# Les DSpic

Par Jean Marc Masson

Le document qui suit une suite de travaux personnels.

Il est rédigé de la façon suivante :

Volume 1

Partie 1 : notes de différents sites Internet + travail personnel

Partie 2 : travail personnel à partir des notices techniques

Volume 2

Partie 1 :E/S – DMA - Conversion A/D-  modèle mémoire

Partie 2 :  UART – SPI – I2C – USB – Bus CAN

Partie 3 :

- Exemples
- Notes de Mauricio Gomez

Il est possible que malgré plusieurs lectures il subsiste encore quelques fautes.

Il est demandé au lecteur d'être indulgent étantdonné le degré de vision du rédacteur.

Remarque : Les tableaux et les figures sont copiés dans les notices techniques

Quelques références :

- Beginner's guide to programming the PIC24/dsPIC par Thomas Kibalo
- Miconcontrollers from assembly langage to C using the pic 24  familly

  Par Robert Reese/J.W.Bruce/Bryan  A. Jones

- Initiation à la programmation sur dsPIC
  CREMMEL Marcel Lycée Louis Couffignal STRASBOURG
- Logiciels de :

  - Mauricio Emilio Gomez
  - Jean Marc Masson

Les logiciels de Mauricio Gomez portent son nom

Volume 1

Partie 1

Introduction

Le document présent n'a pas la prétention de tout couvrir. Il donne cependant un certain nombre d'explications sur la constitution, le fonctionnement et la configuration des DSpic.

Le document comporte 3 parties :

- Un aperçu donnant les informations de façon générale avec quelques détails
- Une étude des différents modules et des configurations
- Des applications et des configurations particulières liées à des cours

Le technicien ou l'ingénieur se trouve confronté à un certain nombre de choix :

Choix de la structure matérielle du système micro-programmé :

Suivant les applications, 3 choix sont possibles :

- – Microcontrôleur avec CPU, ROM, RAM et périphériques adaptés à l'application intégrés sur la même puce. Les constructeurs proposent un choix très vaste : on trouve des microcontrôleurs à 8 broches pour les applications les plus simples (thermostat par exemple) jusqu'à plus de 100 broches s'il l'application exige un LCD graphique.
  Le choix dépend de l'application, de la vitesse de traitement, de la consommation, du prix et des habitudes de l'entreprise.
- – DSP = Digital Signal Processor. On choisit ces processeurs quand une forte puissance de calcul est nécessaire (ex : codeurs/décodeurs MP3 ou DVD). Ils sont coûteux et gourmands en énergie. Le dsPIC de Microchip est un mixage de µC et de DSP moyennement performant. Il est bien adapté aux applications "audio" et de commandes de moteurs asservis.
- – Ordinateur type PC : c'est une plate-forme universelle, capable de réaliser toutes les fonctions imaginables, mais pas toujours adaptée aux applications. Il existe des PC "industriels" robustes, conçus pour fonctionner dans un environnement agressif.

Choix du langage et de l'outil de développement :
Au final, le µP exécute toujours une suite d'instructions "machines" (ou "assembleur") placéeś dans la mémoire programme à des adresses consécutives.
A l'époque des pionniers, on affectait presque manuellement chaque case de la mémoire par les instructions du programme. La mise au point était très fastidieuse au point d'abandonner souvent en cours de route.
Pour limiter la fuite des clients découragés, les fabricants ont rapidement proposé des outils de développement permettant de raccourcir le temps de mise au point et aussi de faciliter l'élaboration des "gros" programmes en équipes (ex : Windows XP sur Pentium).
Ces outils permettent, avec un ordinateur de bureau :

- – de rédiger le programme dans un éditeur de texte avec toutes ses facilités (copier/coller – fichiers inclus – etc ...). On utilise un des langages décrits ci-dessous.
- – de compiler ce fichier "texte" avec le compilateur pour produire un fichier "binaire". Ce dernier regroupe l'ensemble des instructions machine du programme telles qu'elles seront placées dans la mémoire du microcontroleur. Le transfert dans cette mémoire est réalisé par un programmateur.
- – de simuler le programme à l'aide du simulateur avant son transfert effectif dans la mémoire du microcontrôleur.
  Cette étape est obligatoire, car la première écriture d'un programme comporte toujours des erreurs (bug=cafard), même quand il est très simple.

– de debugger le programme implanté dans l'environnement réel ("in circuit") avec le "debugger".

Les langages proposés sont (pour les applications à base de µC) :
– l'assembleur (ou langage machine) : on est au plus près du µP et on utilise les mnémoniques de ses instructions que l'assembleur va traduire en code "machine". Exemple sur dsPIC :
Travail de l'assembleur
Adresse (16 bits)

0x1E00 0x1E02

| Mnémonique des instructions |
| --- |
| mov #10,W0 mov W0,0x0802 |

| Code machine (placé en mémoire programme organisée en mots de 24 bits) |
| --- |
| 0x2000A0 |
| 0x884010 |

On constate qu'une instruction est codée sur un seul mot de 24 bits dans la mémoire programme. Ce type de codage permet d'augmenter la vitesse de traitement en réduisant le nombre d'accès mémoire par instruction.
On utilise le langage assembleur quand les temps d'exécution et la taille du programme sont critiques.
– le C : c'est le langage le plus utilisé (Windows XP est écrit en C).
L'ingénieur ou le technicien programmeur n'a pas besoin de connaître le jeu d'instruction du µP utilisé pour réaliser un programme performant (rapide et compact). Il utilise des constantes, des variables, des opérateurs, des structures de programmations et d'E/S standardisés.
Le compilateur C, généralement fourni par le fabricant du µP, se charge de traduire le programme en langage "assembleur" puis en code machine.
Ainsi, un programme écrit en C pour un certain µP peut être facilement transposé à un autre µP, même d'un autre fabricant.
Les problèmes subsistants sont généralement liés aux structures des périphériques spécifiques à chaque µC. Mais le temps de conversion est beaucoup plus court que s'il fallait réécrire tout le programme avec de nouvelles instructions.
La difficulté pour nous, électroniciens, est la maîtrise des finesses du langage. Mais c'est le travail des informaticiens. Les compétences d'un électronicien se limitent aux quelques notions nécessaires à l'analyse et à la modification de fonctions microprogrammées simples et liées à des structures matérielles (E/S, interruptions, CAN, CNA, liaisons séries, ...).
Exemple de compilation C sur dsPIC :
Travail du compilateur C
Travail de l'assembleur
Adr. (16 bits)

0x0182 0x0184
0x0186
0x0188 0x018A 0x018C
Lignes de programme en C
Traduction en assembleur
VAR2=VAR2+VAR1;
mov.w 0x0806,0x0002
mov.w 0x0804,0x0000
add.w 0x0002,0x0000,0x0000
mov.w 0x0000,0x0806
VAR_LOC =VAR2<<2;
mov.w 0x0806,0x0000
sl 0x0000,#2,0x0004

| Code machine (24 bits) |
|---|
| 0x804031 |
| 0x804020 |
| 0x408000 |
| 0x884030 |
| 0x804030 |
| 0xDD0142 |

Notes : dans cet exemple :

- ▪ VAR1 et VAR2 sont les identificateurs de 2 variables de taille 16 bits placées en RAM aux adresses respectives 0804h et 0806h
- ▪ VAR_LOC est une variable 16 bits placée dans le registre W4 (adresse 0x0004) du CPU (le dsPIC comporte 16 registres de travail, nommés W0 à W15).
  En fait, le programmeur en C n'a le plus souvent pas à se soucier de ces détails techniques (sauf la taille des variables). Le compilateur fait généralement les meilleurs choix d'affectation en mémoire des variables nécessaires au programme (dans l'exemple donné, la variable VAR_LOC est placée dans un registre CPU ce qui réduit les temps d'accès et augmente par conséquent la vitesse de traitement).

– les autres langages (PASCAL, BASIC, ...) sont très peu utilisés pour développer des applications à base de microcontroleur.

Analyse du "source"

```
#include  <p30f2010.h>
int main(void) {
 TRISE=0xFFF0;  while (1)  {
  LATE = LATE + 1;  }
}
```

CLK
S2
t

– La  première ligne indique au compilateur d'inclure le fichier"p30f2010.h".Ce fichier texte respecte la syntaxe "C" et définit toutes les constantes liées au µC utilisé : entre autres les adresses des registres de configuration et des ports d'E/S. Par exemple :
– TRISE représente le contenu (16bits) de l'adresse 0x02D8 pour un dsPIC30F2010
– LATE représente le contenu (16 bits) de l'adresse 0x02DC pour un dsPIC30F2010
– La structure

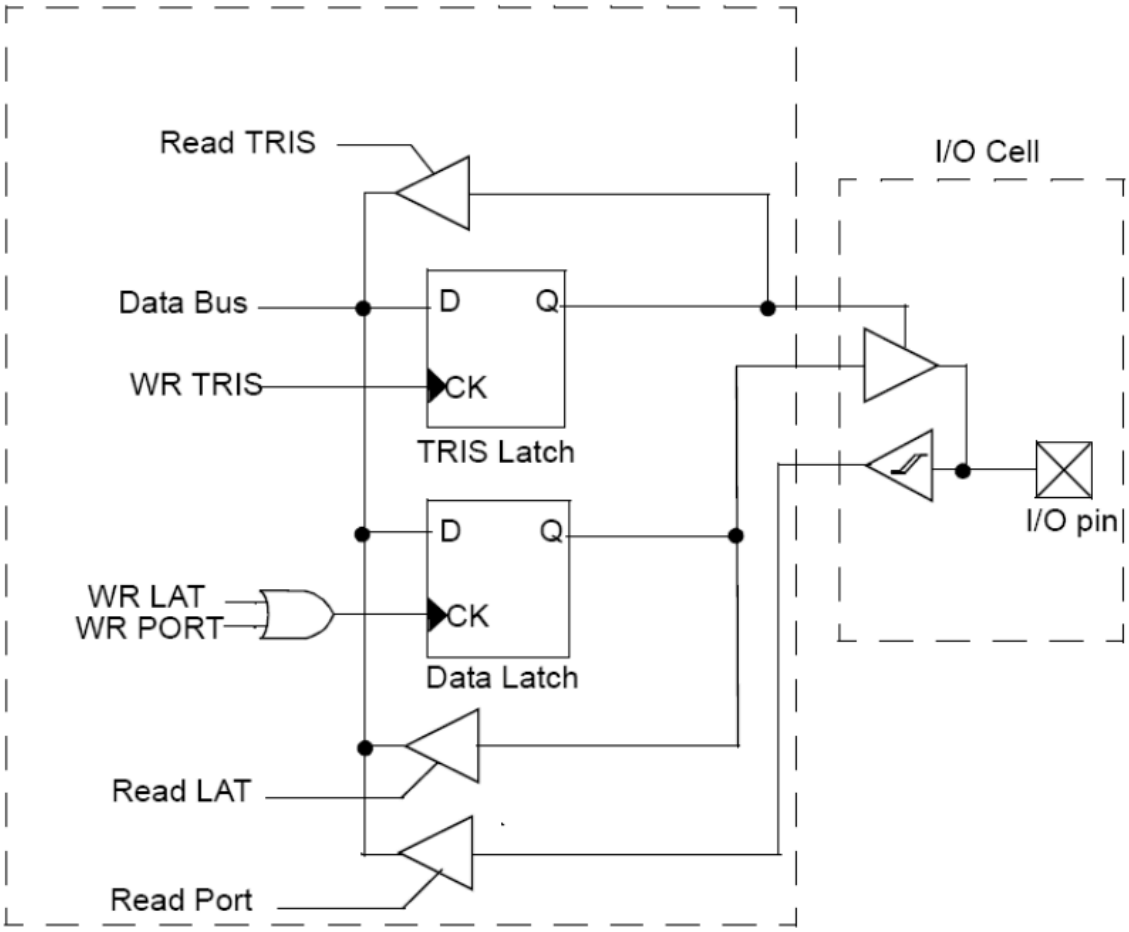
```
void main(void)
{ }
```

– –
–
définit le bloc du programme principal (= main) exécuté au "reset". Le mot "main" ne peut pas être remplacé par un autre. Si cette fonction se termine (pas de boucle sans fin) → "reset".
Note : les accolades {} sont toujours utilisées par paire pour définir le début et la fin d'un bloc de lignes de programme.
Il s'agit d'une affectation : la case mémoire TRISE d'adresse 0x02D8 est  affectée avec la valeur 0xFFF0. Noter le ";" qui est le terminateur de la ligne de programme. Structure : while (1)

```
{
}
```

Le µP exécute en boucle (indéfiniment) les lignes du bloc de programme délimité par les 2 accolades car le test (1) est toujours vrai (toute valeur différente de 0 est synonyme de "vrai"). Le bloc de la structure "while" ne comporte qu'une ligne programme:
D'un point de vue mathématique: elle est fausse ! En fait, il ne s'agit pas d'une égalité, mais d'une affectation : le registre LATE (16 bits)

est affecté avec la valeur actuelle de LATE augmentée de 1. En résumé : à droite de "=" : valeurs actuelles
TRISE=0xFFF0;
LATE = LATE + 1;
à gauche de "=" : valeur future après calculs
Pour bien comprendre le programme, il faut décrire ce que représentent TRISB et LATB dans le µC :




Chaque broche d'E/S du µC comporte la structure dessinée ci-contre.
Les signaux de contrôle "Read TRIS", "WR TRIS", "WR_LAT", "WR PORT", "Read LAT" et "Read Port" sont communs pour toutes les broches (jusqu'à 16) d'un port (A, B, C ...). Ils sont produits par une structure de décodage d'adresse qui reconnaît les adresses correspondantes.
Exemple : si "I/O pin" correspond à la broche RE3 (bit 3 du port E) :

- – Data Bus = D3 interne

- – WR TRIS est activé par une écriture à l'adresse TRISE=0x2D8

– WR LAT est activé par une écriture à l'adresse LATB=0x2DC

- – Read LAT est activé par une lecture à l'adresse LATE=0x2DC
- – Read Port est activé par une lecture à l'adresse PORTE=0x2DA

Les sorties des ports du µC sont de type "3 états" :

- – états logiques "0" ou "1" quand le buffer est actif (TRISEbits.x = "0")
- – état haute impédance ("HiZ") quand le buffer est inactif (TRISEbits.x = "1")
  L'indice "x" représente le bit concerné du port (0 à 15, mais tous sont rarement utilisés). Les états logiques en mode "sortie" sont ceux des bascules "Data Latch"
  A noter que les états des broches sont lus via un buffer à trigger de Schmitt.
- → Comment est configure´le portE après l'exécution de la ligne "TRISE=0xFFF0;"?
- → Comment évoluent les sorties du port E à l'exécution de la ligne "LATE=LATE+1;"

**Configuration générale**

Les DsPic sont de petits microcontrôleurs 16 bits très puissants que l'on peut utiliser pour tout ou presque :

- Contrôle de moteur par PWM,
- Conversion analogique numérique 16 bits (ADC),
- Acquisition de signaux numériques (Input Capture)
- Communication CAN (natif), i2c (natif), UART (natif), SPI (natif), etc.
- Une foultitude de timers,
- etc.

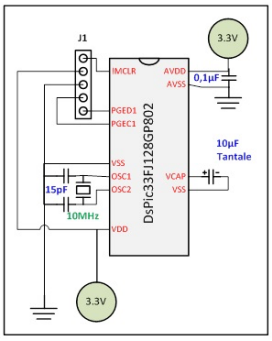
Il existe plusieurs familles de DsPIC : 30F, 32, 33F, 33E. Seuls les 33F sont détaillés ici.

Dans une même famille, tous les microcontrôleurs utilisent les drivers de la même façon, avec le même code, sous réserve qu'ils disposent du driver. En effet, dans la famille des 33F il va exister plusieurs circuits différents, la différence se situant au niveau du nombre de

drivers, d'E/S, de timers, de place mémoire, etc.
Dans la suite, les explications seront apportées sur un DsPIC33FJ128MC802 ou un DsPIC33FJ64GP802, dont la principale différence est un PWM pour le premier. La programmation peut se faire en C ou en assembleur. Les exemples proposés sont en C.

**Schéma de câblage**

Tous les microcontrôleurs de la famille des DsPIC33FJ sont câblés pareils :




Le connecteur J1 est utilisé pour la programmation du DsPic, en utilisant un ICD2 ou un Pickit 3.

1 !MCLR
2 +Vcc
3 GND
4 PGEC1
5 PGED1

Les ports suivants sont nécessairement câblés pour pouvoir utiliser le circuit, conformément à la figure précédente :

- **!MCLR** : La patte sert à la fois à la programmation et à la mise en marche du circuit : !MCLR = 0/NC - Circuit inactif, !MCLR = VDD : Circuit actif.
- **PGEDi/PGECi** : broches de programmation, elles peuvent aussi avoir d'autres applications lors de l'usage du circuit. Il y a plusieurs mappages des PGEC/D possibles, nous utilisons ici le premier.
- **VSS/AVSS** : « ground », 0 logique. La broche "A-" est normalement la broche utilisée pour faire passer de la puissance dans le circuit, mais en règle générale il vaut mieux tout avoir sur la même source et utiliser ensuite des composants fait pour les aspects de puissance.
- **OSC1 / OSC2** : entrées du quartz. Voir le chapitre horloge pour le

détail d'utilisation.

- **VDD/AVDD** : 3,3V (+/- 10%)
- **VCAP** : Cette entrée doit être reliée à la masse par un condensateur de 10µF, l'idéal étant un tantale polarisé (CMS ou non d'ailleurs). Sans ce condensateur, le circuit ne marchera pas et ne pourra même pas être programmé.

**Horloge**

Les DsPIC peuvent être cadencés assez haut, mais ils utilisent pour cela un PLL interne (Phase Lock Loop). Le principe du PLL est détaillé sur internet, nous ne verrons ici que l'application. Trois types d'horloges sont utilisées :

- Quartz,
- Oscillateur,
- Horloge interne.

De manière générale, il est déconseillé d'utiliser des horloges trop rapides. En effet, les signaux "carrés" sont plus mauvais à ces fréquences. Une bonne pratique est de dédier un quartz de fréquence raisonnable (8 ou 10 MHz) à chaque circuit, et d'en augmenter la fréquence par un PLL.

**Utilisation d'un quartz**

Les quartz sont des cristaux passifs que le microcontrôleur utilise pour générer une horloge. Le quartz est relié aux broches OSC1 et OSC2 du microcontrôleur. Les quartz utilisables sont de deux types, en fonction de leur vitesse d'horloge :

- XT : quartz jusqu'à 10 MHz,
- HS : quartz de 10 MHz à 40 MHz.

Chaque broche OSC1 et OSC2 est également reliée à la masse (GND), par l'intermédiaire d'un condensateur de 15 µF. A noter que ça marche aussi sans le condensateur, mais ils sont recommandés dans la documentation du circuit.

Dans : **MPLAB** : Configure\configuration bits...

- Oscillator mode : Primary oscillator (XT, HS, EC) w/ PLL
- Internal External switch over mode : Start up device with FRC, then automatically switch to ...
- Primary oscillator source : HS Oscillator mode (pour cause de quartz utilisé à 10 MHz)

*Nota* : les quartz étant passifs, ils sont dédiés à un seul circuit et unique circuit, on ne peut pas utiliser un quartz pour plusieurs circuits.

**Utilisation d'un oscillateur**
ADU
**Utilisation de l'horloge interne**
ADU
**Configuration de l'horloge**
Ci-dessous un bout de code permettant de paramétrer le PLL au démarrage du circuit :

```
void oscConfig(void)
{
PLLFBD=30;
// M=32
CLKDIVbits.PLLPOST=0;
// N1=2
CLKDIVbits.PLLPRE=0;
// N2=2
OSCTUN=0;
// Tune FRC oscillator, if FRC is used
RCONbits.SWDTEN=0;
// Disable Watch Dog Timer
__builtin_write_OSCCONH(0x03);
// Initiate Clock Switch to Primary Oscillator with PLL (NOSC=0b011)
__builtin_write_OSCCONL(0x01);
// Start clock switching
while (OSCCONbits.COSC != 0b011);
// Wait for Clock switch to occur
while(OSCCONbits.LOCK!=1) {};
// Wait for PLL to lock
}
```

Ce code est prévu pour un quartz de 10 MHz, permettant de passer la fréquence de cycle du microcontrôleur à 40 MHz (une opération toutes les 1 / [40.E6], soit 25 ns).
**Configuration des ports d'entrées / sorties**
La configuration des ports d'E/S est réalisée de base par les trois registres suivants :

- **AD1PCFGL'** : Ports analogiques, à positionner à [0xffff] pour les désactiver,
- **_TRISBx / _TRISAx** : Direction de la broche RBx / RAx :
- 1 : broche en entrée,
- 0 : broche en sortie.
- **_RBx / _RAx** : Etat de la broche :

- 1 : broche à VDD (+3,3V),
- 0 : broche à VSS (GND).

*Nota* : Lorsque l'on passe plusieurs broches à l'état logique 1 l'une à la suite de l'autre, notamment avec des très hautes fréquences, il est possible que le condensateur qui sert à modifier l'état de la broche ne puisse pas suivre et alimenter toutes les broches. Dans ce cas, il faut séparer les opérations de changement d'état de plusieurs cycles (au moins deux ou trois) pour permettre au condensateur de recharger.
En plus des manipulations standards, il existe deux autres opérations associées à l'utilisation des drivers : redirection des ports d'entrée ou de sortie : **_RPxR** (output) et **RPINRx** (input).

- **_RPxR** : les _RPxR sont des registres associés à **ADU**
- **RPINRx** : la plupart des drivers ont leurs entrées redirigeables vers l'une ou l'autre des broches RPx du circuit. Pour chaque driver, il faut trouver le bon RPINRx et indiquer dans ce registre le numéro de la broche à laquelle associer l'entrée.

Pour chaque driver, les redirections de port seront précisées au cas par cas.

## Les timers

Les timers sont probablement les fonctions les plus utiles d'un microcontrôleur. Ils permettent de réaliser des opérations avec une fréquence particulière, avec des actions spécifiques à chaque révolution du timer. Le principe est qu'un registre va être incrémenté régulièrement, proportionnellement à la fréquence Fcy, jusqu'à une certaine valeur. Lorsque cette valeur est atteinte, une interruption est déclenchée qui permet de réaliser l'action voulue à la révolution du timer.
Les DsPIC disposent de plus ou moins de timer selon les séries, celui utilisé ici dispose de 5 timers. Nous verrons plus loin les détails. Certains timers sont utilisés par des fonctions (comme les timers 2 et 3 avec l'Input Capture, définit plus loin), ou peuvent être combinés pour former des timers 32 bits (de révolution très largement supérieure). Les applications présentées ici utiliseront plusieurs de ces fonctions, mais sans être exhaustif.

### Principe

Voir plus loin

### Paramétrage du timer

Le code ci-dessous permet de paramétrer un timer pour avoir une interruption cadencée à la fréquence voulue :

```
void init_timer1(void)
```

{
_T1IE=0; // interruption timer
_T1IF=0; // flag d'interruption remis à zéro
T1CON=0x0020;
/* configuration du registre de contrôle :
Le bit de poids fort <Bit_15> commande le démarrage du timer,
Les bits <5-4> permettent de ralentir la vitesse du timer, en divisant la fréquence de mise à jour du timer par une valeur spécifique (ce que l'on appelle un prescaler. Ainsi, au lieu d'être incrémenté à chaque coup d'horloge (Fcy), le compteur du timer est incrémenté tous les 8, 64 ou 256 coups :

- <5-4> = 0 ; // A chaque coup d'horloge
- <5-4> = 1 ; // tous les 8 coups
- <5-4> = 2 ; // tous les 64 coups
- <5-4> = 3 ; // tous les 256 coups */

PR1= <<valeur>>;
// Lorsque le compteur du timer atteint cette valeur, une interruption est générée et le timer remis à 0
T1CONbits.TON=1; // démarrage du timer
}
Pour faciliter la manipulation du timer, on définit en général la donnée suivante :
**#define timer1 IEC0bits.T1IE**
Avec 1 pour le timer 1, 2 pour le timer 2 et ainsi de suite. Cela permet au microcontrôleur de générer les interruptions, ce qu'il ne fait pas sinon.
*Nota 1* : même si les interruptions sont désactivées, le timer fonctionne si T1CONbits.TON=1. *Nota 2* : vérifier que le registre IEC0 contient bien le bit du timer associé. En particulier, pour les timer 4 et 5 il faut chercher sur le registre IEC1 :
**#define timer4 IEC1bits.T4IE**
A titre informatif, le tableau ci-dessous donne quelques réglages de base pour générer des interruptions aux périodes indiquées, sur la base de réglage de la PLL et du quartz utilisé :
**Tableau**

| | |
|---|---|
| Horloge | 10 MHz |
| M | 38 |
| N1 | 0 |

| | |
|---|---|
| N2 | 0 |
| FCAN | 50 MHz |
| Période | 20 ns |

**Interruption timer**

Une interruption timer est définie dans la routine suivante :

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt (void)
{
TMR1=0; // remise à 0 du compteur
Ajouter le code ici
IFS0bits.T1IF=0; // remise à 0 du drapeau d'interruption du timer
}
```

*Nota 1* : le code ne doit pas être trop long, sinon son exécution peut masquer la survenue d'autres interruptions (voir priorité des interruptions), *Nota 2* : TMR4 et TMR5 sont sur IFS1.

**Cas d'utilisations des timers**

Les timers sont utilisés dans les applications qui suivent dans plusieurs cas :

- Génération de la commande d'un servomoteur (pulse d'une milliseconde sur une période de 20ms),
- Génération d'un signal pour LED infrarouge conforme protocole IRDa,
- Commande d'un moteur pas à pas avec une carte de commande,
- Etc.

Ces différents cas seront détaillés dans la description complète des fonctions, au niveau des applications elles-mêmes.

**Input Capture et interruptions externes**

Il existe deux fonctions assez similaires qui réagissent à un changement d'état d'une entrée du microcontrôleur en générant une interruption. Ces fonctions sont :

- Input Capture, permettant via un couplage aux timers de mesurer précisément le temps écoulé entre deux changements d'état,
- Interruptions externes : principalement utilisées pour du comptage d'occurrence de changement d'état.

**Paramétrage de l'Input Capture**

La fonction d'Input Capture (IC) est très utile pour la mesure précise d'une période entre deux changements d'états d'une entrée, sur le même front ou sur des fronts différents. Il est également possible d'utiliser, comme pour les timers, un prescaler générant une interruption au bout de la 4eme, 16eme, etc. survenue du changement d'état attendu.

