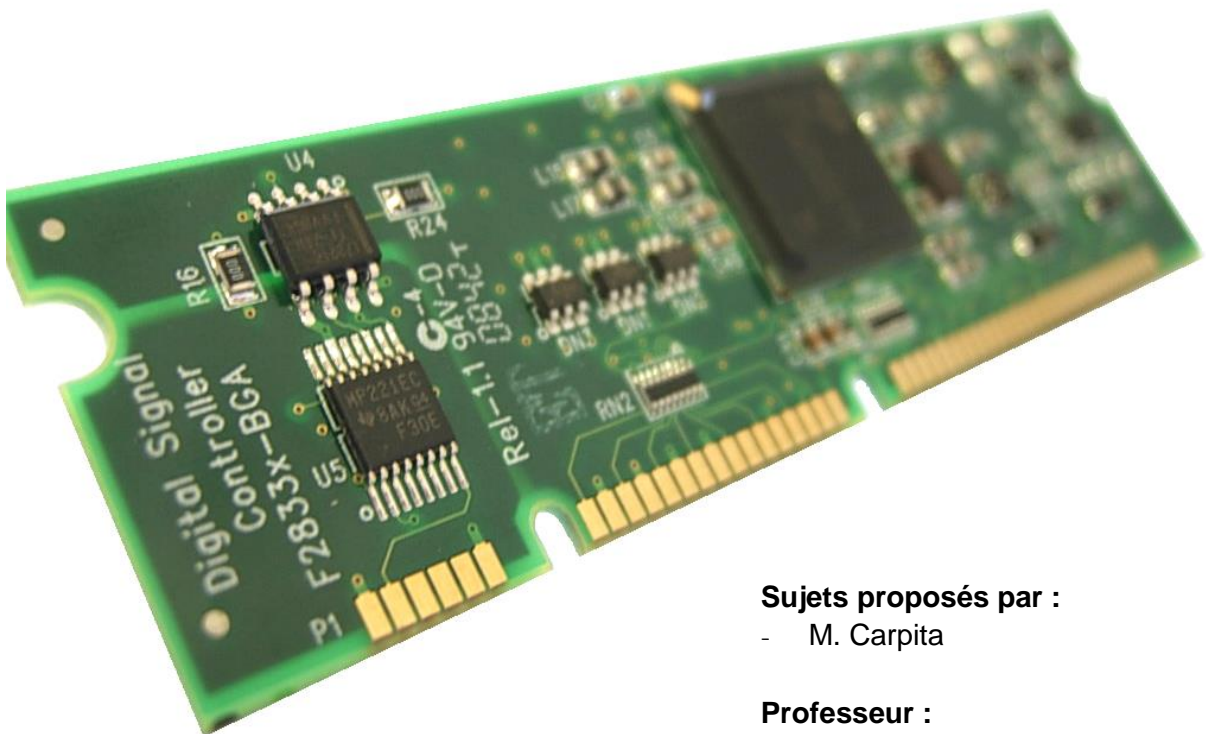


MA_PowELSys

Introduction au DSP

**Sujets proposés par :**

- M. Carpita

Professeur :

- Mauro Carpita

Motivation

Dans le cadre de cette première séance de laboratoire, l'apprentissage d'une utilisation simple d'un DSP permettra d'aborder le fonctionnement du logiciel « Code Composer » utilisé pour la programmation du processeur, ainsi que certaines fonctions d'un DSP.

Objectif

Les objectifs à atteindre lors de la séance sont :

- Apprendre le fonctionnement du logiciel « Code Composer ».
- Allumer et éteindre une LED à une fréquence de 5 Hz.

Matériel

Matériels permettant d'atteindre les objectifs :

- Logiciel Code Composer Studio
- Installation CESAR

Documentation

Une marche-à-suivre est fournie à titre d'aide-mémoire ou afin de combler d'éventuelle retard lors de la séance de laboratoire.

Travail demandé

Liste du travail à effectuer lors de la séance :

- Installation de « Code Composer Studio 7.0.0 »
- Programmation du DSP pour atteindre l'objectif :
 - Affecter les registres adéquats
 - Affecter les GPIO
 - Ecrire la routine ADC

Marche à suivre – Introduction au DSP

Instructions pour l'installation de Code Composer et pour la réalisation de l'objectif de la séance.

1. Installation du logiciel Code Composer Studio 7.0.0

Cette installation est à réaliser pour disposer de CCS sur votre ordinateur personnel.

