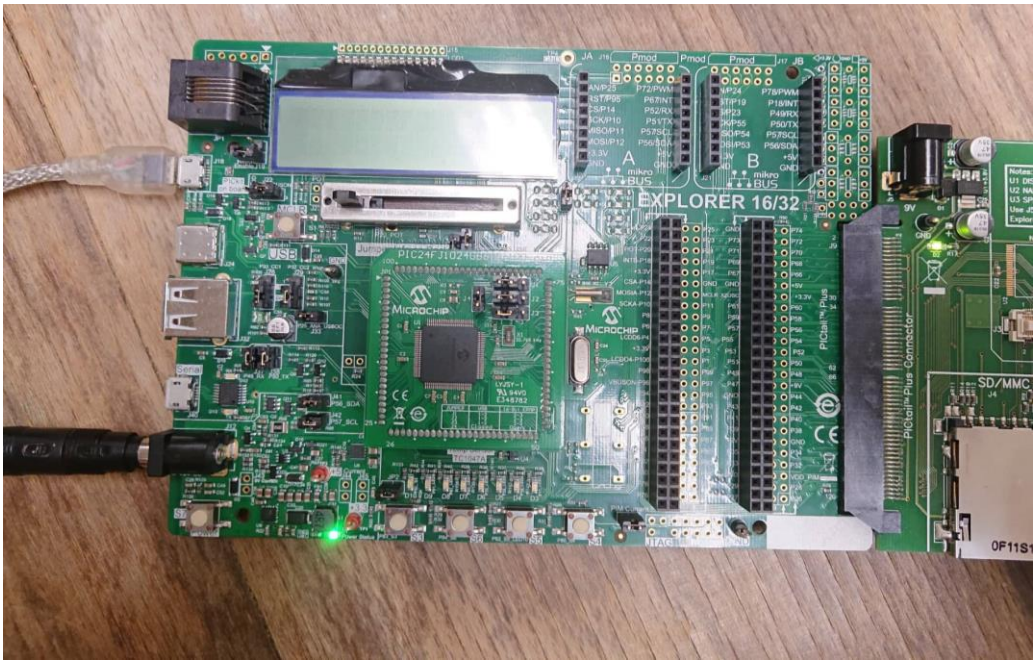
ESIREM
ÉCOLE SUPÉRIEURE D'INGÉNIEURS
NUMÉRIQUE ET MATÉRIAUX

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

Compte rendu TP

Systèmes microprogrammés



Arthur PARROD et Mathieu HERIQUE

2ère année cycle ingénieur ESIREM Dijon

Encadrant : M. Barthélémy HEYRMAN

Réalisé lors du second semestre 2020

Table des matières

Table des illustrations

Introduction

Durant notre premier semestre de seconde année du cursus ingénieur, nous avons eu des cours de programmation sur systèmes microprogrammés. Ce compte-rendu présente un projet que nous avons réalisé lors des séances de travaux pratique pour mettre en œuvre les éléments vus en cours. Nous aurons l'occasion de manipuler et de programmer des microcontrôleurs PIC24 de chez Microship et d'appliquer les notions vues en travaux dirigés avec les outils de simulation. Les objectifs du projet que nous avons réalisés sont de récupérer une température et réaliser des sauvegardes de ces dernières sur un EEPROM grâce à une communication SPI. Nous intégrerons à notre projet une gestion de la date et l'heure et un jeu du temps. Ce projet comportera deux dossiers. Le premier contiendra le projet avec un affichage réalisé sur un écran LCD tandis que le second contiendra le projet avec un affichage réalisé sur écran tactile. Dans un premier temps, nous présenterons les différents composants dont nous aurons besoin dans notre projet puis nous présenterons la gestion de l'assemblage de ces composants.

Les deux dossiers comportant le projet sont disponibles en téléchargement au lien suivant :

https://github.com/MHerique/Projet_Syst_Micropro_Herique_Parrod.git

1 Présentation de la carte et de l'environnement de développement

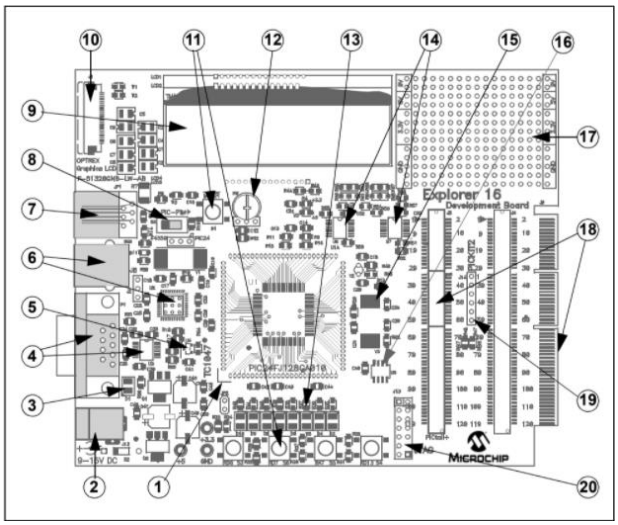
Pour l'ensemble de nos TP nous avons utilisé un microcontrôleur PIC24FJ1024GB610 posé sur une carte « explorer 16/32 Developement Board» (50001589b). Pour programmer le microcontrôleur, nous avons utilisé l'ide MPLAB et l'extension MCC de ce dernier.

