Projet 2006/2007

V.A.E.
Vélo à Assistance Électrique

Programme SVM pour moteur BLDC équipé de capteurs à effet Hall

Implanté dans un dsPIC30F2010
Version V2 MPLAB C30
(incrément de phase)

Note : le principe de la modulation SVM (Space Vector Modulation) est décrit dans le document "Générateur PWM SVM triphasé pour moteur BLDC- Schémas et descriptions pour EPLD".

Sommaire

1. Schéma fonctionnel (figure 1)

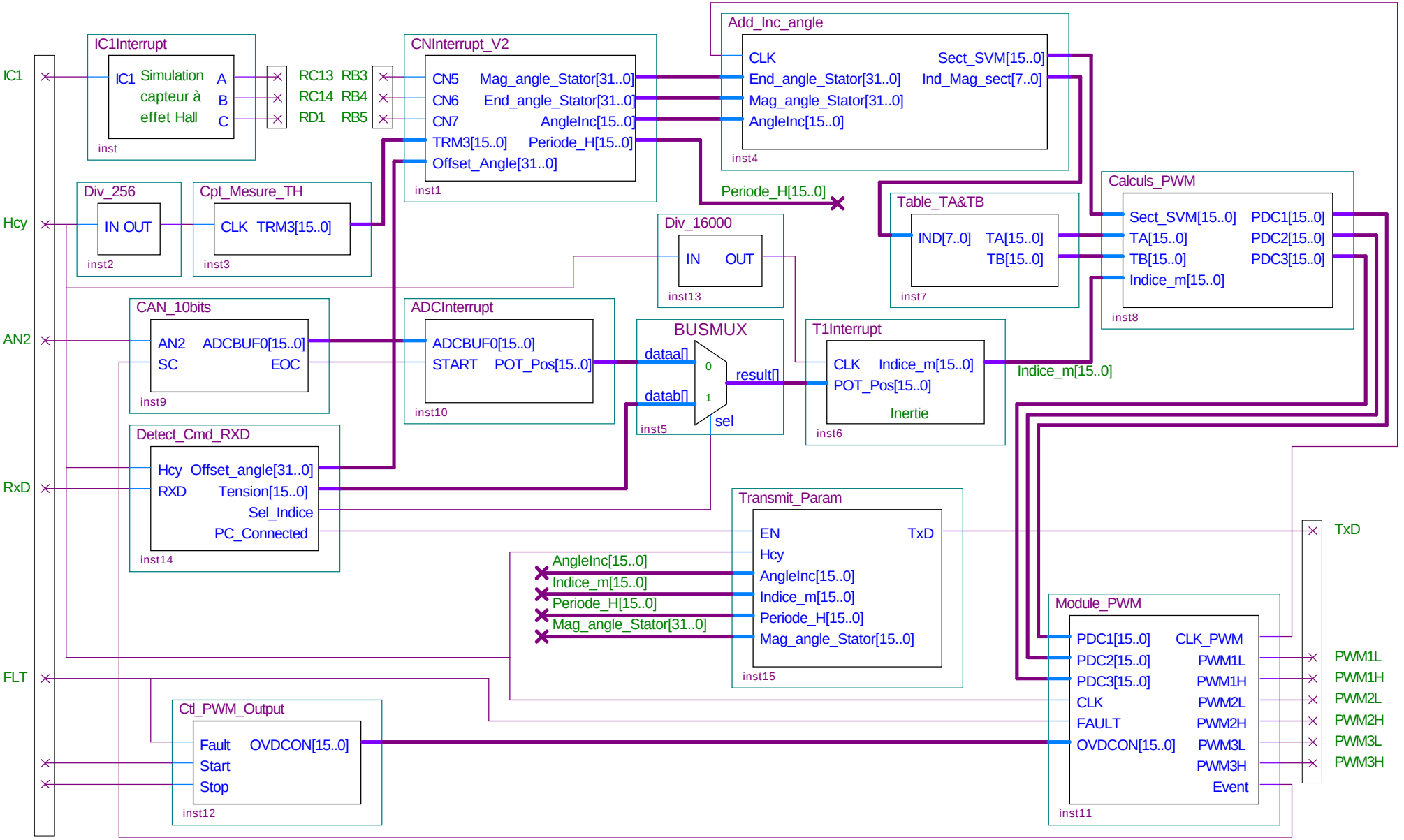
On exploite au mieux les structures internes du dsPIC30F2010 pour soulager le CPU :

- le module PWM qui produit les 6 signaux de commande des 3 bras PWM
- un timer pour mesurer la durée d'un secteur magnétique
- deux autres timer pour produire les signaux SVM synchrones des capteurs à effet Hall
- le CAN interne pour régler l'amplitude des tensions d'alimentation du moteur (dans cette version)

1.1 Principe

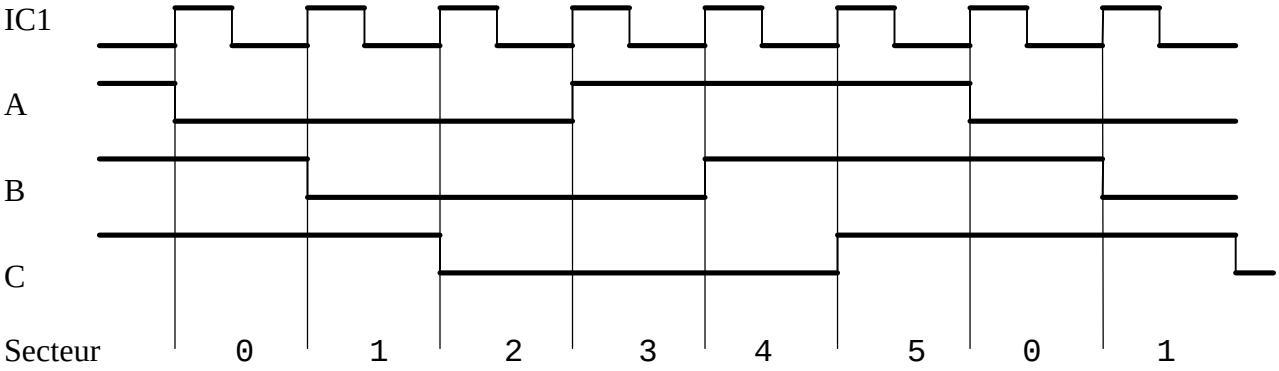
Aucun asservissement n'est réalisé dans cette version (V2).
Un potentiomètre agit directement sur l'amplitude des tensions d'alimentation du moteur. Ses capteurs à effet Hall indiquent à tout instant la position angulaire du moteur (résolution = 60° magnétiques). La phase relative des tensions d'alimentation triphasées est maintenue constante (avec un offset réglable) quelle que soit la vitesse du moteur de sorte à obtenir un couple maximum (déphasage de 90° entre le champ magnétique produit par les tensions d'alim triphasées et la position magnétique du rotor aimanté).

Figure 1 : schéma fonctionnel de degré 1



1.2 IC1Interrupt

La fonction permet de simuler les signaux produits par les capteurs à effet Hall en simulation logicielle ou si on ne dispose pas du moteur.
Cette fonction est réalisée par un programme d'interruption déclenché au flanc montant de IC1.
Elle produit 3 signaux A, B et C conformes au chronogramme suivant :



Chaque combinaison des signaux A, B et C identifie l'un des 6 secteurs magnétiques de 60° (voir étude SVM).

1.3 Div_256 et Cpt_Mesure_TH

Ces 2 fonctions sont réalisées par des structures matérielles intégrées dans le µC :

- Div_256 : division de fréquence par 256
- Cpt_Mesure_TH : compteur 16 bits en binaire naturel

L'état TRM3 du compteur est utilisé par la fonction CNInterrupt pour mesurer l'intervalle de temps qui sépare 2 changements d'état des capteurs à effet Hall (c'est à dire la durée d'un secteur magnétique).
La fréquence d'horloge Hcy vaut 14,74MHz (avec un quartz de 7,37MHz) :

- la résolution de mesure de temps vaut 256/(14,74MHz)=17,36µS
- le plus long intervalle de temps mesurable vaut : 65536*17,36µS=1,14S

Avec un moteur à 20 paires de pôles, une mesure de "100" (au moins 2 chiffres significatifs) correspond à 575 tr/mn et une mesure de "65535" à 0,88tr/mn.
Cette mesure sera utilisée pour produire un champ magnétique statorique qui suive au mieux la position du rotor.

1.4 CNInterrupt

Cette fonction synchronise le champ statorique avec la position du rotor aimanté du moteur.
Les capteurs à effet Hall donnent la position du rotor avec une résolution de 60° magnétiques. Cette caractéristique est médiocre, mais les capteurs sont posés avec soins de sorte que leurs changements d'états déterminent avec précision la position du rotor à cet instant là.
La fonction "CNInterrupt" est réalisée par un programme d'interruption activé au changement d'état d'un des 3 capteurs à effet Hall, c'est à dire au début de chaque secteur magnétique.
Elle réalise les opérations suivantes :

- échantillonnage de l'état actuel de TRM3,
- calcul de la durée "Periode_H" du dernier secteur en utilisant l'échantillon précédent de TRM3 et affectation de PR2 du Timer 2 (voir §1.5),
- calcul de l'incrément de phase "AngleInc" utilisé par la fonction "Add_Inc_angle"
- calcul de l'angle magnétique initial du stator "Mag_angle_Stator" au début du nouveau secteur (on ajoute "Offset_Angle" pour obtenir une avance ou un retard de phase du champ tournant par rapport au rotor).
- calcul de l'angle magnétique du stator à la fin du secteur détecté : "End_angle_Stator"

1.5 Add_Inc_angle

Cette fonction fait partie du programme d'interruption "PWMInterrupt". Celui-ci est activé au rythme de la fréquence PWM (CLK_PWM), soit 20kHz dans cette version.
La fonction réalise l'addition suivante :

Mag_angle_Stator ← Mag_angle_Stator + AngleInc (modulo 360°)

L'incrément d'angle statorique "AngleInc" est calculé par la fonction CNInterrupt pour obtenir une fréquence des signaux d'alimentation conforme à la durée d'un secteur : F_{STATOR} = 1/(6 * durée d'un secteur). Compte tenu du codage de la variable "Mag_angle_Stator" (voir ci-dessous), la fréquence des signaux statoriques est obtenue par la relation suivante : $F_{STATOR} = \text{AngleInc} \times \frac{F_{PWM}}{6 \times 2^{21}}$

Le résultat de l'addition est limité à la valeur "End_angle_Stator" de sorte à ne jamais faire passer le vecteur du champ statorique dans le secteur suivant (voir §1.4).
La variable "Mag_angle_Stator" a une taille de 32 bits et sa valeur évolue entre 0 et ($6*2^{21}$-1) pour représenter 0° à 360°. Cette taille permet d'obtenir une résolution de réglage de la fréquence statorique suffisante et le codage facilite l'extraction des composantes "Sect_SVM" et "Ind_Mag_sect" :

N° du secteur :	bits 23 à 21 (0 à 7 : codage sur 3 bits)
Position dans le secteur :	bits 20 à 13 (0 à 255 : codage sur 8 bits)

1.6 Table_TA&TB

Il s'agit des tables TA et TB décrites dans l'étude SVM. En fait, une seule est utilisée (la table TA), les coefficients TB étant obtenus en lisant la table TA par la fin :

- TA = Table_TA[Ind_Mag_sect]
- TB = Table_TA[255-Ind_Mag_sect]

La table comporte 256 éléments.

1.7 Calculs_PWM

Cette fonction fait partie du programme d'interruption "PWMInterrupt". Elle réalise les calculs décrits au §2.5 dans l'étude SVM. Les formules dépendent du n° du secteur actuel donné par "Sect_SVM".
La fonction multiplie aussi les valeurs "TA" et "TB" avec l'indice de modulation "Indice_m" et réalise ainsi le réglage de niveau.
Les éléments des tables TA et TB sont calculés pour obtenir des rapports cycliques de 100% quand "Indice_m"=100.

1.8 Module_PWM

Cette fonction est réalisée par une structure matérielle intégrée dans le µC. Elle produit les 6 signaux PWM qui contrôlent les 6 transistors MOS des 3 bras de commande.
Elle est paramètrée comme suit :

- fréquence PWM=20kHz
- mode "centré"
- temps mort = 500nS
- sorties PWM à l'état inactif à l'activation de FAULT (sur-intensité)
- déclenchement du CAN 10 bits à la fréquence de 20kHz/16=1,25kHz

L'entrée OVDCON est contrôlée par la fonction Ctl_PWM_Output :

- pour charger les condensateurs de bootstrap dans les phases de démarrage
- pour bloquer les MOS dans les phases d'arrêt du moteur.

1.9 CAN_10bits

Cette fonction est réalisée par une structure matérielle intégrée dans le µC.
Il s'agit d'un CAN de résolution 10 bits. La tension de référence choisie est VDD.
Pour l'instant seule l'entrée AN2 est utilisée. Un potentiomètre permet de régler la tension sur cette broche entre 0V et VDD pour produire un signal numérique compris entre 0 et 1023.
Le CAN est déclenché par le module PWM au rythme de 1,25kHz.
La fin de conversion active la fonction ADCInterrupt par interruption.

1.10 ADCInterrupt

Cette fonction est réalisée par un programme d'interruption déclenché par la fonction "CAN_10bits". Elle réalise les opérations suivantes :

- mise à l'échelle pour que "Indice_m" varie entre 0 et 99,
- limiter la vitesse de variation de l'indice de modulation pour éviter des surintensités dans le moteur.

1.11 Detect_Cmd_RxD

Cette fonction utilise L'UART du dsPIC30F2010. Ce coupleur est configuré au format suivant : 38400 bauds, pas de parité, 1 bit Stop, pas de poignée de main.
Cette fonction analyse en permanence les octets reçus sur le port UART pour identifier une commande en provenance du PC. Le format des commandes est le suivant :

Octets reçus				
1	2	3	4	5
0xAA	Code	Dh	Dl	0xAA

1° octet : 0xAA
2° octet : code de la commande
3° et 4° octets : poids fort et poids faible du paramètre
5° octet : 0xAA

Les 1° et 5° octets d'encadrement sont fixes et permettent l'identification de la commande à tous moments.
Commandes :

- Code 'B' : affectation de "Offset_angle" : Offset_angle=(Dh*256+Dl)*16
- Code 'C' : le paramètre Dh affecte l'information de sélection "Sel_indice" :
  - Dh = 'P' (Poignée) : Sel_indice = 0 : l'indice de modulation est pilotée par la poignée
  - Dh = 'T' (Tension) : Sel_indice = 1 : l'indice de modulation est pilotée depuis le PC (info "Tension")
- Code 'T' : affectation de "Tension" : Tension=Dh*256+Dl
- Code 'P' : le paramètre Dh affecte l'information "PC_Connected" :
  - Dh = 1 : PC_Connected = 1
  - Dh = 0 : PC_Connected = 0

1.12 Transmit_Param

Cette fonction utilise L'UART du dsPIC30F2010. Ce coupleur est configuré au format suivant : 38400 bauds, pas de parité, 1 bit Stop, pas de poignée de main.
Elle est validée par le signal "PC_Connected" de la fonction "Detect_Cmd_RxD".
Elle transmet alors les paramètres importants du contrôleur suivant le même format que celui décrit au §1.11.
Les paramètres sont transmis successivement au rythme de 16mS. Ce rythme est déterminé par le "Timer 1" dans le programme d'interruption T1Interrupt.
Liste des paramètres transmis :

- Code 'P' : Dh/Dl = "Periode_H"
- Code 'I' : Dh/Dl = "Indice_m"
- Code 'A' : Dh/Dl = "AngleInc"
- Code 'B' : Dh/Dl = "Offset_Angle" divisé par 16
- Code 'S' : Dh/Dl = "Mag_angle_Stator" divisé par 2^{13}

Note : sur une période de 128mS, chaque paramètre est transmis une fois, sauf 'S' qui l'est 4 fois.

2. Descriptions détaillées des fonctions

2.1 Déclaration des constantes non mémorisées et des variables

2.1.1 Constantes non mémorisées

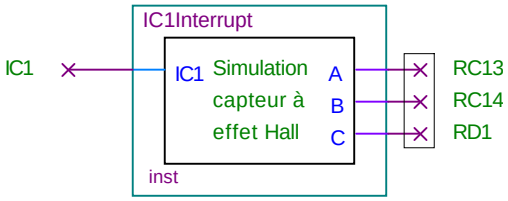
```
/*****************************
* Constantes non mémorisées  *
*****************************/
#define FCY      4000000*16/4 // xtal = 4Mhz; PLLx16 -> 16 MIPS
#define FPWM     20000        // 20 kHz, so that no audible noise is present.
#define PTPERI   400          // (FCY/FPWM-1)>>1 : moitié de la période PWM
#define PTPER_x2 800          // (FCY/FPWM-1) : période PWM
#define CW       0            // Clock Wise direction
#define CCW      1            // Counter Clock Wise direction
#define POT_Min  179          // Résultat du CAN sur position mini (0,87V)
#define POT_Max  883          // Résultat du CAN sur position mini (4,28V)
#define POT_Inc  10           // Incrément de POT_Pos_Delay toutes les mS
#define POT_Dec  5            // Décrément de POT_Pos_Delay toutes les mS
```

