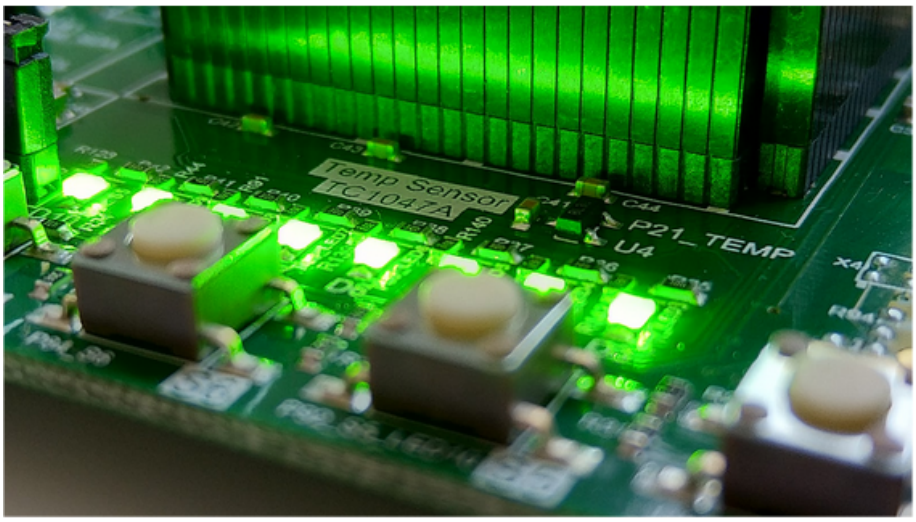
Projet de Physique P6
STPI/P6/2019 - 41
Programmation informatique embarquée : Initiation au Microprocesseur

Nathan GOUJON
Victor LESUR
Souhail NOUREDDINE
Lucas PREEL
Jérôme SAIVRES
Clémence STRZEPEK

Juin 2019



A l'intention de Mr Richard Grisel

RÉSUMÉ

Date de remise du rapport : 17/06/2019

Référence du projet : STPI/P6/2019 – 41

Intitulé du projet : Programmation informatique embarquée : Initiation au Microprocesseur

Type de projet : expérimental / simulation

Objectifs du projet : Les objectifs de ce projet sont :
→ De comprendre le fonctionnement général d'un microprocesseur (PIC32) à travers plusieurs expériences et simulations. Pour cela, nous avions à disposition le logiciel MPLAB X IDE.
→ De s'initier à la programmation au langage C.
→ De traiter des signaux à l'aide du microprocesseur.

Mots-clefs du projet : microprocesseur, programmation, mplab

Table des matières

1 Introduction

Ce projet a eu pour but de nous initier à la compréhension puis à l'utilisation des microprocesseurs ainsi que de nous familiariser au logiciel MPLAB. Les microprocesseurs ont révolutionné le monde de l'informatique grâce à la miniaturisation des composants qui les rend moins encombrants qu'un processeur traditionnel, et donc plus facilement utilisables dans des systèmes embarqués. On les retrouve donc dans plusieurs domaines : ordinateurs, calculatrices, jeux, robotiques, télévisions, etc. Mais que se passe-t-il dans un microprocesseur ? Que permet-il de faire ?

Pour notre part, nous avons utilisé une carte PIC32 et non une DSP, sujet traité par le second groupe. La différence est faible : on la trouve notamment dans l'encodage des bits.

Une lecture préalable de la documentation fournie avec notre microprocesseur s'est révélée indispensable, même si quelque peu indigeste, pour comprendre ses fonctionnalités. Nous avons ensuite pu réaliser plusieurs manipulations avec le PIC32.

Dans ce rapport nous allons présenter les connaissances que nous avons pu acquérir sur le fonctionnement des microprocesseurs ainsi que les résultats obtenus lors des diverses expériences que nous avons effectuées, ces dernières nous ayant permis de mieux appréhender les différentes possibilités d'utilisation de ce type de matériel informatique.

2 Méthodologie / Organisation du travail

Lors de chaque séance nous avons réalisé deux types de travaux qui sont : la compréhension de code C et la réalisation de manipulation avec le microprocesseur. C'est pourquoi nous avons naturellement choisi de diviser notre groupe en deux avec un ordinateur et un microprocesseur par demi-groupe. Ainsi nous avons pu réaliser chaque manipulation "en double" et donc valider nos résultats et s'entraider quand nous étions en difficulté. En effet, la plupart d'entre nous n'ayant jamais programmé en C, il nous arrivait souvent de bloquer lorsqu'il s'agissait de coder.

Concernant la rédaction du rapport, nous avons tout d'abord élaboré le plan général ensemble en faisant le bilan des séances réalisées avec Mr Grisel. Ensuite, lorsque nous étions en autonomie (5 dernières séances), un demi-groupe a commencé la rédaction du rapport pendant que l'autre terminait la dernière manipulation. Enfin nous avons réparti les différents paragraphes à rédiger en fonction de ce que chacun avait le mieux compris et affectionnait le plus. Ainsi lors de notre dernière séance nous avons réalisé ensemble le poster et clôturé la rédaction de notre rapport de projet.

Enfin, nous avons, dans un grand moment de convivialité, pris une photo de groupe que nous ne manquerons pas de joindre à notre poster.

