01/01/2017

Pierre FLODROPS GE5A

Polytech Clermont-Ferrand – Génie Electrique

Utilisation du module High Speed ADC enclenché par PWM sur microcontrôleur PIC33FJ06GS101A

Note d’application

Table des matières

[Introduction 2](#_Toc472612837)

[Programmation sur microcontrôleur 3](#_Toc472612838)

[Développement 3](#_Toc472612839)

[Architecture du microcontrôleur PIC33FJ06GS101A 3](#_Toc472612840)

[Utilisation de l’oscillateur auxiliaire 4](#_Toc472612841)

[Paramétrage de l’oscillateur auxiliaire 8](#_Toc472612842)

[Génération de signaux MLI 8](#_Toc472612843)

[Conversion Analogique Numérique 9](#_Toc472612844)

[Paramétrage convertisseur analogique numérique 11](#_Toc472612845)

[Conclusion 12](#_Toc472612846)

Table des figures

[Figure 1: Type d’architecture de microcontrôleur contenant plusieurs périphériques 3](#_Toc472613199)

[Figure 2: Diagramme des blocs fonctionnels du microcontrôleur PIC33FJ06GS101A 4](#_Toc472613200)

[Figure 3: Schéma du réseau d'horloges du PIC33FJ06GS101A 5](#_Toc472613201)

[Figure 4: Formule mathématique permettant de déterminer la fréquence en sortie de l'oscillateur auxiliaire 6](#_Toc472613202)

[Figure 5: Registre ACLKCON permettant le paramétrage de l'oscillateur auxiliare 7](#_Toc472613203)

[Figure 6: Fonctionnement interne du déclenchement du convertisseur 9](#_Toc472613204)

[Figure 7: Architecture du convertisseur A/N 10](#_Toc472613205)

# Introduction

Dans le cadre du projet de fin d’étude s’étalant sur la quatrième et cinquième année en Génie Electrique à Polytech Clermont-Ferrand, il est demandé aux élèves ingénieurs de rédiger un document technique détaillant un point précis du projet sur lequel les étudiants ont travaillé. L’objectif de ce travail est d’établir un document de référence, réutilisable et exhaustif.

Le sujet abordé dans cette note d’application concerne l’utilisation, sur microcontrôleur, des modules dit *High speed PWM and ADC* (Génération de MLI et CAN haute vitesse). Ces types de modules ont été utilisés dans la réalisation du Projet « Acquisition Rapide Multivoies » mené sur l’année 2016-2017. Ce projet a pour but de convertir (conversion analogique numérique) de façon simultanée et à une fréquence d’au moins 500 000 kHz quinze signaux caractéristiques d’un moteur électrique triphasé. Une fois ces conversions réalisées, les résultats sont mémorisés afin de pouvoir afficher sur un graphique l’évolution des signaux acquis et ainsi déterminer le bon fonctionnement du moteur électrique sur lequel ont été prélevés les signaux.

Afin de réaliser la fonction de conversion analogique numérique à la fréquence désirée, il a été décidé de retenir une solution à base de microcontrôleurs possédant un module de conversion Analogique Numérique haute vitesse. Ce module de conversion fonctionne de pair avec un module de génération de signaux MLI. Il est exposé, dans ce document, la marche à suivre permettant d’utiliser ces modules.

Dans une première partie, il est rappelé succinctement le principe de la programmation sur microcontrôleur avant de s’intéresser aux trois étapes permettant la conversion analogique numérique sur microcontrôleur PICFJ06GS101A. Ces trois étapes sont, dans l’ordre, le paramétrage de l’oscillateur auxiliaire, la génération de signal MLI comme signal de trigger pour le convertisseur analogie numérique dont le paramétrage constitue la dernière partie de ce document.

