

**TP transdisciplinaire en EEA**  
**Principes de programmation du dsPIC30F4012**

## 1 Structure d'un microcontrôleur

Un microcontrôleur est un composant associant dans un même boîtier un coeur de microprocesseur, de la mémoire et des périphériques.

### 1.1 Microprocesseur

Le microprocesseur a pour rôle d'exécuter un programme composé d'une succession d'instructions rangées en mémoire. Pour cela, il dispose d'une *unité de décodage d'instructions*, d'une *unité arithmétique et logique* et de *registres*. Ces derniers sont destinés :

- à manipuler les données (registres de travail ou W) ;
- à indiquer l'état du processeur (registres SR) ;
- ou à piloter le moteur DSP intégré au microprocesseur.

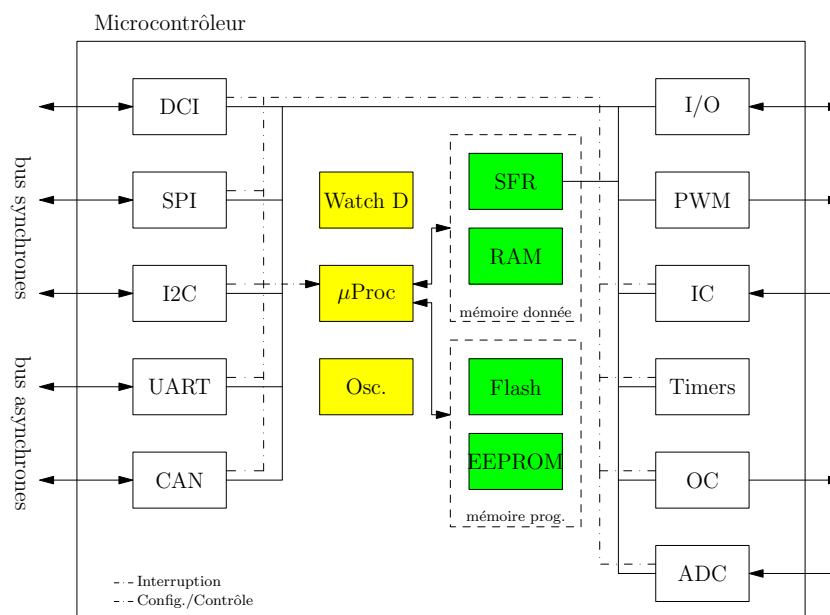


FIG. 1 : Structure interne d'un micro-contrôleur

Dans un langage évolué tel que le C, les registres ne sont presque jamais manipulés par le programmeur.

### 1.2 Mémoire

Le programme et les données sont stockées différemment selon l'architecture du microcontrôleur. Dans le cas de la structure de type Harvard utilisée par Microchip, trois types de mémoires sont disponibles et utilisés comme suit :

- la RAM représente quelques ko de mémoire volatile : elle accueille les variables utilisées par le programme ;
- la FLASH représente quelques centaines de ko voire quelques Mo : elle accueille le programme ainsi que les données constantes. Cette mémoire est conservée en cas de rupture de l'alimentation ;
- l'EEPROM est disponible pour stocker les données de configuration critiques d'un programme en cas de perte d'alimentation et les restituer au redémarrage. Généralement chaque microcontrôleur en embarque 1 ou 2 ko.

## Espace programme

Les mémoires FLASH et l'EEPROM sont accédées par un bus de 24bits permettant d'adresser potentiellement 16Mo appelés *espace programme*. l'espace programme est pointé par un registre appelé *compteur programme* ou PC dont le rôle est de dépiler les instructions du programme. L'ensemble de cet espace d'adressage n'est pas uniquement occupé par du code programme ou par les données de l'EEPROM. Le début de la zone accueille la table des adresses des routines d'interruption et les 8Mo d'adresse haute sont réservés pour stocker quelques éléments tels l'identifiant du microcontrôleur ou ses bits de configuration. Finalement, la majorité de l'espace programme ne contient pas réellement de mémoire physique. Une grande partie des adresses sont inutilisées et pointent dans le vide.

