Initiation à la programmation sur MSP 430

CREMMEL Marcel

Lycée Louis Couffignal

STRASBOURG

Initiation à la programmation sur MSP 430

Objectifs :
- ➢ Apprentissage de l'environnement de développement IAR
- ➢ Introduction à la programmation de microcontroleurs en langage C
- ➢ Analyse de petits programmes en "C" et "assembleur"

1. Généralités sur l'élaboration d'un programme

On ne confie pas l'élaboration d'un programme complet à un électronicien. Mais il doit assimiler quelques notions pour être capable de :

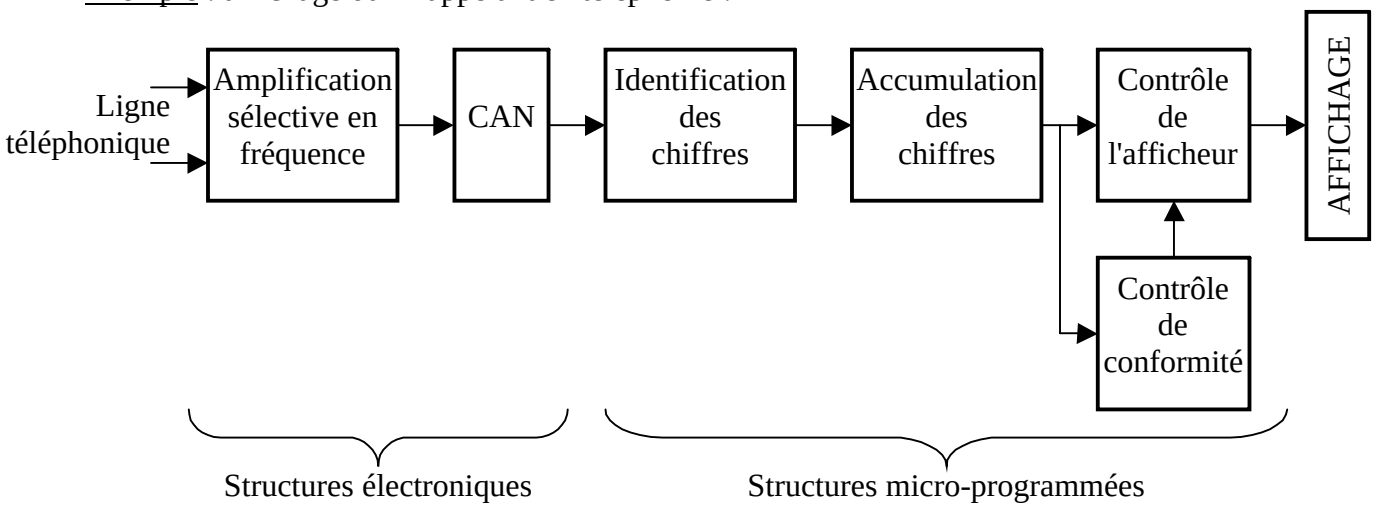
- lire et comprendre des segments de programme liés étroitement aux structures matérielles intégrées ou associées au microcontroleur (entrées-sorties binaires, gestion de capteurs et d'actionneurs, CAN et CNA, claviers, afficheurs, écrans, …),
- modifier ou écrire ce type de segment de programme,
- et surtout de tester et valider ces segments isolément et au sein du programme principal.

L'élaboration d'un programme est divisée en plusieurs étapes :

1. **Analyse des fonctions principales à réaliser :**
   Le travail consiste à décomposer chacune des fonctions principales en fonctions secondaires suffisamment simples. Les descriptions de ces F.S. sont détaillées pour permettre leurs élaborations sans maîtriser l'amont et l'aval et ainsi travailler efficacement en équipes.
   On décide aussi dans cette phase de la répartition des fonctions :
   - réalisées par le "hard",
   - réalisées par le "soft",
   - réalisées de façon mixte (hard + soft)

   Du point de vue du traitement du signal par une fonction, il y a une distinction importante entre les réalisations "électronique" (notamment analogique) et micro-programmées :
   - Dans une structure électronique, toutes les fonctions sont opérationnelles simultanément (dans un amplificateur par exemple).
   - Une structure micro-programmée ne comporte qu'un seul processeur et celui-ci ne peut traiter qu'une seule fonction à la fois. Cela peut poser problème si plusieurs fonctions sont chargées de réagir "instantanément" à des événements extérieurs : il faut imaginer par exemple toutes les fonctions réalisées par l'UC embarquée dans une voiture qui doivent réagir aux ordres de l'automobiliste, gérer le fonctionnement du moteur, s'occuper de l'ABS etc.

