

Circuits numériques programmables

AP4 S8 2022-2023 BX

Email: antoine.pirog@junia.com

Plan du cours

I. Le microcontrôleur et ses périphériques

- I. Rappels : Généralités sur les microcontrôleurs et le langage C
- II. Configuration d'un microncontrôleur

II. Le FPGA

- I. Architecture des circuits logiques programmables
- II. Le langage de description matériel VHDL

Organisation du cours

- **Module de circuits numériques programmables AP4**
(65% de l'UE Systèmes électroniques et Télécommunications)

SÉANCES DE COURS	
	Volume horaire
Cours/TD	10h (6h + 4h)
TP	42h (18h30 + 23h30)

EVALUATION			
	Durée	Nombre	Pondération
DS	1h30	2	20% + 30%
TP/Projet	3h - 3h30	6	40%
TP	3h30 – 4h	6	10%
TOTAL			100%

Organisation du cours

- **Module de circuits numériques programmables AP4**
(65% de l'UE Systèmes électroniques et Télécommunications)
- 1^{ère} partie : Microcontrôleurs

SÉANCES DE COURS		
	Nombre de séances	Volume horaire
Cours/TD	En début de TP	6
TP	6	18h30

EVALUATION			
	Durée	Nombre	Pondération
DS	1h30	1	20%
TP	3h - 3h30	6	40%
TOTAL			60%

Rappels

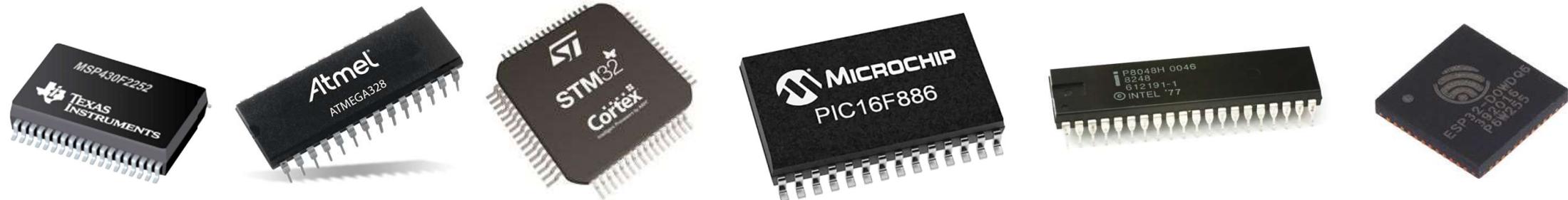
- 1) Généralités sur les microcontrôleurs
- 2) Le langage C pour les microcontrôleurs

Rappels

- 1) Généralités sur les microcontrôleurs**
- 2) Le langage C pour les microcontrôleurs

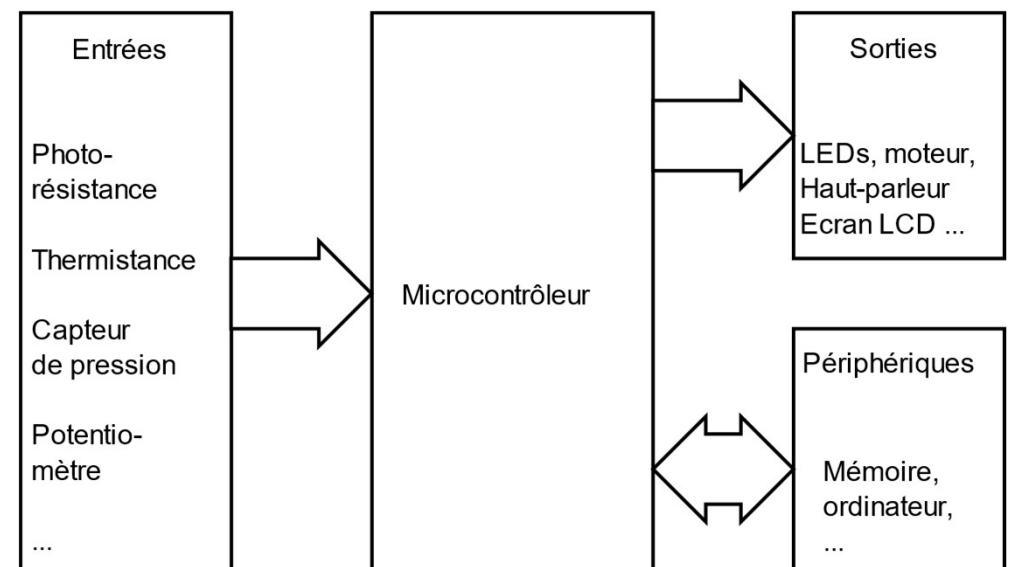
Principaux fondeurs

- Microcontrôleurs
 - Microchip (PIC10, PIC12, PIC16, PIC24, PIC32)
 - STMicroelectronics (ST6/7/8/10, STM32)
 - Atmel (AT89, ATtiny, ATmega, AVR32...)
 - Espressif (ESP32, ESP82...)
 - Intel (MCS-48/51/151/251/96...)
 - Analog devices (Blackfin, SHARC, ADSP...)
 - Texas instruments (TMS370, MSP430...)



Le microcontrôleur

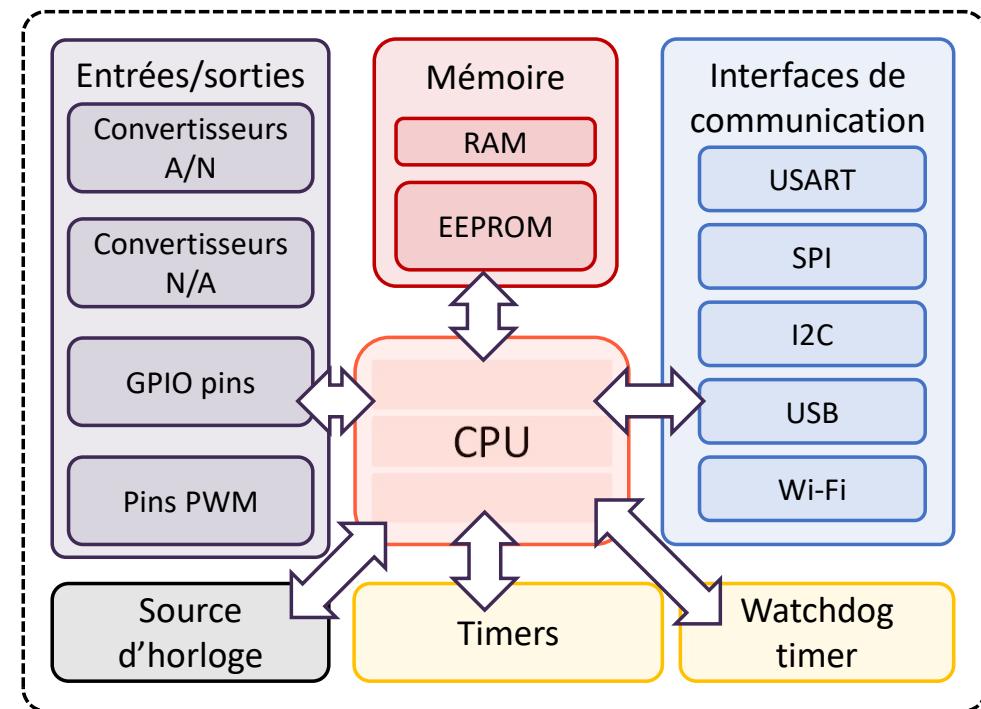
- Un microcontrôleur est un micro-ordinateur complet, composé d'**un microprocesseur et d'organes matériels sur une même puce**, et structuré comme un système à microprocesseurs
- Les entrées/sorties sont dédiées à la communication directe avec des systèmes périphériques
- Les entrées et sorties sont tant logiques qu'analogiques



Anatomie d'un microcontrôleur

On trouve notamment dans les microcontrôleurs:

- Un **processeur** (CPU)
- Une **mémoire programme** (EPROM, EEPROM ...)
- Une **mémoire vive** (RAM)
- Une **horloge** (oscillateur)
- Des **timers**
- Des **convertisseurs A/N et N/A**
- Des **GPIO** (General Purpose Input/Output)
- Des **sorties PWM**
- Des **interfaces de communication** (série, SPI, ...)



... Sur la même puce !

Pourquoi utiliser un microcontrôleur ?

- Piloter des composants externes
 - Entrées / sorties simples, asynchrones
 - Entrées / sorties complexes (gérées par des modules spécifiques):
 - Interfaces de communication (UART, SPI, I2C, etc.)
 - Convertisseurs (CAN/CNA)
- Développement « logiciel »
 - Fluidité de développement du langage C
 - Prise de décision
 - Machines d'états complexes

Registres d'un microcontrôleur

- Les **registres** dans un microcontrôleur sont des emplacements internes de mémoire dont l'accès est extrêmement rapide
- On distinguera différent rôles des registres d'un microcontrôleur:
 - Les registres **spécialisés**
 - Fonctionnement interne du CPU
 - Configuration des modules
 - Fonctionnement des modules
 - Entrées/sorties
 - Des registres **généraux**
 - Stockage temporaire de données
 - Stockage temporaire d'adresses
- Les registres sont accessibles en lecture et/ou écriture par :
 - Une adresse au niveau matériel
 - Un nom au niveau utilisateur

Configuration des périphériques

- La configuration des différents périphériques se fait via des **registres**
- Chaque registre contient une valeur de taille **1 octet** (8 bits) :
 - Il faut identifier le rôle de chaque bit dans la documentation du registre
 - Soit chaque bit est individuellement un paramètre vrai/faux (0/1)
 - Soit la valeur du registre est découpée en plusieurs paramètres de [1,2,3,4,5,6,7] bits
 - Soit la valeur du registre est un paramètre unique entre 0 et 255
 - Soit la valeur du registre est un fragment d'un paramètre entre 0 et >255
 - Certains bits peuvent être inutilisés