2.1.2 Variables

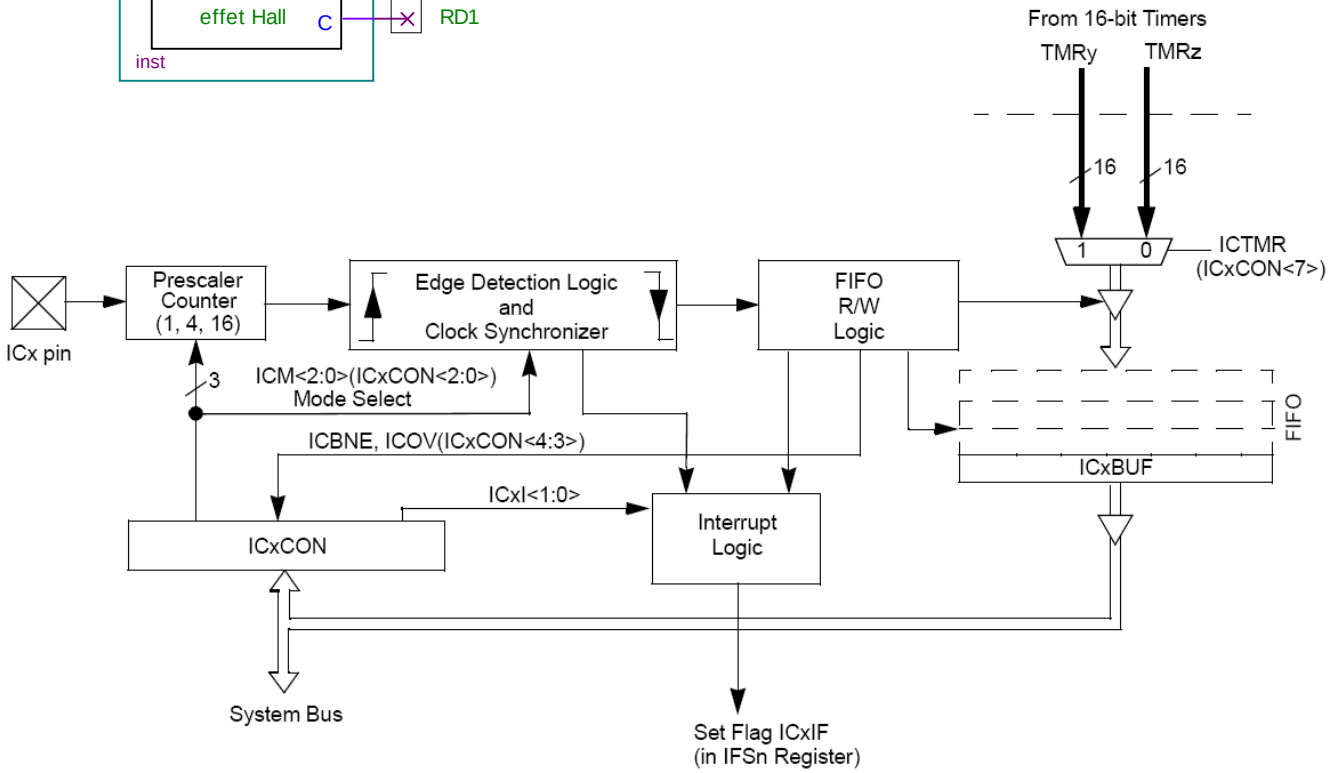
```
/*****************************
*       Variables            *
*****************************/
long     Mag_angle_Rotor;  //Angle magnétique du rotor sur 24 bits
                           //En référence aux capteurs à effet Hall
                           //  - modulo (6*2^21)
                           //  - résolution : 360/(6*2^21)=28,6µ°
                           //  - bits 23 à 21 = n° du secteur (0 à 5)
                           //  - bits 20 à 13 : indice dans le secteur (0 à 255)
long     Mag_angle_Stator; //Angle magnétique du stator
                           //Codage comme Mag_angle_Rotor
unsigned int AngleInc;     //Incrément d'angle à chaque interruption PWM
                           //Résolution=28,6µ°
long     End_angle_Stator; //Angle d'arrêt du champ magn statorique
long     Offset_Angle;     //Avance ou retard de phase entre champ
unsigned int HallValue;    //Etat actuel des 3 capteurs Hall
unsigned int Sector;       //N° actuel du secteur (0 à 5)
unsigned int LastSector;   //Etat précédent de Sector pour déter. le sens de rotation
unsigned int SimSector;    //Compteur de secteur pour simuler les capteurs
unsigned int MotorStalledCounter=0; // Compteur d'interruption RTI 1mS pour
                                    // détecter que le moteur est à l'arrêt
unsigned int Indice_m;     //Indice de modulation : 0 à 99
unsigned int T0,TA,TB;     //Paramètres pour calculs SVM
unsigned int POT_Pos;      //Position actuelle du potentiomètre
unsigned int POT_Pos_Delay;//Position du potentiomètre avec inertie
unsigned int Compt_T1;     //Compteur d'interruption Timer 1 (1mS)
unsigned int PastCapture;  // \ Pour mesure de la période
unsigned int ActualCapture;// / entre 2 secteurs du moteur
unsigned int Periode_H;    // Durée d'un secteur
// Indicateurs de fonctionnement
struct
{
 unsigned MotorRunning      :1; // Comme son nom l'indique
 unsigned Current_Direction :1; // Sens de rotation courant
 unsigned Required_Direction:1; // Sens de rotation désiré
 unsigned PC_Connected      :1; // Indique la connexion d'un PC
 unsigned Consigne          :2; // Consignes : 0 : pot sur AN2
                                //             1 : tension depuis RS232
                                //             2 : vitesse depuis RS232
                                //             3 : couple  depuis RS232
 unsigned unused            :10;
}Flags;
// Buffers RX et TX pour l'UART
#define RX_BufSize 16         // Taille du Buffer_RX
```

```
char Buffer_RX[RX_BufSize];   // Buffer circulaire de réception
char *ptrRX_WRdata=Buffer_RX; // Pointeur d'écriture ds Buffer_RX
char *ptrRX_RDdata=Buffer_RX; // Pointeur de lecture ds Buffer_RX
#define TX_BufSize 32         // Taille du Buffer_TX
char Buffer_TX[TX_BufSize];   // Buffer circulaire de transmission
char *ptrTX_WRdata=Buffer_TX; // Pointeur d'écriture ds Buffer_TX
char *ptrTX_RDdata=Buffer_TX; // Pointeur de lecture ds Buffer_TX
unsigned char Buffer_Cmd[5];  // Buffer pour la réception des commandes
```

2.2 IC1Interrupt



On exploite la voie 1 des fonctions "Input Capture" (section 13 du manuel de référence) :



Toutes les ressources disponibles ne sont pas utilisées : le périphérique est configuré pour provoquer une demande d'interruption au ↑ de IC1.
La broche affectée à IC1 n'est pas configurable : broche 15 = RD0 sur dsPIC30F2010.

2.2.1 Registres associés

Register 13-1: ICxCON: Input Capture x Control Register

Upper Byte:							
U-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
—	—	ICSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R-0, HC	R-0, HC	R/W-0	R/W-0	R/W-0
ICTMR	ICI<1:0>		ICOV	ICBNE	ICM<2:0>		
bit 7							bit 0

ICSILD	= 0	Input capture module will continue to operate in CPU Idle mode
ICTMR	= 0	TMR3 contents are captured on capture event (sans effet)
IC1<1:0>	= 00	Interrupt on every capture event
ICOV	= 0	No input capture overflow occurred
ICBNE		indicateur
ICM<2:0>	= 011	Capture mode, every rising edge

2.2.2 Sources

Initialisation du périphérique :

```
/*****************************************************************
  Function:        void InitIC1(void)
  Description:     Initialisation "Input Capture 1"
                    - interruption au flanc HILO de IC1
                   Le programme d'interruption produit les signaux
                   de simulation des capteurs à effet Hall
*****************************************************************/
void InitIC1(void)
{
 TRISDbits.TRISD0=1;  //IC1=RD0 en entrée
 TRISCbits.TRISC13=0; // Simulation capteur Hall A sur RC13
 TRISCbits.TRISC14=0; // Simulation capteur Hall B sur RC14
 TRISDbits.TRISD1 =0; // Simulation capteur Hall B sur RD1
 IC1CON=0x0003;       // Interruption au flanc montant
 IPC0bits.IC1IP=1; // Priorité faible
 IFS0bits.IC1IF=0; // Raz indicateur interruption
 PORTCbits.RC13=0; // Hall A
 PORTCbits.RC14=1; // Hall B
 PORTDbits.RD1=1;  // Hall C
}
```

Programme d'interruption associé :

```
/*****************************************************************
  Function:        void _ISR _IC1Interrupt (void)
  Description :    Production des signaux de simulation des capteurs
                   à effet Hall
*****************************************************************/
// Table de transcodage N° secteur -> Hall A,B,C
// Ne sert qu'à la simulation des capteurs à effet Hall
const char HallTable[]={6,4,5,1,3,2};

void _ISR _IC1Interrupt (void)
{
 int HallValue;
 // Production des signaux de simulation des capteurs Hall
 if (SimSector!=5) SimSector++;
              else SimSector=0;
 HallValue=HallTable[SimSector]; //Etats des capteurs
 if (HallValue&0x0001) PORTCbits.RC13=1; // Hall A
                  else PORTCbits.RC13=0;
 if (HallValue&0x0002) PORTCbits.RC14=1; // Hall B
                  else PORTCbits.RC14=0;
 if (HallValue&0x0004) PORTDbits.RD1=1;  // Hall C
                  else PORTDbits.RD1=0;
 IFS0bits.IC1IF=0;// Raz indicateur interruption
}
```

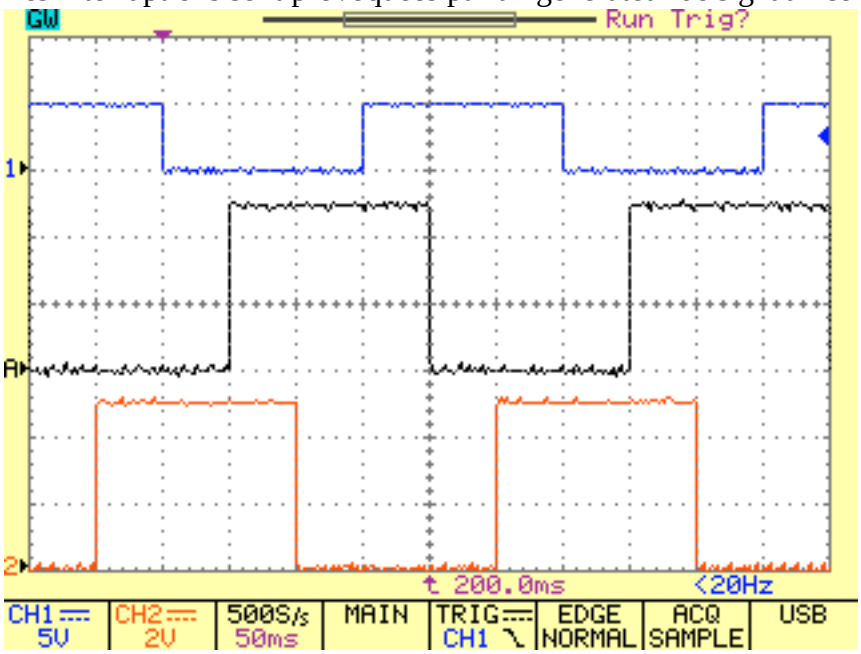
Une variable "compteur modulo 6" est utilisée. Elle est incrémentée à chaque activation du programme. Elle est alors utilisée comme indice dans la table de transcodage pour obtenir la séquence A, B, C désirée.

2.2.3 Relevés expérimentaux

Attention : le programme principal doit affecter le registre comme suit :

```
ADPCFG=~0x0004;// RB2=AN2=entrée analogique
               // Autres entrées analogique = E/S normales
```

Les interruptions sont provoquées par un générateur de signaux connecté sur l'entrée IC1 du µC.



Hall A

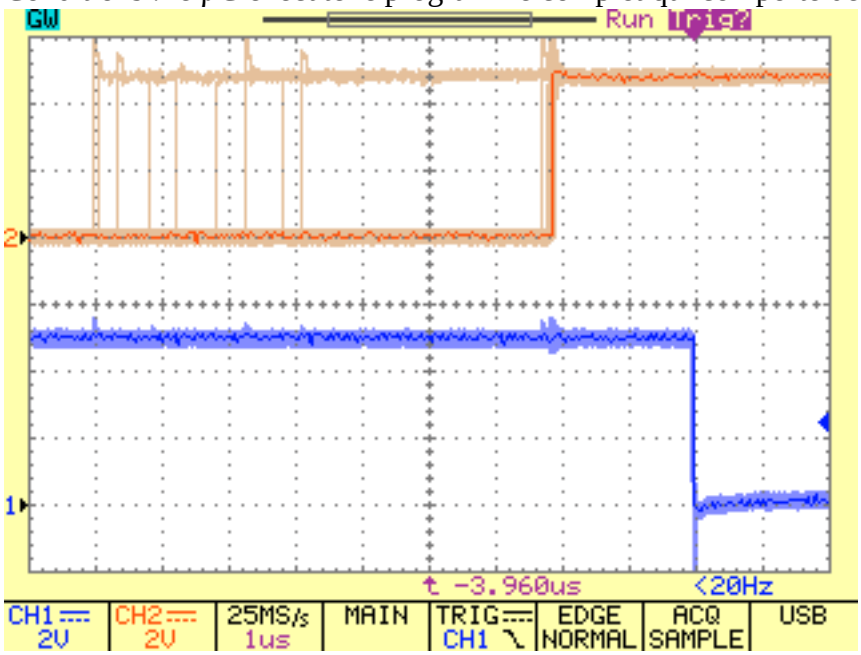
Hall B

Hall C

Résultat conforme

Temps de réaction

Conditions : le µC exécute le programme complet qui comporte de nombreuses autres interruptions.



IC1

Hall A

L'oscilloscope est déclenché au ↓ de la sortie "Hall A" (RC13). On constate que le temps de réaction est généralement de 2,2µS environ, mais il peut atteindre 9µS.
Ce temps de réaction est négligeable dans tous les cas (durée d'un secteur > 10mS)
Conclusion : l'interruption "IC1" a, par défaut, une priorité haute. C'est inutile ici : on peut la réduire pour minimiser la perturbation des autres programmes d'interruption.

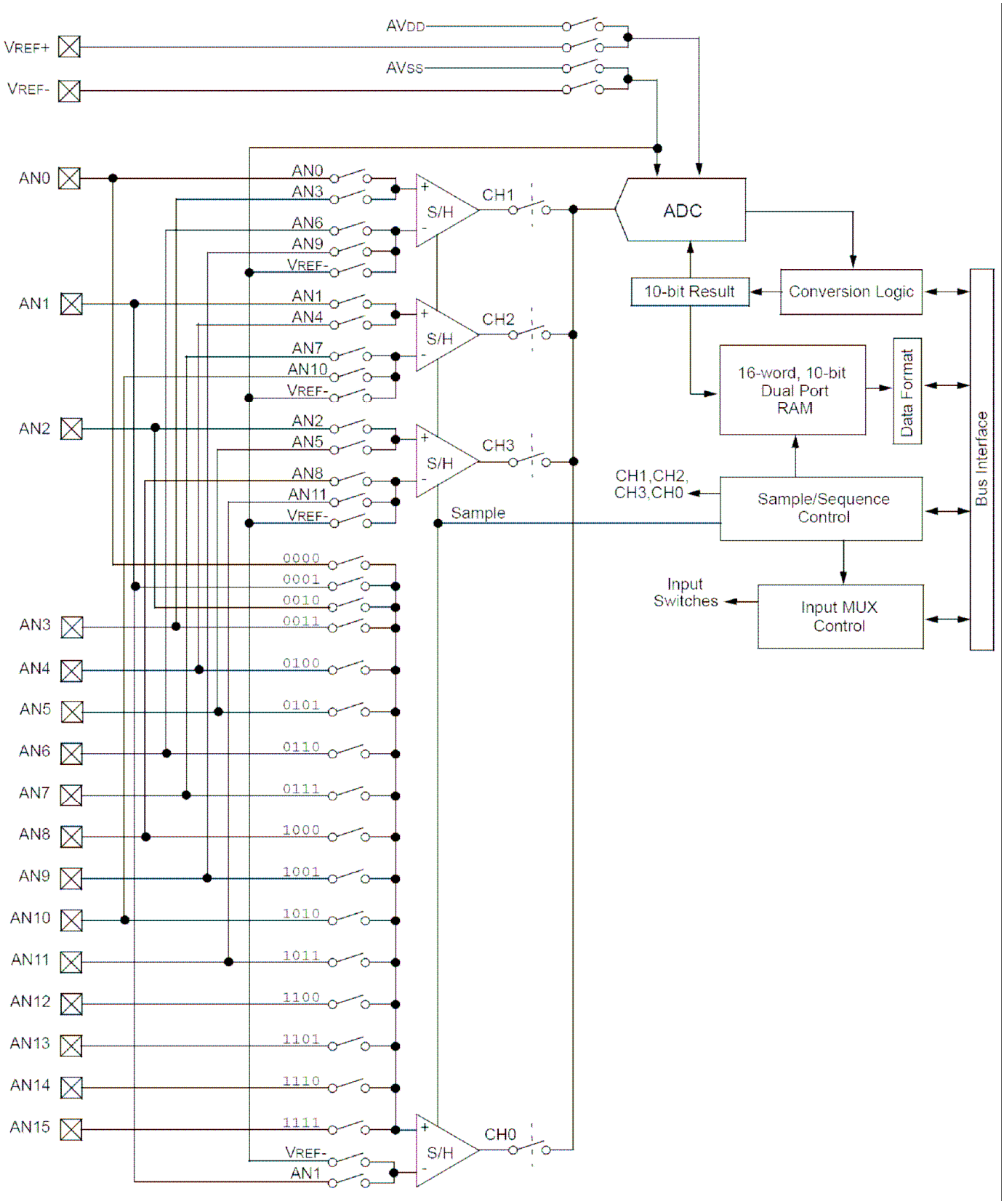
On obtient des temps de réaction max de 11µS en insérant la ligne suivante dans le programme d'initialisation :

```
IPC0bits.IC1IP=1; // Priorité faible
```

Ils restent parfaitement négligeables

2.3 CAN 10bits

Cette fonction est réalisée par une structure matérielle intégrée dans le µC.



Seules les entrées AN0 à AN5 sont disponibles sur un dsPIC30F2010.

La structure comporte tous les éléments nécessaires à la conversion (sauf une vraie source de référence) :

- un multiplexeur analogique pour choisir la source de référence (AVDD ou externe)
- un multiplexeur pour choisir l'entrée à convertir (l'échantillonnage peut être simultané pour 4 entrées)
- un CAN rapide (jusqu'à 1M ech/s)
- un registre FIFO à double accès de profondeur 16 mots.

Tous ces composants sont configurés via des registres 16 bits :

Register 17-5: ADPCFG: A/D Port Configuration Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

ADPCFG = 0x0004 : RB2=AN2=entrée analogique

Register 17-1: ADCON1: A/D Control Register 1

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0
ADON	—	ADSIDL	—	—	—	FORM<1:0>	
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0 HC, HS	R/C-0 HC, HS
SSRC<2:0>			—	SIMSAM	ASAM	SAMP	DONE
bit 7							bit 0

ADON = 1 : convertisseur opérationnel à la fin de la configuration
ADSIDL = 0 : module opérationnel en mode "idle"
FORM = 0 : résultat codé en binaire décalé (Bd)
SSRC = 3 : conversion déclenchée par le module PWM
SIMSAM = 1 : échantillonnage simultané
ASAM = 1 : échantillonnage dès que la phase de conversion est terminée
SAMP = 1 : car ASAM = 1
DONE : indicateur de fin de conversion

Register 17-2: ADCON2: A/D Control Register 2

Upper Byte:							
R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0
VCFG<2:0>			reserved	—	CSCNA	CHPS<1:0>	
bit 15							bit 8

Lower Byte:							
R-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BUFS	—	SMPI<3:0>				BUFM	ALTS
bit 7							bit 0

VCFG = 0 : VREF_H=AVDD et VREF_L=AVSS
CSCNA = 0 : pas de "scan" des entrées
CHPS = 0 : conversion de CH0 (voir schéma précédent)
BUFS = 0 : sans effet car BUFM = 0
SMPI = 0 : interruption à la fin de chaque conversion
BUFM = 0 : registre FIFO configuré en mots de 16 bits
ALTS = 0 : utilisation de la configuration "MUX A"

Register 17-3: ADCON3: A/D Control Register 3

Upper Byte:							
U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	SAMC<4:0>				
bit 15							bit 8

Lower Byte:							
R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADRC	—	ADCS<5:0>					
bit 7							bit 0

SAMC = 0 : sans effet dans ce mode de fct
ADRC = 0 : horloge CAN issu de l'horloge système
ADCS = 7 : TAD = 4.Tcy = 250nS (conversion en 12x0,25=3µS)

Register 17-4: ADCHS: A/D Input Select Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CH123NB<1:0>		CH123SB	CH0NB	CH0SB<3:0>			
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CH123NA<1:0>		CH123SA	CH0NA	CH0SA<3:0>			
bit 7							bit 0

CH123NB = 0: sans effet ici ("MUXB")
CH123SB = 0 : sans effet ici ("MUXB")
CH0NB = 0 : sans effet ici ("MUXB")
CH0SB = 0 : sans effet ici ("MUXB")
CH123NA = 0: sans effet car conversion de CH0
CH123SA = 0 : sans effet car conversion de CH0
CH0NA = 0 : entrée négative de CH0 = VREF-
CH0SA = 2 : AN2 sur entrée positive de CH0

Register 17-6: ADCSSL: A/D Input Scan Select Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL15	CSSL14	CSSL13	CSSL12	CSSL11	CSSL10	CSSL9	CSSL8
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSSL7	CSSL6	CSSL5	CSSL4	CSSL3	CSSL2	CSSL1	CSSL0
bit 7							bit 0

Registre sans effet ici.