Étape 1 :

Sous le répertoire suivant se trouve le dossier d'installation du logiciel :

<https://www.ti.com/tool/download/CCSTUDIO/7.0.0.00042>

Faire une copie du dossier .zip sur son bureau personnel et extraire le dossier.

Suite à cela lancer l'application « ccs_setup_7.0.0.00042 ».

Étape 2 :

Après avoir accepté les conditions d'utilisation, la famille de produit est demandée. Choisissez « C2000 real-time MCUs » (Figure 1).

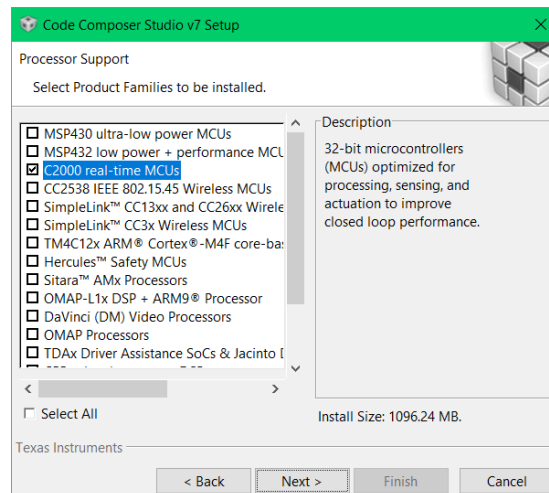


Figure 1 : Sélection du type d'installation

Lors de la sélection des émulateurs, cochez Blackhawk Debug Probes (Figure 2).

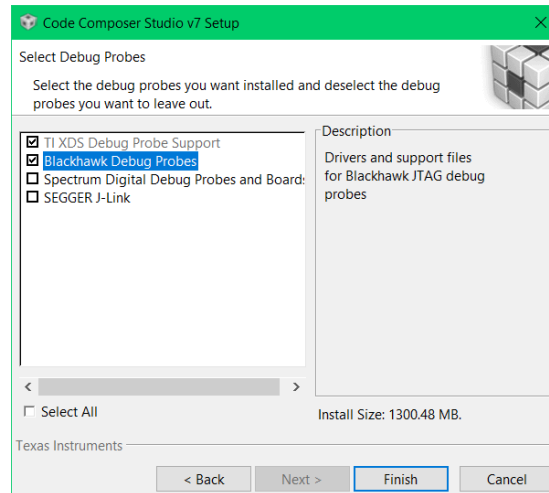


Figure 2 : Sélection des émulateurs

L'emplacement du nouveau répertoire devra également être spécifié.

Tout ceci fait, l'installation démarre. Elle peut prendre une bonne heure dans certain cas.


L'installation terminée, ouvrez Code Composer.

2. Création d'un projet

Étape 1 :

Dès l'ouverture du logiciel, il est demandé de créer un dossier workspace (sur le bureau –pas directement le bureau lui-même– ou ailleurs)

Étape 2 :

Créer un nouveau projet en cliquant d'abord sur la flèche à côté de cette icône  (en haut à gauche) puis sur « CCS Project ». Cette fenêtre apparaît (Figure 3).