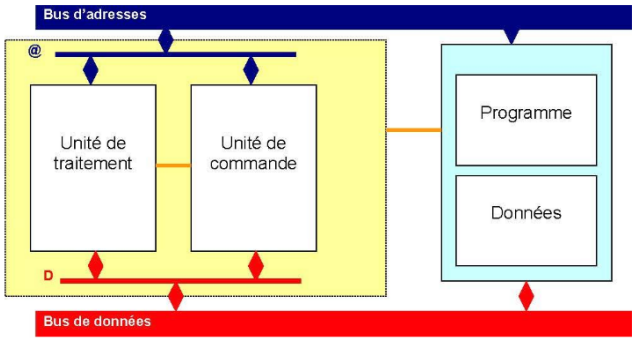
3 Travail réalisé et résultats

3.1 Travail de documentation

3.1.1 Qu'est ce qu'un microprocesseur ?

Les premiers ordinateurs réalisés dans les années 1950 fonctionnaient avec des tubes à vide, l'unité centrale en contenait plusieurs dizaines de milliers. Par la suite, l'invention du transistor, ainsi que celle du circuit intégré et notamment du microprocesseur dans les années 1970 par Intel, ont permis de révolutionner les recherches. En effet, l'invention des microprocesseurs, nouveaux centres de commande des ordinateurs, les a rendu considérablement plus petits et bien plus performants que ceux des années 1950. Tous les composants qui constituent un processeur ont été soudés sur un seul circuit intégré, le microprocesseur. Ceci a permis un gain de rapidité, une réduction des coûts grâce au remplacement de plusieurs circuits par un seul mais aussi un gain de fiabilité en supprimant le risque principal de panne (i.e. les connexions entre les composants du processeur).

Un microprocesseur est une puce de silicium de quelques millimètres de côté supportant un circuit intégré, qui a pour rôle de lire l'instruction en mémoire, la décoder, l'exécuter et recommencer le même procédé pour l'instruction suivante. Pour cela, il est notamment composé d'une unité de commande, d'une unité de traitement, de registres qui stockent temporairement les données et de bus qui transmettent les instructions sous forme de signaux pour échanger l'information.



L'unité de commande a pour mission principale de rechercher en mémoire l'instruction codée sous forme binaire et d'effectuer son décodage pour enfin réaliser son exécution.

L'unité de traitement joue un rôle crucial dans le fonctionnement du microprocesseur. C'est elle qui assure les traitements nécessaires à l'exécution des instructions.

Il existe également d'autres unités :

→ l'unité de cache mémoire : stocke les informations les plus récurrentes pour gagner en rapidité d'accès, la mémoire cache étant beaucoup plus rapide que la mémoire centrale ;

→ l'unité d'interface de bus : crée des échanges de données entre le microprocesseur et d'autres composantes comme la mémoire vive par exemple ;
→ l'unité de décodage : permet de décomposer et analyser l'instruction du registre d'instructions.

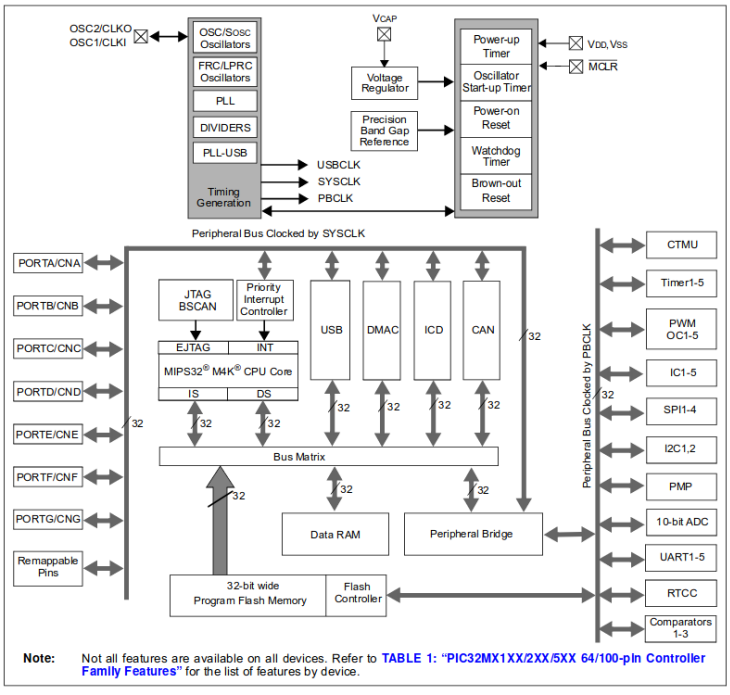
Les registres du microprocesseur sont de petites zones mémoires dans le microprocesseur, permettant de mémoriser temporairement des informations comme des adresses.

De plus, il nous a semblé important de vous présenter le processus d'exécution d'une instruction.

1. Recherche de l'instruction à traiter : l'instruction à traiter contenue dans l'ordinateur est placée dans le bus d'adresses par l'unité de commande, puis les données contenues à cette adresse sont placées dans le bus de données , l'instruction est alors stockée dans le registre instruction ;
2. Décodage : l'unité de commande transforme l'instruction en une suite de commandes permettant de la traiter ; dans le cas où l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande récupère la valeur sur le bus de données ;
3. Exécution : le programme est exécuté, l'unité de commande se prépare pour l'instruction suivante.