2.3.1 Source InitADC10

```
/*****************************************************************
  Function:        void InitADC10(void)
  Description :    Initialisation de l'ADC 10 bits :
                   1. Conversion voie AN2 seulement (poignée d'accélération)
                   2. Conversion déclenchée par PWM (toutes les 16 périodes)
                   3. Interruption EOC validée
*****************************************************************/
void InitADC10(void)
{
 ADPCFG=~0x0004;// RB2=AN2=entrée analogique
 ADCON1=0x006E; // Codage en Bd sur 10 bits
                // Déclenché par module PWM
                // Echantillonnage auto
 ADCON2=0x0000; // VREF+=AVDD et VREF-=AVSS
                // Conversion de CH0 avec MUXA
                // Interruption EOC validée
                // Buffer 16 bits
 ADCHS=0x0002;  // Sélection AN2 (potentiomètre)
 ADCON3=0x0007; // Horloge système
                // Tad=4Tcy soit 250nS
 IFS0bits.ADIF=0;  // Clear ISR flag
 IEC0bits.ADIE=1;  // Enable interrupts
 ADCON1bits.ADON=1;// turn ADC ON
}
```

2.4 ADCInterrupt

La fonction est simple : elle consiste à lire le résultat de la conversion effectuée par "CAN_10bits"

```
/*****************************************************************
  Function:        void _ISR _ADCInterrupt (void)
  Description :    Programme d'interruption déclenché à chaque EOC du CAN 10 bits
                   Résultat de la conversion de AN2 rangé dans POT_Pos en Bd
*****************************************************************/
void _ISR _ADCInterrupt (void)
{
 IFS0bits.ADIF=0; // Raz indicateur interruption
 POT_Pos=ADCBUF0; // Lecture position du potentiomètre
}
```

2.5 BUS_MUX

L'indice de modulation "Indice_m" provient normalement de la poignée d'accélération via la fonction "ADCInterrupt". Mais en phase de test, elle peut provenir du PC via la liaison série et la fonction "Detect_Cmd_RxD".
Ces 2 fonctions affectent la même variable "POT_Pos", mais jamais simultanément ! Ceci est facilement réalisé en inhibant l'interruption qui active la fonction "ADCInterrupt" à partir d'un ordre en provenance du PC (signal "PC_Connected") : voir la description de "Detect_Cmd_Rxd".

2.6 T1Interrupt

Cette fonction est réalisée par un programme d'interruption activé par le "Timer 1" chaque mS.
Elle met à l'échelle l'information "POT_Pos" suivant sa source (poignée ou PC) pour l'adapter à l'indice de modulation qui doit rester entre 0 et 99.

Poignée : $\text{Indice\_m} = \dfrac{99 \times (\text{POT\_Pos\_Delay} - \text{POT\_min})}{\text{POT\_max} - \text{POT\_min}}$

POT_Pos_Delay : POT_Pos avec inertie : valeur de POT_min à POT_max
POT_min : valeur de POT_Pos avec la poignée au minimum (179 sur maquette)
POT_max : valeur de POT_Pos avec la poignée au maximum (883 sur maquette)

PC : $\text{Indice\_m} = 99 \bullet \text{POT\_Pos} \bullet 2^{-10}$ (dans ce mode : "Tension" = "POT_Pos")

Si la source est la poignée, une action "intégrale" ajoute de l'inertie à la commande pour limiter la sur-intensité dans le moteur (pas de protection dans cette version). Ainsi, POT_Pos_Delay

- ne peut augmenter que de "POT_Inc" (x99/1024 au niveau de l'indice) toutes les 32mS
- ne peut diminuer que de "POT_Dec" (x99/1024 au niveau de l'indice) toutes les 32mS

Ainsi, avec POT_Inc=10, en mettant la poignée à fond, l'indice passe de 0 à 99 en 96x32mS=3s environ.

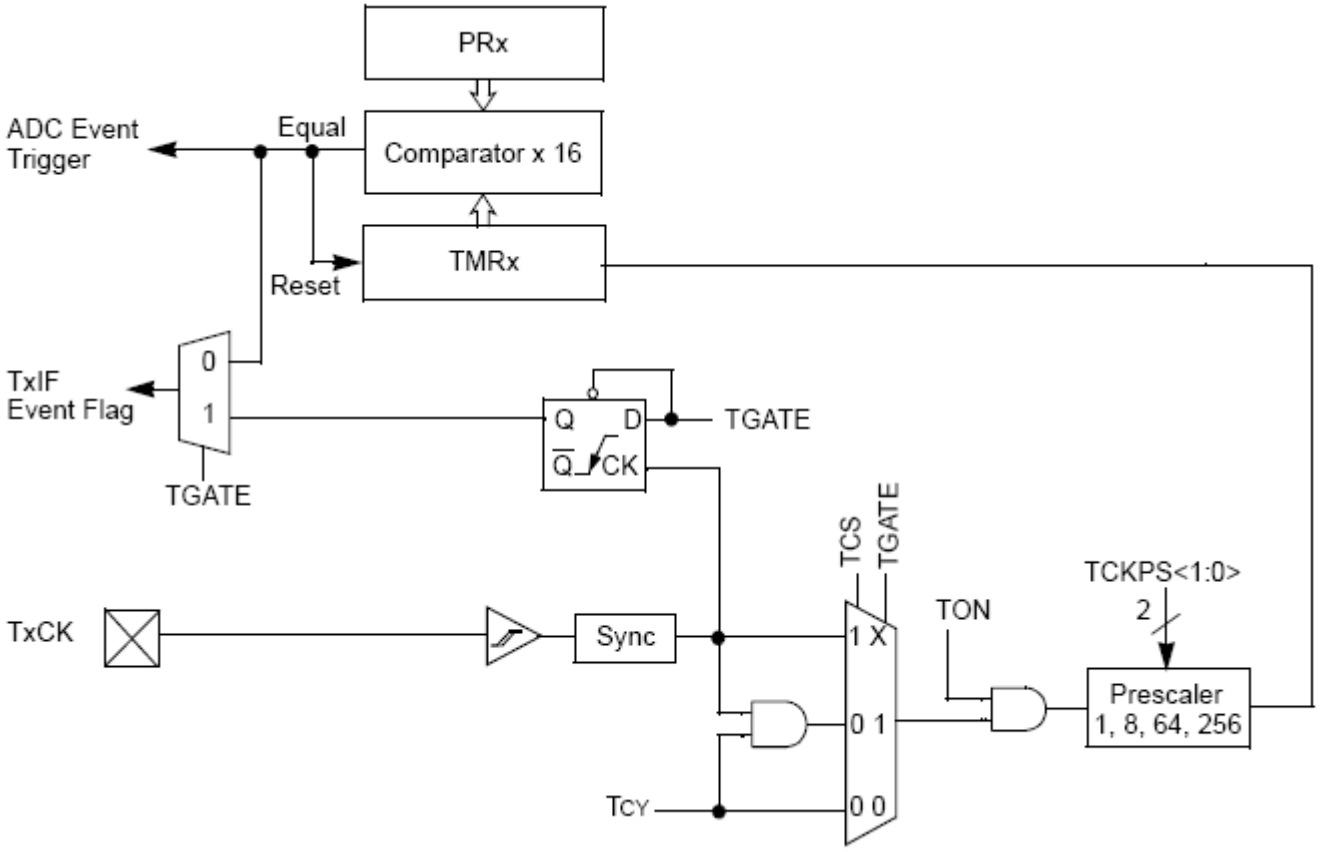
"T1Interrupt" lance aussi la fonction "Transmit_Param" qui transmet régulièrement les paramètres importants du contrôleur au PC.

Source :

```
/*****************************************************************
  Function:        void _ISR _T1Interrupt (void)
  Descrition :     Activé toutes les mS.
                   Traitement :
                    - Affectation "Indice_m" avec inertie
                    - Transmission des paramètres si Flags.PC_Connected=1
*****************************************************************/
void _ISR _T1Interrupt (void)
{
 IFS0bits.T1IF=0; //Acquitement interruption
 Compt_T1++;
 if (!(Compt_T1 & 0x001F)) //32mS ?
  {
   if (Flags.Consigne==0) // Poignée ?
   {
    if (POT_Pos > POT_Pos_Delay) POT_Pos_Delay+=POT_Inc;
    if (POT_Pos < POT_Pos_Delay) POT_Pos_Delay-=POT_Dec;
    if (POT_Pos_Delay < POT_Min) POT_Pos_Delay=POT_Min;
    if (POT_Pos_Delay > POT_Max) POT_Pos_Delay=POT_Max;
    Indice_m=((unsigned long)99*POT_Pos_Delay)
             /(POT_Max-POT_Min)-(99*POT_Min)/(POT_Max-POT_Min);
   }
   if (Flags.Consigne==1) // Consigne "Tension" depuis PC ?
   {
    Indice_m=((unsigned long)99*POT_Pos)>>10;
   }
  }
 // Transmission données série
 Transmit_Param();
 // MotorStalledCounter++;    // Pour détecter que le moteur est à l'arrêt
 if (MotorStalledCounter>1000) // Pas de chgt de secteur pdt 1 s ?
    StopMotor(); // Oui -> arrêt du moteur
}
```

2.7 Div_256 et Cpt_Mesure_TH

Ces fonctions sont réalisées par le périphérique "Timer 3" du µC. Il comporte un compteur 16 bits associé à un comparateur binaire pour réaliser un compteur modulo PR3 (PR3 est un registre accessible au CPU).



Register 12-3: TxCON: Type C Time Base Register

Upper Byte:							
R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
TON	—	TSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:							
U-0	R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	U-0
—	TGATE	TCKPS<1:0>		—	—	TCS	—
bit 7							bit 0

Le "Timer 3" est configuré comme suit :

- TON = 1 : Timer 3 "ON"
- TSIDL = 0 : Continue module operation in Idle mode
- TGATE = 0 et TCS=0 : sélection du signal d'horloge T_{CY} (T3CK non utilisé)
  indicateur T3IFG positionné au reset du compteur TMR3
- TCKPS<1:0>=11 : pré-division par 256

2.7.1 Source

Initialisation

```
/*********************************************************************
  Function:        void InitTMR3(void)
  Description:     Initialisation du Timer 3 (Timer C)
                   Horloge = Fcy/256 = 62,5kHz
                   Utilisé pour mesurer la vitesse du moteur
*********************************************************************/
void InitTMR3(void)
{
 T3CON=0x0030; // Horloge interne Fcy/256 = 62,5kHz
 TMR3=0;       // Raz du compteur pour retarder la 1° demande d'interruption
 PR3=0xFFFF;   // Comptage de 0x0000 à 0xFFFF au reset
 T3CONbits.TON=1;// Timer 3 "On"
}
```

2.7.2 Test

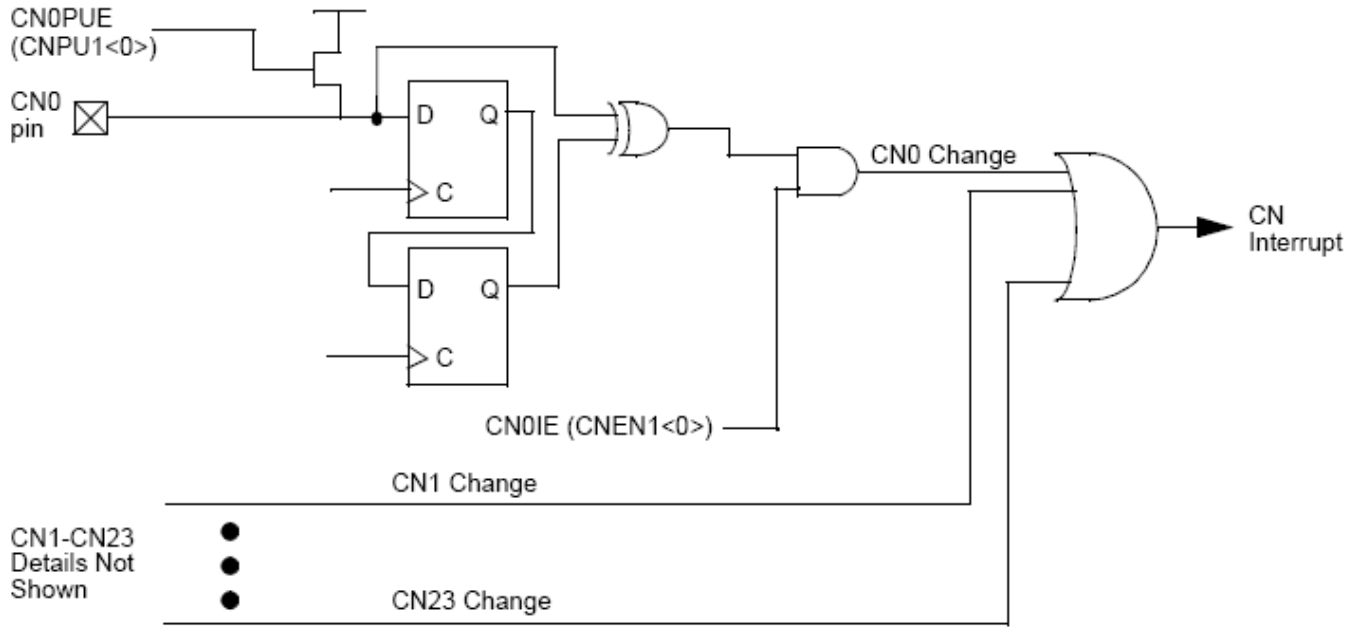
Aucune sortie n'est directement disponible. Le test est réalisé dans l'étude de la fonction CNInterrupt.

2.8 CNInterrupt

Cette fonction synchronise le champ statorique avec la position du rotor aimanté du moteur.
Les capteurs à effet Hall donnent la position du rotor avec une résolution de 60° magnétiques. Cette caractéristique est médiocre, mais les capteurs sont posés avec soins de sorte que leurs changements d'états déterminent avec précision la position du rotor à cet instant là.

On exploite une particularité des µC dsPIC qui intègrent une structure permettant de provoquer la même interruption quand une broche quelconque des ports d'E/S associés change d'état. Ces broches sont identifiées par l'acronyme CN qui signifie "Change Notification"; elles sont au nombre de 8 sur le dsPIC30F2010 (CN0 à CN7).
Cette structure est parfaitement adaptée à la fonction "CNInterrupt" : les 3 capteurs à effet Hall sont câblés sur CN5 (A), CN6(B) et CN7(C). La structure "CN" est configurée pour provoquer une demande d'interruption dès qu'une des 3 entrées change d'état.
De plus, CN5, CN6 et CN7 correspondent aux broches RB3, RB4 et RB5 du même port B ce qui facilite la lecture.

2.8.1 Structure "CN"



Les changements d'états sont détectés par la structure classique des 2 bascules D associées à une porte OU Exclusif. Chacune des entrées CNx en est équipée.
Chaque résultat de détection peut être masqué par une porte ET.

Les résultats validés sont associés avec une porte OU pour produire le signal de demande d'interruption.

Registres associés :

Register 11-1: CNEN1: Input Change Notification Interrupt Enable Register1

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CN15IE	CN14IE	CN13IE	CN12IE	CN11IE	CN10IE	CN9IE	CN8IE
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CN7IE	CN6IE	CN5IE	CN4IE	CN3IE	CN2IE	CN1IE	CN0IE
bit 7							bit 0

Les bits ouvrent ou ferment la porte ET correspondante et valident donc la fonction CN pour la broche considéré.

Register 11-3: CNPU1: Input Change Notification Pull-up Enable Register1

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CN15PUE	CN14PUE	CN13PUE	CN12PUE	CN11PUE	CN10PUE	CN9PUE	CN8PUE
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CN7PUE	CN6PUE	CN5PUE	CN4PUE	CN3PUE	CN2PUE	CN1PUE	CN0PUE
bit 7							bit 0

Ces bits permettent d'insérer une résistance de "pull-up" sur la broche associée (voir schéma).