# Programmation sur microcontrôleur

Un microcontrôleur est un composant électrique constitué, dans un même boitier, d’un microprocesseur, de mémoire et de périphériques. Il existe un nombre très important de périphériques pouvant être implantés dans un microcontrôleur. Ils permettent de réaliser plusieurs fonctions de base qui assemblées donnent au microcontrôleur une fonction complexe. C’est en cela que les microcontrôleurs sont devenus une clé incontournable dans tous les systèmes automatisés, notamment dans l’industrie. L’avantage de ce composant appartenant à la logique programmée est qu’il peut réaliser une multitude de fonctions que l’utilisateur définit lui-même en fonction de son projet, contrairement aux composants à logique câblée qui en fonction de comment ils sont réalisés ne peuvent supporter qu’une seule fonction prédéfinie.



Figure 1: Type d’architecture de microcontrôleur contenant plusieurs périphériques

L’étape de programmation de ces puces électroniques nécessite une chaine de développement (logiciel, compilateur, boot loader) propre à chacun des constructeurs. Elle fait également appel à différents types de langages de programmation informatique qui permettent de définir une succession d’étapes logiques donnant au microcontrôleur sa fonctionnalité.

# Développement

## Architecture du microcontrôleur PIC33FJ06GS101A

Le microcontrôleur Microchip PIC33FJ06GS101A auquel nous nous intéressons est un microcontrôleur à architecture 16 bits, conçu notamment pour le control de moteurs électriques. En effet, il possède un processeur de signal numérique, en anglais « Digital Signal Processeur » (DSP), permettant d’exécuter des applications de traitement numérique du signal le plus rapidement possible et permettant ainsi un control moteur optimal.

La figure 2 suivante représente l’architecture interne du microcontrôleur.



Figure 2: Diagramme des blocs fonctionnels du microcontrôleur PIC33FJ06GS101A

## Utilisation de l’oscillateur auxiliaire

Le caractère haute vitesse des modules de signaux MLI et convertisseur Analogique Numérique résident dans le fait que ces deux périphériques sont cadencés par un oscillateur auxiliaire constitué d’une PLL permettant de multiplier jusqu’à un facteur 16 une fréquence d’entrée et qui permet donc d’obtenir une fréquence de fonctionnement des modules très importante.

La figure **3** ci-après montre le schéma du circuit d’horloge permettant de déterminer quels sont les différents signaux pouvant être utilisés pour cadencer le microcontrôleur ainsi que l’architecture de l’oscillateur auxiliaire servant à imposer une fréquence de travail aux modules de signaux MLI et convertisseur analogique numérique.



Figure 3: Schéma du réseau d'horloges du PIC33FJ06GS101A

Il est possible de définir différents signaux d’horloges comme signaux source de l’oscillateur auxiliaire. Il est possible d’utiliser, dans un premier temps les signaux POSCCLK, issu de l’oscillateur primaire ou FRCCLK, issu d’un oscillateur RC interne ayant pour fréquence nominale 7.37MHz. Grâce à la PLL de l’oscillateur auxiliaire, la fréquence de ces deux signaux peut être multipliée par un facteur 16. Dans un deuxième temps, un autre signal d’horloge, Fvco issu de la PLL de l’oscillateur primaire peut être défini comme signal source de l’oscillateur auxiliaire. Une fois le signal d’horloge source choisit, l’oscillateur possédant un diviseur de fréquence, il est possible de diviser ce signal afin qu’il puisse cadencer les modules de signaux MLI et convertisseur à la fréquence désirée.

La figure 4 suivante met en évidence la formule permettant de déterminer la fréquence ACLK en sortir de l’oscillateur auxiliaire en fonction d’une fréquence d’entrée REFCLK :



Figure 4: Formule mathématique permettant de déterminer la fréquence en sortie de l'oscillateur auxiliaire

Le registre permettant l’utilisation de l’oscillateur auxiliaire est le registre ACLKCON. Il est constitué tel que le montre la figure 5 suivante :



Figure 5: Registre ACLKCON permettant le paramétrage de l'oscillateur auxiliare

### Paramétrage de l’oscillateur auxiliaire

Ci-après est présenté un code détaillé en langage C d’une fonction my\_Init\_aux\_clk() permettant le paramétrage de l’oscillateur auxiliaire :

 1 #include "system\_header.h"

 2 #include<xc.h>

 3

 4 void my\_Init\_aux\_clk (void){

 5

 6 /\* Auxilary Clock configuration\*/

 7 ACLKCONbits.ASRCSEL = 0; ACLKCONbits.FRCSEL = 1; // Select Reference Clock source for Aux Clock: FRCCLK is the source clock