Enfin, nous présentons ci-dessous l'architecture du PIC32 sur laquelle on retrouve les éléments exposés précédemment :

FIGURE 1-1: PIC32MX1XX/2XX/5XX 64/100-PIN BLOCK DIAGRAM

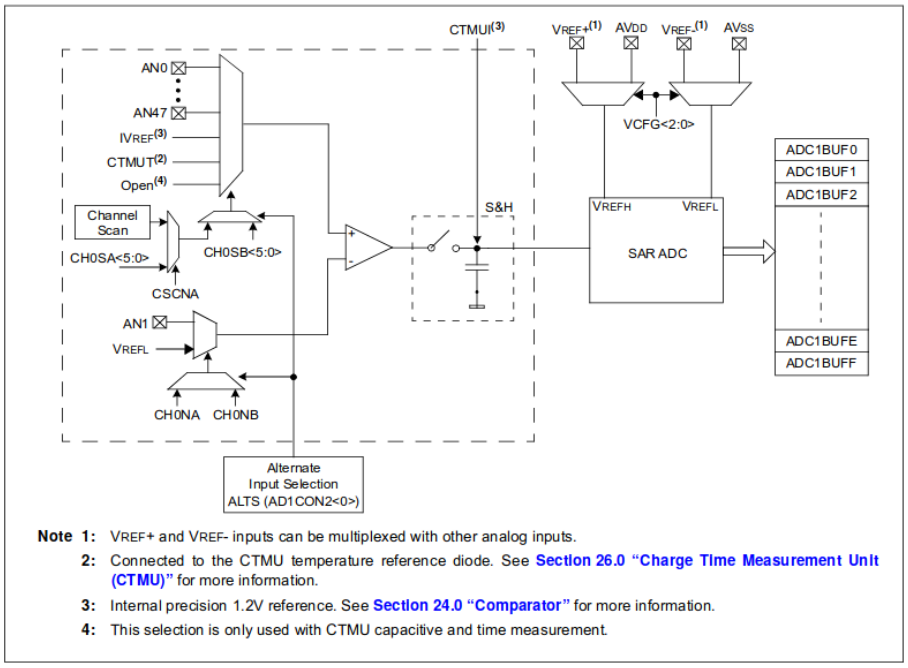


3.1.2 Analog-to-Digital Converter (ADC)

Un convertisseur analogique-numérique (ADC) est un montage électronique permettant de convertir une grandeur analogique en une valeur numérique.

Le schéma ci-dessous illustre le fonctionnement de l'ADC 10bits. Ce dernier peut avoir jusqu'à 16 broches d'entrées analogiques (de AN0 à AN15). Il existe également deux broches d'entrées analogiques pour les connexions externes.

FIGURE 22-1: ADC1 MODULE BLOCK DIAGRAM



Note 1: VREF+ and VREF- inputs can be multiplexed with other analog inputs.
2: Connected to the CTMU temperature reference diode. See Section 26.0 "Charge Time Measurement Unit (CTMU)" for more information.
3: Internal precision 1.2V reference. See Section 24.0 "Comparator" for more information.
4: This selection is only used with CTMU capacitive and time measurement.

3.1.3 Timers

Le timer joue le rôle de "chef d'orchestre" lors du déroulement d'un programme. En effet, grâce aux interruptions qu'il génère, les différentes opérations du programme sont ordonnées temporellement.

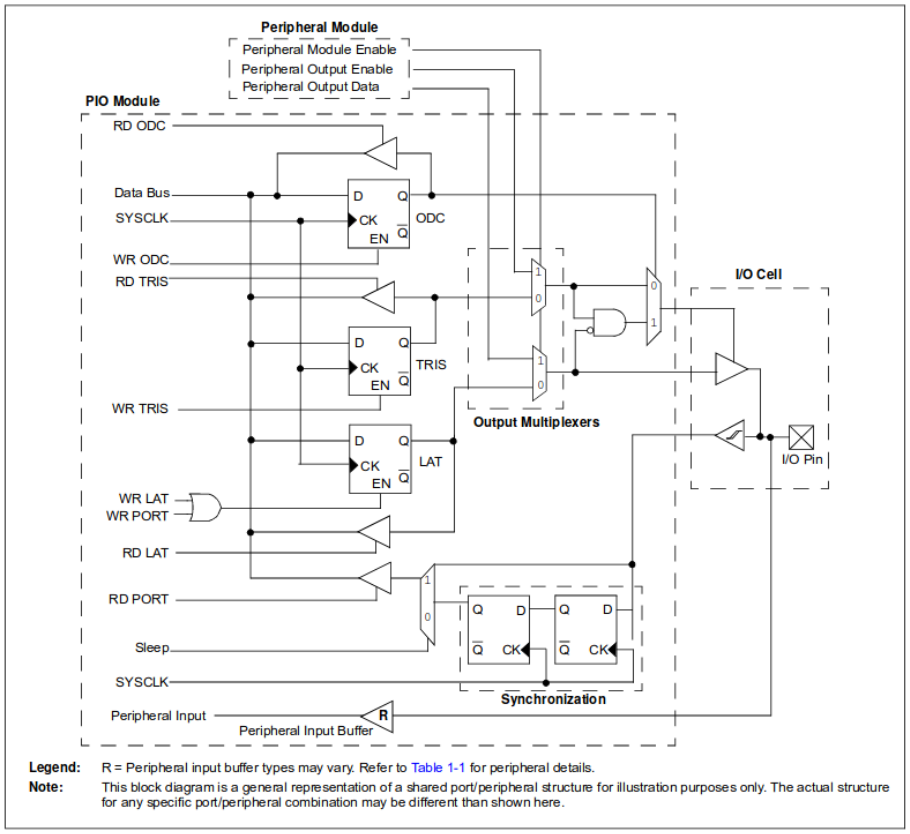
Le PIC32 peut avoir deux types de timers différents, en fonction du modèle utilisé : type A ou type B. Les caractéristiques et les différences des deux types sont listées ci-dessous.