```
void init_input_capture(void)
{
IC1CON = 0x0001;
IC7CON = 0x0001;
RPINR7 = 0x000B;
RPINR10 = 0x000A;
_IC1IF = 0;
_IC1IE = 1;
_IC7IF = 0;
_IC7IE = 1;
}
```

Le registre ICxCON permet le paramétrage du module d'input capture. Dans notre cas, seuls les paramètres suivants sont intéressants :

- Bit 7 – ICTMR : lorsque l'interruption survient, le contenu d'un timer est recopié dans un registre associé à l'input capture. Si ICTMR = 1, c'est le contenu du timer 2 (ie TMR2) qui est recopié dans le registre, sinon c'est le contenu du timer 3 (ie TMR3),
- Bit <6 – 5> - ICI<1:0> : décale la génération de l'interruption de 1 à 4 survenue de l'évènement scruté. Par exemple, si ce paramètre vaut 3, et que l'IC est programmée pour détecter tous les fronts montants sur une broche, l'interruption sera générée au troisième front montant et non à chaque fois,
- Bit <2 - 0> - ICM<2:0> : précise l'évènement scruté. Un paramètre à 0 désactive l'IC. Un paramètre à 1 génèrera une interruption à chaque front montant ou descendant. Dans ce mode le décalage de l'ICI<1:0> ne fonctionne pas. Un paramètre à 2 ou plus entrainera une interruption, après décalage de ICI<1:0>, sur chaque front montant, chaque front descendant, tous les 4, 16 front montants, etc. se reporter à la notice du composant.

Dans le paramétrage de l'exemple, l'IC va générer une interruption à chaque front montant ou descendant.

Il faut ensuite associer à chaque IC une broche, et on utilise pour ça le registre RPINR vu précédemment. Dans l'exemple, le RPINRx de l'IC1 est RPINR7 octet de poids faible, auquel on a associé la valeur 11 (0xB). Ainsi, l'entrée de l'IC1 est la broche RP11 (ie RB11).

Enfin, de la même manière que pour les timers, il est nécessaire d'activer les interruptions sur l'IC :

- _IC1IE = 1;

Après s'être assuré au préalable que l'interruption n'est pas en cours,

sans quoi le code ne sera jamais appelé :
- _IC1IF = 0;

**Interruption Input Capture**

Une interruption d'input capture est définie dans la routine suivante :

```
void __attribute__((interrupt, no_auto_psv)) _IC1Interrupt(void)
{
Value = IC1BUF;
// Ajouter TMR3 = 0 ou TMR2 = 0 si il est nécessaire de
recommencer à compter depuis 0 (et ainsi éviter les rollovers)
_IC1IF = 0;
}
```

La valeur du timer 2 ou 3 (selon réglage) est copiée dans le registre ICxBUF, mais pas remise à 0. Si besoin, il est possible de la remettre à 0 en utilisant directement les registres de comptage TMRx. Noter qu'il est possible, pour « by-passer » une limitation du circuit, d'utiliser le TMR d'un autre timer, sous réserve que celui-ci soit remis à 0 juste après et qu'aucun autre soft ne s'en serve.

**Paramétrage de l'interruption externe**

Les interruptions externes sont en réalité une extension des fonctions d'entrée / sortie, mais avec la génération d'une interruption à chaque front montant. Si la fonction ressemble beaucoup à l'IC, elle est cependant limitée sans possibilité de prescaler. A noter enfin qu'il ne vaut mieux pas utiliser l'INT0 qui est très haute dans la liste des priorités d'interruption, et qui peut donc rapidement poser problème pour l'exécution du code.

```
void init_interrupts(void)
{
RPINR0 = 0x0900;
// RPINR0 => octet de poids fort pour le choix de la broche associée
à l'interruption 1 (INT1), donc ici sur la broche RP9 (RB9)
_INT1EP = 0;
_INT1IF = 0;
// vérification que l'interruption n'est pas en cours
_INT1IE = 1;
// Activation de l'interruption
}
```

Le bit _INT1EP contrôle le sens de génération de l'interruption :
- 1 – interruption sur le front descendant,
- 0 – interruption sur le front montant.

**Interruption externe**

Une interruption externe est définie dans la routine suivante :

```
void __attribute__((interrupt,no_auto_psv))_INT1Interrupt(void)
{
// Ajouter code ici
_INT1IF = 0;
}
```

## Les communications

### Introduction

Cette première étude est une simple entrée en matière. Nous verrons plus loin dans le volume 2 une étude plus détaillée.

Les DsPIC33FJ possèdent plusieurs modes de communication gérés automatiquement par le circuit, qu'il ne reste qu'à paramétrer :

- I2C,
- UART,
- ECAN.

Dans le projet décrit par la suite, seule I2C et ECAN seront utilisés, l'I2C en maître et l'ECAN en émission / réception. Les drivers de communication sont définis dans des fichiers indépendants :

- ecan.c / ecan.h
- i2c_Func.c / i2c.h

Les fichiers ne sont pas disponibles ici car ils appartiennent à leurs créateurs, et donc restent disponibles sous leur contrôle uniquement.

### Utilisation de l'I2C

Dans les applications du projet, le circuit est utilisé comme maître I2C pour lire des télémètres à ultrason.

Il est possible d'utiliser l'i2C pour la communication globale avec le circuit. Cependant, il faut penser que l'I2C n'est pas fait pour la communication à distance (de l'ordre du mètre et plus), en particulier dans un environnement CEM sévère (par exemple avec de puissants moteurs brushless à proximité).

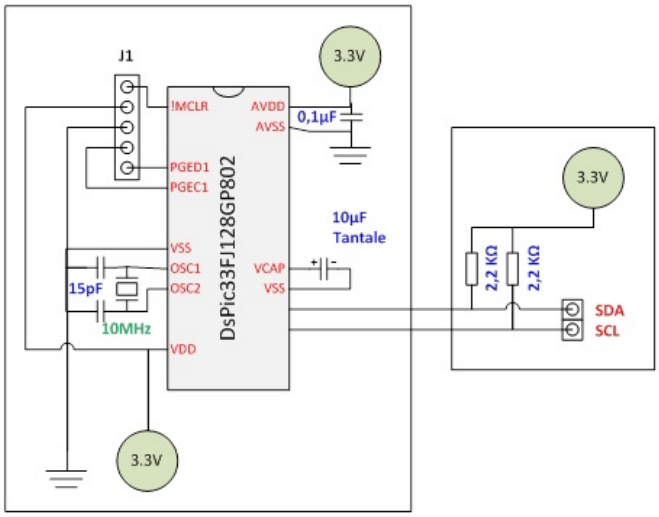
C'est pourquoi dans les applications du projet, les communications internes seront réalisées en ECAN (plus robuste), et l'I2C limité à l'interface avec les composants qui ne comprennent que ça (typiquement les sonars à ultrason).

### Câblage de l'I2C

Le principe de l'i2c est qu'il est possible d'utiliser des composants sur plusieurs niveaux de tension, par exemple entre les DsPIC33F (en 3,3V) et les capteurs à ultrason (en 5V).

Pour cela, on utilise sur chaque ligne des résistances de pull-up,

portant la tension de la ligne à la tension nécessaire pour que tout le monde comprenne (c'est un bus, donc il peut y avoir plusieurs composants). Tous les **sda** (serial data) sont à relier ensemble, tous les **scl** (serial clock) aussi.




**Paramétrage de l'I2C**

La fonction I2C est initialisée facilement (en mode maitre) à partir de la fonction donnée dans le fichier i2c_Func.c : InitI2C().
Cette fonction peut cependant être (légèrement) modifiée :

- I2C1BRG : ce registre permet de modifier la fréquence de communication de l'I2C. L'I2C fonctionne principalement en 140 KHz et 400 KHz. A noter, puisque c'est le maître qui donne l'horloge, il est normalement possible de mettre la valeur que l'on souhaite dans la limite des 400 KHz max.

Il n'y a pas de redirection (à ma connaissance) sur les broches de l'I2C, il faut nécessairement utiliser les broches RB8 / RB9 ou RB5 / RB6 (I2C complémentaire, utilisable en modifiant les bits de configuration (Configure\ Configuration Bits… \ ALTI2C (Alternate I2C Pins)).

**Interruption sur l'I2C**

En mode maitre, c'est le chip qui décide d'initier la communication au moment opportun. Les esclaves peuvent également initier la communication, mais cette fonction ne sera pas étudiée ici.
Il existe deux fonctions pour la communication I2C :

- Write : écrit un certain nombre de données dans le chip à l'adresse indiquée. La définition des données écrites pour les deux

composants doit faire l'objet d'une ICD .

- Read : Un premier cycle d'écriture va positionner l'esclave en lecture (en général, l'envoi de l'adresse du composant esclave « +1 », par exemple 0xE2 : adresse de l'esclave en écriture, 0xE3 : adresse de l'esclave en lecture), puis un redémarrage permet à l'esclave de transmettre ses données.

La fonction standard d'écriture du fichier i2c_Func.c est utilisable en l'état .

On l'appelle ainsi :

**LDByteWriteI2C(0xE2, 0, 0x51);**

Avec :

- 0xE2 : adresse de l'esclave,
- 0 : adresse de démarrage pour l'écriture des données,
- 0x51 : la (les) valeur(s) à écrire.

Pour la fonction de lecture, nous avons modifié légèrement la fonction donnée dans le fichier.

On l'appelle ainsi :

**LDByteReadI2C(0xE2, 0x02, i2c_buffer_out_1, 4);**

Avec :

- 0xE2 : adresse de l'esclave,
- 0 x02: adresse de démarrage de la lecture des données,
- I2c_buffer_out_1 : une structure de type [unsigned char i2c_buffer_out_1[20]] permettant de récupérer les données en lecture,
- 4 : le nombre de bits à lire.

La fonction est modifiée comme suit :

```
unsigned int LDByteReadI2C(unsigned char ControlByte, unsigned char Address, unsigned char *Data, unsigned char Length)
{
IdleI2C();
//wait for bus Idle
StartI2C();
//Generate Start Condition
WriteI2C(ControlByte);
//Write Control Byte
IdleI2C();
//wait for bus Idle
WriteI2C(Address);
//Write start address
IdleI2C();
```

```
//wait for bus Idle
RestartI2C();
//Generate restart condition
WriteI2C(ControlByte | 0x01);
//Write control byte for read
IdleI2C();
//wait for bus Idle
getsI2C(Data, Length);
//read Length number of bytes
NotAckI2C();
//Send Not Ack
StopI2C();
//Generate Stop
}
```

De plus, avant de démarrer une communication, il faut forcer les esclaves à revenir en idle mode. Ajouter les deux lignes suivantes dans le Main :

- StartI2C();
- StopI2C();

En règle générale ça marche…

**Utilisation du CAN**

Le bus CAN est plus robuste à la CEM que l'I2C, il est donc plus judicieux de l'utiliser au lieu de l'i2c pour toutes les communications sur lesquelles on a le choix.

Le protocole CAN est standardisé, le plus simple est de choisir des composants qui le possèdent en natif.

DsPIC33F : il faut choisir dans la gamme des composants qui ont de l'ECAN en natif. C'est le cas notamment des composants 64MC802.

Avant toute chose, il y a deux choses à savoir sur le CAN indépendamment de sa programmation :

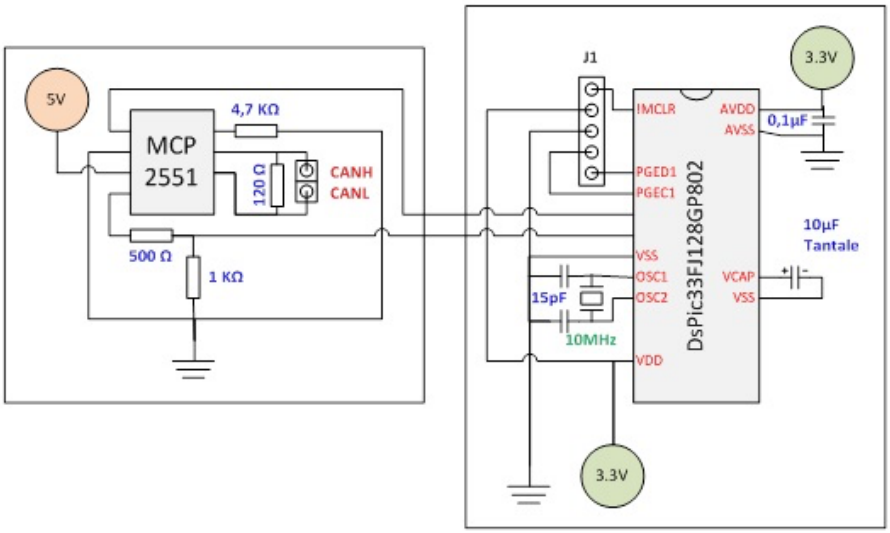
1 Il est impératif d'utiliser un driver matériel indépendant (convertissant le can+/can- en CAN_H / CAN_L), par exemple des MCP2551 de chez microchip,

2 Il est impératif d'avoir au moins deux composants lorsque l'on utilise un réseau CAN. En effet, un seul composant a besoin qu'on lui réponde pour compléter son cycle de transmission de données.

Sans ces deux précautions, il est impossible de faire marcher un module CAN.

**Câblage de l'ECAN**

Comme expliqué précédemment, il n'est pas possible de câbler deux composants par leurs can+ / can- directement. Il faut les câbler par leurs CAN_H / CAN_L après ajout d'un driver matériel et surtout d'un composant en face. Le schéma ci-dessous précise le montage, tous les CAN_H sont à relier ensemble, les CAN_L ensemble également.




**Attention, le MCP2551 se connecte en 5V uniquement. Il est aussi très sensible aux variations sur le 5 volts et grille facilement, il faut protéger le réseau 5V si d'autres éléments électromécaniques (servomoteurs par exemple) sont présents dessus.**

**Paramétrage du module CAN**

Dans un réseau CAN, les composants échangent des données sous la forme de trames de 8 octets. Le principe est sensiblement différent de l'i2c, dans lequel on adresse un composant puis une adresse de données dans le composant, avant de lire ou d'écrire à cette adresse. Dans le CAN, on adresse des données.

Le principe est que chaque « type » d'information a sa propre adresse. En effet, un composant va transmettre une information avec une adresse indiquant à la fois le type et l'origine de l'information. Par exemple, un détecteur va transmettre régulièrement ses données sur le réseau CAN avec une référence (le terme est plus explicite qu'adresse) propre à la fois au composant et au type de données qu'il transmet. Ensuite, tous les composants du réseau qui doivent lire ces données sont paramétrées pour filtrer les données

échangées sur le réseau et lire précisément celle-ci.
Le réseau va traiter les cas de collision automatiquement (par un jeu magique de comparaison des bits dans les références, on va dire que c'est sans objet dans l'application courante).
Comme pour l'i2c, il est possible d'utiliser le code tout fait développé par microchip et contenu dans les fichiers ***. L'initialisation se fait par les fonctions :
initECAN();
initDMAECAN();
Il faut nécessairement utiliser la DMA dans l'exemple. On peut faire sans mais je n'ai pas creusé plus loin.
Pour l'initialisation de l'ECAN (ECAN pour extended CAN, possibilité d'utiliser des références sur 29 bits au lieu de 11 pour avoir plus de messages) il faut paramétrer les filtres et masques qui permettent de limiter les données lues. Dans l'exemple, nous n'utilisons que les adresses standards (sur 11 bits).
Les lignes à modifier sont :

- C1RXM0SID=CAN_FILTERMASK2REG_SID(0x15E) : On applique un masque permettant de lire tous les bits de 350 à 360 (0x15E *** ADU).
- C1RXF0SID=CAN_FILTERMASK2REG_SID(0x0x15E) : adu.

De même, il est possible de modifier la vitesse de transmission via la valeur BRP_VAL définit dans le ecan.h :
**#define BRP_VAL ((FCAN/(2*NTQ*BITRATE))-1)**
Avec :
**#define FCAN 40000000**
// Fréquence de cycle, Fcy en réalité.
**#define BITRATE 125000**
// vitesse de communication
**#define NTQ 20**
// 20 Time Quanta in a Bit Time (ne pas changer)
C'est le paramètre BITRATE qui modifie la vitesse de communication sur le réseau CAN. Hélas, la valeur de 125 kHz n'est pas choisie au hasard, elle a été codée en dur dans le module linux can.ko. Donc il faut nécessairement utiliser cette valeur.
Les ports de communication utilisés sont les broches RB2 et RB3. Pour l'association de ces broches au driver CAN, on utilise deux registres de redirection :

- _TRISB2 = 0;
- // Broche RB2 en sortie

- _RP2R = 0x10;
- // C1TX = RB2, allocation de la broche RB2 à la fonction TxCAN
- _TRISB3 = 1;
- // Broche RB3 en entrée
- RPINR26 = 0x03;
- // C1RX = RB3, allocation de la broche RB3 à la fonction RxCAN

**Interruption sur CAN**

La fonction suivante est directement issue de l'application note *** :

```
void __attribute__((interrupt,no_auto_psv))_C1Interrupt(void)
{
/* check to see if the interrupt is caused by receive */
if(C1INTFbits.RBIF)
{
/* check to see if buffer 1 is full */
if(C1RXFUL1bits.RXFUL1)
{
/* set the buffer full flag and the buffer received flag */
canRxMessage.buffer_status=CAN_BUF_FULL;
canRxMessage.buffer=1;
}
/* check to see if buffer 2 is full */
else if(C1RXFUL1bits.RXFUL2)
{
/* set the buffer full flag and the buffer received flag */
canRxMessage.buffer_status=CAN_BUF_FULL;
canRxMessage.buffer=2;
}
/* check to see if buffer 3 is full */
else if(C1RXFUL1bits.RXFUL3)
{
/* set the buffer full flag and the buffer received flag */
canRxMessage.buffer_status=CAN_BUF_FULL;
canRxMessage.buffer=3;
}
else;
/* clear flag */
C1INTFbits.RBIF = 0;
}
```

```
else if(C1INTFbits.TBIF)
{
/* clear flag */
C1INTFbits.TBIF = 0;
}
else if(C1INTFbits.ERRIF)
{
C1INTFbits.ERRIF = 0;
C1TR01CONbits.TXREQ0=0;
}
/* clear interrupt flag */
IFS2bits.C1IF=0;
}
```

La routine d'interruption est composée de trois parties :

- La première [C1INTFbits.RBIF] est liée à une interruption suite à la réception de données. Dans ce cas, l'un des trois buffers de réception est rempli et le flag de réception levé [canRxMessage.buffer_status > 0]. On récupère les données via la fonction rxECAN(&canRxMessage) dans le corps de la fonction main.
- La seconde [C1INTFbits.TBIF] est liée à la transmission de données. L'interruption est levée dès lors qu'une transmission s'est bien passée. Ne sert à rien dans l'application, mais y ajouter une LED qui clignote permet de savoir rapidement que le réseau CAN est opérationnel.
- La dernière n'est pas active par défaut, elle permet le traitement des erreurs [C1INTFbits.ERRIF].

Pour activer la génération des interruptions, il faut ajouter les lignes suivantes dans le corps de la fonction main :

```
/* Enable ECAN1 Interrupt */
IEC2bits.C1IE=1;
/* enable Transmit interrupt */
C1INTEbits.TBIE=1;
/* Enable Receive interrupt */
C1INTEbits.RBIE=1;
/* Activer les interruptions sur erreur */
C1INTEbits.ERRIE = 1;
```

La récupération des données à traiter se fait par la fonction :

```
rxECAN(&canRxMessage);
```

La structure canRxMessage est également définie dans l'application

note ***, reprise ci-dessous :

```
/* message structure in RAM */
typedef struct{
/* keep track of the buffer status */
unsigned char buffer_status;
/* RTR message or data message */
unsigned char message_type;
/* frame type extended or standard */
unsigned char frame_type;
/* buffer being used to send and receive messages */
unsigned char buffer;
/* 29 bit id max of 0x1FFF FFFF
* 11 bit id max of 0x7FF */
unsigned long id;
unsigned char data[8];
unsigned char data_length;
}mID;
```

La création de la structure est réalisée comme suit :

- mID canTxMessage;
- mID canRxMessage;

Avec une structure pour la transmission de message et l'autre pour la réception.

Pour la transmission, il faut utiliser la fonction suivante :

- sendECAN(&canTxMessage);

La structure canTxMessage est instanciée directement dans le code. Enfin, les deux paramètres message_type et frame_type sont initialisés avec les constantes suivantes dans l'exemple :

```
canTxMessage.message_type=CAN_MSG_DATA;
// Le message est un message de données simple
canTxMessage.frame_type=CAN_FRAME_STD;
// L'ID du message est standard, sur 11 bits.
```

**Les signaux analogiques**

Les microcontrôleurs sont des composants logiques, c'est-à-dire que toutes les données qu'ils traitent sont des données logiques (« 0 » / « 1 ») dont le niveau électrique dépendant de la technologie (0/5V ou 0/3.3V). Il n'est donc, à priori, pas possible de commander un équipement avec un signal analogique variant entre 0 et 5V par exemple, ou de disposer d'une entrée codant son information sur son niveau électrique.

Pour palier à cela, il existe deux fonctions complémentaires :

- ADC : Analogic to Digital Converter, ou convertisseur analogique / numérique. Le fonctionnement détaillé d'un ADC ne sera pas abordé ici, seule sa mise en œuvre sera traitée.
- PWM : Pulse Width Modulation, ou modulation par amplitude d'un signal carré. De même, seule la mise en œuvre avec un Chip sera abordée ici.

**Convertisseurs Analogiques / Numériques**

Le module ADC va convertir un signal analogique, de niveau 0 à 3.3V ou 5V, en une donnée numérique de type flottant ou entier, signé ou non (**AD1CON1\FORM<1:0>**). Le principe est de réaliser un échantillonnage du signal à une fréquence plus ou moins haute, puis de convertir ce signal de manière régulière ou évènementielle (**AD1CON1\SSRC<2:0>**).

Ce module utilisera obligatoirement les « PIN » du circuit ayant une capacité de conversion analogique en numérique, à savoir sur le GP/MC802 les ports A0-1, B0-3 et B12-15. Il est en outre possible de choisir les références, positives et négatives, utilisées pour l'échantillonnage du signal (**AD1CON2\VCFG<2:0>**).

```
// Format des données converties : 0 = Entier non signé
AD1CON1bits.FORM = 0;
// Source de l'horloge de conversion. J'utilise ici GP Timer 3 : à chaque échéance du timer, une conversion intervient
AD1CON1bits.SSRC = 2;
// Contrôle de l'échantillonnage : 1 => un nouvel échantillonnage démarre automatiquement après la dernière conversion
AD1CON1bits.ASAM = 1;
// Opération en mode 10 Bits. La donnée est convertie sur 10 Bits. Il est possible d'opérer sur 12 également
AD1CON1bits.AD12B = 0;
// Scan des entrées sélectionnées pour CH0+ lors de l'échantillonnage A
AD1CON2bits.CSCNA = 1;
// Conversion du canal CH0 uniquement
AD1CON2bits.CHPS = 0;
// Horloge de conversion dérivée de l'horloge système (réglable)
AD1CON3bits.ADRC = 0;
// Durée de conversion : (n + 1) fois le temps d'opération du chip Tcy, ici 63 => Tad = 0,025ns * (63+1) = 1,6µs
AD1CON3bits.ADCS = 63;
// Nombre de canaux utilisés pour le scan : ici NUM_CHS2SCAN = 1
```

