François SCHNEIDER – Lycée Victor-Hugo BESANÇON.

STS SE

Développement de microcontrôleurs MSP430 avec validation fonctionnelle PROTEUS

Découverte du processeur MSP430-F249

Premier programme Configuration Code Composer Studio et PROTEUS Simulation Validation

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

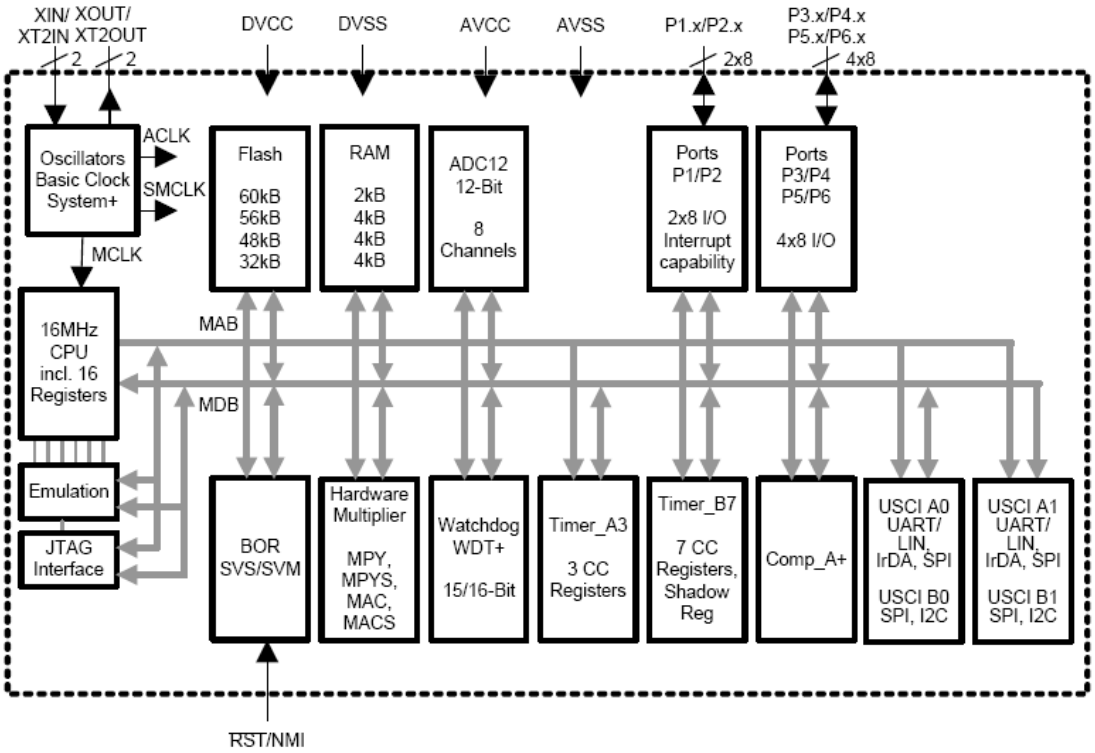
Processeur MSP430-F249 et outils :

Notices techniques du microprocesseur MSP430-F249 et d'outils de développement :

- MSP430x23x, MSP430x24x(1), MSP430x2410 MIXED SIGNAL MICROCONTROLLER
- MSP430x2xx Family : notice technique famille MSP430x2xx.
- MSP430-Hxxx-E HEADER BOARD : carte intéressante pour réaliser des projets sans souder le circuit CMS, disponible chez RS pour le F249.
- MSP-FET430U64 de Texas : Sonde émulation USB-JTAG avec carte cible.

Microprocesseur F249 : bilan rapide des ressources :

Ci-dessous vous trouvez le schéma bloc du F249. Vous repèrerez au fur et à mesure des questions ci-dessous les différentes fonctions :



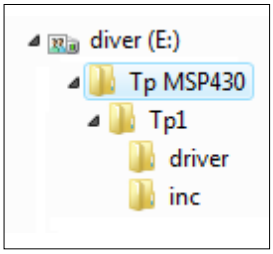
- Quelle est sa tension d'alimentation ?
- Expliquer ce que veut dire « 16-Bit RISC Architecture » :
- Décrire ses ressources en mémoire. Qu'est ce que de la mémoire Flash ?
- Expliquer le système d'horloge et les différents modes possibles :
- Quel est e rôle de l'interface JTAG ?
- Combien possède-t-il de ports entrée/sortie ? Expliquer la différence entre les P1/P2 et P3 à P5 ?
- Donner et expliquer les principales caractéristiques de ses ressources « Conversion analogique numérique ».
- De même avec les ressources « comparateurs ».
- De même avec les ressources « timer ».
- De même avec les ressources « interfaces série ».