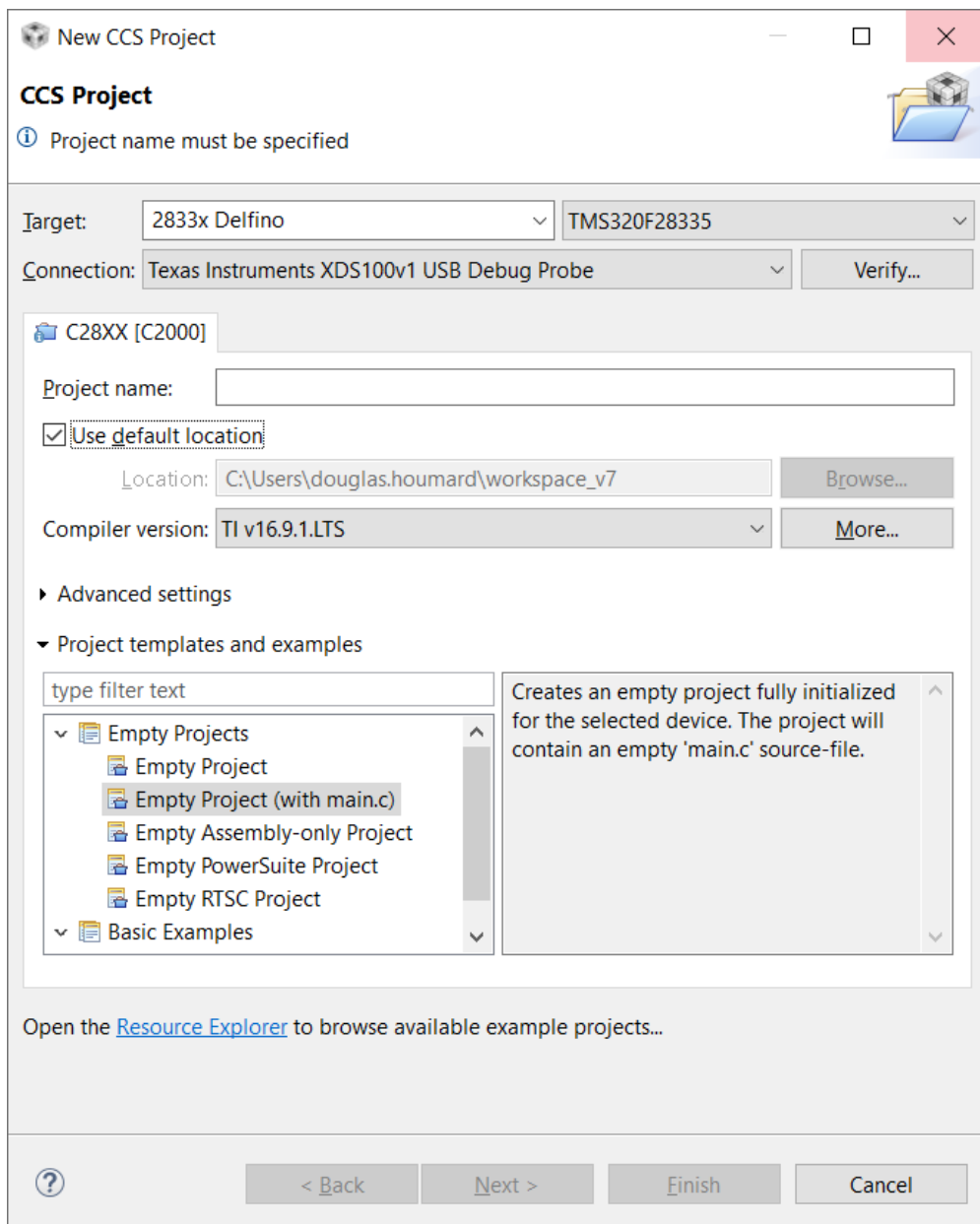


Figure 3 : Paramètres pour la création d'un nouveau projet

Dans cette fenêtre :

- Rentrez un nom de projet sous « Project name ».
- Sous « Target » choisissez « 2833x Delfino » et à côté « TMS320F28335 ».
- Sous « Connection » choisissez « Texas Instruments XDS100v1 USB Emulator » ou « Blackhawk USB2000 Controller » en fonction de l'appareillage utilisé.

A la fin cliquez sur « Finish ».

Un nouveau dossier portant le nom de votre projet est apparu dans l'onglet « Project Explorer » qui vous permet de visualiser son contenu.

De base, le dossier est relativement vide et n'est pas utilisable en l'état. C'est pourquoi il est nécessaire de rajouter un certain nombre de fichiers que vous trouverez dans les fichiers Teams liés au laboratoire n°1.

Avant de copier ces fichiers, supprimez les fichiers suivants :

- Le « main.c » qui se trouve à la racine
- Le « 28335_RAM_Ink.cmd » qui se trouve à la racine

Copiez les fichiers que vous avez téléchargés. Pour les coller dans le projet, il faut faire un clic droit sur le nom du projet dans la fenêtre « Project explorer » et clic gauche sur « Paste ».

Dans le dossier « source », il faut exclure le « main_MAT2DSP.c ». Pour faire cela, faites un clic droit dessus, puis cliquez sur « Exclude from Build... ». A présent le nom du fichier est en gris et son symbole a été barré, ce qui signifie qu'il n'est plus actif.

Tous les fichiers nécessaires sont maintenant présents.

Par défaut le projet va chercher les fichiers .h dans la racine, ce qui dans notre cas n'est pas suffisant puisque d'autres se trouvent dans le dossier « include » du projet (ajouté précédemment).