Available Timer Types	Secondary Oscillator	Asynchronous External Clock	Synchronous External Clock	16-bit Synchronous Timer/Counter	32-bit Synchronous Timer/Counter (see Note 1)	Gated Timer	Special Event Trigger
Type A	Yes	Yes	Yes	Yes	No	Yes	No
Type B	No	No	Yes	Yes	Yes	Yes	Yes

3.1.4 I/O ports

Les entrées-sorties du microprocesseur sont appelées I/O ports. Elles permettent d'effectuer les échanges d'informations entre le processeur et les périphériques associés.

FIGURE 11-1: BLOCK DIAGRAM OF A TYPICAL MULTIPLEXED PORT STRUCTURE



Legend: R = Peripheral input buffer types may vary. Refer to Table 1-1 for peripheral details.
Note: This block diagram is a general representation of a shared port/peripheral structure for illustration purposes only. The actual structure for any specific port/peripheral combination may be different than shown here.

3.2 MPLAB

MPLAB est un IDE (Integrated Development Environment) développé par la société Microchip qui permet de développer des applications pour les cartes du même nom. Il présente une interface graphique permettant d'améliorer l'espace de travail à l'aide de nombreuses fonctionnalités. Il possède deux modes de développement distincts qui correspondent chacun à une phase de l'élaboration du programme. Ainsi, dans un premier temps, MPLAB, comme tout IDE, permet l'écriture du code, en C ou en assembleur, et divise les fichiers en une arborescence propre à son mode de fonctionnement en "projets".

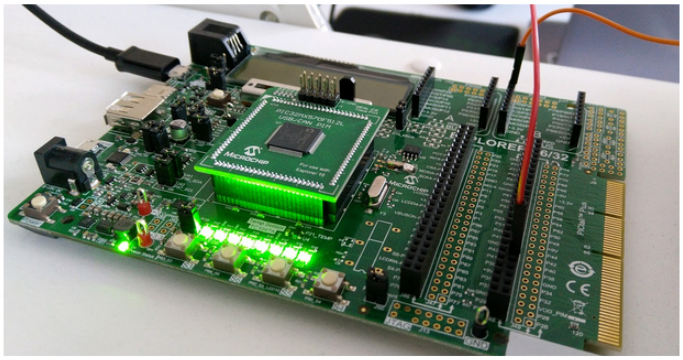
Outre ces caractéristiques, MPLAB fournit un mode Simulation, qui constitue son principal intérêt. En effet, il permet de tester son code dans un environ-

nement virtuel protégé sans risque d'endommager la carte en situation réelle. De ce fait, l'utilisateur a accès à de nombreuses options permettant de paramétrer les variables de son choix comme la fréquence d'échantillonnage et les valeurs initiales des variables. Une fois que le code a été testé et considéré comme fonctionnel, nous avons la possibilité de par MPLAB de compiler le code puis de le charger sur la carte pour l'exécuter à partir de l'ordinateur.

3.3 Manipulations

Nos séances ont consisté à réaliser des programmes (un par séance) ayant pour but de tester une fonctionnalité précise du microprocesseur. Pour cela, nous avions à disposition l'initialisation du code. Notre travail consistait à le compléter et à l'exécuter sur le simulateur avant de l'exécuter sur la carte elle même. Voici les projets réalisés dans l'ordre chronologique de leur réalisation.

3.3.1 Clignotement des LED



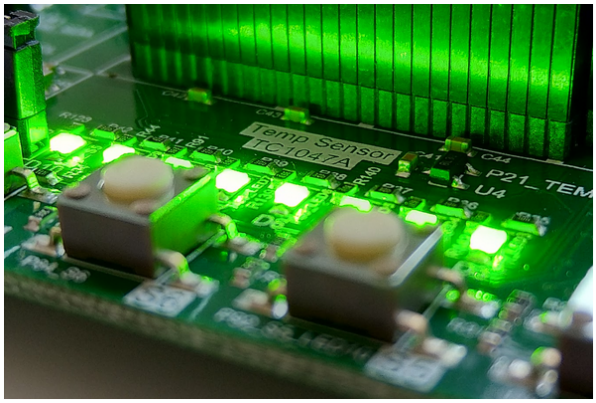
Une de nos premières manipulations a consisté à faire clignoter les LEDs de la carte. Pour ce faire, nous avons eu recours à l'utilisation des commandes LATA pour modifier les bits liés à ces 8 LEDS, qui permettent de les allumer (1) ou de les éteindre (0) (on préfère ici LATA au lieu de PORTA car on se préoccupe uniquement d'écrire sur le port A, et non pas de lire l'état actuel) :

→ TRISA = 0xC600 (on mets les LEDS de carte en sortie) ;
→ LATA = 0x0000 (on réinitialise l'état des LEDS) ;
→ LATA = 0x000F (ce qui revient à 0b00001111 : on allume les 4 LEDs des bits de poids faible, ce qui correspond aux 4 LEDs de droite) ;
→ LATA ^=0xF0 (ce qui revient à 0b11110000 : l'instruction XOR permet de renvoyer 1 si le bit était à 0 et 0 si le bit était à 1, cette instruction permet donc d'inverser l'état des 4 LEDs des bits de poids fort, soit les 4 LEDs de gauche).