```
AD1CON2bits.SMPI = (NUM_CHS2SCAN-1);
//AD1CSSH/AD1CSSL: A/D Input Scan Selection Register
//AD1CSSH = 0x0000;
AD1CSSLbits.CSS0=1;
// Enable AN4 for channel scan
//AD1PCFGH/AD1PCFGL: Port Configuration Register
AD1PCFGL=0xFFFF;
// AD1PCFGH=0xFFFF;
AD1PCFGLbits.PCFG0 = 0;
// AN4 as Analog Input
IFS0bits.AD1IF = 0;
// Clear the A/D interrupt flag bit
IEC0bits.AD1IE = 1;
// Enable A/D interrupt
AD1CON1bits.ADON = 0;
// Turn on the A/D converter
```

## partie 2
## Les différents modules du DSpic

### Architecture du microcontrôleur

Les microcontrôleurs de la famille dsPIC30F sont des processeurs 16-bits modifiés d'architecture Harvard avec un jeu d'instructions amélioré. Les instructions ont 24-bits de large. Les registres de travail ont 16-bits de large. Il ya 16 registres de travail; le registre de travail 16 (de W15)

fonctionne comme un pointeur de pile de logiciels.

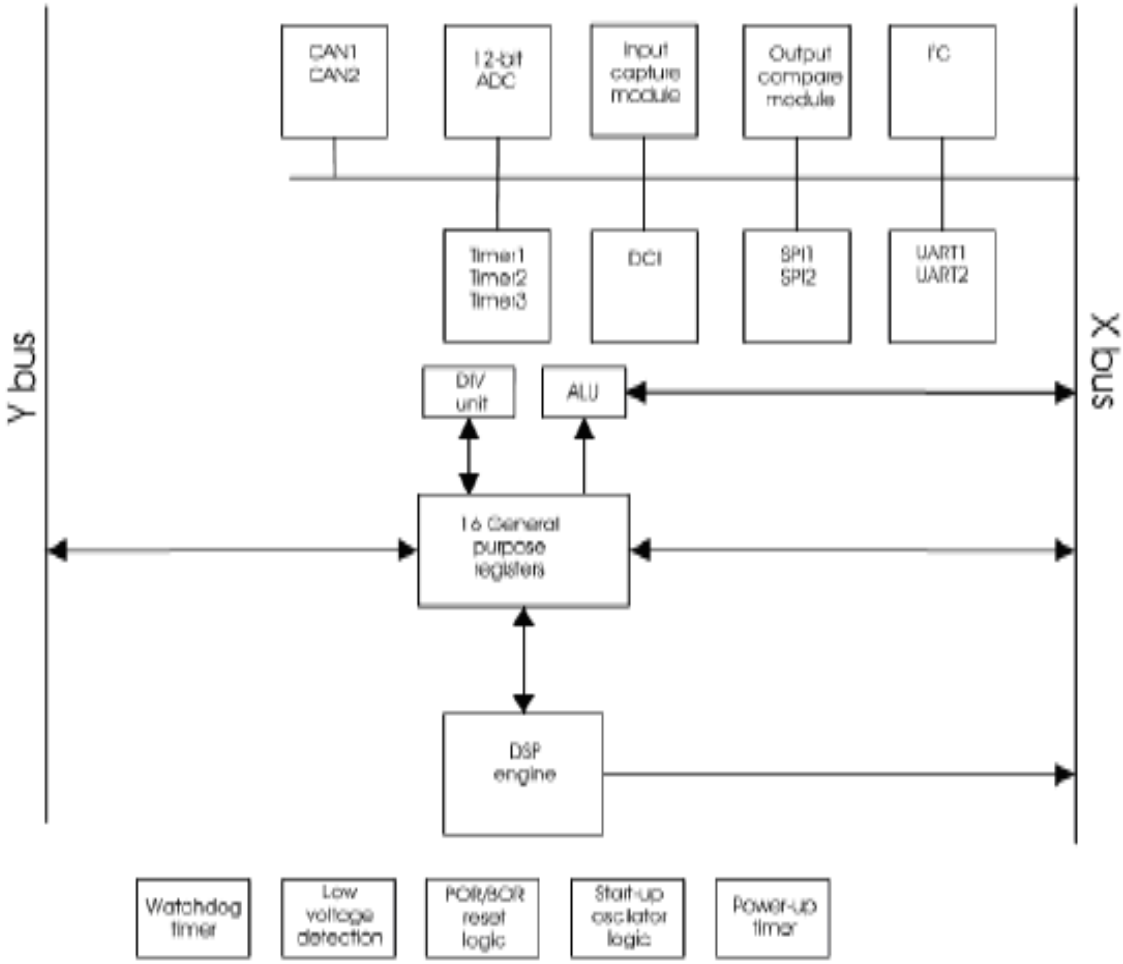



Schéma des microcontrôleurs de la famille dsPIC30F

## Abréviations

| | |
|---|---|
| AD | Analogue-to-Digital |
| DSP | Digital Signal Processing |
| UART | Universal Asynchronous Receiver Transmitter |
| I²C | Inter Integrated Circuit |
| PLL | Phase Locked Loop |
| PWM | Pulse Width Modulation |
| MCU | Microprocessor Unit |
| IVT | Interrupt Vector Table |
| AIVT | Alternate Interrupt Vector Table |
| SFR | Special Function Registers |
| SP | Stack Pointer |
| PC | Program Counter |
| IRQ | Interrupt Request |
| ACC | Accumulator |
| CAN | Control Area Network |
| DCI | Data Converter Interface |
| RTC | Real-Time Clock |
| GP | General Purpose |
| PSV | Program Space Visibility |
| RAM | Random Access Memory |
| ROM | Read Only Memory |
| AGU | Address Generator Unit |
| EA | Effective Address |
| ALU | Arithmetic Logic Unit |
| SPI | Serial Peripheral Interface |
| FIFO | First In – First Out |
| LIFO | Last In – First Out |
| PR | Period |
| SAR | Successive Approximation |
| SH | Sample and Hold |
| VREF | Voltage Reference |
| BUF | Buffer |
| MUX | Multiplexer |
| FFT | Fast Fourier Transform |
| INC | Increment |
| DEC | Decrement |
| MS | Master-Slave |
| SAT | Saturation |
| IPMI | Intelligent Management Interface |
| DTMF | Dual Tone Multiple Frequency |
| FIR | Finite Impulse Response |
| IIR | Infinite Impulse Response |

| | |
|---|---|
| FRC | Fast RC Oscillator (internal) |
| FSCM | Fail-Safe Clock Monitor |
| EXTR | External RESET |
| BOR | Brown-out Reset |
| LVD | Low-Voltage Detect |

**Les bits de configuration**

Avant de démarrer La programmation d'un microcontrôleur, Il est nécessaire d'en connaître explicitement l'horloge définissant la vitesse d'exécution des instructions, la source de l'horloge , les protections contre les instabilités de l'alimentation, la protection de l'exécution irrégulière des parties de programmes ( via le timer watchdog qui sera décrit plus loin) etc..

La configuration d'un microcontrôleur peut être réalisée par 4 registres :

| Register | Name | Address |
|---|---|---|
| FOSC | Oscillator Configuration Register | 0xF80000 |
| FWDT | Watchdog Timer Configuration Register | 0xF80002 |
| FBORPOR | BOR and POR (Voltage Protection) Configuration Register | 0xF80004 |
| FGS | General Code Segment Configuration Register | 0xF8000A |

Ces registres sont dans la moitié partie haute de la mémoire programme et ont une largeur de 24 bits.

Il est important de noter que la configuration des registres est valide seulement durant la configuration du microcontrôleur résultant d'un reset du microcontrôleur.

**Configuration du registre Oscillateur (FOSC)**

Il est utilisé pour ajuster l'horloge du microcontrôleur. L'horloge influence directement la vitesse d'exécution des instructions. Il est

donc essentiel que l'horloge soit sélectionnée correctement.. Il est tout aussi essentiel de sélectionner l'oscillateur source.
La source peut être :
L'oscillateur interne RC (FRC- oscillateur interne rapide RC, LPRC- oscillateur basse puissance interne RC) ou oscillateur externe (ERC- oscillateur externe RC, XT-quartz externe).
L'oscillateur RC interne est insuffisant dans la majorité des applications du microcontrôleur. Il varie en fonction de la tension d'alimentation et de la température ce qui le rend inacceptable. il est possible d'utiliser un quartz résonateur externe comme source . Sa fréquence peut varier entre 200 Khz et 25 Mhz. La troisième option est l'utilisation d'une horloge externe . Si l'on utilise une source externe ou un quartz résonateur, il est possible d'utiliser un PLL interne et d'incrémenter l'horloge interne d'un facteur 4,8, ou 16.

L'horloge interne, de période TCY, contrôle l'exécution des instructions, elle est obtenue quand l'horloge ( PLL ou directe) est divisée par 4. Prenons un quartz résonateur de 10 Mhz et un PLL 4x. L'horloge interne sera 4x10Mhz =40 Mhz. Cette valeur est divisée par 4 pour obtenir 10 Mhz (période de 10 ns). Ce qui implique qu'une instruction sera exécutée en 100 ns. Dans la majorité des cas il est très important de connaître le temps d'exécution des différentes parties du programme donc de connaître la période de l'horloge interne.
La configuration du registre oscillateur permet de sélectionner la source de l'horloge, contrôle l'activation et le mode du PLL, valide la permission de commuter une autre source d'horloge. Cette capacité permet d'ouvrir un large spectre d'applications pour la famille du DSPIC 30F.
Il est possible de permettre une stabilité supplémentaire de l'horloge en commutant plus rapidement la source.
La structure du FOSC est la suivante :

| name | ADR | 23-16 | 15-14 | 13 | 12 | 11 | 10 | 9-8 | 7 | 6 | 5 | 4 | 3-0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FOSC | 0xF80000 | - | FCKSM<1:0> | - | - | - | - | FOS<1:0> | - | - | - | - | FPR<3:0> |

```
FCKSM <1:0> - Clock switching selection
              Clock monitoring selection
       1x - Switching between different sources of the clock is disabled.
            Clock monitoring is disabled.
       01 - Switching between different sources of the clock is enabled.
            Clock monitoring is disabled.
       00 - Switching between different sources of the clock is enabled.
            Clock monitoring is enabled.

FOS <1:0> - Definition of the clock source.
     11 - Primary oscillator selected by FPR<3:0>
     10 -Internal low power RC oscillator
     01 -Internal fast RC oscillator
     00 - Low power 32kHz internal oscillator (timer 1)

FPR <3:0> - Selection of the mode of generating the internal clock
     1111 - EC PLL 16x - External clock mode with 16x PLL enabled.
            External clock at  OSC1, OSC2 pin is I/O
     1110 - EC PLL 8x - External clock mode with 8x PLL enabled.
            External clock at  OSC1, OSC2 pin is I/O
     1101 - EC PLL 4x - External clock mode with 4x PLL enabled.
            External clock at  OSC1, OSC2 pin is I/O
     1100 - ECIO - External clock mode without PLL.

           External clock at OSC1, OSC2 pin I/O
     1011 - EC - External clock mode. OSC2 pin is system clock output
(Fosc/4)
     1010 - Reserved
     1001 - ERC - External RC oscillator mode. OSC2 pin is system clock
output (Fosc/4)
     1000 - ERCIO - External RC oscillator mode. OSC2 pin is I/O
     0111 - XT PLL 16x - XT crystal oscillator mode with 16xPLL enabled. 4
- 10MHz crystal
     0110 - XT PLL 8x - XT crystal oscillator mode with 8xPLL enabled. 4 -
10MHz crystal
     0101 - XT PLL 4x - XT crystal oscillator mode with 46xPLL enabled. 4 -
10MHz crystal
     0100 - XT -  XT crystal oscillator mode (4 - 10MHz crystal)
     001x - HS - HS crystal oscillator mode (10 - 25MHz crystal)
     000x - XTL - XTL crystal oscillator mode (200kHz - 4MHz crystal)
```

## Configuration du registre du timer Watchdog (FWDT)

Elle est utilisée pour commuter le timer watchdog ‘’ON-Off’’ et pour l’ajustement ‘’ preseting’’. La valeur preset est un nombre divisant l’horloge interne RC obtenue.  Dans ce cas, l’horloge du timer watchdog.
Le timer  watchdog est indépendant de l’horloge interne qui commande l’exécution des instructions. De cette façon il est possible d’obtenir une opération du  timer même si l’horloge interne est en panne.

La fréquence de l'oscillateur RC du timer watchdog est de 512 Khz. Après division par 4 on obtient 128 Khz qui est l'horloge de base du timer watchdog.
L'horloge de base 128Khz est divisée par deux valeurs programmables. la première est le prescaler A et le second le prescaler B. Après division de l'horloge timer watchdog est obtenue. Un registre 8 bits est incrémenté avec cette horloge. Quand la valeur du registre atteint un maximum de 255, le timer watchdog remet à zéro le microcontrôleur ou commute celui-ci de l'état inactif à actif.
Si le microcontrôleur est dans un état inactif (SLEEP), le watchdog commute à l'état actif et l'exécution du programme courant continue.Si le microontrôleur est dans un mode actif, c'est-à-dire que le programme courant tourne le watchdog définira le temps de remise à zéro du microcontrôleur.
L'intervalle de temps entre le moment où le watchdog envoie un reset et un signal de réveil est calculé comme suit :
La période de l'horloge de base est de 7,8125 us. Cette valeur est multipliée par les prescalers A et B. Ceci donne la période du timer watchdog. Cette période multipliée par 255 donne la valeur maximale d'incrémentation du registre. Cette valeur est le temps que le timer watchdog devra attendre avant de générer un signal ''reset/wake – up'' au microcontrôleur. Le temps d'attente en millisecondes, dépend des valeurs des prescalers A et B.

| Prescaler B value | Prescaler A value | | | |
|---|---|---|---|---|
| | 1 | 8 | 64 | 512 |
| 1 | 2 | 16 | 128 | 1024 |
| 2 | 4 | 32 | 256 | 2048 |
| 3 | 6 | 48 | 384 | 3072 |
| 4 | 8 | 64 | 412 | 4096 |
| 5 | 10 | 80 | 640 | 5120 |
| 6 | 12 | 96 | 768 | 6144 |
| 7 | 14 | 112 | 896 | 7168 |
| 8 | 16 | 128 | 1024 | 8192 |

| | | | | |
|---|---|---|---|---|
| 9 | 18 | 144 | 1152 | 9216 |
| 10 | 20 | 160 | 1280 | 10240 |
| 11 | 22 | 176 | 1408 | 11264 |
| 12 | 24 | 192 | 1536 | 12288 |
| 13 | 26 | 208 | 1664 | 13312 |
| 14 | 28 | 224 | 1792 | 14336 |
| 15 | 30 | 240 | 1920 | 15360 |
| 16 | 32 | 256 | 2048 | 16384 |

Exemple

Considérons l'exécution d'une partie de programme dépendant d'un signal externe  et ne dépassant pas 100mS.

Dans ce cas le timer watchdog peut être activé pour une période de 100 mS. Si le signal dure plus de 100 mS , le microcontrôleur est remis à zéro et l'exécution du programme redémarre au début. Le timer watchdog devrait être configuré pour performer selon une fonction donnée. Bien qu'il soit possible  d'attendre 100 mS, on utilisera une période de 112 mS comme temps minimum. Ceci est réalisé avec les valeurs du prescaler A=8 et B=7.

Si le programme fonctionne correctement, il est nécessaire de remettre à zéro la valeur du registre watchdog par l'instruction en assembleur CLRWDT. Le timer watchgog est configuré avec le registre FWDT.

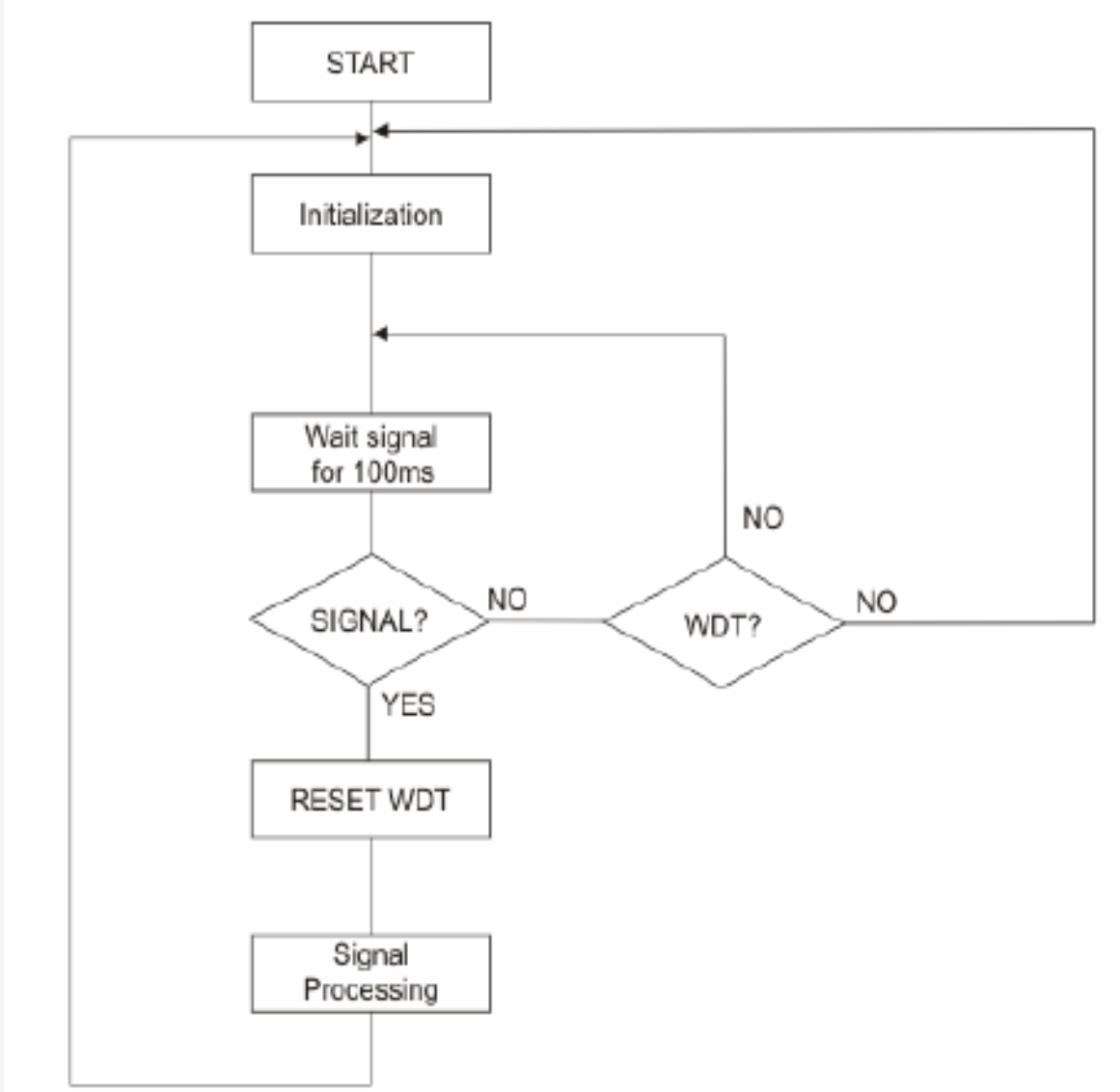
| name | ADR | 23-16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5-4 | 3-0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FWDT | 0xF0002 | - | FWDTEN | - | - | - | - | - | - | - | - | - | FWPSA<1:0> | FPR<3:0> |

**Table 2-3 The configuration register od the watchdog timer FWDT**

**FWDTEN** - Watchdog enable configuration bit
1 - Watchdog enabled; LPRC osillator can not be disabled
0 - Watchdog disabled

**FWPSA** <1:0> - Selected value of prescaler A
11 = 1:512
10 = 1:64
01 = 1:8
00 = 1:1

**FWPSB** <1:0> - Selected value of prescaler B
1111 = 1:16
1110 = 1:15
....
....
....
0001 = 1:2
0000 = 1:1




**Configuration du registre de protection voltage**

Il sert à définir la valeur minimale de la tension d'alimentation, la commutation ''on/off'' du circuit de remise à zéro du microcontrôleur si la tension vient à descendre au-dessous de la valeur spécifiée et pour l'ajustement de la sortie PWM.

La bonne exécution du programme requière une stabilité de l'alimentation. La famille du microcontrôleur dsPIC30F a la capacité de définir une tension d'alimentation minimale permettant un fonctionnement correct du circuit. Si la tension descend au-dessous de cette limite, le circuit interne remettra à zéro le microcontrôleur et attendra que la tension revienne au-dessus de la limite en utilisant un délai spécifié.

L'ajustement de la sortie PWM se fera un peu plus loin. Il est possible de sélectionner l'état des broches PWM (haute impédance ou sortie) ainsi que la polarité du signal actif. Cette polarité peut être sélectionnée de façon indépendante.

| name | ADR | 23-16 | 15 | 14-11 | 10 | 9 |
|---|---|---|---|---|---|---|
| FBORPOR | 0x80004 | - | MCLREN | - | PWMPIN | HPOL |

**Table 2-4. The voltage protection configuration register FBORPOR**

| 8 | 7 | 6 | 5-4 | 3-2 | 1-0 |
|---|---|---|---|---|---|
| LPOL | BOREN | - | BORV<1:0> | - | FPWRT<1:0> |

```
MCLREN - Pin function enable bit
         below the specified value
         1 - Pin function enabled (default)
         0 - Pin is disabled

PWMPIN - The state of PWM at device reset
         1 - PWM pin is in the state of high impedance at device reset
         0 - PWM pin is configured as output at device reset

HPOL - High side polarity bit
       1 - High side output pin has active-high output polarity
       0 - High side output pin has active-low output polarity

LPOL - Low side polarity bit
       1 - Low side output pin has active-high output polarity
       0 - Low side output pin has active-low output polarity

BOEN - Enable bit
       1 - PBOR enabled
       0 - PBOR disabled

BORV <1:0> - Minimum supply voltage select bits
      11 - 2.0V
      10 - 2.7V
      01 - 4.2V
      00 - 4.5V

FPWRT <1:0> - Power-on reset timer value selection bits
       11 - 64ms
       10 - 16ms
       01 - 4ms
       00 - 0ms
```

## Configuration du registre de protection de la mémoire programme (FGS)

Elle est utilisée pour le code de protection/écriture de protection ou pour protéger l'espace de la mémoire programme. Ceci inclus tout le

programme mémoire utilisateur à l'exception de la table de vecteur d'interruption.

Si la mémoire programme est protégée par un code, la mémoire programme du circuit ne peut être lue par un circuit utilisant un programme sériel ou un programmeur.
Le registre  utilise seulement 2  bits de configuration. Ces 2 bits peuvent être programmés comme un groupe et il est impossible d'en modifier un seul.
Si le code de protection ou l'écriture de protection sont validés, la plupart des circuits  peuvent  modifier un seul bit.

| name | ADR | 23-16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FGS | 0x8000A | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

```
GCP - Program memory code protect enable
      1 - Program memory is not code protected
      0 - Program memory is code protected

GWRP - Program memory write protect enable
       1 - Program memory is not write protected
       0 - Program memory is write protected
```

**Les interruptions et les traps**

**Les interruptions**
Les causes des interruptions sont multiples. La famille du dsPIC30F permet 41 sources d'interruptions et 4 types de traps.
Pour chaque interruption ou trap le microcontrôleur a clairement défini plusieurs  séquences de programmes spécifiés par la table de vecteur d'interruption (IVT).

L'IVT contient  l'adresse initiale des routines d'interruption pour chaque interruption ou trap.
L'IVT est située dans la mémoire programme entre 0x000004 et ox00007E.

En plus de la table de vecteur d'interruption, la famille  du dsPIC33F contient une table  alternée (AIVT). L'AIVT est entre 0x000084 et 0x0000FE

- **IFS0**<15:0>, **IFS1**<15:0>, and **IFS2**<15:0> are the registers containing all the interrupt request flags. Each source of interrupt has a status bit set by the respective peripherals or external signals. These flags are cleared by the user software.
- **IEC0**<15:0>, **IEC1**<15:0>, and **IEC2**<15:0> are the registers containing all the interrupt enable control bits. These control bits are used to individually enable interrupts from the peripherals or external signals.
- **IPC0**<15:0>, **IPC1**<15:0>, ... **IPC10**<7:0> are the registers used to set the interrupt priority level for each of the 41 sources of interrupt.
- **IPL**<3:0> are the bits containing the current CPU priority level. **IPL**<3> bit is located in the register **CORCON** and the remaining three bits **IPL**<2:0> are in the STATUS register (SR) of the microcontroller.
- **INTCON1**<15:0> and **INTCON2**<15:0> are the registers containing global interrupt control functions. **INTCON1** contains the control and status bits for the processor trap sources; **INTCON2** controls the external interrupt requests behaviour and the use of the alternate vector table by setting the **ALTIVT** bit (**INTCON2**<15:0>).

La nidification d'interruption est activée par le bit NSTDIS du registre de contrôle INTCON1. L'imbrication des routines d'interruption peut être éventuellement désactivée en activant ce bit. Cela peut être important s'il est nécessaire pour réaliser une partie du programme sans une interruption qui pourrait changer l'état qui est la base de cette partie du programme ou si les changements de routine d'interruption de certaines variables  sont d'importance pour la poursuite de l'exécution du programme.
Le niveau de priorité de chaque interruption est affecté en réglant le bit de priorité d'interruption IP <2: 0> pour chaque source d'interruption. Les bits IP <2: 0> sont les moins significatifs 3 bits de chaque quartet (4 bits)

dans le registre IPCX. Bit. 3 de chaque quartet iz toujours à zéro. L'utilisateur peut affecter 7 niveaux de priorité, de 1à 7. Le niveau 7 est le niveau le plus haut et 1 le niveau le plus bas pour des interruptions non masquables.
C’est à dire, pour les interruptions qui pourraient être activées par les bits de contrôle des IECx.

Attention!
Si une interruption est affectée un niveau de priorité égal à 0, c’est comme si aucune interruption était autorisée par les bits situés dans IECx.
La priorité de l'ordre naturel est spécifiée par la position d'une interruption dans la table de vecteur (IVT). Elle est utilisée seulement pour résoudre les conflits entre les interruptions d’un utilisateur ayant affecté le même niveau de priorité . Ensuite, l'interruption du niveau naturel de priorité plus élevé est exécutée en premier. A titre d'exemple, le tableau montre pour le microcontrôleur dsPIC30F4013 l'IVT avec toutes les sources d'interruptions, le numéro d'interruption dans la table de vecteur, et le nombre qui définit la priorité de l'ordre naturel.

| INT Num | Vector Num | IVT Address | AIVT Address | Interrupt Source |
|---|---|---|---|---|
| Highest Natural Order Priority | | | | |
| 0 | 8 | 0x000014 | 0x000094 | INT0 - External Interrupt 0 |
| 1 | 9 | 0x000016 | 0x000096 | IC1 - Input Capture 1 |
| 2 | 10 | 0x000018 | 0x000098 | OC1 - Output Compare 1 |
| 3 | 11 | 0x00001A | 0x00009A | T1 - Timer 1 |
| 4 | 12 | 0x00001C | 0x00009C | IC2 - Input Capture 2 |
| 5 | 13 | 0x00001E | 0x00009E | OC2 - Output Compare 2 |
| 6 | 14 | 0x000020 | 0x0000A0 | T2 - Timer 2 |
| 7 | 15 | 0x000022 | 0x0000A2 | T3 - Timer 3 |
| 8 | 16 | 0x000024 | 0x0000A4 | SPI1 |
| 9 | 17 | 0x000026 | 0x0000A6 | U1RX - UART1 Receiver |
| 10 | 18 | 0x000028 | 0x0000A8 | U1TX - UART1 Transmitter |
| 11 | 19 | 0x00002A | 0x0000AA | ADC - ADC Convert Done |
| 12 | 20 | 0x00002C | 0x0000AC | NVM - NVM Write Complete |

| | | | | |
|---|---|---|---|---|
| 13 | 21 | 0x00002E | 0x0000AE | SI2C - I2C Slave Interrupt |
| 14 | 22 | 0x000030 | 0x0000B0 | MI2C - I2C Master Interrupt |
| 15 | 23 | 0x000032 | 0x0000B2 | Input Change Interrupt |
| 16 | 24 | 0x000034 | 0x0000B4 | INT1 - External Interrupt 1 |
| 17 | 25 | 0x000036 | 0x0000B6 | IC7 - Input Capture 7 |
| 18 | 26 | 0x000038 | 0x0000B8 | IC8 - Input Capture 8 |
| 19 | 27 | 0x00003A | 0x0000BA | OC3 - Output Compare 3 |
| 20 | 28 | 0x00003C | 0x0000BC | OC3 - Output Compare 4 |
| 21 | 29 | 0x00003E | 0x0000BE | T4 - Timer 4 |
| 22 | 30 | 0x000040 | 0x0000C0 | T5 - Timer 5 |
| 23 | 31 | 0x000042 | 0x0000C2 | INT2 - External Interrupt 2 |
| 24 | 32 | 0x000044 | 0x0000C4 | U2RX - UART2 Receiver |
| 25 | 33 | 0x000046 | 0x0000C6 | U2TX - UART2 Transmitter |
| 26 | 34 | 0x000048 | 0x0000C8 | Reserved |
| 27 | 35 | 0x00004A | 0x0000CA | C1 - Combined IRQ for CAN1 |
| 28-40 | 36-48 | 0x00004C – 0x000064 | 0x0000CC – 0x0000E4 | Reserved |
| 41 | 49 | 0x000066 | 0x0000E6 | DCI - CODEC Transfer Done |
| 42 | 50 | 0x000068 | 0x0000E8 | LVD - Low Voltage Detect |
| 43-53 | 51-61 | 0x00006A – 0x00007E | 0x0000EA – 0x0000FE | Reserved |
| Lowest Natural Order Priority | | | | |

Exemple:
L'exemple montre comment le dsPIC réagit à un front montant sur la broche RF6  (INT0). pour chaque front montant la valeur au port D est incrémenté de 1.
REMARQUE: Les broches qui peuvent être utilisées pour les interruptions externes dépendent du modèle. Cet exemple est fait pour dsPIC30F6014A