REGISTER 11-10: TRISB: PORTB TRI-STATE REGISTER

| R/W-1/1 |
|---------|---------|---------|---------|---------|---------|---------|---------|
| TRISB7 | TRISB6 | TRISB5 | TRISB4 | TRISB3 | TRISB2 | TRISB1 | TRISB0 |
| bit 7 | | | | | | | bit 0 |

REGISTER 27-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	U-0	R/W-0/u
TMR1CS<1:0>		T1CKPS<1:0>		T1OSCEN	T1SYNC	—	TMR1ON
bit 7							bit 0

Attention : Les registres sont volatils (effacés à chaque mise hors tension)

Configuration des périphériques - exemple

REGISTER 27-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	U-0	R/W-0/u
TMR1CS<1:0>	T1CKPS<1:0>	T1OSCEN	<u>T1SYNC</u>	—	—	TMR1ON	
bit 7							bit 0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
 u = Bit is unchanged x = Bit is unknown -n/n = Value at POR and BOR/Value at all other Resets
 '1' = Bit is set '0' = Bit is cleared

bit 7-6 **TMR1CS<1:0>**: Timer1 Clock Source Select bits

- 11 = LFINTOSC
- 10 = Timer1 clock source is pin or oscillator:
 - If T1OSCEN = 0:
External clock from T1CKI pin (on the rising edge)
 - If T1OSCEN = 1:
Crystal oscillator on SOSCI/SOSCO pins
- 01 = Timer1 clock source is system clock (FOSC)
- 00 = Timer1 clock source is instruction clock (Fosc/4)

bit 5-4 **T1CKPS<1:0>**: Timer1 Input Clock Prescale Select bits

- 11 = 1:8 Prescale value
- 10 = 1:4 Prescale value
- 01 = 1:2 Prescale value
- 00 = 1:1 Prescale value

bit 3 **T1OSCEN**: LP Oscillator Enable Control bit

- 1 = Dedicated secondary oscillator circuit enabled
- 0 = Dedicated secondary oscillator circuit disabled

bit 2 **T1SYNC**: Timer1 Synchronization Control bit

- 1 = Do not synchronize asynchronous clock input
- 0 = Synchronize asynchronous clock input with system clock (FOSC)

bit 1 **Unimplemented**: Read as '0'

bit 0 **TMR1ON**: Timer1 On bit

- 1 = Enables Timer1
- 0 = Stops Timer1 and clears Timer1 gate flip-flop

Rappels

- 1) Généralités sur les microcontrôleurs
- 2) Le langage C pour les microcontrôleurs**

Introduction au langage C pour les microcontrôleurs

La syntaxe du langage C utilisée en microcontrôleurs est **identique à celle vue en cours de C ***

* À quelques exception près, spécifiques au système embarqué cible (routines d'interruption, accès mémoire)

Les différences entre le langage C (vu l'an dernier) sur un ordinateur et le langage C embarqué (vu dans ce cours) proviennent de la différence de **cible** :

- Langage C sur **ordinateur** :

- Manipulation de données
- Manipulation de fichiers
- Interactions avec la console
- Portabilité du code

- Langage C **embarqué** (pour **microcontrôleur**) :

- Manipulation de données
- Manipulation bas-niveau de mémoire
- Spécifique à la cible

Structure d'un programme en C

```

/*
 * File:    main.c
 * Author:  antoine.pirog
 *
 * Demo program for embedded C structure
 */

#include "configbits.h"
#include <xc.h>

#define LED1 LATDbits.LATD0
#define LED2 LATDbits.LATD1
#define DIR_LED1 TRISDbits.TRISD0
#define DIR_LED2 TRISDbits.TRISD1

const int delay_cycles = 500; // Nb of delay cycles

void main(void) {
    init_leds();

    while(1){          // Inf. loop : Blink LED1 & LED2
        LED1 = 1;       // LED1 ON
        LED2 = 0;       // LED2 OFF
        delay_approx(); // wait
        LED1 = 0;       // LED1 OFF
        LED2 = 1;       // LED2 ON
        delay_approx(); // wait
    }
}

void init_leds(){
    DIR_LED1 = 0; LED1 = 0; // LED1 : output, OFF
    DIR_LED2 = 0; LED2 = 0; // LED2 : output, OFF
}

void delay_approx(){
    for(int i=0; i<delay_cycles; i++)
}

```

Section dédiée à la **documentation**

Section dédiée aux **inclusions**

Section dédiée aux **définitions** :

Constantes, macro-fonctions...

Section dédiée aux **déclarations globales**

Section dédiée au **programme principal**

Note 1 : en embarqué, la fonction main ne prend pas d'argument et ne renvoie rien

Note 2 : Le programme principal est contenu dans une boucle infinie (le programme ne se termine jamais)

Section dédiée aux **sous-programmes**

Ecrire dans un registre

Certains bits de registres sont accessibles en écriture (indiqué par un « W » dans la documentation).

On pourra utiliser les syntaxes suivantes pour écrire dans ces registres:

Note : certaines syntaxes requièrent également l'accès en lecture (marqué par un « R »)

Ecriture d'un registre entier (3 syntaxes équivalentes)

```
LATB = 0xC2; // -----
LATB = 194; // Mise à "1100 0010" de LATB
LATB = (3 << 6) + (1 << 1); // -----
```

Mise à 1 d'un bit seul (3 syntaxes équivalentes)

```
LATB |= (1 << 5); // -----
LATB |= 0x20; // Mise à 1 du bit #5 de LATB
LATBbits.LATB5 = 1; // -----
```

Mise à 0 d'un bit seul (3 syntaxes équivalentes)

```
LATB &= ~(1 << 3); // -----
LATB &= ~0x08; // Mise à 0 du bit #3 de LATB
LATBbits.LATB3 = 0; // -----
```

Inversion d'un bit seul

```
LATB ^= 0x40; // Inversion du bit #6 de LATB
```

Sous MPLABX, on peut utiliser la syntaxe **NOM_DU_REGISTREbits.NOM_DES_BITS** pour accéder facilement à un bit ou une plage nommée de bits d'un registre. On devra se référer à la documentation et/ou à l'autocomplétion de MPLABX.

Lire un registre

Certains bits de registres sont accessibles en lecture (indiqué par un « R » dans la documentation).

On pourra utiliser les syntaxes illustrées par les exemples suivants pour lire dans ces registres:

Lecture d'un registre entier

```
char x = PORTB;           // Lecture du register entier
if (ADRESH > 31) {/* ... */} // Test sur un register entier
```

Lecture d'un bit seul

```
if (PORTB & 0x80) {/* ... */} // Test par masquage du bit #7
if (PORTBbits.RB5) {/* ... */} // Test sur un bit nommé
```

Lecture d'une plage de bits

```
char x = (PORTA & 0x07);    // Masquage des 3 bits de poids faible
char y = (PORTA & 0xF0) >> 4; // Masquage des 4 bits de poids fort
char y = OPTION_REG.PS        // Lecture d'une plage nommée
```

Manipulation de registres - Exemples

Etat initial du registre :

TRISB	0	0	1	1	0	0	0	0
--------------	---	---	---	---	---	---	---	---

```
TRISB |= 0x05;
```

Etat final du registre ?

TRISB								
--------------	--	--	--	--	--	--	--	--

Etat initial du registre :

T1CON	1	1	1	1	1	1	1	1
--------------	---	---	---	---	---	---	---	---

```
T1CON = (1 << 6) + 1;
```

Etat final du registre ?

T1CON								
--------------	--	--	--	--	--	--	--	--

Etat initial du registre :

LATA	1	0	0	0	0	1	0	0
-------------	---	---	---	---	---	---	---	---

```
LATA |= 15;
```

```
LATA ^= 128;
```

Etat final du registre ?

LATA								
-------------	--	--	--	--	--	--	--	--

Etat initial du registre :

LATC	1	1	0	1	0	1	1	0
-------------	---	---	---	---	---	---	---	---

```
LATC &= ~(0x80 | (0x03 << 2));
```

Etat final du registre ?

LATC								
-------------	--	--	--	--	--	--	--	--

Configuration d'un microcontrôleur

Cas pratique du PIC 16F171x

Déroulé du cours

- On verra dans un premier temps :
 - Des éléments généraux de **syntaxe en C** pour les microcontrôleurs
 - Des généralités sur le **fonctionnement et la configuration d'un microcontrôleur**, illustrées par le modèle de µC **PIC16F1719**
 - Les prérequis pour **lire et comprendre la documentation technique**
 - Les **exemples fondamentaux de périphériques**
- On apportera ensuite des compléments plus pointus à mesure que les séances de TP progressent.