2.8.2 Sources

Intialisations :

```
/*****************************************************************
  Function:        void InitHallSensor(void)
  Description:     Initialisation des entrées "capteurs" :
                    - interruption à chaque flanc de chaque signal
  Hall A -> CN5 (RB3)
  Hall B -> CN6 (RB4)
  Hall C -> CN7 (RB5)
*****************************************************************/
void InitHallSensor(void)
{
 // Initialisations CN
 TRISB|=0x38;      // Ports capteurs Hall en entrée (RB3, RB4 et RB5)
 CNPU1=0;         // Pas de "pull up"
 CNEN1=0xE0;       // Validation interruption CN sur RB5, RB6 et RB7
 IFS0bits.CNIF=0;// Raz indicateur d'interruption CN
 IPC3bits.CNIP=7;// Interruption CN : priorité la plus élevée pour éviter
                 // d'être interrompu par d'autres programmes
                 // CNInterrupt affecte des variables utilisées par
                 // d'autres programmes d'interruption
}
```

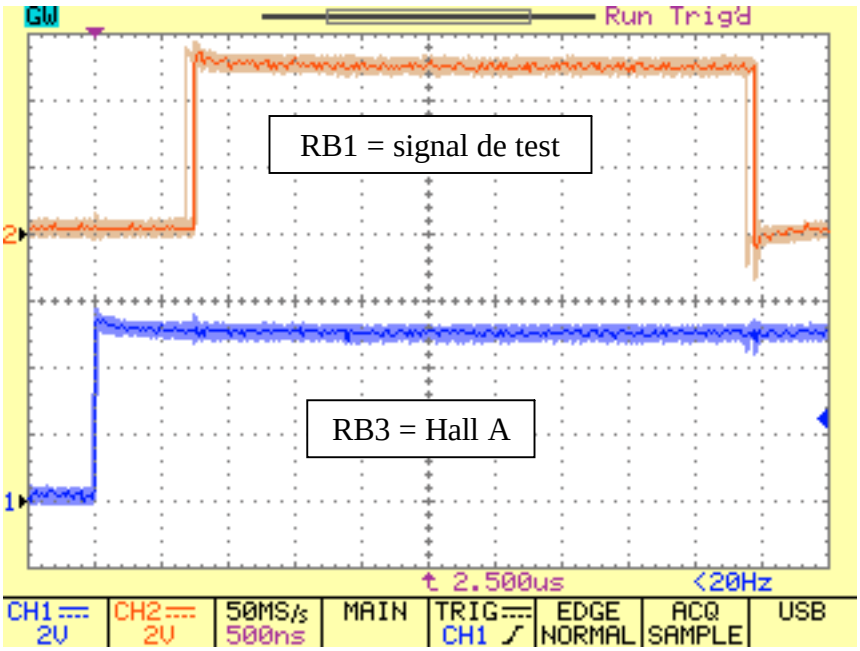
Programme d'interruption CNInterrupt :

```
/*****************************************************************
  Function:        void _ISR _CNInterrupt (void)
  Description :    Les entrées des capteurs à effet Hall sont précédées d'un
                   filtre RC de type passe-bas pour réduire les rebonds.
                   Programme d'interruption activé à chaque changement d'état
                   des capteurs Hall. Traitement :
                    - Affecter la variable "Sector"
                    - Calculer la durée du dernier secteur magnétique
                    - En déduire l'incrément d'angle "AngleInc"
                    - Affecter la variable "Mag_angle_rotor"
                    - Affecter les variables "Mag_angle_Stator"
                      et "End_angle_Stator"
*****************************************************************/
// Table de transcodage Hall A,B,C -> N° secteur
// Les valeurs d'entrées 0 et 7 sont anormales et renvoient -1
const char SectorTable[]={-1,3,5,4,1,2,0,-1};

void _ISR _CNInterrupt (void)
{
 ActualCapture=TMR3;// Capture immédiate de TMR3
 IFS0bits.CNIF=0;   // Raz indicateur interruption
 HallValue=(unsigned int)((PORTB>>3)&0x0007);//Lecture Hall
 Sector=SectorTable[HallValue];// N° secteur à partir de la table
 // Changement de secteur ?
 if ((Sector!=LastSector)&&(Sector!=-1))
 {
  LastSector=Sector; // Rafraichissement LastSector
  // Calcul de la période d'un secteur magnétique
  Periode_H=ActualCapture-PastCapture;// Soustraction de nombres NON signés
    // Periode_H=temps écoulé entre 2 captures :
    //    20mS -> Periode_H=1250
    //    1S   -> Periode_H=62500
  if (Indice_m < 2) Periode_H=65535;
  PastCapture=ActualCapture;
  //Calcul de l'incrément d'angle
  AngleInc=__builtin_divud((unsigned long)PTPERI*2*8192,Periode_H);
  //Affectation de l'angle magnétique du rotor
  Mag_angle_Rotor=(unsigned long)Sector<<21;
  //Affectation de l'angle magnétique du stator
  if (Flags.Required_Direction==CW) //Avance ou retard de phase
   {
    Mag_angle_Stator=Mag_angle_Rotor+Offset_Angle;
    if (Mag_angle_Stator>=6*2097152) Mag_angle_Stator=Mag_angle_Stator-6*2097152;
    // Angle d'arrêt du champ tournant statorique
    End_angle_Stator=Mag_angle_Stator+2097152-AngleInc; //1 secteur
    if (End_angle_Stator>=6*2097152) End_angle_Stator=End_angle_Stator-6*2097152;
   }
  else
   {
    Mag_angle_Stator=Mag_angle_Rotor-Offset_Angle;
    if (Mag_angle_Stator<0) Mag_angle_Stator=Mag_angle_Stator+6*2097152;
    // Angle d'arrêt du champ tournant statorique
    End_angle_Stator=Mag_angle_Stator-2097152+AngleInc; //1 secteur
    if (End_angle_Stator<0) End_angle_Stator=End_angle_Stator+6*2097152;
   }
  MotorStalledCounter=0;// Raz car le moteur tourne
 }
}
```

Commentaires :

– La capture de TRM3 (compteur Cpt_Mesure_TH) n'est pas réalisée par des structures matérielles. Elle est donc effectuée par la première ligne du programme pour limiter les erreurs.
Pour évaluer le temps d'échantillonnage, on ajoute 2 lignes dans le programme qui mettent la broche RB1 à "1" au début et à "0" à la fin.
Le chronogramme ci-contre montre que le temps de réaction est parfaitement négligeable (période d'horloge de TRM3=16µS).
Cela est aussi dû à la haute priorité donnée à ce programme (voir initialisations).



– La variable "Sector" est affectée en utilisant l'état lu des capteurs à effet Hall au travers d'une table de transcodage pour obtenir un résultat qui évolue entre 0 et 5 (soit 6 secteurs).

– Les autres variables ne sont affectées qu'en cas de changement de secteur valide ("`if ((Sector!=LastSector)&&(Sector!=-1))`") pour ne pas réagir à un rebond ou une lecture invalide (problème de câblage ou de capteur).

– La mesure de la durée d'un secteur ("Periode_H") est simplement réalisée par soustraction de l'échantillon précédent (précédente interruption) à l'échantillon actuel.
La gamme de mesure est limitée par la précision recherchée et la valeur max (registre 16 bits, soit 65535) :
Periode_H = 100 (précision de 1%) correspond à une durée d'un secteur de 1,6mS (5,2 tr/s pour un moteur à 20 paires de pôles)
Periode_H = 65535 correspond à une durée d'un secteur de 1,05s (1/126 tr/s pour un moteur à 20 paires de pôles)
Cette mesure est testée au §2.5.2

– L'incrément d'angle statorique est calculé de sorte à obtenir une fréquence des signaux d'alimentation conforme à la durée d'un secteur : F_{STATOR} = 1/(6 * durée d'un secteur).
Compte tenu du codage de la variable "Mag_angle_Stator", la fréquence des signaux statoriques est obtenue par la relation suivante : $F_{STATOR} = AngleInc \times \frac{F_{PWM}}{6 \times 2^{21}}$

On en déduit : $AngleInc = F_{STATOR} \times \frac{6 \times 2^{21}}{F_{PWM}} = \frac{1}{6 \times \frac{256}{F_{CY}} \times Periode\_H} \times \frac{6 \times 2^{21}}{F_{CY}} \over 2 \times PTPER$

Soit $AngleInc = \frac{2 \times PTPER}{Periode\_H} \times 2^{13}$ (PTPER = 400 pour obtenir F_{PWM} = 20kHz)

Exemples: Periode_H=100 → AngleInc=65536=8*2^{13} (32 incréments sur un secteur)
Periode_H=65535 → AngleInc=100 =12,2E^{-3}*2^{13} (20971 incréments sur un secteur)

– Si les capteurs sont bien positionnés, l'angle actuel de rotor est simplement obtenu par la relation :

```
Mag_angle_Rotor=Sector << 21;
```

Cela correspond à une multiplication par 2^{21}. Cette forme donne un programme plus rapide car le dsPIC possède un module qui réalise un nombre quelconque de décalages en une seule instruction (sur 16 bits), soit un cycle Tcy.

Rappel: Mag_angle_Rotor évolue entre 0 et $6*2^{21}$-1 pour représenter un angle entre 0° et presque 360°. Une variation de 2^{21} correspond à un secteur de 60°.

- L'angle "Mag_angle_Stator" est obtenu en additionnant (modulo $6*2^{21}$) le décalage "Offset_Angle" à l'angle du rotor.
- L'algorithme de rotation du champ statorique prévoit son blocage à la fin du secteur actuel tant que le prochain secteur n'a pas été détecté par les capteurs.
  Cet angle de "fin de secteur" est obtenu en additionnant (2^{21}-AngleInc) à l'angle actuel et est mémorisé dans la variable "End_angle_Stator"
- La variable "MotorStalledCounter" permet de détecter que le moteur est bloqué ou tourne très lentement. Elle est incrémentée toutes les mS dans le programme d'interruption T1Interrupt.
  Le programme CNInterrupt n'est activé que si le moteur tourne : la variable " MotorStalledCounter" est alors mise à 0.

2.9 Add_Inc_angle, Table_TA&TB et Calculs_PWM

Ces 3 fonctions sont réalisées dans le même programme d'interruption : **PWMInterrupt**. Celui-ci est lancé à chaque période PWM (fréquence 20kHz).

Le programme est la traduction informatique de la commande SVM (document descriptif : voir page 1). L'information de base est la variable "Mag_angle_Stator" qui indique l'angle du vecteur du champ tournant statorique à obtenir. Elle a une taille de 32 bits et sa valeur évolue entre 0 et ($6*2^{21}$-1) pour représenter 0° à 360°. Cette taille permet d'obtenir une résolution de réglage de la fréquence statorique suffisante (voir la formule ci-dessous) et le codage facilite l'extraction des composantes "Sect_SVM" et "Ind_Mag_sect" :

N° du secteur : bits 23 à 21 (0 à 7 : codage sur 3 bits)
Position dans le secteur : bits 20 à 13 (0 à 255 : codage sur 8 bits)

Ces 2 informations sont facilement obtenues par les opérations suivantes :

N° du secteur : `Sect_SVM=Mag_angle_Stator>>21`
Position dans le secteur : `Ind_Magn_sect=(Mag_angle_Stator&0x001FE000)>>13`

La variable "Ind_Magn_sect" sert d'indice dans le tableau "Table_TA" pour obtenir les coefficients TA et TB correspondantes. Ils sont multipliés par l'indice de modulation "Indice_m" pour le réglage de l'amplitude des tensions appliquées au moteur.
Le programme calcule ensuite le coefficient T0 puis les 3 rapports cycliques PDC1, PDC2 et PDC3 en fonction du n° du secteur pour produire les 3 fem permettant de placer le vecteur du champ statorique dans la position représentée par "Mag_angle_Stator".

La fonction calcule ensuite la prochaine position de cet angle (pour la prochaine interruption, soit 1/20kHz=50µS plus tard) en additionnant "AngleInc". Cet incrément d'angle statorique est calculé par la fonction CNInterrupt pour obtenir une fréquence des signaux d'alimentation conforme à la durée d'un secteur : F_{STATOR} = 1/(6 * durée d'un secteur). Compte tenu du codage de la variable "Mag_angle_Stator" (voir ci-dessus), la fréquence des signaux statoriques est obtenue par la relation suivante :

$$F_{STATOR} = AngleInc \times \frac{F_{PWM}}{6 \times 2^{21}}$$

Le résultat de l'addition est limité à la valeur "End_angle_Stator" de sorte à ne jamais faire passer le vecteur du champ statorique dans le secteur suivant (voir §1.4).

Notes :

Les valeurs des éléments du tableau "Table_TA" ont été calculées pour obtenir des rapports cycliques maximum de 100% (PDCx = 2*PTPER : voir étude module PWM)

La variable "Mag_angle_Stator" est affectée dans 2 programmes d'interruption : "CNInterrupt" et "PWMInterrupt". Pour éviter les aléas dus à l'interruption d'un programme par l'autre, le programme "CNInterrupt" est inhibé pendant l'exécution de "PWMInterrupt".

2.9.1 Source PWMInterrupt

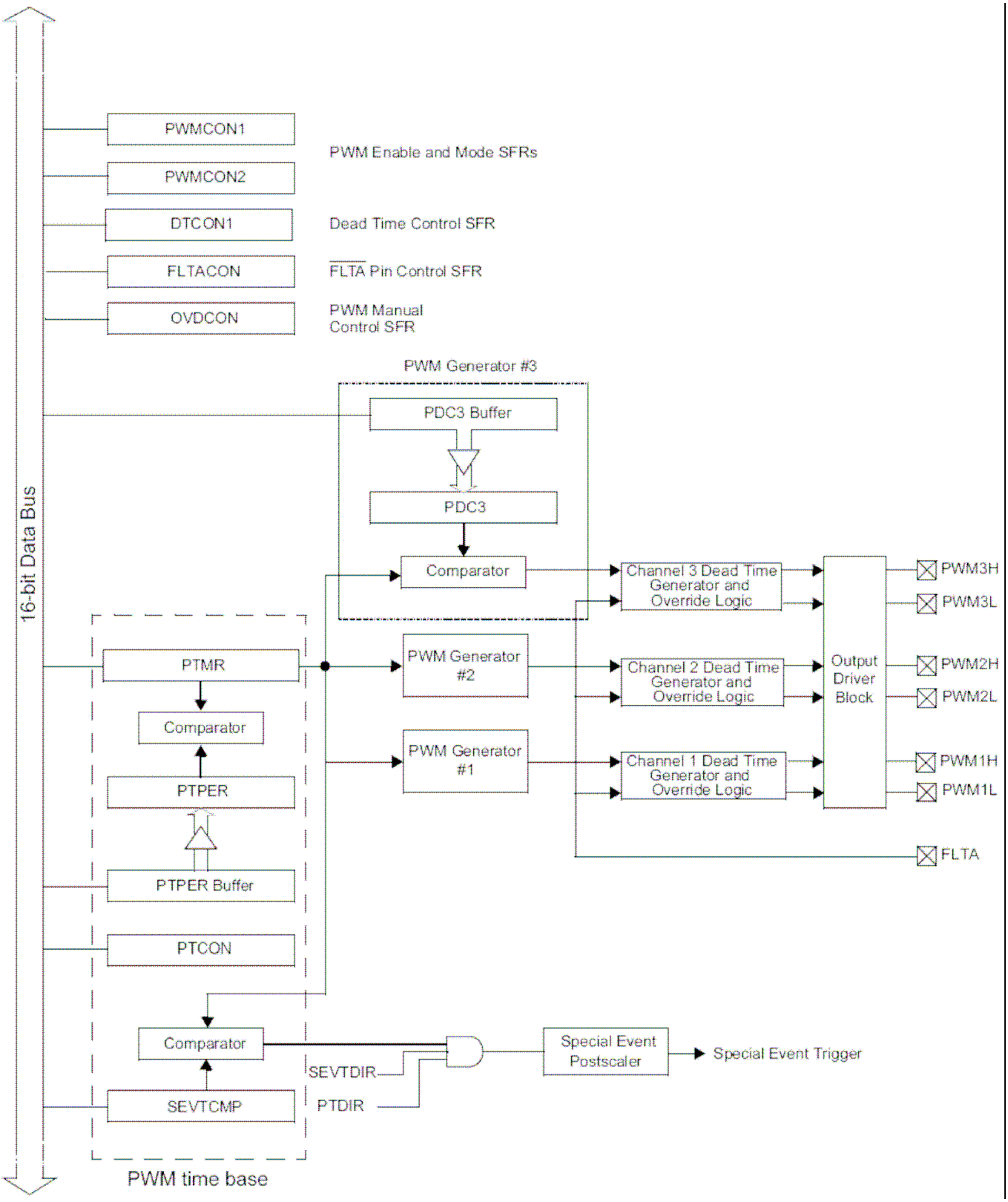
```
/*****************************************************************
  Function:        void _ISR _PWMInterrupt (void)
  Description:     Programme d'interruption exécuté à chaque période PWM
                   Fait tourner le moteur d'un angle "AngleInc" soit 1m° mag.
                   Amplitude proportionnelle à Indice_m (0 à 99)
*****************************************************************/
void _ISR _PWMInterrupt (void)
{
 unsigned int Sect_SVM;      //N° du secteur (0 à 5)
 unsigned int Ind_Magn_sect; //Indice dans le secteur (0 à 255)
 long         EcartPhase ;   //Ecart de phase jusqu'à la fin du secteur
 // Indices pour calculs PWM
 Sect_SVM=Mag_angle_Stator>>21; //N° du secteur actuel
 // Indice dans le secteur actuel (0 à 255)
 Ind_Magn_sect=(Mag_angle_Stator&0x001FE000)>>13;
 TA=((long)Table_TA[Ind_Magn_sect]*(long)Indice_m)>>8;//Résultat sur 16 bits
 TB=((long)Table_TA[255-Ind_Magn_sect]*(long)Indice_m)>>8;//Résultat sur 16 bits
 T0=(PTPER_x2-TA-TB)>>1;
 switch (Sect_SVM)
  {
   case 0 : PDC1=T0;          PDC2=T0+TA;       PDC3=PTPER_x2-T0; break;
   case 1 : PDC1=T0+TB;       PDC2=T0;          PDC3=PTPER_x2-T0; break;
   case 2 : PDC1=PTPER_x2-T0; PDC2=T0;          PDC3=T0+TA;       break;
   case 3 : PDC1=PTPER_x2-T0; PDC2=T0+TB;       PDC3=T0;          break;
   case 4 : PDC1=T0+TA;       PDC2=PTPER_x2-T0; PDC3=T0;          break;
   case 5 : PDC1=T0;          PDC2=PTPER_x2-T0; PDC3=T0+TB;       break;
   default : break;
  }
 // Inhiber interruption CN
 IFS2bits.PWMIF=0;// Clear interrupt flag
 IPC3bits.CNIP=0; // Inhiber interruption CN
 // Incrément ou décrément de phase
 EcartPhase=End_angle_Stator-Mag_angle_Stator;
 if (EcartPhase < 0) EcartPhase=6*2097152+EcartPhase;
 if (EcartPhase >= AngleInc)
  {
   if (Flags.Required_Direction==CW)
    {
     Mag_angle_Stator+=AngleInc;
     if (Mag_angle_Stator>=6*2097152) //6*2^21
         Mag_angle_Stator-=6*2097152;
    }
   else
    {
     Mag_angle_Stator-=AngleInc;
     if (Mag_angle_Stator<6*2097152)
         Mag_angle_Stator+=6*2097152;
    }
  }
 IPC3bits.CNIP=7; // Valider interruption CN
}
```