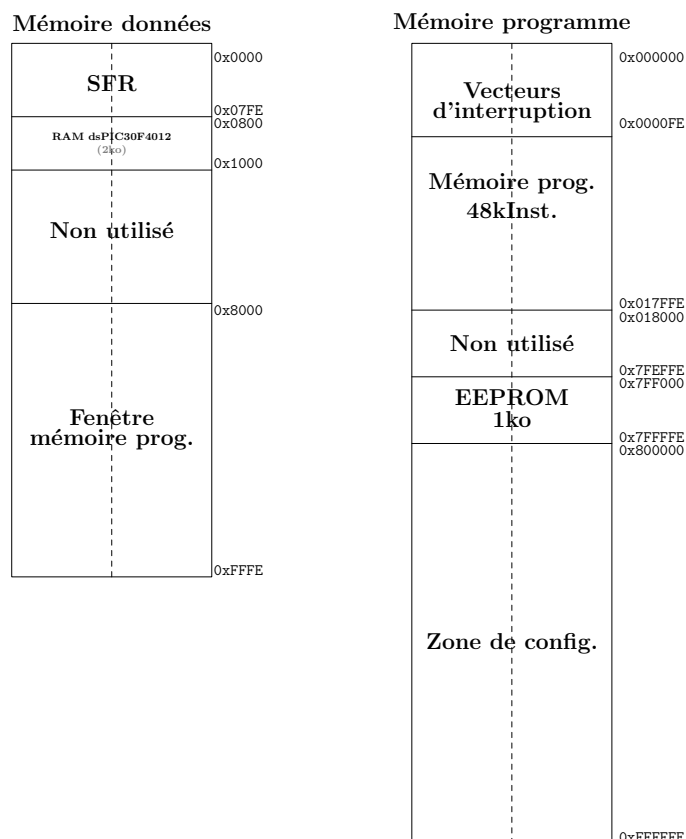


FIG. 2 : Espaces d'adressage données et programme

## Espace données

Le microprocesseur accède à la mémoire RAM à l'aide d'un bus de données 16bits qui ne permet d'adresser que 64ko de mémoire. Une partie de cet espace d'adressage (environ 2ko) est réservée pour accéder aux registres de périphériques appelés SFR (*special function registers*). Ces registres permettent de configurer et de piloter chacun des périphériques présents dans le microcontrôleur. Le découpage des espaces mémoire est présenté sur la figure 2 dans le cas spécifique du pic30F4012.

## 1.3 Périphériques

Les périphériques intégrés au microcontrôleur sont nombreux et aux fonctionnalités multiples :

- interfaces dédiées aux bus de communications synchrones (SPI, DCI, I2C) ;
- interfaces dédiées aux bus de communication asynchrones (UART, CAN) ;
- entrées-sorties TTL ou à collecteur ouvert ;
- timers ;
- input-capture ;

- output-compare ;
- générateur de modulation de largeurs d'impulsions (PWM).

Grâce au mécanisme d'interruption décrit en section 2.1, chacun de ces périphériques est apte à indiquer au microprocesseur l'occurrence d'évènements nécessitant un traitement immédiat.

## 2 Structure d'un programme sur microcontrôleur

Les programmes implantés sur les microcontrôleurs possèdent tous la même structure : les actions périodiques ou déclenchées par les périphériques sont insérées au sein de routines d'interruption. Parmi ces actions, on peut citer :

- l'acquisition de valeurs analogiques à intervalle de temps régulier ;
- la lecture d'une valeur reçue sur un bus de communication ;
- la détection d'un front sur une entrée à fréquence élevée ou lorsque l'instant du front doit être consigné avec précision.

Les actions dont la prise en charge n'est pas urgente ou dont le traitement est long sont insérées dans une boucle sans fin. Il s'agit généralement des actions d'interaction avec un opérateur humain parmi lesquelles on trouve :

- la détection d'un appui sur un bouton (méthode de scrutation aussi appelée pooling) ;
- la mise à jour d'un afficheur LCD.

### 2.1 Routines d'interruption

Une routine d'interruption est une fonction particulière qui ne prend pas de paramètre et qui ne retourne aucun résultat. **Elle n'est jamais explicitement appelée par le programme** mais son exécution est déclenchée sur évènement d'un périphérique (timer, réception d'un octet sur un bus de communication, fin de transmission d'un octet, fin de conversion analogique, détection d'un front, overflow de certains registres ...)

#### 2.1.1 Fonctionnement

Lorsqu'un périphérique souhaite signaler un évènement, un drapeau est positionné dans l'un des registres **IFSx**. L'interruption n'est appelée que si le programme l'a autorisée. A cet effet, les registres nommés **IECx** contiennent un ensemble de bits dont chacun est lié à une interruption autorisée (valeur à 1) ou non prise en compte (valeur à 0). Lorsque l'interruption est autorisée, le passage à 1 du drapeau correspondant du registre **IFSx** déclenche immédiatement les actions suivantes :

1. les contenus du pointeur d'exécution et du registre d'état sont mémorisés sur la pile ;
2. le registre d'exécution est chargé avec l'adresse de la routine d'interruption qui s'exécute immédiatement ;
3. la routine efface le drapeau d'interruption pour éviter une relance immédiate de l'interruption une fois la routine quittée ;
4. après exécution de la dernière instruction de la routine, le pointeur d'exécution et l'état du processeur sont restitués ;
5. l'exécution du programme principal reprend là où il avait été quitté.

Pour le compilateur C30, chaque source d'interruption est liée à une routine grâce à son nom : ainsi l'interruption du timer 1 doit être nommée `_T1Interrupt()`, l'interruption indiquant la réception d'un octet sur l'UART se nomme `_U1RXInterrupt()`.

## 2.2 Structure type d'un programme

Le code est organisé comme suit :

1. les routines d'interruptions sont définies ;
2. au sein de la fonction `main`, les périphériques sont initialisés, puis activés ;
3. les sources d'interruptions pour lesquelles les routines ont été définies sont autorisées (*i.e. demandées*) ;
4. une boucle sans fin est ajoutée, vide si l'ensemble des actions est déclenchée sur interruption ou contenant les actions dont le traitement ne revet pas de caractère d'urgence.