Pour pouvoir accéder au dossier, il faut ajouter le chemin d'accès en faisant un clic droit sur le nom du projet et sélectionner « Properties ». Ensuite cliquez sur « Include Options ».

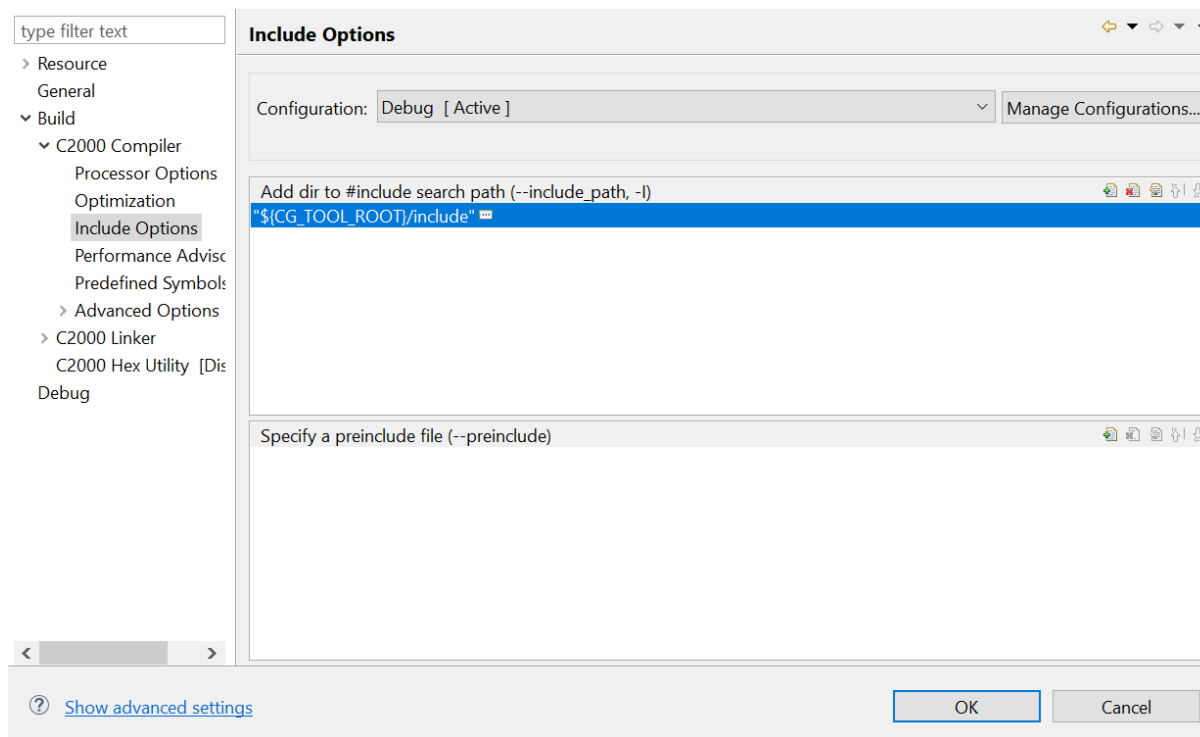


Figure 4 : Fenêtre include options avant paramétrage

La fenêtre de la Figure 4 apparaît.

Cliquez sur l'onglet avec le + en haut à droite de la fenêtre « Add dir to #include search path » et écrivez : "../include" puis OK.

Vous devez vous retrouver avec la fenêtre en Figure 5.