```
Void IntDetection void () {// org 0x0014 Interruption sur INT0
LATD ++;
IFS0.F0 = 0; // drapeau interruption effacé
}

void main () {
TRISD = 0; // PORTD en sortie
```

```
TRISF = 0xFFFF; //  PORTF en entrée
IFS0 = 0; // drapeau Interruption effacé
IEC0 = 1; // Interruption est réglée sur un front montant sur INT0
(RF6)
while (1) asm nd;
}
```

Résumons l'algorithme comment le dsPIC traite les interruptions dans cet exemple. On utilise deux ports, comme le PORTD en sortie pour afficher le nombre d'événements d'interruption et le  PORTF en entrée; cela signifie qu'une interruption se produit lorsque sur INT0 (RF6)  une logique 0 passe à 1.

Dans le registre IEC0, le bit le moins significatif (IEC0.0) est réglé pour permettre la réaction d'interruption INT0. La signification des autres bits est présentée dans le tableau . Quand une interruption se produit, la fonction IntDetection  est appelée.

Comment le dsPIC «sait»  appeler exactement cette fonction? Par l' instruction org dans la table de vecteur d'interruption (voir tableau) à l'emplacement de mémoire  0x000014 où est écrit la fonction IntDetection.

Qu'est-ce que le dsPIC fait quand une interruption se produit, c'est à dire quand à un niveau logique 1 sur RF6 apparaît après un 0 logique? Dans un premier temps, il écrit un niveau  logique1  dans le bit le moins significatif du registre de IFS0. Puis, il teste si le bit INT0 interruption est activé (bit le moins significatif de IEC0). Si oui, il lit dans la table de vecteur d'interruption quelle partie du programme doit être exécuté. Le compilateur mikroC sera,à la position 0x000014, écrira le code où la fonction IntDetection commence et le dsPIC exécutera cette fonction pour revenir au programme principal.

Deux opérations sont effectuées dans la fonction d'interruption. Dans un premier temps, l'indicateur d'interruption est effacé (DsPIC ne le fait pas automatiquement, mais quitte ceci pour le logiciel de l'utilisateur). Ensuite, la valeur du port D est incrémentée de 1 avec LATD ++.

Qu'arriverait-il si l'indicateur d'interruption dans le registre de IFS n'a pas été effacé? Si cette ligne est omise, le dsPIC attendra jusqu'à la première interruption, pour  appeler et exécuter la fonction IntDetection.

Quelle est la différence? Le registre IFSO indiquera toujours  qu'une interruption s'est produite. Dès que le microcontrôleur retourne au programme principal et exécute une instruction, il devrait comprendre

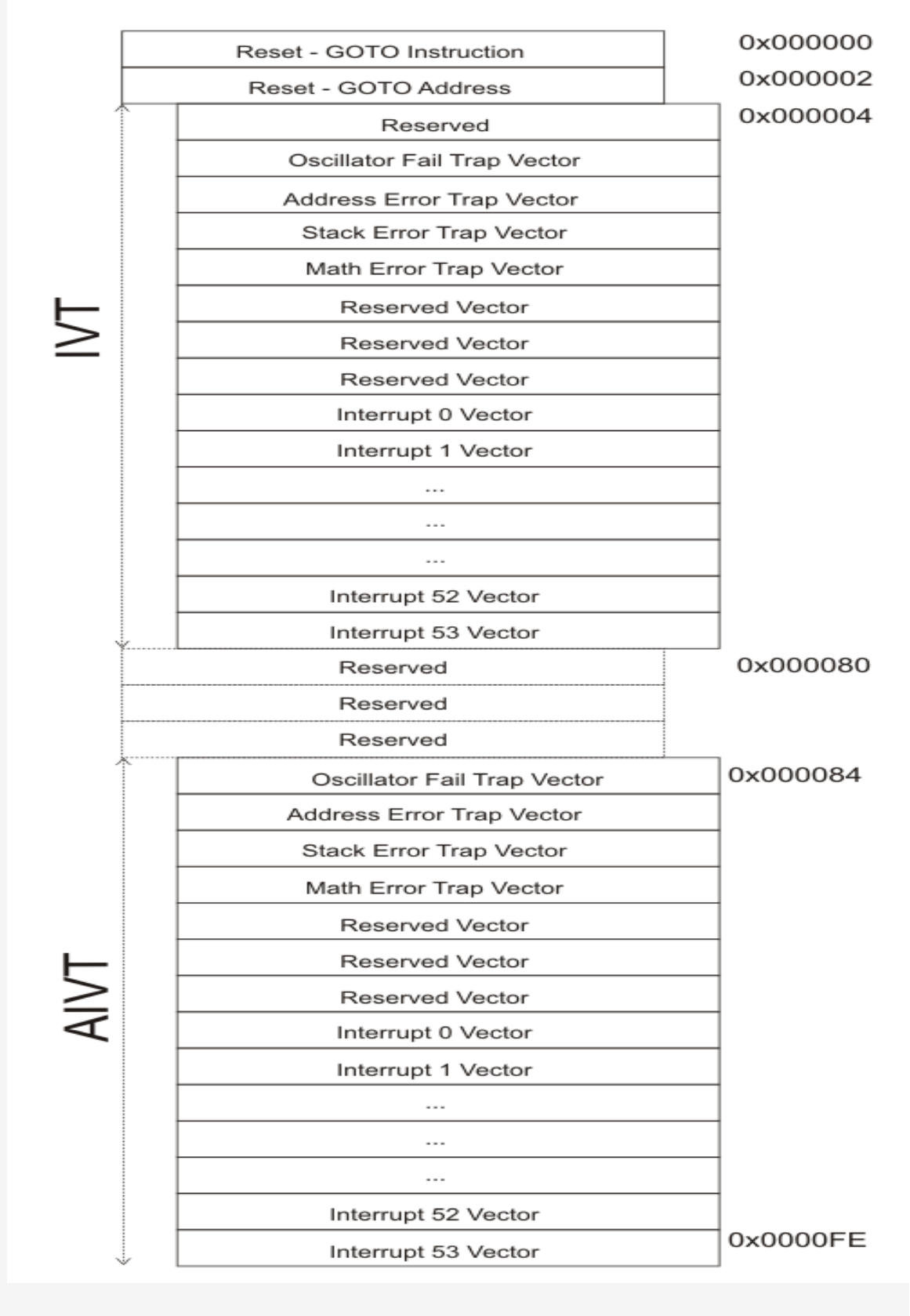
qu'une interruption s'est produite de nouveau et la fonction IntDetection sera appelée à  nouveau. Cela signifie que la fonction IntDetection serait exécutée après chaque instruction du programme principal. Le lecteur est invité à effacer la ligne IFS0.F0 : = 0; et voir ce qui se passe.
Quel est le but de while (1) asm nop ;? Lorsque le dsPIC arrive à la fin d'un programme, il commence dès le début comme en cas de réinitialisation. Cependant, à la fin du programme, le compilateur insère une boucle sans fin pour éviter ce qui se passe. Cela a été inséré ici pour tenir le lecteur au courant de l'existence de cette boucle et que reset dsPIC n'est pas possible.

**Les traps**

Les traps peuvent être considérés comme des interruptions qui ne pourraient pas être masquées en réglant les bits de contrôle d'interruption dans le registre de validation interapt IECx. Les traps sont destinés à fournir à l'utilisateur un moyen pour corriger une opération erronée au cours de débogage et de développement d'applications.
NOTE: Si l'utilisateur n'a pas l'intention d'utiliser des mesures correctives dans le cas d'une erreur d'état trap, la table des vecteurs d'interruption est chargée avec l'adresse d'une routine d'interruption par défaut contenant uniquement l'instruction RESET. Si la table de vecteur d'interruption est chargée avec une erreur adresse ou une adresse qui ne veut pas dire n'importe quelle routine, l'adresse trap erronée sera générée et peut conduire à un RESET.
Les conditions de déroutement ne peuvent être détectées lorsque le trap se produit. Le calcul d'itinéraire généré par le trap doit être en mesure de supprimer l'erreur qui mène à elle. Chaque source de trap a une priorité fixe, niveau allant de 8 à niveau 15. Cela signifie que le bit IPL 3 est toujours réglé au cours du traitement de tout trap.
Les sources de traps sont classées dans des groupes avec la culture de priorité d'interruption.
A. Des erreurs arithmétiques sont générées si pendant les opérations arithmétiques les erreurs suivantes se produire:
1. Diviser par zéro est tenté; l'opération sera arrêtée et le trap diviser par zéro  sera généré (interruption niveau de priorité 8),
2. Si l'erreur trap arithmétique  est activée, et si au cours d'une opération mathématique le débordement de l'accumulateur A ou B se produit (report à partir de 31 bits) et les bits protection dans les

accumulateurs (mode de saturation 31 bits) ne sont pas activés (priorité d'interruption niveau 9),
3. Si l'erreur  trap arithmétique est activée, et si au cours d'une opération mathématique un débordement catastrophique de l'accumulateur A ou B se produit (de fin de campagne du bit 39); la saturation de l'accumulateur est empêchée (interruption niveau de priorité 10),
4. Si durant opération de décalage  la valeur de décalage dépasse 16 bits lors du traitement des mots, ou 8 bits lors  des octets de traitement (interruption de niveau de priorité 11).
B. les traps d'erreur d'adresse sont générés si les situations d'exploitation suivantes se produisent (interruption niveau de priorité 13):
1. Un mot de données mal aligné (2 octets) recherche est tenté (accès par mot avec une adresse impaire),
2. Les données d'une extraction d'un espace d'adressage de données  inexpliqué du microcontrôleur sont une tentative,
3. Une tentative par le programme pour répondre à un emplacement mémoire de programme inappliqué du microcontrôleur,
4. L'accès à une instruction au sein de l'espace vectoriel est tenté
5. Une branche ou instruction de saut spécifie une adresse de l'espace d'adresse inappliquée du microcontrôleur,
6. Sur modification du compteur de programme (PC) , une location de mémoire programme non existante dans le processeur.
C. « Stack » erreur est générée si les événements suivants se produisent (interruption niveau de priorité 13):
1. Le pointeur de pile (SP) dépasse la valeur de la SPLIM (Stack Pointer LIMit - défini par l'utilisateur), c'est à dire la pile est dépassée
2. La valeur de pointeur de pile est inférieure à 0x0800, c'est à dire la pile de dépassement de la partie inférieure.
D. Échec Oscillateur interruption est généré si l'oscillateur externe échoue; la fiabilité de fonctionnement du microcontrôleur est alors basée sur l'oscillateur RC interne (niveau d'interruption de priorité 14).
Les traps peuvent être classés en deux groupes: logiciels et matériels . Le groupe logiciel contient des traps de priorités 8-11 qui se posent à la suite d'opérations arithmétiques. Les routines d'interruption causées par les traps logiciels sont emboîtables. Le groupe matériel contient des traps de priorités de 12 à 15. Les routines d'interruption causées par les traps matériels ne sont pas emboîtables. Le trap de

priorité la plus élevé est généré lorsque deux traps matériels sont en conflit.
L'organisation de la table des vecteurs d'interruption et autre table de vecteurs dans le programme de la mémoire de la famille des microcontrôleurs dsPIC30F est représenté sur la Fig.




En cas de manipulation d'interruption il est nécessaire de connaître la

séquence d'interruption. La séquence d'interruption commence par écrire une demande d'interruption, de périphériques ou des signaux externes, Dans une demande d'interruption le bit du registre IFSx est au début de chaque cycle d'instruction. Une demande interruption (IRQ) est identifiée par le positionnement du bit correspondant dans le registre IFSx. L'interruption est générée uniquement s'il est activé, c'est à dire si le bit correspondant dans le registre IECx est défini. Au cours de l'exécution du cycle d'instruction courant des niveaux de priorité de toutes les demandes pendantes, les interruptions sont évaluées. Si le niveau de priorité d'une interruption en attente est plus élevé que la priorité du niveau traité par le microcontrôleur (défini par les bits IPL <3: 0>), une interruption est générée.
Sur une interruption générée, le microcontrôleur permet d'empiler la valeur du PC courant ( compteur ordinal)
et l'octet de poids faible du registre d'état du processeur (SRL). L' octet SRL contient le niveau de priorité de l'interruption en cours de traitement . Après que le microcontrôleur ait défini le nouveau niveau de priorité d'interruption dans le registre d'état, égal au niveau interrompu. Des interruptions d'un niveau inférieur sont empêchées en procédant de cette façon. Le microcontrôleur ne prend pas automatiquement en charge le reste du contexte du processeur. Cette tâche est entreprise par le compilateur qui sauve le PC, SRL, et les valeurs des autres registres importants.
L'instruction RETFIE (retour d'interruption) désigne le retour d'interruption et dépile l'adresse de retour du PC et l'octet de poids faible du registre d'état du processeur pour retourner le microcontrôleur au niveau de l'état et la priorité existant avant l'interruption.

Attention!
L'utilisateur peut abaisser le niveau de priorité de l'interruption en cours de traitement en écrivant une nouvelle valeur dans le registre d'état (SR) lors du traitement de l'interruption, mais la routine d'interruption doit effacer le bit de demande d'interruption dans le registre IFSx afin d'éviter une récidive de génération d'interruption. La génération d'interruptions récurrentes pourrait facilement conduire à Stack Overflow et à la génération d'une interruption d'erreur de pile et la remise à zéro du microcontrôleur.
REMARQUE: Malgré le traitement d'interruption le bit IPL 3 est toujours zéro. Il est positionné lorsque le processeur traite des traps.

Pour la manipulation d'interruptions dans la pratique, il est nécessaire de savoir que la famille de microcontrôleurs dsPIC30F prend en charge jusqu'à cinq sources d'interruptions externes, INT0 - INT4. Ces entrées sont sensibles à un front, c'est à dire qu'elles exigent une transition d'un niveau logique bas vers haut ou vice versa, afin de générer une interruption. Le registre contient des bits de INTCON2 INT0EP - INT4EP définissant la polarité du front sensible, à savoir front montant ou descendant du signal.
Très souvent, les interruptions sont utilisées pour réveiller le microcontrôleur en mode SLEEP ou en mode IDLE si le microcontrôleur est dans ces états quand une interruption est générée. Les états SLEEP ou IDLE qui sont utilisés lors de la minimisation de la consommation d'énergie ou de la minimisation du bruit au cours de la mesure du convertisseur analogique - numérique sont obligatoires. Le microcontrôleur se réveillera et lancera l'exécution de la routine d'interruption seulement si la demande d'interruption est activée dans le registre IECx.
L'exemple suivant montre comment un trap fonctionne dans la pratique.

Exemple:
Un programme contient une erreur. Il ya une chance de division par zéro. Étant donné que ce n'est pas correcte, il est possible que cette erreur réinitialise le dsPIC. Cet exemple utilise le code de l'exemple d'interruptions. Sur chaque front montant du bit RF6 la valeur sur le port B est décrémentée de 1 et se rapproche ainsi de zéro, ce qui est indésirable à cause du risque de diviser par la valeur du port B dans le programme principal.
Le programme est le suivant:

```
int a;
void IntDet () {// org 0x0014 vecteur INT0
LATB--; // PortB est décrémenté
IFS0.F0 = 0; // drapeau interruption  effacé
}
void TrapTrap () {org 0x000C
INTCON1.F4 = 0;
/ * le problème est résolu par la mise
du port B à une valeur différente de zéro * /
LATB = 3;
LATD ++;
```

```
}
void main () {
TRISB = 0;
TRISD = 0;
TRISF = 0xFFFF;
LATB = 3;
LATD = 0;
IFS0 = 0; // drapeau interruption effacé
INTCON1 = 0; // drapeau trap effacé
IEC0 = 1; // interruption sur front montant INT0 (RF6) activée
while (1) {
a = 256 / LATB; // si LATB = 0 erreur est survenue et Trap est
appelée
}
}
```

Une description des registres d'interruption du microcontrôleur dsPIC30F4013 est présentée

REMARQUE: lecture de bits sans fonction affectée donne '0'.

| Name | ADR | 15 | 14-11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|
| INTCON1 | 0X0080 | NSTDIS | - | OVATEN | OVBTEN | COVTE |

**Table 3-2. Interrupt control register - INTCON1**

| 7-5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|
| - | MATHERR | ADDRER | STKERR | OSCFAIL | - | 0x0000 |

**Table 3-2. continued**

**NSTDIS** – Interrupt nesting disable bit
**OVATEN** – Accumulator A overflow trap enable bit
**OVBTEN** – Accumulator B overflow trap enable bit
**COVTE** – Catastrophic overflow trap enable bit
**MATHERR** – Arithmetic error trap status bit
**ADDRERR** – Address error trap status bit
**STKERR** – Stack error trap status bit
**OSCFAIL** – Oscillator failure trap status bit

| name | ADR | 15 | 14 | 13 | 12 | 11 | 10 | 9 |
|---|---|---|---|---|---|---|---|---|
| INTCON2 | 0x0082 | ALTIVT | - | - | - | - | - | - |

**Table 3-3. Interrupt control register 2 - INTCON2**

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | INT2EP | INT1EP | INT0EP | 0x0000 |

**Table 3-3. continued**

```
ALTIVIT – Enable alternate interrupt vector table bit
INT0EP, INT1EP, INT2EP – External interrupt edge detect polarity bits
```

| name | ADR | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| IFS0 | 0x0084 | CNIF | MI2CIF | SI2CIF | NVMIF | ADIF | U1TXIF | U1RXIF | SPI1IF |

**Table 3-4. Interrupt flag status register - IFS0**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|
| T3IF | T2IF | OC2IF | IC2IF | T1F | OC1IF | IC1IF | INT0IF | 0x0000 |

**Table 3-4. continued**

```
CNIF – Input change notification flag status bit
MI2CIF – I2C module (master mode) transmitter interrupt flag status bit
SI2CIF – I2C module (slave mode) receiver interrupt flag status bit
NVMIF – Non-volatile memory write complete interrupt flag status bit
ADIF – A/D conversion complete interrupt flag status bit
U1TXIF – UART1 transmitter interrupt flag status bit
U1RXIF – UART1 receiver interrupt flag status bit
SPI1IF – SPI1 interrupt flag status bit
T3IF – Timer 3 interrupt flag status bit
T2IF - Timer 2 interrupt flag status bit
OC2IF – Output compare channel 2 interrupt flag status bit
IC2IF – Input capture channel 2 interrupt flag status bit
T1IF – Timer1 interrupt flag status bit
OC1IF – Output compare channel 1 interrupt flag status bit
IC1IF – Input captue channel 1 interrupt flag status bit
INT0IF – External interrupt 0 flag status bit
```

| name | ADDR | 15-12 | 11 | 10 | 9 | 8 | 7 |
|---|---|---|---|---|---|---|---|
| IFS1 | 0x0086 | - | C1IF | - | U2TXIF | U2RXIF | INT2IF |

**Table 3-5. Interrupt flag status register - IFS1**

| 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|
| T5IF | T4IF | OC4IF | OC3IF | IC8IF | IC7IF | INT1IF | 0x0000 |

**Table 3-5. continued**

```
C1IF – CAN1 (combined) interrupt flag status bit
U2TXIF – UART2 transmitter interrupt status bit
U2RXIF – UART2 receiver interrupt flag status bit
INT2IF – External interrupt 2 flag status bit
T5IF – Timer5 interrupt flag status bit
T4IF – Timer4 interrupt flag status bit
OC4IF – Output compare channel 4 interrupt flag status bit
OC3IF- Output compare channel 3 interrupt flag status bit
IC8IF – Input capture channel 8 interrupt flag status bit
IC7IF - Input capture channel 7 interrupt flag status bit
INT1IF – External interrupt 1 flag status bit
```

| name | ADDR | 15 | 14 | 13 | 12 | 11 | 10 | 9 |
|---|---|---|---|---|---|---|---|---|
| IFS2 | 0x0088 | - | - | - | - | - | LVDIF | DCIIF |

**Table 3-6. Interrupt flag status register 2 - IFS2**

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | 0x0000 |

**Table 3-6. continued**

```
LVDIF – Programmable low voltage detect interrupt flag status bit
DCIIF – Data converter interface interrupt flag status bit
```

| name | ADDR | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| IEC0 | 0x008C | CNIE | MI2CIE | SI2CIE | NVMIE | ADIE | U1TXIE | U1RXIE | SPI1IE |

**Table 3-7. Interrupt enable control register 0 - IEC0**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|
| T3IE | T2IE | OC2IE | IC2IE | T1E | OC1IE | IC1IE | INT0IE | 0x0000 |

**Table 3-7. continued**

```
CNIE – Input change notification interrupt enable bit
MI2CIE – I2C module (master mode) transmitter interrupt enable bit
SI2CIE – I2C I2C module (slave mode) receiver interrupt enable bit
NVMIE – Non-volatile memory write complete interrupt enable bit
ADIF – A/D conversion complete interrupt enable bit
U1TXIE – UART 1 transmitter interrupt enable bit
U1RXIE – UART 1 receiver interrupt enable bit
SPI1IE – SPI1 interrupt enable bit
T3IE – Timer3 interrupt enable bit
T2IE – Timer2 interrupt enable bit
OC2IE – Output compare channel 2 interrupt enable bit
IC2IE – Input capture channel 2 interrupt enable bit
T1IE – Timer1 interrupt enable bit
OC1IE – Output compare channel 1 interrupt enable bit
IC1IE – Input capture channel 1 interrupt enable bit
INT0IE – External interrupt 0 enable bit
```

| name | ADDR | 15-12 | 11 | 10 | 9 | 8 | 7 |
|---|---|---|---|---|---|---|---|
| IEC1 | 0x008E | - | C1IE | - | U2TXIE | U2RXIE | INT2IE |

**Table 3-8. Interrupt enable control register 1 - IEC1**

| 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|
| T5IE | T4IE | OC4IE | OC3IE | IC8IE | IC7IE | INT1IE | 0x0000 |

**Table 3-8. continued**

```
C1IE – CAN1 (combined) interrupt enable bit
U2TXIE – UART2 transmitter interrupt enable bit
U2RXIE – UART2 receiver interrupt enable bit
INT2IE – External interrupt 2 enable bit
T5IE – Timer5 interrupt enable bit
```

```
T4IE – Timer4 interrupt enable bit
OC4IE – Output compare channel 4 interrupt enable bit
OC3IE – Output compare channel 3 interrupt enable bit
IC8IE – Input capture channel 8 interrupt enable bit
IC7IE – Input capture channel 7 interrupt enable bit
INT1IE – External interrupt 1 enable bit
```

| name | ADDR | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| IEC2 | 0x0090 | - | - | - | - | - | - | - | LVDIE | DCIIE |

**Table 3-9. Interrupt enable control register 2 - IEC2**

| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | - | 0x0000 |

**Table 3-9. continued**

```
LVDIE – Programmable low voltage detect interrupt enable bit
DCIIE – Data converter  interface interrupt enable bit
```

| name | ADDR | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| ICP0 | 0x0094 | - | T1IP<2:0> | | | - | OC1IP<2:0> | | |
| ICP1 | 0x0096 | - | T3IP<2:0> | | | - | T2IP<2:0> | | |
| ICP2 | 0x0098 | - | ADIP<2:0> | | | - | U1TXIP<2:0> | | |
| ICP4 | 0x009C | - | OC3IP<2:0> | | | - | IC8IP<2:0> | | |
| ICP5 | 0x009E | - | INT2IP<2:0> | | | - | T5IP<2:0> | | |
| ICP6 | 0x00A0 | - | C1IP<2:0> | | | - | SPI2IP<2:0> | | |
| ICP7 | 0x00A2 | - | - | - | - | - | - | - | - |
| ICP8 | 0x00A4 | - | - | - | - | - | - | - | - |
| ICP9 | 0x00A6 | - | - | - | - | - | - | - | - |
| ICP10 | 0x00A8 | - | - | - | - | - | LVDIP<2:0> | | |

**Table 3-10. Special function registers associated wuth interrupt controllers**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|
| - | IC1IP<2:0> | | | - | INT0IP<2:0> | | | 0x0000 |
| - | T2IP<2:0> | | | - | IC2IP<2:0> | | | 0x0000 |
| - | U1TXIP<2:0> | | | - | SPI1IP<2:0> | | | 0x0000 |
| - | SI2CIP<2:0> | | | - | NVMIP<2:0> | | | 0x0000 |
| - | IC7IP<2:0> | | | - | INT1IP<2:0> | | | 0x0000 |
| - | T4IP<2:0> | | | - | OC4IP<2:0> | | | 0x0000 |
| - | U2TXIP<2:0> | | | - | U2RXIP<2:0> | | | 0x0000 |
| - | - | - | - | - | - | - | - | 0x0000 |
| - | - | - | - | - | - | - | - | 0x0000 |
| - | - | - | - | - | - | - | - | 0x0000 |
| - | DCIIP<2:0> | | | - | - | - | - | 0x0000 |

**Table 3-10. continued**

