PROJET INITIATION A LA RECHERCHE

Implantation de capteurs communicants enfouis dans le béton à l'aide de l'outil de développement eZ430-RF2500 de Texas Instruments

Hamza AYZI
Ismaïl ELHASNAOUI
Année 2012

IMA 4 SA
Groupe 2
Lecteur et encadrant: Alexandre BOE

SOMMAIRE

INTRODUCTION

Le projet initiation à la recherche est un module obligatoire pour pouvoir valider notre semestre 8 à Polytech'Lille. Ce projet doit permettre à nous élèves-ingénieurs de nous initier à la recherche avec un long travail de documentation et de réflexion.

Notre choix s'est porté sur la gestion des capteurs enfouis. En effet, les réseaux de capteurs sans fil sont de plus en plus utilisés dans divers domaines. L'intérêt récent a été porté sur la faible consommation de puissance, ce qui est toujours un problème pour les applications autonomes. L'outil de développement eZ430-RF2500 équipé du microcontrôleur MSP430F2274 et de l'émetteur-récepteur sans-fil CC2500, est un outil adapté aux applications à basse consommation et à faible coût.

Lors de ce projet les caractéristiques de cet outil seront exploitées afin de réaliser un réseau de capteurs autonomes enfouis dans le béton dans le but d'étudier le vieillissement de ce dernier. Le coté autonome du projet sera mis en avant pas l'insertion d'une horloge temps réel qui aura trois principaux objectifs:

- La synchronisation des mesures à travers la génération d'une alarme configurable.
- L'optimisation de la consommation d'énergie en réveillant le dispositif que lorsqu'il faut effectuer une mesure et/ou transmettre les données stockées.
- Dater les mesures.

Afin de mettre en œuvre ce dispositif nous avons opté pour une architecture où un maître (une carte eZ430-RF2500) sert de point d'accès pour un certain nombre de capteurs et qui s'occupera de récupérer les données et les transmettre par liaison série à un PC.

Dans ce rapport, nous commencerons par vous présenter le matériel mis à notre disposition pour pouvoir réaliser notre projet.

Dans un deuxième temps, le travail réalisé tout au long de ces 10 semaines sera abordé en détail.

Nous continuerons avec les différents problèmes rencontrés et les solutions que nous avons apportés pour en résoudre une partie.

On conclura en tirant un bilan de ce projet, et on évoquera l'influence de ce dernier sur le choix probable de l'orientation que nous envisageons de donner à notre carrière professionnelle.

Présentation du matériel

I- L'outil de développement eZ430-RF2500

L'eZ430-RF2500 est un outil de développement complet avec une interface USB permettant de programmer le microcontrôleur MSP430 ainsi que l'émetteur-récepteur sans fil CC2500.
La carte électronique est équipée en plus du MSP430 qui peut fonctionner à 16MHz et du CC2500 de 18 E/S configurables, deux Leds, un bouton poussoir et une antenne.

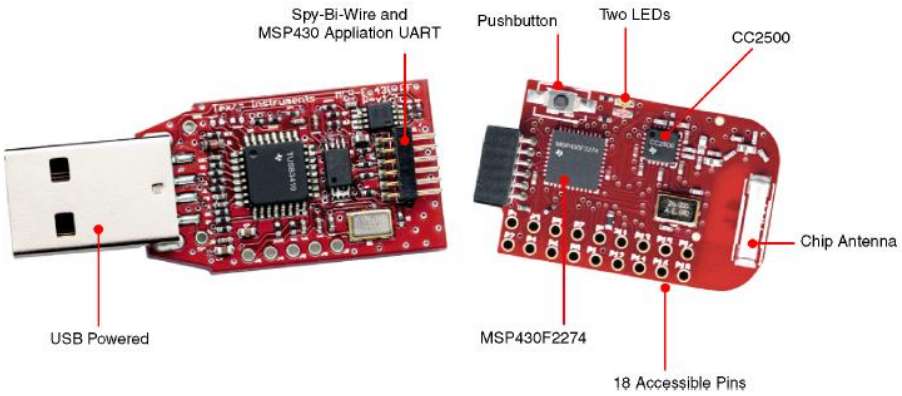

Figure 1. eZ430-RF2500

Nos premiers testes seront réalisés avec le support batterie comme support d'alimentation.
Ce dernier peut contenir deux piles AAA et il est doté de 6 E/S.

Figure 2. Le support batterie du eZ430-RF2500

La figure 3 représente les différentes entrées-sorties. A la figure 10, vous trouverez un tableau détaillant les différentes fonctions de chaque E/S.

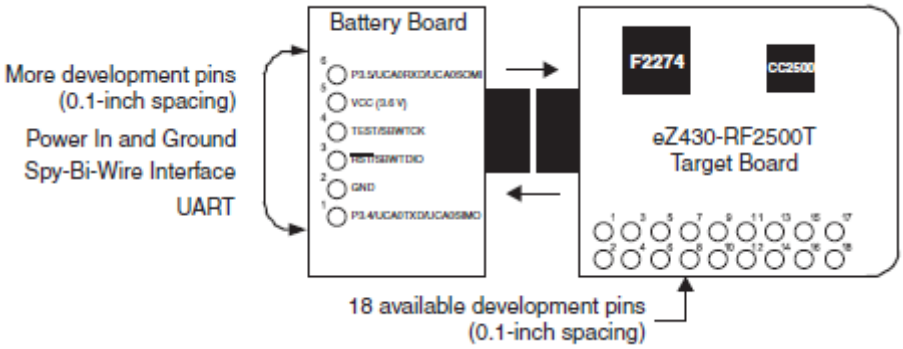

Figure 3. Les entrées/sorties du support batterie et du eZ430-RF2500

II- Le microcontrôleur MSP430F2274

La série MSP430 de Texas Instruments est une série de microcontrôleurs 16bits à très basse consommation fonctionnant à une fréquence max de 16MHz. En plus de ses 16 registres il est équipé d'un capteur de température, ce qui représente une économie pour notre projet. Trois signaux d'horloge sont générés : l'horloge maîtresse (MCK) qui est utilisée par le CPU et qui est alimentée par un oscillateur haute vitesse DC0, l'horloge auxiliaire (ACLK) et la Sub-main clock (SMCLK) qui est utilisée par certains périphériques.
Il existe cinq modes de fonctionnement basse consommation (LPM0- LPM4) obtenus en désactivant sélectivement certaines horloges. La figure 4 représente la consommation de courant pour chaque mode.
Pour notre application nous utiliserons *le mode LPM3*, qui représente un mode où les interruptions sont actives.

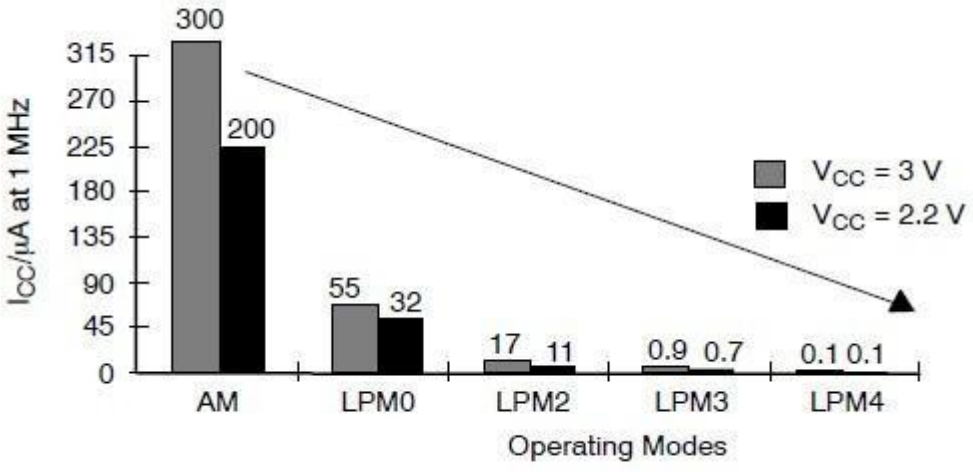

Figure 4. La consommation du courant pour chaque mode

Une fois la routine d'interruption exécutée, l'MSP430 rentre dans un des modes afin d'économiser de l'énergie en attendant une nouvelle interruption.

III- L'horloge RTC MAXIM DS1390

L'horloge RTC est une horloge temps réel qui une fois alimentée nous permets d'avoir en permanence la date et des interruptions (alarmes) programmées. En effet nous pouvons configurer la date et l'alarme en écrivant dans les registres spécifiques de l'horloge à travers une liaison SPI. La figure 5 représente les différents registres de lecture et d'écriture de l'horloge.

Comme nous pouvons le voir sur cette même figure, le composant est capable de nous fournir les centièmes de secondes, les secondes, les minutes, l'heure, le jour de la semaine, la date, le mois, le format d'heure (24 ou 12) et tout ceci sous format BCD.

