François SCHNEIDER – Lycée Victor-Hugo BESANÇON.

STS SE

Développement de microcontrôleurs MSP430 avec validation fonctionnelle PROTEUS

Microprocesseur MSP430-F249 – Interruptions port P1, mise en veille, Code Composer Studio, PROTEUS Simulation Validation

Prérequis : langage C, Code Composer Studio, PROTEUS ISIS simulation d'un microprocesseur.

Notices techniques du microprocesseur MSP430-F249 :

- MSP430x23x, MSP430x24x(1), MSP430x2410 MIXED SIGNAL MICROCONTROLLER
- MSP430x2xx Family : notice technique famille MSP430x2xx.

Description du support d'activités :

Nous proposons de développer les fonctions drivers bas niveau d'un ensemble GPS. Le schéma modèle pour la simulation avec PROTEUS est donné page suivante.

- Le développement est effectué afin d'obtenir la consommation la plus faible.

Ceci veut dire, qu'il faut utiliser les modes de veille du microprocesseur et que cela impose de travailler avec une programmation événementielle (interruption).

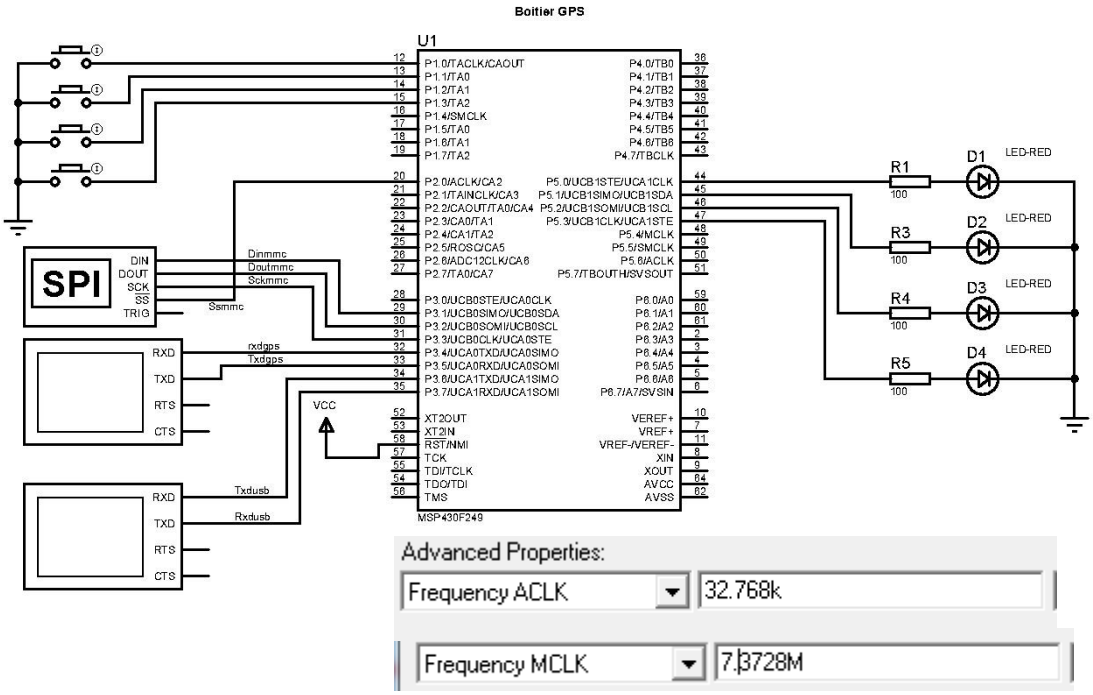
Nous trouvons les éléments :

- 4 boutons reliés aux entrées P1.0 à P1.3 du port P1, qui sont interruptibles et qui possèdent en interne des résistances de tirage. Ces résistances de tirage seront utilisées pour fixer le niveau haut sur ces entrées.
- 4 Leds sont reliées à P5.0 à P5.3, elles serviront à afficher le mode de fonctionnement du GPS.
- Une mémoire MMC utilisée en mode SPI. Elle est reliée au module SPI B0.
- Un GPS série : 9600Bd, 1 bit stop pas de parité. Il est relié au module série A0.
- Un module USB/RS232 :19200Bd, 1 bit stop pas de parité. Il est relié au module série A1.
- Un quartz 7,3728MHz relié au processeur.