Le PIC24FJ1024GB610 a une mémoire flash de 1024 Kbytes et 32Kbytes de RAM. Il peut réaliser jusqu'à jusqu'à 16 MIPS Opération et fonctionner à 32 MHz avec l'option PLL. Son oscillateur interne tourne à une fréquence de 8 Mhz. Son tableau de registres de travail est à 16*16bits. Il comporte 100 ports dont 85 pins entrée/sortie. Ses autres caractéristiques sont rassemblées dans ce tableau :

Device	Memory		Pins		Analog			Digital								RTCC	USB OTG
	Program (bytes)	Data (bytes)	Total	I/O	10/12-Bit A/D (ch)	Comparator	CTMU	16/32-Bit Timer	IC/OC/PWM	MCCP/SCCP	I²C	SPI	UART w/IrDA®	EPMP/EPSP	CLC		
PIC24FJ1024GB610	1024K	32K	100	85	24	3	Y	5/2	6/6	3/4	3	3	6/2	Y	4	Y	Y

Tableau 1 : caractéristique du PIC utilisé

La carte « explorer 16/32 » est une carte de développement avec de nombreux composants électroniques qui sont déjà intégré auquel on peut brancher notre pic. Sur notre carte nous avons notamment des dels, des boutons, un capteur de température, un lcd. On peut retrouver l'ensemble des composants disponible sur la carte grâce à sa datasheet ou au schéma ci-dessous.



1. 100-pin PIM riser, compatible with the PIM versions of selected Microchip PIC MCU and dsPIC DSC devices
2. Direct 9 VDC power input that provides +3.3V and +5V (regulated) to the entire board
3. Power indicator LED
4. RS-232 serial port and associated hardware
5. On-board analog thermal sensor
6. [No longer supported] USB connectivity for communications and device programming/debugging
7. Standard, 6-wire In-Circuit Debugger (ICD) connector for connections to an MPLAB ICD programmer/debugger module
8. Hardware selection of PIM or soldered on-board microcontroller
9. 2-line by 16-character LCD
10. Provisioning on PCB for add on graphic LCD
11. Push button switches for device Reset and user-defined inputs
12. Potentiometer for analog input
13. Eight indicator LEDs
14. 74HCT4053 multiplexers for selectable crossover configuration on serial communication lines
15. Independent crystals for precision microcontroller clocking (8 MHz) and RTCC operation (32.768 kHz)
16. Serial EEPROM
17. Prototype area for developing custom applications
18. Socket and edge connector for PICtail™ Plus card compatibility
19. Six-pin interface for the PICkit™ Programmer
20. JTAG connector pad for optional boundary scan functionality

Figure 1 : schéma de la carte Explorer 16/32 et de ces principaux composants

Datasheet: http://ww1.microchip.com/downloads/en/devicedoc/50001589b.pdf

MPLAB Ide est un logiciel extensible et hautement configurable qui intègre des outils puissants pour aider à découvrir, configurer, développer, déboguer et qualifier les conceptions intégrées pour la plupart des microcontrôleurs et contrôleurs de signaux numériques de Microchip. Grâce à son mode MCC il est possible de configurer facilement tous les ports et périphériques utilisés pour la réalisation de notre projet.

2 Afficheur LCD

Comme nous avons vu dans précédemment notre carte est équipée d'un afficheur LCD. Ce lecteur LCD permet d'afficher en même temps deux lignes de 16 caractères chacune. Dans cette partie nous allons voir comment l'implémenter.

Pour l'implémenter dans notre programme nous avons juste à ajouter le header et le .c nommé lcd ainsi qu'initialiser notre lcd dans notre programme avec la fonction InitLCD_b(). En regardant dans le header nous pouvons décider d'utiliser trois fonctions déjà définies :

- LCDHome() qui permet de reset la place du curseur d'écriture à la ligne 1 premier caractère.
- putsLCD_b() qui prend en paramètre un char de 16 caractères. Cette fonction permet d'écrire sur la ligne où le curseur est positionné.
- newLine() qui permet d'aller à la ligne suivante.

Nous avons aussi créer une fonction nommé clearLCD() qui permet de supprimer tout ce qui est affiché sur l'afficheur en réécrivant sur les deux lignes un chaine de caractère vide puis en repositionnant le curseur à sa position d'origine. Le schéma du branchement de l'afficheur est disponible en figure 2.

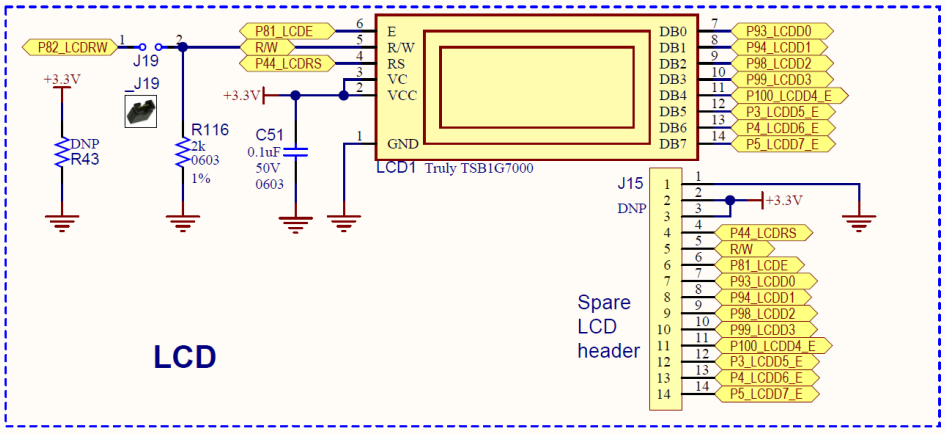


Figure 2 : schéma de l'afficheur LCD

3 Gestion du capteur de température

Dans un second temps, nous allons nous intéresser au capteur de température présent sur la carte. Pour cela, nous utiliserons une ADC afin de convertir les données envoyées par le capteur de température. Nous aurons donc besoin du capteur de température. Nous verrons dans un premier temps le fonctionnement du capteur puis de l'ADC et ensuite, nous assemblerons ces deux parties et comment récupérer la bonne valeur en degrés.