   Les schémas fonctionnels des parties micro-programmées sont en fait des diagrammes temporels où les fonctions traitent le signal ou l'information dans un ordre déterminé.
   Exemple : affichage du n° appelant en téléphonie :



La structure électronique fournit un flux numérique continu représentatif du signal analogique prélevé sur la ligne téléphonique.
La fonction d'identification des chiffres réalise une démodulation de ce signal et fournit, s'il existe, le code du chiffre ou de la lettre identifié.
Ce caractère est accumulé dans une mémoire par la fonction suivante pour reconstituer le n° appelant.
Quand le dernier chiffre a été accumulé, le numéro de téléphone est transmis aux 2 fonctions "Contrôle de conformité" et "Contrôle de l'afficheur". Cette dernière activera l'affichage si aucune erreur est détectée.
On constate que l'information est traitée en étapes temporelles à partir de la fonction "identification des chiffres" car le numéro appelant est transmis séquentiellement. Ces fonctions peuvent donc être réalisées par une structure programmée.

Note : la fonction "Identification des chiffres" doit être opérationnelle à tout moment (un nouveau chiffre peut se présenter alors que le précédent est encore traité par la fonction d'accumulation par exemple). Comment, dans ce cas, le processeur peut-il s'occuper des autres fonctions ? En fait, il lui reste du "temps libre" entre 2 échantillons successifs fournis par le CNA. Ce temps est utilisé pour réaliser les autres fonctions. Ce genre de traitement est facilement mis en œuvre par l'utilisation des interruptions.

2. Choix de la structure matérielle du système micro-programmé :

Suivant les applications, 3 choix sont possibles :

- Microcontroleur avec CPU, ROM, RAM et périphériques adaptés à l'application intégrés sur la même puce. Les constructeurs proposent un choix très vaste : on trouve des microcontroleurs à 8 broches pour les applications les plus simples (thermostat par exemple) jusqu'à plus de 100 broches s'il l'application exige un LCD graphique.
  Le choix dépend de l'application, de la vitesse de traitement, de la consommation, du prix et des habitudes de l'entreprise.
- DSP = Digital Signal Processor. On choisit ces processeurs quand une forte puissance de calcul est nécessaire (ex : codeurs/décodeurs MP3 ou DVD). Ils sont coûteux et gourmands en énergie. Peu de microcontroleurs intègrent un DSP, mais ils sont de plus en plus rapides.
- Ordinateur type PC : c'est une plate-forme universelle, capable de réaliser toutes les fonctions imaginables, mais pas toujours adaptée aux applications. Il existe des PC "industriels" robustes, conçus pour fonctionner dans un environnement agressif.

3. Choix du langage et de l'outil de développement :

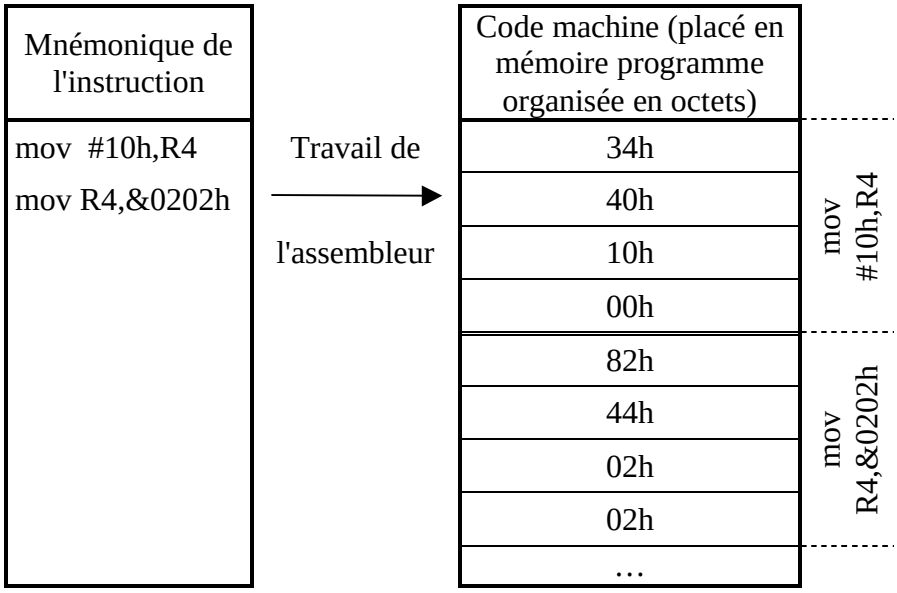
Au final, le µP exécute toujours une suite d'instructions "machines" (ou "assembleur") placé dans la mémoire programme à des adresses consécutives.
A l'époque des pionniers, on affectait presque manuellement chaque case de la mémoire par les instructions du programme ! La mise au point était très fastidieuse au point d'abandonner souvent en cours de route.
Pour limiter la fuite des clients découragés, les fabricants ont rapidement proposé des outils de développement permettant de raccourcir le temps de mise au point et aussi de faciliter l'élaboration des "gros" programmes en équipes (ex : Windows XP sur Pentium).