Remarques :

- Le nombre d'interfaces série impose un microcontrôleur possédant au moins un module SPI et deux module RS232. Pour cette raison nous avons choisi un MSP430F249.
- Le composant MSP430F249 possède une gestion de modes de veille très évoluée. Il est donc bien adapté au fonctionnement désiré.
- 2 systèmes d'horloge pour minimiser la consommation :
    - « XT2 Oscillator » → **MCLK,**
    - « LFXT1 Oscillator » → 32768Hz → **ACLK**.
- Certaines fonctions ne sont pas implémentées ici : affichage graphique, oscillateur à quartz 7,3728MHz…

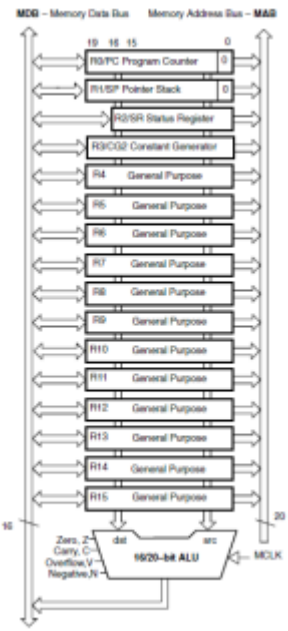
Schéma pour la simulation sous Proteus :



Microprocesseur F249 - les registres – mode de fonctionnement :

Vous trouvez les informations pour compléter cette partie paragraphe 4, document « slau144e.pdf ».

- Le processeur possède 4 registres particuliers : R0 à R3, donner la taille et le rôle de chacun.
- Une ressource s'appelle ALU, donner sa fonction.



Microprocesseur F249, mise en veille, interruptions masquables - SR ou STATUS REGISTER BITS :

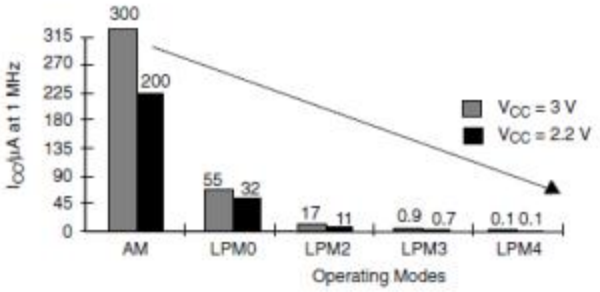
- Bit GIE : indiquer son rôle. Que veut dire interruption masquable, non masquable ?
- Bits SCG1, SCG0, OSCOFF et CPUOFF : indiquer leurs rôles.

La fonction « _BIS_SR() » permet d'écrire dans le registre « STATUT ».

- En vous aidant du fichier de déclaration « msp430f249.h », quelle action a sur le processeur la ligne de programme suivante :
  « _BIS_SR(LPM4_bits + GIE) » ?

Les modes de fonctionnement, la consommation et les ressources :

. Typical Current Consumption of 21x1 Devices vs Operating Modes



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 disabled

Applications à faible consommation :

Issue de la notice MSP430F249.

Often, the most important factor for reducing power consumption is using the MSP430's clock system to maximize the time in LPM3. LPM3 power consumption is less than 2 µA typical with both a real-time clock function and all interrupts active. A 32-kHz watch crystal is used for the ACLK and the CPU is clocked from the DCO (normally off) which has a 6-µs wake-up.