3.1 Capteur de température

En regardant dans la datasheet de notre carte on apprend que notre capteur est déjà connecté à l'ADC sur le port 4 comme nous le voyons sur la figure 3. Le capteur de température se nomme : TC1047A, son nom nous permettra de trouver la fiche technique.

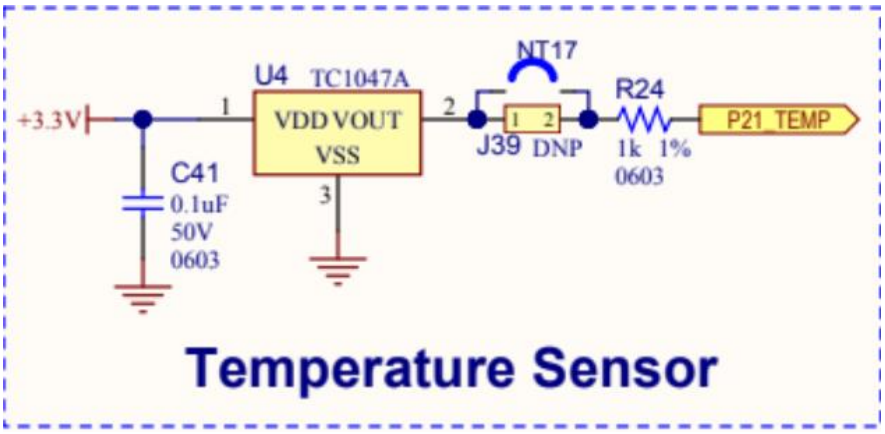


Figure 3 : schéma de branchement du capteur de température

D'après la fiche technique, ce capteur marche pour une température comprise entre -40 et 125 degrés. Dans cet intervalle le capteur va renvoyer un voltage entre 0V et 3.3V. La précision de notre capteur est de $\pm$ 2°C à 25°C.

3.2 ADC

Nous allons maintenant nous intéresser au périphérique ADC qui signifie « Analog to Digital Converter ». Son rôle est de transformer un une tension analogique en un nombre binaire. Pour configurer l'ADC il suffit d'aller sur l'option MCC de Mplab.

Comme nous pouvons le voir sur la figure 4, lors de la configuration de l'ADC, nous commençons par activer ce dernier. Ensuite nous avons décidé de cocher « Enable Auto-Sampling », cette fonctionnalité l'ADC de s'autogérer il commencera la conversion quand il aura reçu toutes les valeurs. Pour pouvoir utiliser cette fonctionnalité il faut aussi mettre le conversion trigger à « Internal counter sampling and start conversion ». Comme la température ne change pas souvent on peut prendre notre temps pour bien convertir c'est pourquoi on met le temps acquisition à une valeur importante, ici toutes les 6.25 µs.

Après avoir régler ces fonctionnalités on règle le format de sortie du convertisseur. Comme vu dans la partie sur le capteur la valeur de sortie de ce dernier est entre 0 et 3.3V on va donc régler output format à « Absolute decimal result, unsigned, right justified » car on n'a besoin que d'un nombre positif.

Il nous reste à mettre les pins d'entrées utilisées par l'ADC dans pin manager de MCC. Ici nous avons besoin que AN4. Nous aurions pu cocher le scan Enable mais comme on a un seul pin cela ne sert pas à grand-chose.

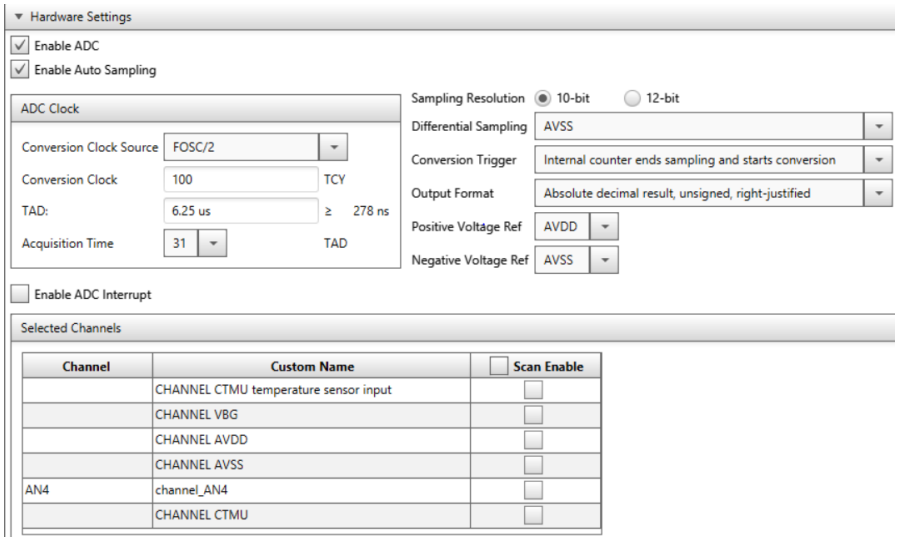


Figure 4 : configuration de l'ADC dans MCC

3.3 Récupération de la température

Maintenant que l'ADC est configuré pour récupérer et transformé en donné binaire les informations du capteur il nous reste à implémenter dans notre code l'ADC. Dans un premier temps nous utilisons la fonction « ADC1_SoftwareTriggerEnable() » qui permet de commencer la premier échantillonnage.