WRITE ADDRESS	READ ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
80h	00h	Tenths of Seconds				Hundredths of Seconds				Hundredths of Seconds	0–99 BCD
81h	01h	0	10 Seconds			Seconds				Seconds	00–59 BCD
82h	02h	0	10 Minutes			Minutes				Minutes	00–59 BCD
83h	03h	0	12/$\overline{24}$	$\overline{AM}$/PM 10 Hour	10 Hour	Hour				Hours	1–12 +AM/PM 00–23 BCD
84h	04h	0	0	0	0	0	Day			Day	1–7 BCD
85h	05h	0	0	10 Date		Date				Date	01–31 BCD
86h	06h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century BCD
87h	07h	10 Year				Year				Year	00–99 BCD
88h	08h	Tenths of Seconds				Hundredths of Seconds				Alarm Hundredths of Seconds	0–99 BCD
89h	09h	AM1	10 Seconds			Seconds				Alarm	00–59 BCD
8Ah	0Ah	AM2	10 Minutes			Minutes				Alarm	00–59 BCD
8Bh	0Bh	AM3	12/$\overline{24}$	$\overline{AM}$/PM 10 Hour	10 Hour	Hour				Alarm Hours	1–12 + AM/PM 00–23 BCD
8Ch	0Ch	AM4	DY/$\overline{DT}$	10 Date		Day				Alarm Day	1–7 BCD
						Date				Alarm Date	01–31 BCD
8Dh	0Dh		0	BBSQI	RS2	RS1	INTCN	0	AIE	Control	DS1390/93/94
		$\overline{EOSC}$	0	X	X	X	X	0	X		DS1391
			0	BBSQI	RS2	RS1	$\overline{ESQW}$	0	AIE		DS1392
8Eh	0Eh	OSF	0	0	0	0	0	0	AF	Status	—
8Fh	0Fh	TCS3	TCS2	TCS1	TCS0	DS1	DS0	ROUT1	ROUT0	Trickle Charger	—

Figure 5. Table des adresses des registres de l'horloge RTC

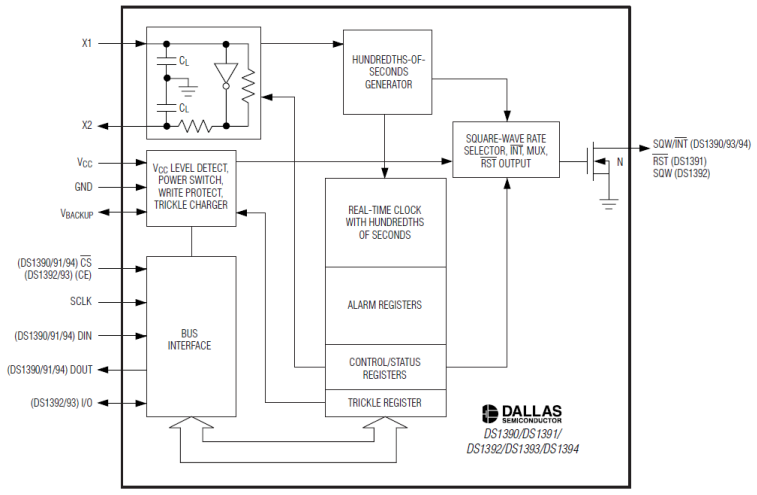

Figure 6. Diagramme fonctionnel de l'horloge RTC

Travail réalisé

I – Description des algorithmes

Pour ce projet on aura deux algorithmes bien distincts celui de l'Access Point et celui de l'End Device.

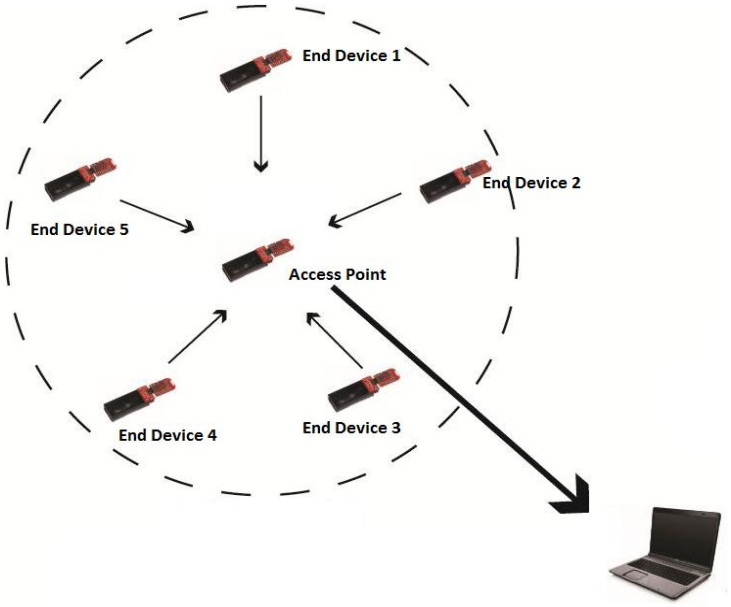

Figure 7. Architecture du Réseau

1- L'algorithme de l'Access Point (figure 8) :

Le rôle d'un Access Point et de récupérer les données transmises par les différents End Device (capteurs) et de les envoyer par liaison série a un PC qui s'occupera de la collecte des informations. Le processus d'identification des End Device par l'Access Point ainsi que le protocole réseau seront détaillés dans la partie *V- Communication sans fil*.

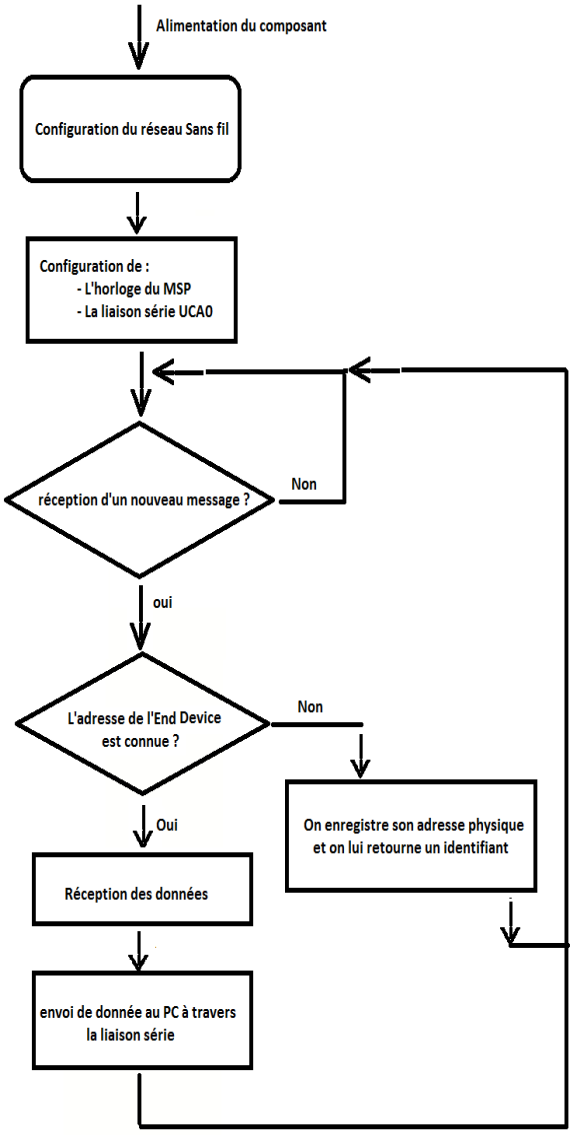

Figure 8. Algorithme de l'Access Point

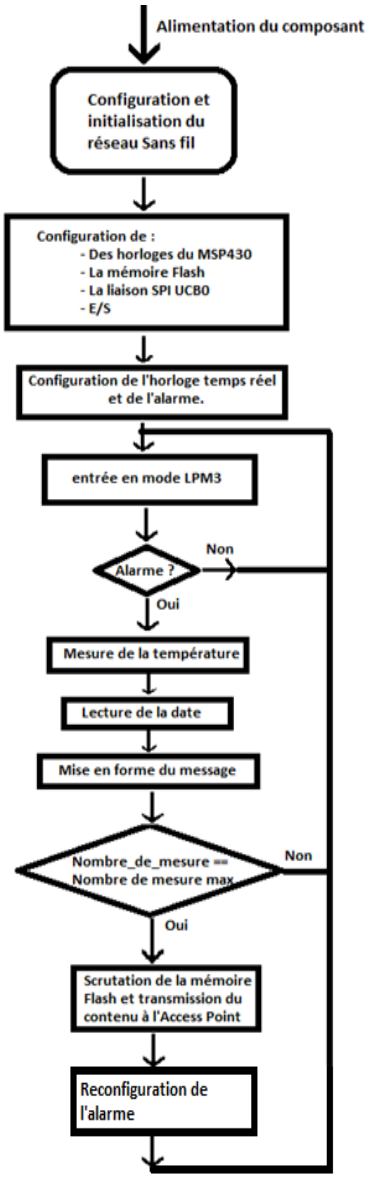

Figure 9. Algorithme de l'End Device

2- <u>L'algorithme de l'End Device (figure 9) :</u>

A la première mise sous tension l'End Device s'occupera de la configuration du réseau sans fil et de son horloge temps réel, ensuite l'MSP430 va s'endormir en attendant une interruption qui sera générée par l'alarme. En effet une fois l'interruption détectée la routine d'interruption va mesurer la température à travers le convertisseur analogique numérique, lire la date via une liaison SPI, mettre en forme la donnée, la stocker dans la mémoire flash et ensuite tester s'il a atteint le nombre de mesures maximum. Si le teste est vrai la mémoire flash sera scruter et son contenu sera envoyé à l'Access Point. Sinon le microcontrôleur se remettra en mode basse consommation et attendra un nouveau réveil.

II- Les configurations : horloge, E/S et des modes LPM

Cette partie sera consacrée à la description des registres de configuration et la présentation de la configuration que nous avons choisis nous pour notre projet.
Ces configurations se font en modifiant le contenu des registres de configuration adéquats dans le MSP 430. En effet chaque périphérique possède un certain nombre de registre qui lui sont attribués afin de réaliser différentes opérations (écriture, lecture, configuration, statu …).

1- L'Horloge :

La fréquence de l'horloge maitresse (MCLK) qu'on a choisit est de 1MHz. Elle peut être divisée et utilisée comme source d'ACLK ou SMCLK. Une division peut être nécessaire pour des périphériques qui fonctionnent à des fréquences inférieure à 1MHz, ce qui le cas avec l'horloge RTC.

Les registres de configuration sont :
BCSCTL1 : permet de fixer la fréquence de MCLK et l'oscillateur source.
BCSCTL2 : permet de choisir la fréquence d'ACLK et SMCLK.
DCOCTL : Choix de l'oscillateur et de la catégorie de fréquence.

La configuration choisit :

```
BCSCTL1 = CALBC1_1MHZ;     //choix de DCO comme source et 1MHz comme fréquence
DCOCTL = CALDCO_1MHZ;     // 0-1MHz comme catégorie et DC0 comme source
BCSCTL2 = DIVS_1;                // on divise MCLK/2 pour avoir SMCLK=500KHz
```

2- Les entrées/ sorties :

L'eZ430-RF2500 est équipé de 16 E/S configurables. Chaque pin peut être utilisé comme entrée ou sortie avec 4 fonctions différentes, la figure 10 détaille les différentes fonctions de chaque E/S. Il existe 4 ports P1-P4, P1 et P2 sont les seuls ports pouvant être câblé comme entrée d'interruptions et seul P2 et P3 sont physiquement accessible. La configuration, la lecture et l'écriture s'effectue à travers 8 registres :

PxIN : Registre de lecture. Égale à 1 si l'entrée est active à 0 sinon
PxOUT : Registre d'écriture.
PxDIR : Registre permettant de configurer la direction. 1 : Sortie et 0 : Entrée
PxSEL et PxSEL2 : Sélection de la fonction. 0 : I/O, 1 : fonction n°1, 2 : fonction n°2 …
PxIE : Registre d'interruption. 1 : activée, 0 : désactivée *(valable que pour P1 et P2)*
PxIFG : Flag d'interruption. 1 : interruption détectée, 0 : sinon.
PxIES : Choisir sur quel front il faut détecter l'interruption. 0 front montant et 1 : front descendant.

