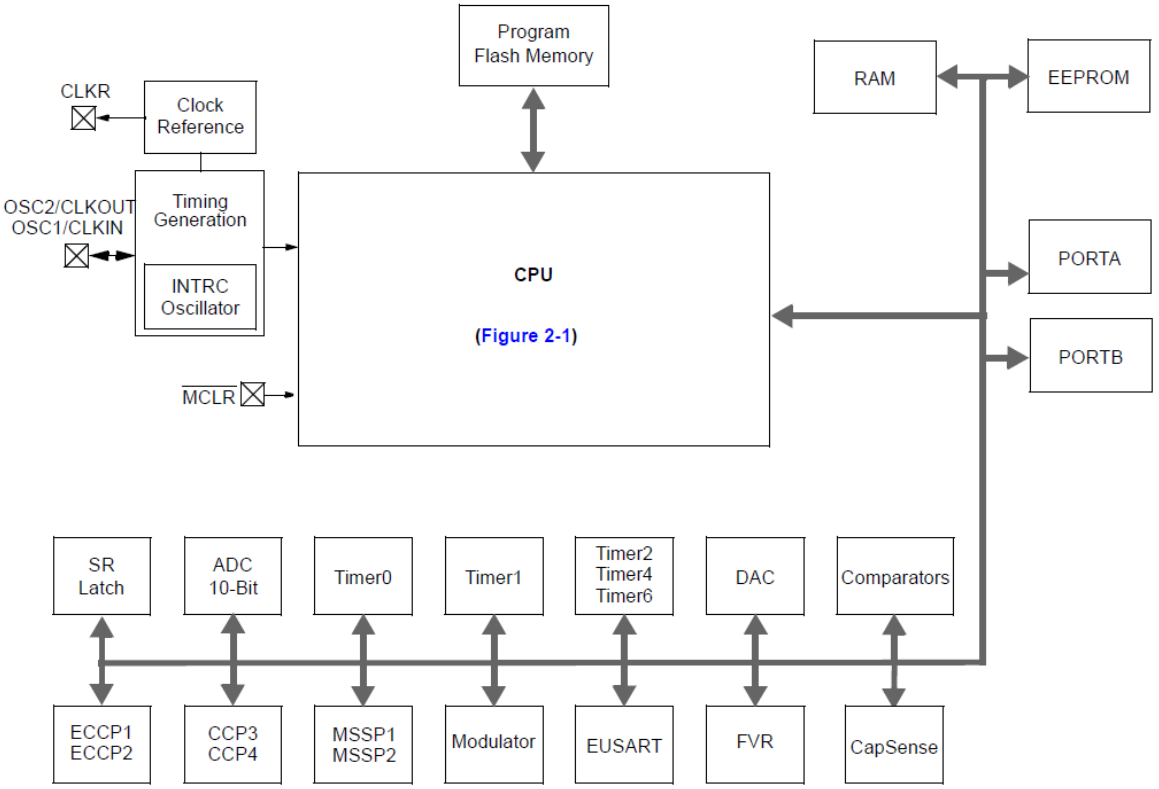
Timer (Compteur 8 ou 16bits)

ADC (Conv. Analog. → Digit.)

USART (Interface de communication série)

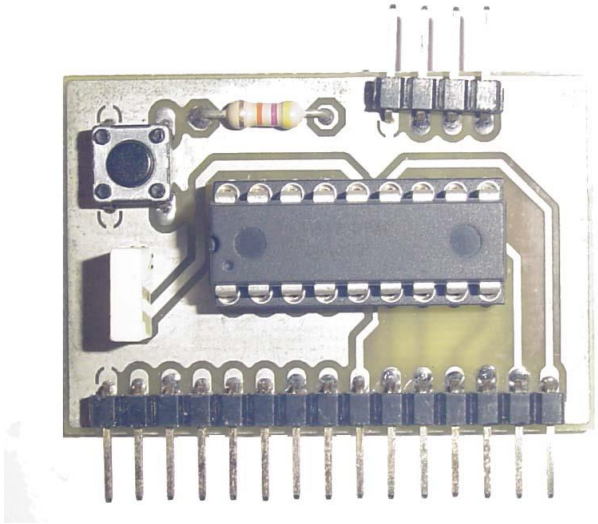
CCP (PWM, MLI, Modulateur de largeur d'impulsions)

Comparateur, SSP (I2C, SPI)

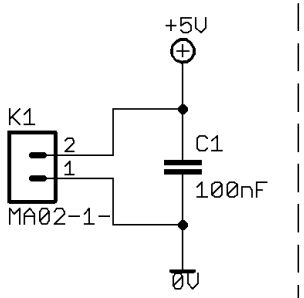


Carte d'étude à µC PIC 16F

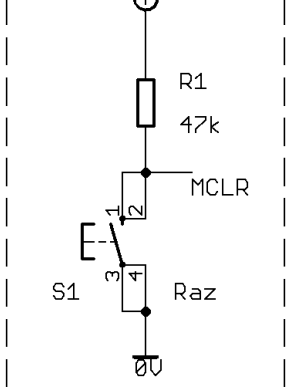
- **La carte d'étude**
  - **µC PIC 16F88, 16F1847**
  - **Bouton Reset**
  - **Connecteur pour les applications**
  - **Connecteur pour la programmation et le dialogue avec le PC**