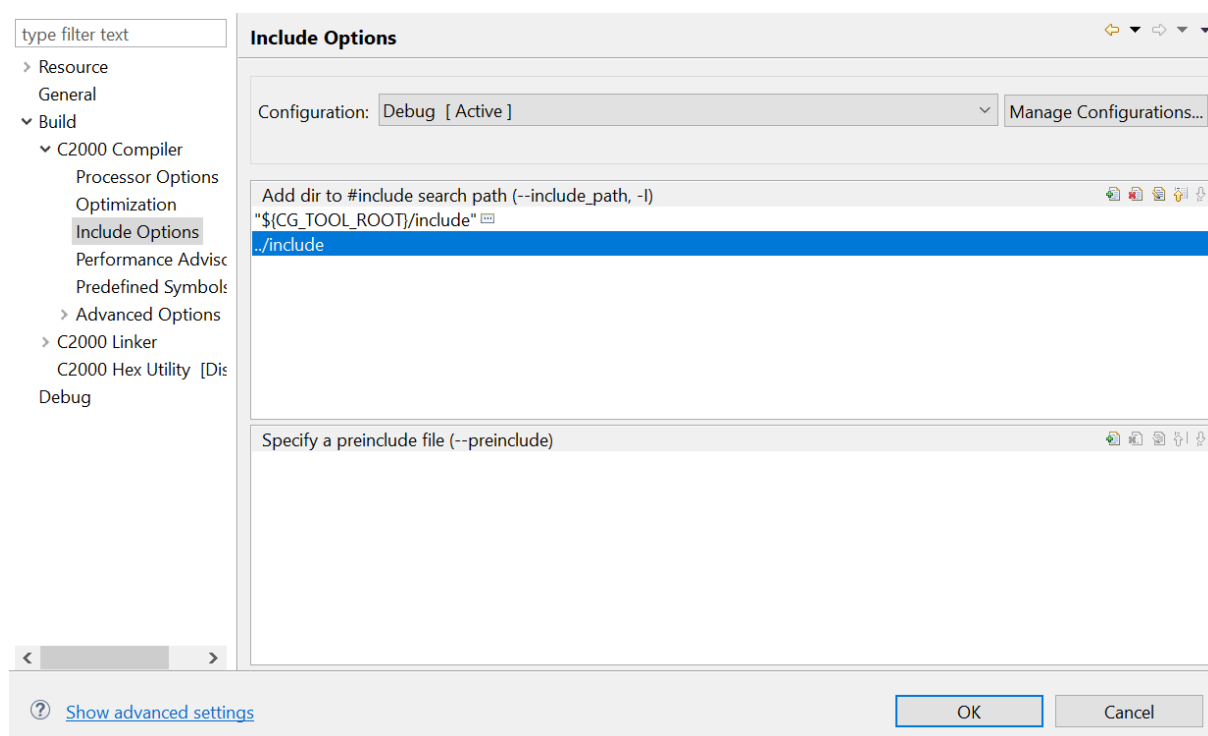



Figure 5 : Fenêtre include options après paramétrage

Et cliquez sur « OK ».

A cet instant, si vous le compilez,  votre projet est fonctionnel malgré quelques « Warnings » dû à un problème de compatibilité. Ceux-ci ne posent pas de soucis, ils peuvent donc être ignorés.

3. Programmation du DSP pour allumer et éteindre une LED à une fréquence de 5 Hz

L'idée générale sera de configurer une routine de service d'interruption (ISR) de l'ADC à l'aide du signal de commande triangulaire d'un PWM (Pulse Width Modulator). Le code à l'intérieur de la routine permettra à la LED de clignoter à une fréquence de 5 Hz.

Seules les fonctions des registres modifiés seront traitées et commentées. Il est clair que ces derniers se composent de nombreuses autres fonctions mais il serait trop fastidieux de toutes les aborder.

Afin de mieux comprendre la configuration de chacun de ces registres référez-vous aux datasheets correspondantes disponibles dans les fichiers Teams de laboratoire.

Étape 1 – Programmation de l'ADC :

Deux registres, se trouvant dans le fichier « Adc.c », sont à configurer (ce fichier est dans le dossier « source ») :

- ADCTRL1
- ADCTRL2

Ces registres seront configurés à l'aide du Tableau 1.

XCLKIN	SYSCLKOUT	HISPCLK	ADCTRL3 [4-1]	ADCTRL1 [7]	ADCCLK	ADCTRL1 [11-8]	SH Width
30 MHz	150 MHz	$HISPCP = 1$ $150 \text{ MHz} / (2 \times 1) = 75 \text{ MHz}$	$ADCLKPS = 0$ 75 MHz	$CPS = 1$ $75 \text{ MHz} / (2 \times 1) = 37.5 \text{ MHz}$	37.5 MHz	$ACQ_PS = 3$ $(1/37.5 \text{ MHz}) \times (3+1) = 106.7 \text{ ns}$ (Acquisition time)	106.7 ns

Tableau 1 : Horloge du quartz à l'ADC

• ADCTRL1

Il permet de :

- Configurer le **diviseur d'horloge** de l'ADC. (CPS)
Bit de configuration : 7
- Configurer la **durée de la fenêtre d'acquisition**. C'est-à-dire qu'il détermine le temps où l'interrupteur est fermé. (ACQ_PS)
Bits de configuration : 8 à 11