Ces outils permettent, avec un ordinateur de bureau :

- de **rédiger** le programme dans un éditeur de texte avec toutes ses facilités (copier/coller – fichiers inclus – etc …). On utilise un des langages décrits ci-dessous.
- de **compiler** ce fichier "texte" avec le compilateur pour produire un fichier "binaire". Ce dernier regroupe l'ensemble des instructions machine du programme telles qu'elles seront placées dans la mémoire du microcontroleur. Le transfert dans cette mémoire est réalisé par un programmateur.
- de **simuler** le programme à l'aide du simulateur avant son transfert effectif dans la mémoire du microcontroleur.
  Cette étape est obligatoire, car la première écriture d'un programme comporte toujours des erreurs (bug=cafard), même quand il est très simple.
- de **debugger** le programme implanté dans l'environnement réel ("in circuit") avec le "debugger".

Les langages proposés sont (pour les applications à base de µC) :

– l'**assembleur** (ou langage machine) : on est au plus prêt du µP et on utilise des mnémoniques de ses instructions que l'assembleur va traduire en code "machine".
Exemple sur MSP430 :



On constate qu'une instruction est codée sur plusieurs octets consécutifs dans la mémoire programme. Certains fabricants ont décidé de coder les instructions dans une seule case mémoire, mais la taille est augmentée (ex : les µC PIC de Microchip ont une mémoire programme de largeur 14 bits) pour permettre la mémorisation d'informations complémentaires (n° du registre, adresses source et/ou destination). Ce type de codage permet d'augmenter la vitesse de traitement en réduisant le nombre d'accès mémoire par instruction.

On utilise le langage assembleur quand les temps d'exécution et la taille du programme sont critiques.

– **le C :** c'est le langage le plus utilisé (Windows XP est écrit en C).
L'ingénieur ou le technicien programmeur n'a pas besoin de connaître le jeu d'instruction du µP utilisé pour réaliser un programme performant (rapide et compact). Il utilise des constantes, des variables, des opérateurs, des structures de programmations et d'E/S standardisés. Le compilateur C, généralement fourni par le fabricant du µP, se charge de traduire le programme en langage "assembleur" puis en code machine.
Ainsi, un programme écrit en C pour un certain µP peut être facilement transposé à un autre µP, même d'un autre fabricant.
Les problèmes subsistants sont généralement liés aux structures des périphériques spécifiques à chaque µC. Mais le temps de conversion est beaucoup plus court que s'il fallait réécrire tout le programme avec de nouvelles instructions.

La difficulté pour nous, électroniciens, est la maîtrise des finesses du langage. Mais c'est le travail des informaticiens ! Les compétences d'un électronicien se limitent aux quelques notions nécessaires à l'**analyse et à la modification de fonctions microprogrammées simples** et liées à des structures matérielles (E/S, interruptions, CAN, CNA, liaisons séries, …).