Microprocesseur F249 : premier programme, première simulation.

Vous disposez sur votre poste des programmes *« Code Composer Studio v4 en version CCS-FREE »* et de « Proteus » avec la clef pour MSP430. *« Code Composer Studio » travaille sous forme de projet organisé dans un dossier.*

Proposition d'organisation des dossiers : je vous propose de créer un dossier « Tp MSP430 », qui vous servira de répertoire de base pour la suite. Pour ma part, j'ai créé ce dossier dans mon disque « E : » :

Les dossiers contiennent :



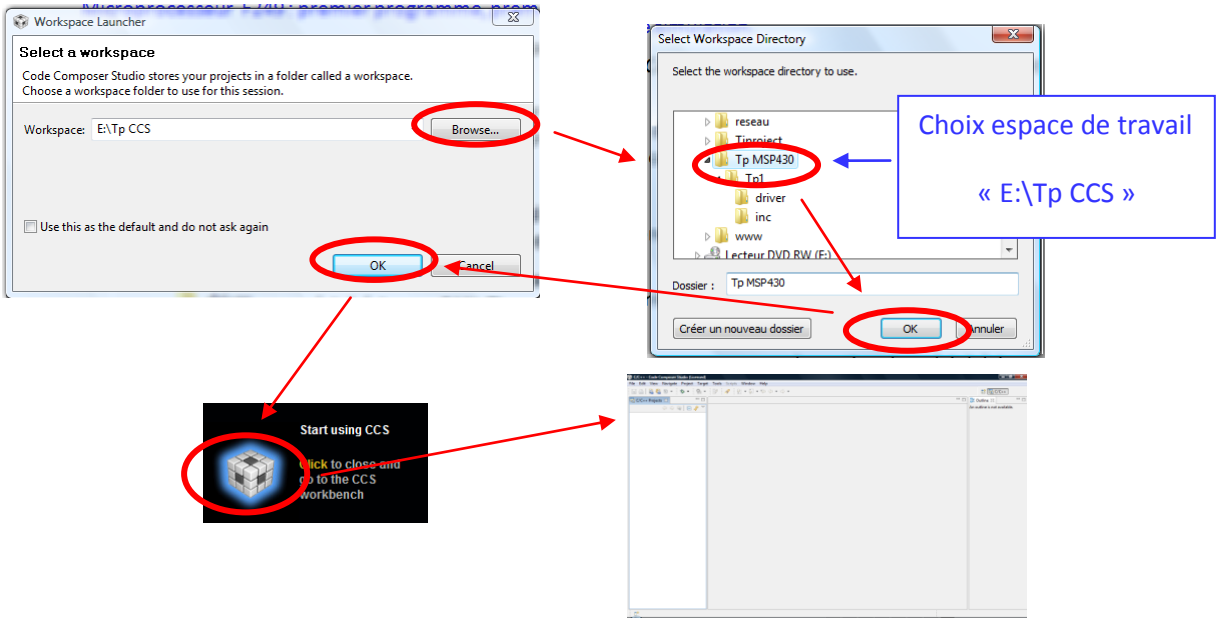
Tp1 : un fichier « main.c » et différents fichiers « .c » contenant les programmes de traitement.