Sources bibliographique du chapitre

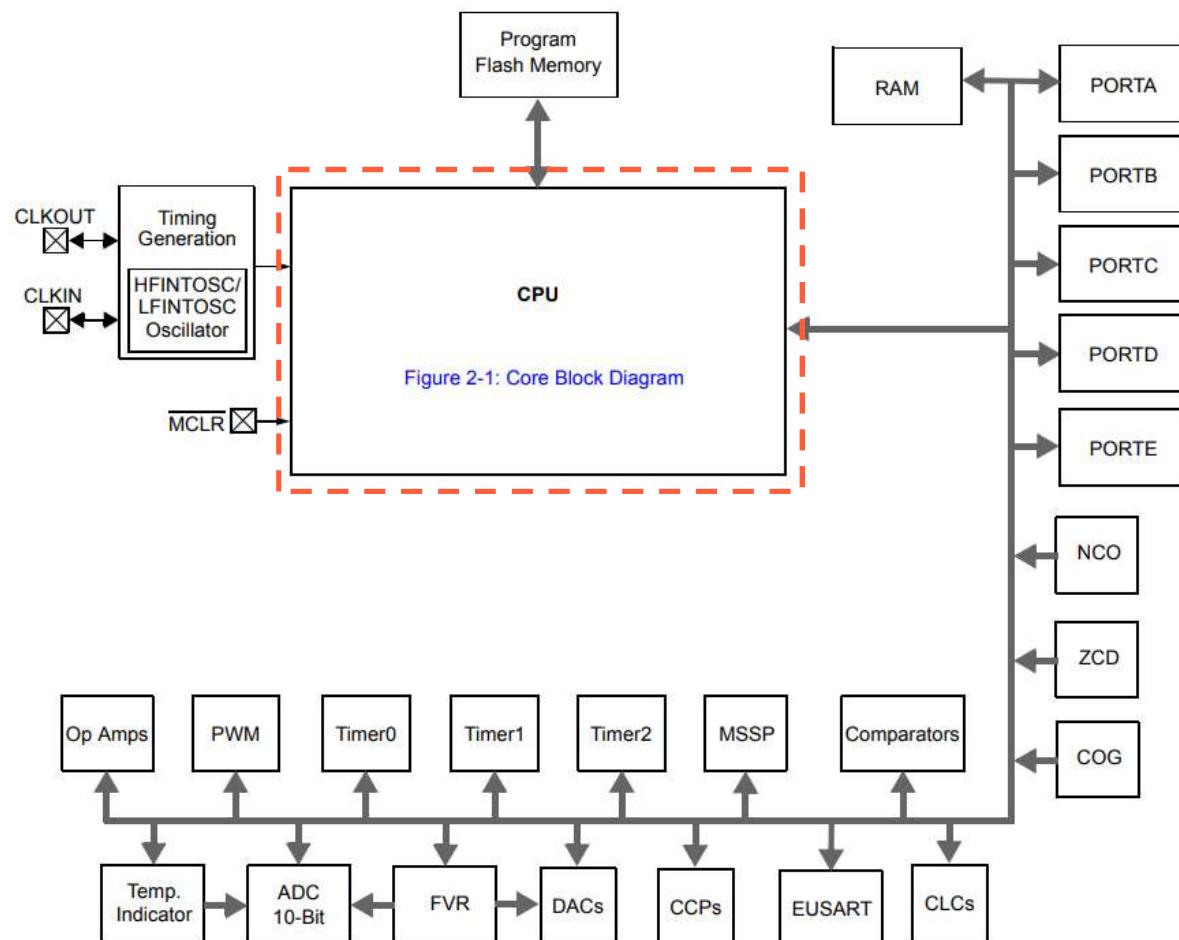
- [1] Fiche technique de la famille de microcontrôleurs PIC16(L)F1717/8/9
http://ww1.microchip.com/downloads/en/DeviceDoc/PIC16F1717_8_9-data-sheet-40001740C.pdf
- [2] Microchip Developer Help
<http://microchipdeveloper.com/8bit:peripherals>
- [3] MPLABX XC8 User guide
<https://ww1.microchip.com/downloads/en/devicedoc/50002053g.pdf>

Documentations techniques disponibles dans :

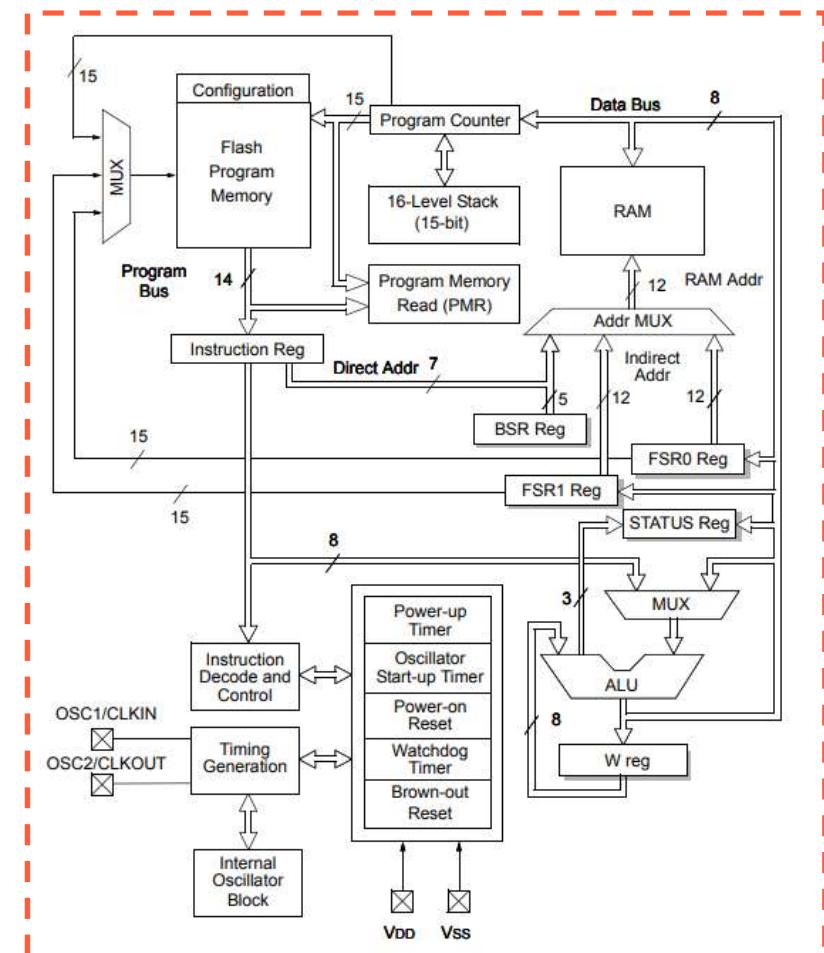
Junia-learning FPGA & Microcontrôleurs > Microcontrôleurs > TPs > Ressources documentaires

En cas de doute, le sommaire de [1] se trouve p. 10 ; la documentation détaillée de tous les périphériques y est accessible.
La référence [2] comprend un sommaire vers une documentation agrémentée d'exemples

PIC16F1717/9 – schéma-blocs



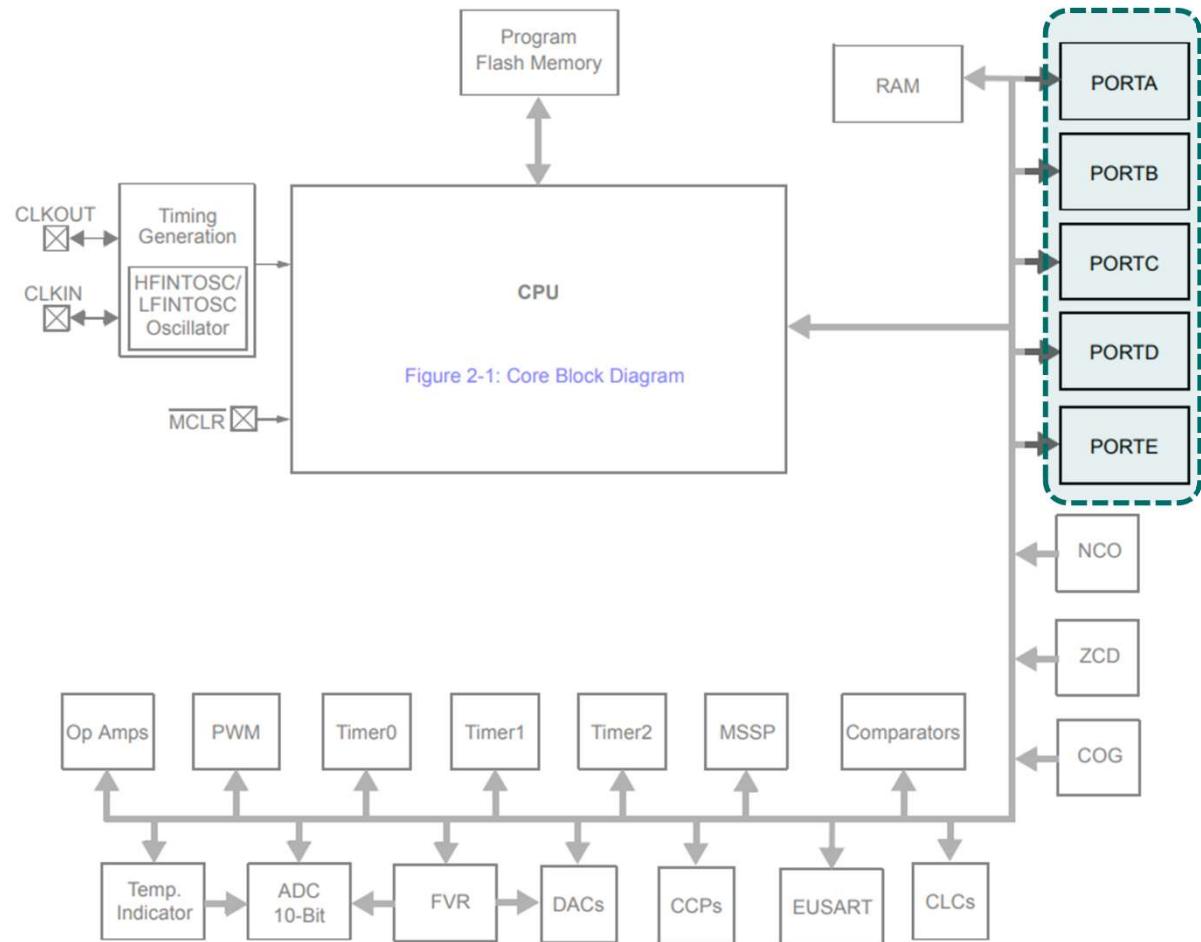
[1] Section 1 – Device overview (p. 15)



[1] Section 1 – Device overview (p. 23)

Les GPIO

[1] section 11 (p. 123)



Les Generic Purpose I/O ports (GPIO)

- Les GPIO sont les **périphériques** les plus simples
- Ce sont des broches génériques **configurables en entrées/sorties** vers d'autres systèmes
- Les GPIO sont regroupées par 8 par **port**
- Le PIC16F1719 a à disposition les **ports A, B, C, D, E**
[1] Section 11, Tableau 11-1 p. 123