La dernière instruction (inversion de l'état des LEDs de gauche) est simplement mise dans une boucle, ce qui nous a permis à l'issue de la compilation du code sur la carte de faire clignoter les LEDs de gauche, pendant que les LEDs de droite restent simplement allumées.

3.3.2 Mesure d'une température

Nous avons également effectué une mesure de température.



La manipulation consistait à poser son doigt sur le capteur thermique du microprocesseur. La température captée est alors convertie par la carte en une valeur numérique située dans la variable (Vtemp). On peut alors observer l'évolution de la température à l'écran en affichant la valeur de (Temperature). Cela nous a permis de mesurer une température de 23 C°.

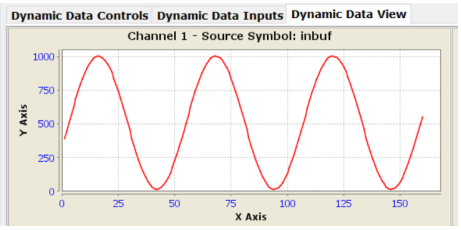
3.3.3 Acquisition d'un signal généré par le GBF

Le but de cette manipulation est d'apprendre à acquérir un signal provenant du GBF à l'aide du microprocesseur. Nous avons donc réalisé un code qui récupère la valeur du signal a un instant T en Entrée de la carte (AnalogRead) et qui la stocke dans un tableau de 160 valeurs (Inbuff).

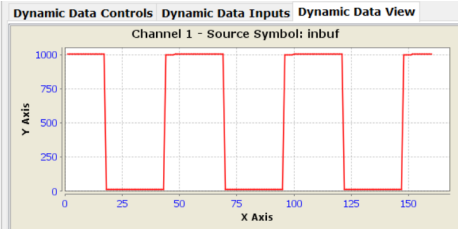
```
76  int main(void)
77  {
78
79      pasquantif = 3.3/1024;
80      i = 0;
81      LATA = 0x0000;  // Reset latcha
82      TRISA = 0xC600; // LEDs Explorer 16 en sortie
83      LATA = 0x000F; // 4 Leds du bas ON au dÃ©marrage
84      //TRISBSET = 0x30; //port B input sur 4 et 5
85      ANSELA = 0x0600;
86      ANSELBSET = 0x0231;  // AN4 et AN5 pour explorer 16 RB4 et RB5
87      adcConfigureManual();   // Configure ADC
88      AD1CON1SET = 0x8000;    // Enable ADC
89      LATA ^= 0xF0;   // inversion LEDs du haut
90      while(1)
91      {
92          valeurpot = analogRead(1);
93          inbuf[i] = valeurpot;
94          i++;
95          if (i==SIZE)
            {i=0;}
97
98      }
99  }
```

Afin de tester si notre programme fonctionnait nous avons envoyé différents types de signaux avec le GBF, ainsi on obtient les graphique suivant dans l'onglet Dynamic Data View :

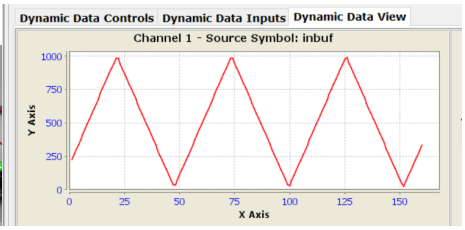
→ Signal sinusoïdale :



→ Signal carré :



→ Signal rampe :



On observe bien sur ces différentes images l'acquisition des signaux de natures différentes provenant du GBF. Nous savons donc maintenant acquérir un signal, la prochaine étape est d'apprendre à le filtrer pour ensuite utiliser la carte pour traiter du signal en temps réel.

3.3.4 Traiter un signal à l'aide d'une fonction de filtrage

Le but de cette manipulation était de tester une fonction de filtrage à l'aide du simulateur de MpLab et d'un fichier contenant un signal à filtrer.
Cette expérience s'est déroulée sur deux séances.

Dans un premier temps, M. Grisel nous a fourni un tableau récapitulatif des acquisitions de ce signal de sinus perturbé. Le but était de filtrer ce signal grâce à une fonction de filtrage qui peut être obtenue grâce à Latis Pro. Comment avons-nous obtenu cette fonction ?

Il faut dans un premier temps trouver les coefficients qui caractérisent la fonction de filtrage. Pour cela, on utilise un programme tel que dsPIC FD ou dsPIC FD Lite qui détermine ces coefficients. Une fois ces coefficients trouvés, on peut définir la fonction de filtrage « inverse » de la forme :

$$f[n] = a_o[n] + a_1[n-1] + a_2[n-2] + ... + a_8[n-8]$$

Nous avons donc implémenté la fonction dans MPLAB :

```
int main(void)
{

    pasquantif = 3.3/1024;
    LATA = 0x0000;  // Reset latcha
    TRISA = 0xC600; // LEDs Explorer 16 en sortie
    LATA = 0x000F; // 4 Leds du bas ON au dÃ©marrage
    //TRISBSET = 0x30; //port B input sur 4 et 5
    ANSELA = 0x0600;
    ANSELBSET = 0x0231;  // AN4 et AN5 pour explorer 16 RB4 et RB5
    adcConfigureManual();   // Configure ADC
    AD1CON1SET = 0x8000;    // Enable ADC
    LATA ^= 0xF0;   // inversion LEDs du haut
    counter=0;
    i = 0;
    coeffs[0]=0.07156372070312500;
    coeffs[1]=0.09909057617187500;
    coeffs[2]=0.12176513671875000;
    coeffs[3]=0.13665771484375000;
    coeffs[4]=0.14184570312500000;
    coeffs[5]=0.13665771484375000;
    coeffs[6]=0.12176513671875000;
    coeffs[7]=0.09909057617187500;
    coeffs[8]=0.07156372070312500;
```