Dans la boucle while() on attend que la fonction ADC1_IsConversionComplete(channel_AN4) retourne 1. Cela indique que l'échantillonnage ainsi que la conversion sont finis et que nous pouvons récupérer les données.

Ces données sont des volts il faut les convertir en degrés. Pour ce faire on regarde le tableau trouver dans la datasheet du capteur qui montre les volt envoyé en fonction de la température. Il faut donc convertir en faisant cette formule :

$$solution_{degrès} = -40 + \frac{\left((out_{conversion} * 3.22) - 100\right)}{10}$$

4 Périphérique RTCC

Nous allons maintenant ajouter une fonctionnalité nous permettant d'afficher la date et l'heure sur l'affichage LCD. Pour ce faire, il nous est nécessaire d'utiliser le périphérique RTCC qui signifie « Real-Time Clock and Calendar ». Comme les autres composants, il peut être configuré depuis MCC. Ce périphérique nécessite l'utilisation du second oscillateur de notre microcontrôleur pour fonctionner. Cet oscillateur est réglé à une fréquence de 32.768 kHz. Comme nous pouvons le voir sur la figure 5, lors de la configuration du RTCC avec MCC, nous pouvons activer RTCC, configurer la date et l'heure initiale ou encore de créer des alarmes. Ces dernières peuvent par exemple permettre de déclencher un mécanisme à un moment précis de la journée ou encore tout simplement de réaliser un réveil. Il est possible d'utiliser des interruptions envoyées par RTCC.

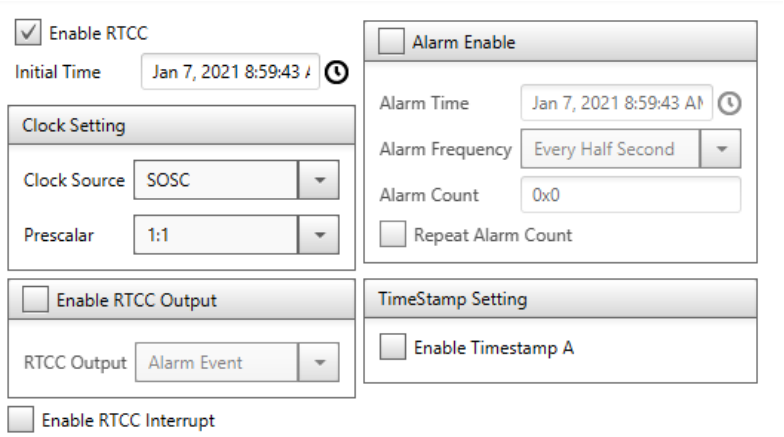


Figure 5 : configuration du périphérique RTCC dans MCC

Lorsque nous avons défini notre périphérique, il nous reste à l'implanter dans notre code. Pour gérer la date et l'heure, il nous est nécessaire d'utiliser une structure appelée « tm ». Cette structure est définie dans la bibliothèque « time ». Elle contient les variables nécessaires pour enregistrer les valeurs de l'heure, de la date ou encore le numéro de la semaine. Au début de notre programme, nous appelons la commande permettant d'initialiser le périphérique RTCC. Cette dernière l'initialise avec l'heure et la date définie dans MCC. Nous utiliserons par la suite la fonction « RTCC_TimeGet(&timer) ». Elle prend en paramètre un pointeur vers une variable « tm » comme vu précédemment et la remplie avec la date et l'heure actuelle. Il ne nous reste plus qu'à l'afficher sur l'écran LCD.

Il existe des fonctions permettant de régler l'heure ou la date lorsque le programme tourne. Cela peut permettre d'ouvrir les applications de ce module. Par exemple, il deviendrait possible de récupérer l'heure par satellite et de mettre à jour le périphérique RTCC en conséquence.

5 Stockage de valeurs

Dans cette partie, nous allons voir comment enregistrer des valeurs dans une mémoire. Pour cela, nous utiliserons une EEPROM dans laquelle nous enregistrerons les valeurs de température récupérée à la sortie de l'ADC. Nous aurons besoin d'un périphérique SPI pour communiquer avec l'EEPROM. Nous verrons dans un premier temps le fonctionnement de l'EEPROM puis du SPI et ensuite, nous assemblerons ces deux parties et nous verrons comment gérer l'enregistrement dans notre code.

5.1 EEPROM

Le composant EEPROM utilisé s'appelle « 256K SPI Bus Serial EEPROM » et a comme numéro 25LC256. Ces informations nous permettent de retrouver le composant sur le schéma de la carte exploreur 16/32 et de trouver sa fiche technique et de comprendre comment mettre en œuvre la mémoire. Elle peut enregistrer jusqu'à 32 768 octets qui sont accessible via le SPI.

La figure 6 présente le schéma de l'EEPROM sur la carte exploreur 16/32. Elle est composée de 8 entrées ou sorties. Nous en connecterons 4, dont $\overline{CS}$ qui permet de sélectionner l'EEPROM et qui fonctionne sur front bas. Comme ce composant est esclave dans la communication SPI, en plus de cette entrée $\overline{CS}$, une entrée SCK permet à la mémoire de recevoir l'horloge permettant de synchroniser la communication. Notons que la fréquence de fonctionnement maximal pour cette EEPROM est de 10 MHz. Le port SI est l'entrée sur lequel le signal d'information rentre tandis que le port SO est la sortie de la mémoire.