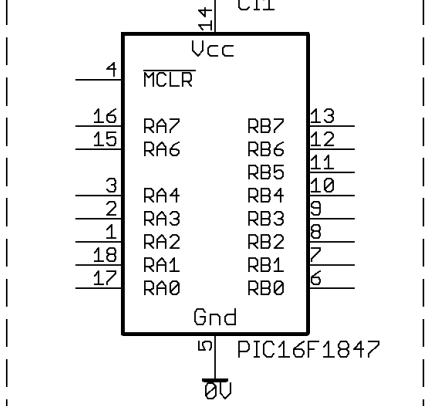
FS11



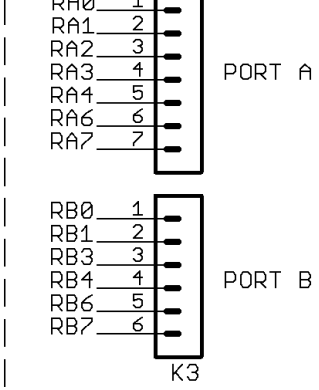
FS12



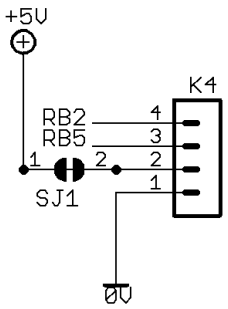
FS13



FS14



FS15

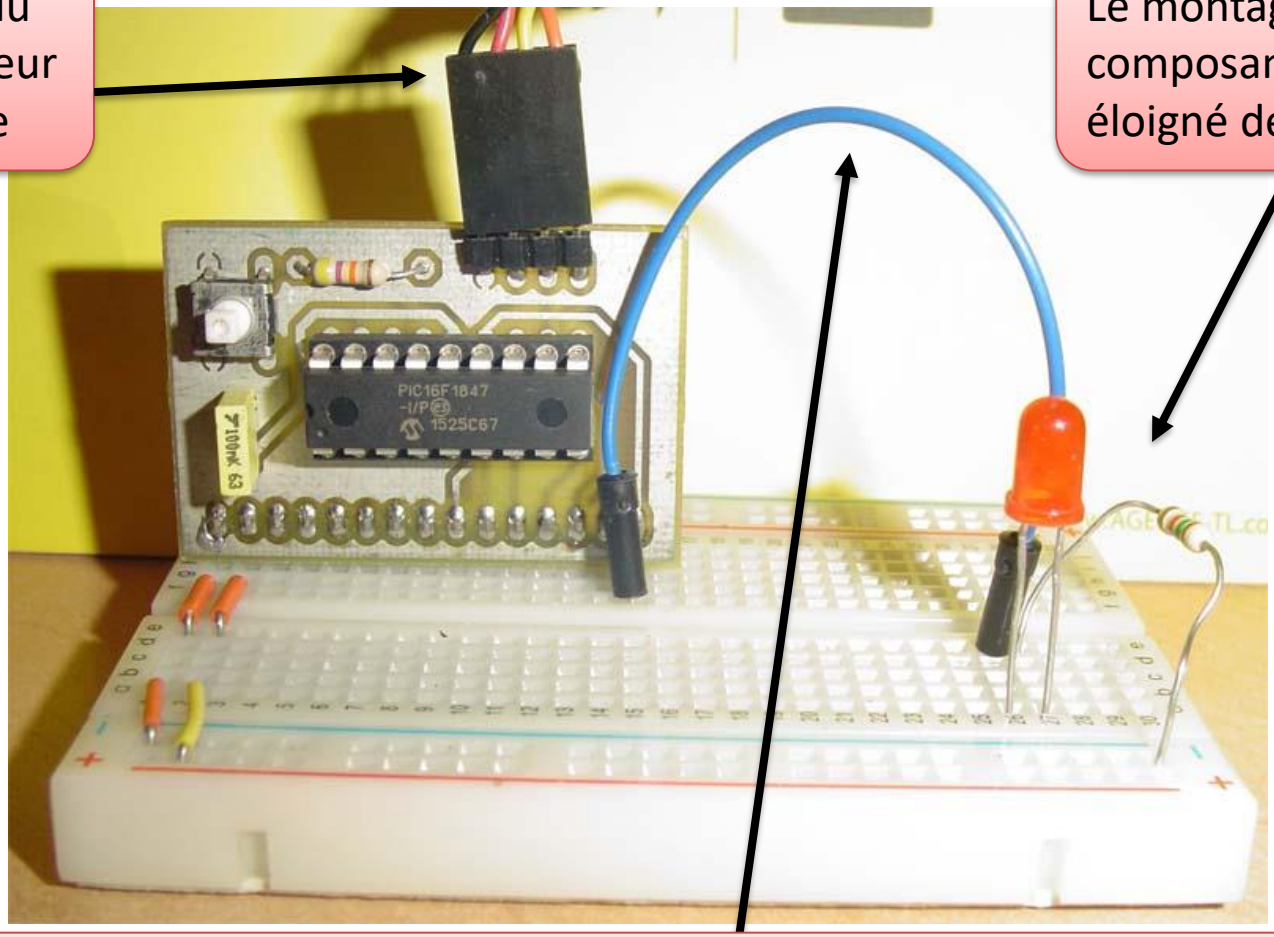


Carte d'étude à µC PIC 16F

- **Branchement d'une carte d'étude**

Fil noir du connecteur à gauche

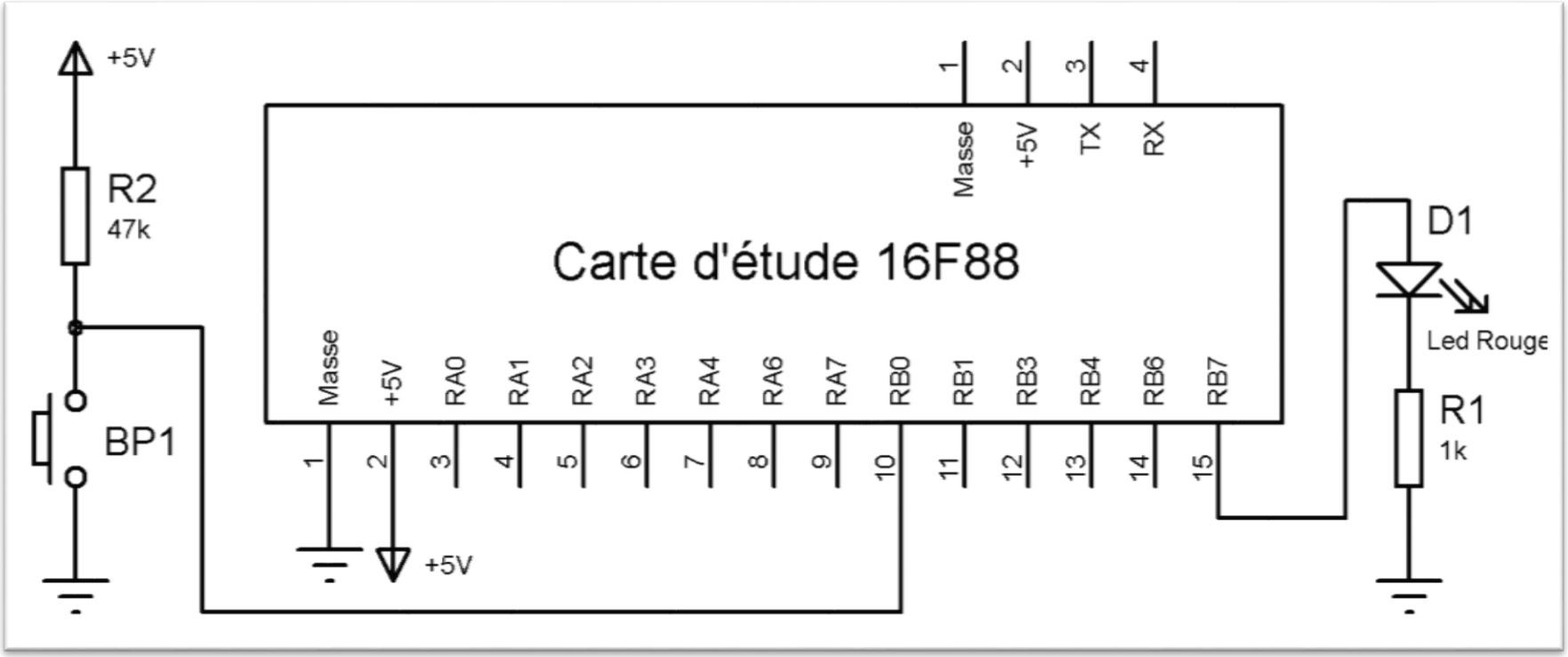
Le montage des composants est éloigné de la carte



Un fil isolé fait la liaison électrique entre la carte et les composants

Carte d'étude à µC PIC 16F

- Connecteur de la carte d'étude
- Branchement d'une Led sur la ligne RB7
- Branchement d'un bouton poussoir sur la ligne RB0



Compilateur XC8

- **Le compilateur XC8 traduit un texte en code machine**