```
T1IP<2:0> - Timer1 interrupt priority bits
OC1IP<2:0> - Output compare channel 1 interrupt priority bits
IC1IP<2:0> – Input capture channel 1 interrupt priority bits
INT0IP<2:0> - External interrupt 0 priority bits
T3IP<2:0> - Timer3 interrupt priority bits
T2IP<2:0> - Timer2 interrupt priority bits
OC2IP<2:0> - Output compare channel 2 interrupt priority bits
IC2IP<2:0> -Input capture channel 2 interrupt priority bits
ADIP<2:0> - A/D conversion complete interrupt priority bits
U1TXIP<2:0> - UART1 transmitter interrupt priority bits
U1RXIP<2:0> - UART1 receiver interrupt priority bits
SPI1IP<2:0> - SPI1 interrupt priority bits
CNIIP<2:0> - Input change notification interrupt priority bits
MI2CIP<2:0> - I2C module (master mode) transmitter interrupt priority level
bits
SI2CIP<2:0> - I2C module (slave mode) receiver interrupt priority level
bits
NVMIP<2:0> - Non-volatile memory write interrupt priority bits
OC3IP<2:0> - Output compare channel 3 interrupt priority bits
IC8IP<2:0> - Input compsre channel 8 interrupt priority bits
IC7IP<2:0> - Input compare channel 7 interrupt priority bits
INT1IP<2:0> - External interrupt 1 priority bits
INT2IP<2:0> - External interrupt 2 priority bits
T5IP<2:0> - Timer5 interrupt priority bits
T4IP<2:0> - Timer4 interrupt priority bits
OC4IP<2:0> - Output compare channel 4 interrupt priority bits
C1IP<2:0> - CAN1 (combined)interrupt priority bits
SPI2IP<2:0> - SPI2 interrupt priority bits
U2TXIP<2:0> - UART2 transmitter interrupt priority bits
U2RXIP<2:0> - UART2 receiver interrupt priority bits
LVDIP<2:0> - Programmable low voltage detect interrupt priority bits
DCIIP<2:0> - Data converter interface interrupt priority bits
```

**Les Timers**

introduction

Timers sont des périphériques de base de chaque microcontrôleur. Selon le modèle, la famille dsPIC30F propose plusieurs modules de minuterie 16 bits. Le microcontrôleur dsPIC30F4013 contient cinq modules de la minuterie.

Chaque module de temporisateur contient une minuterie 16 bits / compteur constitué par les registres suivants:

· TMRx - registre de compteur minuterie 16 bits,

· PRx - registre contenant la valeur de la période sur 16-bits,

· TxCON - registre de contrôle 16 bits pour le mode sélection de la minuterie .

Chaque module de minuterie a également des bits associés pour commande d'interruption:

· TxIE - bit de validation de commande d'interruption,

· TxIF - bit d'état du drapeau d'interruption,

· TxIP <2: 0> - trois bits de commande de priorité d'interruption (dans le registre d'interruption IPCX).

La plupart des minuteries dans la famille de microcontrôleurs dsPIC30F ont le même circuit fonctionnel. Selon leurs différences fonctionnelles, ils sont classés en trois types: A, B, ou C. Les temporisateurs des types B et C peuvent être combinés pour former un temporisateur 32 bits.

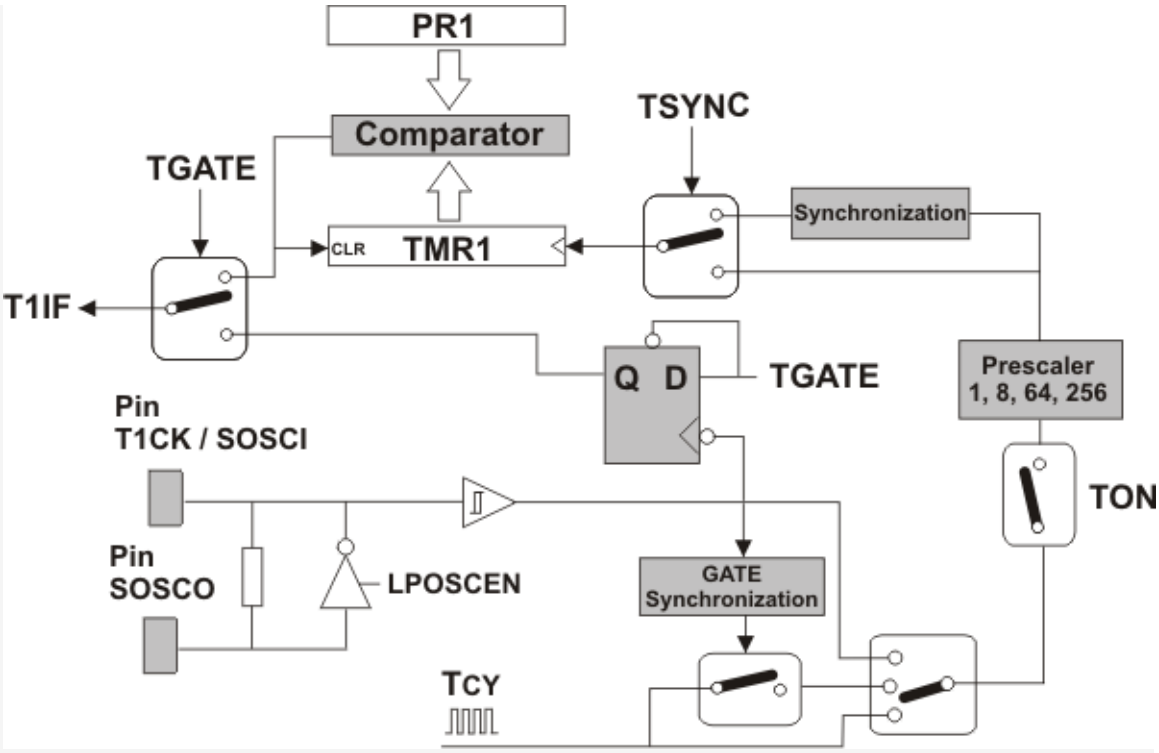
Type de minuterie A

Une minuterie est disponible sur la plupart des appareils dsPIC30F. Le Timer1 est une minuterie de type A. Une minuterie de type A a les caractéristiques uniques suivantes sur les autres types:

· Peut être actionnée par le dispositif de faible puissance oscillateur 32 Khz,

· Peut être actionnée dans un mode asynchrone à partir d'une source d'horloge externe.

La caractéristique unique d'un type A est qu'il peut être utilisé comme l'horloge temps réel (RTC). Un schéma bloc du type A est représentée sur la Fig.




Une minuterie de type A est un objectif général (GP) de minuterie 16 bits et dans le microcontrôleur dsPIC30F4013 il est noté comme module timer1.
Le module Timer1 est un temporisateur 16 bits destiné à être utilisé principalement comme un compteur de temps réel, ou de période, ou d'événement compteur dans le mode « free-running ». Le Timer1 peut être configuré pour fonctionner en tant que:

1. un temporisateur 16 bits,
2. un compteur synchrone de 16 bits,
3. un compteur asynchrone 16 bits.

Le timer1 a aussi la capacité de fonctionner comme une minuterie fermée, pour sélectionner un prescaler, fonctionner pendant des états SLEEP ou des états de repos du microcontrôleur, ainsi que de générer une demande d'interruption lorsque la valeur du registre TRM1 est égale à celle de la période du registre PR1. La condition pour générer une demande d'interruption peut être le front descendant d'un signal de déclenchement externe (TGATE).

La commande du fonctionnement du module de timer1 est déterminée en réglant les bits de configuration spéciale registre 16 bits de fonction (SFR) T1CON.

**mode de minuterie 16 bits**

Dans le mode de minuterie 16 bits, la valeur du compteur est incrémentée par TMR1 chaque cycle d'instruction jusqu'à ce qu'elle soit égale à la valeur prédéterminée par le registre PR1, lorsque le compteur est remis à "0" et redémarré. À ce moment, une demande d'interruption pour le module de timer1 T1IF (dans le registre IFS0) est générée. Le traitement de cette requête dépend du bit de validation d'interruption T1IE (dans le registre IECO). Le module de minuterie continue à fonctionner pendant la routine d'interruption.
LE module Timer1 peut fonctionner pendant l'état IDLE si le bit TSIDL (T1CON <13>) est remis à zéro, mais si ce bit est positionné, le compteur continue à fonctionner après que le processeur soit éveiller à partir de l'état IDLE.
Attention!
L'Interruption du module de timer1 est générée sur une demande d'interruption fixée par module de timer1 seulement si l'interruption est activée en réglant le bit T1IE dans le registre IEC0. Le bit de demande d'interruption.
T1IF dans le registre de IFS0 doit être remis à 0 par le logiciel après que l'interruption est générée.
L'exemple suivant montre comment le module timer1 peut être utilisé dans le temporisateur 16 bits

Exemple:
Allumer et éteindre une LED sur le port D environ quatre fois par seconde. Cet exemple utilise un module timer1 ayant horloge 256 fois plus lente que l'horloge dsPIC. A chaque 10 000 coups d'horloges timer1 appelle la routine d'interruption Timer1Int et change la valeur sur le port D.

```
Void Timer1Int void () {// org 0x1A adresse Minuteur1 dans le tableau de vecteur d'interruption
LATD = ~ PORTD; // inversion PORTD
IFS0 = IFS0 & 0xfff7; // réinitialisation drapeau Interruption
}
void main () {
TRISD = 0; //  PORTD en sortie
LATD = 0xAAAA; // Valeur initiale du port D
```

```
IPC0 = IPC0 | 0x1000; // Niveau de priorité est 1
IEC0 = IEC0 | 0x0008; //  interruption Timer1 activée
PR1 = 10000; // Période d'interruption est 10 000 coups  horloges
T1CON = 0x8030; // Timer1 activé (horloge interne divisée par 256)
while (1) asm nop; // Boucle sans fin
}
```

Comment peut-on calculer la période de demande d'interruption? Laissez l'horloge interne réglée sur
10MHz. La période est 100 ns. Puis l'horloge est divisée par 256 (diviseur réduit l'horloge 1: 256) pour former l'horloge de minuterie, il suit que l'horloge de la minuterie est de 100 * 256 = 25600ns c'est-à-dire 25.6µs. A chaque 10 000 coups d'horloge une interruption est demandée, c'est à dire à chaque 256ms soit environ 4 fois par seconde.

T = 10000 * 25.6µs = 256ms ~ ¼s.

**Mode compteur synchrone 16 bits**

Dans le mode de compteur 16 bits synchrone les incréments de TMR1 de valeur de compteur s'effectuent sur chaque front montant d'une source d'horloge externe. Dans ce mode, la phase de l'horloge interne est synchronisée avec l'horloge externe. Lorsque la valeur du compteur TMR1 est égale à la valeur prédéfinie du registre PR1, le compteur TRM1 est remis à zéro  et commence à compter à partir de "0". À ce moment, une demande d'interruption pour le module de timer1 T1IF (dans le IFS0 de registre) est générée. Le traitement de cette demande dépend de l'activation du bit interruption T1IE  (dans le registre IEC0). Le module de minuterie continue à fonctionner pendant la routine d'interruption.

Le but du module timer1 est de permettre la mesure des périodes d'horloge des signaux très rapide, par exemple la détection automatique de la vitesse de communication de l'interface série universel UART.

Le module de timer1 pourrait fonctionner dans ce mode pendant l'état IDLE si le bit TSIDL (T1CON <13>)
est remis à zéro; si ce bit est positionné, le compteur continue à fonctionner après que le processeur soit éveiller à partir de l'état IDLE .

L'exemple suivant montre comment le module timer1 peut être utilisé dans le mode compteur synchrone.

Exemple:
Compter chaque dixième impulsion et incrémenter la valeur sur le port D. Dans cet exemple timer1 est utilisé pour compter les impulsions d'horloge externe à la broche T1CK. Après dix impulsions l'interruption Timer1Int se produit et la valeur du port D est incrémentée. Schéma de la figure montre l'interconnexion entre le timer1 et la source d'horloge externe.

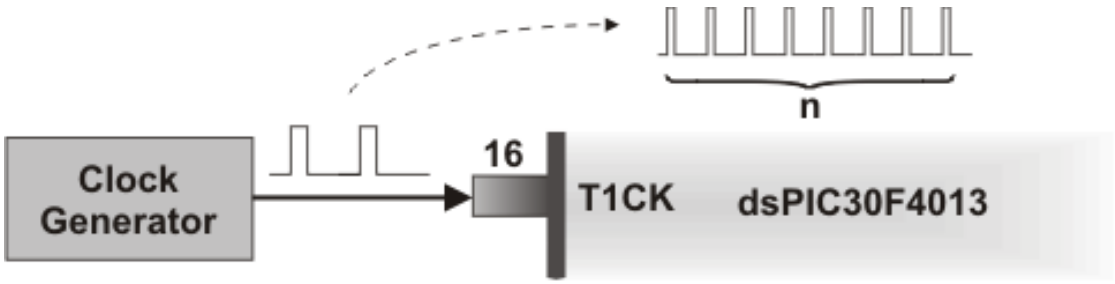



Schéma de la connexion d'une source d'horloge externe dsPIC30F4013

```
Void Timer1Int void () {// org 0x1A adresse Minuteur1 dans le tableau de vecteur d'interruption
LATD = ~ PORTD; // Valeur  PORTD est incrémentée
IFS0 = IFS0 & 0xfff7; // Demande d'interruption  réinitialisée
}
void main () {
TRISD = 0; //  PORTD en sortie
TRISC = 0x4000; // PORT<14> = 1   broche T1CK en entrée
LATD = 0; // Valeur initiale du port D
IPC0 = IPC0 | 0x1000; // Niveau de priorité est 1
IEC0 = IEC0 | 0x0008; // interruption Minuteur1  activée
PR1 = 10; // Période d'interruption est 10 coups horloge
T1CON = 0x8006; // Timer1 est compteur synchrone d'
Impulsions extérieures
while (1) asm nop; // Boucle sans fin
}
```

Comment configurer le module  timer 1 pour fonctionner en mode synchrone 16 bits?
Le rapport Prescaler 1: 1 est sélectionné, horloge externe TCS = 1 est activée, et le fonctionnement de timer1

Module TON = 1 est activé (T1CON = 0x8006;). En définissant le bit TRISC <14> = 1 la broche
PORTC <14> = 1 est configurée comme entrée.

**Mode compteur asynchrone 16 bits**

Dans le mode de compteur 16 bits asynchrone TMR1 les incréments de valeur de compteur s'effectuent sur chaque front montant d'une source d'horloge externe, mais la phase de l'horloge interne n'est pas synchronisée avec l'horloge externe. Lorsque la valeur du compteur TMR1 est égale à la valeur prédéfinie le registre PR1, le compteur TMR1 est remis à zéro et commence à compter à partir de "0". À ce moment, une demande d'interruption pour le module de timer1 T1IF (dans le IFS0 de registre) est générée. Le traitement de cette demande dépend de l'activation du bit d'interruption T1IE (dans le registre IEC0). Le module de minuterie continue à fonctionner pendant la routine d'interruption.
Le module de timer1 pourrait fonctionner dans ce mode pendant l'état IDLE si le bit TSIDL (T1CON <13>)
est remis à zéro; si ce bit est positionné, le compteur continue à fonctionner après que le processeur soit éveillé à partir de l' état IDLE.
Exemple:
Comptez chaque 800ème impulsion et incrémenter la valeur sur le port D. Dans cet exemple timer1 est utilisé pour
compter chaque 8 impulsions d'une horloge externe sur T1CK .
Après 100 impulsions une routine d'interruption Timer1Int est appelée et la valeur au port D est incrémentée.

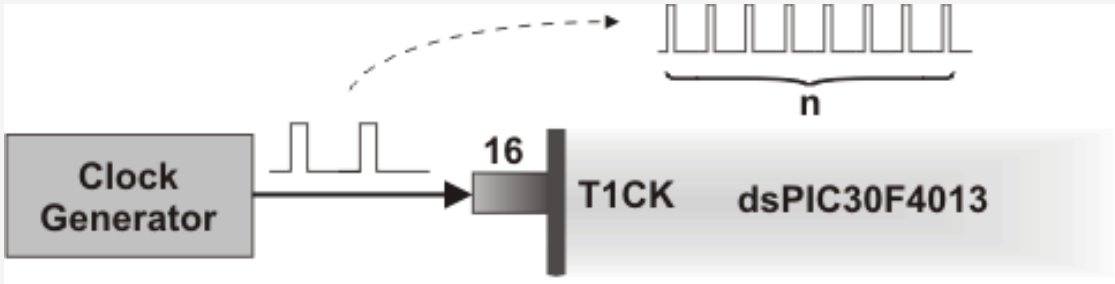



Schéma de la connexion d'une source d'horloge externe dsPIC30F4013

```
void Timer1Int void () {// org 0x1A adresse Minuteur1 dans le tableau de vecteur d'interruption
LATD ++; // Valeur PORTD  incrémentée
IFS0 = IFS0 & 0xfff7; // Demande d'interruption est réinitialisée
}
void main () {
TRISD = 0; //  PORTD en sortie
TRISC = 0x4000; // PORT <14> = 1 T1CK en d'entrée
LATD = 0; // Valeur initiale du port D
IPC0 = IPC0 | 0x1000; // Niveau de priorité est 1
IEC0 = IEC0 | 0x0008; // interruption Timer1  activée
PR1 = 100; // Période d'interruption est 100 coups horloge
T1CON = 0x8012; // Timer1 est compteur synchrone d’impulsions extérieures
while (1) asm nop; // Boucle sans fin
}
```

**mode de minuterie Gated**

Lorsque le module  timer1 fonctionne en mode de minuterie 16 bits, il peut être configuré pour permettre la mesure de la durée d'un signal de déclenchement externe (grille d'accumulation de temps). Dans ce mode, le compteur TRM1 est incrémenté par l'horloge interne d'instruction (TCY) aussi longtemps que le signal  externe de GATE (broche de T1CK) est au niveau logique haut. Pour que le module fonctionne en timer1, il est nécessaire que le bit TGATE  de commande (T1CON <6>) soit activé, l'horloge interne (TCS = 0) est non sélectionnée, et le fonctionnement de la minuterie est activée (TON = 1).

Le module de timer1 pourrait fonctionner dans ce mode pendant l'état IDLE si le bit TSIDL (T1CON <13>)
est remis à zéro; si ce bit est positionné, le compteur continue à fonctionner après que le processeur est éveillé
à partir de l’état IDLE.

Exemple:

Utilisez timer1 dans l'accumulation de temps mode fermé. Le signal de validation de grille est appliqué à la
broches T1CK. Mesurer la largeur du signal et afficher le résultat sur le port D.

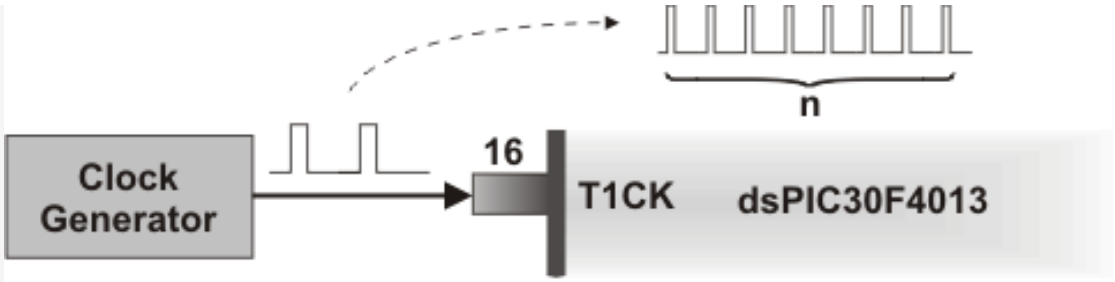



Schéma de la connexion d'une source d'horloge externe dsPIC30F4013

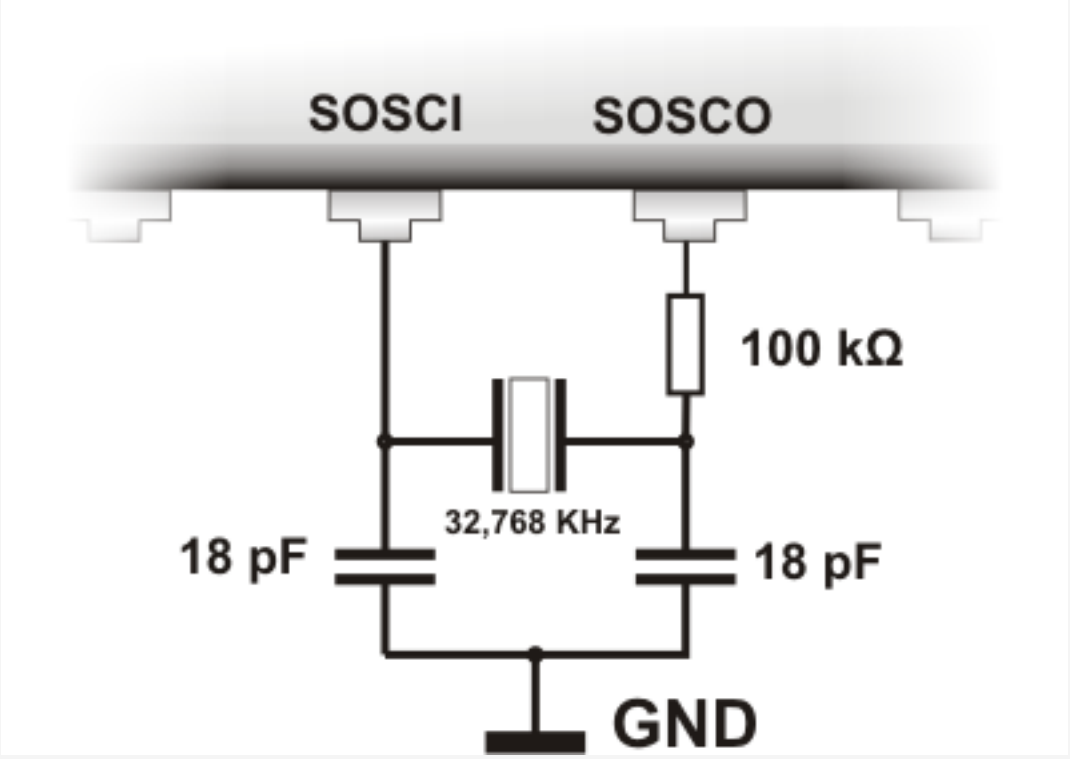
```
voidTimer1Int void () {// org 0x1A adresse Minuteur1 dans le tableau de vecteur d'interruption
LATD = TMR1; // La durée d'impulsion est affichée au port D
IFS0 = IFS0 & 0xfff7; // Demande d'interruption est traitée
}
main () {
TRISD = 0; //  PORTD en sortie
TRISC = 0x4000; // PORT <14> = 1 T1CK est en entrée
LATD = 0; // Valeur initiale du port D
IPC0 = IPC0 | 0x1000; // Niveau de priorité est 1
IEC0 = IEC0 | 0x0008; // interruption Timer1 activée
PR1 = 0xFFFF; // Période est maximale
T1CON = 0x8040; // Timer1 est activé, l'horloge interne jusqu'à T1CK = 1
while (1) asm nop; // Boucle sans fin
}
```

Pourquoi le registre PR1 est mis à la valeur maximale? La raison principale est de permettre la mesure aussi longtemps que des intervalles de temps sont possibles, c'est à dire l'interruption ne vient pas en conséquence de l'égalisation des registres TMR1 et PR1 mais, si possible, à la suite de la chute  du signal de porte.
Dans le fonctionnement du module de timer1 les applications nécessitent souvent l'utilisation de l'étage d'adaptation ce qui permet que l'horloge peut être réduit en le rapport 1: 1, 1: 8; 1:64 ou 1: 256. De cette façon le champ des applications du module de timer1 est élargi considérablement. La sélection du prédiviseur (prescaler)se fait par le réglage de la commande des bits de TCKPS <1:> (T1CON <5: 4>). Le prédiviseur compteur est remis à chaque écriture dans le registre de compteur TMR1 ou registre de contrôle T1CON, ou après

la réinitialisation du microcontrôleur. Cependant, il est important de noter que l'étage d'adaptation compteur ne peut pas être remis à zéro lorsque le module de timer1 est interrompue (TON = 0) le diviseur horloge est arrêté.
Attention!
Le registre compteur TMR1 n'est pas réinitialisé lorsque une nouvelle valeur est écrite dans le registre de commande T1CON. Le contenu du registre TMR1 doit changer après avoir écrit une nouvelle valeur dans ce registre.
Le fonctionnement du module timer1 en mode SLEEP n'est possible que si les conditions suivantes sont remplies:
· Le fonctionnement du module de timer1 est activé (TON = 1),
· Module de timer1 utilise une horloge externe,
· bit de commande TSYNNCH (T1CON <2>) est remis à zéro définissant une horloge externe asynchrone comme source. C'est à dire l'horloge est indépendante de l'horloge interne du microcontrôleur.
De cette manière, il est admis que le module de comptage timer1 continue tant que le registre TMR1 est égal à la présélection (période) du registre PR1. Puis, TMR1 est réinitialisé et une interruption est générée.
Une demande d'interruption pour le module de timer1, si le bit de validation est positionné, peut réveiller le microcontrôleur du mode SLEEP.
Le module de timer1 génère une demande d'interruption lorsque les valeurs du registre de compteur TMR1 et prédéfinie (période) du registre-PR1 sont égales. La demande d'interruption est effectuée par la mise en position des Bits T1IF dans le registre d'interruption IFS0. Si le bit T1IF du registre IEC0 est positionné, une interruption est générée. Le bit T1IF doit être une réinitialisation logicielle dans la routine d'interruption.
Si le module de timer1 fonctionne dans le mode d'accumulation de temps fermé, la demande d'interruption sera générée à chaque front descendant du signal de déclenchement externe, c'est à dire à la fin de chaque cycle d'accumulation.

**Mode de fonctionnement horloge temps réel (RTC)**
Le module timer1 peut être réglé pour fonctionner dans le mode de fonctionnement de l'horloge en temps réel, c'est à dire une horloge

en temps réel. De cette façon, on obtient les informations des instants de temps (heures, minutes, secondes) servant à la preuve d'événements. Les principales caractéristiques du fonctionnement du mode RTC du module de timer1 sont: l'utilisation de l'oscillateur 32 kHz, diviseur 8 bits, faible puissance, la capacité de génération d'une demande d'interruption RTC. Comme tout le fonctionnement du module timer1, ce mode est réglé par les bits de contrôle dans le registre T1CON. Bien que le module timer1utilise l'horloge de l'oscillateur à 32 kHz pour un fonctionnement en mode RTC, le restant du microcontrôleur est capable de fonctionner avec une autre horloge qui est réglable par le bit commande dans le registre de contrôle FOSC.
Dans ce mode, lorsque les bits de commande sont TON = 1, TCS = 1, et TGATE = 0, le registre de compteur TMR1 est incrémenté par chaque front montant de l'oscillateur 32 Khz basse puissance jusqu'à ce que la valeur du registre de compteur de TMR1 soit égale à la valeur prédéterminée du registre PR1; le compteur registre TMR1 est ensuite remis à zéro.
Attention!
Pour que timer1 fonctionne correctement en mode RTC, le bit de contrôle TSYNC doit être réinitialisé. En réglant TSYNC = 0 le mode asynchrone du module timer1 est défini. Aussi, le bit LPOSCEN (T1CON <1>) doit être positionné pour désactiver tous les autres modes de fonctionnement, à l'exception RTC et permettre le réveil du microcontrôleur à partir de l'état SLEEP par le module de timer1. Il est très important que le bit TSIDL soit remis à 0 pour permettre le fonctionnement du module timer1 pendant l'état IDLE.
Avec ce module timer1 configuré, dans l'état SLEEP, le TRC continuera le fonctionnement cadencé de l'oscillateur 32kHz et les bits de contrôle ne seront pas modifiés.
Lorsque la condition de génération d'une demande d'interruption est satisfaite (TMR1 = PR1), le bit T1IF dans le registre d'interruption IFS0 est positionné. Si l'interruption correspondante est activée (l'activation du bit T1IE est définie dans le registre IEC0), une interruption du microcontrôleur est générée. Pendant l'interruption de routine le bit T1IF doit être remis à zéro, sinon aucune autre demande d'interruption ne pourrait être détectée par le module timer1.




Raccordement d'un oscillateur à quartz en mode RTC

| name | ADR | 15 | 14 | 13 | 12-7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TMR1 | 0x0100 | Timer 1 Register | | | | | | | | | | | 0x0000 |
| PR1 | 0x0102 | Period Register 1 | | | | | | | | | | | 0xFFFF |
| T1CON | 0x0104 | TON | - | TSIDL | - | TGATE | TCKPS1 | TCKPS0 | - | TSYNC | TCS | - | 0x0000 |

**Table 4-1 Registers of timer1 module**