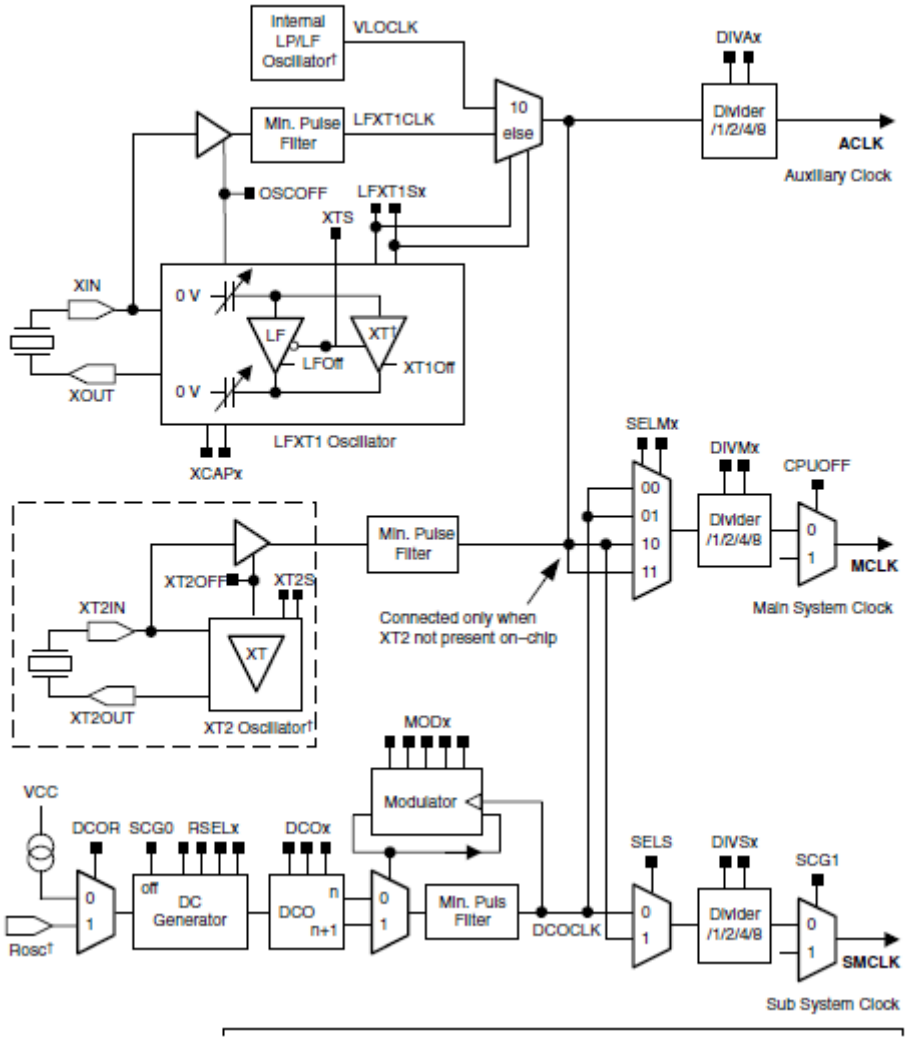
- Use interrupts to wake the processor and control program flow.
- Peripherals should be switched on only when needed.
- Use low-power integrated peripheral modules in place of software driven functions. For example Timer_A and Timer_B can automatically generate PWM and capture external timing, with no CPU resources.
- Calculated branching and fast table look-ups should be used in place of
- flag polling and long software calculations.
- Avoid frequent subroutine and function calls due to overhead.
- For longer software routines, single-cycle CPU registers should be used.

Remarques :

- La suite va être développée comme une suite de fiches de découverte de la programmation des drivers bas niveau (écrire, lire un octet…) de cette application.
- Le microprocesseur fonctionnera en mode LPM3 en mode veille, en lpm0 en mode normal et AM lorsqu'il sera en interruption.

Le système d'horloge du MSP430F249 voir chapitre 5.

Figure 5–1. Basic Clock Module+ Block Diagram



3 horloges ACLK, MCLK et SMCLK sont produites à partir de différentes sources. Citer les différentes sources d'horloge :

-

4 registres permettent de gérer le système d'horloge : DCOCTL, BCSCTL1, BCSCTL2, and BCSCTL3. On désire avoir une fréquence de 32768Hz sur ACLK. XIN et XOUT sont câblées avec un quartz horloger à 32,768kHz. Le système d'horloge est configuré de façon à avoir en LFXT1CLK le signal XIN. Définir la valeur à donner à LFXT1sx et DIVAx.

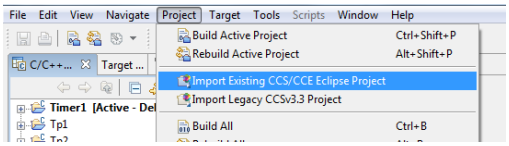
-

Organisation d'un projet avec CCS. Production du projet et de la base du programme : chaque nouveau projet sera réalisé sur la même base.

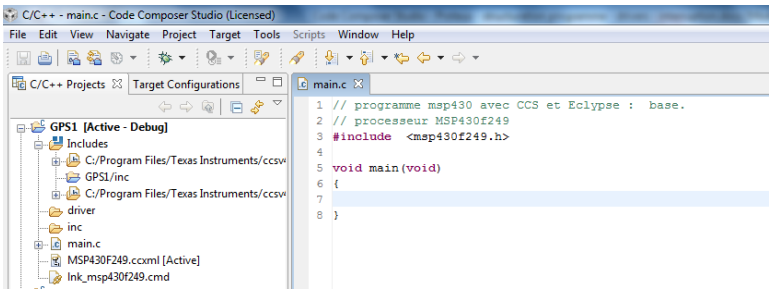
Production du projet et de l'organisation des dossiers.