Un tel code type est présenté ci-dessous et l'organigramme correspondant sur la figure 3 .

La communication de données entre les routines d'interruption et le corps du programme (*main*) est assurée par des variables globales, c'est à dire déclarées en dehors de tout corps de fonction.

```

1  void __attribute__((__interrupt__,__no_auto_psv)) _[Periph1Name]Interrupt( void ){
2
3      ...                               // Actions à exécuter dans la routine
4
5      IFSxbits.[Periph1Name]IF=0;       // Acquiescement de l'interruption
6  }
7
8      ...
9  void main(void){
10
11      init[Periph1Name]([Init args]);    // Appel des fonctions
12      init[Periph2Name]([Init args]);    // d'initialisation des périphériques
13      ...
14      enable[Periph1Name]Interrupt();    // autorisation des interruptions
15                                          // déclenchées par Periph1
16
17      while(1){
18          if([Event]){
19              [Low priority action];      // pooling d'évènements peu prioritaires
20          }
21      }
22  }
```

## 2.3 Les entrées-sorties numériques (I/O)

Les entrées-sorties numériques sont groupées en ports 16 bits (A0 à A15, B0 à B15, ...) et sortent sur des broches respectivement nommés RA0 à RA15, RB0 à RB15 ... Le nombre de broches dédiés aux I/O effectivement présentes sur un microcontrôleur dépend de sa taille et tous les bits d'un port ne sont pas nécessairement présents physiquement sous forme d'une broche. La figure ?? présente le brochage du dsPIC30F4012.

Chaque broches I/O admet en entrée ou fixe en sortie, selon sa direction, des tensions TTL. Par défaut, après reset du microcontrôleur, les broches configurées pour utiliser les ports d'entrées-sorties sont programmées en entrées pour éviter d'imposer des tensions sur des broches connectées à des circuits externes fixant leur propre niveau de tension. Sur le dsPIC30F4012, il n'est pas possible de fournir ou d'absorber un courant supérieur à **25mA** par broche avec une limite supérieure à **200mA** pour l'ensemble du microcontrôleur.

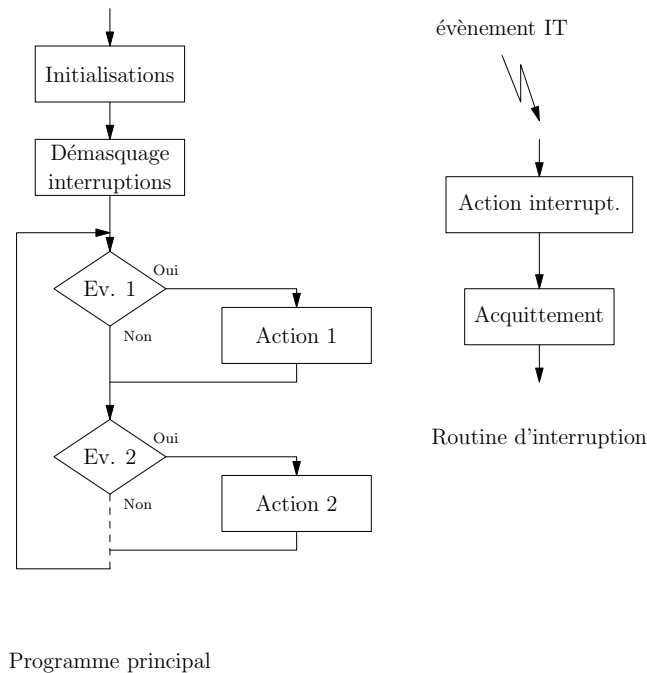


FIG. 3 : Organigramme d'un programme type sur micro-contrôleur

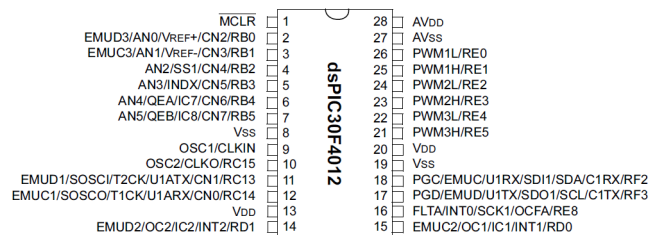


FIG. 4 : Brochage du dsPIC30F4012

### ⚠ Attention :

Les broches d'un microcontrôleur sont généralement associées à différentes fonctionnalités multiplexées : le nom de la broche indique alors ces fonctionnalités par priorité décroissante. Ainsi, une broche AN7/RE8 est utilisée en entrée analogique dès lors que le module de conversion analogique est activé et que la programmation de la voie AN7 le prévoit. Selon cette nomenclature, le module I/O est généralement le moins prioritaire : il est donc important de vérifier qu'aucun périphérique ne masque ce module sur les broches devant être utilisées en I/O.