```
TON – Timer1 module on bit (TON=1 starts timer, TON=0 stops timer)
TSIDL – Stop in IDLE mode bit (TSIDL=1discontinue module operation when
        microcontroller enters IDLE mode,
TSIDL = 0 continue module operation in IDLE mode)
TGATE – Timer gated time accumulation mode enable bit (TCS must be set to
logic 0 when TGATE=1)
TCKPS<1:0> - Timer input clock prescale select bits
        00 – 1:1 prescale value
        01 – 1:8 prescale value
        10 – 1:64 prescale value
        11 – 1:256 prescale value
TSYNC – Timer external clock input synchronization select bit
(TSYNC=1 synchronize external clock input, TSYNC=0 do not synchronize
external clock input)
TCS – Timer clock source select bit
(TCS=1 external clock from pin T1CK, TCS=0 internal clock Fosc/4)
```

**NOTE:** Unimplemented bits read as '0'.

Minuterie Type B

La minuterie Type B est une minuterie 16 bits présente dans la plupart des appareils de la famille des dsPIC30F

. Il est noté comme le module timer2 et le module timer4 dans dsPIC30F4013. La minuterie a les caractéristiques spécifiques suivantes:

· Une horloge de type B peut être concaténée avec un temporisateur de type C pour former un temporisateur 32 bits,

· La synchronisation d'horloge pour un compteur de type B est effectuée après l'étage d'adaptation.

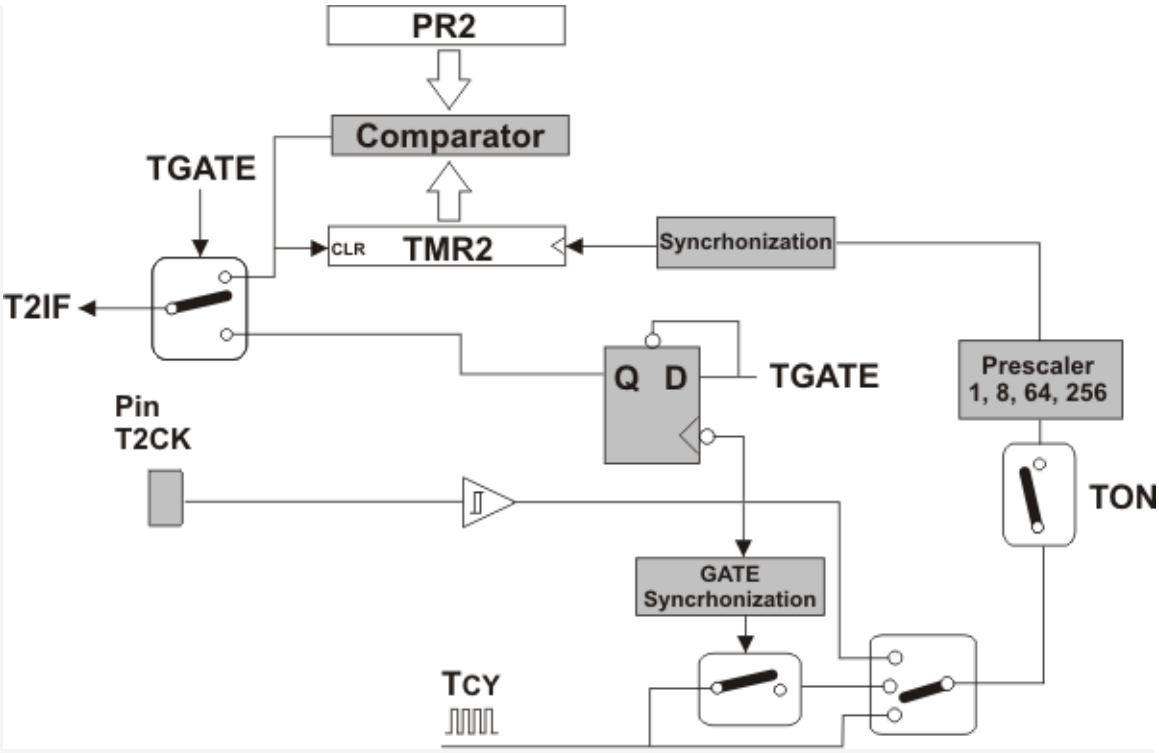



schéma fonctionnel

**Minuterie type C**

Le temporisateur type C est un temporisateur 16 bits. Le plus souvent, il est utilisé en association avec le type B pour former un 32 bits à usage général. Les timers de ce type, s'ils sont présents, dans la famille des dsPIC30F sont désignés par le module de timer3 et le module timer5 comme dans dsPIC30F4013. La minuterie a les caractéristiques spécifiques suivantes:

· Une horloge de type C peut être concaténée avec une minuterie de type B pour former un temporisateur 32 bits,

· Au moins une minuterie de type C a la capacité de déclencher une conversion A / D.

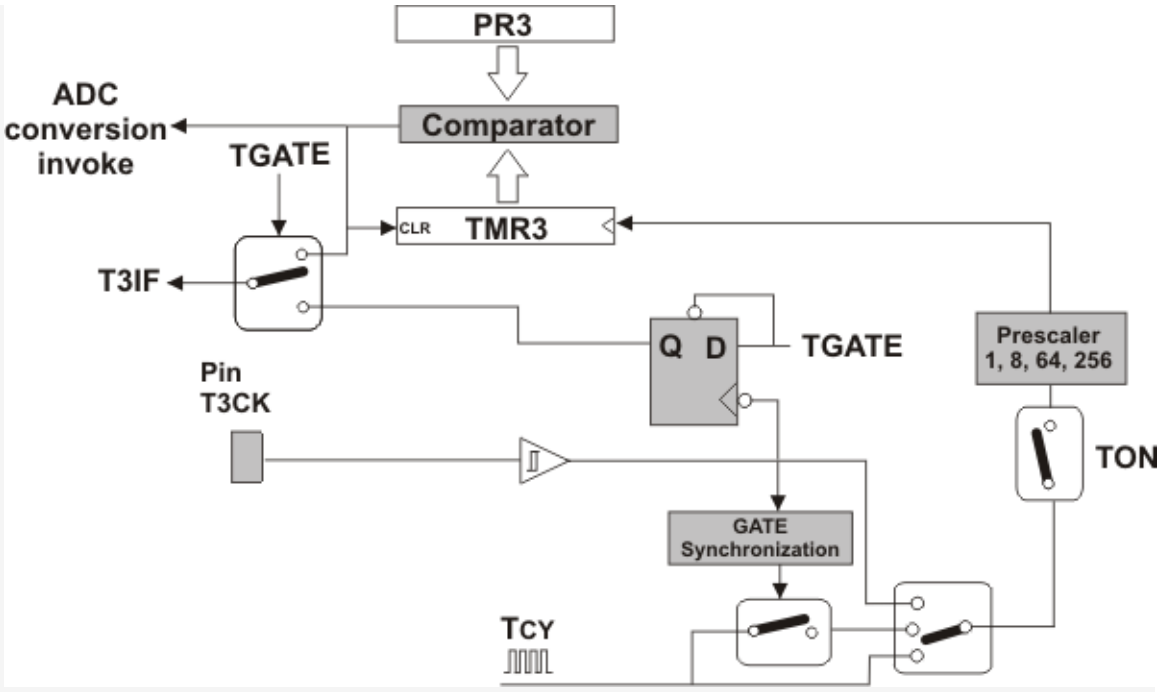



schéma fonctionnel

Attention !
La Pin T3CK n’existe pas dans les dispositifs dsPIC30F4013. La figure montre l'organisation de la minuterie d'un dispositif dsPIC30F6014A. En outre, le module de timer5 n'a pas la capacité de déclencher une conversion A / D.

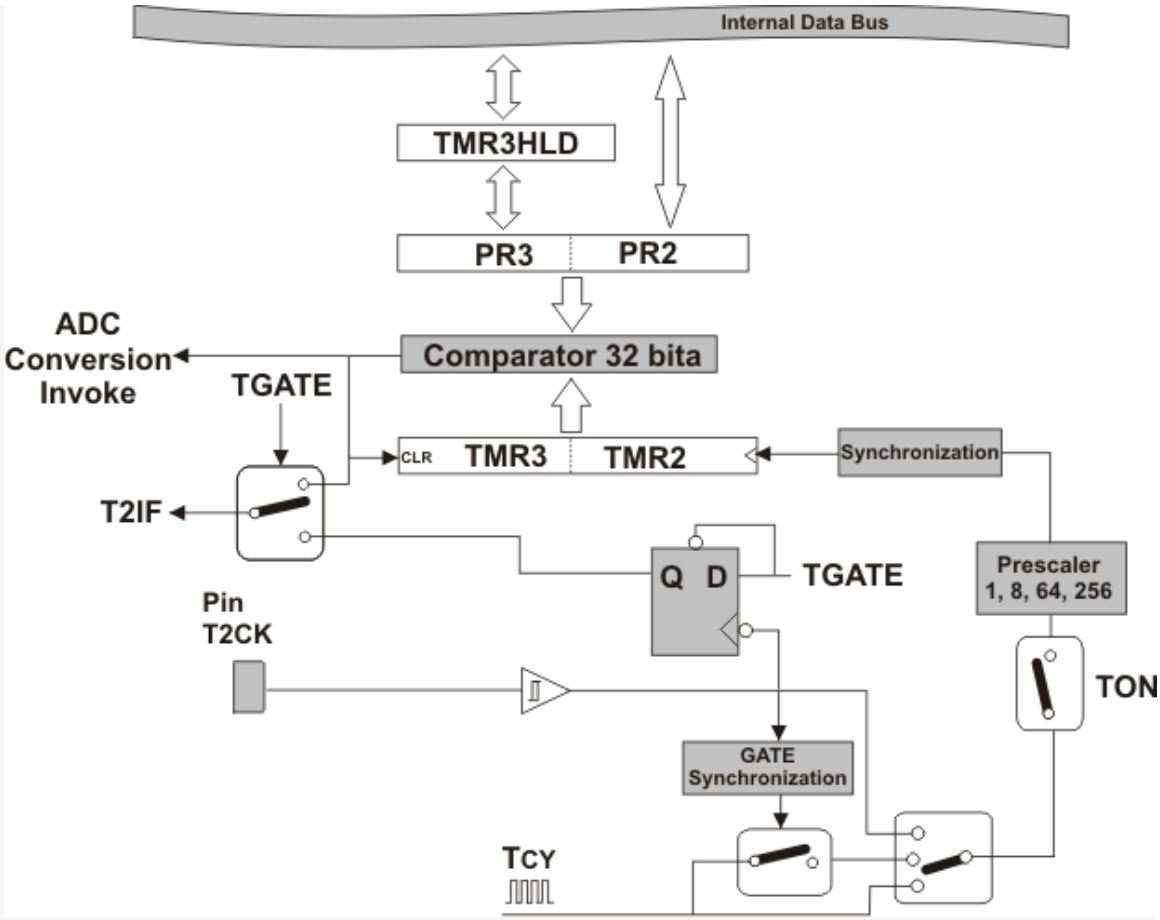



Schéma fonctionnel de la minuterie concaténée 32 bits

La minuterie de 32 bits de la Fig. est formée en combinant les modules de TIMER2 et Minuterie3. Les modules de timer4 timer5 pourraient être combinés d'une manière similaire. La minuterie 32 bits formée est une minuterie à usage général (GP) et peut être configurée par les bits de contrôle dans les modes suivants:

· Minuterie 32 bits.

· Compteur 32 bits synchrone.

Il est significatif de noter que les modules de TIMER2 et TIMER3 pourraient indépendamment fonctionner comme minuteries de 16 bits dans les modes opérationnels tout comme le module de timer1, sauf dans le mode compteur asynchrone. Il en va de même pour les modules de timer4 et timer5.

Le Timer2 et des modules de TIMER3 utilisent d'autres périphériques, la capture ou Input Output comparer,

afin de réaliser de manière simple un signal d'impulsion modulé en largeur (PWM) ou un convertisseur A / D pour déclencher la conversion à des moments d'échantillonnage précis qui ne est pas possible avec le timer4 et timer5.
Les timer2 / module 3 32 bits concaténés ont la capacité de:
· A / D événement déclencheur,
· Fonctionner dans un mode d'accumulation de temps,
· Diviseur sélectionneur,
· Fonctionner en état IDLE,
· Générer une demande d'interruption sur les registres compteurs concaténés TMR2 / TMR3 égale avec le registre « preset » PR3 / PR2.
Le réglage du mode de fonctionnement du module de concaténation timer2 / 3 est effectué par les bits de contrôle dans T2CON et T3CON.
Attention!
Si les timer2 et trois fonctionnent de façon indépendante, les modes de fonctionnement sont ajustés par
T2CON et T3CON. S' ils opèrent en tant que module de concaténation timer2 / 3, alors les bits de commande T2CON sont adaptés et des bits de commande de T3CION sont ignorés. L'horloge timer2 et les entrées de porte sont utilisées pour concaténer timer2 / module 3 32 bits. Une interruption de la minuterie 32 bits est générée avec le drapeau T3IF et T3IE correspondant au bit de validation.

**Mode de minuterie 32 bits**
Dans le mode de minuterie 32 bits la minuterie est incrémentée à chaque cycle d'instruction jusqu'à ce que la valeur du registre compteur concaténé TMR3 / TMR2 soit égale aux registres préréglés concaténés PR3 / PR2. Ensuite, le registre de compteur est remis à «0», et une demande d'interruption est générée avec le bit T3IF.
REMARQUE: la lecture synchrone du registre 32 bits TMR3 / TMR2 est effectuée par la lecture du registre timer2 16 bits comme le mot moins significatif (LS). Pendant la lecture du registre TMR2 la valeur du registre TMR3 est transférée au registre temporaire TMR3HLD. Le procédé de la lecture 32 bits est conclu à la lecture de la valeur du Mot le plus significatif (MS) du registre TMR3HLD.

L'écriture 32 bits synchrone est réalisée en deux étapes inversement à la lecture. Au début, le mot le plus significatif est écrit dans le TMR3HLD puis le mot le moins significatif est écrit dans TMR2. Pendant l'écriture dans le registre TMR2 les Valeurs de TMR3HDL et du registre compteur TMR3 sont transférées à TMR2.
L'exemple suivant montre comment les timer2 et 3 peuvent être utilisés en minuterie 32 bits.
Exemple:
Allumer et éteindre une LED sur le port D environ une fois toutes les deux secondes.
L'exemple utilise le timer2 et trois ayant horloge 256 fois plus lente que celle du dsPIC.
À chaque 100 000 coups d'horloges de timer1 une routine d'interruption Timer23Int est appelée et la valeur au port D est modifiée.

```
Void Timer23Int void () {// org 0x22 Adresse dans la table de vecteur d'interruption de timer3
LATD = ~ PORTD; // Inverser port D
IFS0 = 0; // Demande d'interruption Effacée
}
void main () {
TRISD = 0; // Port D en sortie
LATD = 0xAAAA; // Valeur initiale port D
IPC1 = IPC1 | 0x1000; // Priorité Minuterie3 est 1
IEC0 = IEC0 | 0x0080; // interruption Minuterie3 activée
PR2 = 34464; // Période d'interruption est de 100 000 coups horloges
PR3 = 0x0001; // Total PR3 / 2 = 1 * 65 536 + 34 464
T2CON = 0x8038; // Timer2 / 3 est activé, l'horloge interne est divisée par 256
while (1) asm nop; // Boucle sans fin
}
```

Comment fonctionne un ensemble timer2 / 3 pour le mode de minuterie 32 bits?
Le bit d'interruption correspondant c'est à dire bit de priorité T3IPC = 1, bit de demande d'interruption T3IF = 0, et bit de validation d'interruption T3IE = 1 sont positionnés en premier (dans les timer2 / 3 concaténés contrôlés par des bits de commande du module de timer3 et le fonctionnement du timer2 / module 3 est commandé par des bits de commande du module timer2). Le fonctionnement de la

minuterie  TON est activée = 1, le fonctionnement 32 bits est sélectionné T32 = 1, et dans ce cas, l'étage d'adaptation est configuré pour rapport de 1: 256
TCKPS <1: 0> = 11. Le registre période PR2 / 3 contient la valeur 100 000 répartie selon la formule PR3 / 2 = PR3 * 65536 + PR2, PR3 = 1 et PR2 = 34464.
Comment peut-on calculer la période d'interruption des appels? Laissez l'horloge interne être ajustée à 10MHz. La période de correspondante  est 100 ns. Etant donné que l'horloge est divisée par 256 (le prédiviseur réduit l'horloge 1: 256) pour former l'horloge de la minuterie, il s'ensuit que 100 ns * 256 = 25600ns, C'est à dire 25.6µs. À chaque 100 000 coups d'horloge une interruption est appelée, c'est à dire à chaque 2.56s soit environ une fois toutes les deux secondes. T = 100 000 * 25.6µs = 2.56s.

**Mode compteur synchrone  32 bits**

Dans un mode de comptage de 32 bits synchrone la minuterie concaténée TMR3 / TMR2 est incrémentée sur chaque front montant du signal d'horloge externe qui est synchronisé avec la phase du signal d'horloge interne du microcontrôleur. Lorsque la valeur du registre de compteur TMR3 / TMR2 est égale à la valeur prédéterminée du registre PR3 / PR2, le contenu du registre TMR3 / TMR2 est remis à «0», et une demande d'interruption est établi par le bit T3IF.
L'exemple suivant montre comment le module timer2 / 3 peut être utilisée dans mode compteur synchrone le 32 bits.
Exemple:
Utilisez le timer2 / 3 pour compter les impulsions d'horloge externe à la broche T1CK. Après 10 impulsions une interruption Timer23Int se produit et augmente la valeur sur le port D.

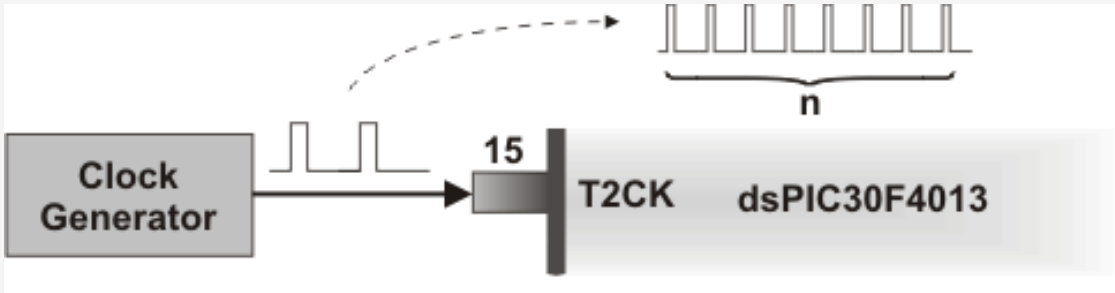



Schéma de connexion

```
Void Timer23Int void () {// org 0x22 Adresse dans la table de vecteur
d'interruption de timer3
LATD ++; // Incrémente la valeur de PORTD
IFS0 = 0; // Demande d'interruption Effacée
}
void main () {
TRISD = 0; //  PORTD en sortie
TRISC = 0x2000; // PORTC <13> = 1 T2CK en entrée
LATD = 0; // Valeur initiale du PORTD
IPC1 = IPC1 | 0x1000; // Priorité d'interruption de timer3 est 1
IEC0 = IEC0 | 0x0080; // Interruption de timer3 activée
PR2 = 10; // Période interruption est 10 coups horloge
PR3 = 0; // Total PR3 / 2 = 0 * 65 536 + 10
T2CON = 0x800A; // Timer2 / 3 est compteur synchrone d'impulsions
 extérieures
while (1) asm nop; // Boucle sans fin
}
```

Comment fonctionne un ensemble minuterie Module 2/3 dans un mode de compteur synchrone? Prescaler 1: 1 est sélectionné, horloge externe est activée TCS = 1, mode 32 bits est activé T32 = 1, le fonctionnement du module de timer1 est activé TON = 1 dans le registre de contrôle T2CON (en mode 32 bits les bits de contrôle T3CON sont ignorés), les bits d'interruption de timer2 / 3 module sont définis (dans le timer2/ 3 interruptions et sont commandés par les bits de commande de module de timer3), la priorité d'interruptions T3IPC = 1, interruption T3IF demande de bit = 0, et d'interrompre bit de validation T3IE = 1.

**Mode d'accumulation de temps fermé**

Le module de minuterie 32 bits concaténés peut être utilisé dans les modes. Cette d'accumulation de temps permet que le compteur TMR3 / TMR2 soit incrémenté par l'horloge interne de TCY aussi longtemps que l'état du signal de porte externe (broche T2CK) est à 1. De cette façon, on peut mesurer la longueur du signal externe. Afin de faire fonctionner la minuterie de 32 bits dans ce mode, il est nécessaire de définir le  bit  de contrôle  TGATE (T2CON <6>) pour activer ce mode, sélectionnez la source d'horloge TCS = 0, et permettre le fonctionnement minuterie TON = 1 dans le registre de contrôle T2CON  . Dans ce mode, le timer2 est la source d'horloge interne. Les bits de commande T3CON du timer3 sont ignorés.

La demande d'interruption est générée sur le front descendant du signal de déclenchement externe ou lorsque la valeur du compteur TMR3 / TMR2 atteint la valeur prédéfinie dans le registre PR3 / PR2.

Attention!

Le front descendant du signal de la porte externe ne réinitialise pas le compteur TMR3 / TMR2; si on le souhaite, ce doit être fait par le logiciel de l'utilisateur.

Exemple:

Utilisation d'une minuterie 2/3 dans le mode d'accumulation de temps. Le signal GATE est appliqué à la broche T1CK. La longueur du signal est mesurée et affichée au port D.

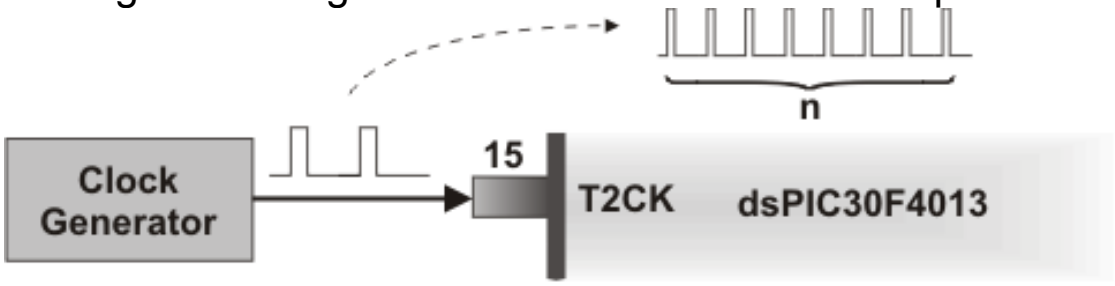



Schéma de la connexion timer2 / 3 à une source d'horloge externe

```
Void Timer2Int void () {// org 0x20 Adresse dans la table de vecteur d'interruption de timer2
LATD = TMR2; // Longueur de signal s'affiche au port D
IFS0.F6 = 0; // Demande d'interruption effacée
}
void main () {
T2CON = 0; // Arrête le registre de contrôle et réinitialise Timer2
TMR2 = 0; // Efface le contenu du registre de minuterie
PR2 = 0xFFFF; // Charge le registre période à 0xFFFF
IFS0.F6 = 0; // Demande d'interruption effacée
T2CON.F6 = 1; // Configure Timer2 pour le mode d'accumulation de temps « Gated »
T2CON.F15 = 1; // Lance Timer2
TRISD = 0; //  PORTD en sortie
TRISC = 0x2000; // PORTC <13> = 1 en entrée
```