Le programme est testé au §2.10

2.10 Module PWM

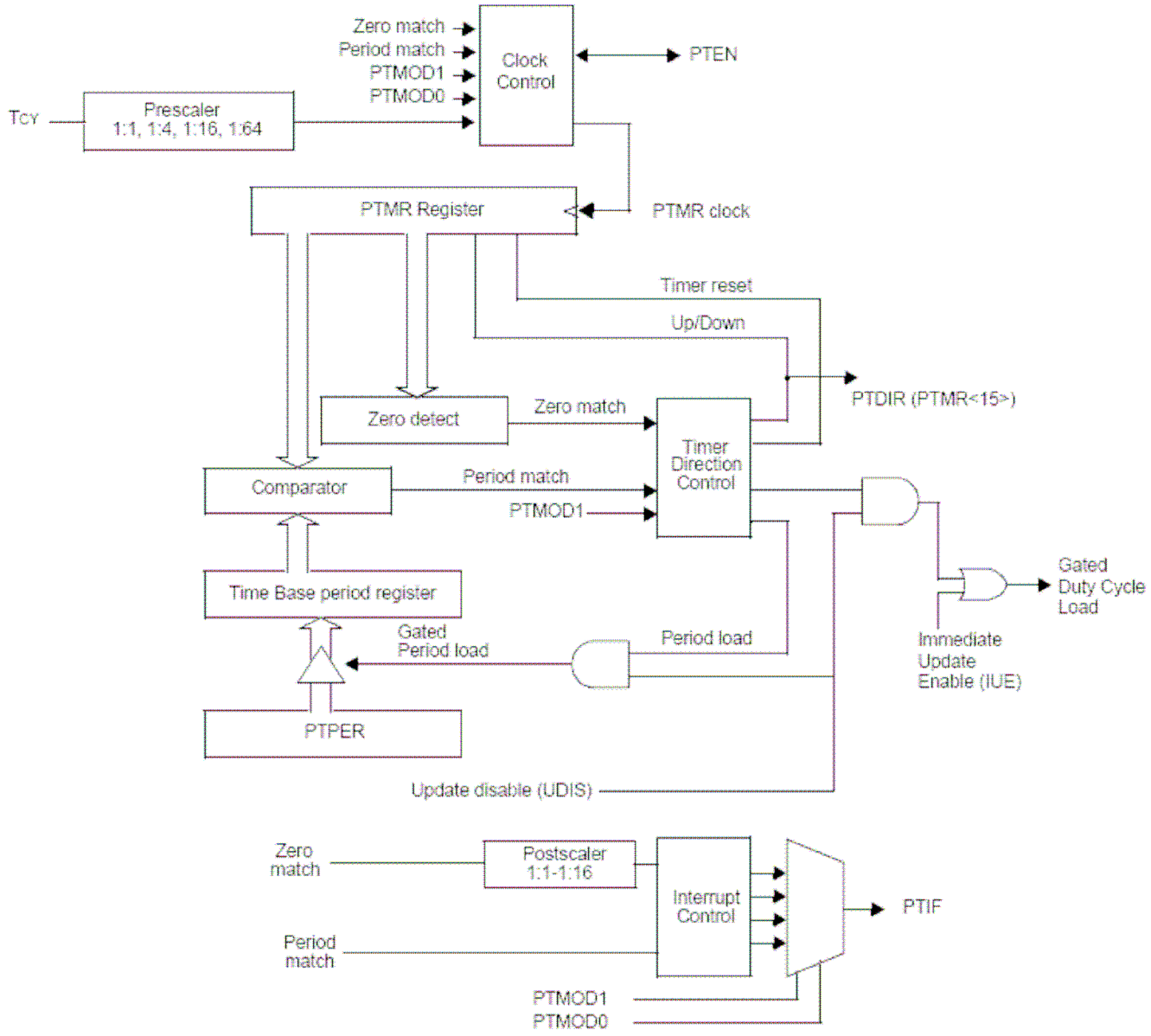
Cette fonction est réalisée par une structure matérielle intégrée dans le dsPIC30F2010. Elle comporte :

- 3 générateurs PWM indépendants pilotés par une fréquence de 2xFcy, soit 32MHz dans cette application.
- 2 sorties configurables par générateur pour piloter les 2 transistors MOS du bras associé
- un générateur de temps mort programmable
- un registre de forçage des sorties
- une entrée de protection qui place immédiatement les sorties PWM dans un état programmé
- rafraichissement des rapports cycliques synchrone ou asynchrone



2.10.1 Base de temps PWM

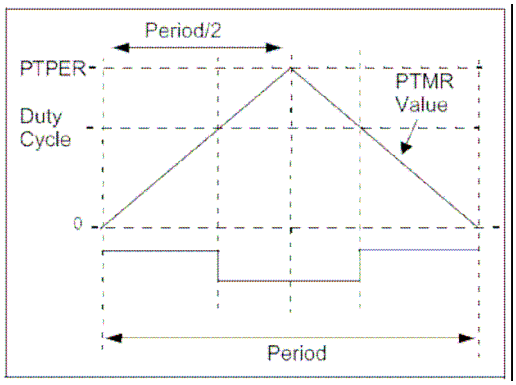
La base de temps est assez complexe comme le montre le schéma ci-dessous :



Le cœur de la base de temps est le compteur PTMR dont l'état est utilisé par les 3 générateurs PWM. La logique associée permet de faire fonctionner ce compteur selon 4 modes :

- 1 : "libre" : comptage modulo PTPER
- 2 : "monostable" : comptage jusqu'à PTPER puis arrêt
- 3 : "comptage/décomptage" modulo PTPER avec une interruption par période
- 4 : "comptage/décomptage" modulo PTPER avec deux interruptions par période

Pour cette application, la base de temps est configurée pour fonctionner dans le **mode 3**.



Ce mode est également nommé "**center aligned PWM**" comme l'illustre la figure ci-contre. On y a représenté la valeur d'un registre "rapport cyclique" (PDCx) et le résultat de sa comparaison avec le compteur PTMR (sortie PWM).
A noter que la période du signal PWM vaut 2 fois PTMR.

Rapport cyclique = $\frac{PDCx}{2 \times PTPER}$ (avec bit LSB de PDCx à 0 : voir §2.10.2)

Période PWM :
La période PWM est imposée par l'application. Pour ce programme on a choisit 50µS pour obtenir une fréquence de découpage inaudible (20 kHz) et suffisamment élevée pour bien lisser les courants du moteur.
Elle est déterminée par le facteur de "pré-division" et le contenu du registre PTPER :

$$\text{"Période PWM"} = 2 \times T_{CY} \times \text{Prediviseur} \times (\text{PTPER}+1)$$

Pour obtenir 50µS avec une fréquence F_{CY} de 16MHz, on obtient :

- Prediviseur = 1
- **PTPER = 399**

Avec cette valeur, la résolution de réglage des rapports cycliques est de 1/800, ce qui très satisfaisant pour cette application.
Attention : le compteur PTMR et le registre PTPER ont une **taille utile de 15 bits** (et non 16).

Register 15-1: PTCON: PWM Time Base Control Register

Upper Byte:

R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
PTEN	—	PTSIDL	—	—	—	—	—
bit 15							bit 8

Lower Byte:

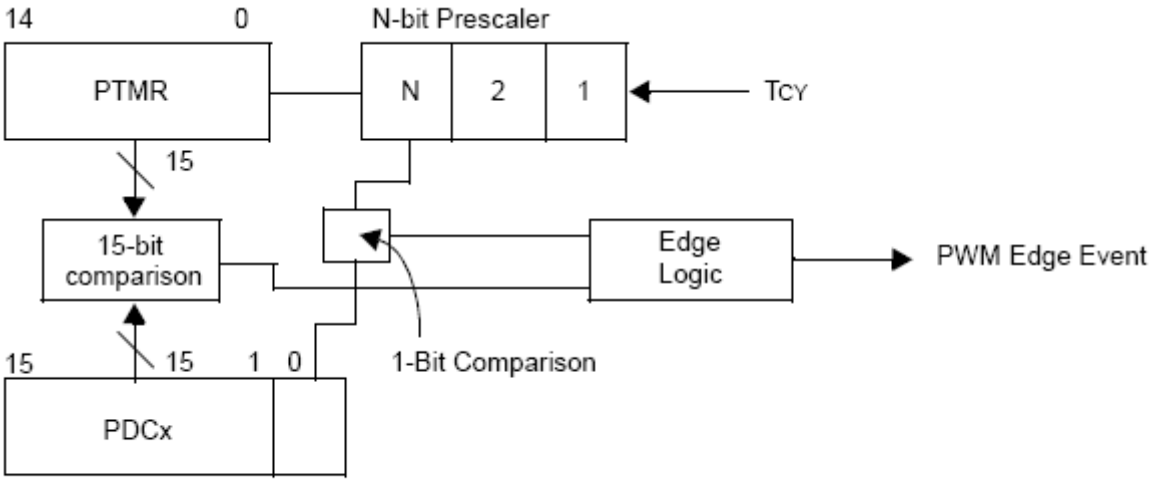
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PTOPS<3:0>				PTCKPS<1:0>		PTMOD<1:0>	
bit 7							bit 0

PTCON = 0xA002 :

PTEN = 1 : Module PWM validé
PTSILD = 1 : La base de temps PWM s'arrête en mode "veille"
PTOPS = 0 : "Postdivision" par 1 (interruption à chaque période PWM)
PTCKPS = 0 : "Pré-division" par 1
PTMOD = 2 : Mode "up/down" (center aligned PWM)

2.10.2 Générateur PWM

Chacun des 3 générateurs comporte un registre PDCx et un comparateur binaire sur 16 bits :

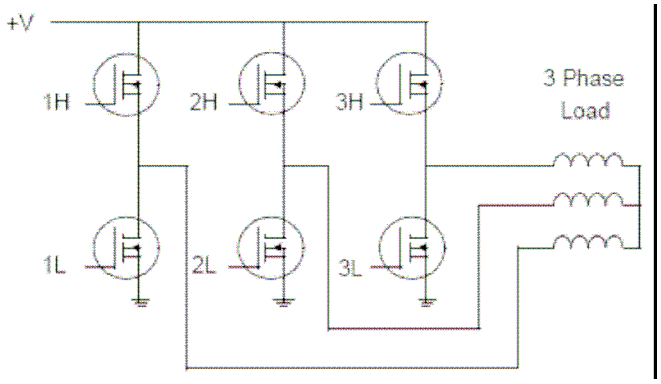


Note: PDCx<0> is compared to the FOSC/2 signal when the prescaler is 1:1.

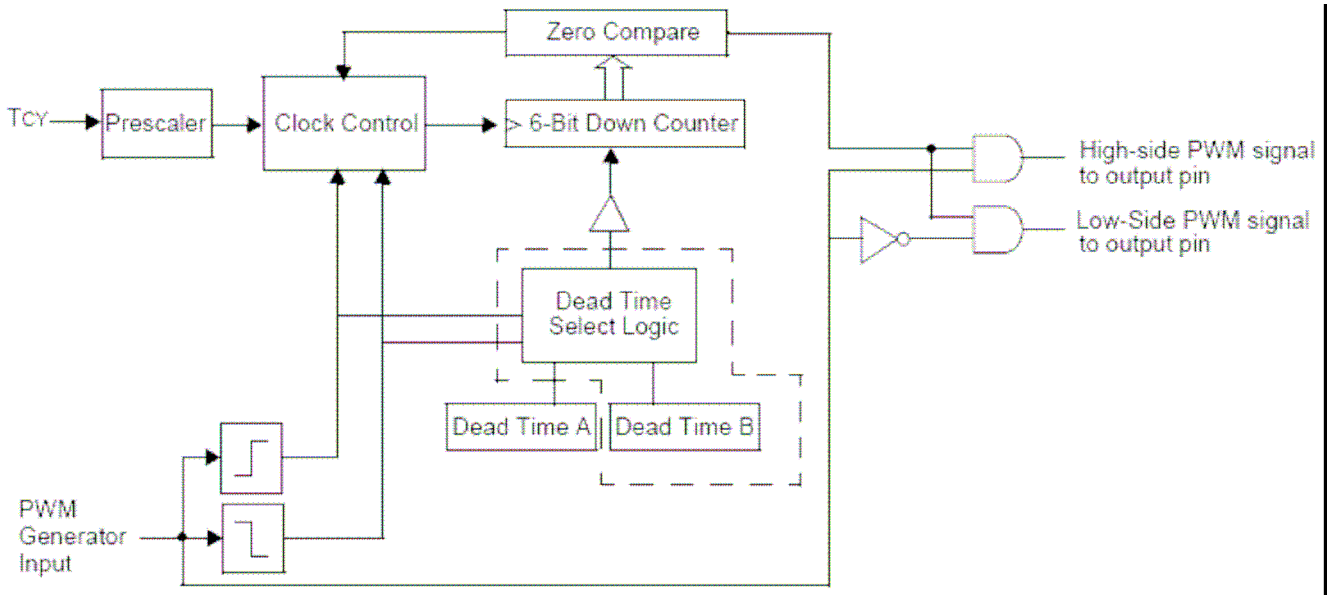
Le bit LSB du registre PDCx est comparé avec le bit de poids fort du prédiviseur ou "Fosc/2" de façon à améliorer la résolution PWM.
Le résultat de la comparaison agit sur les générateurs de temps mort.

2.10.3 Générateurs de temps morts

Cette fonction est indispensable quand on commande des ponts PWM complémentaires. Les 2 transistors de chaque bras fonctionnent en opposition : aux changements d'état l'un passe de l'état ON à l'état OFF et l'autre de l'état OFF à l'état ON.
Malgré les progrès effectués, les MOS ne commutent pas instantanément : ils passent progressivement d'un état à l'autre. Si les 2 MOS d'un bras sont commandés simultanément, ils conduiront simultanément pendant la transition. Cela entraîne de fortes pertes dans les transistors. Pour les éviter, on décale la commande des 2 transistors : c'est la fonction du générateur de temps mort.

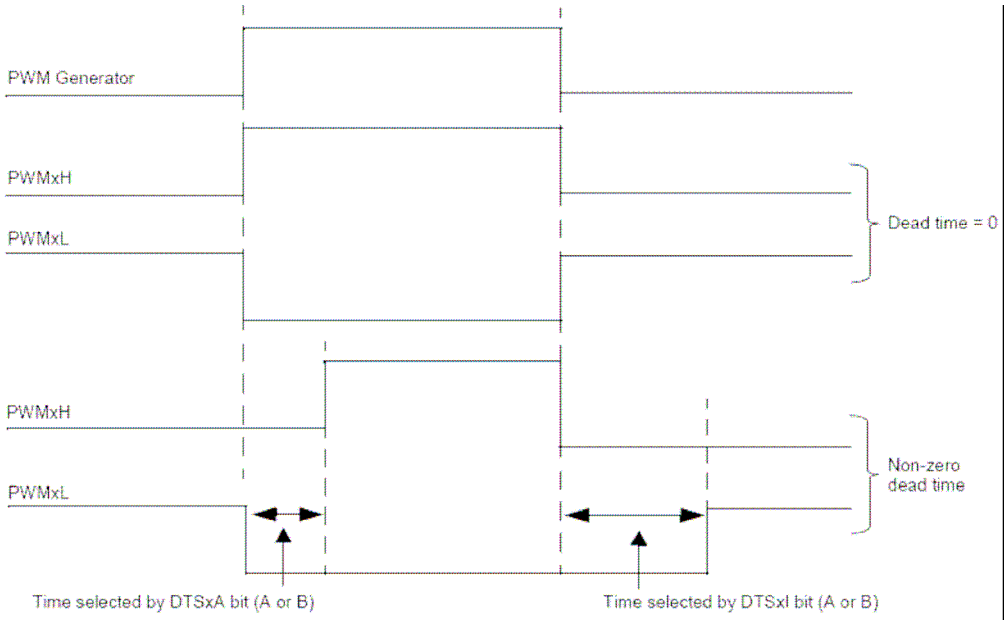


La structure intégrée dans le dsPIC30F2010 permet de régler ce retard dans une très large gamme.



Note: Logic in dashed lines not present on 6-output PWM module.

Chronogramme typique :



Registre associé :

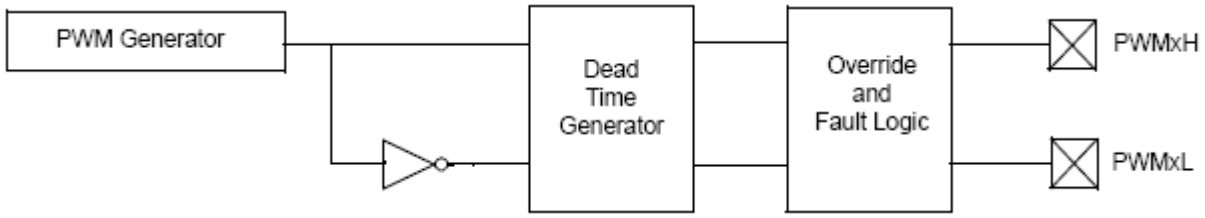
Register 15-7: DTCON1: Dead Time Control Register 1

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DTBPS<1:0>		DTB<5:0>					
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DTAPS<1:0>		DTA<5:0>					
bit 7							bit 0

DTBPS et DTB sans effet car "Dead Time B" non implanté dans le dsPIC30F2010
DTAPS = 00 : Prescaler = 1 (pas de division de fréquence)
DTA = 8 : Temps mort = 8x1xTcy = 542nS

2.10.4 Etages de sortie (en mode complémentaire)



Chacun des 3 étages comporte 2 fonctions :

- "Override" : logique de forçage des sorties
- "Fault logique" : logique de sécurité en cas d'anomalie

Ces 2 fonctions permettent d'imposer un état pré-programmé sur les 2 sorties PWMxL et PWMxH :

- par une fonction logicielle qui affecte le registre OVDCON
- par l'activation de l'entrée FLTA du µC (le logiciel n'intervient pas)

Registres associés :

Register 15-11: OVDCON: Override Control Register

Upper Byte:							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
POVD4H	POVD4L	POVD3H	POVD3L	POVD2H	POVD2L	POVD1H	POVD1L
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
POUT4H	POUT4L	POUT3H	POUT3L	POUT2H	POUT2L	POUT1H	POUT1L
bit 7							bit 0

POVD3H à POVD1H : "1" → Sortie PWMxH contrôlée par le module PWM
: "0" → Sortie PWMxH contrôlée par les bits POUTxH
POVD3L à POVD1L : "1" → Sortie PWMxL contrôlée par le module PWM
: "0" → Sortie PWMxL contrôlée par les bits POUTxL
POUT3H à POUT1H : si POVDxH = "0" :
"0" → PWMxH dans l'état inactif (MOS bloqué)
"1" → PWMxH dans l'état actif (MOS conducteur)
POUT3L à POUT1L : si POVDxL = "0" :
"0" → PWMxL dans l'état inactif (MOS bloqué)
"1" → PWMxL dans l'état actif (MOS conducteur)

Register 15-9: FLTACON: Fault A Control Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FAOV4H	FAOV4L	FAOV3H	FAOV3L	FAOV2H	FAOV2L	FAOV1H	FAOV1L
bit 15							bit 8

Lower Byte:							
R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTAM	—	—	—	FAEN4	FAEN3	FAEN2	FAEN1
bit 7							bit 0

FAOV3H à FAOD1H : "1" → PWMxH dans l'état actif si "faute"
: "0" → PWMxH dans l'état inactif si "faute"
FAOV3L à FAOD1L : "1" → PWMxL dans l'état actif si "faute"
: "0" → PWMxL dans l'état inactif si "faute"
FLTAM : "0" → L'activation de l'entrée FLTA est mémorisée
FAEN3 à FAEN1 : "1" → Paire PWMxH/PWMxL contrôlée par l'entrée "faute"
: "0" → Paire PWMxH/PWMxL non contrôlée par l'entrée "faute"

Note : le dsPIC30F2010 ne comporte pas de générateur PWM d'indice "4"