- **Appel de la librairie principale**

#include <xc.h>

  - *Cette ligne permet d'utiliser les noms des registres internes du µC.*
  - *Le texte « PORTA » remplace le code 0x05*

- **Le compilateur C permet d'utiliser des fonctions communes**

If, else if, else, switch, case, break, while, …

Les structures, les fonctions, les variables, les constantes, …

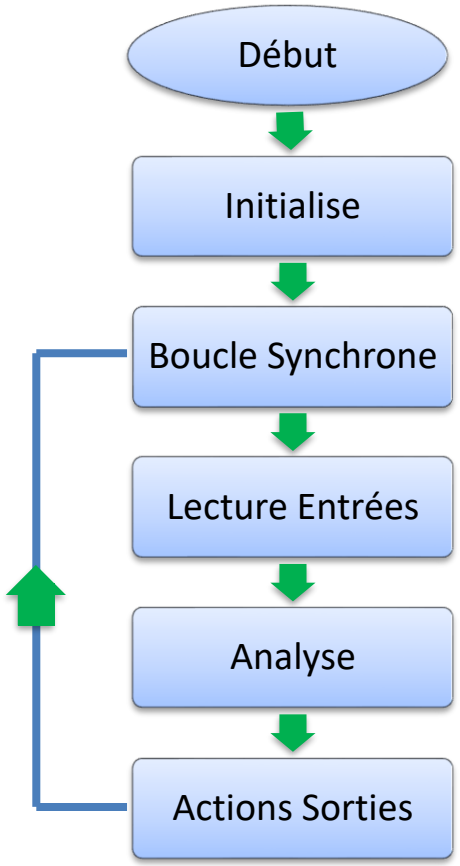
- **Le compilateur C permet d'associer d'autres librairies**

#include <stdio.h> fonction « printf »

#include "user.h" fonctions personnelles

Compilateur XC8

- **Organigramme d'un programme classique**



Chaque bloc représente une fonction organisée par la fonction main().