Driver : fichiers « .c » contenant les programmes de gestion des périphériques : CAN, Timer, …

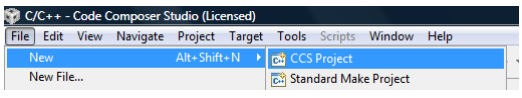
inc : fichiers « .h » contenant les fichiers déclaration

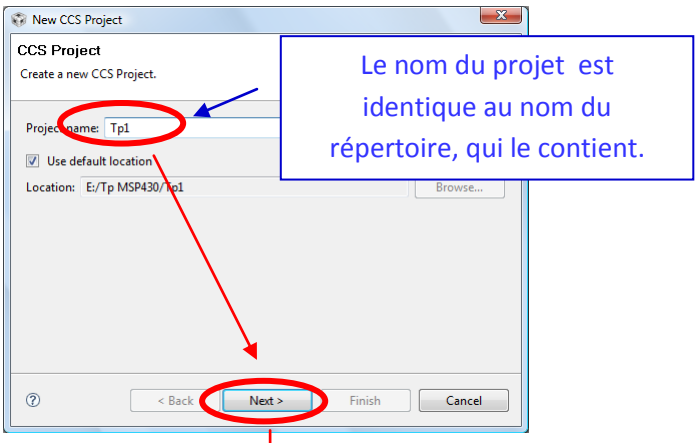
Bien entendu *« Code Composer Studio » possède ses propres fichiers librairie « .h ».*

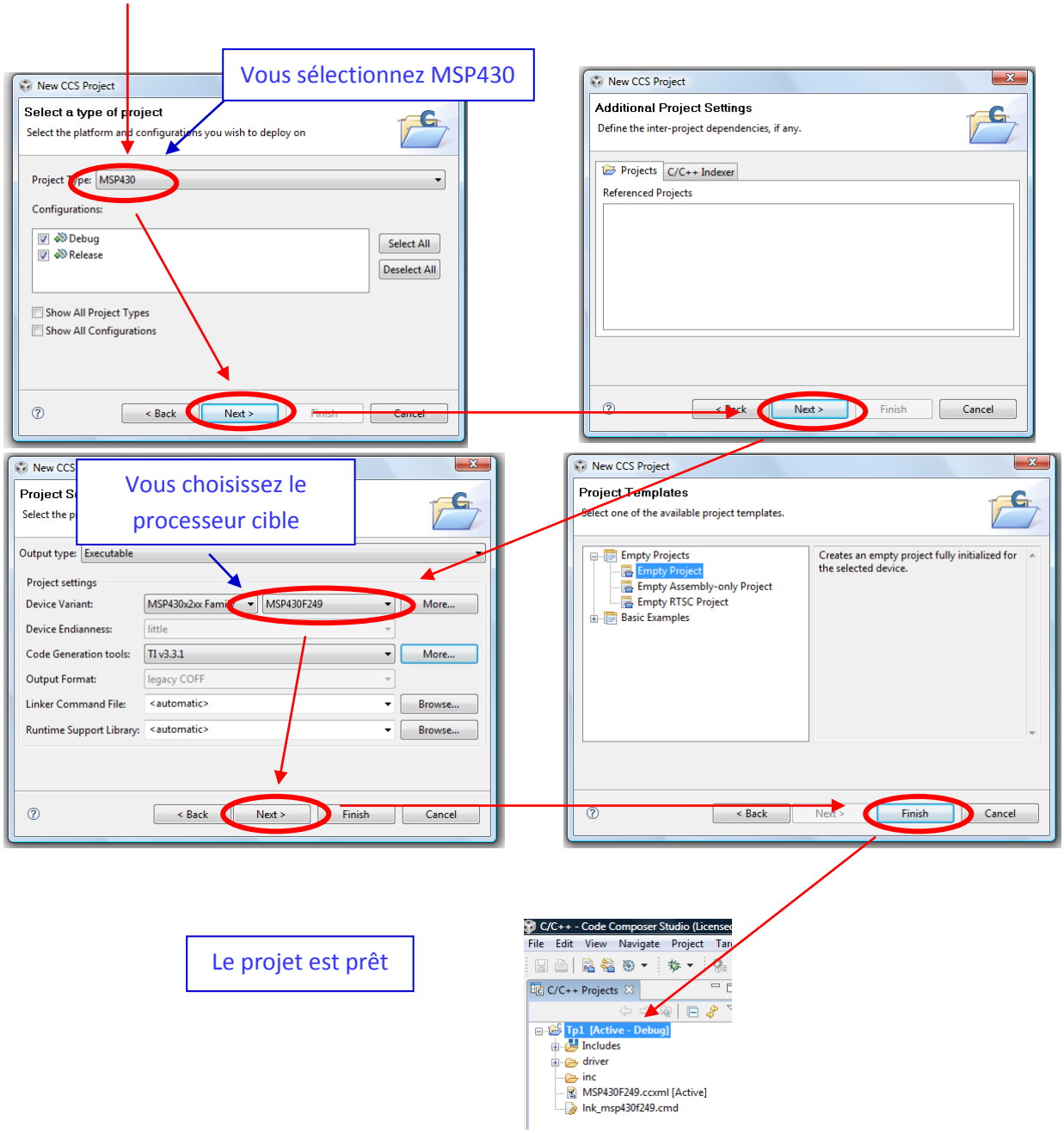
Lancement de *« CCS : Code Composer Studio »* : *vous configurez l'espace de travail au passage.*