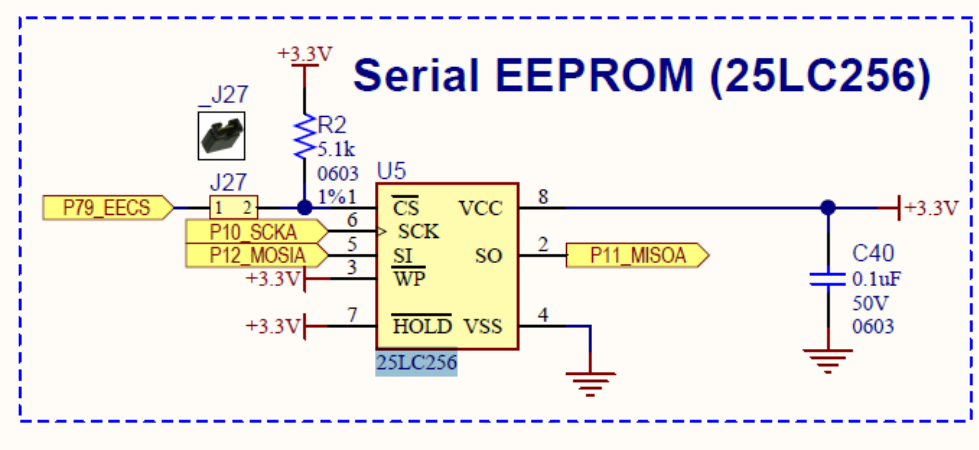


Figure 6 : schéma de connexion de l'EEPROM

L'EEPROM comporte un registre avec son statut. Ce dernier peut être lu à n'importe quel moment, même si l'EEPROM est en train d'écrire dans sa mémoire. Ce registre nous permettra de savoir si l'écriture est finie ou non pour éviter d'essayer de lire ou d'écrire une autre information si l'EEPROM est déjà en train de travailler. Auquel cas, il ne sera pas possible de communiquer avec elle.

Lorsque nous voulons communiquer avec l'EEPROM, il est nécessaire de lui fournir des instructions. Pour ce faire, il faut commencer par sélectionner le composant avec $\overline{CS}$. Lorsque cette entrée est sur front bas, L'EEPROM s'attend à avoir recevoir une séquence. Une séquence peut représenter différentes actions. Dans un premier temps, si nous voulons lire, il est nécessaire d'activer l'écriture en mémoire. Pour ce faire, nous lui envoyons 0x6. Dans le cas où nous avons fini d'écrire, nous enverrons 0X4. Lorsque nous envoyons ces instructions, nous relevons tout de suite le signal sur $\overline{CS}$ pour signaler à l'EEPROM que nous avons fini la communication. Lorsque nous voulons écrire en mémoire, nous devons lui envoyer l'instruction d'écriture 0x2 suivit de l'adresse où le premier octet sera enregistré sur 16 bits puis des données à enregistrées. Au maximum, nous pouvons envoyer 64 octets à la suite à l'EEPROM. Nous pouvons observer cette communication sur la figure 7.

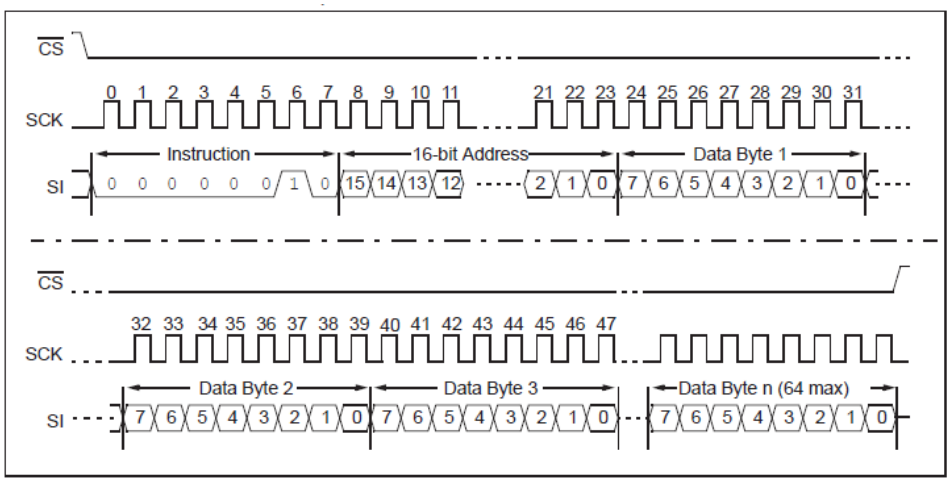


Figure 7 : chronogramme d'envoi de données vers l'EEPROM

La figure 8 présente le chronogramme de lecture de données. Comme pour l'écriture, nous lui envoyons l'instruction 0x3 suivit des 16 bits d'adressage de la première donnée lue puis d'autant d'octet vide que de nombre d'octets à lire.