### 2.3.1 Programmation

Chaque bit d'un port est paramétrable en entrée ou en sortie individuellement. Pour cela, on utilise le registre `TRISn` où `n` indique le port (A, B C, ...). Un bit à 1 place de bit en entrée alors qu'un bit à 0 le programme en sortie. Une fois la direction du port choisie, on lit l'état électrique ou on écrit la valeur TTL à imposer sur la broche à l'aide du registre `PORTn`. Enfin, il est possible que l'état électrique d'une broche soit différent de l'état logique fixé par le programme (court circuit, pilotage d'un circuit capacitif en fréquence élevée). L'état logique peut alors être relu à l'aide du registre `LATn` alors que la lecture de `PORTn` fournira l'état électrique de la broche. La circuiterie interne des entrées-sorties est illustrée sur la figure 5 .

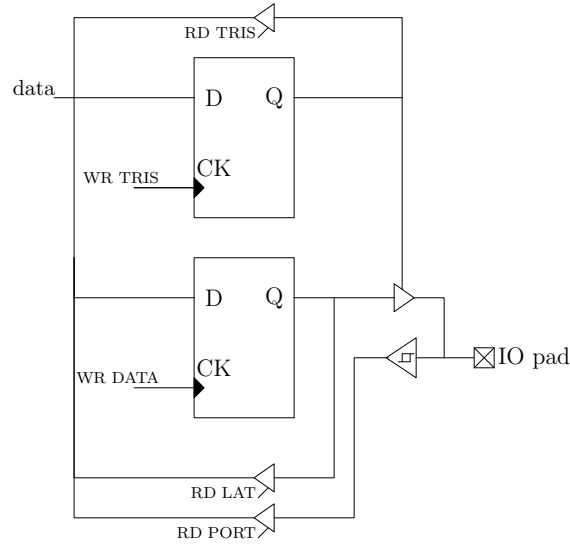


FIG. 5 : Circuiterie des entrées-sorties

## 2.4 Programmation des timers

Les timers sont des modules utilisés pour produire des actions à intervalle de temps constant ou pour mesurer des durées entre deux événements successifs. Dans le premier cas, on parle du mode **output-compare** et dans le second de mode **input-capture**. Sur les microcontrôleurs Microchip, chacun de ces modes est mis en oeuvre en combinant un timer et un module spécifique au mode. Le dsPIC30F4012 dispose de 3 timers indépendants.

### 2.4.1 Utilisation du timer seul

Chaque timer est constitué d'un compteur de 16 bits permettant de balayer les valeurs de 0 à 65535. Le quantum temporel d'incrémement est donné par le temps de cycle  $T_{cy} = \frac{1}{F_{cy}}$  où la fréquence de cycle  $F_{cy}$  est donnée par :

$$F_{cy} = \frac{7.372 \times 10^6 \times M_{PLL}}{4} Hz$$

$M_{PLL}$  est un facteur multiplicatif programmable qui vaut au maximum 16. Ainsi on obtient la fréquence maximale  $F_{cy} = 29.488 MHz$  soit  $T_{cy} = 34 ns$ .

A ce rythme, chaque compteur 16 bits atteint 65535 en 0,2ms, ce qui est un temps souvent trop court en regard des signaux à générer ou à mesurer. Pour allonger la durée de comptage, on peut exploiter deux pistes :

- cascader les timers 2 et 3 afin de produire un timer 32 bits et atteindre une durée de comptage pouvant atteindre de 145s ;
- ralentir l'horloge de comptage en divisant  $F_{cy}$  : c'est le rôle des prescalers.

### Principe de fonctionnement

Le fonctionnement du timer est le suivant (voir figure 6) :

1. le compteur 16 bits **TMRi** s'incrémement toute les  $\alpha T_{cy}$  où  $\alpha$  représente la valeur de prescaler ;
2. lorsqu'il atteint la valeur cible définie dans le registre **PRi**,
  - il est remis à 0 ;
  - une demande d'interruption est levée ;

En mode 32 bits le fonctionnement est identique, la valeur finale de comptage est inscrite dans la concaténation des registres **PR3** (mot de poids fort) et **PR2** (mot de poids faible). le registre de comptage **TMR3** est incrémenté sur overflow du registre de comptage **TMR2**. Lorsque **TMR3:TMR2** = **PR3:PR2**, les registres de comptage sont remis à 0 et une demande d'interruption est levée émanant du timer 3.

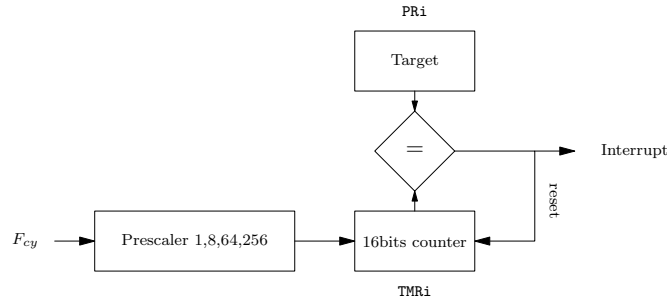


FIG. 6 : Utilisation d'un timer en générateur d'évènements périodiques

### Programmation du Timer pour le déclenchement d'actions périodiques

Une librairie constituée des fichiers `periph_tmr.c` et `periph_tmr.h`, disponible sur votre environnement numérique de travail (Moodle du cours EEA0601) permet de simplifier le déclenchement d'actions à intervalles réguliers. Afin de répondre à la majorité des besoins, cette librairie exploite les timers 2 et 3 en mode 32 bits. Les fonctions disponibles sont :

- `InitTimer3(unsigned long periode)` permet d'initialiser le timer 3 en mode 32bits afin qu'il lève une demande d'interruption toute les  $3,39 \times \text{periode} \mu\text{s}$  avec  $\text{period} \in \{0, \dots, 2^{32} - 1\}$ .
- `Timer3InterruptEnable(void)` autorise le déclenchement d'interruptions par le timer 3 et active le comptage.