Après chaque interruption le Flag doit être réinitialisé à zéro par le programme : PxIFG = 0

Table 1. eZ430-RF2500T Target Board Pinouts

Pin	Function	Description
1	GND	Ground reference
2	VCC	Supply voltage
3	P2.0 / ACLK / A0 / OA0I0	General-purpose digital I/O pin / ACLK output / ADC10, analog input A0
4	P2.1 / TAINCLK / SMCLK / A1 / A0O	General-purpose digital I/O pin / ADC10, analog input A1 Timer_A, clock signal at INCLK, SMCLK signal output
5	P2.2 / TA0 / A2 / OA0I1	General-purpose digital I/O pin / ADC10, analog input A2 Timer_A, capture: CCI0B input/BSL receive, compare: OUT0 output
6	P2.3 / TA1 / A3 / VREF– / VeREF– / OA1I1 / OA1O	General-purpose digital I/O pin / Timer_A, capture: CCI1B input, compare: OUT1 output / ADC10, analog input A3 / negative reference voltage output/input
7	P2.4 / TA2 / A4 / VREF+ / VeREF+ / OA1I0	General-purpose digital I/O pin / Timer_A, compare: OUT2 output / ADC10, analog input A4 / positive reference voltage output/input
8	P4.3 / TB0 / A12 / OA0O	General-purpose digital I/O pin / ADC10 analog input A12 / Timer_B, capture: CCI0B input, compare: OUT0 output
9	P4.4 / TB1 / A13 / OA1O	General-purpose digital I/O pin / ADC10 analog input A13 / Timer_B, capture: CCI1B input, compare: OUT1 output
10	P4.5 / TB2 / A14 / OA0I3	General-purpose digital I/O pin / ADC10 analog input A14 / Timer_B, compare: OUT2 output
11	P4.6 / TBOUTH / A15 / OA1I3	General-purpose digital I/O pin / ADC10 analog input A15 / Timer_B, switch all TB0 to TB3 outputs to high impedance
12	GND	Ground reference
13	P2.6 / XIN (GDO0)	General-purpose digital I/O pin / Input terminal of crystal oscillator
14	P2.7 / XOUT (GDO2)	General-purpose digital I/O pin / Output terminal of crystal oscillator
15	P3.2 / UCB0SOMI / UCB0SCL	General-purpose digital I/O pin USCI_B0 slave out/master in when in SPI mode, SCL I2C clock in I2C mode
16	P3.3 / UCB0CLK / UCA0STE	General-purpose digital I/O pin USCI_B0 clock input/output / USCI_A0 slave transmit enable
17	P3.0 / UCB0STE / UCA0CLK / A5	General-purpose digital I/O pin / USCI_B0 slave transmit enable / USCI_A0 clock input/output / ADC10, analog input A5
18	P3.1 / UCB0SIMO / UCB0SDA	General-purpose digital I/O pin / USCI_B0 slave in/master out in SPI mode, SDA I2C data in I2C mode

Table 2. Battery Board Pinouts

Pin	Function	Description
1	P3.4 / UCA0TXD / UCA0SIMO	General-purpose digital I/O pin / USCI_A0 transmit data output in UART mode (UART communication from 2274 to PC), slave in/master out in SPI mode
2	GND	Ground reference
3	$\overline{\text{RST}}$ / SBWTDIO	Reset or nonmaskable interrupt input Spy-Bi-Wire test data input/output during programming and test
4	TEST / SBWTCK	Selects test mode for JTAG pins on Port1. The device protection fuse is connected to TEST. Spy-Bi-Wire test clock input during programming and test
5	VCC (3.6V)	Supply voltage
6	P3.5 / UCA0RXD / UCA0SOMI	General-purpose digital I/O pin / USCI_A0 receive data input in UART mode (UART communication from 2274 to PC), slave out/master in when in SPI mode

Figure 10. Tableau des fonctions de chaque E/S

La configuration choisie :

```
P1DIR  = 0xFF;          // P1 représente la sortie des Leds, on le configure en sortie
P1OUT = 0x00;           // on éteint les Leds
P3SEL  = 0x3E;          // P3.1 et 2.3  configurés en mode USCI_B0  et P3.0 en I/O
P3DIR  = 0x01;          // P3.0 comme sortie (le CS de la RTC)
P2DIR  = 0x00;          // configuré en entrée
P2SEL  = 0x00;          // en mode I/0
P2IE    = 0x01;         // Interruption active
P2IES   = 0x01;         // détection sur front descendant
```

3- Les modes LPM

Sachant que le MSP430 passe la majorité de son temps à attendre une interruption, il est important de réduire sa consommation d'énergie durant les périodes d'inactivité en éteignant les horloges qui ne sont pas utilisées. Il existe cinq modes de faibles puissances LPM0-4.
Nous utiliserons lors de ce projet le mode LPM3, un mode qui permet de garder une seule horloge et de consommer le moins d'énergie possible. Pour cela on ajoute la ligne ***__bis_SR_register(LPM3 bits);*** à la fin de notre programme principal.

SCG1	SCG0	OSCOFF	CPUOFF	MODE	CPU and Clocks Status
0	0	0	0	Active	CPU is active. All enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled. SMCLK, ACLK are active.
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks are disabled.

Figure 11. Les différents modes LPM

III- Acquisition de la température

Le microcontrôleur MSP430 est équipé d'un capteur de température intégré. La tension ressentie est convertie en tension en utilisant l'équation :

$$Vtemp = 0{,}00355*(Tempc) + 0{,}986$$

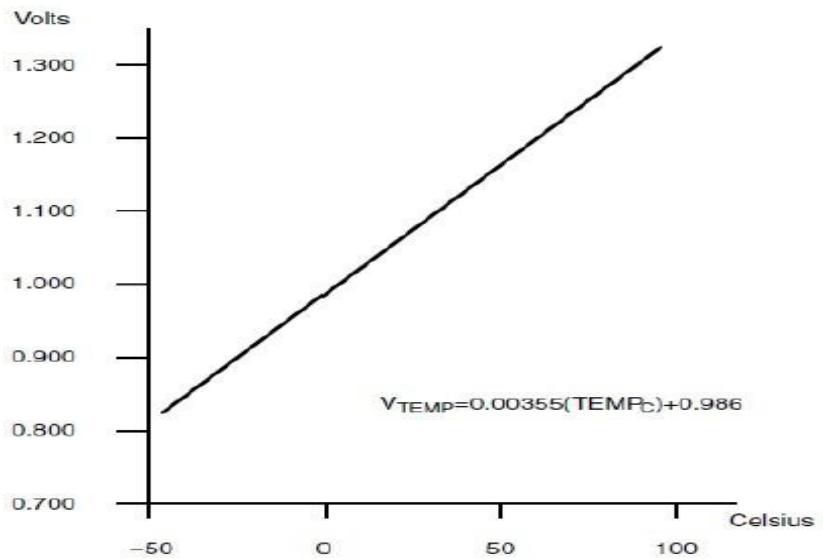

Figure 12. Courbe de conversion T°/ tension

La tension de sortie du capteur est numérisée à travers le CAN du MSP430 - *ADC10* – et stockée dans le registre ADC10MEM du CAN. En lisant ce dernier, on peut récupérer la tension.
La conversion de la tension lue dans le registre, en température, s'effectue à l'aide de l'équation :

*Temp_en_degC = ((tension - 673) * 4230) / 1024;*

IV- Fonctionnement de la mémoire Flash et stockage des données

La transmission sans fil est très gourmande en énergie. Afin de minimiser au maximum la consommation, le stockage temporaire des mesures nous parait être un choix judicieux. De plus le microcontrôleur nous offre la possibilité de pouvoir stocker un nombre important de mesure.
La transmission s'effectuera après un certain nombre de mesures.

1- <u>La segmentation de la mémoire :</u>

La mémoire est divisée en segments. On peut écrire des bits, octets ou des mots. La plus petite taille de la mémoire qui peut être effacée est le segment.
Il existe deux sections de la mémoire : la mémoire principale et d'information. Il n y a pas de différence dans le fonctionnement de ces deux sections. La seule différence réside dans la taille des segments et de leur adresse physique. En effet la mémoire d'information comporte quatre segments de 64 octets et la principale 64 segments de 512 octets. Les segments sont divisés en blocs.

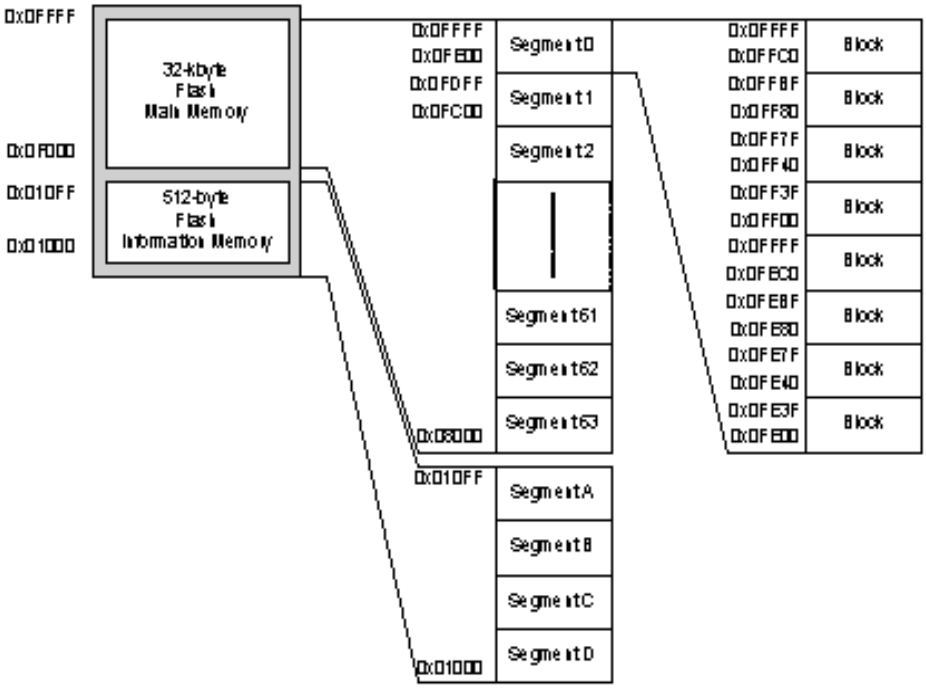

Figure 13. Les segments de la mémoire flash

2- Les opérations sur la mémoire flash :

La mémoire flash est par défaut en mode lecture fonctionnant comme une ROM. Afin d'écrire ou d'effacer des données, le mode doit être sélectionné en modifiant les registre FCLT1-3. La modification se fait en choisissant un de ces bits :

BLKWRT : écrire un block.
WRT : écrire un mot.
MERAS : effacer la mémoire principale.
ERASE : effacer un segment.