 8 ACLKCONbits.ENAPLL = 1; // Auxiliary clock disabled for initialization

 9 ACLKCONbits.SELACLK = 1; // Auxiliary oscillators provides the source clock for auxiliary clock divider

10 ACLKCONbits.APSTSCLR = 0b111; // N : Auxiliary clock divider : div 1

11 while(ACLKCONbits.APLLCK != 1){}; /\* Wait for Auxiliary PLL to Lock \*/

12 }

Les deux instructions de la ligne 7 permettent de configurer les bits de sélection des deux multiplexeurs en amont de la PLL de l’oscillateur auxiliaire. L’association des deux permet dans le cas présent de sélectionner le signal d’horloge FRCCLK comme étant la source de la PLL.

L’instruction des lignes 8 et 9 permettent de paramétrer les bits de sélection des deux multiplexeurs en amont du diviseur de fréquence. Dans le cas présent, le signal choisi est le signal FRCCLK issu de la PLL de l’oscillateur auxiliaire.

La ligne 10 permet de sélectionner le facteur de division du signal d’entrée du diviseur. Dans le cas présent, la combinaison choisie (0b111) permet de sélectionner un facteur de division égale à 1.

Enfin, une fois que toutes les configurations précédentes sont effectuée, la dernière chose à faire est d’attendre que la PLL soit verrouillée et stable. C’est le rôle de la ligne 11 qui permet d’attendre que le bit indiquant le verrouillage de la PLL passe à l’état haut.

## Génération de signaux MLI

En plus de pouvoir généraux des signaux de type MLI, le module PWM peut également générer un signal de déclenchement « trigger » interne pour enclencher une conversion permettre en interne par le convertisseur Analogique Numérique. Ceci permet de définir une fréquence de conversion fixe et d’avoir un résultat de conversion à des temps déterminés.

Sur le microcontrôleur utilisé, le module de PWM possède de sorties et offre la possibilité de déclencher deux signaux de trigger pour deux valeurs du compteur. Ceci permet une optimisation du temps de conversion puisque sur une période de PWM, il est possible de déclencher deux signaux de trigger.

En tenant compte de cela, les registres à configurer sont les registres suivant :

* PTPER : Permet de définir la période du signal MLI et donc du signal de trigger. Registre 16 bits
* TRGCON1 : Permet d’inclure un temps d’attente (bits TRGSTRT), un facteur permettant de diviser la fréquence du signal de trigger et de combiner (bits TRGDIV) ou non les deux signaux de trigger pouvant être généré par le module PWM (bit DTM).
* TRIG1 : Valeur du premier compteur comparé avec le compteur de PWM pour le déclenchement du premier signal de trigger. Registre 16 bits
* STRIG1 : Valeur du second compteur comparé avec le compteur de PWM pour le déclenchement du second signal de trigger. Registre 16 bits n

La figure 6 suivante permet de comprendre comment sont utilisés les registres explicités ci-dessus en interne du module de PWM.



Figure : Fonctionnement interne du déclenchement du convertisseur

## Conversion Analogique Numérique

Le module de conversion Analogie Numérique du microcontrôleur utilisé possède plusieurs types de circuits échantillonneur/bloqueur. Un type dit dédié et un autre dit partagé. Les circuits dédiés sont réservés à une seule entrée analogique et permet ainsi un échantillonnage quasi instantané. A l’opposé, le circuit partagé permet à plusieurs entrées analogiques d’être échantillonnées. Dans ce dernier cas, il est important de gérer les accès des différentes entrées afin qu’il n’y ait pas de conflit. La figure 6 suivante montre la répartition des différentes entrées analogiques du convertisseur sur les circuits bloqueur échantillonneur. Les deux types de circuit se partagent en revanche un seul module de conversion.