☞ *Attention :*

La déclaration de la routine d'interruption `_T3Interrupt( void)` est indispensable préalablement à l'utilisation de la fonction `Timer3InterruptEnable(void)` sous peine de plantage.

#### 2.4.2 Mode output-compare

En association avec le module output-compare, un timer peut être utilisé pour activer une sortie TTL à chaque fois que le compteur atteint une valeur cible primaire inscrite dans le registre `OCxR` puis la désactiver lorsqu'il atteint une valeur cible secondaire inscrite dans le registre `OCxSR`. On peut ainsi générer des impulsions de durée variable et précise sans gestion logicielle ou même des modulations de largeur d'impulsion (MLI<sup>1</sup>). Ce mode n'étant généralement pas utilisé dans les projets d'EEA0601, il n'est pas décrit davantage ici.

#### 2.4.3 Mode input-capture

En association avec le module input-capture, un timer est utilisé pour mesurer précisément l'intervalle de temps entre deux évènements sans scrutation logicielle. Le fonctionnement de ce mode est illustré sur la figure 7.

Sur l'occurrence d'un front sur la broche `ICx`, le contenu du timer  $y$  est instantanément mémorisé dans le registre `ICxBUF`. Le type de front détecté est paramétrable. Simultanément, l'exécution de la routine d'interruption `ICxInterrupt` est déclenchée (voir section 2.1) : cette routine a principalement pour objectif de traiter la valeur du timer mémorisée. Ainsi pour calculer la largeur  $\Delta_t = t_2 - t_1$  de l'impulsion rectangulaire illustrée sur la figure 7, on utilisera l'organigramme présenté sur la figure 8 . Le PIC30F4012 ne propose que deux broches d'entrées en input-capture (`IC7` et `IC8`).

### Programmation du mode input-capture

Le mode input-capture est facilement utilisable grâce à la librairie constituée de `periph_ic.c` et `periph_ic.h` présents sur votre environnement numérique de travail.

Toutefois, il sera souvent nécessaire de modifier cette librairie ainsi que `periph_tmr` afin de les adapter aux spécificités de votre application. En particulier la gestion du timer destiné à horodater l'instant d'occurrence des fronts (timer 3 par défaut) est laissée à la charge du développeur qui doit veiller à ne

<sup>1</sup>Cette dernière fonctionnalité est plus simple à implémenter au moyen du module PWM spécifique décrit en section 15 de la documentation de référence.

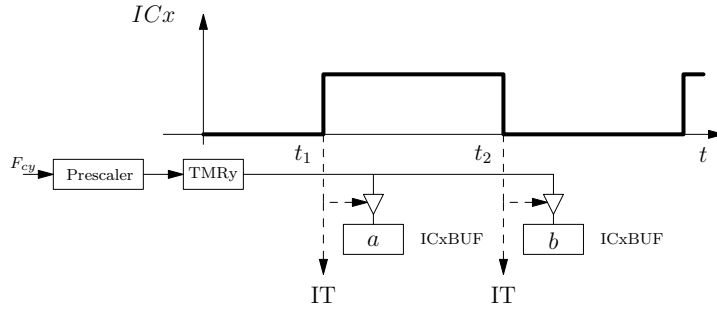
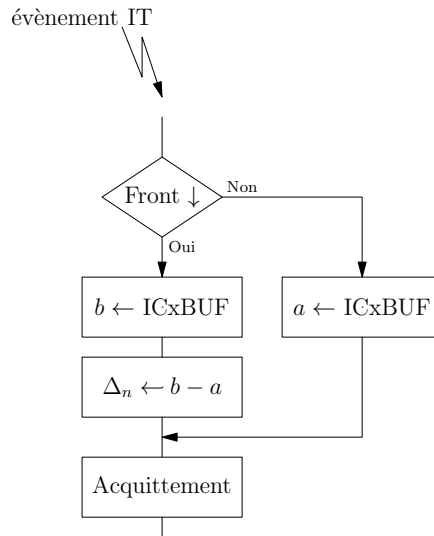


FIG. 7 : Fonctionnement du mode input-capture

FIG. 8 : Détermination d'un intervalle de temps entre deux fronts ( $\Delta_n$  est une image de  $\Delta_t$ )

pas interférer avec les timers en charge de la génération d'événements à intervalles réguliers (c.f. section 2.4.1).

La librairie ne contient qu'une fonction déclinée pour chaque entrée ICx  $x \in \{7, 8\}$  :

- `ICxInterruptEnable(unsigned int edge)` active la détection de fronts de type `edge` sur l'entrée ICx et autorise le déclenchement d'une interruption lors de l'occurrence de cet événement. `edge` peut prendre les valeurs `RISING`, `FALLING`, ou `EVERY`.