2.10.5 Programme d'initialisation

```
/*********************************************************************
  Function:        void InitMCPWM(void)
  Description:     Initialisation des 3 modules PWM comme suit :
                   1. FPWM = 20000 Hz
                           2. PWM aligné au centre
                   2. Sorties PWM complémentaires
                   3. Affectation rapport cyclique à 50%
                   4. ADC déclenché par le "PWM special trigger"
                   5. Temps mort de 500nS
*********************************************************************/
void InitMCPWM(void)
{
 TRISE=0xFFC0;  // PWMxx en sortie (RE0 à RE5), FLTA=RE8 en entrée
 PTPER=PTPERI;  // Moitié de la période PWM
 OVDCON=0x0000; // Toutes les sorties PWM à "0"
 DTCON1=0x0008; // Temps mort 500 ns
 PWMCON1=0x0077;// Validation des 6 sorties PWM
                // Mode "complémentaire"
 PDC1=PTPER;    // \
 PDC2=PTPER;    //  > Rapports cyclique 50% : 0V aux bornes du moteur
 PDC3=PTPER;    // /
 SEVTCMP=1;     // ADC déclenché par PWM
 PWMCON2=0x0F02;// Postscale = 16 : déclenchement toutes les 16 PWM
                // Rafraichissement PWM synchrones période PWM
                // Rafraichissement validé
 PTCON=0xA002;  // PWM validé en mode "centré"
                // La base de temps PWM s'arrête en mode "veille"
 FLTACON=0x0007;// L'entrée FLTA place immédiatement les sorties
                // PWM à l'état inactif.
}
```

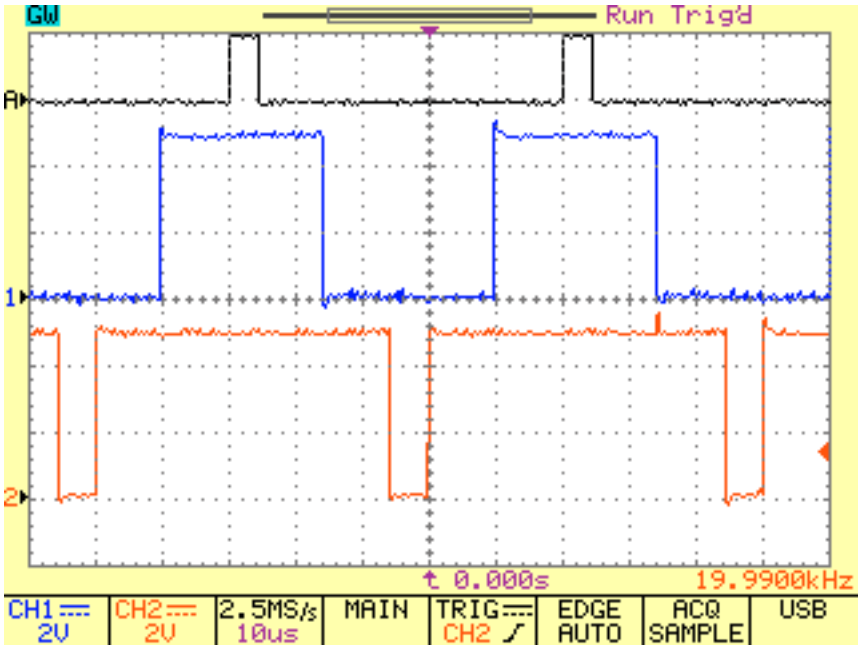
Note : le programme d'interruption associé (PWMInterrupt) est décrit au §2.5.1

2.10.6 Tests

On utilise un programme principal de test qui fixe les rapports cycliques : PWM1 = 10%, PWM2 = 50% et PWM3 = 90%.

2.10.6.1 Base de temps PWM

```
/*****************************************************************
  Function:        void main(void)
  Description:     Appelée au reset
*****************************************************************/
int main(void)
{
 InitMCPWM(); // Initialisation PWM
 PDC1=80;  // 10%
 PDC2=400; // 50%
 PDC3=720; // 90%
 OVDCON=0x3F00;     // PWM0-5 pilotés par module PWM
 while (1) {};
}
```



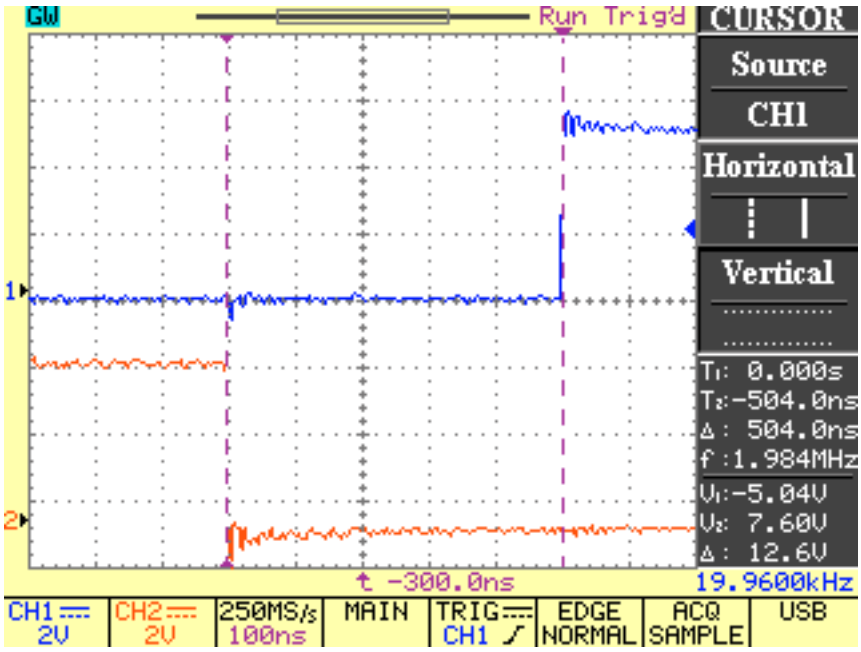
PWM1H

PWM2H

PWM3H

La fréquence et les rapports cycliques sont conformes
On remarque aussi le mode de fonctionnement "centré"

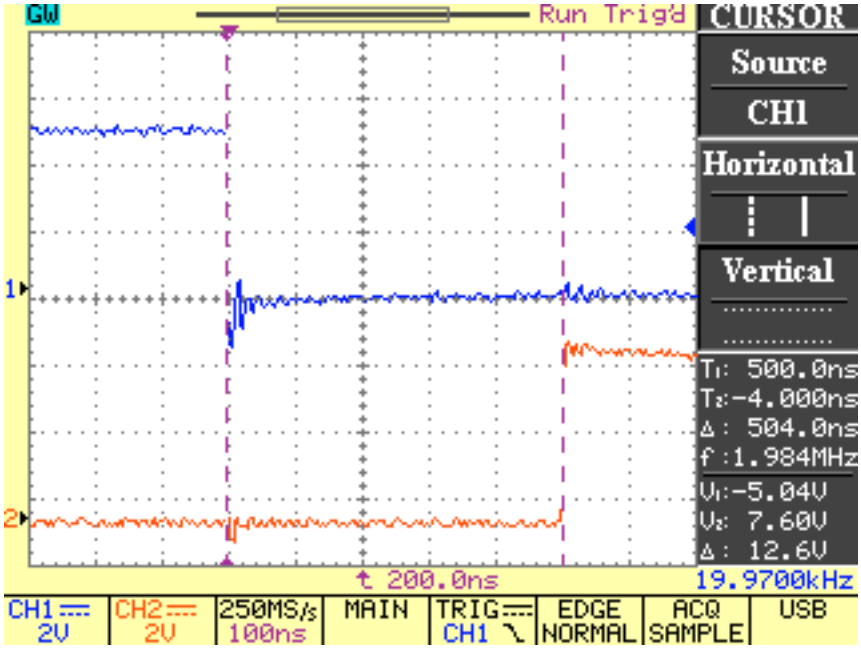
2.10.6.2 Temps morts



PWM1H

PWM1L

Le temps mort dans cette phase est conforme (500nS)



PWM1H

PWM1L

Le temps mort dans cette phase est également conforme (550nS)

2.10.6.3 Action de FLTA

On constate que toutes les sorties PWM passent à l'état bas à l'activation de l'entrée FLTA ("0").
Elles restent dans cet état jusqu'à l'acquitement du défaut : raz de l'indicateur FLTAIF du registre IFS2.

2.11 Test du programme PWMInterrupt

Il s'agit de vérifier l'algorithme SVM : la production des 3 tensions inter-phases sinusoïdales et déphasées de 120°. On vérifie aussi la durée d'exécution du programme PWMInterrupt qui doit être largement inférieure à sa période d'activation.
Pour ce faire, on utilise le programme de simulation des capteurs IC1Interrupt déclenché par un générateur externe connecté sur IC1. On connecte les 3 sorties simulées sur les 3 entrées des capteurs à effet Hall (RC13 avec RB3, RC14 avec RB4 et RD1 avec RB5).
Une observation directe des signaux PWM ne permet pas de vérifier l'algorithme : la grandeur caractéristique est **la valeur moyenne des signaux PWM**. Ainsi, les 3 sorties " PWM1H" à " PWM3H" du dsPIC sont reliées à 3 filtres passe-bas du 1° ordre (R=10KΩ, C=33nF) de fréquence de coupure Fc=480Hz pour en extraire leurs valeurs moyennes.
Les composantes de fréquence F_{PWM} (20kHz) et ses harmoniques sont atténués par ce filtre. Par contre, la fréquence des signaux d'alim du moteur ne dépasse pas 100Hz à la vitesse max; cette fréquence n'est pas atténuée par le filtre (CQFD).

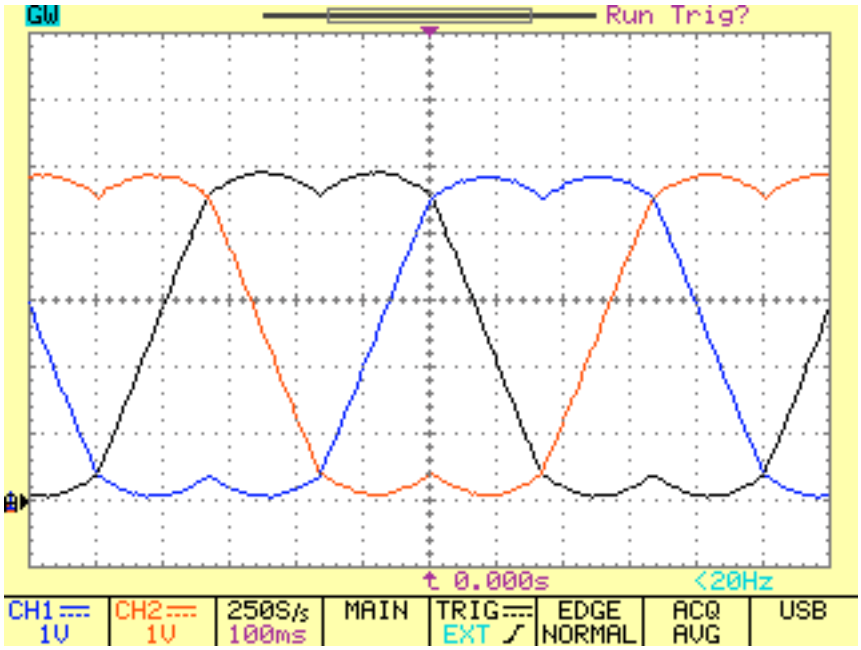
2.11.1 Programme "main" de test

```
/*********************************************************************
  Function:        void main(void)
  Description:     Appelée au reset
*********************************************************************/
int main(void)
{
 U1MODE=0; //Inhiber l'UART à cause du bootloader
 TRISB=0x003C;// RB0 et RB1 en sortie pour tests
 // Init variables
 SimSector=0;
 Indice_m=0;
 Offset_Angle=0;
 Mag_angle_Stator=0; Mag_angle_Rotor=0; AngleInc=628;
 Flags.Required_Direction==CW;
 InitADC10(); // Intialisation ADC 10 bits
 InitTMR1();  // Initialisation Timer 1 pour RTI de 1 ms
 InitTMR3();  // Initialisation Timer 3 pour mesure vitesse
 InitHallSensor(); // Initialisations entrées capteurs Hall
 InitMCPWM(); // Initialisation PWM
```

```
 InitIC1();          // IC1 déclenche la production des signaux Hall de test
 OVDCON=0x3F00;      // PWM0-5 pilotés par module PWM
 // Validation des interruptions
 IEC0bits.T1IE=1;    // Validation interruption Timer 1
 IEC0bits.CNIE=1;    // Validation interruption CN
 T2CONbits.TON=1;    // Timer 2 "ON" : le moteur tourne
 IEC0bits.IC1IE=1; // Validation interruption IC1 (TEST)
 while (1) {};       // Boucle sans fin
}
```

2.11.2 Sorties PWM

Fréquence générateur = 6Hz → fréquence statorique = 1Hz (AngleInc=629)

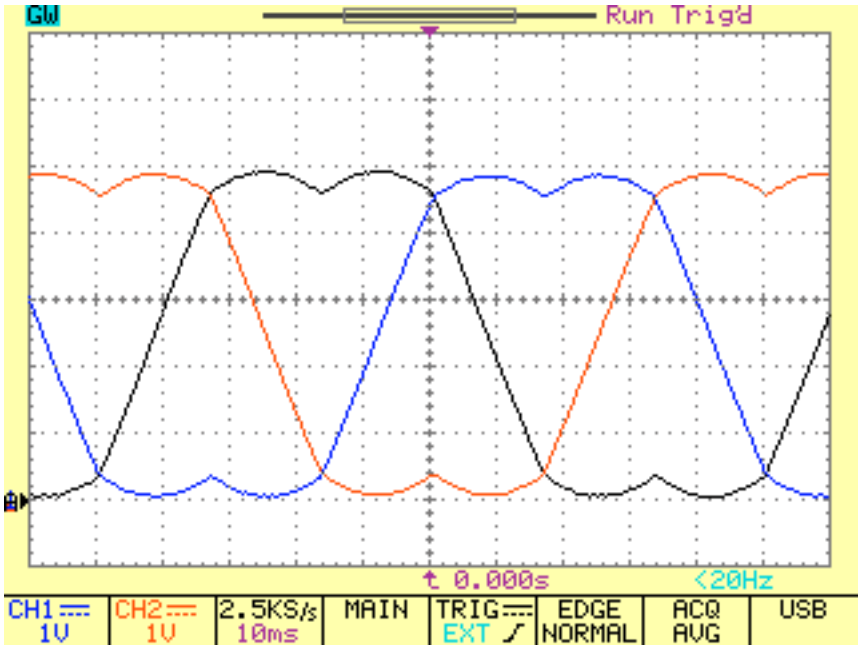


Synchro sur capteur Hall A (RB3)
Moyennage : 8

Bleu : PWM1H
Noir : PWM2H
Rouge : PWM3H

Résultat conforme

Fréquence générateur = 60Hz → fréquence statorique = 10Hz (AngleInc=6291)

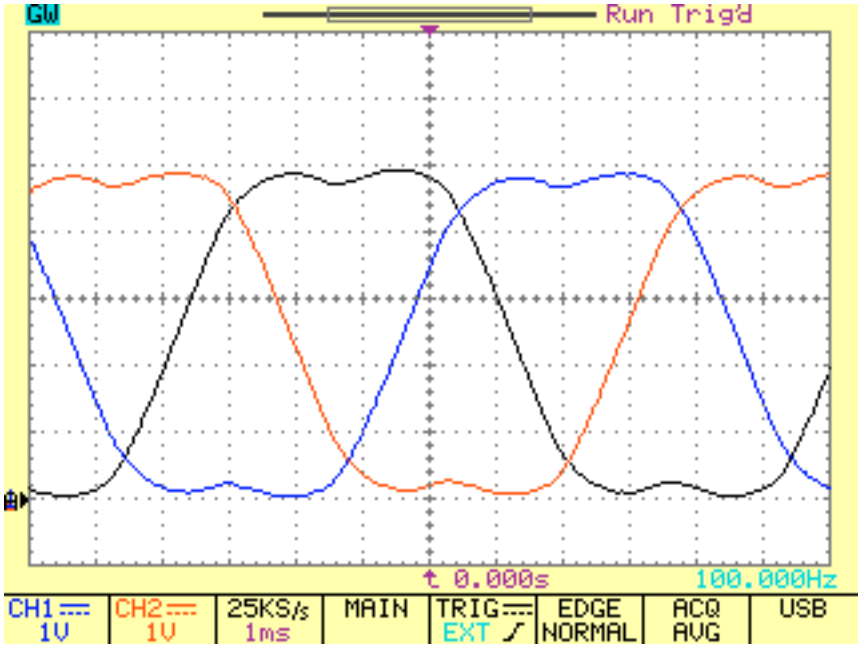


Synchro sur capteur Hall A (RB3)
Moyennage : 8

Bleu : PWM1H
Noir : PWM2H
Rouge : PWM3H

Résultat conforme

Fréquence générateur = 600Hz → fréquence statorique = 100Hz (AngleInc=62915)