```
IEC0.F6 = 1; //  activation d'interruption Timer2
while (1) asm nop; // Boucle sans fin
}
```

Pourquoi la période des registres PR2 et PR3 est mise à la valeur maximale et comment peut-on mesurer de plus longues impulsions?
La raison principale est de mesurer des impulsions les plus longues que possible, c'est à dire que l'interruption ne se produit pas parce que les registres concaténés TMR3 / TMR2 et PR3 / PR2 sont égaux mais, c'est la conséquence du front descendant du signal de porte.
La Mesure d'impulsions encore plus longues  peut être accomplie en réglant sur le diviseur les rapports de réduction 1: 8, 1:64, ou 1: 256. De cette manière, l'étendue de mesure est prolongée mais le la précision est réduite.


Description du mode d'accumulation de temps fermée du timer2 /3 32-bit
Le timer2 / 3 concaténé  a la capacité de déclencher une conversion A / D. Ceci est réalisé en fixant les bits de commande correspondants dans le registre de ADCON <7: 5> (SSRC <2,0> = 010). Quand la valeur du compteur  registre TMR3 / TMR2 devient égale à celle de Registre PR3 / PR2, le convertisseur A / D est triggée et une demande d'interruption est générée.
REMARQUE: Les modules de temporisation 2 et 3 pourraient fonctionner indépendamment en timers 16 bits. Ils pourraient

être configurés pour fonctionner dans les modes suivants: minuterie, 16 bits, compteur 16 bits synchrone, et 16-bit gated accumulation de temps. Les modules de temporisation 2 et 3 ne peuvent pas fonctionner comme compteurs asynchrones 16 bits ni comme sources d'horloge de temps réel (RTC).
En tant que source d'horloge, les timer2 et trois utilisent une source d'horloge externe ou l'horloge interne FOSC / 4 avec la possibilité de choisir le rapport de démultiplication du diviseur de tête 1: 1, 1: 8, 1:64, ou 1: 256.
La sélection du rapport de réduction est obtenue par les bits de commande TCKPS <1: 0> (T2CON <5: 4> et T3CON <5: 4>).
Lorsque le timer2 et 3 forment la concaténation de minuterie 32 bits , puis le rapport de démultiplication du diviseur de tête est sélectionné par le module timer2 et les bits de commande correspondants du module timer3 sont ignorés. Le compteur diviseur est remis à zéro seulement si: l'écriture dans les registres TMR2 ou TMR3, écriture dans les registres PR2 ou PR3, ou le microcontrôleur est remis à zéro. Il est important de noter que le compteur diviseur ne peut pas être remis à zéro lorsque le module timer1 est désactivé (TON = 0) car l'horloge du compteur diviseur est arrêtée.
Aussi, écrire dans T2CON / T3CON ne change pas le contenu de TMR2 / TMR3.
En mode SLEEP, les timer2 et trois ne sont pas fonctionnels l'horloge système est inhibée.
En mode IDLE timer2 et 3 modules continueront l'opération si le bit TSIDL est effacé. Si ce bit est activé, le timer2 et 3 s'arrêtent jusqu'à ce que le microcontrôleur soit éveillé à partir du mode IDLE.
Les caractéristiques des timer4 et 5 sont très semblables à ceux des timer2 et 3. S'ils sont concaténés pour former un temporisateur 32 bits, ils peuvent fonctionner dans les mêmes modes que les timer2 et 3. La seule différence est que les modules timer4 et 5 ne sont pas utilisés par d'autres périphériques Input capture et comparaison de sortie comme les timer2 et 3. En outre, le Module timer5 n'a pas la capacité comme le module de timer3 de déclencher une conversion A / D.

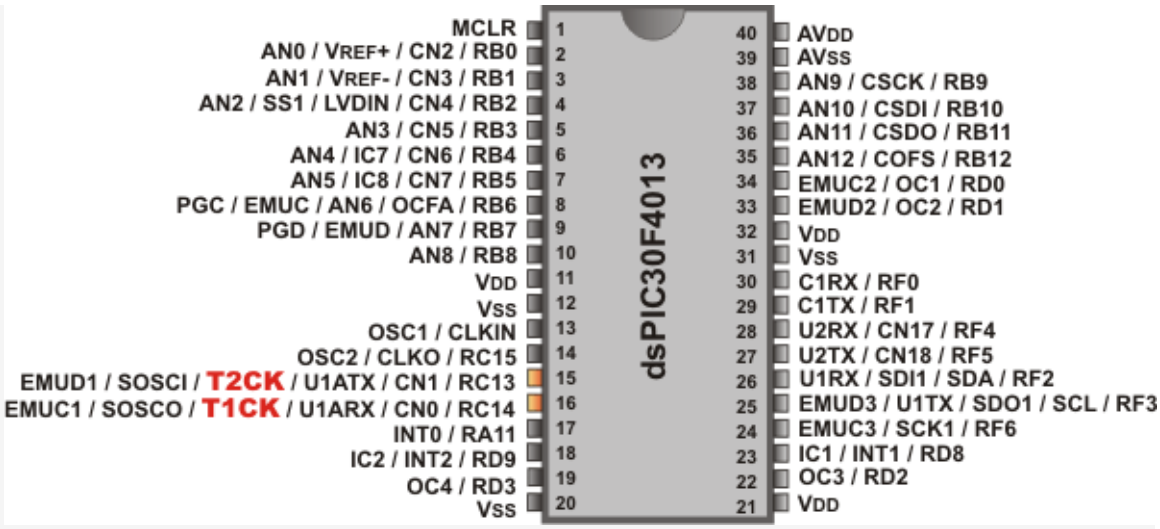
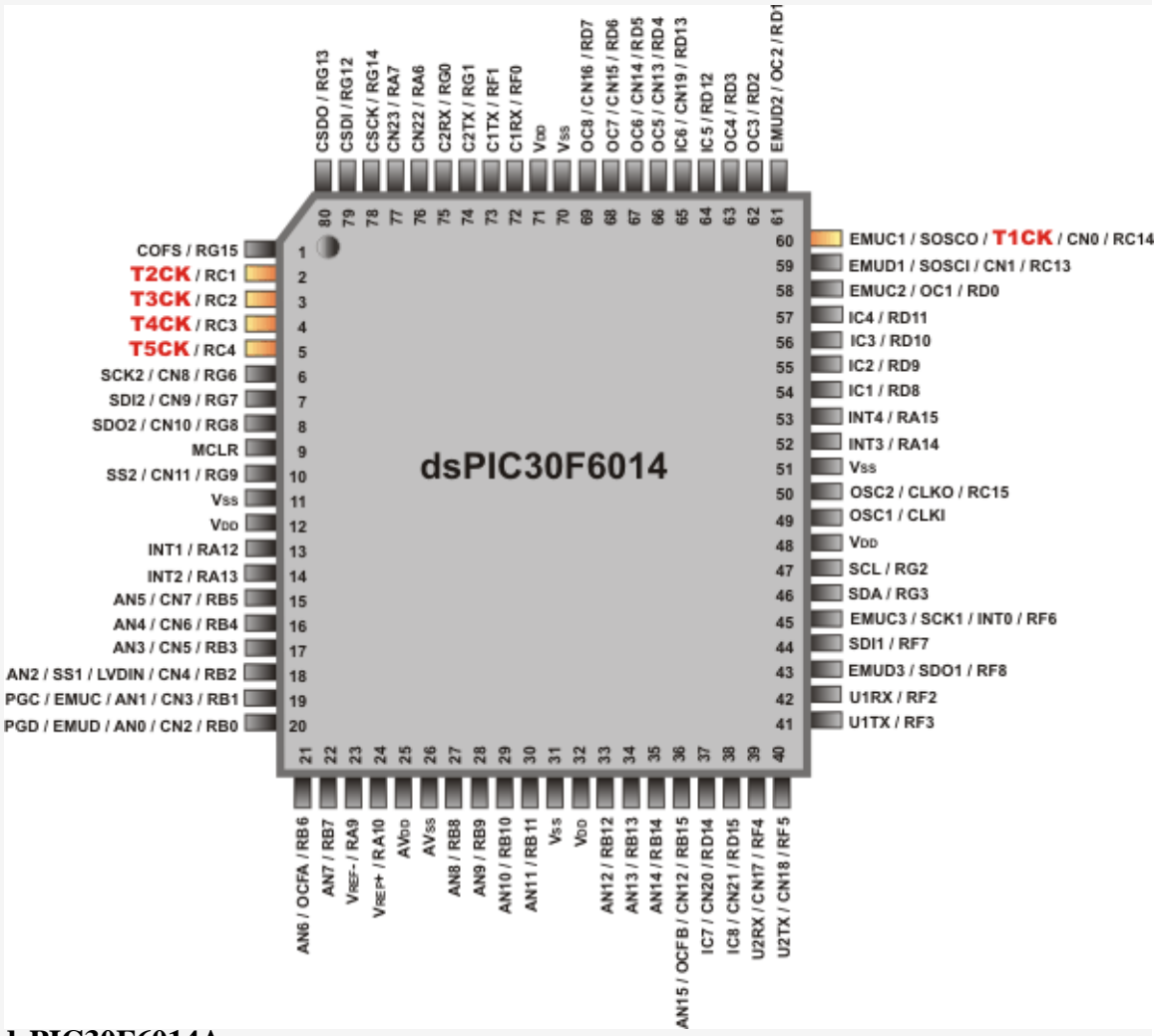



Figure. Schéma dsPIC30F4013




**dsPIC30F6014A**

| name | ADR | 15 | 14 | 13 | 12-7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TMR2 | 0x0106 | Timer2 register | | | | | | | | | | | 0x0000 |
| TMR3HLD | 0x0108 | Timer3 holding register (32-bit operation only) | | | | | | | | | | | 0x0000 |
| TMR3 | 0x010A | Timer3 register | | | | | | | | | | | 0x0000 |
| PR2 | 0x010C | Period register 2 | | | | | | | | | | | 0xFFFF |
| PR3 | 0x010E | Period register 3 | | | | | | | | | | | 0xFFFF |
| T2CON | 0x0110 | TON | - | TSIDL | - | TGATE | TCKPS1 | TCKPS0 | T32 | - | TCS | - | 0x0000 |
| T3CON | 0x0112 | TON | - | TSIDL | - | TGATE | TCKPS1 | TCKPS0 | - | - | TCS | - | 0x0000 |

**Table 4-2 Description of the registers of the timer2 and 3 modules**

```
TON – Timer on control bit (TON=1 starts the timer, TON=0 stops the timer)
TSIDL – Stop in IDLE mode bit (TSIDL=1 discontinue timer operation when
device enters
        IDLE mode, TSIDL=0 continue timer operation in IDLE mode)
TGATE – Timer gated time accumulation enable bit (TCS must be set to 0 when
TGATE=1)
TCKPS<1:0> – Timer input clock prescale select bits
      00 – 1:1 prescale valu
```

```
          01 – 1:8 prescale value
          10 – 1:64 precale value
          11 – 1:256 prescale value
    T32 – Timer 32-bit mode of timer4 and 5 select bit (T32=1 32-bit mode
    selected, T32=0 timer2
          and 3 modules operate in 16-bit mode)
    TCS – Timer clock source select bit
          (TCS=1 external clock from pin T1CK, TCS=0 internal clock FOSC/4)
```

**NOTE:** Reading unimplemented bits gives '0'.

**Entrée Capture**

introduction

Le module de saisie d'entrée a pour fonction de capturer la valeur courante du compteur de minuterie sur
un événement d'entrée. Ce module est principalement utilisé pour les mesures de fréquences ou périodes et les mesures d'impulsion.

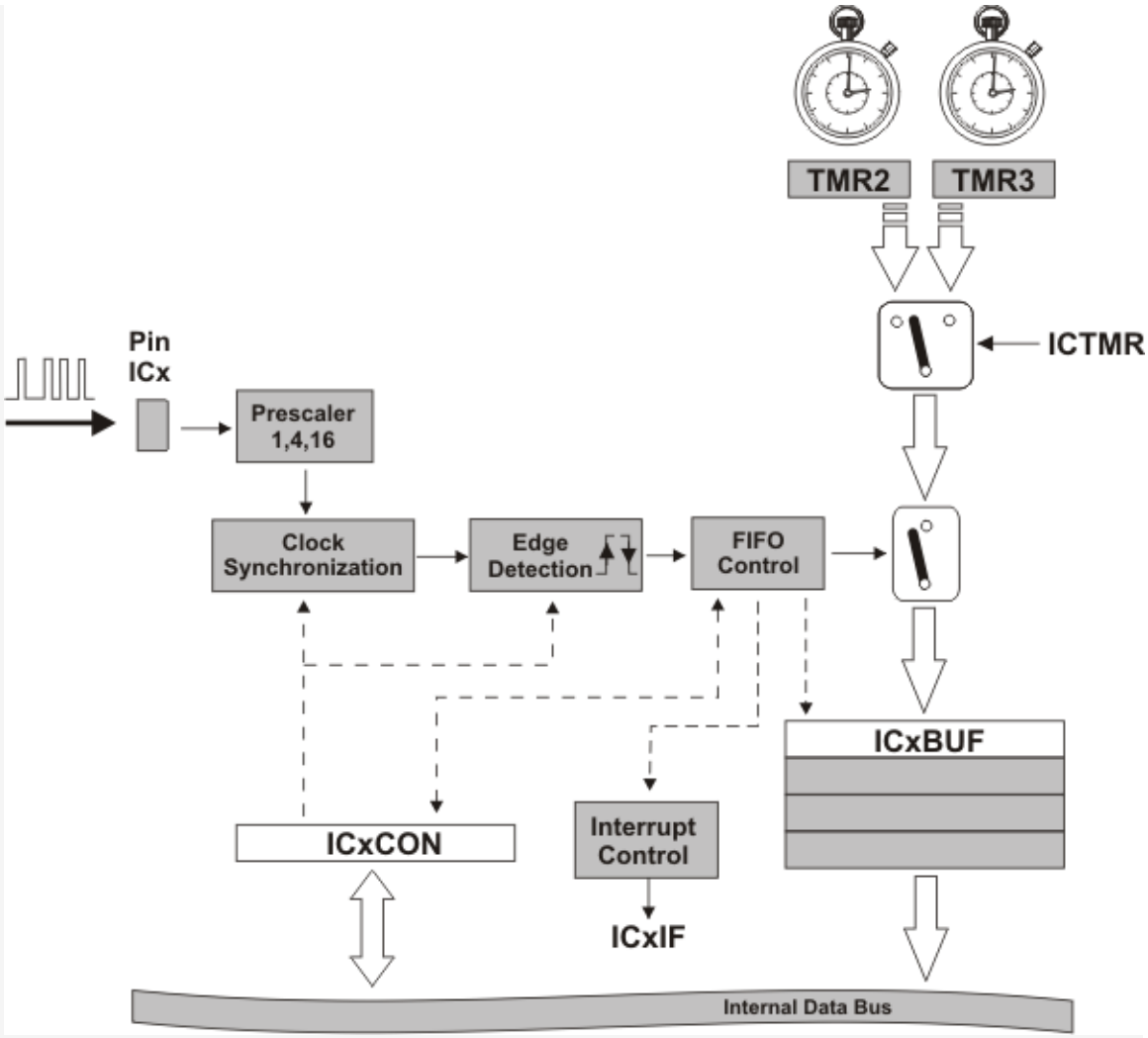
Le microcontrôleur dsPIC30F4013 contient quatre modules d'entrée de capture, alors que dsPIC30F6014A contient huit modules d'entrée de capture.

Le module d'entrée de capture a de multiples modes sélectionnables via le registre ICxCON (bit de commande ICM <2: 0>):

· Sélection par mode de signal d'entrée externe,

· Interruption par le mode de signal d'entrée externe.

Le module de saisie d'entrée contient une mémoire tampon FIFO à quatre niveaux. En réglant les bits de commande un utilisateur peut sélectionner le nombre de captures du compteur avant que le module de saisie d'entrée génère une demande d'interruption.




**Mode d'entrée de capture du signal externe**

Dans la famille des microcontrôleurs dsPIC30F la sélection par le mode de signal d'entrée externe implique la  sélection de la valeur du compteur TMR2 ou TMR3 en fonction du signal d'entrée externe à la broche ICx. La capture peut être effectuée en fonction du signal d'entrée externe:

· Sur chaque front descendant du signal d'entrée appliqué à la broche ICx,

· Sur chaque front montant du signal d'entrée appliqué à la broche ICx,

· Sur chaque front montant et chaque front descendant du signal d'entrée appliqué à la broche ICx,

· Sur chaque quatrième front montant du signal d'entrée appliqué à la broche ICx,

· Sur chaque 16 fronts montants  signal d'entrée appliqué à la broche ICx,
La sélection du mode d'entrée de capture est effectuée en réglant le bit de commande CIM <2: 0> en
Le registre ICxCON  <2: 0>. En outre, en réglant les bits de commande CIM <2: 0> le rapport de réduction dans
l'étage d'adaptation 1, 4 ou 16 est réglé.
NOTE: Le compteur registre du module de saisie d'entrée est effacé sur RESET ou commutation off.

**Mode de capture Simple**

Le mode de capture simple, ou le mode de capture pure et simple, est le mode de la prise d'entrée lorsque la capture est effectuée sur chaque front montant ou chaque front descendant du signal d'entrée extérieur à la broche d'entrée ICx. Dans ce mode, la logique du module de saisie d'entrée détecte le changement du niveau logique sur la broche d'entrée ICx, synchronise le changement de la phase de l'horloge interne, capture la valeur du compteur  TMR2 ou TMR3 et le met dans la FIFO.
Le prédiviseur fonctionne avec le rapport 1: 1, c'est à dire sans réduction.
Étant donné que le module d'entrée de capture comprend une mémoire tampon FIFO à quatre niveaux, en activant le bit de contrôle ICI <1: 0> (ICxCON <6: 5>), il est possible de sélectionner le nombre de captures avant qu'une interruption soit générée. De cette façon, la capture de signaux externes rapides est rendue possible parce que tandis que les valeurs de compteur sont capturés et mis dans le tampon FIFO, il est possible de lire les valeurs précédentes
dans la mémoire tampon et les transférer à la mémoire de données.
La sélection du compteur du module de temporisation qui doit être capturée est effectuée par réglage du bit de commande ICTMR (ICxCON <7>). Il est possible de sélectionner les compteurs 16 bits TMR2(ICTMR = 1) ou TMR3 (ICTMR = 0).
Exemple:
Cet exemple démontre le fonctionnement du module de saisie d'entrée dans la capture simple . La valeur du compteur de timer2 TMR2 est capturée sur le front descendant du signal IC1
 (broche RD8). La valeur capturée est mise au PortB.

```
Void Input1CaptureInt () {org 0x16{
LATB = IC1BUF; // Lire  valeurs capturées et mise dans PortB
IFS0.F1 = 0; // Effacer  bits IC1IF (IFS <1>)
```

```
}
void main () {
TRISB = 0; // PORTB en sortie
TPVB = 0; // Valeur initiale du PORTB
TRISD = 0x0100; // Sélectionnez la PIN IC1 (RD8) comme entrée
IPC0 = IPC0 | 0x0010; // Interruption niveau de priorité IC1IP <2: 0> =
1
IEC0 = IEC0 | 0x0002; // Interruption Module Entrée Comparer
validée
PR2 = 0xFFFF; // registre  PR2  au maximum, TIMER2 free-running
T2CON = 0x8030; // Minuterie 2 fonctionne avec diviseur 1: 256 et
horloge interne
IC1CON = 0x0082; // Configuration de l'entrée module de capture 1,
TMR2 sélectionné,
// Capture sur front descendant
while (1) asm nop; // Boucle sans fin
}
```

Au cours de la  remise à zéro de la routine d'interruption le drapeau Capture d'entrée est obligatoire et la valeur capturée est lue à partir du tampon FIFO. En fixant timer2 le préréglage du registre  PR2 est fixé à la valeur maximale, afin d'assurer le fonctionnement de la minuterie dans le mode à roue libre sur la gamme complète des valeurs, de 0 à 65535. Le module de capture d'entrée 1 est configuré pour capturer valeurs de minuterie 2 sur le front descendant du signal à la broche de IC1.

**Mode de capture Prescaler**

Dans ce mode de fonctionnement du module de saisie d'entrée du signal externe est redimensionné par le rapport 1: 4 ou 01 :16 en positionnant le bit de commande ICM <2: 0> pour les valeurs 100 ou 101 respectivement.

Ainsi, il est possible que le module de saisie d'entrée saisisse la valeur totale du compteur TMR2 ou  TMR3 4 ou 16 périodes du signal externe à la broche ICx. C'est la voie de la mesure du taux de comptage moyen en faisant la moyenne de 4 ou 16 périodes d'un signal d'entrée externe.

En réglant le bit contrôle  IC1 <1: 0> (ICxCON <6: 5>), il est également possible, comme dans le mode de capture simple, de sélectionner le nombre de captures après quoi une demande d'interruption est générée.

La sélection du module de temporisation qui doit être échantillonnée et se fait en positionnant le bit de commande
ICTMR (ICxCON <7>).
NOTE: Si la base de temps est incrémentée par chaque cycle d'instruction, alors le résultat de la capture sera disponible dans le tampon FIFO un ou deux cycles après que l'instruction synchrone change à la broche d'entrée ICx, en phase avec l'horloge interne du microcontrôleur. Un exemple de réglage de la valeur capturée retardée par un ou deux cycles d'instructions est représenté sur TCY

Attention!
Avant que le mode de fonctionnement soit modifié, le module d'entrée de capture doit être éteint, à savoir le bit de commande ICM <2: 0> effacé . Si le mode est changé sans effacer ICM <2: 0>, il ya un contenu résiduel dans le compteur diviseur qui conduit à l'échantillonnage et interrompt prématurément la génération de demande.

Exemple:
Cet exemple démontre le fonctionnement du module de saisie d'entrée dans le mode capture prescaler.
. L'exemple montre la saisie des valeurs du compteur timer2 TMR2 sur chaque quatrième front montant du signal IC1 (broche RD8). La valeur capturée est mise au PortB.

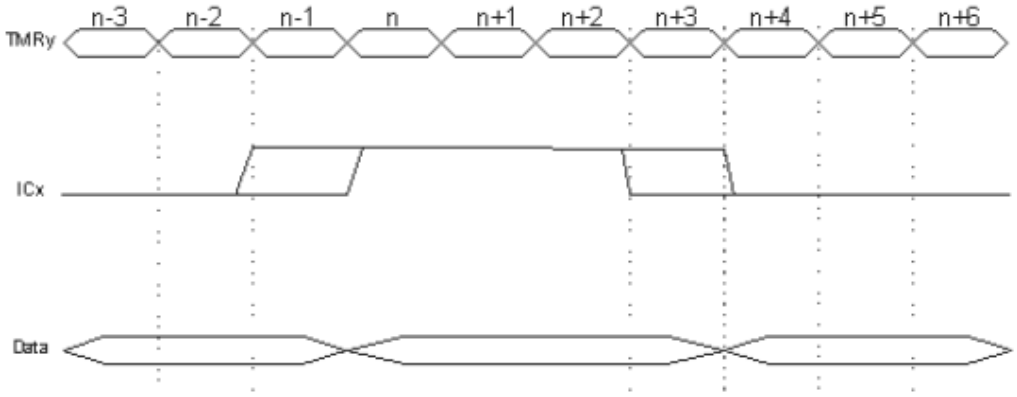
```
Void Input1CaptureInt () {org 0x16
TPVB = IC1BUF; // Lire valeur capturée et mise au PORTB
IFS0.F1 = 0; // Effacer bits IC1IF (IFS <1>)
}
void main () {
TRISB = 0; //  PORTB iz en sortie
TPVB = 0; // Valeur initiale PORTB
TRISD = 0x0100; // Sélectionnez la PIN IC1 (RD8) comme entrée
IPC0 = IPC0 | 0x0010; // Niveau de priorité d'interruption
IEC0 = IEC0 | 0x0002; // Interruption module de capture d'entrée validée
PR2 = 0xFFFF; // registre PR2  au maximum, TIMER2 free-
```

running
T2CON = 0x8030; // Minuterie 2 fonctionne avec diviseur 1: 256 et horloge interne
IC1CON = 0x0084; // Configuration de l'entrée module de capture 1,
TMR2 sélectionnée,
// Capture sur chaque 4 fronts montants
while (1) asm nop; // Boucle sans fin
}
Au cours de la compensation de la routine d'interruption de demande d'interruption le module drapeau Capture d'entrée est obligatoire et la valeur capturée est lue à partir du tampon FIFO. En fixant timer2 la présélection du registre PR2 est fixée à la valeur maximale pour assurer le fonctionnement de la minuterie dans le libre mode de fonctionnement sur toute la plage de valeurs, de 0 à 65535. Entrée Module 1 est configuré pour capturer les valeurs de minuterie 2 sur chaque quatrième bord montant du signal à la broche IC1.

**Mode de détection de front**

Capturer la valeur du compteur TMR2 ou TMR3 peut être fait sur chaque montée et chaque descente du front du signal d'entrée externe appliqué à la broche ICx. Le mode de détection de bord est choisi en mettant les bits de commande ICM <2: 0> (ICxCON <2: 0>) à 001. Dans ce mode, le diviseur compteur ne peut pas être utilisé. Le module de requête d'interruption de saisie d'entrée est généré sur chaque augmentation et chaque front descendant (ICxIF bit est mis). Il pas possible de générer une interruption au bout de 2, 3 ou 4 captures en fixant les bits de commande ICI <1: 0> (ICxCON <6: 5>) parce que dans ce mode, ils sont ignorés. Chaque événement de capture génère une interruption. Une conséquence de pas de débordement de la mémoire tampon FIFO est possible.
NOTE: Si la base de temps est incrémentée par chaque

cycle d'instruction, alors le résultat de la capture sera disponible dans le tampon FIFO un ou deux cycles après que l'instruction synchrone change sur la broche d'entrée ICx, en phase avec l'horloge interne du microcontrôleur. Un exemple de réglage de la valeur capturée retardée par un ou deux cycles d'instructions est représenté sur la TCY




Exemple de réglage de la valeur capturée retardé par 1 ou 2 cycles d'instructions TCY

**Lecture des données à partir du tampon FIFO**

- Chaque module d'entrée de capture comprend un niveau quatre (16 bits) de mémoire tampon FIFO pour le logement des captures. L'accès aux captures dans le tampon FIFO est effectué via le registre ICxBUF. En outre, il existe deux indicateurs d'état ICBNE (ICxCON <3>) et Icov (ICxCON <4>) définissant l'état du tampon FIFO. Le drapeau d'état ICBNE indique que le FIFO n'est pas vide. Ce drapeau est effacé par le matériel lorsque le dernier mot est lu dans la mémoire tampon FIFO ou au cours de la remise à zéro du module d'entrée de capture par réglage des bits de contrôle ICM <2: 0> à la valeur 000. ICBNE est également remis à 0 lors de RESET.

L'autre état Icov du drapeau désigne l'état de débordement de la mémoire tampon FIFO, c'est à dire quand, après quatre capture qui n'ont pas été transférées à la mémoire de données de capture et qu'une cinquième est en cours .

Aucune demande d'interruption est générée puis, le bit Icov est positionné et les valeurs des cinq captures et toutes les captures suivantes sont ignorées. La compensation de ce bit est fait par le matériel à la lecture de toutes les quatre captures de la mémoire tampon FIFO, ou en réinitialisant de le module capture d'entrée. En outre, le RESET du microcontrôleur efface ce drapeau.

**Mode d'interruption de signal externe**

Les broches d'entrée du module d'entrée de capture peuvent être utilisées comme sources supplémentaires d'interruption externes si le module de saisie d'entrée est configuré pour fonctionner dans le mode d'interruption de signal externe.

Ce qui s'accompli lorsque les bits de configuration ICM <2: 0> sont fixés à 111. Puis, les broches d'entrée ICx sur le front montant de générent une demande ICxIF interruption. Si le bit ICxIE interruption activer est réglé et le niveau de priorité d'interruption ICxIP <2: 0> est défini, le microcontrôleur entre une interruption.

REMARQUE: Pour fonctionner dans ce mode, il est nécessaire d'effacer les bits de contrôle ICI <1: 0>(ICxCON <6: 5> = 0). Ensuite, l'interruption est générée indépendamment de la lecture de la mémoire FIFO. En outre, le bit statut Icov doit être effacé.

Le module d'entrée de capture est très souvent utilisé par le module UART pour la détection automatique de la vitesse de transmission. Le mode de détection automatique de débit du module UART est configuré comme suit: le bit de commande ABAUD (UxMODE <5>) est réglé pour dire que l'axe UART RX est connecté en interne à l'entrée capture entrée du module. La broche ICx est déconnectée au reste du module capture d'entrée. La vitesse de transmission est mesurée en mesurant la largeur du bit de démarrage lorsqu'un Caractère NULL est reçu. Il faut veiller à ce que le module de saisie d'entrée soit configuré pour le mode de détection de bord. Pour chaque microcontrôleur de la famille dsPIC30F l'affectation du module de capture pour chaque UART a été défini.

**Opération de capture d'entrée en mode SLEEP**

- en mode veille, l'horloge système est désactivée,

c'est à dire le module de saisie d'entrée ne peut fonctionner en tant que source d'interruption externe. Ce mode est activé en réglant les bits de contrôle ICM <2: 0> à 111. Un front montant sur la broche

d'entrée ICx générera une interruption du module de capture d'entrée. Si l'alarme est activée pour cette broche d'entrée, le microcontrôleur va se réveiller de son sommeil.
Dans le cas où le module de saisie d'entrée a été configuré pour un mode autre que ICM <2: 0> = 111  le microcontrôleur passe en mode SLEEP, aucun signal externe, montant ou descendant, peut générer un état de réveil du sommeil.

**opération  d'entrée de capture en mode IDLE**

- Fonctionnement du module d'entrée de capture en mode IDLE est spécifié par les bits de commande  ICSIDL  (ICxCON <13>). Si ICSIDL = 0, le module continue à fonctionner en mode IDLE avec toutes les fonctionnalités dans tous les modes mentionnés ci-dessus. Le diviseur est entièrement fonctionnel dans ce mode.

Si ICSIDL = 1, le module s'arrête en mode IDLE. Puis, le module de saisie d'entrée peut fonctionner seulement dans le mode d'interruption de signal externe, c'est à dire les bits de commande CIM <2: 0> = 111. Un front montant sur la broche d'entrée ICx va générer une interruption du module de capture d'entrée. Si l'alarme est activée pour cette broche d'entrée, le microcontrôleur va réveiller à partir de l'état IDLE.
Dans le cas où le module de saisie d'entrée a été configuré pour un mode différent le microcontrôleur passe en mode de veille lorsque le bit de contrôle ICSIDL est activé, aucun signal externe, montant ou descendant , et peut générer une condition de réveil de l'état IDLE.
Attention!
Lorsque le module de saisie d'entrée est activé, un logiciel d'utilisateur pour permettre que la broche d'entrée ICx est réalisé sous la broche d'entrée en fixant le bit de TRIS associé. En validant le module  capture d'entrée  la broche  ICx n'est pas configurée automatiquement. En outre, tous les autres périphériques utilisant cette broche sont désactivés.

| name | ADR | 15 | 14 | 13 | 12-8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Reset State |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IC1BUF | 0x0140 | Input 1 Capture Buffer Register | | | | | | | | | | | | 0xuuuu |
| IC1CON | 0x0142 | - | - | ICSIDL | - | ICTMR | ICI<1:0> | | ICOV | ICBNE | ICM<2:0> | | | 0x0000 |
| IC2BUF | 0x0144 | Input 2 Capture Buffer Register | | | | | | | | | | | | 0xuuuu |
| IC2CON | 0x0146 | - | - | ICSIDL | - | ICTMR | ICI<1:0> | | ICOV | ICBNE | ICM<2:0> | | | 0x0000 |
| IC7BUF | 0x0158 | Input 7 Capture Buffer Register | | | | | | | | | | | | 0xuuuu |
| IC7CON | 0x015A | - | - | ICSIDL | - | ICTMR | ICI<1:0> | | ICOV | ICBNE | ICM<2:0> | | | 0x0000 |
| IC8BUF | 0x015C | Input 8 Capture Buffer Register | | | | | | | | | | | | 0xuuuu |
| IC8CON | 0x015E | - | - | ICSIDL | - | ICTMR | ICI<1:0> | | ICOV | ICBNE | ICM<2:0> | | | 0x0000 |