- Créer un projet nommé « Gps1 » avec un microprocesseur « MSP430F249 ».
- Définir l'extension du fichier de sortie : « .coff ».
- Créer un dossier nommé « inc » pour placer les fichiers « en-tête » personnels.
- Déclarer le dossier « inc » dans la liste des inclusions (fichiers .h).
- Créer un dossier nommé « driver » pour y placer les fichiers « librairies » personnels (fichiers .c).
- Rendez votre nouveau projet comme « projet actif ».
- Vous créez bien sur un fichier « main.c » dans lequel vous écrivez la fonction main()

Bien sûr, si vous possédez déjà un projet construit suivant cette organisation, il vous est possible de le copier dans votre dossier « Espace de travail » de CCS et de l'importer.



Voici l'allure de votre projet, ainsi que le fichier main.c.



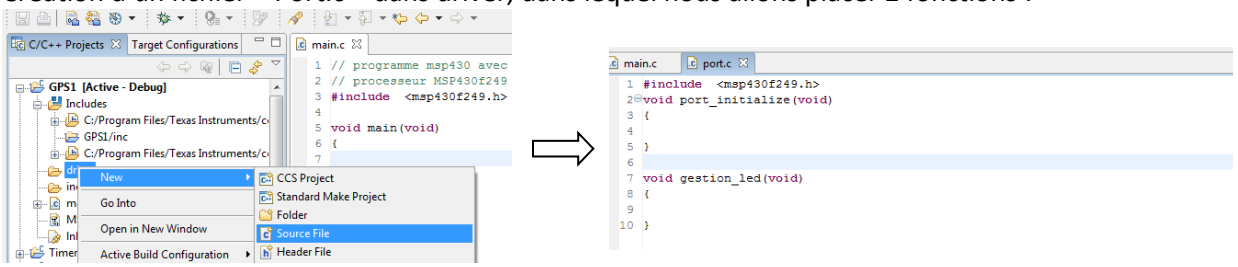
Activité 1. Cette activité est simple, et va vous permettre d'apprendre à travailler avec les dossiers « inc » et « driver ».

1. Le programme permet d'afficher sur les LEDS, l'état des boutons poussoirs.

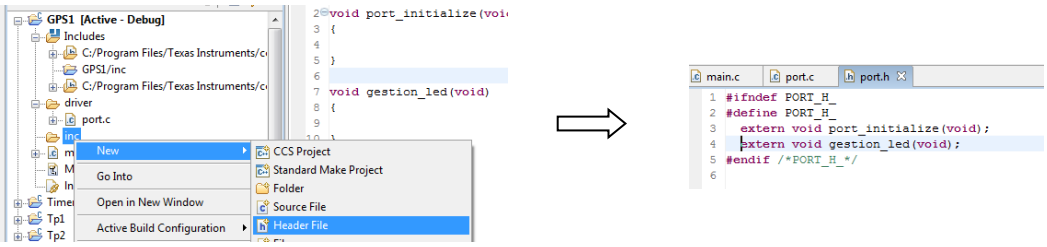
   Démarche de conception du programme :

   - Création des fichiers programmes '*.C' dans le dossier driver.
   - Création des fichiers « entête » dans le dossier inc.
   - Modification du fichier main :
     - Déclaration des fichiers « entête »
     - Appel des fonctions.

   Création d'un fichier « Port.c » dans driver, dans lequel nous allons placer 2 fonctions :



Création des fichiers « entête » « port.h » dans le dossier inc, dans lequel nous allons déclarer le prototype des fonctions de « port.c »

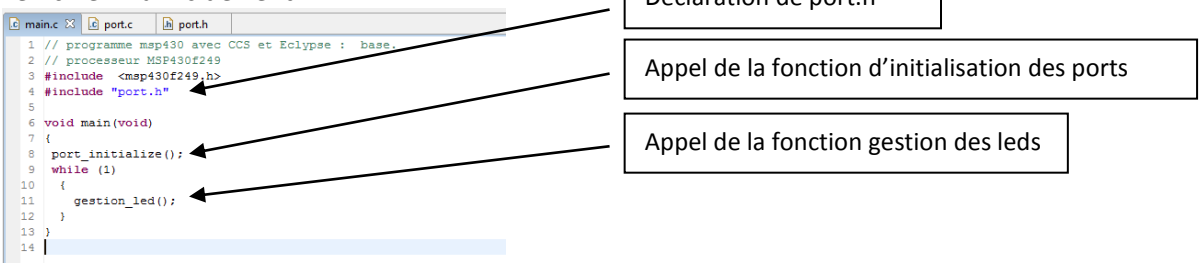


On constate au niveau du fichier « port.h » les directives, qui ont été produites automatiquement :

```
#ifndef PORT_H_
#define PORT_H_
#endif /*PORT_H_*
```

Elles permettent, si ce fichier « entête » est inclus dans plusieurs fichiers programmes, d'avoir la partie déclaration incluse une seule fois.