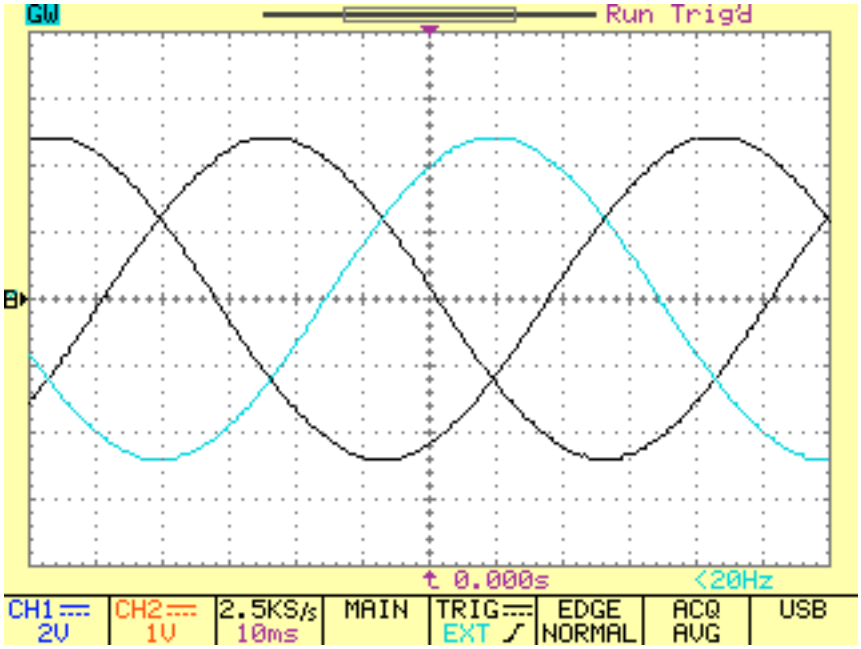


Synchro sur capteur Hall A (RB3)
Moyennage : 8

Bleu : PWM1H
Noir : PWM2H
Rouge : PWM3H

Résultat conforme

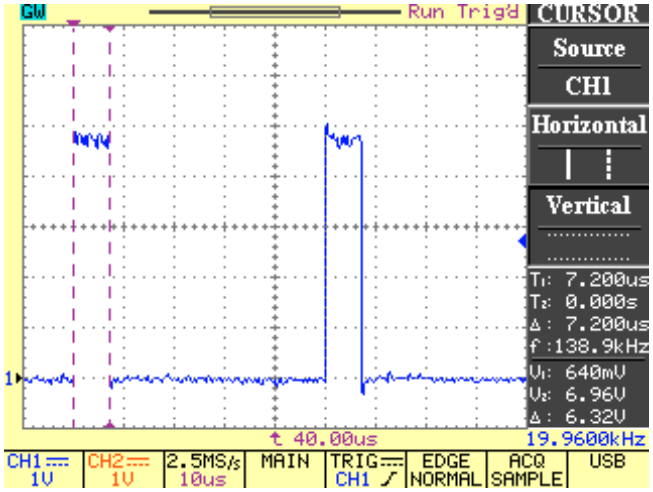
Tensions différentielles



De gauche à droite :
PWM1L-PWM2L
PWM2L-PWM3L
PWM3L-PWM1L

Résultat conforme

2.11.3 Durée d'exécution de PWMInterrupt

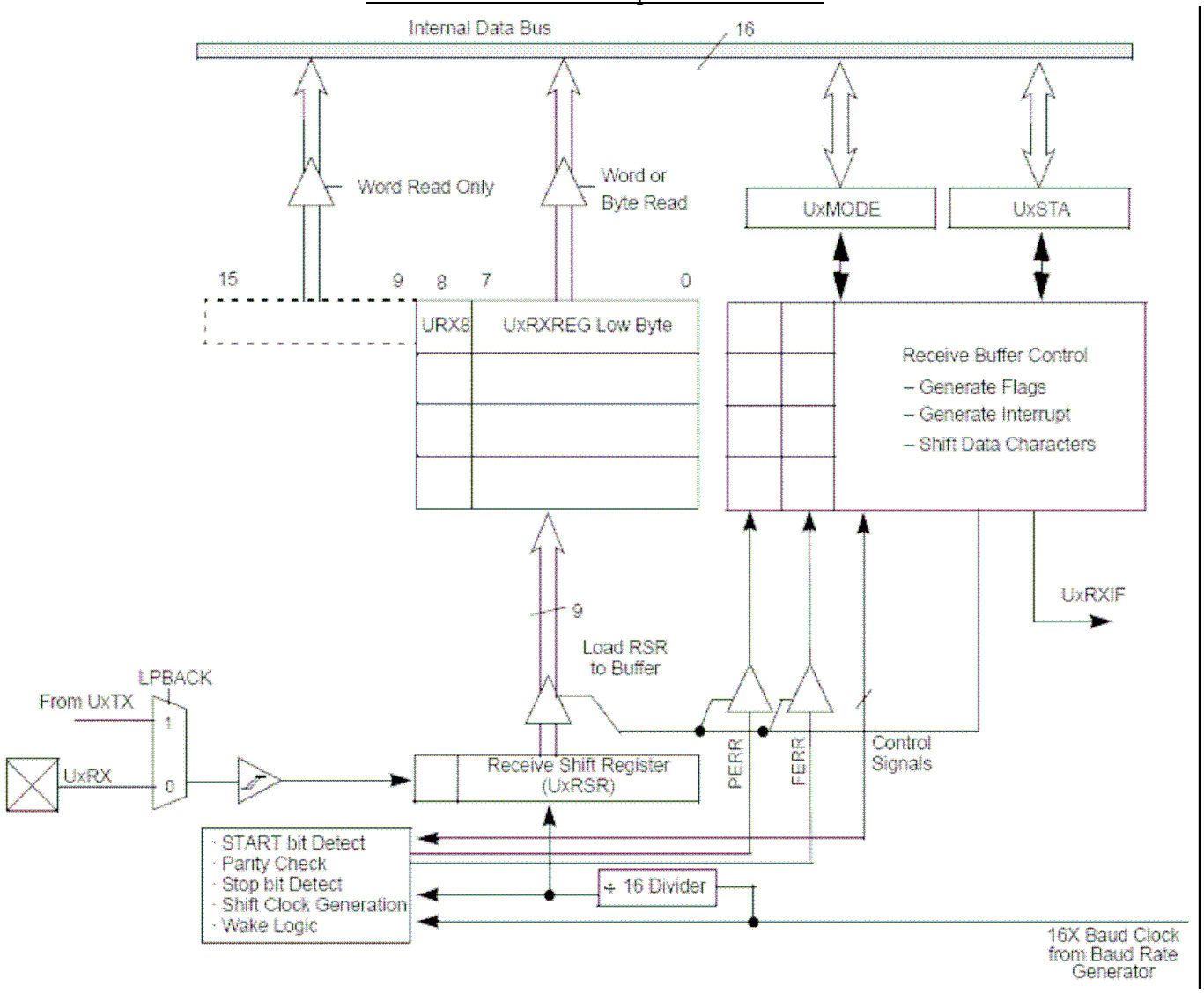


Le port RB0 est mis à "1" au début de PWMInterrupt et remis à 0 à la fin de ce programme d'interruption. Cela permet une mesure approximative du traitement correspondant.
Sur ce relevé, on constate une durée de 7,2µS. C'est une valeur très satisfaisante compte-tenu de la période des interruptions : 50µS (15%).
Il peut être interrompu par des programmes de courtes durée et ne rallonge que légèrement les programmes qu'il interrompt.

2.12 Detect_Cmd_RxD

Cette fonction utilise l'UART du dsPIC30F2010. Ce coupleur est configuré au format suivant : 38400 bauds, pas de parité, 1 bit Stop, pas de poignée de main.
La fonction est appelée dans la boucle du programme principal. Elle place l'éventuel caractère reçu dans un buffer circulaire et détecte l'arrivée d'une commande sur les 5 derniers (voir §1.11)

Schéma du module réception de l'UART



Le récepteur est équipé d'un registre FIFO d'une profondeur de 4 octets géré automatiquement.

2.12.1 Registres associés

Register 19-1: UxMODE: UARTx Mode Register

Upper Byte:

R/W-0	U-0	R/W-0	U-0	U-0	R/W-0	U-0	U-0
UARTEN	—	USIDL	—	reserved	ALTIO	reserved	reserved
bit 15							bit 8

Lower Byte:

R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	—	—	PDSEL<1:0>		STSEL
bit 7							bit 0

UARTEN = 1 : UART validé
USIDL = 0 : UART validé en mode "idle"
ALTIO = 0 : le dsPIC30F2010 ne comporte pas de broches alternatives
WAKE = 0 : fonction "Wake up" inhibée

LPBACK = 0 : fonction "LoopBack" inhibée
ABAUD = 0 : fonction "Auto baud" inhibée
PDSEL = 0 : format 8 bits, sans parité
STSEL = 0 : 1 bit stop

Register 19-2: UxSTA: UARTx Status and Control Register

Upper Byte:							
R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL	—	—	—	UTXBRK	UTXEN	UTXBF	TRMT
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/C-0	R-0
URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA
bit 7							bit 0

UTXISEL à TRMT : concernent la transmission : voir §2.13
URXISEL = 0 : interruption à chaque caractère reçu
ADDEN = 0 : mode "Adress Detect" inhibé
RIDLE : indicateur d'état du récepteur
PERR : indicateur d'erreur de parité
FERR : indicateur d'erreur de trame
OERR : indicateur d'erreur de dépassement du tampon FIFO
URXDA : indicateur de réception d'un ou plusieurs caractère

Register 19-3: UxRXREG: UARTx Receive Register

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	R-0
—	—	—	—	—	—	—	URX8
bit 15							bit 8

Lower Byte:							
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
URX<7:0>							
bit 7							bit 0

URX8 : non utilisé ici
URX : prochain caractère à lire

Register 19-5: UxBRG: UARTx Baud Rate Register

Upper Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRG<15:8>							
bit 15							bit 8

Lower Byte:							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRG<7:0>							
bit 7							bit 0

Formules de calcul : $$Baud\ Rate = \frac{FCY}{16 \bullet (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{16 \bullet Baud\ Rate} - 1$$

2.12.2 Fonction InitUART

```
/*********************************************************************
  Function:        void InitUART(void)
  Description:     Initialisation de l'UART
                     - 38400 bauds, 8 bits, pas de parité, 1 bit Stop
                     - interruption sur TX si buffer plein
                     - interruption sur RX à chaque caractère
*********************************************************************/
void InitUART(void)
{
 U1MODE=0x8000; // UART validé, 8 bits, pas de parité, 1 bit stop
 U1STA =0x8400; // Interruption TX si buffer vide
                // UART TX validé
                // Interruption RX à chaque caractère
 U1BRG =(FCY/(16*38400))-1; // 38400 bauds (erreur +0,16%)
}
```

2.12.3 Sources

De nombreuses sources d'interruptions sont utilisées dans cette application. Pour éviter de perdre des caractères, on a décidé d'utiliser le principe du buffer de réception circulaire.

Rappel de la déclaration des constantes et variables associées :

```
// Buffer RX pour l'UART
#define RX_BufSize 16         // Taille du Buffer_RX
char Buffer_RX[RX_BufSize];   // Buffer circulaire de réception
char *ptrRX_WRdata=Buffer_RX; // Pointeur d'écriture ds Buffer_RX
char *ptrRX_RDdata=Buffer_RX; // Pointeur de lecture ds Buffer_RX
```

Le programme d'interruption "U1RXInterrupt" est activé à la réception de chaque caractère. Celui-ci est rangé dans "Buffer_RX" avec le pointeur d'écriture "ptrRX_WRdata".
La boucle sans fin du programme principal appelle la fonction " Detect_Cmd_RxD". Celle-ci appelle à son tour la fonction "ReadRXD" qui lit le dernier caractère reçu dans "Buffer_RX" avec le pointeur "ptrRX_RDdata", s'il existe.
La fonction détecte alors une commande dans les 5 derniers caractères reçus et l'exécute.

Le format et la liste des commandes sont donnés au §1.11

Programme d'interruption "U1RXInterrupt"

```
/*********************************************************************
  Function:        void _ISR _U1RXInterrupt (void)
  Description :    Programme d'interruption déclenché à la réception
                   d'un caractère
*********************************************************************/
void _ISR _U1RXInterrupt (void)
{
 IFS0bits.U1RXIF=0; // Clear interrupt flag
 while (U1STAbits.URXDA)
 {
  *ptrRX_WRdata++=U1RXREG;
  if (ptrRX_WRdata==Buffer_RX+RX_BufSize) ptrRX_WRdata=Buffer_RX;
 }
}
```

Fonctions "ReadRXD" et "Detect_Cmd_RxD"

```
/*********************************************************************
  Function:        int ReadRXD(void)
  Description:     Lecture d'un octet dans le buffer de réception
                     - renvoi de 0 si aucun caractère reçu
                     - renvoi de 1 si caractère reçu :
                        - caractère placé dans "c"
                        - ptrRX_RDdata incrémenté modulo RX_BufSize
*********************************************************************/
```

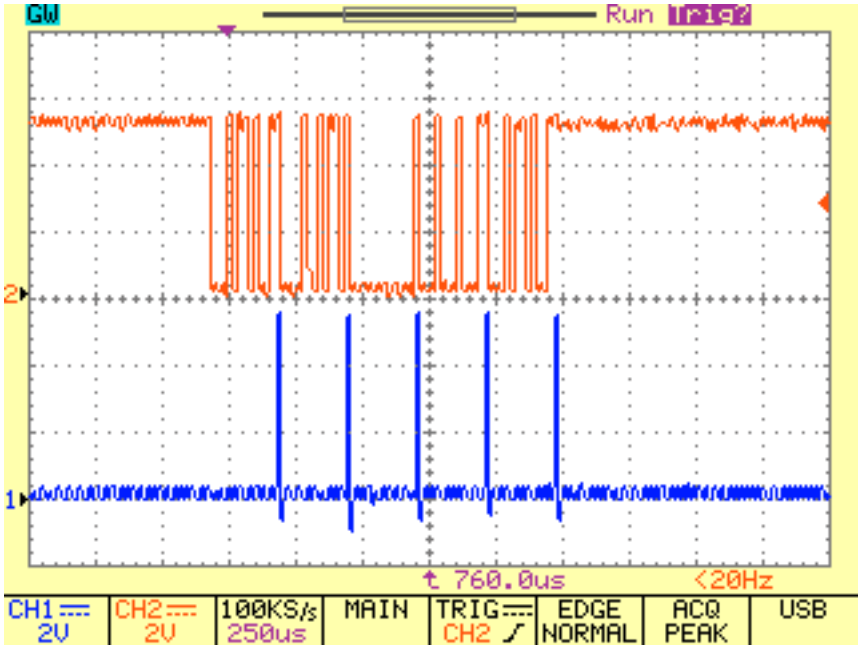
```
int ReadRXD(char *c)
{
 if (ptrRX_RDdata==ptrRX_WRdata) return(0); // Pas de caractère reçu
 else
 {
  *c=*ptrRX_RDdata++;
  if (ptrRX_RDdata==Buffer_RX+RX_BufSize) ptrRX_RDdata=Buffer_RX;
  return(1);
 }
}

/*****************************************************************
  Function:        void Detect_Cmd_RXD(void)
  Description:     Détection et exécution d'une éventuelle commande reçue
                   Les commandes ont toujours une longueur de 5 octets :
                             0xAA, code, Dh, Dl, 0xAA
                      - code : identifie la commande
                      - Dh, Dl : paramètre de la commande
*****************************************************************/
void Detect_Cmd_RXD(void)
{
 int i;
 char c;
 if (!ReadRXD(&c)) return;
 for (i=1; i<5; i++) Buffer_Cmd[i-1]=Buffer_Cmd[i];
 Buffer_Cmd[4]=c;
 if ((Buffer_Cmd[0]==0xAA) && (Buffer_Cmd[4]==0xAA))
 {
  switch(Buffer_Cmd[1]) // Code de la commande
  {
   case 'B' : // Avance/retard de phase (à multiplier par 16)
   {
    Offset_Angle=Buffer_Cmd[2]<<8;
    Offset_Angle+=Buffer_Cmd[3];
    Offset_Angle<<=4;
    break;
   }
   case 'C' : // Type de consigne (dans Buffer_Cmd[2])
   {
    switch(Buffer_Cmd[2])
    {
     case 'P' : // Poignée
     {
      Flags.Consigne=0;
      IFS0bits.ADIF=0;  // Raz indicateur interruption
      IEC0bits.ADIE=1;  // Validation interruption CAN
      break;
     }
     case 'T' : // Réglage "Tension" depuis PC via RS232
     {
      IEC0bits.ADIE=0;  // Inhibition interruption CAN
      IFS0bits.ADIF=0;  // Raz indicateur interruption
      Flags.Consigne=1;
      break;
     }
    }
    break;
   }
   case 'T' : // Consigne "Tension" (0 à 1023)
   {
    POT_Pos=Buffer_Cmd[2]<<8;
    POT_Pos+=Buffer_Cmd[3];
    break;
   }
```

```
   case 'P' : // Liaion avec PC
   {
    if (Buffer_Cmd[2]) Flags.PC_Connected=1;
    else
     {
      Flags.PC_Connected=0;
      Flags.Consigne=0; // Poignée
      IFS0bits.ADIF=0;  // Raz indicateur interruption
      IEC0bits.ADIE=1;  // Validation interruption CAN
     }
    break;
   }
  }
 }
}
```

2.12.4 Tests

La mise au point des programmes s'est faite avec le simulateur MPLAB.
Expérimentalement, on peut vérifier le temps de réaction du programme d'interruption "U1RXInterrupt" malgré les nombreux autres programmes d'interruption.
Ce test a été réalisé avec une version opérationnelle du programme et son contrôle depuis le PC. Pour obtenir un flux de réception, on manipule en permanence le réglage de "tension" sur le PC.
Pour observer la prise en compte du caractère reçu, le port RB0 est mis à "1" au début et remis à "0" à la fin de la fonction "U1RXInterrupt".

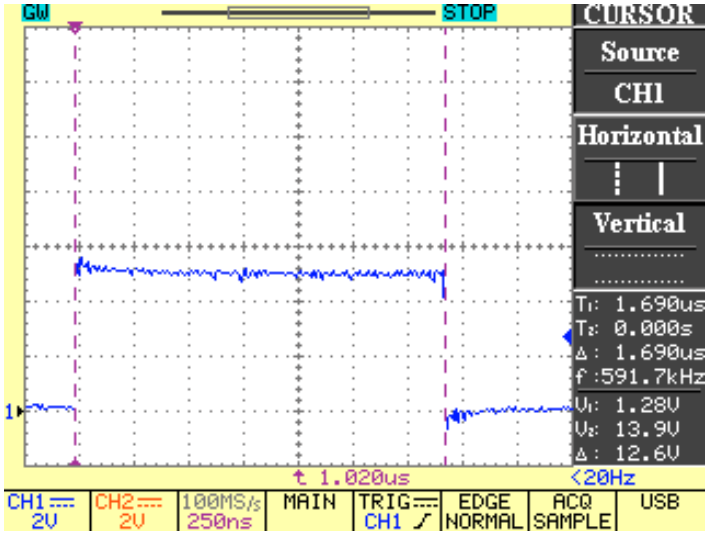


Acquire en mode "Détection crête"

U1RX

RB0

Conclusion : les 5 caractères de la commande reçue sont pris en compte sans retard appréciable.