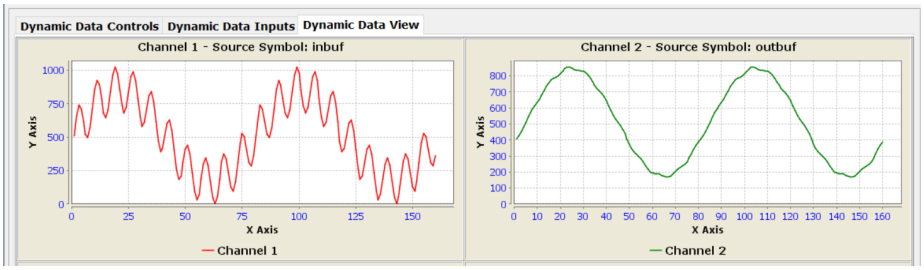
(a) Initialisation des coefficients

```
    for (i=0;i<SIZE;i++)
    {
        valeurpot = analogRead(1);
        inbuf[i] = valeurpot  ;
    }
    /*if (i==SIZE)
        i=0; */
    for (i = 0; i<SIZE;i++)
    {
        outbuf[i] = 0;
        for (j = 0; j <SIZECOEF; j++)
        {
            indice = i-j;
            if (indice < 0) indice += 160;
            outbuf[i] += (int)(inbuf[indice] * coeffs[j]);
        }
    }

}
}
```

(b) Fonction de filtrage

Pour chaque valeur de inbuf (acquisitions du sinus perturbé) on applique le filtre qui lui ajoute la somme des 8 valeurs précédentes auxquelles on a appliqué les coefficients (combinaison linéaire des 8 valeurs précédentes). On obtient ainsi un tableau outbuf avec les acquisitions filtrées. Le résultat de ce filtrage est visualisable dans l'onglet Dynamic Data View (cf. 3.2).



A gauche, nous pouvons observer le signal avant le filtrage et à droite, le résultat obtenu après le filtrage. Le signal non filtré comporte deux composantes principales : un signal de 100Hz plus un autre de 1kHz, comparable à un signal obtenu à l'aide d'un GBF. Le but du filtrage était ici d'éliminer le bruit : la composante à 1kHz. On observe bien sur le signal de droite que le signal à 1 kHz a presque disparu. Les résultats sont satisfaisants.

3.3.5 Traitement d'un signal en temps réel : utiliser la fonction de filtrage !

Nous savons dorénavant acquérir un signal provenant d'un GBF et filtrer un signal. Nous allons ainsi combiner les deux : traiter un signal provenant d'un GBF avec un filtre et observer le résultat obtenu sur un second ordinateur muni de Latis Pro en temps réel.

Exceptionnellement, nous nous sommes joint au deuxième groupe lui aussi encadré par Mr Grisel (DSP) pour la réalisation de cette manipulation. En effet, nous n'avons pas réussi à paramétrer la carte, le code ne se compilait pas. Étant en autonomie et sans les connaissances nécessaires pour corriger le code, nous avons récupéré le début de code C que Mr Grisel a fourni au groupe DSP et nous avons réalisé la manipulation avec le microprocesseur DSP. Voici le début de code C permettant d'initialiser le DSP :

```
104   void adcConfigureManual()
105   {AD1PCFGLbits.PCFG5 = 0;          // ensure AN5/RB5 is analog (Analog Pot)
106   AD1PCFGLbits.PCFG4 = 0;           // AN4/RB4 analoga
107   AD1PCFGLbits.PCFG1 = 0;
108   AD1CON1 = 0x0000;     // disable ADC before configuration
109   AD1CON1bits.AD12B = 1;
110   // Clearing SAMP bit ends sampling and starts conversion
111   // mode 12 bits  Form integer
112   // 00 = Integer (DOUT = 0000 dddd dddd dddd)
113   // Integer format
114   AD1CON2 = 0x0000; // no sc  an
115   // AD1CON2<15:13> set voltage reference to pins AVSS/AVDD
116   // Clock derived from system clock
117   // Auto sample time bits = n'est pas utilise avec timer
118   // TAD = 16 * TCY . TCY=1/40 us donc TAD = 16*25n = 400n un peu juste
119   // Pour mettre TAD>=1 us il faut TAD = 40 * TCY
120   AD1CON3 = 0x0005;
121   AD1CSSL = 0x0000; // pas de scan
122   AD1CHS0 = 0x0005; // AN5 ADC2 Potentiometre Explorer 16 AN9 Thermal
123   AD1CON1bits.ADON = 1;  // turn ADC on
124
125   // paramètres SPI
126   /* The following code sequence shows SPI register configuration for Master mode */
127   IFS2bits.SPI2IF = 0; // Clear the Interrupt flag
128   IEC2bits.SPI2IE = 0; // Disable the interrupt
129   // SPI1CON1 Register Settings
```