⚠ *Attention :*

- La déclaration de la routine d'interruption `_ICxInterrupt(void)` est indispensable préalablement à l'utilisation de la fonction `ICxInterruptEnable(unsigned int edge)` sous peine de plantage.
- Si vous êtes amené à modifier ou compléter la librairie, il est important de renommer les fichiers associés afin de ne pas confondre votre code avec sa version originelle.

#### 2.4.4 Détermination de la valeur d'un prescaler

Comme expliqué précédemment, le prescaler a pour rôle de diviser la fréquence d'horloge  $F_{cy}$  et donc de ralentir la vitesse d'incrémentatation des timers. Sa valeur doit être fixée en fonction :

- de la durée  $\Delta_t$  maximale souhaitée entre deux déclenchements d'actions successives ;
- de l'estimation de l'intervalle de temps maximum observable  $\Delta_t$  entre deux fronts successifs sur ICx en mode input-capture.



Dans les deux cas, le prescaler doit satisfaire :

$$Pr > \frac{\Delta_t F_{cy}}{2^{16}}$$

⚠ *Attention :*

Dès lors que cette condition est respectée, il est préjudiciable de surévaluer la valeur du prescaler sous peine de perdre en résolution temporelle (incrément par seconde) : en particulier, la mesure d'intervalle entre 2 fronts rapprochés sur ICx peut devenir imprécise.

## 2.5 Le module de génération d'une modulation de largeur d'impulsion

Le microcontrôleur dsPIC30F4012 intègre un module permettant de générer jusqu'à 6 signaux PWM (Pulse Width Modulation) de période commune dont les sorties sont couplées deux à deux. Le fonctionnement de ce module est précisément décrit en section 15 du *dsPIC30F Family reference manual* [\[7\]](#).

### 2.5.1 Fonctionnement du module PWM

Chaque broche PWMxH et PWMxL,  $x \in \{1, 2, 3\}$  peut, au choix être utilisée par le module PWM ou comme une I/O logique. La période de la modulation est commune et produite par un unique timer 16 bits programmable intégré au module PWM et incrémenté au rythme  $\frac{F_{cy}}{N}$  où  $N \in \{1, 4, 16, 64\}$  désigne une valeur de prescaler programmable.

Dans le mode *Edge aligned*, l'ensemble des sorties PWMxH utilisées passent à l'état haut lorsque le compteur du timer PTMR atteint sa valeur finale. En revanche, chaque sortie passe et à l'état bas lorsque le timer atteint une valeur spécifiée dans le registre DCYx pour cette voie comme le montre la figure 9. Ce fonctionnement permet de générer un signal MLI de période et de rapport cyclique souhaités.

Lorsque des voies PWMxL sont utilisées par le module PWM, elles fonctionnent en opposition des voies PWMxH associées à moins que le mode *independent output* n'ait été explicitement requis. Ce fonctionnement complémente facilite la commande des hacheurs en interdisant aux transistors haut et bas d'un même bras de pont d'être simultanément passants ou bloqués (avec l'adjonction d'un éventuel temps mort programmable).

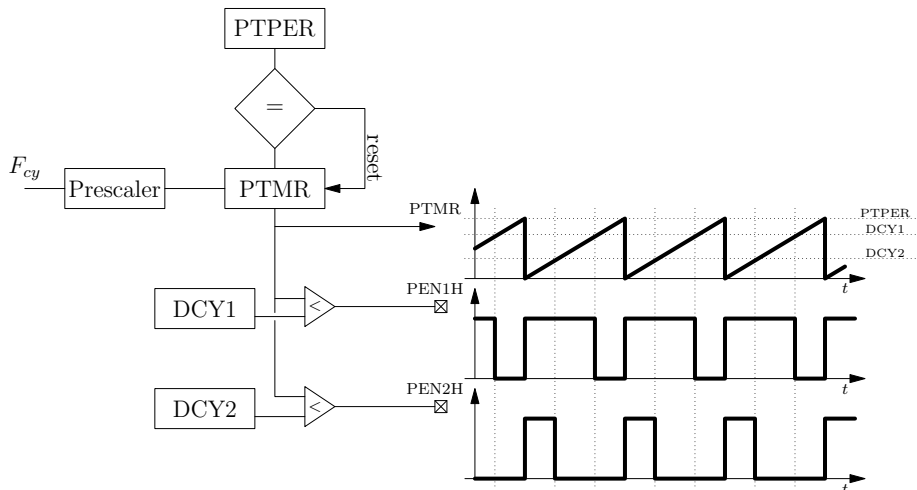


FIG. 9 : Fonctionnement du module PWM

### 2.5.2 Programmation du module PWM

Les fichiers `periph_pwm.c` et `periph_pwm.h` présents sur le cours Moodle offrent une librairie de fonctions de base permettant de programmer le module PWM. Les fonctions disponibles sont :

- `InitPWM(unsigned int period, enum PWM_PIN_INIT_FLAGS PIN_Init)` : initialise le module PWM pour un signal de période  $\text{period} \times 2.17\mu\text{s}$ . `PWM_PIN_INIT_FLAGS PIN_Init` fixe les broches à utiliser pour la PWM et doit être choisi selon une combinaison (et logique) de `PEN1L`, `PEN2L`, `PEN3L`,

PEN1H, PEN2H et PEN3H. Il est aussi possible de dissocier les broches L et H de la voie  $i$  (*independent mode*) en adjoignant l'élément INDEPi.