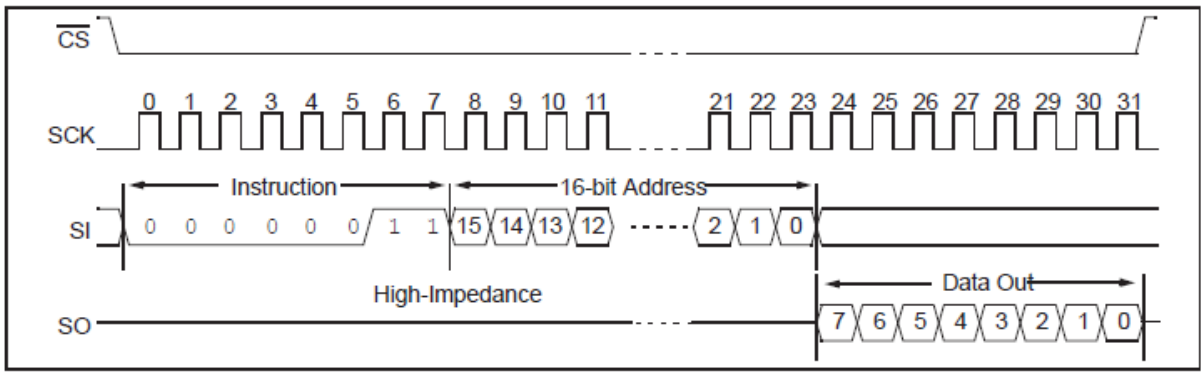


Figure 8 : chronogramme de lecture des données

5.2 Communication SPI

Notre carte comporte 3 SPI. Nous n'en aurons pas besoin d'un seul. Un Périphérique SPI permet une communication full-duplex. Ce système est du type maître-esclave. Un composant est déclaré comme maître, c'est lui qui dicte l'horloge synchronisant la communication. De plus, c'est le composant Maitre qui dirige la communication, quand communiquer et sur combien d'octets. La communication Spi permet à la fois d'envoyer des données et d'en recevoir. Toutefois, il est nécessaire d'envoyer des données pour en recevoir et nous réceptionnons des données lors de l'envoi d'information. Cela est dû au fait que les deux composants de la communication contiennent un registre sur 8 bits. La sortie d'un registre est branchée à l'entrée de l'autre registre dont sa sortie est reliée à l'entrée du premier registre. Lorsqu'un octet est envoyé, les données des deux registres sont échangées sur 8 fronts d'horloges. Lorsque la communication se fait de manière unidirectionnelle, le composant n'ayant rien à transmettre rempli son registre avec une valeur « vide » en général 0. Il est possible de brancher plusieurs esclaves sur le même maître grâce à une entrée sur les esclaves permettant de les sélectionner ($\overline{CS}$ dans le cas de notre EEPROM).

Le Spi peut être défini à l'aide de MCC comme nous pouvons le voir figure 9. Nous allons le définir comme maître, avec une communication sur 8 bits et une fréquence de communication de 8 MHz. Notons que nous utiliserons le mode SPI 0. La valeur de ce mode est définie en fonction de la polarité de l'horloge et de sa phase. Ici, l'horloge est à 0 en front bas et à 1 en front haut et sa phase permet un fonctionnement sur front descendant.

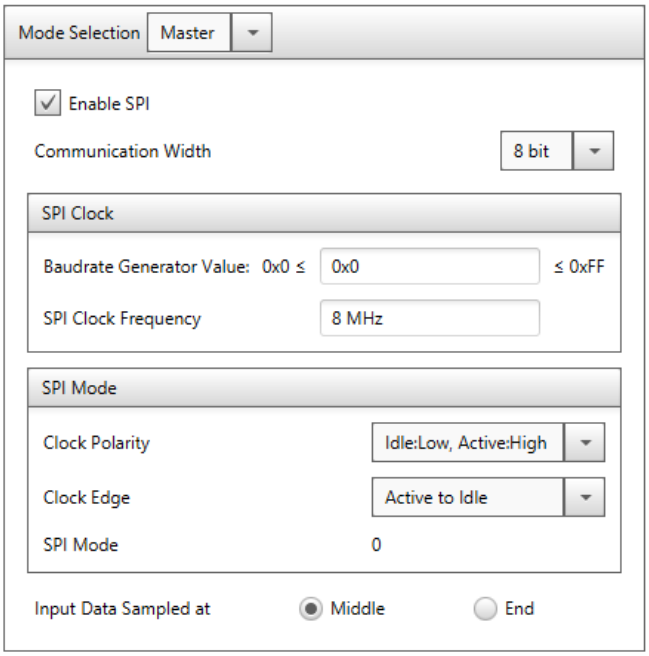


Figure 9 : configuration du périphérique SPI avec MCC

5.3 Sauvegarde et lecture des données

Maintenant que nous avons étudié notre EEPROM et notre Spi qui nous permettrons de sauvegarder nos valeurs de températures, nous allons réaliser les fonctions permettant la sauvegarde et la lecture. Ces dernières sont contenues dans les fichiers « EEPROM.c » et « EEPROM.h ». Nous avons 3 fonctions. La première permet de lire le registre et renvoie une valeur booléenne en fonction de l'état du premier bit de ce registre. Cela permet de savoir si l'EEPROM est en train d 'écrire des données en mémoire ou non. Si la valeur retournée est vraie alors elle est en train d'écrire. Ensuite, nous avons une fonction de lecture des données en mémoire. Elle prend en paramètre un tableau de deux entiers non signés sur 8 bits donnant l'adresse du premier bit Lu. Il faut aussi lui passer le nombre de bits lu ainsi qu'un tableau d'entier non signé de longueur du nombre d'octets à récupérer. Cette fonction se charge de récupérer les données et de ne retourner que les données utiles (les 3 premiers octets récupérés lors de la transmission n'étant pas des données utiles). Une fonction permettant de lire fonctionne sur le même principe, mais par manque de temps ne permet que d'écrire sur 1 octet, ce qui nous suffit pour enregistrer une mesure du thermomètre.