- Initialise : initialisation du µC, de ses périphériques utilisés et de ses variables.
- Boucle Synchrone : Boucle qui exécute le programme principal, à intervalle de temps régulier.
- Lecture Entrées : enregistre les informations externes et internes (registres)
- Analyse : Traite les informations reçues et prend des décisions.
- Actions Sorties : Met à jour les lignes en sorties du µC et les registres internes.

Exo 01 et 02 – Piloter une ligne en sortie

- **Configurer une ligne en sortie**

TRISB7 = 0;

  - *Cette ligne indique que la ligne RB7 du µC devient une sortie.*
  - *Ce code doit être écrit avant la boucle while (1)*

- **Écrire une valeur sur la ligne**

LATB7 = 1;

  - *Cette ligne indique que la ligne RB7 est mise à 1 (on applique du 5V).*

- **Utiliser des macros**
  - *Il est possible de remplacer ces lignes de code complexes par des textes plus simples et expressifs.*

#define DIR_DEL_ROUGE TRISB7

#define DEL_ROUGE LATB7

  - *Les macros doivent être déclarées dans un fichier séparé qui sera compilé avec le reste du programme.*

Opérateurs logiques et bit à bit

- **Les manipulations bit à bit sont très utilisées dans l'embarqué.**
- **Chaque action est associée à un registre.**
- **Il faut savoir mettre un bit à 1 ou à 0, sans modifier les autres.**

Opérateurs Logiques	Opérateurs bit à bit
|| représente l'opération OU	| représente l'opération OU
&& représente l'opération ET	& représente l'opération ET
! représente l'opération NON	^ représente l'opération OU exclusif
	~ représente l'opération NON
Attention : ces opérateurs considèrent les valeurs entières	***Attention*** : ces opérateurs se réalisent bit à bit

Masquage

- **Mise à 1 d'un bit dans un registre par masquage :**

Registre	b7	b6	b5	b4	b3	b2	b1	b0
Masque	?	?	?	?	?	?	?	?

Résultat	b7	b6	b5	b4	1	b2	b1	b0

- **Opérateur : ???**

Masquage

- **Mise à 1 d'un bit dans un registre par masquage :**

Registre	b7	b6	b5	b4	b3	b2	b1	b0
Masque	0	0	0	0	1	0	0	0

Résultat	b7	b6	b5	b4	1	b2	b1	b0

- **Opérateur : OU bit à bit**

```
Registre = Registre | 0b00001000
Registre = Registre | (1<<3)
Registre |= 0x08
```

Masquage

- **Mise à 0 d'un bit dans un registre par masquage :**

Registre	b7	b6	b5	b4	b3	b2	b1	b0
Masque	?	?	?	?	?	?	?	?

Résultat	b7	b6	0	b4	b3	b2	b1	b0

- **Opérateur : ???**

Masquage

- **Mise à 0 d'un bit dans un registre par masquage :**

Registre	b7	b6	b5	b4	b3	b2	b1	b0
Masque	1	1	0	1	1	1	1	1

Résultat	b7	b6	0	b4	b3	b2	b1	b0

- **Opérateur : ET bit à bit**

```
Registre = Registre & 0b11011111
Registre = Registre & ~(1<<5)
Registre &= 0xDF
```

Exo 02 – Générer une temporisation

- **Les fonctions de temporisations dépendent de la fréquence de l'horloge interne.**

```
#define _XTAL_FREQ 32000000
```

  - *Cette ligne indique que la fréquence du µC est de 32MHz.*
  - *Placer cette ligne avant le début du programme*

- **Générer une temporisation**

```
__delay_ms(200);
```

  - *Cette ligne attend 200ms avant de passer à l'instruction suivante.*

```
__delay_us(50);
```

  - *Cette ligne attend 50µs avant de passer à l'instruction suivante.*

```
_delay(10);
```

  - *Cette ligne attend 10 unités de temps (1 unité = 4 x 1/Fréq).*

Exo 03 – Piloter une ligne en entrée

- **Configurer une ligne en entrée**

TRISB0 = 1;

  - *Cette ligne indique que la ligne RB0 du µC devient une entrée.*
  - *Ce code doit être écrit avant la boucle while (1).*
  - *A la mise sous tension du µC (Reset) par défaut, toutes les lignes du µC sont en entrées.*

- **Lire l'état (0 ou 1) de la ligne**

valeur = RB0;

  - *Cette ligne indique que la valeur de la ligne RB0 est mémorisée dans la variable «Valeur».*
  - *Placer cette ligne de code dans la boucle infinie pour lire régulièrement son état.*
  - *La variable «Valeur» doit être déclarée avant le début du programme.*

Exo 04 – Lire une tension analogique

Registres :

- **ANSELA, ANSELB**

ANS<6:0> = 1 → Analog I

ANS<6:0> = 0 → Digital I/O

- **ADCON0**

CHS<2:0> = 0 → AN0

ADON = 1 → ADC actif

GO/DONE → Start/Finish

- **ADCON1**

ADFM = 0 → 8 bits

ADFM = 1 → 10 bits

VCFG<1:0> = 00 → Réf. Int.