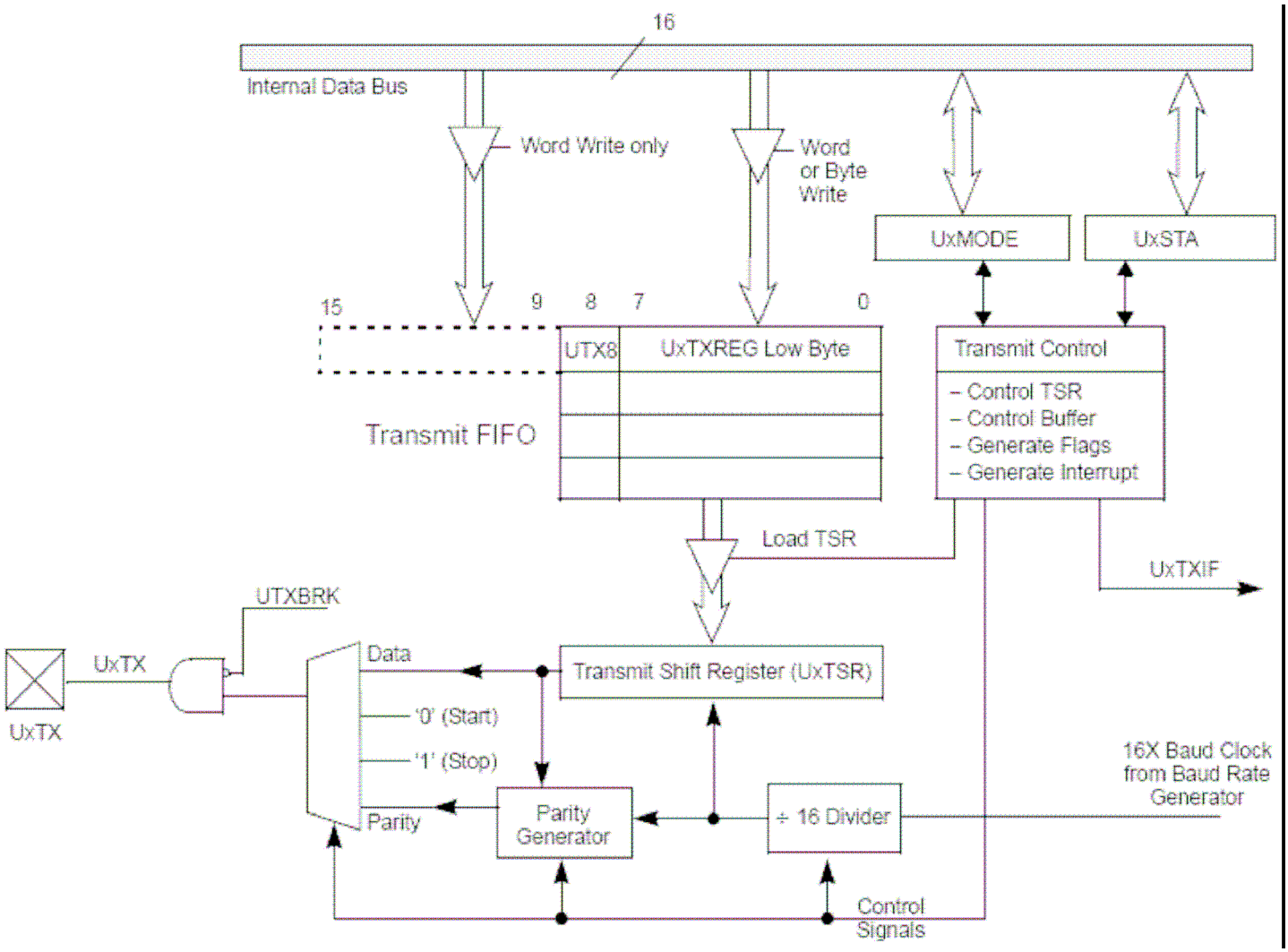


La durée du programme d'interruption est de l'ordre de 1,7µS ce qui ne devrait pas perturber les autres programmes d'interruption.

2.13 Transmit_Param

Cette fonction utilise l'UART du dsPIC30F2010. Ce coupleur est configuré au format suivant : 38400 bauds, pas de parité, 1 bit Stop, pas de poignée de main.
La fonction "Transmi_Param" est lancée par T1Interrupt à chaque interruption de "Timer 1", soit toutes les mS.

Schéma du module réception de l'UART



Le récepteur est équipé d'un registre FIFO d'une profondeur de 4 octets géré automatiquement.

2.13.1 Registres associés

Certains registres ont déjà été décrits au §2.12.1

Register 19-2: UxSTA: UARTx Status and Control Register

Upper Byte:

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL	—	—	—	UTXBRK	UTXEN	UTXBF	TRMT
bit 15							bit 8

Lower Byte:

R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/C-0	R-0
URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA
bit 7							bit 0

URXISEL à URXDA : voir §2.12.1
UTXISEL = 1 : interruption au transfert du dernier octet dans le tampon FIFO dans le registre de transmission
UTXBRK = 0 : pas de transmission de "Break" (TxD normal)
UTXEN = 1 : transmetteur validé : broche U1TX affectée à l'UART

UTXBF : indicateur de tampon FIFO plein
TRMT : indicateur : le registre de transmission et le tampon FIFO sont vides

Register 19-4: UxTXREG: UARTx Transmit Register (Write Only)

Upper Byte:							
U-0	U-0	U-0	U-0	U-0	U-0	U-0	W-x
—	—	—	—	—	—	—	UTX8
bit 15							bit 8

Lower Byte:							
W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
UTX<7:0>							
bit 7							bit 0

UTX8 : non utilisé ici
UTX : registre d'entrée du tampon FIFO

2.13.2 Fonction InitUART

Voir §2.12.2

2.13.3 Sources

De nombreuses sources d'interruptions sont utilisées dans cette application. Pour éviter de perdre des caractères, on a décidé d'utiliser le principe du buffer de transmission circulaire.
Rappel de la déclaration des constantes et variables associées :

```
// Buffers TX pour l'UART
#define TX_BufSize 32         // Taille du Buffer_TX
char Buffer_TX[TX_BufSize];   // Buffer circulaire de transmission
char *ptrTX_WRdata=Buffer_TX; // Pointeur d'écriture ds Buffer_TX
char *ptrTX_RDdata=Buffer_TX; // Pointeur de lecture ds Buffer_TX
unsigned char Buffer_Cmd[5];  // Buffer pour la réception des commandes
```

Les caractères à transmettre sont placés dans "Buffer_TX" en utilisant le pointeur "ptrTX_WRdata" par la fonction "Transmit_Param", via les fonctions "WriteTXD" (écriture d'un octet) et "WriteIntTXD" (écriture de 2 octets).

Une interruption est provoquée quand le registre d'entrée du tampon FIFO de l'UART se vide. Le programme "U1TXInterrupt" charge alors le prochain caractère à transmettre dans ce tampon FIFO depuis le buffer de transmission "Buffer_TX" en utilisant le pointeur "ptrTX_RDdata".
La transmission s'arrête naturellement quand "Buffer_TX" est vide.
Dans ces conditions, pour transmettre un nouveau caractère, la fonction "WriteTXD" déclenche une demande d'interruption en positionnant l'indicateur "U1TXIF" quand "Buffer_TX" est vide.

Le format et la liste des paramètres transmis sont donnés au §1.12

La fonction principale "Transmit_Param" fait appel aux fonctions
- "WriteRXD" qui attend une place libre dans le tampon FIFO et y place le caractère donné en paramètre
- "WriteIntTXD" qui utilise "WriteRXD" pour transmettre dans l'ordre le poids fort et le poids faible de l'entier 16 bits donné en paramètre

Programme d'interruption "U1TXInterrupt"

```
/*****************************************************************
  Function:        void _ISR _U1TXInterrupt (void)
  Description :    Programme d'interruption déclenché quand le buffer
                   de transmission de 4 octets est vide
*****************************************************************/
void _ISR _U1TXInterrupt (void)
{
 IFS0bits.U1TXIF=0; // Clear interrupt flag
 if (ptrTX_RDdata!=ptrTX_WRdata)
 {
  U1TXREG=*ptrTX_RDdata++;
  if (ptrTX_RDdata==Buffer_TX+TX_BufSize) ptrTX_RDdata=Buffer_TX;
 }
}
```

Fonctions WriteTXD, WriteIntTXD et Transmit_Param

```
/*****************************************************************
  Function:        void WriteTXD(char c)
  Description:     Ecriture d'un octet dans le buffer d'émission
                     - attendre libération U1TXREG
                     - envoi du caractère reçu
                     - ptrTX_WRdata incrémenté modulo TX_BufSize
*****************************************************************/
void WriteTXD(char c)
{
 int Nb_char;
 // Attendre libération du buffer
 do
  {
   // Calcul du nb de caractères restant à transmettre
   Nb_char=ptrTX_WRdata-ptrTX_RDdata;
   if (Nb_char < 0) Nb_char=Nb_char+TX_BufSize;
  }
 while (Nb_char > (TX_BufSize-2));
 // Caractère placé dans le buffer circulaire
 *ptrTX_WRdata++=c;
 if (ptrTX_WRdata==Buffer_TX+TX_BufSize) ptrTX_WRdata=Buffer_TX;
 // Provoquer la 1° interruption si 1° caractère
 if (Nb_char==0) IFS0bits.U1TXIF=1;
}
/*****************************************************************
  Function:        void WriteIntTXD(int i)
  Description:     Ecriture d'un entier dans le buffer d'émission
*****************************************************************/
void WriteIntTXD(int i)
{
 WriteTXD(i>>8); // Octet MSB d'abord
 WriteTXD(i);    // Puis octet LSB
}

/*****************************************************************
  Function:        void Transmit_Param (void)
  Descrition :     Lancé par T1Interrupt toutes les mS.
                   Traitement :
                    - Transmission des paramètres si Flags.PC_Connected=1
*****************************************************************/
```

```
void Transmit_Param (void)
{
 if (Flags.PC_Connected) switch (Compt_T1 & 0x007F) // Période = 128mS
  {
   case 0x00 : // Période secteur (entête = P)
    {
     WriteTXD(0xAA); WriteTXD('P');
     WriteIntTXD(Periode_H); WriteTXD(0xAA);
     break;
    }
   case 0x10 : // Indice (entête = I)
    {
     WriteTXD(0xAA); WriteTXD('I');
     WriteIntTXD(Indice_m); WriteTXD(0xAA);
     break;
    }
   case 0x20 : // Incrément d'angle (entête = A)
    {
     WriteTXD(0xAA); WriteTXD('A');
     WriteIntTXD(AngleInc); WriteTXD(0xAA);
     break;
    }
   case 0x30 : // Offset angle (entête = B)
    {
     WriteTXD(0xAA); WriteTXD('B');
     WriteIntTXD(Offset_Angle>>4); WriteTXD(0xAA);
     break;
    }
   case 0x40 : // Angle stator (entête = S)
    {
     WriteTXD(0xAA); WriteTXD('S');
     WriteIntTXD(Mag_angle_Stator>>13); WriteTXD(0xAA);
     break;
    }
   case 0x50 : // Angle stator (entête = S)
    {
     WriteTXD(0xAA); WriteTXD('S');
     WriteIntTXD(Mag_angle_Stator>>13); WriteTXD(0xAA);
     break;
    }
   case 0x60 : // Angle stator (entête = S)
    {
     WriteTXD(0xAA); WriteTXD('S');
     WriteIntTXD(Mag_angle_Stator>>13); WriteTXD(0xAA);
     break;
    }
   case 0x70 : // Angle stator (entête = S)
    {
     WriteTXD(0xAA); WriteTXD('S');
     WriteIntTXD(Mag_angle_Stator>>13); WriteTXD(0xAA);
     break;
    }
  }
}
```

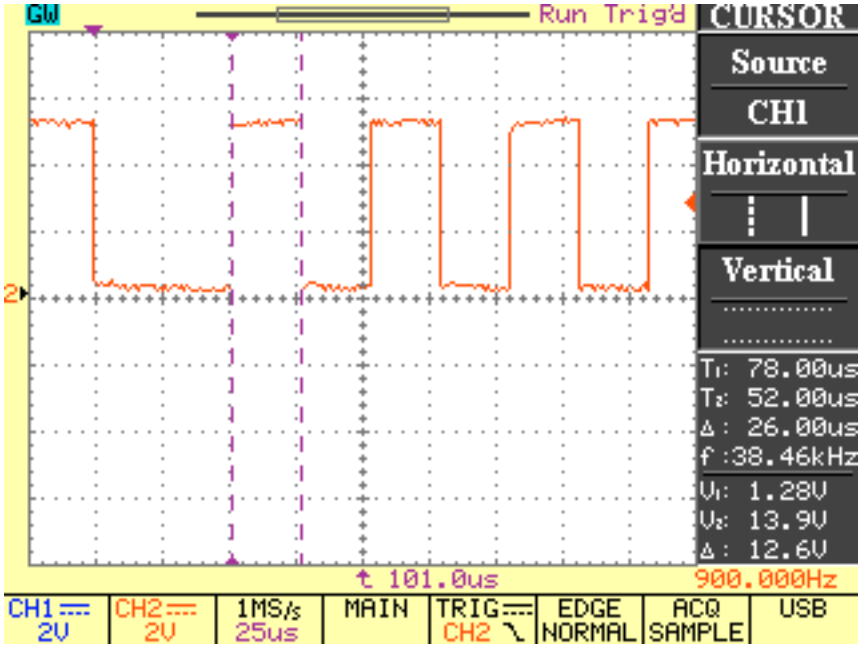
Commentaires :

- L'ensemble des paramètres est transmis toutes les 128mS. Pour ce faire, une variable est utilisée comme compteur d'interruptions : "Compt_T1" (voir "T1Interrupt" au §2.1). Les bits 6 à 0 sont utilisés pour identifier 8 instants espacés de 128/8 = 16mS. Un paramètre différent est transmis à chacun de ces instants. Cela permet de repartir les tâches dans le temps.
- Certains paramètres sont codés sur 32 bits, mais le format ne permet que 16 bits. Dans ce cas, une mise en forme est nécessaire pour transmettre les 16 bits les plus significatifs.

2.13.4 Tests

La mise au point des programmes s'est faite avec le simulateur MPLAB.

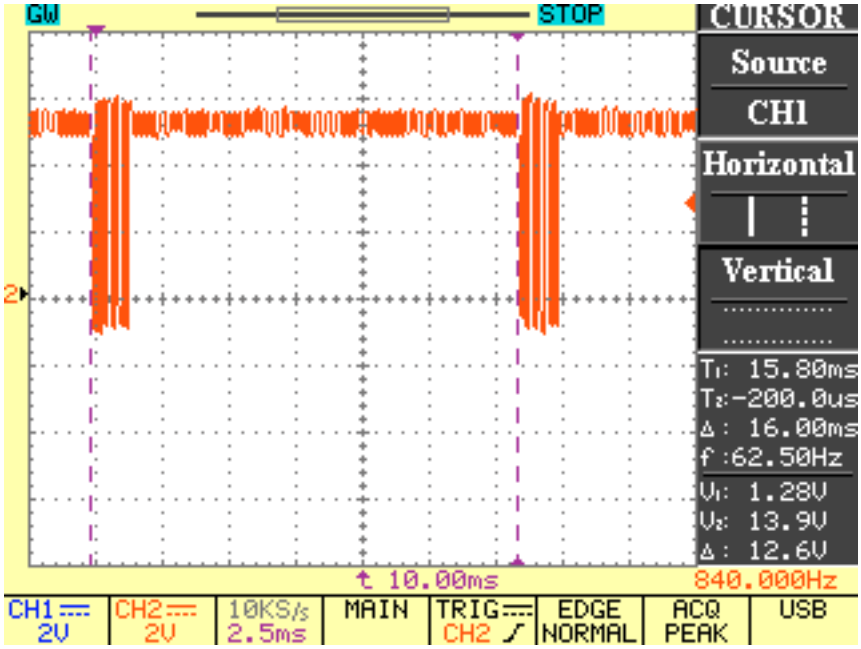
Format



U1TX

Le premier caractère transmis est toujours 0xAA
On en déduit facilement le débit : 38,48k bauds. Conforme.

Périodicité des transmissions

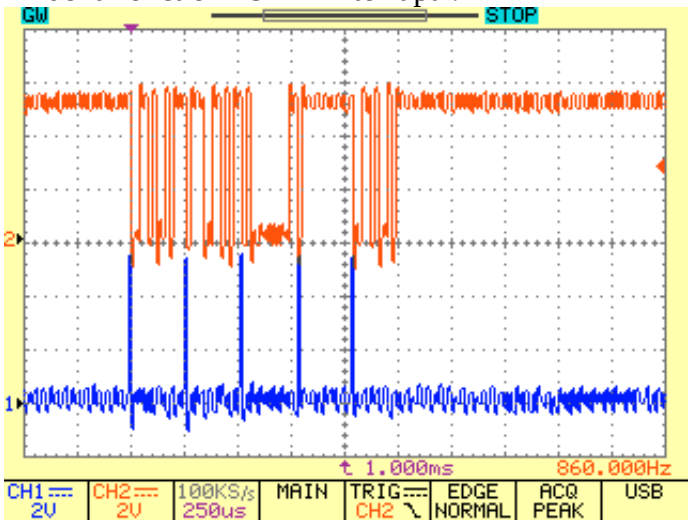


Acquire en mode "Détection crête"

U1TX

L'intervalle entre 2 transmission est bien de 16mS

Expérimentalement, on peut vérifier le temps de réaction du programme d'interruption "U1TXInterrupt" malgré les nombreux autres programmes d'interruption.
Ce test a été réalisé avec une version opérationnelle du programme et son contrôle depuis le PC.
Pour observer la prise en compte du caractère reçu, le port RB0 est mis à "1" au début et remis à "0" à la fin de la fonction "U1TXInterrupt".

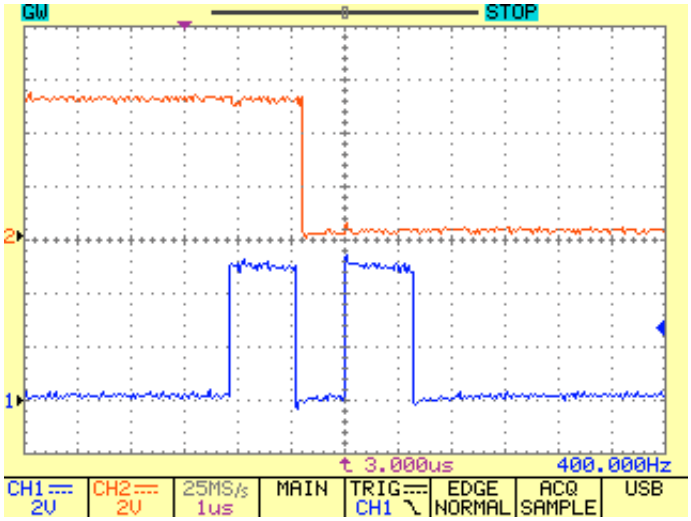


U1TX

RB0

Les 5 caractères d'une commande sont transmis sans retard appréciable

1° caractère d'une commande



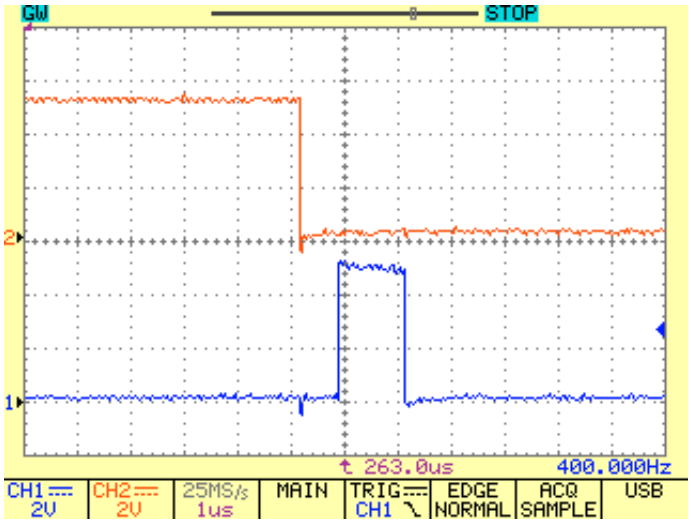
Mémoire : 25000 échantillons

U1TX

RB0

Le programme d'interruption est activé 2 fois de suite. C'est normal :

2° caractère d'une commande



Mémoire : 25000 échantillons

U1TX

RB0

Le programme d'interruption est bien appelé qu'une seule fois et très rapidement après le bit Start (moins d'une µS).
Sa durée est faible et ne devrait pas perturber les autres programmes

Schéma de la maquette