Il est donc paramétré comme cela :

```
AdcRegs.ADCTRL1.all = 0x0380;
```

```

bit 15      0      : reserved
bit 14      0      : RESET
bit 13-12   00     : SUSMODE      00 = don't stop on emulation suspend
bit 11-8    0011   : ACQ_PS       Acquisition window time 0011 = 3*ADCclk
bit 7       1      : CPS          Core clock prescaler 0 = x/1
bit 6       0      : CONT_RUN     Continuous run 0 = Start/Stop mode
bit 5       0      : SEQ_OVRD     Sequencer Override
bit 4       0      : SEQ_CASC     Cascaded sequencer operation 1=cascaded
bit 3-0     0000   : reserved

```

• ADCTRL2

Il permet de :

- Configurer l'**autorisation d'interruption du séquenceur 1**. (INT_ENA_SEQ1)
Bit de configuration : 11
- Configurer le **mode d'interruption du séquenceur 1**. (INT_MOD_SEQ1)
Bit de configuration : 10
- Configurer l'**autorisation au séquenceur 1 de débiter la conversion lorsqu'apparaît la pulsation créée par le signal PWM**. (ePWM_SOCA_SEQ1)
Bit de configuration : 8

Ici, l'objectif est que la conversion débute sur commande du signal PWM. Et au moment où la conversion est terminée, la routine d'interruption de l'ADC est déclenchée. Sachant cela, le registre se configure ainsi :

```
AdcRegs.ADCTRL2.all = 0x0900;
```

```

bit 15    0    : EPWM_SOCA_SEQ    1 = sequencer started by EPwm_SOCA
bit 14    0    : RST_SEQ1         Reset sequencer 1
bit 13    0    : SOC_SEQ1         Force Start Of Conversion of Sequencer 1
bit 12    0    : reserved
bit 11    1    : INT_ENA_SEQ1     1 = Sequencer 1 interrupt enable
bit 10    0    : INT_MOD_SEQ1     SEQ1 interrupt mode
bit 9     0    : reserved
bit 8     1    : EPWM_SOCA_SEQ1   1 = sequencer1 started by EPwm_SOCA
bit 7     0    : EXT_SOC_SEQ1     1 = Enable external SOC from GPIOA
bit 6     0    : RST_SEQ2         Reset sequencer 2
bit 5     0    : SOC_SEQ2         Force Start Of Conversion of Sequencer 2
bit 4     0    : reserved
bit 3     0    : INT_ENA_SEQ2     1 = Sequencer 2 interrupt enable
bit 2     0    : INT_MOD_SEQ2     SEQ2 interrupt mode
bit 1     0    : reserved
bit 0     0    : EPWM_SOCA_SEQ2   1 = sequencer2 started by EPwm_SOCA

```

Étape 2 – Programmation du PWM :

Trois registres, se trouvant dans le fichier « EPwm.c », sont à configurer. Ce fichier est dans le dossier « source » :

- TBCTL
- ETSEL
- ETPS

Note : Il est possible de paramétrer 6 PWM différents, mais seul le PWM1 le sera.

• TBCTL

Il permet de :

- Configurer le **mode de comportement du PWM** lors de l'émulation. (FREE, SOFT)
Bits de configuration : 14 à 15
- Configurer la **source du signal de synchronisation**. (SYNCSEL)
Bits de configuration : 4 à 5
- Configurer la **forme du signal PWM**. (CTRMODE)
Bits de configuration : 0 à 1

Le registre doit être configuré pour que le compteur tourne librement, que la synchronisation se fasse lorsque le compteur passe par 0 et que la forme du signal PWM soit en triangle. Donc :

```
EPwm1Regs.TBCTL.all = 0xE012;
```

bit 15-14	11	:	FREE,SOFT	11 = Free run
bit 13	1	:	PHSDIR	1 = Count up
bit 12-10	000	:	CLKDIV	0 = x/1
bit 9-7	000	:	HSPCLKDIV	0 = x/1
bit 6	0	:	SWFSYNC	Write 1 will force sync.
bit 5-4	01	:	SYNCOSEL	source of SyncOut signal (00:SyncIn, 01:CTR=0, 10:CTR=CMPRB, 11:Disable)
bit 3	0	:	PRDL	0 = TBPRD shadowed
bit 2	0	:	PHSEN	1 = use the TBPHS <u>reg</u> when a SyncIn occurs
bit 1-0	10	:	CTRMODE	00 = up, 01 = Down, 10=Up/Down, 11=Stop