ANSEL: ANALOG SELECT REGISTER (ADDRESS 9Bh) PIC16F88 DEVICES ONLY

U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0

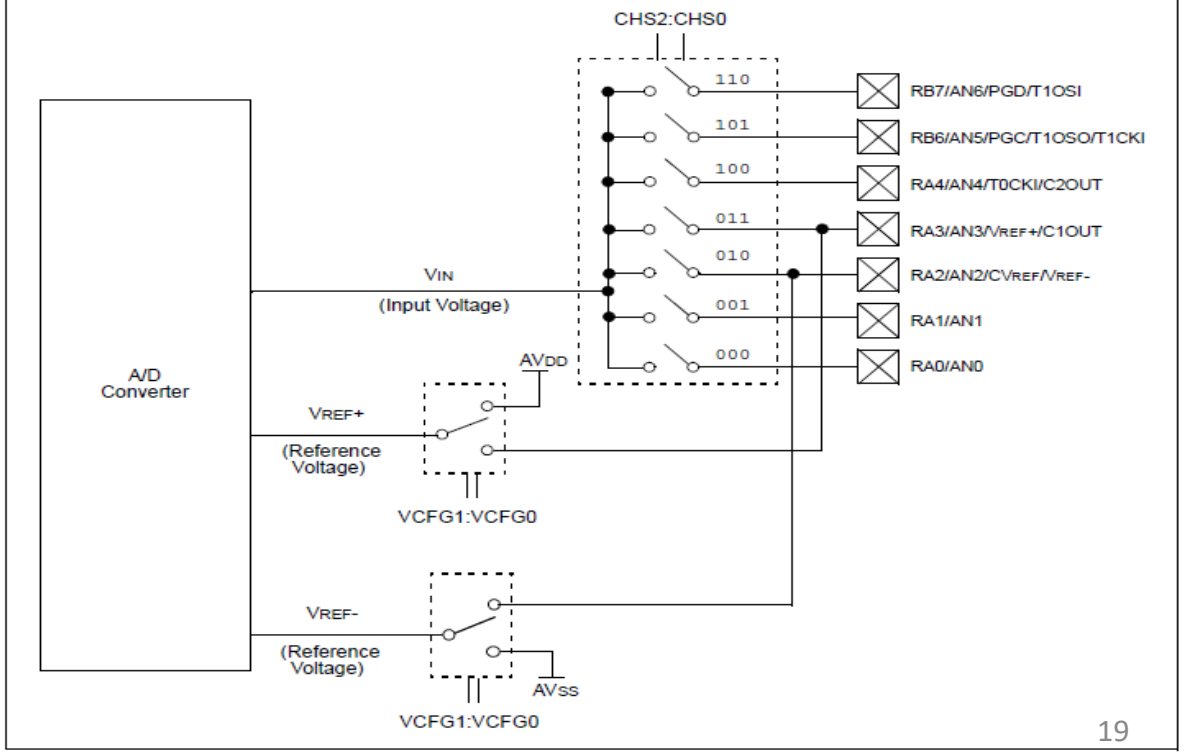
ADCON0: A/D CONTROL REGISTER (ADDRESS 1Fh) PIC16F88 DEVICES ONLY

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

ADCON1: A/D CONTROL REGISTER 1 (ADDRESS 9Fh) PIC16F88 DEVICES ONLY

R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0
ADFM	ADCS2	VCFG1	VCFG0	—	—	—	—

bit 7 bit 0



Exo 04 – Lire une valeur analogique

- **Paramétrer une ligne en entrée analogique**

Init_ADC(); *// Placer cette ligne de code avant la boucle while (1)*

  - *Cette ligne modifie les registres du convertisseur AD.*

- **Sélectionner la ligne à lire**

SelChanADC(0);

  - *La ligne AN0 est reliée au convertisseur ADC.*

- **Lire la valeur analogique**

Pot = Read_Adc_8bits();

  - *Le résultat de la conversion en 8bits est placée dans la variable Pot.*

Exo 04 – Lire une valeur analogique

- **Exemple : On veut mesurer la tension sur un accu 9Volts**
  - *Il n'est pas possible de mesurer directement une tension supérieure à la tension d'alimentation du µC. Il faut réduire par un pont diviseur.*
  - *Quelle précision, pouvons nous avoir ? (8bits ou 10bits). La plus petite valeur mesurable est la résolution du CAN, appelée aussi « le quantum ».*