6 Jeu du temps

Nous allons créer un jeu du temps. Le but de ce jeu est le suivant. Un compte à rebours est affiché à l'écran. Quelques secondes avant la fin du compte à rebours, ce dernier est caché et le joueur doit l'arrêter lorsqu'il atteint 0. Il est possible de mettre différentes difficultés soit en jouant sur le temps où le compte à rebours est masqué soit en jouant sur la tolérance qui permet de gagner. Pour réaliser ce jeu, nous pourrions utiliser le module RTCC précédemment présenté. Cependant, utilise ce périphérique serait assez complexe puisqu'il nécessiterait de récupérer chaque millisecondes ou dizaine de millisecondes les informations du périphérique. Nous allons donc utiliser un « timer » et ses interruptions. Il sera configuré à l'aide de MCC comme sur la figure 10. Nous activerons les interruptions et sa période sera de 10 ms puisque nous afficherons les dizaines de millisecondes sur l'afficheur LCD.

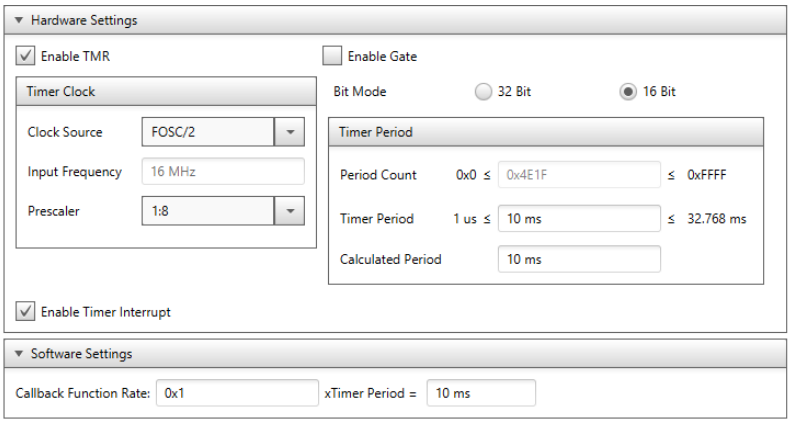


Figure 10 : configuration du périphérique timer avec MCC

Nous avons ajouté deux fichiers « jeu_du_temps.c » et « jeu_du_temps.h » qui contiennent les fonctions utiles au jeu du temps. Nous utilisons deux variables pour définir le compte à rebours. Une pour les secondes et une pour les dizaines de millisecondes. Lorsque l'interruption du « timer » arrive, les dizaines de millisecondes sont décrémentées. Ensuite, une fonction permet de gérer la décrémentation des secondes et l'affichage du jeu. Une fonction d'initialisation permet d'initialiser le jeu à 10,00 secondes.

7 Gestion des différents éléments assemblés

Les différents éléments présentés précédemment ont été implémentés dans le projet « Projet_TP » et sont utilisés dans le fichier main. Le déroulé du programme est le suivant. Après que toutes les variables utilisées sont instanciées, l'initialisation de tous les composants effectué et le convertisseur analogique numérique lancé, nous rentrons dans une boucle infinie qui fera défiler le programme. A chaque itération de la boucle, nous vérifions si la conversion de la température a été effectuée. Si c'est le cas, alors nous enregistrons la valeur convertie et nous relançons la prochaine conversion. Ensuite, une fois sur 75, nous enregistrons la valeur convertis dans l'EEPROM. Nous sauvegardons 10 valeurs, de l'adresse 0x10 à 0x1F. Ces 15 valeurs plus l'enregistrement toutes les 75 itérations permettent de réaliser une moyenne qui puisse être affichées pas la suite. Comme nous ne pouvons pas afficher toutes les informations en même temps, nous avons décidé de découper notre affichage en 4 sous-menus. Le premier signal à l'utilisateur qu'il peut utiliser le bouton S3 pour naviguer dans les sous-menus. Lorsque l'utilisateur appui sur ce bouton, un flag est incrémenté. Il prend pour valeur de 0 à 3 et est utilisé dans une machine à état permettant de gérer l'affichage en fonction du sous-menu à afficher. Le second sous-menu affiche la date et l'heure. Le troisième affiche la température lue en direct ainsi que la moyenne des dix valeurs enregistrées dans la mémoire. Le dernier état gère le jeu du temps. Avant le lancement du jeu effectué à l'aide du bouton S4, nous permettons à l'utilisateur de choisir sa difficulté. Pour la plus élevée il devra appuyer sur S6 et sur S5 pour la plus faible. Lorsque l'utilisateur lance la partie, nous démarrons le timer qui va permettre de décrémenter le compte à rebours. Ensuite, nous vérifions les conditions de fin. Soit le temps est écoulé et donc le joueur a perdu, soit le joueur a appuyé trop tôt sur le bouton soit le joueur a appuyé sur le bouton dans la zone de tolérance. Notons que la zone de tolérance se situe entre t = 0.5s et t=0s mais pas en dessous de 0 seconde.

8 Interface graphique

Pour finir nos travaux pratiques, nous avons essayé d'utiliser un écran tactile relié à notre carte exploreur 16/32. L'écran sera un « Graphics Display PowerTip 4.3in. 480x272 Boar » qui a pour numéro de composant « AC164127-6 ». Cet écran a comme dimension 480 par 272 pixels, sur la figure 11 se trouve une photo de cet écran. Pour contrôler cet écran, nous aurons besoin d'un contrôleur graphique, la partie gauche de la figure 11. Son numéro de composant est « AC164127-5 ».