Exemple sur MSP 430 :

Lignes de programme en C	Traduction en assembleur	Code machine
`VAR2 = VAR2+VAR1;`	`add &VAR1,&VAR2`	92h 52h 00h 02h 02h 02h

`VAR_LOC=VAR2 << 2;`	`mov &VAR2,R12`
	`rla R12`
	`rla R12`
		1Ch
		42h
		02h
		02h
		0Ch
		5Ch
		0Ch
		5Ch

Travail du compilateur C → Travail de l'assembleur →

`add &VAR1,&VAR2` ; `mov &VAR2,R12` ; `rla R12` ; `rla R12`

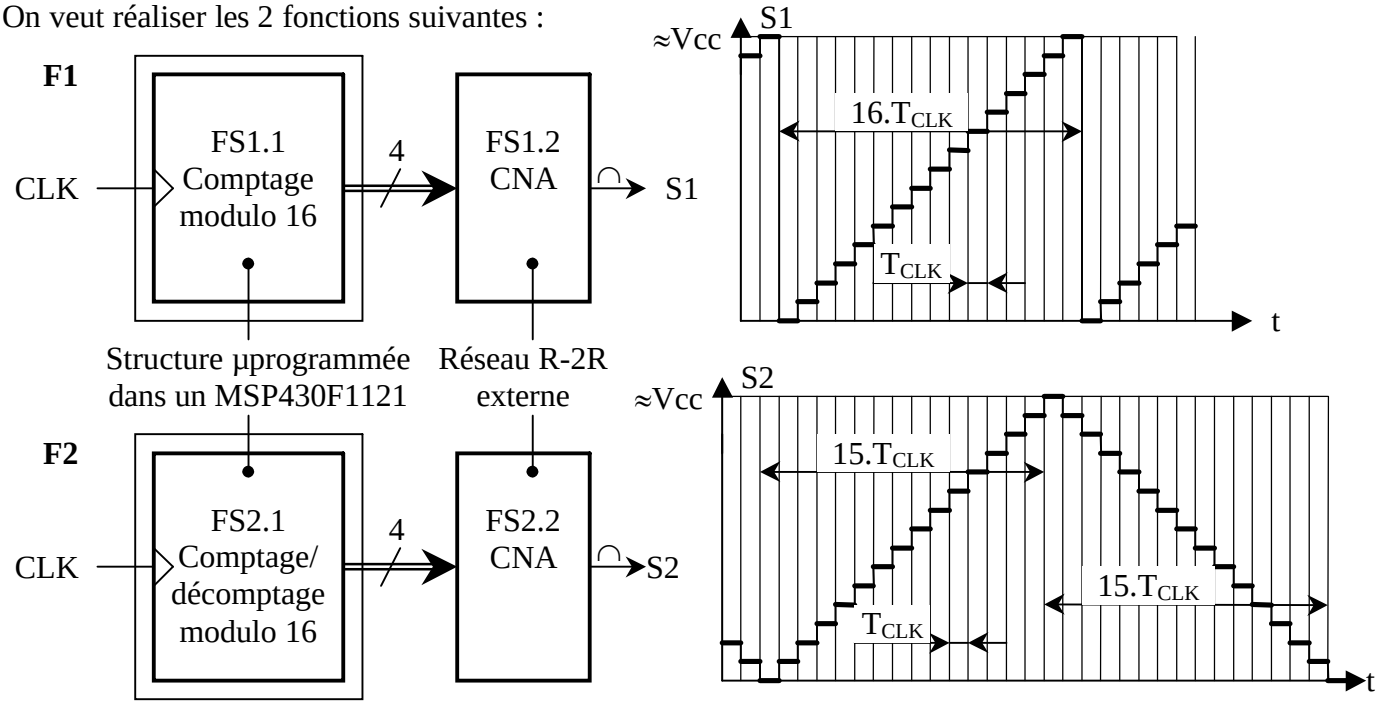
Notes : dans cet exemple :

- VAR1 et VAR2 sont les identificateurs de 2 variables de taille 16 bits placées en RAM aux adresses respectives 0200h et 0202h
- VAR_LOC est une variable 16 bits placée dans le registre R12 du CPU (il en comporte 16, nommés R0 à R15).
  En fait, le programmeur en C n'a pas à se soucier de ces détails techniques (sauf la taille des variables). Le compilateur fait généralement les meilleurs choix d'affectation en mémoire des variables nécessaires au programme (dans l'exemple donné, la variable VAR_LOC est placée dans un registre CPU ce qui réduit les temps d'accès et augmente par conséquent la vitesse de traitement).

– **les autres langages (PASCAL, BASIC, …)** sont très peu utilisés pour développer des applications à base de microcontroleur.