Attention : certaines GPIO partageront (par multiplexage) des fonctions dédiées qui leur sont propres ;
Par ex. sur le PIC16F1719, PORTA0-PORTA3 sont également les entrées analogiques AN0-AN3 des ADC ;
La stratégie de multiplexage « n'importe quelle fonction sur n'importe quelle broche » est extrêmement rare,
et les broches doivent être judicieusement choisies

Configuration des GPIO

Chaque port a 6 registres de configuration :

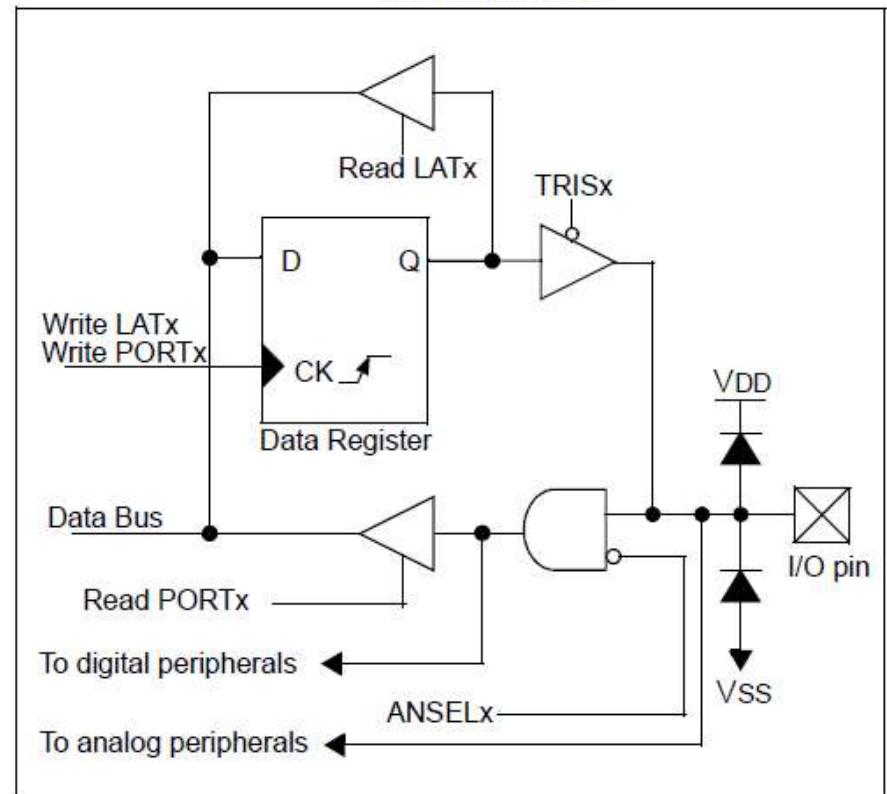
- **TRISx** (direction du port)
- **PORTx** (lecture des niveaux logiques sur les pins)
- **LATx** (latch de sortie)
- **INLVLx** (contrôle du niveau d'entrée)
- **ODCONx** (drain ouvert)
- **SLRCONx** (slew rate)

Certains ports peuvent avoir des registres de configuration supplémentaires :

- **ANSELx** (sélection du mode analogique)
- **WPUx** (weak pull-up)

Chacun des **8 bits** de chaque registre configue l'une des **8 pins** de chaque port

FIGURE 11-1: GENERIC I/O PORT OPERATION



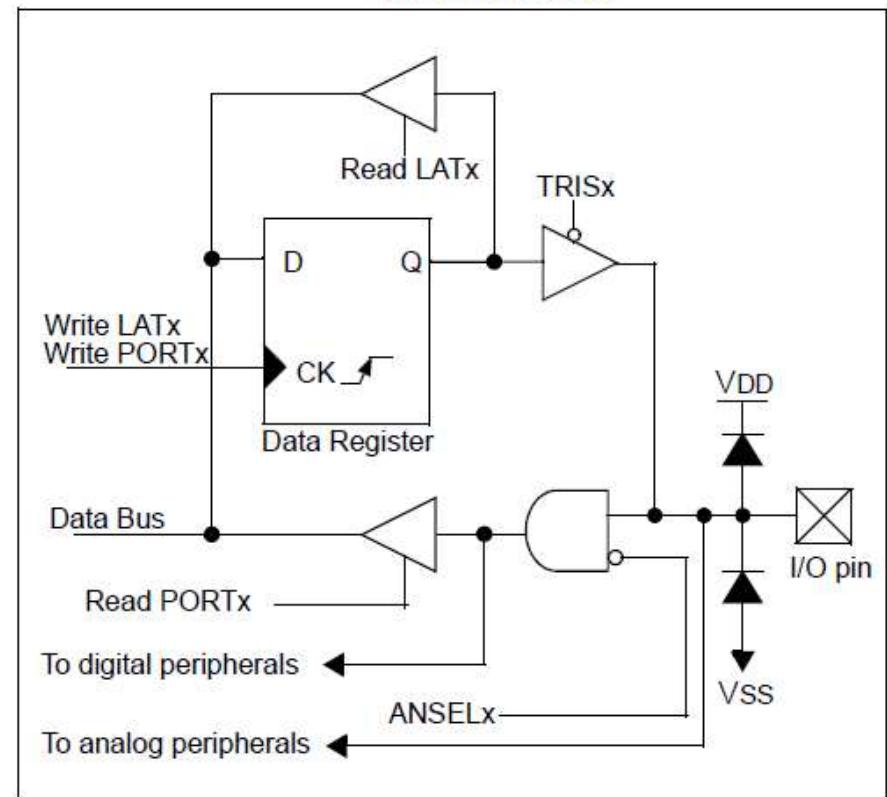
[1] Section 11.0 – I/O ports (p. 123)

Lecture / écriture sur les GPIO avec PORTx et LATx

	En lecture	En écriture
PORTx	Lecture du niveau logique sur les broches du port	Ecriture dans la latch de sortie du port (« Data Register »)
LATx	Lecture du niveau logique mémorisé dans la latch de sortie (« Data Register »)	

- Dans la pratique on pourra se tenir à la règle :
 - Utiliser **LATx pour l'écriture** de valeurs
 - Utiliser **PORTx pour la lecture** de valeur

FIGURE 11-1: GENERIC I/O PORT OPERATION



[1] Section 11.0 – I/O ports (p. 123)

Configuration des GPIO

Configuration du mode entrée (1) /sortie (0)

```
TRISBbits.TRISB3 = 0; // PORTB3 est une sortie
TRISBbits.TRISB5 = 1; // PORTB5 est une entrée
```

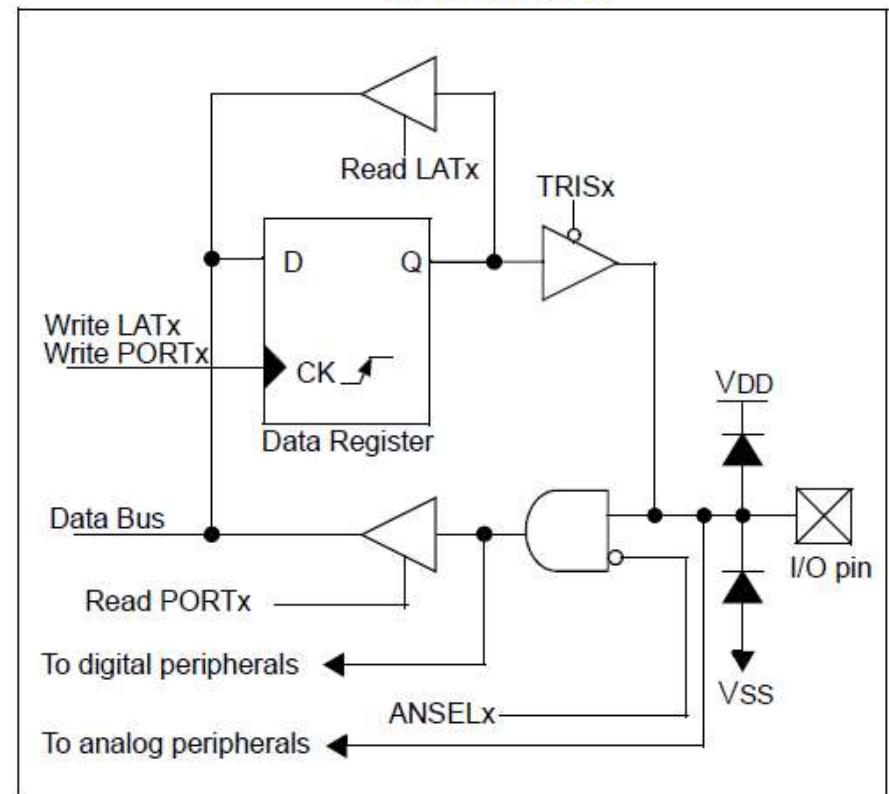
Mise à l'état haut (1) / bas (0) d'une sortie

```
LATBbits.LATB3 = 0; // Mise à l'état bas de PORTB3
LATBbits.LATB3 = 1; // Mise à l'état haut de PORTB3
```

Lecture d'une entrée numérique (test sur un bit)

```
if (PORTBbits.RB5) {/* ... */} // Test si PORTB5 == 1
if (!PORTBbits.RB5) {/* ... */} // Test si PORTB5 == 0
```