• ETSEL

Il permet de :

- Configurer l'autorisation de créer une pulsation pour débiter la conversion par l'ADC. (SOCAEN)
Bit de configuration : 11
- Configurer l'instant où la conversion est lancée. (SOCASEL)
Bit de configuration : 8 à 10

L'ADC autorise le signal PWM à lancer le séquenceur, donc le PWM doit dire à l'ADC quand débiter la conversion. Cette indication est donnée sous forme d'une pulsation (EPWMxSOCA pulse). De plus il faut indiquer à quel moment du signal PWM cette pulsation apparaît. Dans notre cas, cette pulsation apparaîtra à chaque passage par 0. Pour ce faire, voici la configuration du registre :

```
EPwm1Regs.ETSEL.all = 0x0900;
```

bit 15	0	:	SOCBEN	1 = Enable ADC start of conversion B
bit 14-12	000	:	SOCBSEL	001:TBCTR=0, 010:TBCTR=TBPRD, 100:CMPA up, 101:CMPA down, 110:CMPB up, 111:CMPB down
bit 11	1	:	SOCAEN	1 = Enable ADC start of conversion A
bit 10-8	001	:	SOCASEL	001:TBCTR=0, 010:TBCTR=TBPRD, 100:CMPA up, 101:CMPA down, 110:CMPB up, 111:CMPB down
bit 7-4	0000	:	reserved	
bit 3	0	:	INTEN	1 = Enable EPWMx_INT
bit 2-0	000	:	INTSEL	(001: TBCTR = 0, 010: TBCRT=TBPRD 100: CMPA up, 101: CMPA down 110: CMPB up, 111: CMPB down)

- **ETPS**

Il permet de :

- Configurer à quel moment la pulsation (EPWMxSOCA) est générée.
(SOCAPRD)

Bit de configuration : 8 à 9

Pour notre application la pulsation peut être générée dès le premier passage par 0 du signal PWM. Donc le registre devient :

EPwm1Regs.ETPS.all = 0x0100;

<i>bit 15-14</i>	<i>00</i>	<i>:</i>	<i>SOCBCNT</i>	<i>read only</i>
<i>bit 13-12</i>	<i>00</i>	<i>:</i>	<i>SOCBPRD</i>	<i>Number of event that generates the pulse</i>
<i>bit 11-10</i>	<i>00</i>	<i>:</i>	<i>SOCACNT</i>	<i>read only</i>
<i>bit 9-8</i>	<i>01</i>	<i>:</i>	<i>SOCAPRD</i>	<i>Number of event that generates the pulse</i>
<i>bit 7-4</i>	<i>0000</i>	<i>:</i>	<i>reserved</i>	
<i>bit 3-2</i>	<i>00</i>	<i>:</i>	<i>INTCNT</i>	<i>read only</i>
<i>bit 1-0</i>	<i>00</i>	<i>:</i>	<i>INTPRD</i>	<i>Number of event that generates the <u>int</u></i>

Étape 3 – Programmation du fichier *Gpio* et *main* :

A présent qu'une interruption est programmée pour être enclenchée à chaque instant où le signal PWM passe par 0, il faut qu'à l'intérieur de celle-ci un code permette l'enclenchement et le déclenchement d'une LED.

Pour cela les GPIO (General-Purpose Input/Output) doivent être paramétrés. Ce sont des ports d'entrée/sortie utilisables pour plusieurs types de signaux (GPIO, PWM, SCI, CAN, ...).

Les fonctions « GpioCtrlRegs » et « GpioDataRegs » permettent de les configurer et sont constitués de plusieurs registres (Tableau 2).

		Fonction	
		GpioCtrlRegs.	GpioDataRegs.
Registre	1	GPxCTRL	GPxCLEAR
	2	GPxDIR	GPxDAT
	3	GPxMUX1	GPxSET
	4	GPxMUX2	GPxTOGGLE
	5	GPxPUD	
	6	GPxQSEL1	
	7	GPxQSEL2	

Tableau 2 : Liste des « sous-registres »

Ces registres permettent de gérer le contrôle, la direction, la fonction et la synchronisation des GPIO.