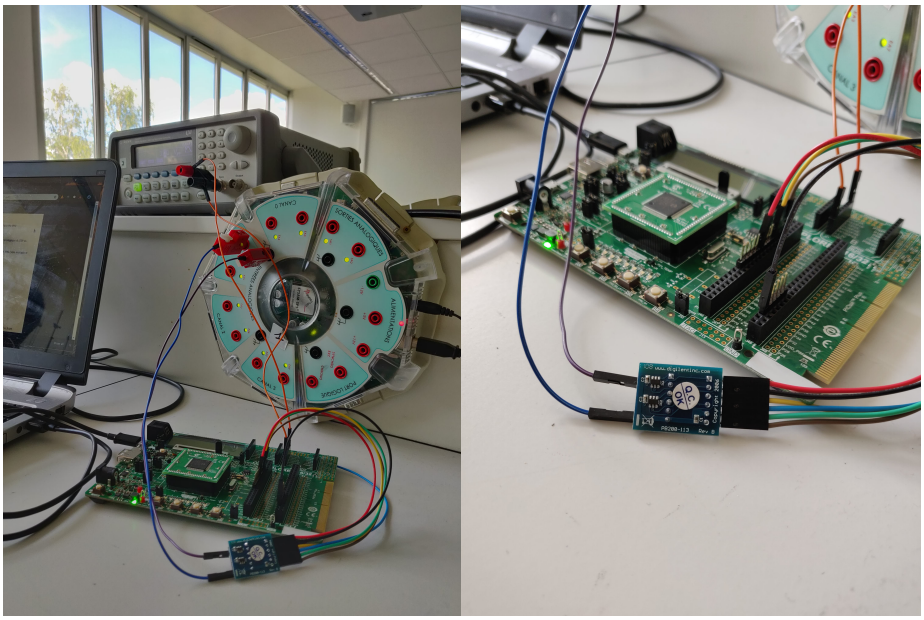
```
130   SPI2CON1bits.DISSCK = 0; // Internal serial clock is enabled
131   SPI2CON1bits.DISSDO = 0; // SDOx pin is controlled by the module
132   SPI2CON1bits.MODE16 = 1; // Communication is word-wide (16 bits)
133   SPI2CON1bits.MSTEN = 1; // Master mode enabled
134   SPI2CON1bits.SMP = 0; // Input data is sampled at the middle of data output time
135   SPI2CON1bits.CKE = 0; // Serial output data changes on transition from
136    // Idle clock state to active clock state
137   SPI2CON1bits.CKP = 0; // Idle state for clock is a low level;
138   SPI2CON1bits.SPRE = 0;
139   SPI2CON1bits.PPRE = 2;
140   SPI2CON2bits.FRMEN = 1;
141   SPI2CON2bits.SPIFSD = 0;
142   SPI2CON2bits.FRMPOL = 1;
143   SPI2CON2bits.FRMDLY = 0;
144   // Donc un front montant est actif
145   // active state is a high level
146   // Frame sync est un pulse au niveau 1
147   SPI2STATbits.SPIEN = 1; // Enable SPI module
148    }
```

Nous avons ensuite regroupé le code permettant d'acquérir un signal provenant du GBF et le code permettant de filtrer le signal. Ce qui nous donne le code suivant pour le DSP :

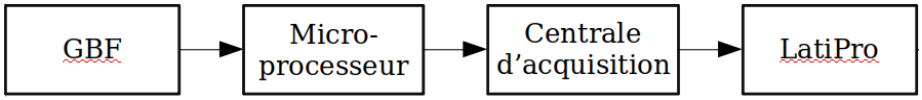
```
i = 0;
 while(1)
{

    buf[i] = analogRead(1); // TC1047
    y[i] = 0;
    for (k = 0; k<9; k++) {
        if ( (long int)(i-k) < 0) {
            y[i] += C[k] * buf[i-k+160];
        }
        else
        {
            y[i] += C[k] * buf[i-k];
        }
        SPI2BUF = y[i];
    }
    i++;
    if (i == 160) {
        i = 0;
    }
}
}
```

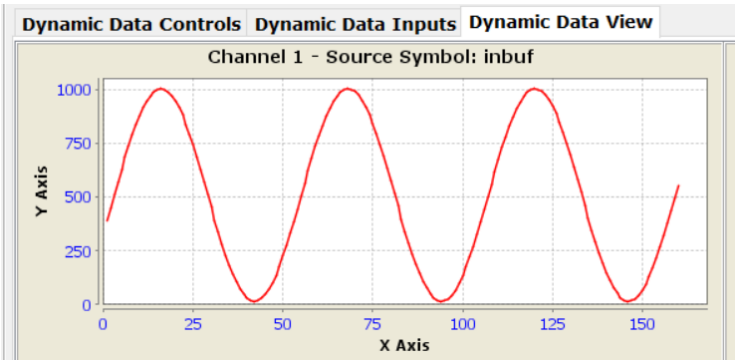
Ensuite, nous avons réalisé le montage conformément à la fiche fourni par Mr Grisel en réalisant le circuit suivant :