8 bits → quantum $= \frac{5Volts}{2^8-1} = \frac{5}{255} = 19,6mV \approx 20mV$

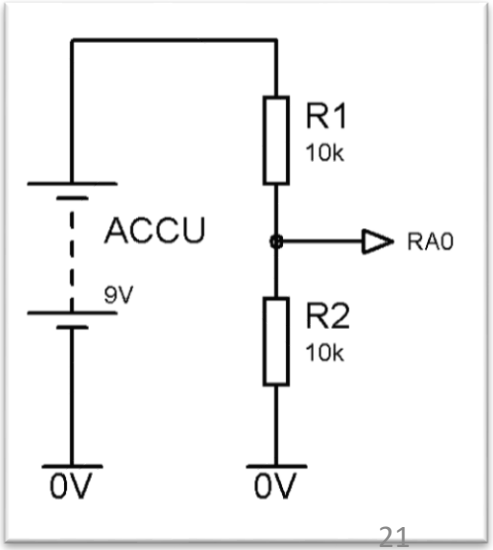
10 bits → quantum $= \frac{5Volts}{2^{10}-1} = \frac{5}{1023} = 4,9mV \approx 5mV$

$$Vaccu = 2 \times quantum \times Valeur\ numérique$$

↑ *pont diviseur par 2 ($R_1 = R_2$)*

Valeur num. en 8bits : $Val_{num} = \frac{9}{2 \times 0,02} = 225$

Valeur décimale : $Val_{déc} = Val_{num} \times 4 = 900$ ***(x10mV)***



Exo 05 – Utiliser un Timer

- **Ne plus utiliser les fonctions de temporisations**
  - **Une temporisation ne fait rien d'autres qu'attendre !**
  - **C'est une perte de temps inutile.**
  - **Il n'est pas possible de faire plusieurs actions en même temps.**
- **Utiliser les Timers à la place de délais**
  - **On paramètre un Timer pour générer un temps fixe de faible valeur (100µs à 10ms).**
  - **La boucle principale s'exécute pour ce temps fixe.**
  - **Un compteur s'incrémente, multiple de ce temps fixe.**
  - **Les actions sont rythmées par le compteur.**

Exo 05 – Configurer un Timer

Registres :

- **TIMER0**

Le compteur 8 bits

- **OPTION_REG**

TOCS = 0 → Horloge int. (F/4)

PSA = 0 → Prédiviseur actif

PS<2:0> = 0 → Choix prédiviseur

- **INTCON**

Registres des Interruptions

Le flag TMR0IF passe à 1 lorsque le compteur déborde.

255 + 1 = 0

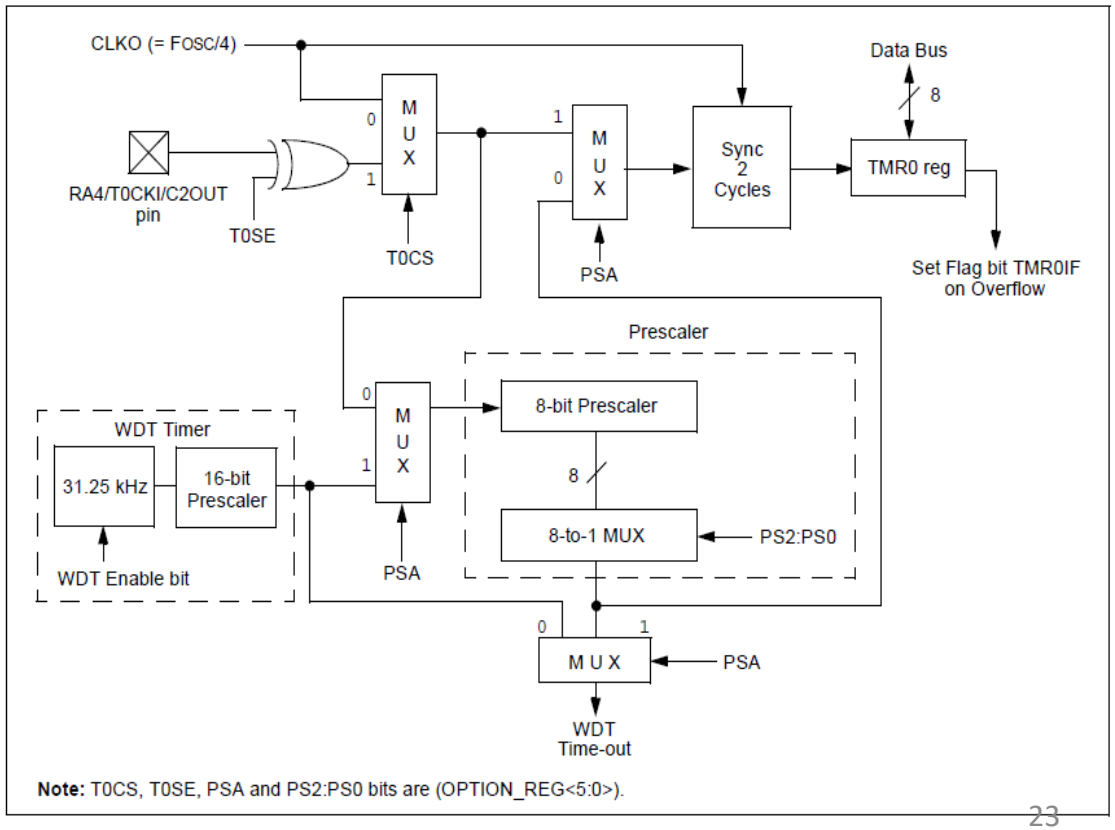
Dépassement de capacité ou la retenue de l'addition

TABLE 6-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
01h,101h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
81h,181h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by Timer0.