Création du projet : CCS est organisé à partir d'un projet.



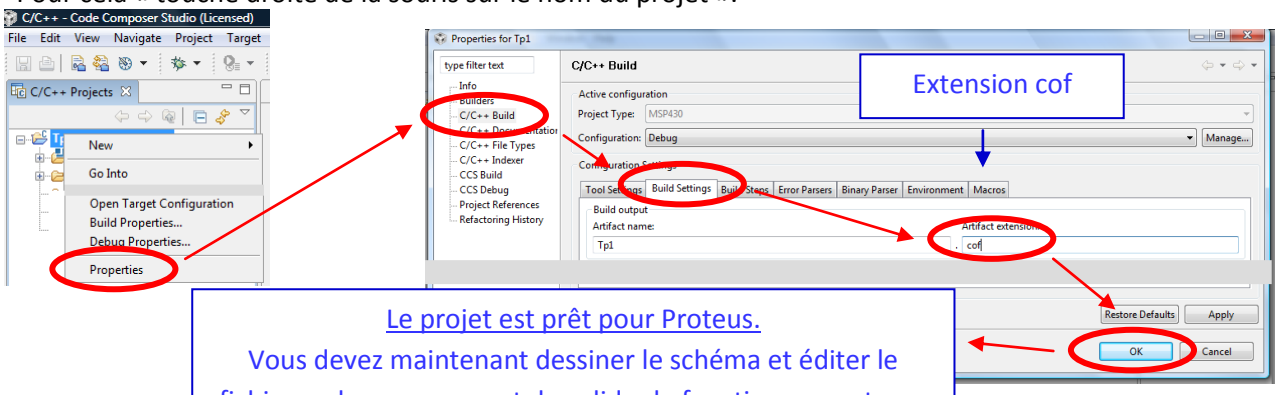




Le projet est prêt

<u>Configuration du projet pour ISIS Proteus</u> : le fichier de sortie de compilation pour être utilisé avec ISIS doit être de format Coff. Son extension doit être cof.

- Pour cela « touche droite de la souris sur le nom du projet ».