Figure : Architecture du convertisseur A/N

Afin d’utiliser le module de conversion analogique numérique, le réglage des registres suivants est nécessaires :

* ADCON
* ADSTAT
* ADBASE
* ADPCFG
* ADCPC

L’utilisation de ces registres est détaillée dans le programme suivant. Il s’agit d’un fonction my\_Init\_adc() permettant de configurer le convertisseur afin que celui-ci soit cadencé par l’oscillateur auxiliaire (réglé précédemment) que chaque conversion soit démarrée par un front montant du signal de MLI (paramétré auparavant) et que la fin d’une conversion déclenche une interruption permettant de récupérer le résultat de la conversion.

### Paramétrage convertisseur analogique numérique

 1 #include "adc\_header.h"

 2 #include <adc.h>

 3

 4 void my\_Init\_adc(){

 5 int result =0;

 6 /\* Only AN0 paring with AN1 is used for the conversion \*/

 7

 8 // AN0 set as Input

 9 TRISAbits.TRISA0 = 1;

10

11 // ADCON: ADC CONTROL REGISTER

12 ADCONbits.ADON = 0; // ADC module is disable during init

13 ADCONbits.ADSIDL = 0 ; // Continues module operation in Idle mode

14 ADCONbits.SLOWCLK = 1; // ADC is clocked by the auxiliary PLL (ACLK)

15 ADCONbits.FORM = 0; //Interger Format Chosen

16 ADCONbits.EIE= 1; // Interrupt is generated after first conversion is completed in the pair

17 ADCONbits.ORDER = 0; // Even (pair) numbered converted first.

18 ADCONbits.SEQSAMP = 0; // Shared S&H is sampled at the same time the dedicated S&H is sampled if the shared S&H is not currently busy with an existing conversion process.

19 ADCONbits.ASYNCSAMP = 1; // The dedicated S&H is constantly sampling and then terminates sampling as soon as the trigger pulse is detected

20 ADCONbits.ADCS = 0b000; // Convresion Clock Divider. no divider

21

22 // ASTAT: ADC STATUS REGISTER

23 /\* Nothing to be done\*/

24 ADSTAT = 0x0000;

25

26 //ADBASE: ADC BASE REGISTER

27 /\* Nothing to be done \*/

28 ADBASE = 0x0000;

29

30 //ADPCFG: ADC PORT CONFIGURATION REGISTER

31 ADPCFG = 0xFFFF; // All other pins are digital pin

32 ADPCFGbits.PCFG0 = 0; // AN0 configured as Analog Input

33

34 //ADCPCO0: ADC CONVERT PAIR CONTROL REGISTER 0

35 ADCPC0bits.IRQEN0 = 1; // Enables IRQ generation when requested conversion of channels AN1 and AN0 is completed

36 ADCPC0bits.TRGSRC0 = 0b00100; // PWM Generator 1 primary trigger is selected

37

38 /\* ADC Pair0 Interrupt \*/

39 IFS6bits.ADCP0IF = 0; // Clear ADC Flag Interrupt

40 IPC27bits.ADCP0IP = 0b101; // ADC Pair 0 Conversion Done Priority. Level 5/7

41 IEC6bits.ADCP0IE = 1; // ADC Pair 0 Interrupt Enable

42

43 ADCONbits.ADON = 1; //Start ADC after initialization done

44 }

45

46 void my\_Switch\_on\_ADC(){

47 ADCONbits.ADON = 1;

48 }

49

50 void my\_Switch\_off\_ADC(){

51 ADCONbits.ADON = 0;

52 }

53

54 void \_\_attribute\_\_((interrupt,no\_auto\_psv)) \_ADCP0Interrupt(void){

55

56 if(IFS6bits.ADCP0IF){ // Check if the interrupt flag is really set

57 result = ADCBUF0; // Read the result of the last conversion

58 IFS6bits.ADCP0IF = 0; // Clear ADC pair 0 flag

59 }

60 }

# Conclusion

Nous avons pu voir dans ce tutoriel les différentes étapes et les registres à configurer dans le PIC33FJ06101A permettant de convertir un signal analogique en échantillons numériques en utilisant les différents modules proposés par le microcontrôleur. Bien que la fonctionnalité décrite soit utile notamment pour le projet d’acquisition rapide multivoies, il est possible de réutiliser ce type d’application pour tout autre système nécessitant des performances de conversion importantes.