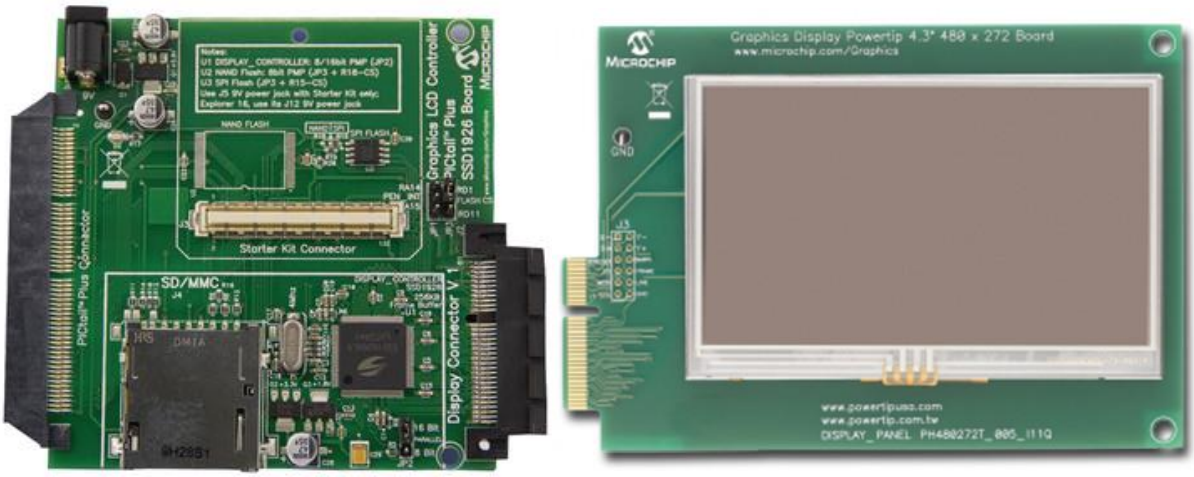


Figure 11 : photo de l'écran tactile et de son contrôleur

Pour afficher ce que nous souhaitons à l'écran, il faut communiquer avec le contrôleur graphique à l'aide de librairies. C'est le contrôleur graphique qui se chargera d'afficher ce que nous lui demandons d'afficher sur l'écran. Nous avons téléchargé un projet qui nous a été donné par l'enseignant avec les librairies nécessaires pour permettre le fonctionnement de l'afficheur. Nous n'aurons plus qu'à comprendre son fonctionnement puis à ajouter nos fonctionnalités.

Après avoir initialisé notre système, aussi bien les composantes que nous utilisons que l'afficheur, nous commençons par définir la police d'écriture définie dans les librairies et appelé « font25 ». Ensuite, lorsque nous voulons afficher quelque chose, il faut commencer par définir la couleur dans laquelle l'affichage sera fait avec une fonction « GFX_ColorSet() ». Chaque pixel affiché après cette fonction sera de la couleur passé en paramètre. Pour écrire sur l'écran, nous utilisons la fonction « GFX_TextStringDraw(int x, int y,char*) ». Cette dernière prend en paramètre les coordonnées où l'affichage débutera ainsi que la chaîne de caractère à afficher.

Cet écran est équipé d'un tactile résistif. Ce qui signifie qu'il est équipé de deux résistances qui permettent de définir où l'utilisateur appui sur l'écran. L'une donne la coordonnée verticale et l'autre la coordonnée horizontale. Comme le microcontrôleur récupère des valeurs analogiques, il utilise le convertisseur ADC pour transformer ces données et position sur l'écran. Or, notre thermomètre utilise aussi l'ADC. Nous allons donc risquer d'avoir une collision entre les initialisations de l'ADC, puisque c'est la dernière initialisation pendant l'exécution du programme qui sera prise en compte. Nous avons différents choix pour résoudre ce problème. Soit nous configurons en plus de la configuration faire par l'afficheur ce dont nous avons besoin pour notre thermomètre, soit nous partons du principe que nous n'utiliserons pas l'écran tactile et donc que nous pouvons écrire notre configuration après l'initialisation de l'écran. Dans notre cas, nous avons choisi la seconde possibilité. Cela n'a pas marché et la sortie de la conversion de notre température a comme valeur la valeur maximale qu'elle peut prendre.

Conclusion

Lors de ce petit projet réalisé pendant les travaux pratiques, nous avons pu mettre en application le fonctionnement théorique des différents composants que nous avons vu en cours. Nous avons aussi pu travailler avec les cartes physiques à la suite des travaux dirigés que nous avions réalisé avec la simulation. Nous avons mis en œuvre l'utilisation de différents périphériques comme les convertisseurs analogiques numériques ou encore le module RTCC ainsi que des interruptions systèmes avec les timers. Pour finir nous avons explorer les possibilités que nous offrait des modules complémentaires comme un écran piloté par un contrôleur graphique avec les librairies associées. Ce dernier n'a pas abouti au résultat escompté puisque nous n'avons pas réussi à résoudre les problèmes d'utilisation de l'ADC à temps. Dans une suite, il serait possible d'ajouter un système de menu contrôlable avec l'écran tactile avec un écran principal comportant la date l'heure et la température et une seconde page proposant le jeu du temps.