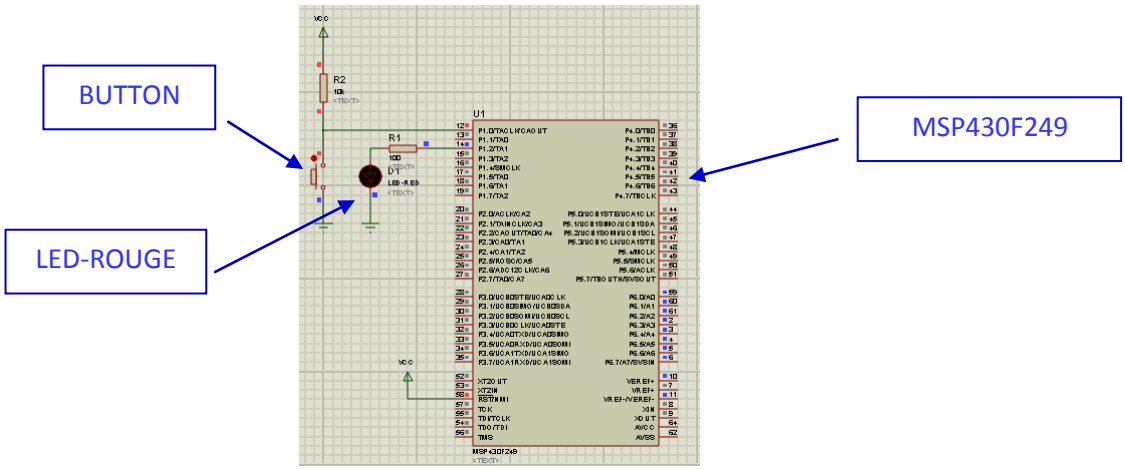


<u>Le projet est prêt pour Proteus.</u>

Vous devez maintenant dessiner le schéma et éditer le fichier en langage c avant de valider le fonctionnement par simulation.

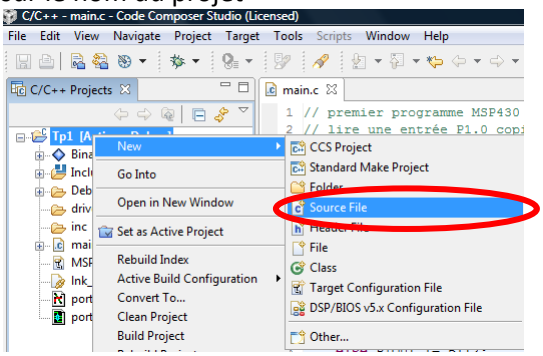
Premier programme – première simulation :

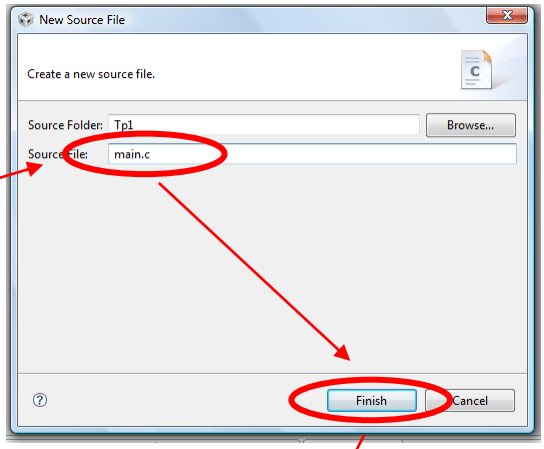
Le schéma du montage est donné ci-dessous avec le nom des composants en bleu (librairie ISIS). Dessinez le schéma et sauvez le dans le dossier Tp1 avec pour nom « port.dsn ».



Le fonctionnement voulu est simple : un appui sur le bouton → la LED éclaire.

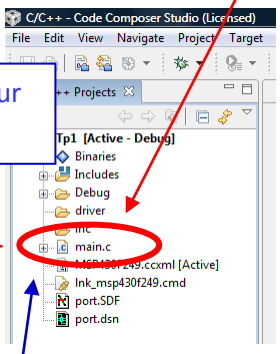
- Création et édition du fichier programme « main.c » : pour cela « touche droite de la souris sur le nom du projet »





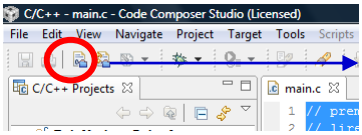

```
#include <msp430f249.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  // Stop watchdog timer
  P1DIR = BIT2;
  // P1.2 en sortie
  P1REN |= BIT0;
  // P1.0 resistance de tirage activée
  while (1)
  {
      // Test P1.0
    if (BIT0 & ~P1IN) P1OUT |= BIT2;
    // P1.2 à 1
    else  P1OUT &= ~BIT2;
    // else P1.2 à 0
  }
}
```

Double clic pour éditer



« Main.c » apparait au niveau de l'explorateur de projet, de même pour le fichier schéma ISIS. Un double clic dessus permet son ouverture pour l'éditer.