Le fichier main.c devient :

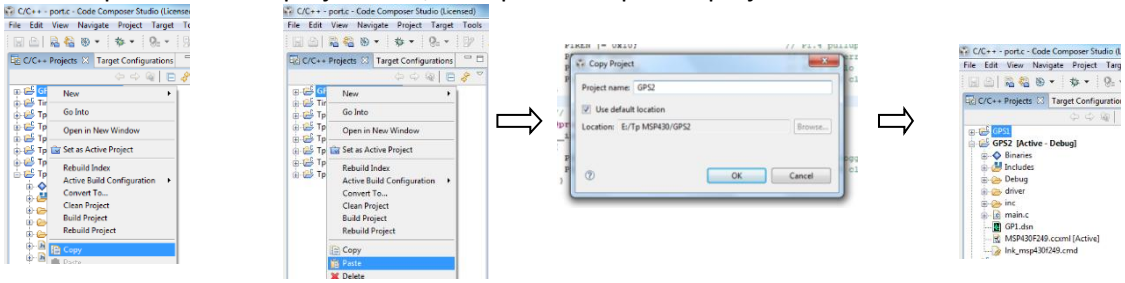


Indiquer les registres qui permettent de configurer le mode de fonctionnement des ports.
Compléter le programme et valider le fonctionnement par simulation.

2. Modification du programme.
Le nouveau programme va piloter seulement la LED D1. Lorsqu'un seul des boutons poussoir est appuyé, la LED éclaire. Modifier le programme et valider le fonctionnement.

Activité 2. Cette activité va consister à mettre le processeur en veille, le réveiller par un appui sur un bouton poussoir, à effectuer l'action associée et à mettre à nouveau le processeur en veille

Dans cette partie nous allons apprendre à utiliser une interruption sur les entrées du port P1.
Pour ne pas détruire le projet GPS1, vous pouvez copier ce projet vers GPS2.



Sur le projet touche droite de la souris

Sur la zone projet touche droite de la souris

1. Le programme permet d'afficher sur les LEDS, l'état des boutons poussoirs.

   Pour obtenir un fonctionnement par interruption, il va falloir modifier :

   - la fonction gestion_led(),
   - la fonction main(),
   - la fonction port_initialize(),
   - et le fichier port.h.

   - La fonction gestion_led() devient une fonction interruption, elle se déclare ainsi :

```
#pragma vector=PORT1_VECTOR
__interrupt void gestion_led(void)
{
  …                       // écriture des lignes de gestions des leds
  P1IFG &= ~0x0F;         // effacement des flags
}
```

     - Indiquez le rôle de la ligne : P1IFG &= ~0x0F;
     - Cette fonction n'est plus appelée par le programme principal, mais par un appui sur un des boutons.

   - La fonction main() devient :

```
void main(void)
{
 port_initialize();
 _BIS_SR(GIE);  // activation interruptions
 while (1) { }
}
```

     - Quel est le rôle :

       - de l'instruction `_BIS_SR(GIE)`,
       - de la boucle `while (1) { }` ?

   - Voici le début de la fonction port_initialize() modifiée.

```
void port_initialize(void)
{
  P1DIR = 0x00;
  P1OUT = 0xFF;
  P1REN |= 0xFF;
  P1IE |= 0x0F;
  P1IES |= 0x0F;
  P1IFG &= ~0x0F;
  ….
}
```

     - Commenter chacune des lignes.

   - Fichier port.h : on enlève simplement la déclaration de la fonction « `gestion_led()` ».

   Modifier le programme et valider le fonctionnement.

2. **Reprendre la question 2. De l'activité A1.**

3. **Mise en veille.**
Pour passer à un fonctionnement basse consommation du processeur, c'est très simple, la fonction main() devient :

```
void main(void)
{
 port_initialize();
 _BIS_SR(LPM3_bits + GIE);  // Mode LPM3/ activation interruptions
}
```

- **Que se passe-t-il après que le processeur a exécuté la fonction :** `_BIS_SR(LPM3_bits + GIE);`

Modifier le programme et valider le fonctionnement.

Activité 3. Cette activité est une activité de synthèse : transmission série.

Vous utilisez l'interface série UCA0 en mode transmission série asynchrone : 8 bits de données, 1 bit de stop, pas de parité et 9600Bd.

Ecrire le driver permettant de transmettre à chaque appui sur un bouton poussoir le numéro du bouton appuyé. Valider le fonctionnement.