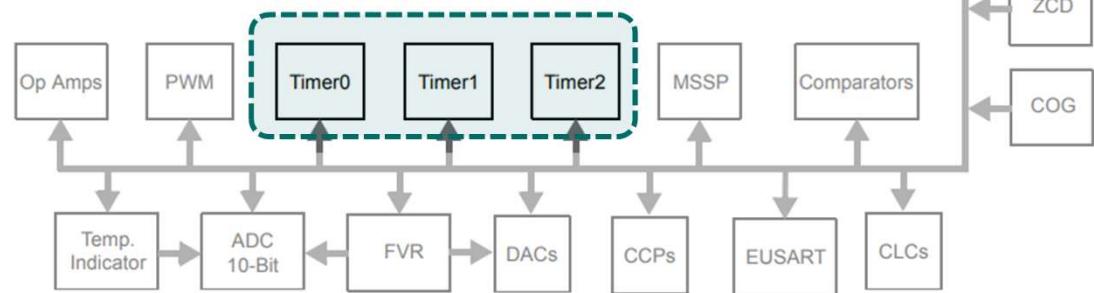
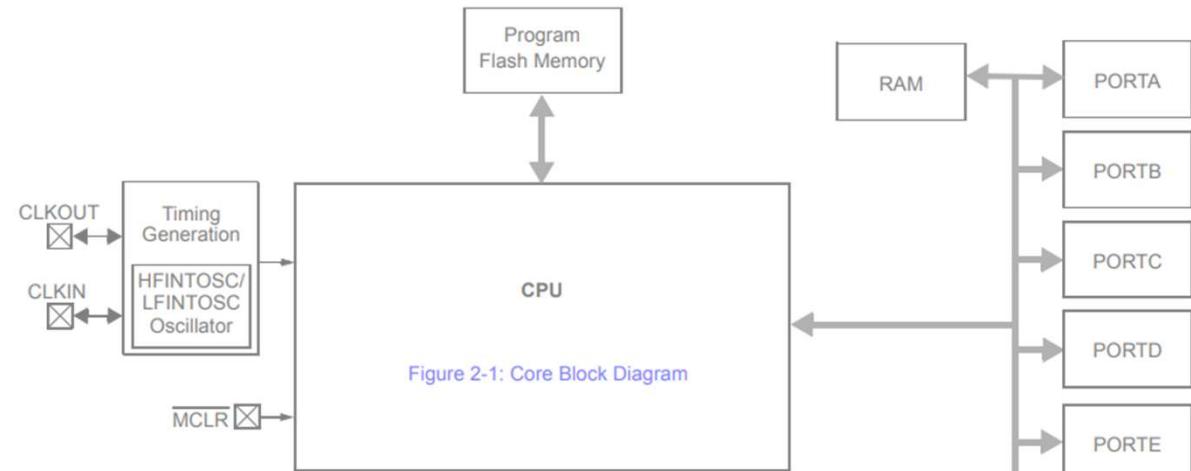
FIGURE 11-1: GENERIC I/O PORT OPERATION



[1] Section 11.0 – I/O ports (p. 123)

Les Timers

[1] Sections 26 (p. 271),
27 (p. 274), 28 (p. 285)



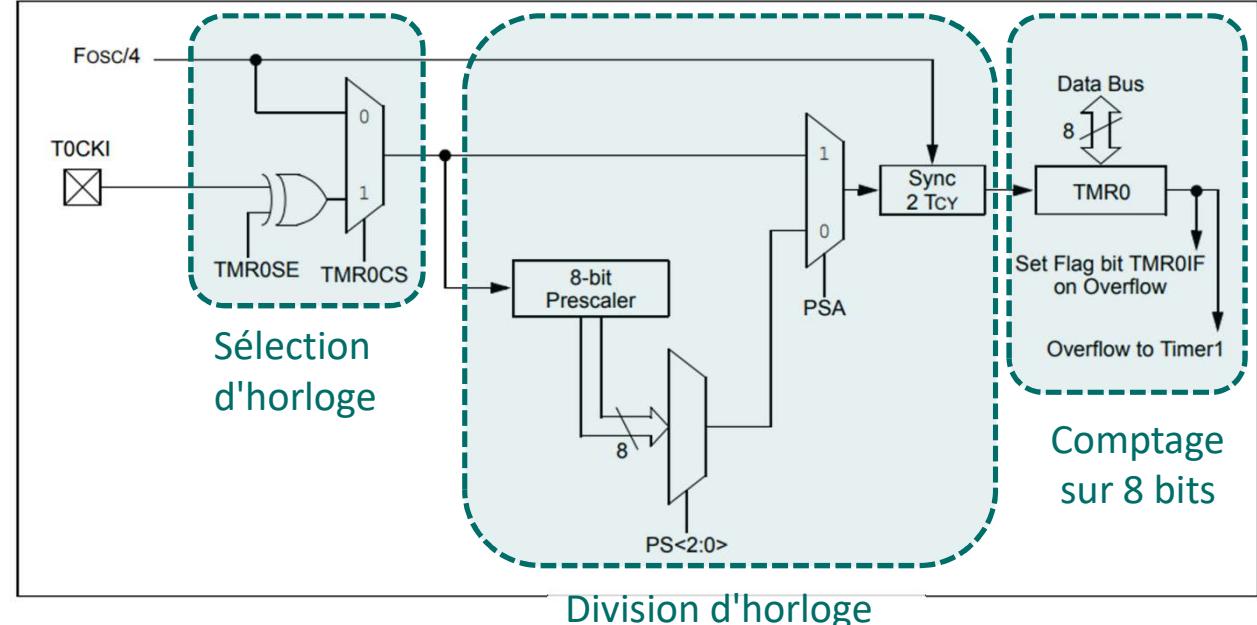
Les timers

- Les timers sont des périphériques permettant de :
 - Réaliser des temporisations précises
 - Compter des événements internes ou externes sur fronts
 - Mesurer des intervalles de temps
- Le PIC16F1719 est doté de :
 - **4 timers 8b** (Timers 0/2/4/6) (*[1] pp. 2, 271, 285*)
 - **1 timer 16b** (Timer 1) (*[1] pp. 2, 274*)

Fonctionnement du Timer 0

OPTION REG	TMROSE	<u>Source Edge</u> (activation sur front montant ou descendant)
	TMROCS	<u>Clock Select</u> (sélection de la source d'horloge)
	PS<2:0>	<u>Prescaler : diviseur d'horloge :</u> 000 ↔ 1:2 001 ↔ 1:4 ... 111 ↔ 1:256
	PSA	<u>Prescaler assignment (prescaler disable)</u>
	TMRO	Registre de valeur du compteur
INTCON	TMROIF	Flag d'overflow (<u>Interruption Flag</u>)

FIGURE 26-1: BLOCK DIAGRAM OF THE TIMER0



Le Timer 0 incrémentera périodiquement la valeur de TMRO et passe le flag TMROIF à 1 dès que le compteur dépasse sa capacité (overflow). Le flag TMROIF ne repassera jamais spontanément à l'état bas. Il ne peut être remis à 0 (« acquittement ») que par le programme.

Calcul de la période du Timer 0

La période $T_{overflow}$ du Timer 0 se calcule comme le temps nécessaire pour incrémenter 256 fois le registre TMR0 :

$$T_{overflow} = 256 \times \left(\frac{\text{prescaler}}{F_{clk}} \right)$$

Avec $\text{prescaler} \in \{1; 2; 4; \dots; 256\}$ et F_{clk} la fréquence de l'horloge sélectionnée (Fosc/4 ou horloge externe)

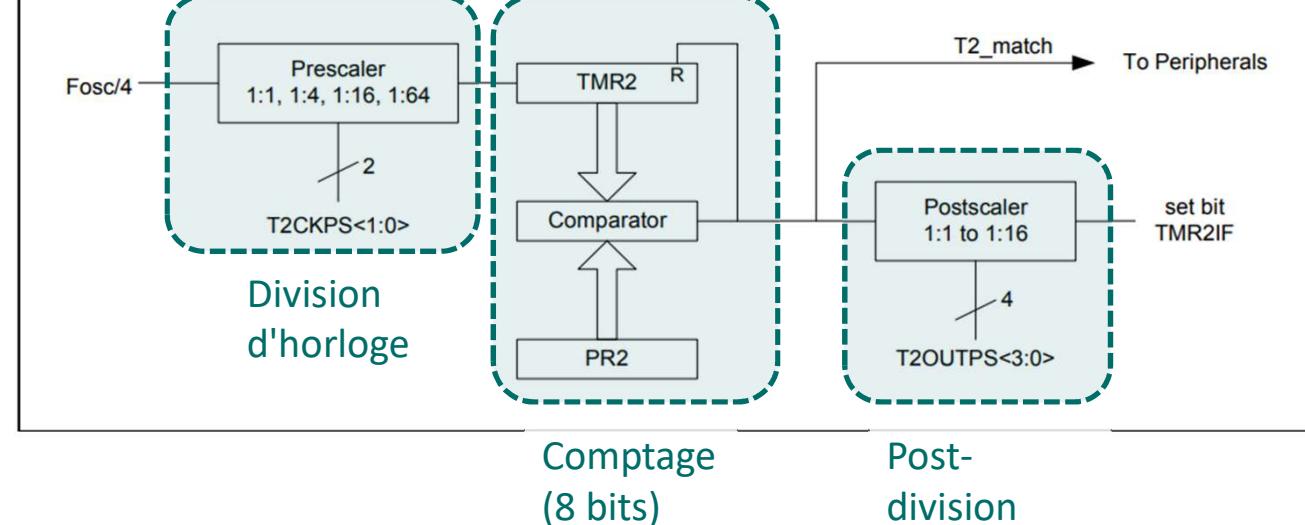
Note : cette formule limite les délais possibles du Timer 0 à des multiples de la période d'horloge par puissances de 2 seulement.
Pour plus de possibilités, il est classique de réinitialiser à une valeur $TMR0_{ini} > 0$ la valeur de TMR0 dès qu'un overflow est détecté ;

$$\text{dans ce cas, on aura plutôt : } T_{overflow} = (256 - TMR0_{ini}) \times \left(\frac{\text{prescaler}}{F_{clk}} \right)$$

Fonctionnement des Timers 2/4/6

T2CON	T2CKPS<1:0>	<u>Prescaler</u> (diviseur d'horloge)
	T2OUTPS	<u>Postscaler</u>
	TMR2ON	Bit d'enable
PR2		Registre de période du compteur
TMR2		Registre de valeur du compteur

FIGURE 28-1: TIMER2 BLOCK DIAGRAM



Le Timer 2 incrémentera périodiquement la valeur de TMR2 et passe le flag T2_match à 1 dès que TMR2==PR2 (et réinitialise TMR2). Un postscaler permet de lever le flag TMR2IF à la même fréquence que T2_match ou à une subdivision de sa fréquence.