FIGURE 6-1: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER



Note: T0CS, T0SE, PSA and PS2:PS0 bits are (OPTION_REG<5:0>).

Exo 05 – Configurer un Timer

Créer une boucle qui s'exécute toutes les 1ms

- **La fonction « Synchronisation() » réalisera cette action**
- **Il faut paramétrer les registres associés au Timer0**

  Fréq. Horloge = 32Mhz → Fréq. Interne = 32/4=8Mhz → Temps cycle = 125ns

  Le timer0 est un compteur 8bits (256)

  1ms / 125ns = 8000 = 250 x 32 250 < 256

  On choisit un Prédiviseur par 32 →PS = 0b100

PS<2:0>: Prescaler Rate Select bits

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

- **Timer0**

  La fonction de synchronisation devra recharger le Timer

  TMR0 = 255 - 250

  En réalité, TMR0 = 0xFF – 249 → erreur de latence

- **Fonction : Synchronisation()**

```
while (!INTCONbits.TMR0IF);      // Attente du débordement du Timer0
TMR0 = TMR0_VALUE;               // Rechargement du Timer0
INTCONbits.TMR0IF = 0;           // Effacement du "flag"
```

Exo 05 – Utilisation d'un Timer

Gestion d'un compteur basé sur un Timer

- **La fonction Synchronisation() permet à la boucle infinie while(1) d'être parcourue toutes les 1 ms.**

```
compteur++;                    // Incrémentation du compteur
if(compteur >= periode)        // Choix de la période
        compteur = 0;
if(valeur > compteur)          // Comparaison
        Action = 1;            // Action
else                           // Facultatif
        Action = 0;
```

Exo 06 – réaliser un dialogue entre la cible et le PC

Paramétrer les registres associés

C'est la fonction Init_UART() qui gère cette tâche.

- **Les 2 lignes TX et RX sont choisies et orientées (Sortie et Entrée)**
- **La vitesse du dialogue est choisie à 57600 baud (nbre de bits par sec.) SPBRG = 138 soit 32M/(57600*4)-1**
- **Le registre d'émission TXSTA = 0x24 (TXEN et BRGH)**
- **Le registre de réception RCSTA = 0x90 (CREN et SPEN)**
- **Le registre contrôle du baud BAUDCONbits.BRG16 = 1**

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
BAUDCON					BRG16			
SPBRGL								
SPBRGH								

Exo 06 – réaliser un dialogue entre la cible et le PC

Envoi d'un caractère

- **Ecrire dans le registre : TXREG**

Lecture d'un caractère

- **Lire le registre : RCREG**

Envoi d'un texte

- **printf ("\tBonjour\r\n");**

  \r → Retour Charriot (code ASCII : 0x0D)
  \n → A la ligne (code ASCII : 0x0A)
  \t → une tabulation (code ASCII : 0x09)

- **printf ("Valeur = %d\r\n",var);**

  Cette ligne affiche "Valeur = " et le contenu de la variable var.

Exo 06 – réaliser un dialogue entre la cible et le PC

Lecture d'un texte

- **Cette information n'est pas prévisible par le programme.**
  - **Elle est asynchrone.**
- **Il faut interrompre le programme lire les caractères reçus.**
  - **Mais il faut que cela soit très rapide.**
- **Il faut savoir quand le message envoyé est fini.**
- **Il faut indiquer au programme principal qu'un message est arrivé.**

Interruptions

- **Lorsqu'un évènement (matériel ou logiciel) intervient, il peut générer une interruption du programme, si le µC est correctement paramétré.**
- **L'interruption de l'exécution du programme principale entraine l'exécution d'une routine très courte. Puis le programme reprend là ou il a été interrompu.**

Exo 06 – réaliser un dialogue entre la cible et le PC

- **Initialiser les registres UART et les interruptions**

Placer les 2 lignes suivantes avant la boucle while (1)

Init_UART();

- *Paramètre les registres UART*

Init_Interruptions();

- *Autorise les interruptions dont l'interruption UART*

- **Vérifie qu'un nouveau message a été reçu**

if(Data_Rdy_UART() == 1)

- *Renvoi 1 si un nouveau message a été reçu, sinon 0*

- **Lit le message**

Read_Data_UART(command);