- Après avoir édité le programme, vous enregistrez et compilez le projet :

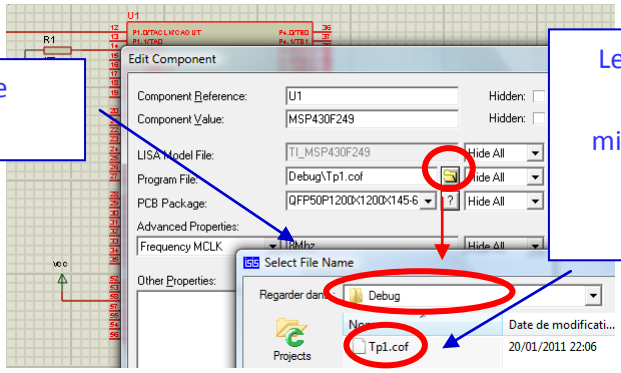


Vérifiez que la compilation s'est déroulée correctement

- Validation du fonctionnement à l'aide d'ISIS par simulation et débogage :

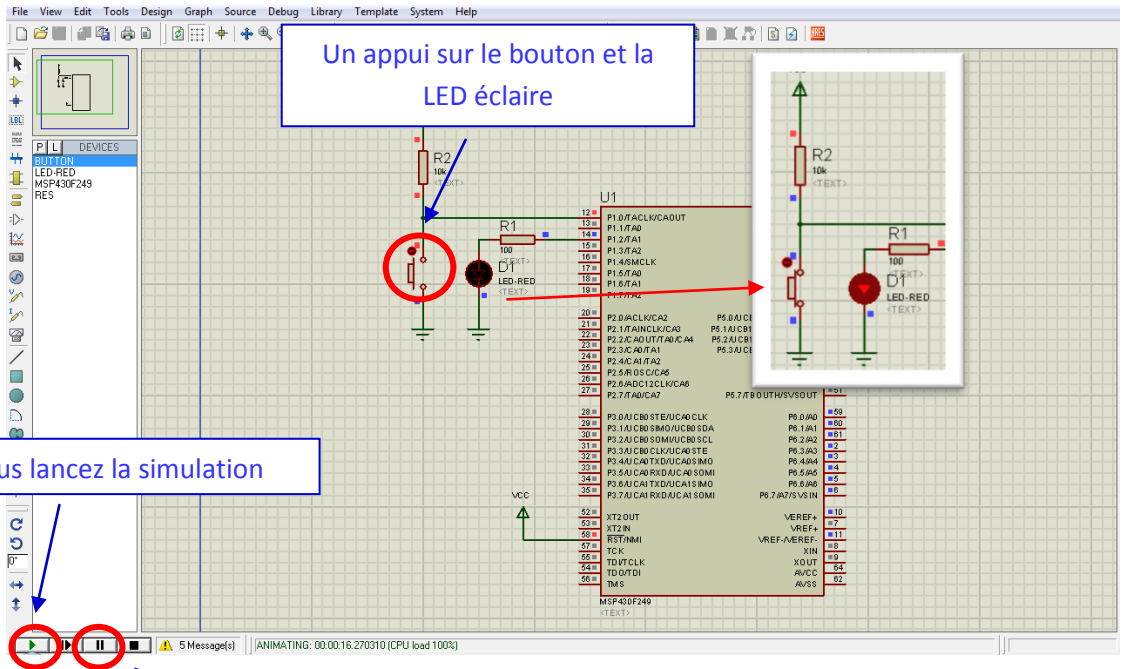
Le schéma a été dessiné précédemment. Vous devez définir les paramètres des composants. Pour le microprocesseur vous définissez la fréquence de l'oscillateur et le fichier programme : Tp1.cof.

Indiquez la fréquence d'horloge à 16MHz



Le fichier du programme à charger dans le microprocesseur Tp1.cof se situe dans le dossier Tp1\Debug\