Le flag TMR2IF ne repassera jamais spontanément à 0. Il ne peut être remis à l'état bas que par le programme.

Le fonctionnement est identique pour les Timers 4 et 6.

Calcul de la période du Timer 2

La période $T_{overflow}$ du Timer 2 se calcule comme le temps nécessaire pour incrémenter le registre TMR2 jusqu'à atteindre PR2 :

$$T_{overflow} = (PR2 + 1) \times \left(\frac{prescaler \times postscaler}{F_{osc}/4} \right)$$

Avec $prescaler \in \{1; 4; 16; 64\}$ et $postscaler \in [1, 16]$

Configuration du Timer 2

28.5 Register Definitions: Timer2 Control

REGISTER 28-1: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	T2OUTPS<3:0>			TMR2ON	T2CKPS<1:0>		
bit 7	bit 0						

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

u = Bit is unchanged

x = Bit is unknown

-n/n = Value at POR and BOR/Value at all other Resets

'1' = Bit is set

'0' = Bit is cleared

bit 7

bit 6-3

Unimplemented: Read as '0'

T2OUTPS<3:0>: Timer2 Output Postscaler Select bits

1111 = 1:16 Postscaler

1110 = 1:15 Postscaler

1101 = 1:14 Postscaler

1100 = 1:13 Postscaler

1011 = 1:12 Postscaler

1010 = 1:11 Postscaler

1001 = 1:10 Postscaler

1000 = 1:9 Postscaler

0111 = 1:8 Postscaler

0110 = 1:7 Postscaler

0101 = 1:6 Postscaler

0100 = 1:5 Postscaler

0011 = 1:4 Postscaler

0010 = 1:3 Postscaler

0001 = 1:2 Postscaler

0000 = 1:1 Postscaler

bit 2

TMR2ON: Timer2 On bit

1 = Timer2 is on

0 = Timer2 is off

bit 1-0

T2CKPS<1:0>: Timer2 Clock Prescale Select bits

11 = Prescaler is 64

10 = Prescaler is 16

01 = Prescaler is 4

00 = Prescaler is 1

[1] p. 287

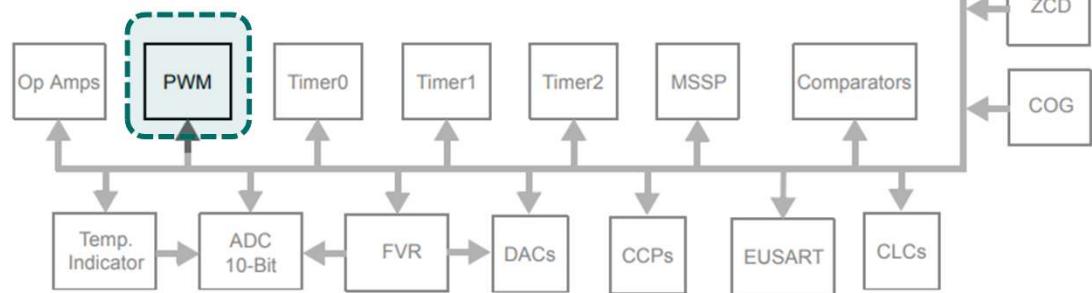
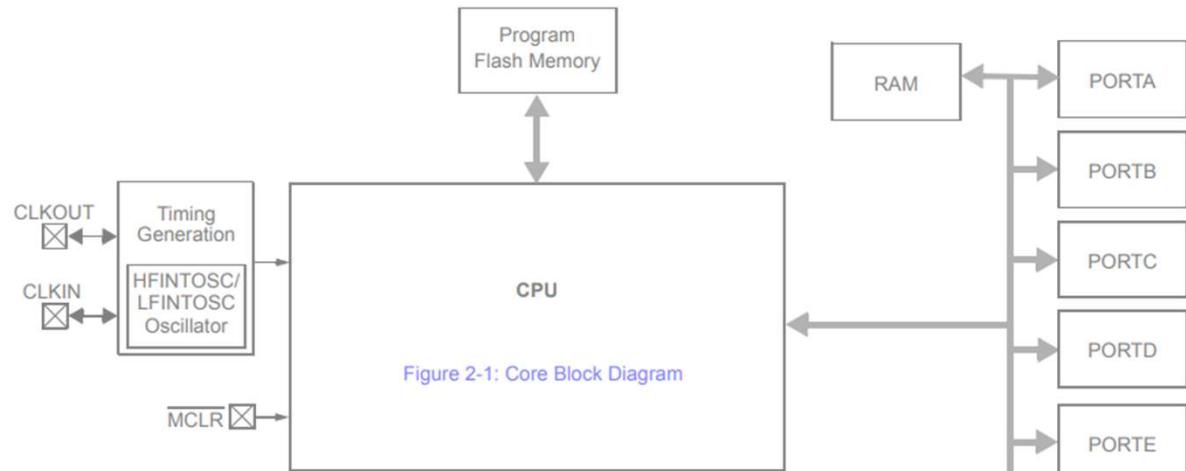
Utilisation des timers

- On pourra exploiter les timers :
 - Par **scrutation** : des tests sont effectués par le programme pour détecter l'overflow (par ex. par une conditionnelle `if` sur le bit d'overflow)
 - Via des **interruptions**, configurées pour se déclencher sur le bit d'overflow du timer (le programme en cours s'interrompt momentanément pour exécuter immédiatement un autre programme)
 - Via les modules de **capture/compare/pwm** : des périphériques internes au microcontrôleur sont déclenchés par l'overflow du timer via des connexions physiques

La PWM

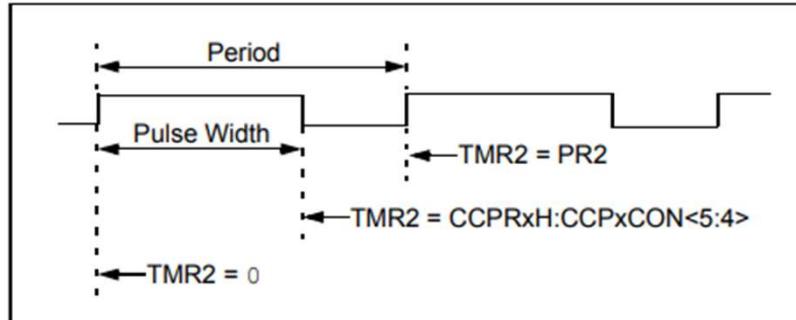
Pulse Width Modulation

[1] Sections 17 (p. 178),
29.3 (p. 293)



Fonctionnement de la PWM

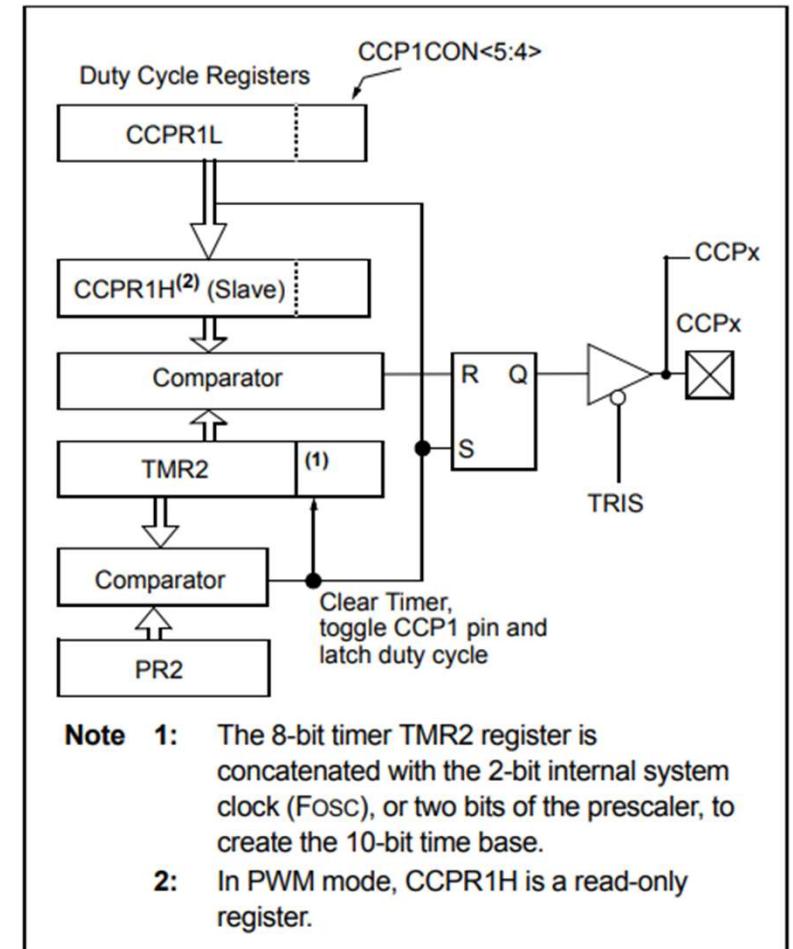
FIGURE 29-3: CCP PWM OUTPUT SIGNAL



La génération de la période de PWM se fait comme le Timer 2 (sans postscaler) ; le rapport cyclique est déterminé par la comparaison de TMR2 à $CCPR1H:CCP1CON<5:4>$:

- Quand $PR2 == TMR2$, la sortie PWM est mise à 1.
- Quand $TMR2 == CCPR1H:CCP1CON<5:4>$, la sortie PWM est mise à 0

FIGURE 29-4: SIMPLIFIED PWM BLOCK DIAGRAM



Calcul de la période et de la largeur de PWM

EQUATION 29-1: PWM PERIOD

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

Note 1: $T_{OSC} = 1/F_{OSC}$

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCPx pin is set. (Exception: If the PWM duty cycle = 0%, the pin will not be set.)
- The PWM duty cycle is latched from CCPRxL into CCPRxH

Note: The Timer postscaler (see [Section 28.1 “Timer2 Operation”](#)) is not used in the determination of the PWM frequency.