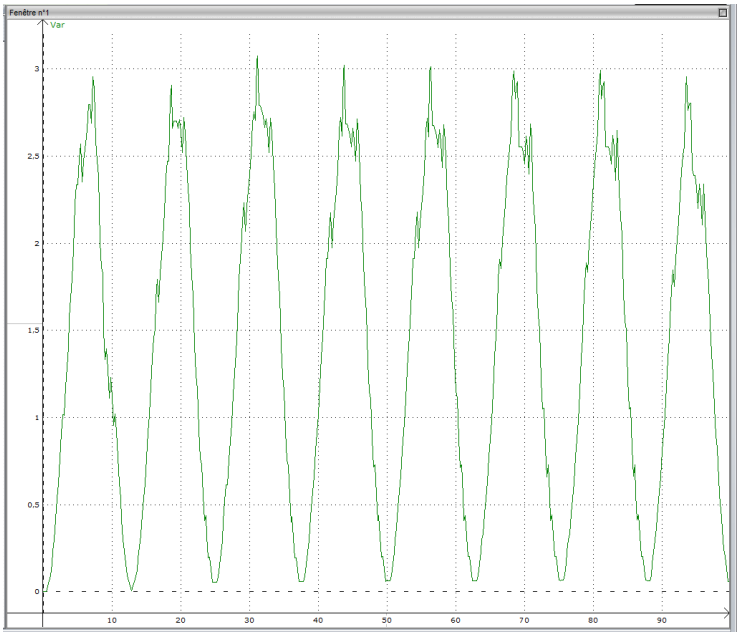


Ce montage peut être schématisé de la manière suivante :



Nous avons ainsi obtenu les résultats suivants :





Le signal est donc bien filtré : c'est un filtre passe bas. De plus on peut observer qu'il y a encore du bruit en sortie du filtre, ceci est probablement dû au fait que le filtre n'est pas parfait, mais aussi du fait que le montage est assez rudimentaire.

3.3.6 Conclusion de nos manipulations

Nous avons tout d'abord réalisé quelques manipulations qui nous ont permis de nous initier à la programmation en C d'un microprocesseur. Ainsi cela nous a permis de réaliser en semi-autonomie un code permettant de traiter un signal à l'aide du microprocesseur en temps réel.

3.4 Difficultés rencontrées

Lors de ce projet, nous avons rencontré plusieurs difficultés. Pour commencer, la documentation sur le PIC32 que l'on peut retrouver, facilement, sur internet est exclusivement en anglais, puisqu'elle s'adresse à un public international. Cela rend la compréhension plus difficile, notamment car on y retrouve du vocabulaire technique, déjà peu connu en français.

La deuxième difficulté a été le codage en langage C. En effet, pour beaucoup d'entre nous ce langage était nouveau, n'ayant jamais eu de cours sur celui-ci. Pour comprendre le code que nous avions à disposition et le compléter, nous avons donc dû chercher des cours sur internet. Or, ces derniers sont généralement composés de plusieurs chapitres et nécessitent donc du temps pour les comprendre.

Enfin, la plus grande difficulté que nous avons rencontrée est notre absence de connaissances en électronique. Nous n'étions pas vraiment en capacité d'identifier les registres que nous devions utiliser, les instructions à réaliser, ou encore les initialisations nécessaires à l'exécution. Cela nous a notamment posé problème durant nos 4 dernières séances où nous étions en autonomie.

4 Conclusion

Tout d'abord, la réalisation de ce projet Physique s'est articulée en 3 phases. La première est celle de recherche et de documentation sur l'architecture et le fonctionnement général d'un microprocesseur, et en particulier le nôtre : le PIC32. La deuxième, la plus longue, est composée de l'ensemble de nos manipulations. Ces dernières ont été d'abord des petites manipulations très simples ayant pour but de nous initier à la programmation en C et de manière plus général à mieux appréhender le fonctionnement d'un microprocesseur. Cette phase s'est achevée par le traitement en temp d'un réel d'un signal provenant d'un GBF par une fonction de filtrage. Enfin, la dernière phase est celle de la synthèse. Nous avons donc pris du recul et fait un bilan de l'ensemble de nos manipulations. Ainsi nous avons mis en commun nos différentes expériences et nos connaissances pour mener à bien la rédaction de ce rapport et de notre poster.

Ensuite, dans le continuité des projets de mathématiques et d'informatique du semestre précédent, ce projet nous a permis de mieux appréhender le travail de groupe. De plus, étant tous intéressés par des départements à dominante informatique, la réalisation de ce projet constitue une bonne expérience. En effet, cela nous a permis de découvrir le langage C que nous pratiquerons très prochainement dans notre cursus d'ingénieur.

Enfin, ce projet nous a permis de découvrir des nouvelles branches des sciences appliquées. En effet, ce projet constitue pour nous une introduction à l'électronique et à la programmation de plus bas niveau, cette dernière couramment utilisée pour résoudre des problématiques d'efficacité algorithmique et de vitesse de calcul.

5 Sources

Cours sur les microprocesseurs :

→ http://www.academiepro.com/uploads/cours/2015_08_21_chapitre_3_le_microprocesseur.pdf

Notice Microship :

→ I/O Ports : http://ww1.microchip.com/downloads/en/DeviceDoc/60001120F.pdf
→ ADC : http://ww1.microchip.com/downloads/en/DeviceDoc/61104E.pdf
→ Timers : http://ww1.microchip.com/downloads/en/DeviceDoc/61105F.pdf