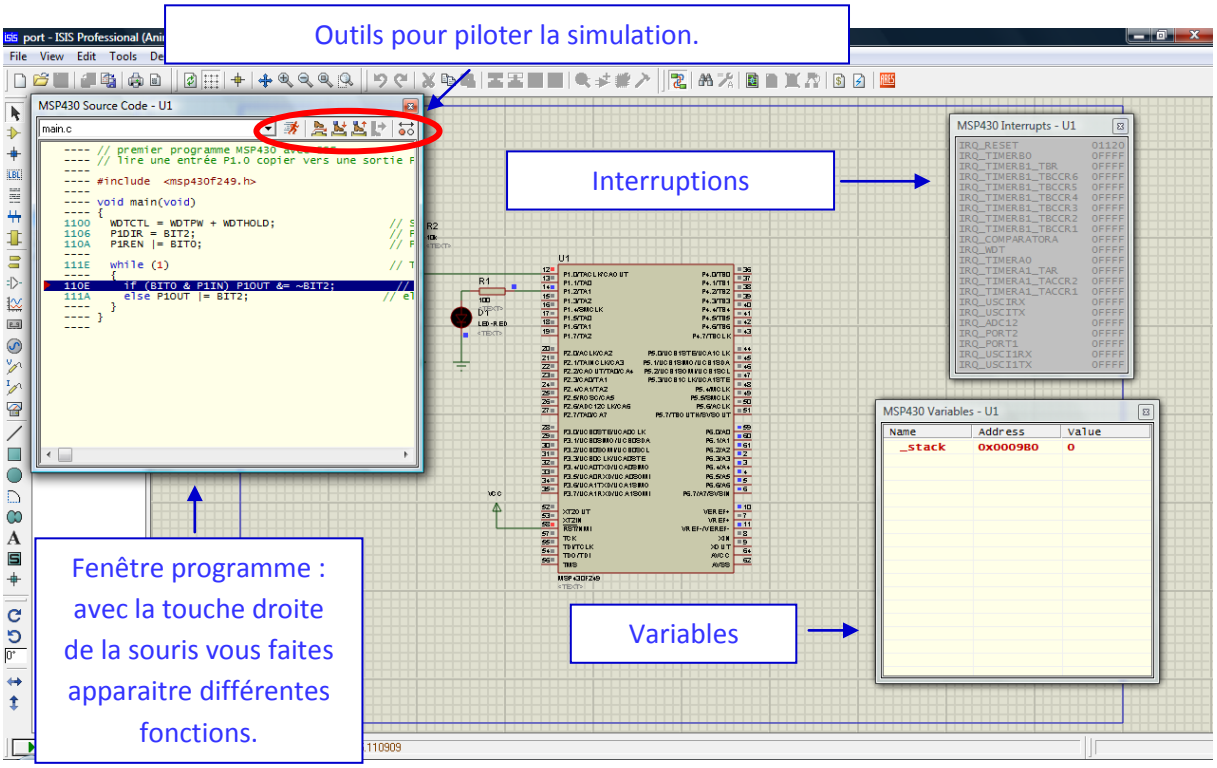
Simulation :



Un appui sur le bouton et la LED éclaire

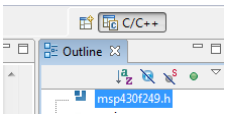
Vous lancez la simulation

La simulation passe en pause. Vous pouvez placer des points d'arrêt, fonctionner en pas à pas…



- Un peu de langage C, compréhension du programme.

  Le fichier « msp430f249.h » contient les déclarations des ressources correspondant au processeur de même nom. Vous ouvrez le fichier en double cliquant dessus à droite de l'écran de ccs.



  - Repérez et expliquez les déclarations de BIT0, BIT1…
  - Repérez la partie déclaration correspondant aux ports 1 et 2, comparez aux ressources internes du processeur et commentez.

```
/************************************************************
* DIGITAL I/O Port1/2 Pull up / Pull down Resistors
************************************************************/
```

  Programme main.c.

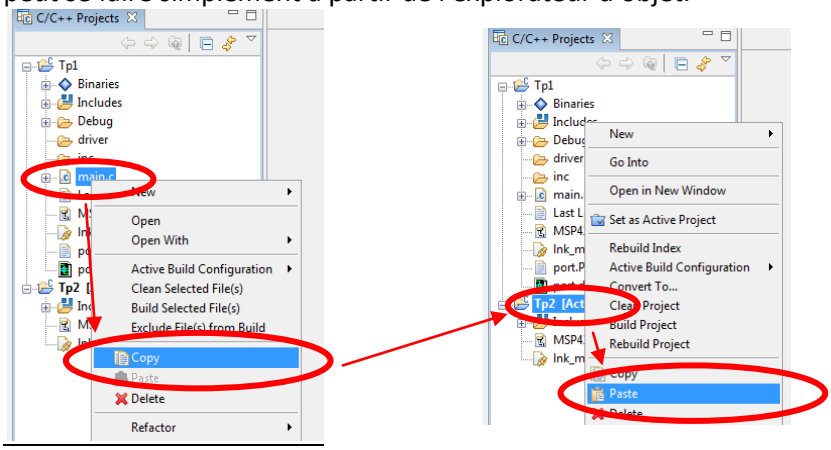
  - Expliquez la ligne « ***#include <msp430f249.h>*** ».
  - Quel rôle joue l'affectation « ***P1DIR = BIT2*** ».
  - Donnez la dénomination des opérateurs suivant :