Dans la documentation ([1] p. 294), on retrouve la formule donnée précédemment pour calculer la période du Timer 2 (la PWM n'utilisant pas le postscaler)

EQUATION 29-2: PULSE WIDTH

$$\text{Pulse Width} = (\text{CCPRxL:CCPxCON<5:4>} \bullet \\ \bullet T_{OSC} \bullet (\text{TMR2 Prescale Value}))$$

EQUATION 29-3: DUTY CYCLE RATIO

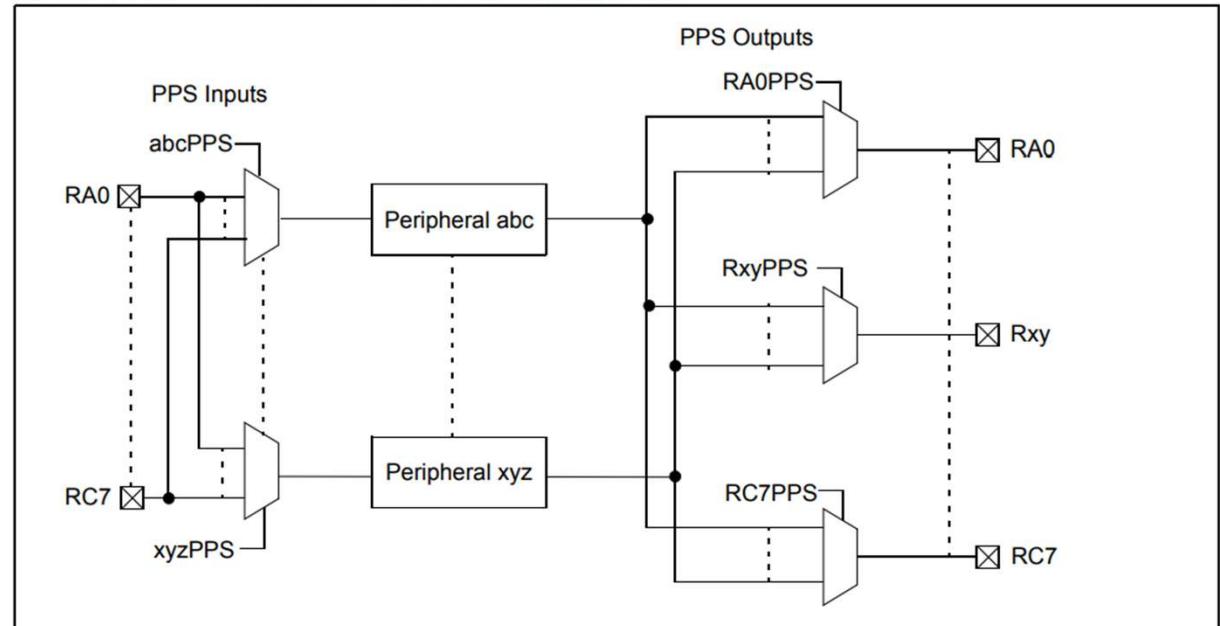
$$\text{Duty Cycle Ratio} = \frac{(\text{CCPRxL:CCPxCON<5:4>})}{4(PR2 + 1)}$$

Attribution de la sortie PWM – le module PPS

PPS : Peripheral Pin Select

- Les entrées/sorties d'un périphérique peuvent être **assignées à des GPIO**
 - L'attribution d'une broche GPIO à l'entrée d'un périphérique se fait via le registre xxxPPS (ex. INTPPS pour la pin d'interruption) [1] Table 12-1 p. 153
 - L'attribution d'une sortie de périphérique à une broche GPIO se fait via le registre RxyPPS (ex. RB3PPS pour assigner à PORTB3)
- [1] Table 12-2 p. 154

FIGURE 12-1: SIMPLIFIED PPS BLOCK DIAGRAM



[1] Section 12.0 (p. 151)

Cette logique d'attribution est vraie non seulement pour les sorties PWM, mais aussi pour la plupart des périphériques du microcontrôleur

Le registre xxxPPS

REGISTER 12-1: xxxPPS: PERIPHERAL xxx INPUT SELECTION

U-0	U-0	U-0	R/W-q/u	R/W-q/u	R/W-q/u	R/W-q/u	R/W-q/u				
—	—	—		xxxPPS<4:0>							
bit 7											bit 0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'
 u = Bit is unchanged x = Bit is unknown -n/n = Value at POR and BOR/Value at all other Resets
 '1' = Bit is set '0' = Bit is cleared q = value depends on peripheral

bit 7-5 **Unimplemented:** Read as '0'

bit 4-3 **xxxPPS<4:3>:** Peripheral xxx Input PORTx Selection bits
See [Table 12-1](#) for the list of available ports for each peripheral.

11 = Peripheral input is from PORTD (PIC16(L)F1717/9 only)

10 = Peripheral input is from PORTC

01 = Peripheral input is from PORTB

00 = Peripheral input is from PORTA

bit 2-0 **xxxPPS<2:0>:** Peripheral xxx Input PORTx Bit Selection bits

111 = Peripheral input is from PORTx Bit 7 (Rx7)

110 = Peripheral input is from PORTx Bit 6 (Rx6)

101 = Peripheral input is from PORTx Bit 5 (Rx5)

100 = Peripheral input is from PORTx Bit 4 (Rx4)

011 = Peripheral input is from PORTx Bit 3 (Rx3)

010 = Peripheral input is from PORTx Bit 2 (Rx2)

001 = Peripheral input is from PORTx Bit 1 (Rx1)

000 = Peripheral input is from PORTx Bit 0 (Rx0)

TABLE 12-1: AVAILABLE PORTS FOR INPUT BY PERIF

Peripheral	Register	PIC16(L)F1717/8/9	
		PORTA	PORTB
PIN interrupt	INTPPS	•	•
Timer0 clock	T0CKIPPS	•	•
Timer1 clock	T1CKIPPS	•	
Timer1 gate	T1GPPS		•
CCP1	CCP1PPS		•
CCP2	CCP2PPS		•
COG	COGINPPS		•
MSSP	SSPCLKPPS		•
MSSP	SSPDATPPS		•
MSSP	SSPSSPPS	•	
EUSART	RXPPS		•
EUSART	CKPPS		•
All CLCs	CLCIN0PPS	•	
All CLCs	CLCIN1PPS	•	
All CLCs	CLCIN2PPS		•
All CLCs	CLCIN3PPS		•

Example: CCP1PPS = 0x0B selects RB3 as the input to CCP1.

[1] Section 12.8 (p. 153)

Le registre RxyPPS

REGISTER 12-2: RxyPPS: PIN Rxy OUTPUT SOURCE SELECTION REGISTER

U-0	U-0	U-0	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u
—	—	—					
RxyPPS<4:0>							
bit 7							bit 0

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

u = Bit is unchanged

x = Bit is unknown

-n/n = Value at POR and BOR/Value at all other Resets

'1' = Bit is set

'0' = Bit is cleared

bit 7-5 **Unimplemented:** Read as '0'bit 4-0 **RxyPPS<4:0>:** Pin Rxy Output Source Selection bits.

Selection code determines the output signal on the port pin.

See [Table 12-2](#) for supported ports and selection codes.*[1] Section 12.8 (p. 154)*

TABLE 12-2: AVAILABLE PORTS FOR OUTPUT BY PERI

RxyPPS<4:0>	Output Signal	PIC16(L)F1717/8/9	
		PORTA	PORTB
11xxx	Reserved		
10111	C2OUT	•	
10110	C1OUT	•	
10101	DT ⁽²⁾		•
10100	TX/CK ⁽²⁾		•
10011	Reserved		
10010	Reserved		
10001	SDO/SDA ⁽²⁾		•
10000	SCK/SCL ⁽²⁾		•
01111	PWM4OUT		•
01110	PWM3OUT		•
01101	CCP2		•
01100	CCP1		•
01011	COG1D ⁽²⁾		•
01010	COG1C ⁽²⁾		•
01001	COG1B ⁽²⁾		•
01000	COG1A ⁽²⁾		•
00111	CLC4OUT		•
00110	CLC3OUT		•
00101	CLC2OUT	•	
00100	CLC1OUT	•	
00011	NCO1OUT	•	
00010	Reserved		
00001	Reserved		
00000	LATxy	•	•

Example: RB3PPS = 0x16 selects RB3 as the comparator 1 output.

Les interruptions

[1] Section 7 p. 87

Qu'est-ce qu'une interruption ?

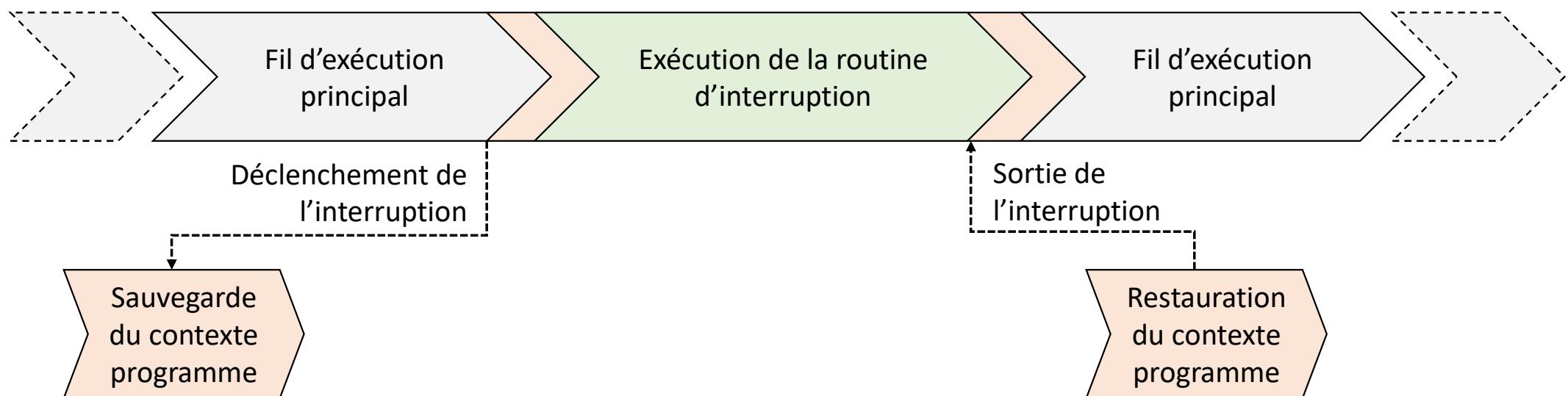
Les **interruptions** sont des événements détectés par le microcontrôleur, qui mettent en pause le déroulé normal du programme pour exécuter un sous-programme d'interruption, puis reprendre le déroulé normal du programme.