L'écriture et la lecture se fait à travers un pointeur qui pointe sur l'adresse où vous désirez écrire ou lire.

Exemple :

```
char *Flash_ptr;
Flash_ptr = (char *)0x1000;                 // @ du pointeur = 0x1000
FCTL3 = FWKEY;
FCTL1 = FWKEY + WRT;                        // écrire un mot

for(i=0;i<3;i++) *Flash_ptr++=value[i];     // l'écriture
for(i=0;i<3;i++) val[i]=*Flash_ptr++;       // la lecture

FCTL1 = FWKEY;
FCTL3 = FWKEY + LOCK;                       // on referme le block
```

On a crée deux fonctions :

void read_flash(int num_mes);
*void write_flash(uint8_t * value, int num_mes);*

La fonction *read_flash* permet de lire ce qui est stocké dans une adresse donnée et l'écrit dans un tableau déclaré en global.

La fonction *write_flash* permet quant à elle d'écrire une donnée l'adresse précisée en argument aussi.

V- La communication sans fil

L'eZ430-RF2500 utilise un protocole réseau appelé ***SimplicIT*** qui est un protocole à faible puissance de Texas Instrument pour des réseaux simples et petits. Il est utilisé dans un grand nombre d'applications de faible puissance tel que les détecteurs de gaz, alarmes et dans d'autres applications de la domotique. Texas Intstruments a inclus dans son CD d'installation une application simple « the eZ430-RF2500 sensor monitor » permettant de voir les nœuds de capteurs et les températures retournées par un réseau, cette application peut en effet être une introduction pour développer une application sans-fil.

1- <u>Format des paquets échangés :</u>

Les données échangées à travers un réseau SimplicIT sont sous forme de paquets, contenant en plus du message à transmettre un nombre d'information concernant l'émetteur, le destinataire et le contenu du message.

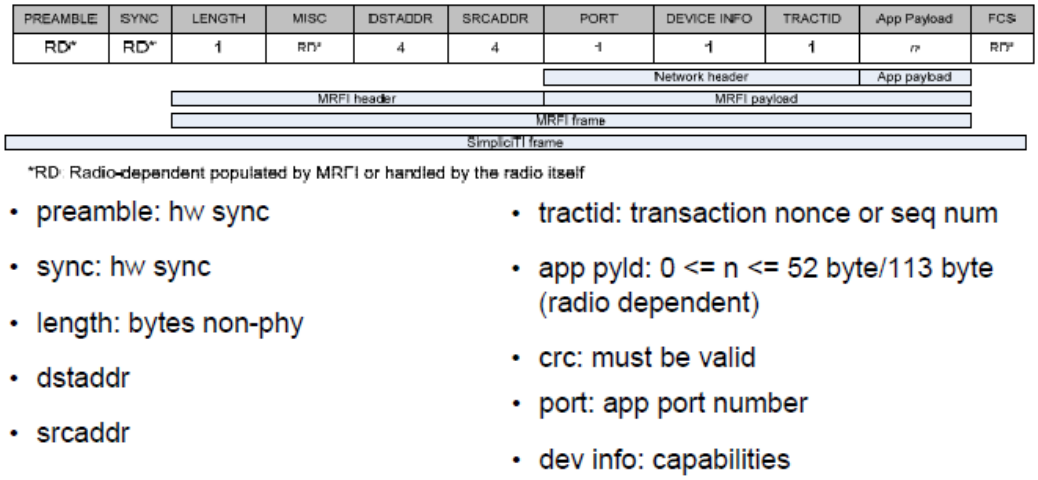

Figure 14. Format des paquets échangés

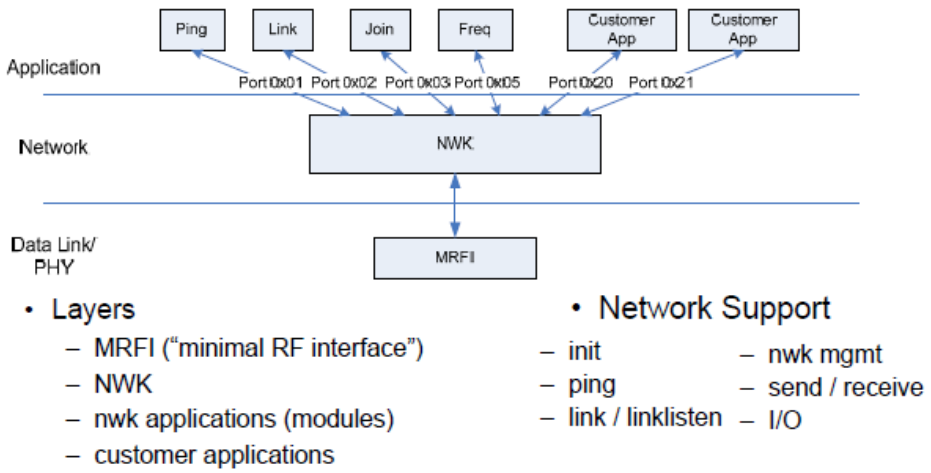

Figure 15. Architecture du protocole

2- Configuration réseau d'un Access Point :

Le rôle d'un Access Point est de récupérer les données des autres capteurs et de les retourner au PC. Mais avant cela l'Access Point doit initialiser le réseau et attribuer à chaque End Device appartenant à son réseau - en fonction de son adresse physique- un identifiant qui sera utilisé pour établir une communication. Tout cela est transparent pour l'utilisateur, ce qui rend la configuration très simple :

Étape 1: initialiser les paramètres radio

```
BSP_Init();
SMPL_Init(0);
```

Étapes 2: identifier les différents ED et établir un lien

```
SMPL_LinkListen(&linkID1);
```

Etapes 3 : attendre la réception d'un message

```
while ((SMPL_SUCCESS == SMPL_Receive(linkID1, msg, &len) {
// do something
}
```

3- Configuration réseau d'un End Device :

Une fois le nombre de mesures max atteint, l'End Device à pour mission de vider sa mémoire flash en envoyant les données à l'Access Point. La configuration réseau se fait une seule fois à sa première mise sous tension. L'initialisation des paramètres radio se fait exactement de la même manière que pour l'AP.

Etape 1 : Initialiser les paramètres radio

```
BSP_Init();
SMPL_Init(0);
```

Etape 2 : demander un identifiant à l'AP et établir un lien

```
SMPL_Link(&linkID);
```

Etape 3 : Envoi de la donnée

```
SMPL_Send(linkID1, donnee, sizeof(donnee))
```

VI- La communication en SPI

La liaison SPI pour (Serial Peripheral Interface) est une liaison série synchrone. Les interfaces communiquent selon un schéma maitre esclave et où le maître s'occupe totalement de la gestion de la communication. En effet sur le même bus, plusieurs esclaves peuvent partager le même maître. La sélection du destinataire se fait par le maître à travers une ligne dédiée appelée Chip Select (CS).

Cette liaison nécessite donc 4 signaux logiques :

SCLK — Horloge (généré par le maître)
MOSI — Master Output, Slave Input (généré par le maître)
MISO — Master Input, Slave Output (généré par l'esclave)
CS — Chip Select, Actif à l'état bas, (généré par le maître)

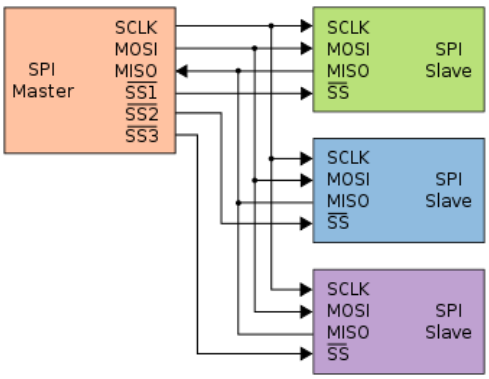

Figure 16. Liaison SPI avec un maître et trois esclaves

Lors de ce projet la liaison SPI sera utilisée pour communiquer avec l'horloge temps réel et aussi d'autres capteurs dans une future amélioration du projet.

Configuration du SPI

La configuration de la liaison SPI s'effectue elle aussi en modifiant le contenu de plusieurs registres spécifiques. Lors de cette configuration on choisira le mode (maitre ou esclave), la fréquence d'horloge ainsi que l'ordre de transmission.

Les registres modifiés sont :

```
UCB0CTL0 |= UCMSB + UCMST + UCSYNC;     // 3-pin, 8-bit, SPI maître, MSB 1st
UCB0CTL1 |= UCSSEL_2;                   // Choix de SMCLK comme horloge
UCB0BR0 = 1;                            // Division de 'horloge SMCLK
UCB0BR1 = 0;
UCB0CTL1 &= ~UCSWRST;
```

VII- Circuit réalisé avec l'horloge RTC

Avec l'aide de Mr BOE et de Mr FLAMEN, nous avons réalisé un typon (figure 17) suivi d'une carte pour pouvoir utiliser notre composant DS1390 comme bon nous semble.

Le composant Q1 est un oscillateur à quartz qui va permettre de générer les 100ème de secondes (voir figure 6). Le condensateur permet de lisser la tension et les deux résistances permettent de créer un diviseur de tension et ainsi obtenir les 1.8V souhaité sachant que le MSP43 délivre le double soit 3.6V.