Le « x » détermine à quel groupe appartient chaque GPIO. Ils sont répartis en trois groupes définis par une lettre :

- GPIO 00 à 31 → Groupe A
- GPIO 32 à 63 → Groupe B
- GPIO 64 à 87 → Groupe C

Par défaut chaque GPIO est configuré en entrée. Donc pour que les LED reçoivent un signal il faut modifier la direction des GPIO concernés.

Les 3 LED sont connectées respectivement sur les GPIO 48, 49 et 50. Il faut donc mettre à « 1 » le bit adéquat. Les registres à modifier se trouvent dans le fichier « Gpio.c » :

```
GpioCtrlRegs.GPBDIR.bit.GPIO48 = 1;    // LED 1
GpioCtrlRegs.GPBDIR.bit.GPIO49 = 1;    // LED 2
GpioCtrlRegs.GPBDIR.bit.GPIO50 = 1;    // LED 3
```

Fonction Registre bit

Dans cette configuration les ports GPIO émettront un signal.

Remarque : Il faut également veiller à ce que les GPIO soient configurés pour transmettre leur propre signal et non pas un signal provenant d'une autre fonction. Par défaut cela est le cas.

A cet instant, pour mettre à l'état haut la sortie du GPIO, donc pour enclencher la LED, il faut paramétrer le registre suivant :

```
GpioDataRegs.GPBDAT.bit.GPIO48 = 1;
```

Et pour l'éteindre, le paramétrer ainsi :

```
GpioDataRegs.GPBDAT.bit.GPIO48 = 0;
```

Cependant il serait compliqué, mais pas impossible, d'arriver à créer un code qui enclenche la LED lors d'une interruption et déclenche la LED lors de la suivante. C'est pourquoi il est préférable de passer par une fonction logique qui permet de changer l'état d'un signal à chaque interruption. La solution est la fonction *Toggle* qui se code comme suit :

```
GpioDataRegs.GPBDAT.bit.GPIO48 ^= 1;
```

Pour une plus grande clarté, le GPIO peut être déclaré au début du fichier *main* et ainsi être renommé par sa fonction. Exemple :

```
#define LED1 GpioDataRegs.GPBDAT.bit.GPIO48
```

Et donc la fonction *Toggle* devient :

```
LED1 ^= 1;
```

A présent, vous pouvez donc ajouter à la fin du fichier « main.c », la fonction *Toggle* dans l'interruption « `interrupt void ADCINT_ISR(void)` ».

A cette étape du laboratoire, la LED ne s'allume pas car il reste à paramétrer la fréquence du signal PWM pour qu'elle clignote à 5 Hz.

Cette action est gérée par le registre :

- **TBPRD**

Il permet de :

- Configurer la période du signal PWM.

Dans le fichier *EPwm.c*, où se trouvent tous les registres concernant la fonction PWM, le registre « TBPRD » est égal à la variable « T_period » qui est configurée par le premier paramètre de la fonction « **InitEPwm1** ». Ce paramètre correspond à la fréquence du signal PWM.

La fonction est appelée au début du *main*, où tous ses paramètres sont initialement mis à 0 ce qui explique le non-fonctionnement de la LED à cet instant. Il faut donc entrer une valeur pour configurer la fréquence tout en prenant garde au fait que lors d'une interruption, seul un changement d'état de la LED est effectué. Il ne faut donc pas paramétrer la fréquence à 5 mais à 10 Hz.

Voilà, l'objectif est atteint, à présent la LED s'allume et s'éteint à une fréquence de 5 Hz (s'allume 5 fois pendant une seconde).

Remarque : Pour allumer les trois LED au même instant, le registre GPxDAT ne peut pas être utilisé car le *bit* correspondant ne se met à « 1 » qu'un coup d'horloge plus tard. En effet car au moment où le *bit* devrait passer à « 1 » le masque de la première instruction (LED1 ^= 1) est remplacé par le masque de la deuxième instruction (LED2 ^= 1), ce qui annule la première et donc n'allume pas la LED1.

En effet le code suivant ne marche pas :

```
LED1 ^= 1;  
LED2 ^= 1;
```

Il faut le remplacer par ces registres conçu exprès pour palier à ce problème :

```
GpioDataRegs.GPBTOGGLE.bit.GPIO48 = 1;    // LED1 ^= 1;  
GpioDataRegs.GPBTOGGLE.bit.GPIO49 = 1;    // LED2 ^= 1;
```