Le microcontrôleur peut donc obéir :

- Aux instructions de son programme (déroulé normal)
- A des **entrées d'interruption**, déclenchant une fonction de façon asynchrone

Chaque entrée d'interruption peut être associée à l'adresse d'une fonction à exécuter (**vecteur d'interruption**)

Le déroulement d'une interruption



- Stockage de l'adresse de retour dans le « return stack »
- Copie des registres processeur dans les « shadow registers »

L'état du processeur est **exactement le même** avant et après l'interruption

La routine d'interruption est généralement très courte (<5% du temps d'exécution), et perturbe peu le cours d'exécution du programme principal (>95% du temps d'exécution)

Configuration d'une interruption

- 1) Programmer le microcontrôleur pour **réagir aux interruptions** (via le Global Interrupt Enable (GIE) et souvent le Peripheral Interrupt Enable (PEIE) dans le registre INTCON) [1]
- 2) Activer les **sources d'interruption** des périphériques voulus (bits situés dans les registres INTCON, PIE1, PIE2, PIE3) [1]
- 3) Déclarer l'interruption (ISR : Interrupt Service Routine) dans le **programme** (voir [3] p. 200)
- 4) Lorsqu'une interruption est déclenchée, il faut que le programme **acquitte le flag d'interruption** qui l'a déclenchée (flags situés dans les registres INTCON, PIR1, PIR2, PIR3) [1]

TABLE 7-1: SUMMARY OF REGISTERS ASSOCIATED WITH INTERRUPTS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
INTCON	GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF	91
OPTION_REG	WPUEN	INTEDG	TMR0CS	TMR0SE	PSA	PS<2:0>			273
PIE1	TMR1GIE	ADIE	RCIE	TXIE	SSP1IE	CCP1IE	TMR2IE	TMR1IE	92
PIE2	OSFIE	C2IE	C1IE	—	BCL1IE	TMR6IE	TMR4IE	CCP2IE	93
PIE3	—	NCOIE	COGIE	ZCDIE	CLC4IE	CLC3IE	CLC2IE	CLC1IE	94
PIR1	TMR1GIF	ADIF	RCIF	TXIF	SSP1IF	CCP1IF	TMR2IF	TMR1IF	95
PIR2	OSFIF	C2IF	C1IF	—	BCL1IF	TMR6IF	TMR4IF	CCP2IF	96
PIR3	—	NCOIF	COGIF	ZCDIF	CLC4IF	CLC3IF	CLC2IF	CLC1IF	97

Legend: — = unimplemented location, read as '0'. Shaded cells are not used by interrupts.

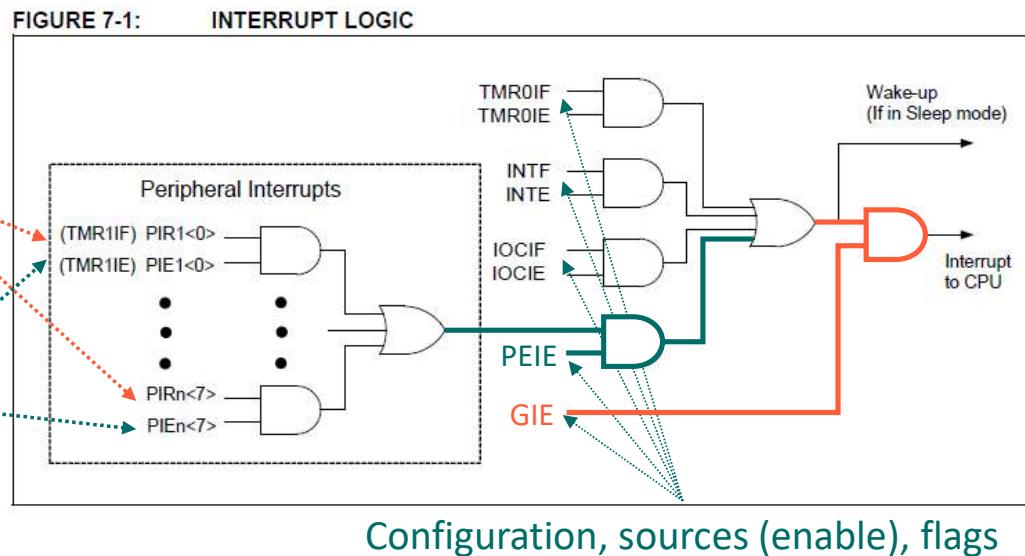
Configuration, sources (enable), flags

Sources d'interruption (bits d'enable)

Flags d'interruption

Configuration d'une interruption

- 1) Programmer le microcontrôleur pour **réagir aux interruptions** (via le Global Interrupt Enable (GIE) et souvent le Peripheral Interrupt Enable (PEIE) dans le registre INTCON) [1]
- 2) Activer les **sources d'interruption** des périphériques voulus (bits situés dans les registres INTCON, PIE1, PIE2, PIE3) [1]
- 3) Déclarer l'interruption (ISR : Interrupt Service Routine) dans le **programme** (voir [3] p. 200)
- 4) Lorsqu'une interruption est déclenchée, il faut que le programme **acquitte le flag d'interruption** qui l'a déclenchée (flags situés dans les registres INTCON, PIR1, PIR2, PIR3) [1]



Configuration d'un interruption - INTCON

bit 7	GIE: Global Interrupt Enable bit 1 = Enables all active interrupts 0 = Disables all interrupts
bit 6	PEIE: Peripheral Interrupt Enable bit 1 = Enables all active peripheral interrupts 0 = Disables all peripheral interrupts
bit 5	TMR0IE: Timer0 Overflow Interrupt Enable bit 1 = Enables the Timer0 interrupt 0 = Disables the Timer0 interrupt
bit 4	INTE: INT External Interrupt Enable bit 1 = Enables the INT external interrupt 0 = Disables the INT external interrupt
bit 3	IOCIE: Interrupt-on-Change Enable bit 1 = Enables the interrupt-on-change 0 = Disables the interrupt-on-change
bit 2	TMR0IF: Timer0 Overflow Interrupt Flag bit 1 = TMR0 register has overflowed 0 = TMR0 register did not overflow
bit 1	INTF: INT External Interrupt Flag bit 1 = The INT external interrupt occurred 0 = The INT external interrupt did not occur
bit 0	IOCIF: Interrupt-on-Change Interrupt Flag bit ⁽¹⁾ 1 = When at least one of the interrupt-on-change pins changed state 0 = None of the interrupt-on-change pins have changed state

REGISTER 7-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0/0	R-0/0						
GIE	PEIE	TMR0IE	INTE	IOCIE	TMR0IF	INTF	IOCIF ⁽¹⁾
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

[1] Section 7.6 (p. 91)

Bits d'enable de sensibilité des interruptions – PIE1

bit 7

TMR1GIE: Timer1 Gate Interrupt Enable bit1 = Enables the Timer1 gate acquisition interrupt
0 = Disables the Timer1 gate acquisition interrupt

bit 6

ADIE: Analog-to-Digital Converter (ADC) Interrupt Enable bit1 = Enables the ADC interrupt
0 = Disables the ADC interrupt

bit 5

RCIE: USART Receive Interrupt Enable bit1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4

TXIE: USART Transmit Interrupt Enable bit1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3

SSP1IE: Synchronous Serial Port (MSSP) Interrupt Enable bit1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2

CCP1IE: CCP1 Interrupt Enable bit1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1

TMR2IE: TMR2 to PR2 Match Interrupt Enable bit1 = Enables the Timer2 to PR2 match interrupt
0 = Disables the Timer2 to PR2 match interrupt

bit 0

TMR1IE: Timer1 Overflow Interrupt Enable bit1 = Enables the Timer1 overflow interrupt
0 = Disables the Timer1 overflow interrupt**REGISTER 7-2: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1**

| R/W-0/0 |
|---------|---------|---------|---------|---------|---------|---------|---------|
| TMR1GIE | ADIE | RCIE | TXIE | SSP1IE | CCP1IE | TMR2IE | TMR1IE |
| bit 7 | | | | | | | bit 0 |

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

u = Bit is unchanged

x = Bit is unknown

-n/n = Value at POR and BOR/Value at all other Resets

'1' = Bit is set

'0' = Bit is cleared

[1] Section 7.6 (p. 92),
 voir aussi PIE2 (p. 93), PIE3 (p. 94)

Flags d'interruption – PIR1

bit 7	TMR1GIF: Timer1 Gate Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 6	ADIF: Analog-to-Digital Converter (ADC) Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 5	RCIF: USART Receive Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 4	TXIF: USART Transmit Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 3	SSP1IF: Synchronous Serial Port (MSSP) Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 2	CCP1IF: CCP1 Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 1	TMR2IF: Timer2 to PR2 Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending
bit 0	TMR1IF: Timer1 Overflow Interrupt Flag bit 1 = Interrupt is pending 0 = Interrupt is not pending

REGISTER 7-5: PIR1: PERIPHERAL INTERRUPT REQUEST REGISTER 1

R/W-0/0	R/W-0/0	R-0/0	R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIF	ADIF	RCIF	TXIF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

[1] Section 7.6 (p. 95),
voir aussi PIR2 (p. 96), PIR3 (p. 97)