2. Programmes simples en C

On veut réaliser les 2 fonctions suivantes :



Il s'agit donc de générateurs de signaux "dents de scie" et "triangle" dont la fréquence est définie par le signal CLK externe.

2.1 Compteur_soft_C.c

Il s'agit d'une première version de la fonction F1 dépourvue d'entrée CLK externe. Cette fonction n'a pas de sens car elle n'a pas d'entrée ! C'est une introduction à la réalisation de la vraie fonction au §2.3.

2.1.1 Préparation de l'environnement de développement IAR :

→ *Suivre la procédure décrite dans le document ressources "Environnement IAR V4"*

→ *Dupliquer le dossier "`Essai_IAR`" dans le répertoire `D:\MSP430` et le renommer "`Compteurs`"*

→ *Y copier le fichier "`Compteur_soft_C.c`" depuis le serveur*

→ *Suivre alors les indications du document ressources "Environnement IAR V4"*

2.1.2 Analyse du "source"

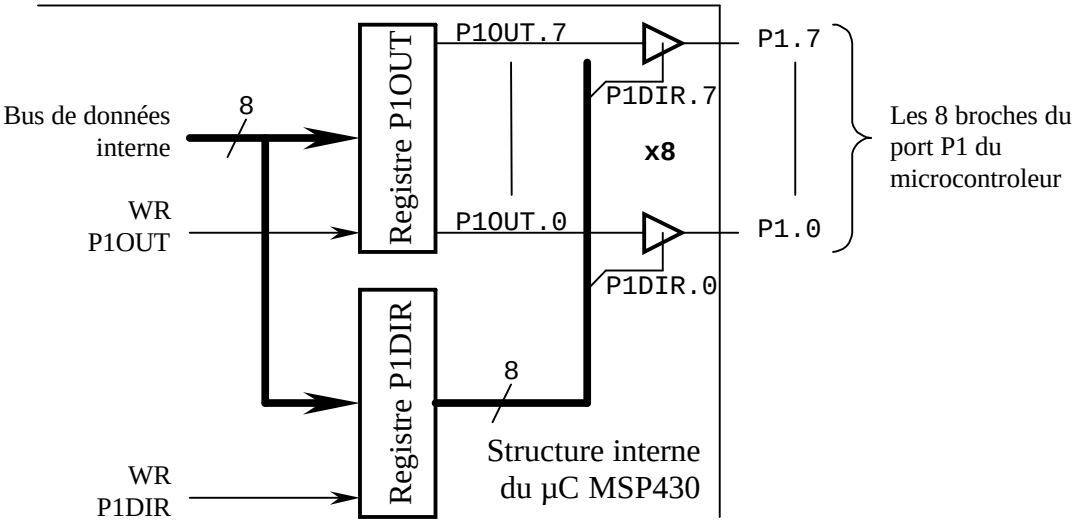
```
#include  <msp430x11x1.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Arrêt du chien de garde
  P1DIR = 0x0F;
  for (;;)
  {
     P1OUT = P1OUT + 1;
  }
}
```

- La première ligne indique au compilateur d'inclure le fichier "`msp430x11x1.h`". Ce fichier texte respecte la syntaxe "C" et définit toutes les constantes liées au µC utilisé : entre autres les adresses des registres de configuration et des ports d'E/S. Par exemple :
  - P1DIR représente le contenu de l'adresse `0022h` (ou `0x0022` avec la syntaxe C)
  - P1OUT représente le contenu de l'adresse `0021h` (ou `0x0021` avec la syntaxe C)
- La structure
  ```
  void main(void)
  {
  }
  ```
  définit le bloc du programme principal (= main). Les significations des mots "void" (vide) seront données plus tard. Le mot "main" ne peut pas être remplacé par un autre.
  Note : les accolades `{}` sont toujours utilisées par paire pour définir le début et la fin d'un bloc de lignes de programme.
- On n'explicite pas pour l'instant la première ligne du programme principal en rapport avec le "chien de garde". Elle inhibe cette fonction du µC.
- `P1DIR = 0x0F;` Il s'agit d'une affectation : la case mémoire P1DIR d'adresse 0022h est affectée avec l'octet 0Fh = 15. Noter le ";" qui est le terminateur de la ligne de programme.
- Structure : `for (;;)`
  ```
  {
  }
  ```
  Le µP exécute en boucle (indéfiniment) les lignes du bloc de programme délimité par les 2 accolades.
  La structure itérative "`for`" sera étudiée plus en détails dans le § suivant.
- Le bloc de la structure "`for`" ne comporte qu'une ligne de programme :
  `P1OUT = P1OUT + 1;`
  D'un point de vue mathématique : elle est fausse ! En fait, il ne s'agit pas d'une égalité, mais d'une affectation : `P1OUT` est affecté avec la valeur actuelle de `P1OUT` augmentée de 1.