Ainsi `InitPWM(460,PEN1L&PEN1H)` initialise une PWM de période 1ms dont les sorties sont en opposition sur PEN1L et PEN1H.

- `PWMEnable(enum MOD_ENABLE onoff)` : active le module PWM.
- `PWM1PulseWidth(unsigned int Ton)` : fixe le rapport cyclique sur la voie 1 selon  $\alpha = \frac{\text{Ton}}{\text{period}}$ .
- `PWM2PulseWidth(unsigned int Ton)` : fixe le rapport cyclique sur la voie 2.

## 2.6 Le module de conversion analogique-numérique

Le dsPIC30F4012 possède 9 entrées analogiques multiplexées vers quatre échantillonneurs bloqueurs étiquetés CH0 à CH3. Les sorties de ces bloqueurs sont quantifiées séquentiellement sur 10 bits par un unique convertisseur analogique-numérique à approximations successives. L'étendue de mesure des données converties est paramétrable

- soit entre AVSS et AVDD (images filtrées de la tension d'alimentation du microcontrôleur) i.e. entre 0 et 5V
- soit entre Vref- et Vref+ qui doivent cependant rester tels que  $\text{AVSS} \leq \text{Vref-} < \text{Vref+} \leq \text{AVDD}$

Comme présenté sur la figure 10, le multiplexeur 1 permet de choisir les voies à numériser, en mode *single ended* (référéncées à la masse) ou *différentiel* (différence de potentiel entre deux ANx). Toutes les combinaisons d'entrées ne sont pas disponibles et on se référera à la documentation du microcontrôleur [☞](#) pour plus de détails. Lorsque les entrées à numériser sont connectées à différents bloqueurs, cela rend possible l'acquisition d'une image simultanée des tensions présentes sur ces entrées. L'unique convertisseur analogique-numérique est alors successivement connecté à chaque bloqueur (multiplexeur 2) afin de quantifier les tensions bloquées. Les valeurs converties sont ensuite stockées dans les registres ADCBUFx,  $x \in \{0, \dots, 15\}$ .

Selon la programmation du module, il est possible de numériser en séquence une même voie ou plusieurs voies bloquées simultanément. De même les registres ADCBUFx peuvent être utilisés comme une FIFO ou au contraire en correspondance (stockage de la valeur associée à ANi dans ADCBUFi).

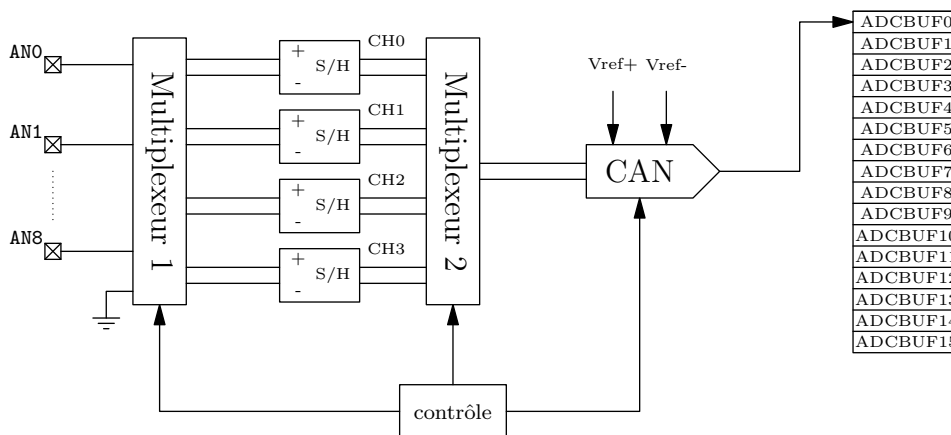


FIG. 10 : Fonctionnement du module de conversion analogique-numérique

### 2.6.1 Programmation du module de conversion analogique-numérique

La programmation du module de conversion analogique-numérique est grandement facilitée par la librairie constituée des fichiers `periph_adc.c` et `periph_adc.h` présents sur le cours Moodle de la matière. Les fonctions fournies sont les suivantes :

- `InitADC0(void)`, `InitADC0and1(void)`, `InitADC4and5(void)` et `InitADC0to3(void)` sont destinées à initialiser le module pour une acquisition simultanée sur les voies AN indiquées.