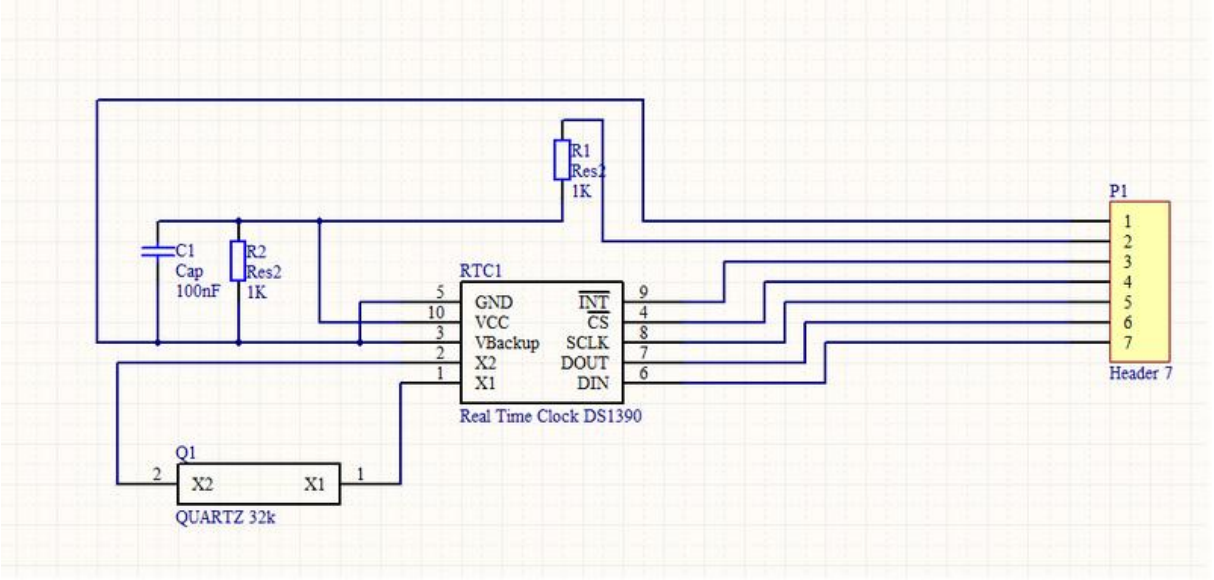

Figure 17. Schéma de la carte RTC

VIII- Configuration de l'horloge RTC

Les lectures et les écritures se feront à l'aide des deux fonctions *read_SPI (int addr_R)* et *write_SPI (int addr_W, int data) voir (partie X).* Après chaque interruption, la date de la nouvelle alarme est mise à jour.

Il existe un registre de contrôle permettant d'activer l'alarme et d'arrêter l'oscillateur interne de la RTC. Pour cela on écrit :

write_SPI (0x8D, 0x85); // on écrit 0x85 dans 0x8D qui est l'adresse de ce registre

Après chaque interruption la remise à zéro du flag est obligatoire :

write_SPI (0x8E, 0x00); // reset du flag qui se trouve à l'adresse 0x8E

Pour nos tests on a choisi le scénario suivant :

- Une mesure tous les 5 secs
- Après 12 mesures c'est-à-dire 1min envoi des données.

La configuration de l'alarme se fait alors en fonction de ce scénario.

IX- Description de l'interruption à travers le MSP430

Quand un événement se produit dans un élément électronique en dehors du microcontrôleur, cet élément informe le MSP430 en changeant l'état électrique de l'entrée d'interruption qui relit l'élément au microcontrôleur. Pour notre projet cette entrée sera l'une des E/S du port 2 *voir*. Ce changement d'état entraine le réveil du MSP430 –sortie du mode LPM - et commence à exécuter une fonction spécifique appelé ISR (Interrupte Service Routine) associée à cette interruption. Une fois l'exécution terminée le microcontrôleur rentre dans le mode basse consommation en attendant une nouvelle interruption.

La sortie alarme de l'horloge RTC – l'entrée interruption du MSP – est à l'état haut -+VCC- au repos. La configuration de la détection d'interruption a été réalisée de sorte à détecter les fronts descendants.

Squelette d'un programme d'interruption :

```
void main (void)
{
  // les différentes configurations

__bis_SR_register(LPM3_bits + GIE);          // on rentre dans le mode LPM3
While(1) ;
}

#pragma vector = PORT2_VECTOR            // la routine d'interruption du port 2
__interrupt void Port2_isr(void)
{
   // votre programme d'interruption
}
```

Problèmes rencontrés

Problèmes résolus :

Problème d'installation du logiciel IAR et Code Composer sur les PC :
Cause : Problème de compatibilité avec Windows 7.
Solution : Utilisation d'un PC portable et driver trouvé pour installer Code Composer sur les PC de bureau.

Le composant RTC fourni au départ ne fournit pas de signal d'alarme.
Solution : Commande d'un nouveau composant sur le site Maxim.

Problème d'interruption dans le programme de lecture et d'envoie des données :
Cause : Le flag qui détecte l'interruption.
Solution : Le remettre à zéro après chaque interruption.

La broche d'interruption de l'horloge RTC ne fonctionnait pas :
Cause : Omission d'une résistance faisant la liaison entre cette broche et un Vcc.
Solution : Ajout de cette résistance.

Le MSP430 sortant des signaux logiques hauts de 3.6V, il a fallu ajouter des diviseurs de tensions pour le Chip Select, l'horloge SPI et l'envoie de données SPI.

Inversion du signal d'interruption :
Cause : L'horloge RTC envoie un signal logique haut au repos et bas lorsqu'il y a une alarme.
Solution : Config du registre d'interruption du MSP430 pour détecter les fronts descendants.

Un des problèmes majeurs qui nous a pris le plus de temps à résoudre a été le fonctionnement de la liaison SPI.
Cause : Après avoir revu tous les registres de configuration, nous pensons que cela était simplement dû à l'utilisation du logiciel Code Composer. En effet pendant qu'un binôme utilisait l'ordinateur portable avec le logiciel IAR, l'autre était sur le PC de la salle et seul Code Composer fonctionne sur ce PC.
Solution : Utilisation d' IAR sur le PC portable.

Problèmes non résolus :

Lecture des données envoyées par l'horloge :
Cause probable : Sur l'analyseur logique, nous observons bien les données envoyés par l'horloge RTC, par exemple nous voyons défiler les secondes si on envoie dans une boucle une demande de lecture des secondes. Le problème est que l'affichage de cette donnée sur le PC est erroné. Nous pensons que ce problème vient soit du programme de lecture ou alors de la valeur des états logiques hauts qui est de 1.8V.
Solution à tester pour les prochains étudiants : Amplifier le signal de réception pour avoir des états hauts à 3.8V. Par manque de temps, nous n'avons pas pu le tester.

Conflit entre la liaison wifi et l'interruption du port 2 :
Cause : Cela est un problème constructeur et vient du fait que la carte eZ430-RF2500 utilise le port d'interruption PORT2_VECTOR pour la liaison wifi et pour les broches physiques P2.X. De ce fait, on ne peut utiliser que l'un ou l'autre.
Solution : Aucune à notre connaissance.

La mémoire flash : impossibilité d'effectuer plusieurs écritures successives sur le même segment de la mémoire.
Solution à tester : Réalisation d'une fonction qui génère des adresses de segment aléatoires.

CONCLUSION

Ce projet a été pour nous un grand enrichissement technique et nous a permis de nous familiariser avec les travaux de recherches. Ces 10 semaines de projet nous ont confortés dans l'idée que nous nous faisions de la recherche.

Nous avons appris à obtenir des résultats en partant de rien au départ.

Bien au-delà du travail que nous avons effectué, nous avons pu voir une vision globale de la recherche et des problèmes auxquels on est confrontées et comment les résoudre.

Le but principal de notre projet est de réduire la consommation d'énergie des capteurs enfouis pour réduire les impacts environnementaux et c'est pourquoi il est nécessaire de continuer la recherche dans ce domaine.

Ce projet a accrue notre motivation de devenir ingénieur, et pourquoi pas s'orienter vers une thèse qui pourra apporter beaucoup à la société d'aujourd'hui.

Annexes

Présentation du logiciel IAR

L'eZ430-RF2500 utilise les logiciels *IAR Embedded Workbench* ou *Code Composer Essentials* comme environnements de développement pour écrire, télécharger et déboguer une application. Le déboguer permet d'exécuter une application en pas à pas ce qui est très pratique en cas de problème.
Lors de ce projet nous avons choisi d'utiliser le logiciel IAR, plus complet que Code composer et aussi parce que Texas Instrument nous fournit dans le CD d'installation un squelette de programme IAR en langage C avec différentes librairies très utiles.