- *Le message reçu est enregistré dans le tableau command.*

Exo 06 – réaliser un dialogue entre la cible et le PC

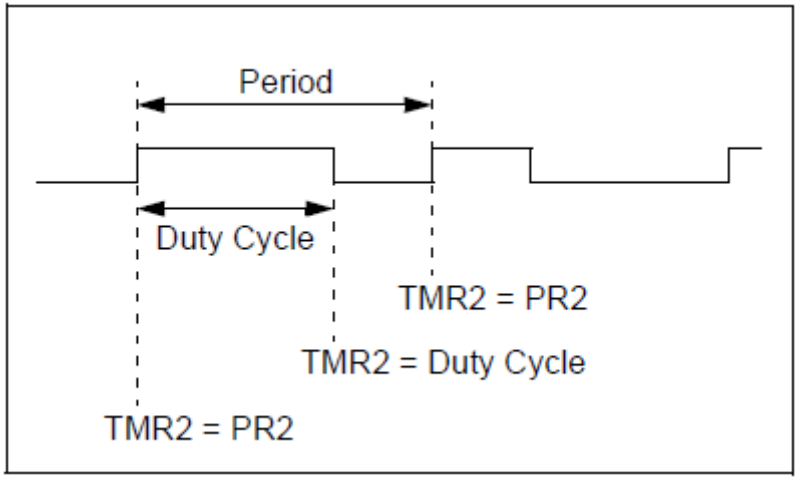
- **Gestion d'un texte reçu, par interruption**

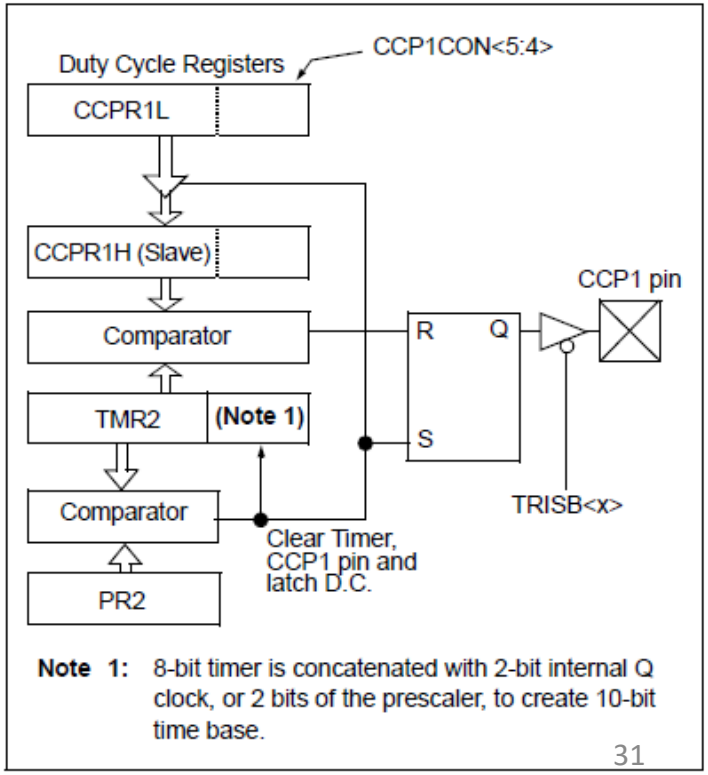
```
if (RCIF)                                   // Interruption par réception d'un caractère
{
        buffer[ptr] = RCREG;                // Enregistre le caractère reçu dans un tableau
                                            // La lecture du registre remet à 0 le flag RCIF

                                            // Fin de ligne ou Retour charriot
        if ((buffer[ptr] == 10) || (buffer[ptr] == 13))
        {
                buffer[ptr] = 0;            // Remplace le caractère par 0
                ptr = 0;                    // Met le pointeur à 0
                buffer_ok = 1;              // Met l'indicateur de message reçu à 1
        }
        else
        {
                ptr++;                      // Incrémente le pointeur
        }
}
```

Exo 07 – Génération d'un signal PWM (MLI)

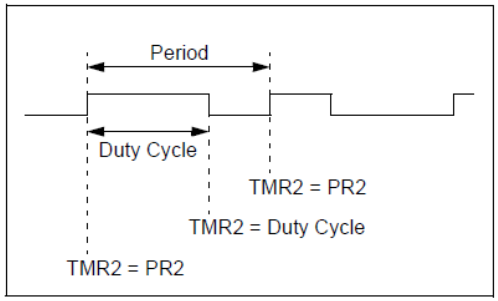
- **Modification de la vitesse d'un moteur** **(ou l'intensité dans une Del)**
- **Variation de la tension moyenne en pulsant la tension d'alimentation**
- **Le moteur (ou l'œil) possède un temps de réponse**
- **Générer une tension variable avec un filtre RC**
- **Générer des signaux spéciaux**

- **Initialisation des registres CCP**
- **Choix de la période PR2**





Exo 07 – Génération d'un signal PWM (MLI)



- **Choisir la période (ou la fréquence)**
  - **Moteur DC avec constante de temps τ=L/R ≈ 1ms**
  - **Période < 1ms (fréq. > 1kHz) → audible (20Hz -20kHz)**
  - **Rapport cyclique de 0 à 100%**
  - **Fréquence cycle 8MHz (32MHz/4) → résolution 125ns**
  - **5kHz → 200µs → PR2 = 99 → Pré-diviseur = 16**
  - **T2CON Registre de contrôle du Timer2**

T2CKPS<1:0> = 0b10 → Pré-diviseur par 16

TMR2ON = 1 → Timer2 activé

$$\text{PWM Period} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

- **CCPxCON Registre de mode → mode PWM → CCPxM<3:0> = 0b00001100**
- **Donner un rapport cyclique à la volée → CCPRx = 50 → 50% dans ce cas**

11h	TMR2	Timer2 Module Register								0000 0000	0000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000