- `InitADC0Seq0to4(void)` et `InitADC0Seq0to5(void)` sont destinées à initialiser le module pour une acquisition séquentielle des voies AN indiquées.
- `StartADC0(unsigned int* ADCxValue0)` bloque la valeur analogique présente sur AN0 puis lance la quantification de la voie bloquée et attend la fin de conversion. Le résultat est stocké dans la variable `ADCxValue0` qui doit être déclarée préalablement à l'appel de la fonction (passage par adresse).
- `StartADC0to3(unsigned int* ADCxValue0, unsigned int* ADCxValue1, unsigned int* ADCxValue2, unsigned int* ADCxValue3)` bloque **simultanément** les valeurs analogiques présentes sur AN $i$   $i \in \{0, 1, 2, 3\}$  puis lance successivement la conversion de chaque voie bloquée et attend la fin de l'ensemble des conversions. Les résultats sont stockés dans les variables `ADCxValue $i$` ,  $i \in \{0, 1, 2, 3\}$  qui doivent être déclarée préalablement à l'appel de la fonction (passage par adresse).
- `StartADC0seq0to4(unsigned int* ADCxValue0, unsigned int* ADCxValue1, unsigned int* ADCxValue2, unsigned int* ADCxValue3, unsigned int* ADCxValue4)` bloque et convertit **successivement** les entrées AN0 à AN4.

## 2.7 Le module de communication série asynchrone

Le module UART (*Universal Asynchronous Receiver Transmitter*) est un contrôleur de communication série présent dans de très nombreux équipements (microcontrôleurs, PC, automates, appareillage de mesure ...) et habituellement associé au standards RS-232. Une communication asynchrone série consiste en l'envoi successif de valeurs binaires transcodées en impulsions électriques sur une paire filaire. Le caractère asynchrone de cette communication provient du fait que le signal d'horloge cadencant la communication n'est pas transmis en parallèle des données : cette horloge doit être reconstituée coté réception à l'aide d'informations *a priori* communes à l'émetteur et au récepteur (le débit par exemple) ou/et de l'observation des trames reçues. Ce format de communication nécessite de fixer certains paramètres identiquement à l'émission et à la réception :


- niveaux électriques ;
- débit symbole (en Bauds) ;
- longueur des trames ;
- signalétique de début et de fin de trame ;
- contrôle d'erreur.

Le premier item est fixé par une norme d'interconnexion électrique telle le standard RS-232. Les autres paramètres sont réglés par programmation de l'UART.

### 2.7.1 Fonctionnement du module UART

La structure du module UART est illustrée sur la figure 11. A l'émission, les données à transmettre sont écrites dans une pile FIFO à 4 emplacements. la donnée la plus ancienne dans la pile est mise en trame (ajout de la signalisation de start, de stop et de l'éventuel bit de parité) puis transférée dans le registre à décalage d'émission dès que ce dernier est vide. La trame est alors immédiatement transmise bit par bit sur Tx $D$  au rythme de l'horloge du *Baud Rate Generator* (BGR). Une interruption peut être déclenchée dès que la pile FIFO est vidée, ce qui permet à l'application de venir à nouveau la remplir.

A la réception du bit de start, le signal électrique acquis sur Rx $D$  est suréchantillonné à un rythme 16 fois plus rapide que l'horloge BGR pour pouvoir se synchroniser au mieux à l'émetteur et les données sont stockées dans le registre à décalage de réception. Lorsque la trame est complète et sans erreur, la donnée utile est transférée dans une FIFO à 4 emplacements et prête à être lue par le programme. Une interruption peut éventuellement être générée lors de l'occurrence de cet évènement.

Le fonctionnement de ce module est précisément décrit en section 19 du *dsPIC30F Family reference manual* .

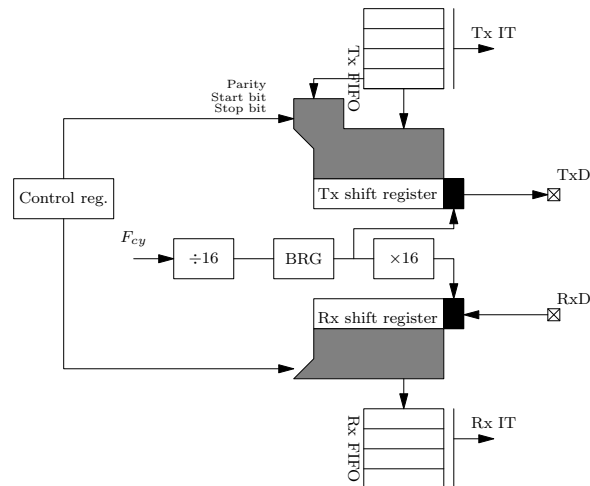


FIG. 11 : Fonctionnement du module UART

### 2.7.2 Programmation du module UART

Afin de faciliter la programmation du module, un ensemble de fonctions sont mises à votre disposition dans les fichiers `uart_periph.c` et `uart_periph.h` présents sur le Moodle du cours. Ainsi :

- `serialInit(baudRate)` permet d'initialiser la liaison UART sur les broches alternatives (broches 11 et 12 sur la figure 4) sans contrôle de parité et avec un débit fixé à `baudRate` ;
- `serialCharReceived()` retourne une valeur non nulle lorsqu'un ou plusieurs octets sont présents dans la FIFO de réception ;
- `serialRead()` lit un octet dans la FIFO de réception et supprime l'octet lu de cette pile ;
- `serialWrite()` écrit un octet dans la FIFO d'émission et attend la fin de transmission.

Cette librairie n'exploite pas le mécanisme d'interruption sur occurrence d'un évènement de réception : par conséquent le programme doit capter l'évènement de réception par polling.