&	
|	
~	

  - Expliquez l'affectation « ***P1OUT &= ~BIT2*** ».

Quelques exercices pour vérifier que vous avez compris. Pour chacun d'eux vous créerez un nouveau projet et un nouveau fichier ISIS.

Petite astuce : si vous voulez copier un fichier depuis un projet vers un autre projet cela peut se faire simplement à partir de l'explorateur d'objet.



- Tp2 : vous ajoutez une LED jaune en sortie P1.3. Cette LED a un fonctionnement inverse de la LED en P1.2.

  Attention : si vous copiez le schéma ISIS de Tp1, penser à changer dans ISIS le fichier programme du microprocesseur.

- Tp3 : rendre le programme plus lisible.

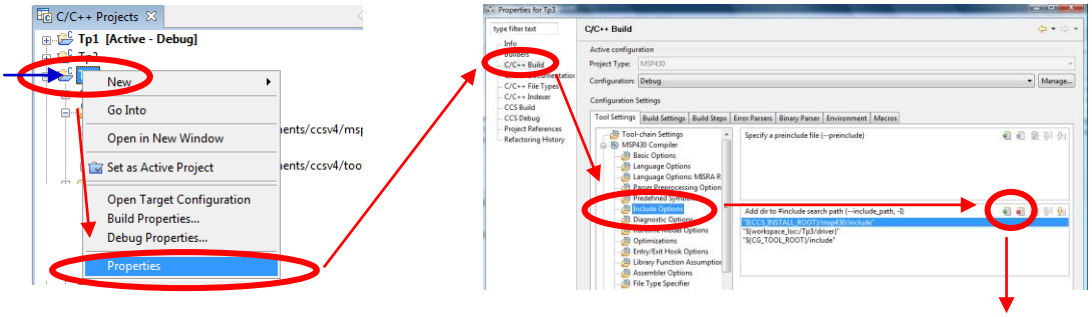
  Pour ce faire nous allons créer un fichier « led.h » que nous allons placer dans le dossier « inc». #define permet de déclarer des macros.

```
#ifndef LED_H_
#define LED_H_

#define led_rouge_on()   (P1OUT |= BIT2)
#define led_rouge_off()  (P1OUT &= ~BIT2)
#define led_jaune_on()   (P1OUT |= BIT3)
#define led_jaune_off()  (P1OUT &= ~BIT3)
#define bouton_on (BIT0 & ~P1IN)

#endif
```

Permet d'intégrer qu'une seule fois « led.h »

Il faut déclarer à CCS, que ce fichier est un fichier à inclure (include).

Touche droite souris sur projet Tp3.



Vous sélectionnez le dossier Tp3\inc

Vous modifiez le fichier « main.c », qui devient :

```
#include  <msp430f249.h>
#include  "led.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;

  P1DIR = BIT2 | BIT3;
  P1REN |= BIT0;

  while (1)
  {
    if (bouton_on)
      {
              led_rouge_on();
              led_jaune_off();
      }
    else
      {
              led_rouge_off();
              led_jaune_on();

      }
  }
}
```

Déclaration de « led.h »

Utilisation des macros

- Tp4 : vous ajoutez un bouton poussoir sur l'entrée P1.1.
  - A la mise sous tension la LED rouge est éteinte.
  - Un appui sur le bouton en P1.0 fait éclairer la LED rouge.
  - Un appui sur le bouton en P1.1 éteint la LED rouge.
  - La LED jaune a un fonctionnement inverse de la LED rouge.