Création d'un nouveau projet :

1- Allez dans **Project->Create New Project**
2- Comme dans la *figure6* choisir **MSP430** dans le champ **tool chain** et **empty project**.
3- Cliquez **OK** pour créer un nouveau projet avec *les paramètres par défaut*. Le nouveau projet va apparaitre dans le **workspace** voir *figure7*.
4- Choisir **File->Add Files** pour ouvrir la boite de dialogue et rajoutez les fichiers .c que vous souhaitiez mettre dans votre projet. Vous pouvez aussi créer un nouveau fichier .C. voir *figure8*

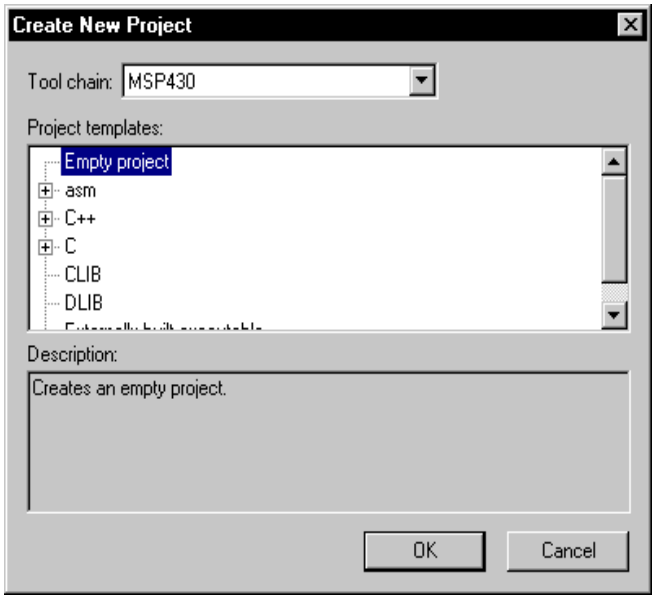

Figure 6. Create New Project

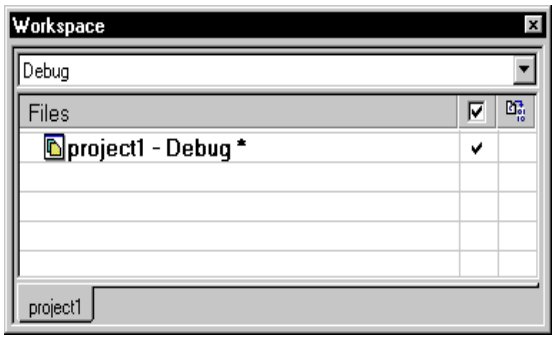

Figure 7. Workspace

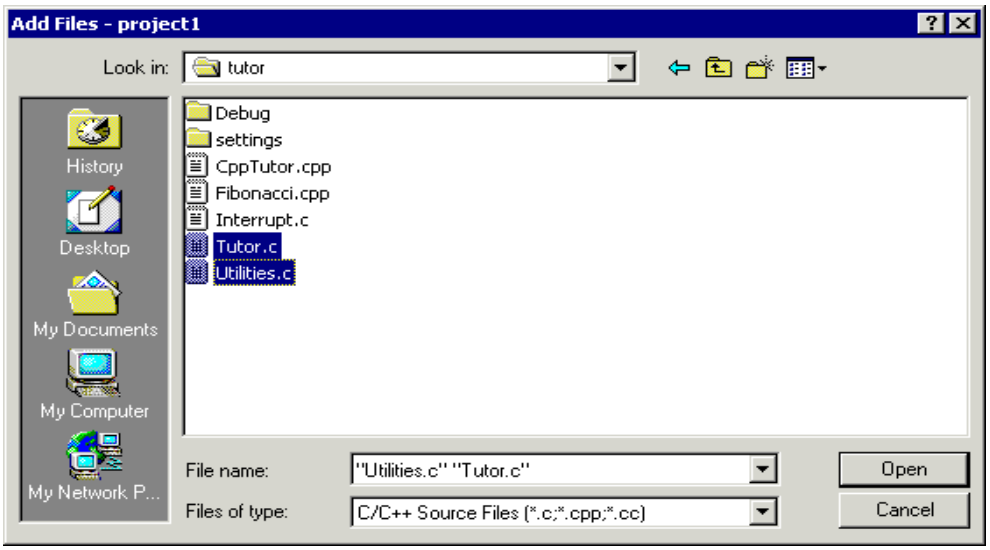

Figure 8. Add files

Modification des options du projet :

Il faut impérativement effectuer ces étapes, la configuration par défaut n'est pas la bonne.

1- Allez dans **Project->Option** et choisissez **MSP430F2274** comme dans la *figure9*.
2- Dans la même boite de dialogue choisir cette fois-ci **Debugger** dans la colonne à gauche de la fenêtre. Modifiez le champ **Driver** en choisissant **FET Debugger**. Voir *figure10*.
3- Dernière étape, dans la catégorie **FET Debugger** sélectionnez **Texas Instrument USB-IF**. figure 11

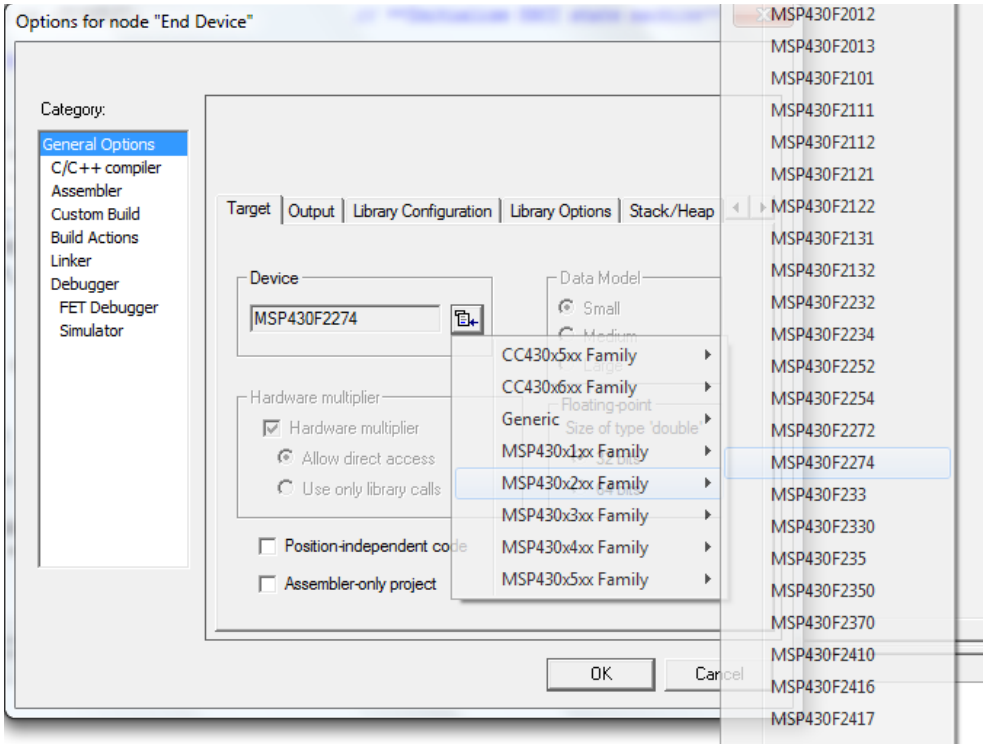

Figure 9. Choix du microcontrôleur

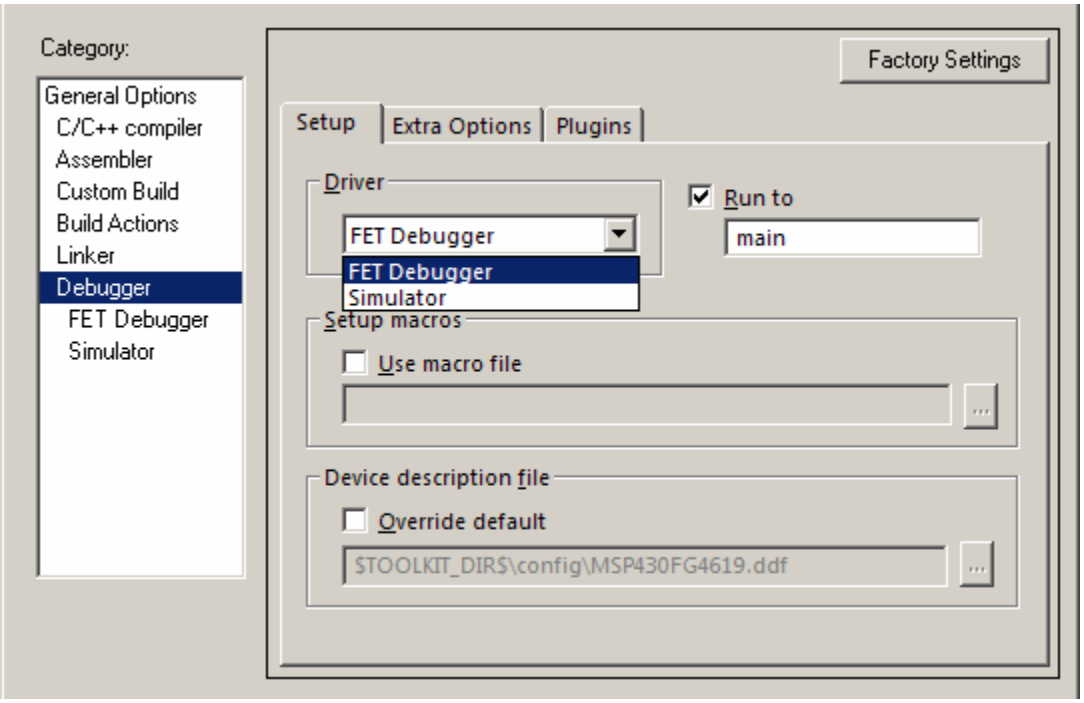

Figure 10. Choix du Driver

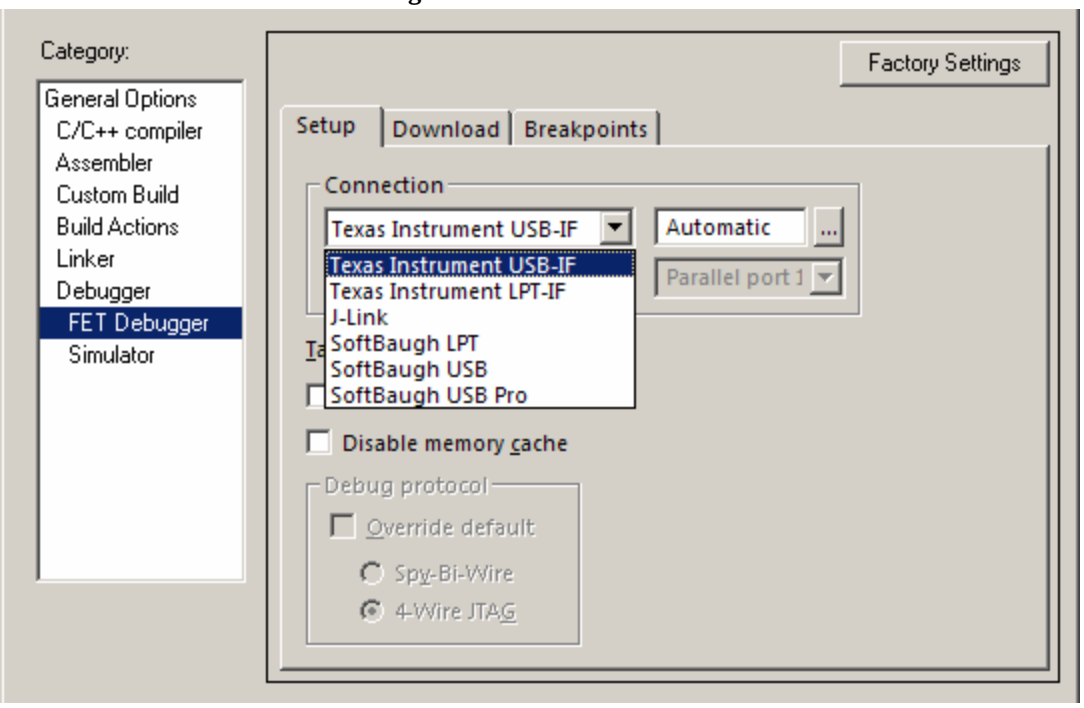

Figure 11. Choix de la Connexion

Télécharger et debugger le projet :