```
ICSIDL – Input captur module stop in IDLE control bit
        (ICSIDL=0 input capture module will continue to operate in IDLE
mode,
        ICSIDL=1 input capture module will halt in IDLE mode)

ICTMR – Input capture timer select bits (ICTMR=0 TMR3 contents are captured
on capture event,
        ICTMR=1 TMR2 contents are captured on capture event)

ICI <1:0> - Select number of captures per interrupt bits
      00 – interrupt on every capture event
      01 – interrupt on every second capture event
      10 – interrupt on every third capture event
      11 – interrupt on every fourth capture event

ICOV – FIFO buffer overflow status flag (read only) bit

ICBNE – FIFO buffer buffer empty status (read only) bit
        (ICBNE=0 FIFO buffer empty, ICBNE=1 FIFO buffer contains at least
one capture value

ICM <2:0> - Input capture mode select bits
      000 – Input capture module turned off
      001 – Capture mode, every edge (rising or falling)
      010 – Capture mode, every falling edge
      011 – Capture mode, every rising edge
      100 – Capture mode, every 4th rising edge
      101 – Capture mode, every 16th rising edge
      110 – Unused (module disabled)
      111 – Input capture module in external signal interrupt mode
      (external source of interrupt requests)
```

## Module OUTPUT COMPARE

Introduction

Le module de comparaison de sortie a pour fonction de comparer la valeur du compteur de base de temps avec la valeur d'un ou de deux registres de comparaison en fonction du mode de fonctionnement sélectionné. Il est capable de générer une impulsion de sortie unique ou une séquence d'impulsions de sortie lorsque le rapport des valeurs correspondent; En outre, il a la capacité de générer des interruptions sur des événements de correspondance comparés.

Le contrôleur  dsPIC30F4013 a 4modules de sortie de comparaison alors que le contrôleur dsPIC6014A en a 8. Chaque canal peut sélectionner des compteurs de base de temps, ou TMR2 TMR3, sera comparé avec les registres compare. Le compteur est sélectionné en utilisant le bit de contrôle OCTSEL (OCxCON <3>).

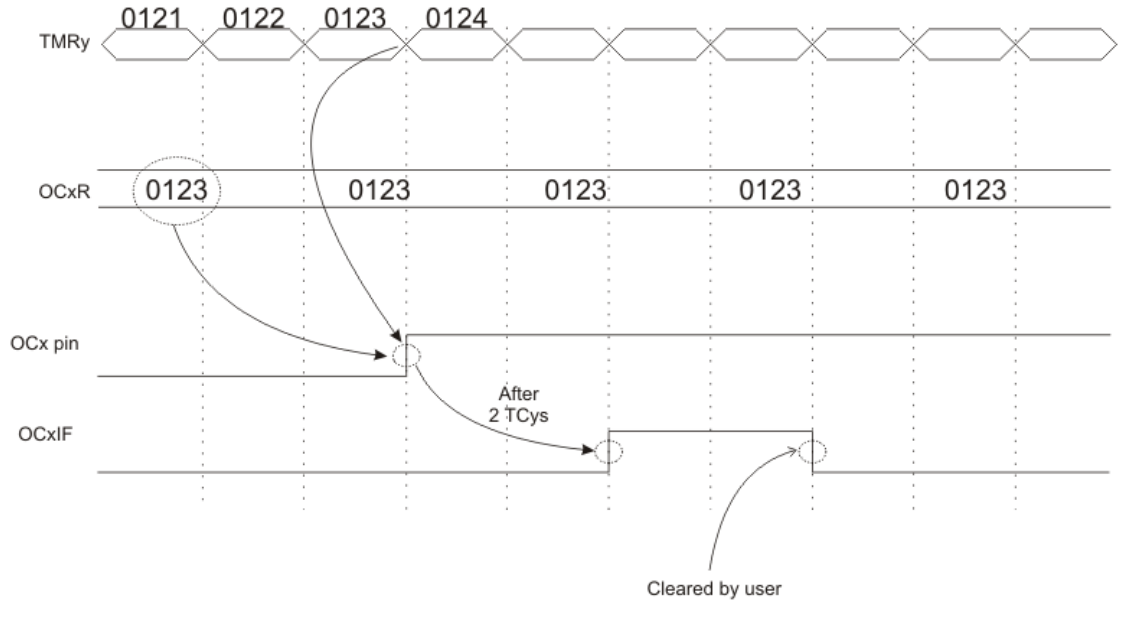
La sortie  module compare possède plusieurs modes de fonctionnement sélectionnables en utilisant les bits de commande OCM <2: 0> (ocXcon <2: 0>):

· Simple mode compare ,

· Double mode compare soit générer une impulsion de sortie ou une séquence d'impulsions de sortie,

· Mode Pulse Width Modulation (PWM).

NOTE: Il est conseillé, avant de passer à un nouveau mode , d'éteindre la sortie compare en remettant à zéro le bit OCM <2: 0>.

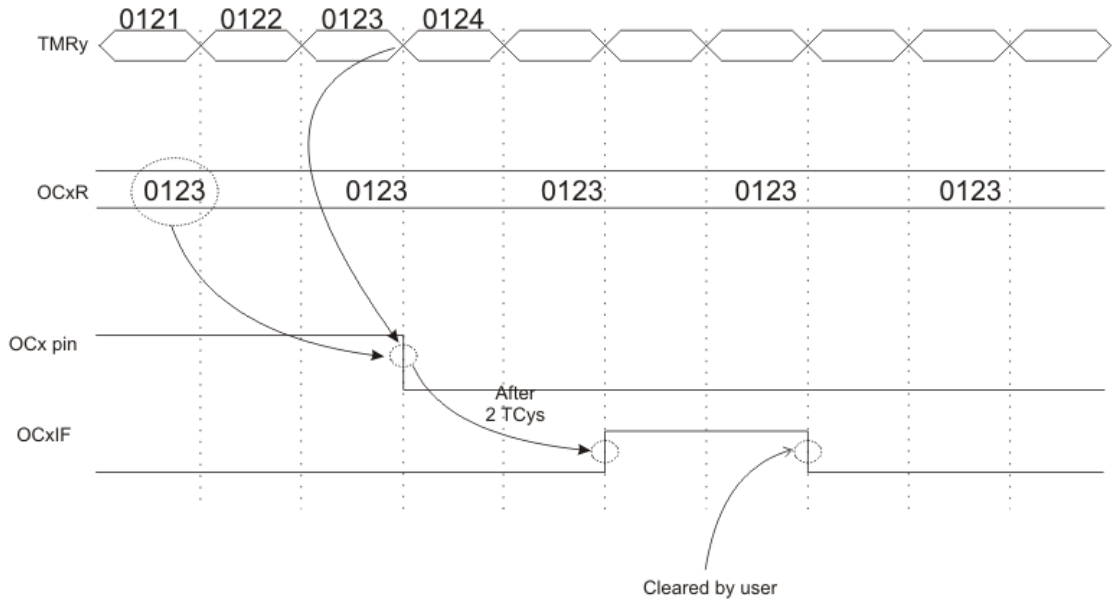


**Simple compare match broche OCx à l'état bas**

Pour configurer la sortie compare dans ce mode, les bits de contrôle OCM <2: 0> sont mis à 010. En outre, le compteur de base de temps (TMR2 ou TMR3) doit être activé. Dans un premier temps, la sortie broches OCx est haute et le reste jusqu'à ce qu'un événement de correspondance se produise entre les registres  TMRy et OCxR.

Une instruction d'horloge  après l'événement  broche OCx est mise  à 0  et le reste  jusqu'à ce qu'un changement du mode ou que le module soit désactivé. TMRy continue le comptage.

Deux instructions d'horloges après que la broches OCx soit à 0 , le drapeau d'interruption OCxIF est généré.




REMARQUE: La broche OCx est à 0 sur une réinitialisation de l'appareil. Dans le mode de comparaison unique, le basculement de l'état de fonctionnement de la broche OCx peut être réglé par le logiciel de l'utilisateur.

Exemple:

Sortie compare du module 1 est dans le mode de comparaison simple: basculer la sortie de courant de la broche OC1.

La sortie de comparaison du module compare les valeurs des registres  OC1R et TMR2 ; sur l'égalité,la sortie de la broche OC1 est basculée

```
Void Output1CompareInt void () {// org 0x18 adresse OC1 dans la
table du vecteur d'interruption
IFS0.F2 = 0; // Efface l'indicateur d'interruption
}
void main () {
TRISD = 0; // OC1 (RD0) en sortie
IPC0 = IPC0 | 0x0100; // Niveau de OC1IP interruption de priorité <2:
0> = 1
IEC0 = IEC0 | 0x0004; // Sortie de comparaison permet une
interruption
OC1R = 10000; // OCR = TMR2 instant de changement de niveau au
OC1
PR2 = 0xFFFF; // PR2 valeur maximale, la base de temps 2 free-
running
T2CON = 0x8030; La base // Time 2 fonctionne avec diviseur 1: 256
et horloge interne
OC1CON = 0x0003; // Sortie compare une configuration du module,
TMR2 sélectionné
// Simple mode Compare, bascule broches OC1
while (1) asm nop; // Boucle sans fin
}
```

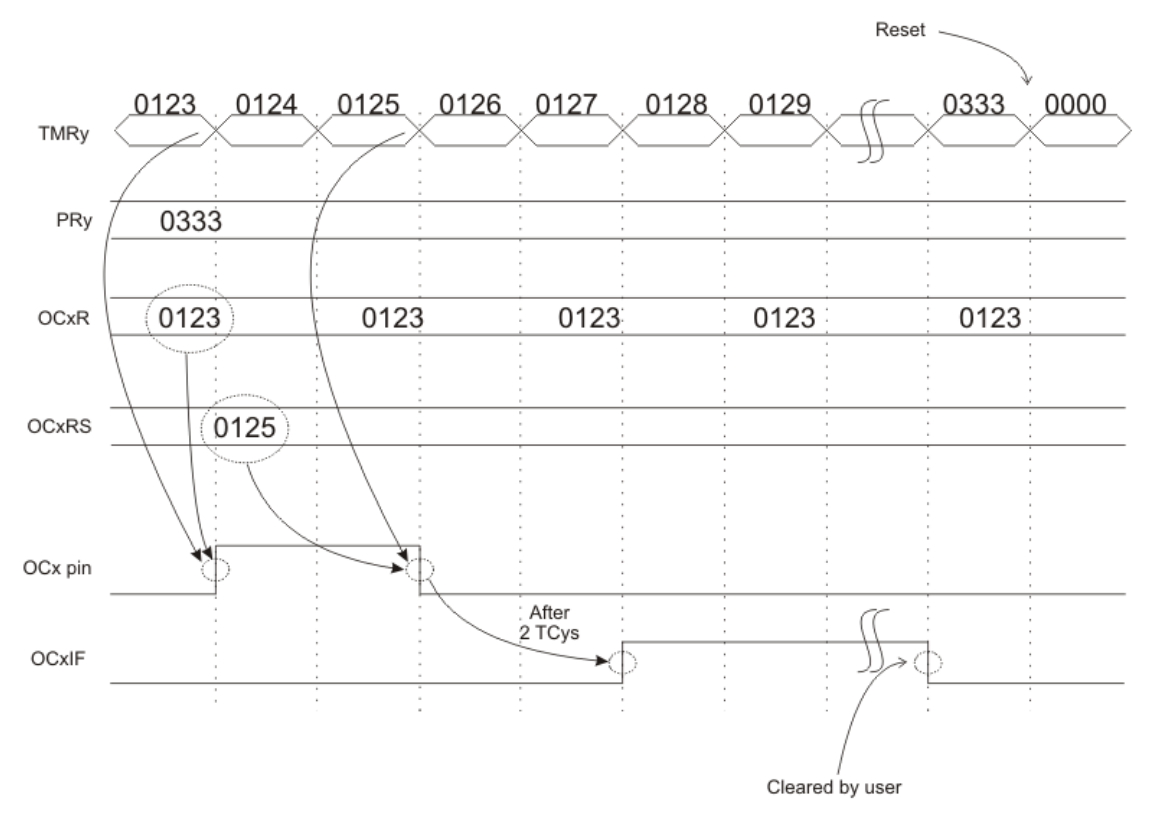
Dans la routine d'interruption de la demande pour le drapeau sortie compare le module d'interruption est réinitialisé. À
la mise en base de temps 2,  registre PR2 préréglé est réglé sur la valeur maximale pour permettre le libre mode de fonctionnement sur toute la plage, 0-65335. La valeur de OC1R définit le temps du changement d'état de la broche OC1, c'est à dire du cycle. Le module compare de sortie est configuré pour changer l'état de la broche sur OC1 simple compare correspondant avec la valeur de OC1R.

**Double compare**

Lorsque les bits de contrôle OCM <2: 0> sont réglés sur 100 ou 101, la sortie compare est configurée pour le double mode compare. Dans ce mode, le module utilise deux registres, OCxR et OCxRS, pour les événements de compare.

Les valeurs des deux registres sont comparées avec le compteur de base de temps TMR2 ou TMR3. Sur un événement compare du registre OCxR et registre TMR2 ou TMR3 (sélectionnable par
bit de commande de OCTSEL), le bord d'attaque de l'impulsion est générée à la broche OCx; le registre OCxRS est ensuite comparé

avec le même registre de base de temps et sur un événement de compare,le bord de fuite à la broche OCX est généré.
En fonction de la valeur du bit de contrôle OCM <2: 0> à la broche de sortie OCX est généré:
· Impulsion unique et une demande d'interruption,
· Une séquence d'impulsions et une demande d'interruption.
Double compare, simple impulsion de sortie à la broche OCx
Lorsque les bits de contrôle OCM <2: 0> sont mis à 100, la sortie compare est configurée pour le compare double (registres OCxR et OCxRS ), le mode d'impulsion de sortie unique. En réglant le bit OCTSEL de commande du compteur de base de temps la comparaison est sélectionnée.
Deux instructions horloges après la broche OCx est à 0, une demande d'interruption OCxIF pour la sortie compare est générée. La broche OCx restera à 0 jusqu'à ce qu'un changement de mode ait été fait ou le module inhibé. Si dans le contenu de base de temps du registre PRy <OCxRS, aucun bord de fuite et demande d'interruption sont générés et si dans PRy <OCxR aucun bord d'attaque est généré à la broche OCx.



Exemple:
Le module de comparaison de sortie compare la valeur des registres OC1R et  OC1RS avec la valeur du compteur TMR2 de la base de temps 2 et en cas de correspondance bascule au niveau logique à la broche OC1.

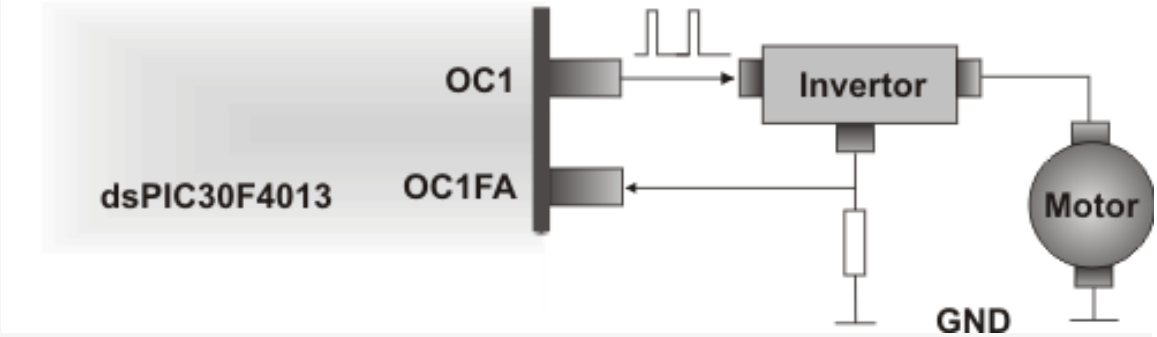
```
void Output1Compare Int () {// org 0x18 Adresse du OC1 dans le tableau interruption
IFS0 = 0; // La réinitialisation drapeau d'interruption module de OC1
}
void main () {
TRISD = IPC0 | 0x0100; // OC1 (RD0) en sortie
IPC0 = IEC0 | 0x0004; // Sortie Compare module 1 activation d'interruption
OC1R = 30000; // Si OC1R = TMR2,  à la broche OC1
OC1RS = 100; // Si OC1RS = TMR2, au bord de fuite OC1
PR2 = 0xFFFF; // PR2 au maximum, base de temps 2 free-running
T2CON = 0x8030; // Time base 2 fonctionne avec diviseur 1: 256 et horloge interne
OC1CON = 0x0005; // Configuration de la sortie Compare module 1,
// Sélectionné séquence TMR2, double comparer correspondance, pulse
while (1) asm nop; // Boucle sans fin
}
```

Dans la routine d'interruption l'indicateur de demande d'interruption de la sortie de comparaison module est réinitialisé.
Dans le préréglage minuterie 2, inscrivez-PR2 est réglé sur la valeur maximale afin de permettre free-running
mode de la minuterie dans toute la gamme des valeurs de 0 à 65535. La valeur de OC1R définit le bord instantané d'attaque à la broche OC1, la valeur de OC1RS définit l'instant du bord de fuite
. Le module de sortie comparateur est configuré pour activer continuellement le niveau logique sur la broche
OC1 sur le double événement compare  avec les valeurs des registres OC1R et OC1RS.

**Le Pulse Width Modulation (PWM) mode**

Lorsque les bits de contrôle OCM <2: 0> sont réglés sur les valeurs 110 ou 111, la sortie module compare  est configurée pour le mode de modulation de largeur d'impulsion (PWM). Le mode PWM est

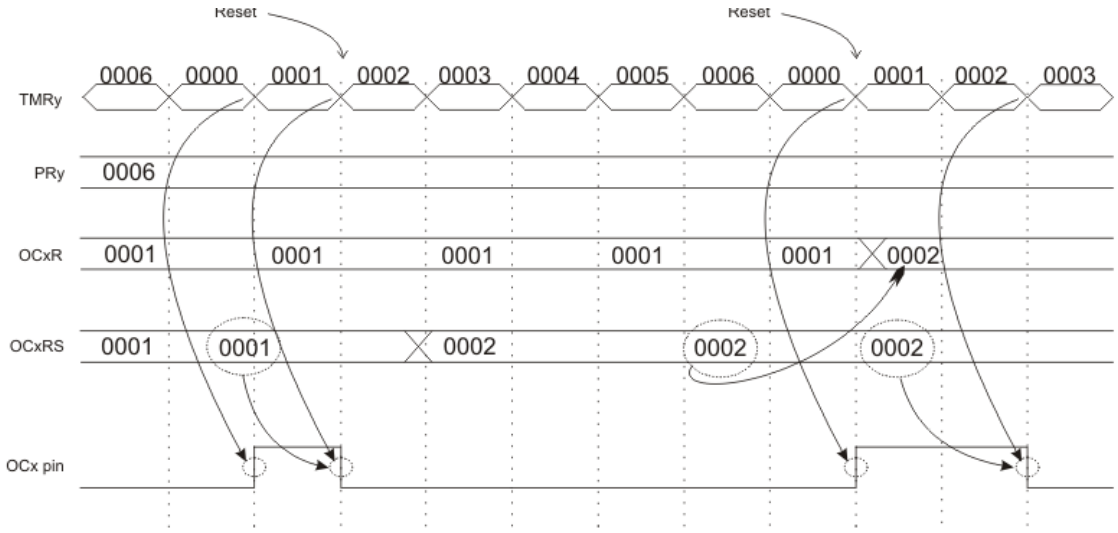
disponible sans intervention de la protection de défaut ou avec une entrée de protection contre les défauts. Pour le deuxième mode de la broche d'entrée  PWM OCxFA ou OCxFB est utilisée.
dsPIC30F4013 connexion à l'onduleur incluant le signal d'erreur de rétroaction. L'onduleur commande le fonctionnement du moteur M
Figue. 6-8 dsPIC30F4013 connexion à l'onduleur incluant le signal d'erreur de contre-réaction




**mode PWM sans apport de protection contre les défauts**
Lorsque les bits de contrôle OCM <2: 0> sont fixés à 110, la sortie comparer module commande dans ce Mode. En mode PWM le registre OCxR est un unique esclave registre de rapport cyclique lire. OCxRS est un registre tampon écrit par l'utilisateur de mettre à jour le PWM cycle.Chaque minuterie à le registre  période événement  (fin de période PWM), le registre de cycle de service, OCxR, est chargé avec le contenu de OCxRS. Le drapeau interruption  est affirmé à la fin de chaque période de PWM.
Lors de la configuration du module pour comparer la sortie en mode de fonctionnement PWM, ce qui suit des mesures devraient être prises:
1. Définissez la période PWM en écrivant sur le registre de période de temporisation, PRy.
2. Définir le cycle de service PWM en écrivant aux registres OCxRS .
3. Ecrire le registre OCxR  avec le cycle de service initial.
4. Activez les interruptions pour la minuterie sélectionnée.
5. Configurer la sortie module compare  pour l'un des deux modes de fonctionnement PWM par écriture 100 pour contrôler les bits OCM <2: 0> (OCxCON <2: 0>).
6. Réglez la valeur de pré-échelonnage TMRy et permettent la base de temps sélectionnée.

NOTE: Le registre de OCxR devrait devenir un seul devoir registre de cycle de lecture avant que la sortie
comparer module est d'abord activé (OCM <2: 0> = 000).**MODE PWM AVEC LA BROCHE D'ENTRÉE DE PROTECTION CONTRE LES DÉFAUTS**
Lorsque la commande Bits OCM <2: 0> est à 111, le module output est configuré pour comparer le fonctionnement de PWMmode .
on utilise le signal provenant de la broche d'entrée OCxFA pour comparer les sorties des canaux 1 à 4 ou de la broche OCxFB pour la sortie compare INPIT canaux 5 à 8. Le signal à la broche d'entrée OCxFA ou OCxFB est une erreur de rétroaction de l'inverseur liés à un état dangereux de fonctionnement possible de l'onduleur. Si la broche d'entrée OCxFA ou OCxFB est à 0, l'onduleur est considéré être dans un (erreur) état dangereux.
Ensuite, la broche de sortie OCx le module de comparaison de sortie fonctionne dans le mode PWM est désactivé automatiquement et la broche est entraînée à l'état haute impédance. L'utilisateur peut choisir de fournir un pull-down ou pull-up afin de définir l'état de la broche OCx qui est en 3 états et déconnectée du reste de la comparaison du module sortie . Dans l'état de l'inverseur
faute, lors de la détection de la condition de défaut et désactivation de la broche OCx, le drapeau interruption respective est affirmé et dans le registre OCxCON le bit de OCFLT (OCxCON <4>) est réglé. Se il est activé,
une interruption du module de sortie de comparaison est générée.
REMARQUE: Les repères de défauts externes, OCxFA ou OCxFB, tandis que le module de comparaison de sortie
fonctionne en mode PWM à la broche d'entrée de protection contre les défauts, va continuer à protéger le module
alors qu'il est en veille ou en mode IDLE.
**calcule période et le cycle PWM**
période PWM
La période de PWM, spécifiée par la valeur dans le registre de PRy de la minuterie sélectionné y, est calculée par:
TPWM = (PRy 1) TCY (valeur de pré-échelonnage TMRy),
et la frequencyby PWM:
fPWM = 1 / TPWM.
Exemple:
Calcul de la période PWM pour un microcontrôleur comportant une horloge de 10 MHz avec x4 PLL,

fréquence d'horloge de l'appareil est 40MHz. La fréquence d'horloge d'instruction est FCY = FOSC / 4, soit 10MHz.
Réglage de minuterie 2 est prescale 4. Calculer la période de PWM pour la valeur maximale
PR2 = 0xFFFF = 65535.
TPWM = (65 535 + 1) x 0.1µs x (4) = 26,21 ms, soit fPWM = 1 / TPWM = 38,14 Hz.
Cyclique 6.4.2 PWM
Le cycle d'utilisation PWM est spécifié par la valeur écrite dans les OCxRS de registre. Il peut être écrit à tout moment à l'intérieur du cycle PWM, mais la valeur de cycle de service est mémorisée dans OCxR lorsque la période PWM est terminée. OCxR est un registre de lecture seulement. Cela fournit un double buffering pour
le rapport cyclique PWM.
Si le registre de cycle de service, OCxR, est chargé avec 0000, le cycle de travail est nulle et la broche OCX reste à 0  tout au long de la période PWM.
Si le registre de cycle de travail est supérieure à PRy, la  broche OCx de sortie restera à 1  tout au long de la période de PWM (rapport cyclique de 100%).
Si OCxR est égal à PRy, la broche OCx sera à 1  dans le premier cycle PWM et 0 dans le cycle PWM ultérieure.
NOTE: Afin d'atteindre le contrôle de PWM, il est nécessaire de permettre l’ajustement de cycle de service. Ceci est accompli en ajustant la valeur de pré-échelonnage,
La valeur d'horloge et la fréquence de PWM pour atteindre la plus grande valeur possible de réaliser ainsi PRy
le plus grand nombre de niveaux de réglage, ce est à dire la plus haute résolution.




**Fonctionnement de la sortie compare en mode SLEEP**

Lorsque l'appareil passe en mode veille, l'horloge système est désactivée. Pendant le sommeil, l'état de la broche de sortie OCx se tient au même niveau qu'avant d'entrer SLEEP. Par exemple, si la broche de sortie OCx était à 1 et le CPU dans l'état SLEEP, la broche restera à 1 jusqu'à ce que le microcontrôleur se réveille.

**Fonctionnement de la sortie compare en mode SLEEP**

Lorsque l'appareil passe en mode IDLE, le bit de commande de OCSIDL (OCxCON <13>) sélectionne si la sortie du module sera comparée, arrêtée ou continuera à fonctionner en mode IDLE.

Si OCSIDL = 0, le module continue de fonctionner seulement si la base de temps sélectionnée est réglée sur fonctionner en mode IDLE (TSIDL = 0).

Si OCSIDL = 1, le module mettra fin à l'opération en mode IDLE comme il le fait pour le mode sommeil.

REMARQUE: Les broches d'entrée OCxFA et OCxFB pendant mode veille ,continuera à contrôler la broche de sortie associée OCx,si elles sont activées , déconnecter la broche de sortie OCx si elles sont à 0.