Une fois que vous aurez terminé de développer votre application, il va falloir la charger dans l'MSP430. Mais avant choisissez **Project->Rebuild All** afin de compiler votre projet. Si vous n'avez ni erreur ni warning votre fenêtre de messages doit ressembler à la *figure12*

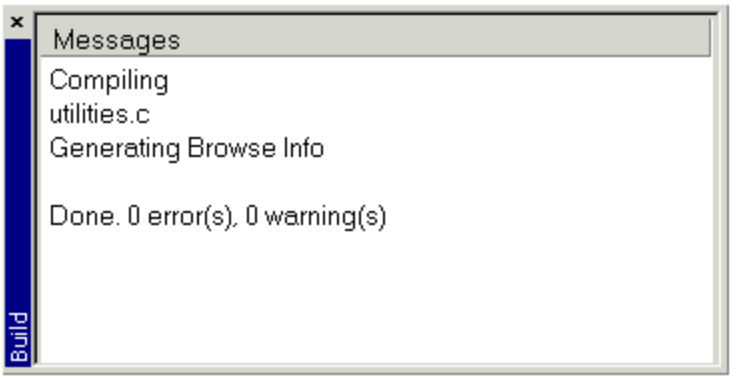

Figure 12. Fenêtres des messages

1- Choisir **Project->Debug** pour charger le projet. Ainsi le *Debugger* sera en attente.
2- **Debug->Go** pour lancer le *Debugging* et **Debug->Stop Debugging** pour arrêter.

Si vous utilisez les supports batterie, le programme que vous auriez implanté dans l'MSP430 se lancera automatiquement une fois alimenté.

Important pour les étudiants qui désirent reprendre notre projet :

Une fois que vous aurez récupéré l'ensemble du projet :

1- Ouvrir **IAR**.
2- Choisir **file-> Open-> workspace** et aller dans l'emplacement où vous avez stocké le projet.
3- Ouvrir le *workspace*.

Une fois le workspace ouvert vous trouverez dans la colonne workspace deux sous projets (Acces point et End device). Chacun des deux sous projets correspond à l'application à implanter en fonction du type du support à programmer (maitre ou capteur).

4- Déplier le sous-programme, vous trouverez le fichier C dans le dossier Application.

Programmes fonctionnels

Programme 1 :

Description : Ce programme permet d'acquérir une température, la sauvegarder et de l'envoyer à l'Access Point.

Code Access Point :

```
#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"

#include "msp430x22x4.h"
#include "vlo_rand.h"

#define MESSAGE_LENGTH 3
void TXString( char* string, int length );
void MCU_Init(void);
void transmitData(int addr, char msg[MESSAGE_LENGTH] );
void transmitDataString(char addr[4], char msg[MESSAGE_LENGTH]);
void createRandomAddress();


__no_init volatile int tempOffset @ 0x10F4; // Temperature offset set at production
__no_init volatile char Flash_Addr[4] @ 0x10F0; // Flash address set randomly

// reserve space for the maximum possible peer Link IDs
static linkID_t sLID[NUM_CONNECTIONS];
static uint8_t  sNumCurrentPeers;

// callback handler
static uint8_t sCB(linkID_t);

// work loop semaphores
static uint8_t sPeerFrameSem;
static uint8_t sJoinSem;
static uint8_t sSelfMeasureSem;


void main (void)
{
  addr_t lAddr;
  bspIState_t intState;

  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  {
  // delay loop to ensure proper startup before SimpliciTI increases DCO
  // This is typically tailored to the power supply used, and in this case
  // is overkill for safety due to wide distribution.
    volatile int i;
    for(i = 0; i < 0xFFFF; i++){}
  }
  if( CALBC1_8MHZ == 0xFF )                 // Do not run if cal values are erased
  {
    volatile int i;
    P1DIR |= 0x03;
    BSP_TURN_ON_LED1();
    BSP_TURN_OFF_LED2();
```

```
  while(1)
  {
    for(i = 0; i < 0x5FFF; i++){}
    BSP_TOGGLE_LED2();
    BSP_TOGGLE_LED1();
  }
}

BSP_Init();

if( Flash_Addr[0] == 0xFF &&
    Flash_Addr[1] == 0xFF &&
    Flash_Addr[2] == 0xFF &&
    Flash_Addr[3] == 0xFF )
{
  createRandomAddress();                // set Random device address at initial startup
}
lAddr.addr[0]=Flash_Addr[0];
lAddr.addr[1]=Flash_Addr[1];
lAddr.addr[2]=Flash_Addr[2];
lAddr.addr[3]=Flash_Addr[3];
SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

MCU_Init();
SMPL_Init(sCB);

// main work loop
while (1)
{
  // Wait for the Join semaphore to be set by the receipt of a Join frame from a
  // device that supports and End Device.

  if (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS))
  {
    // listen for a new connection
    SMPL_LinkListen(&sLID[sNumCurrentPeers]);
    sNumCurrentPeers++;
    BSP_ENTER_CRITICAL_SECTION(intState);
    if (sJoinSem)
    {
      sJoinSem--;
    }
    BSP_EXIT_CRITICAL_SECTION(intState);
  }

  // if it is time to measure our own temperature...
  if(sSelfMeasureSem)
  {
    BSP_TOGGLE_LED1();
    sSelfMeasureSem = 0;
  }

  // Have we received a frame on one of the ED connections?
  // No critical section -- it doesn't really matter much if we miss a poll
  if (sPeerFrameSem)
  {
    uint8_t     msg[MAX_APP_PAYLOAD], len, i;

    // process all frames waiting
    for (i=0; i<sNumCurrentPeers; ++i)
    {
      if (SMPL_Receive(sLID[i], msg, &len) == SMPL_SUCCESS)
      {
        ioctlRadioSiginfo_t sigInfo;
        sigInfo.lid = sLID[i];
        SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SIGINFO, (void *)&sigInfo);
        transmitData( i, (char*)msg );
```

```
        BSP_TOGGLE_LED2();
        BSP_ENTER_CRITICAL_SECTION(intState);
        sPeerFrameSem--;
        BSP_EXIT_CRITICAL_SECTION(intState);
       }
     }
   }
  }
}

/*-------------------------------------------------------------------------
*
-------------------------------------------------------------------------*/
void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();    // first byte can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000 );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO to 1MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;             // MCLK/3 for Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;                    // Clear LOCK & LOCKA bits
  FCTL1 = FWKEY + WRT;                      // Set WRT bit for write operation

  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;                            // Clear WRT bit
  FCTL3 = FWKEY + LOCKA + LOCK;             // Set LOCK & LOCKA bit
}

/*-------------------------------------------------------------------------
*
-------------------------------------------------------------------------*/
void transmitData(int addr, char msg[MESSAGE_LENGTH] )
{
  char addrString[4];

  addrString[0] = '0';
  addrString[1] = '0';
  addrString[2] = '0'+(((addr+1)/10)%10);
  addrString[3] = '0'+((addr+1)%10);


  transmitDataString( addrString, msg );
}

/*-------------------------------------------------------------------------
*
-------------------------------------------------------------------------*/
void transmitDataString(char addr[4], char msg[MESSAGE_LENGTH] )
{
  char temp_string[] = {" XX.XC"};
  int temp = msg[0] + (msg[1]<<8);

  if( temp < 0 )
  {
    temp_string[0] = '-';
    temp = temp * -1;
  }
```

```
  else if( ((temp/1000)%10) != 0 )
  {
    temp_string[0] = '0'+((temp/1000)%10);
  }
  temp_string[4] = '0'+(temp%10);
  temp_string[2] = '0'+((temp/10)%10);
  temp_string[1] = '0'+((temp/100)%10);


    char output_short[] = {"\r\n$ADDR,-XX.XC,V.C"};



    output_short[8] = temp_string[0];
    output_short[9] = temp_string[1];
    output_short[10] = temp_string[2];
    output_short[11] = temp_string[3];
    output_short[12] = temp_string[4];
    output_short[13] = temp_string[5];

    output_short[15] = '0'+(msg[2]/10)%10;
    output_short[17] = '0'+(msg[2]%10);
    output_short[3] = addr[0];
    output_short[4] = addr[1];
    output_short[5] = addr[2];
    output_short[6] = addr[3];
    TXString(output_short, sizeof output_short );
}

/*------------------------------------------------------------------------------
*
------------------------------------------------------------------------------*/
void TXString( char* string, int length )
{
  int pointer;
  for( pointer = 0; pointer < length; pointer++)
  {
    volatile int i;
    UCA0TXBUF = string[pointer];
    while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
  }
}

/*------------------------------------------------------------------------------
*
------------------------------------------------------------------------------*/
void MCU_Init()
{
  BCSCTL1 = CALBC1_8MHZ;                    // Set DCO
  DCOCTL = CALDCO_8MHZ;

  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  TACCTL0 = CCIE;                           // TACCR0 interrupt enabled
  TACCR0 = 12000;                           // ~1 second
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  P3SEL |= 0x30;                            // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 = UCSSEL_2;                      // SMCLK
  UCA0BR0 = 0x41;                           // 9600 from 8Mhz
  UCA0BR1 = 0x3;
  UCA0MCTL = UCBRS_2;
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt
  __enable_interrupt();
}
/*------------------------------------------------------------------------------
* Runs in ISR context. Reading the frame should be done in the
```

```
* application thread not in the ISR thread.
-------------------------------------------------------------------------------*/
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    sPeerFrameSem++;
  }
  else
  {
    sJoinSem++;
  }
  // leave frame to be read by application.
  return 0;
}

/*-------------------------------------------------------------------------------
* Timer A0 interrupt service routine
-------------------------------------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 1;
}

/*-------------------------------------------------------------------------------
* USCIA interrupt service routine
-------------------------------------------------------------------------------*/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{

}
```

Code End Device :

```
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"

void MCU_Init(void);
void read_flash(int num_mes);
void write_flash(uint8_t * value, int num_mes);
uint8_t val[3];
void erase_flash(void);
linkID_t linkID1;
int nbr_mes=0;
__no_init volatile int tempOffset @ 0x10F4; // Temperature offset set at production
__no_init volatile char Flash_Addr[4] @ 0x10F0; // Flash address set randomly

void createRandomAddress();

void main (void)
{
  addr_t lAddr;
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  {
    volatile int i;
    for(i = 0; i < 0xFFFF; i++){}
  }
  if( CALBC1_8MHZ == 0xFF )                 // Do not run if cal values are erased
  {
    volatile int i;
```

```
  P1DIR |= 0x03;
  BSP_TURN_ON_LED1();
  BSP_TURN_OFF_LED2();
  while(1)
  {
    for(i = 0; i < 0x5FFF; i++){}
    BSP_TOGGLE_LED2();
    BSP_TOGGLE_LED1();
  }
}



P1DIR |= 0x03 ;        // config BP
P1DIR &= ~0x04 ;
P1REN |= 0x04 ;
P1IE |= 0x04 ;

P2DIR = 0x27;
P2OUT = 0x00;
P3DIR = 0xC0;
P3OUT = 0x00;
P4DIR = 0xFF;
P4OUT = 0x00;

BSP_Init();

if( Flash_Addr[0] == 0xFF &&
    Flash_Addr[1] == 0xFF &&
    Flash_Addr[2] == 0xFF &&
    Flash_Addr[3] == 0xFF )
{
  createRandomAddress();                // set Random device address at initial startup
}
lAddr.addr[0]=Flash_Addr[0];
lAddr.addr[1]=Flash_Addr[1];
lAddr.addr[2]=Flash_Addr[2];
lAddr.addr[3]=Flash_Addr[3];
SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);


BCSCTL1 = CALBC1_8MHZ;                  // Set DCO after random function
DCOCTL = CALDCO_8MHZ;


TACCR0 = 30;                           // Delay to allow Ref to settle
TACCTL0 &= ~CCIE;
TACTL = TASSEL_2 + MC_1;                // TACLK = SMCLK, Up mode



while (SMPL_NO_JOIN == SMPL_Init((uint8_t (*)(linkID_t))0))
{
  BSP_TOGGLE_LED1();
  BSP_TOGGLE_LED2();
  __bis_SR_register(LPM0_bits + GIE);     // LPM3 with interrupts enabled
}


 while (SMPL_SUCCESS != SMPL_Link(&linkID1))
{
  __bis_SR_register(LPM0_bits + GIE);
  BSP_TOGGLE_LED1();
  BSP_TOGGLE_LED2();
}

// Turn off all LEDs
```

```
  if (BSP_LED1_IS_ON()) BSP_TOGGLE_LED1();
  if (BSP_LED2_IS_ON()) BSP_TOGGLE_LED2();


  __bis_SR_register(LPM0_bits + GIE);
  while (1);

}

void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();    // first byte can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000 );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_8MHZ;                    // Set DCO to 1MHz
  DCOCTL = CALDCO_8MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;             // MCLK/3 for Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;                    // Clear LOCK & LOCKA bits
  FCTL1 = FWKEY + WRT;                      // Set WRT bit for write operation

  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;                            // Clear WRT bit
  FCTL3 = FWKEY + LOCKA + LOCK;             // Set LOCK & LOCKA bit
}


void erase_flash () // erase segments B, C & D
{
      char *Flash_ptrB,*Flash_ptrC,*Flash_ptrD;

      Flash_ptrB = (char *)0x1080;
      Flash_ptrC = (char *)0x1040;              // Initialize Flash segment C ptr
      Flash_ptrD = (char *)0x1000;

      FCTL3 = FWKEY;
      FCTL1 = FWKEY + ERASE ;                   // Set Erase bit, allow interrupts

      *Flash_ptrB = 0;
      *Flash_ptrC = 0;
      *Flash_ptrD = 0;

      FCTL1 = FWKEY;
      FCTL3 = FWKEY + LOCK;
}

void write_flash (uint8_t* value,int num_mes)
{
      int i;
      char *Flash_ptr;
      Flash_ptr = (char *)0x1000 + num_mes*3;

      FCTL3 = FWKEY;
      FCTL1 = FWKEY + WRT;

      for(i=0;i<3;i++) *Flash_ptr++=value[i];

      FCTL1 = FWKEY;
      FCTL3 = FWKEY + LOCK;
```

```
}

void read_flash (int num_mes)
{
      int i;
      char *Flash_ptr;
      Flash_ptr = (char *)0x1000 + num_mes*3;

      FCTL1 = FWKEY;
      FCTL3 = FWKEY;

      for(i=0;i<3;i++) val[i]=*Flash_ptr++;

      FCTL1 = FWKEY;
      FCTL3 = FWKEY + LOCK;
}


#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{

   volatile long temp;
   int degC, volt,i,j;
   int results[2];
   int nbr_mes_max=10;
   uint8_t  msg[3];

   P1IFG = 0;


   ADC10CTL1 = INCH_10 + ADC10DIV_4;       // Temp Sensor ADC10CLK/5
   ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ADC10SR ;
   for( degC = 240; degC > 0; degC-- );    // delay to allow reference to settle
   ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
   while (ADC10CTL1 & ADC10BUSY);
   results[0] = ADC10MEM;
   ADC10CTL0 &= ~ENC;

   ADC10CTL1 = INCH_11;                    // AVcc/2
   ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + REF2_5V ;
   for( degC = 240; degC > 0; degC-- );    // delay to allow reference to settle
   ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
   while (ADC10CTL1 & ADC10BUSY);
   results[1] = ADC10MEM;
   ADC10CTL0 &= ~ENC;
   ADC10CTL0 &= ~(REFON + ADC10ON);        // turn off A/D to save power

   temp = results[0];
   degC = ((temp - 673) * 4230) / 1024;
   if( tempOffset != 0xFFFF )
   {
    degC += tempOffset;
   }
    temp = results[1];
    volt = (temp*25)/512;
    msg[0] = degC&0xFF;
    msg[1] = (degC>>8)&0xFF;
    msg[2] = volt;


     nbr_mes++;
     write_flash(msg,nbr_mes-1);

     if(nbr_mes == nbr_mes_max)
     {
        P1IFG = 0;
```

```
        P1IE &= ~0x04;

        for(j=1;j<=nbr_mes_max;j++)
        {
          for(i=0;i<3;i++) val[i]=0;
          read_flash(j-1);
          for(i=0;i<3;i++) msg[i]=val[i];
          if (SMPL_SUCCESS == SMPL_Send(linkID1, msg, sizeof(msg)))
          BSP_TOGGLE_LED2();
          for(i = 0; i < 0xFFFF; i++){}
        }
        nbr_mes=0;
        P1IE |= 0x04;
        __bis_SR_register(CPUOFF + GIE);
    }
}
```

Programme 2 :

Description : Ce programme permet de configurer l'horloge ainsi que l'alarme. A chaque interruption, le programme exécute la routine d'interruption qui consiste à allumer une LED. En effet, l'action « allumer une LED » peut être remplacer par la mesure de la température et l'envoi vers l'Access Point.

```
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"
#include "msp430x22x4.h"

void MCU_Init(void);
void write_SPI(int addr_W, int data);

void main (void)
{

 WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

 MCU_Init();

// BSP_TOGGLE_LED2();

  // set alarm & interup
 write_SPI(0x8D,0x85);       //85
 write_SPI(0x8E,0x00);    //alarm flag reset
 write_SPI(0x8F,0x00);

 // config Time
 write_SPI(0x80,0x00);   // H_sec
 write_SPI(0x81,0x00);   // sec
 write_SPI(0x82,0x30);   //minutes
 write_SPI(0x83,0x02);   // hour
 write_SPI(0x84,0x04);   // day
 write_SPI(0x85,0x04);   // date
 write_SPI(0x86,0x04);   //month
 write_SPI(0x87,0x04);   // year

 //config alarm

 write_SPI(0x88,0x00);   //H_sec
 write_SPI(0x89,0x5);   //sec
 write_SPI(0x8A,0x30);   //minutes
 write_SPI(0x8B,0x02);   // hour
```

```
  write_SPI(0x8C,0x84);   // date */


  __bis_SR_register(LPM0_bits + GIE);
  while (1);
}

void MCU_Init()
{

  BCSCTL1 = CALBC1_1MHZ;                  // Set DCO
  DCOCTL  = CALDCO_1MHZ;
  BCSCTL2 = DIVS_1;                       // SMCLK= 1MHZ/2 = 500Khz

  // config leds
  P1DIR = 0xFF;
  P1OUT = 0x00;


  P2DIR &= ~0x01;
  P2SEL &= ~0x01;
  P2IE  |= 0x01;
  P2IES |= 0x01;


//  init USCI_B0
  P3SEL |= 0x3E;      //3E                  // P3.1/2.3 USCI_B0 option selected
  P3DIR |= 0x01;                          // P3.0 output direction
  UCB0CTL0 |= UCMSB + UCMST + UCSYNC;       // 3-pin, 8-bit SPI mstr, MSB 1st  + UCSYNC
  UCB0CTL1 |= UCSSEL_2;                   // SMCLK
  UCB0BR0 = 1;      //3
  UCB0BR1 = 0;
  UCB0CTL1 &= ~UCSWRST;                   // **Initialize USCI state machine**

}

void write_SPI(int addr_W, int data)
{
      P3OUT &= ~0x01;
      UCB0TXBUF = addr_W;
      while (!(IFG2 & UCB0TXIFG));


      UCB0TXBUF = data;
      while (!(IFG2 & UCB0TXIFG));
      while (!(IFG2 & UCB0RXIFG));

      P3OUT |= 0x01;
      __delay_cycles(30);
}

#pragma vector = PORT2_VECTOR
__interrupt void p2_isr(void)
{

  P2IFG &= ~0x01;



  MRFI_GpioIsr();
  BSP_TOGGLE_LED2();



  __bis_SR_register(CPUOFF + GIE);

}
```