En résumé : à droite de "=" : valeurs actuelles
à gauche de "=" : valeur future après calculs

Pour bien comprendre le programme, il faut définir ce que représentent P1DIR et P1OUT dans le µC :



Les signaux "`WR P1OUT`" et "`WR P1DIR`" sont activés quand le µP affecte les registres correspondants.
Les 8 sorties du port 1 du µC sont de type "3 états" :

- états logiques "0" et "1" quand le buffer ▷ est actif (P1DIR.x = "1")
- état haute impédance ("HiZ") quand le buffer est inactif (P1DIR.x = "0")

L'activation de chacun des 8 buffers du port 1 est contrôlée individuellement par les états des 8 sorties du registre P1DIR suivant la table de vérité ci-dessous :

P1DIR.x	P1OUT.x	P1.x
0	X	HiZ
1	0	0
	1	1

→ *Que réalise alors la ligne de programme : "* `P1DIR = 0x0F;` *" ?*

2.1.3 Simulation du programme

→ *Vérifier que l'environnement de développement IAR est en mode "simulation"*
→ *Lancer le "Debugger" : un certain nombre de fenêtres de contrôle s'ouvrent, notamment une réplique du "source" avec la première ligne exécutable du programme repérée par un fond bleu.*
→ *Si ce n'est déjà le cas, ouvrir une fenêtre "register" pour observer les états des registres P1DIR et P1OUT.*
→ *Vérifier le programme par une simulation "pas à pas".*

2.1.4 Version "assembleur" du programme

Le µP exécute en fait une suite d'instructions placées dans la mémoire flash et produites par le compilateur.
La version du programme traduite en "assembleur" peut être visualisée en ouvrant une fenêtre "Disassembly".

→ *Compléter alors le tableau suivant :*

C	Traduction "ASM" des instructions "C"
`P1DIR = 0x0F;`	

`for (;;)`
`{`
`  P1OUT = P1OUT + 1;`
`}`	

→ *Décrire les opérations réalisées par chaque instruction en détails*

→ *Donner ci-dessous le contenu de la mémoire "flash" affectée avec ce programme*

Adresse	Contenu	Instruction

Adresse	Contenu	Instruction

2.1.5 Test du programme "in circuit"

→ *Placer le module MSP430F1121 fourni sur une plaquette de connexions sans soudure.*

→ *Enficher la carte "programmateur" dans le port // du PC et connecter le câble plat au connecteur 14 contacts du module. Veiller à respecter l'orientation.*

→ *Vérifier la position de l'inverseur à glissière.*

→ *Alimenter le module MSP430 avec une source continue 5V entre les broches Vcc et Vss. Le programmateur est alimenté par la même source via le câble.*

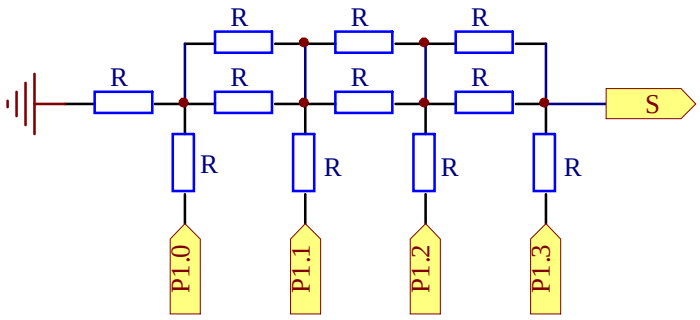
→ *Placer l'environnement de développement en mode "Émulation C" puis lancer le Debugger.*
Le logiciel :

- efface la mémoire flash du µC,
- la programme avec le résultat de la compilation (le code "machine"),
- effectue un "reset" du µC,
- prend le contrôle de celui-ci et le place en mode "arrêt" (halt) sur la première instruction ou ligne de programme "C".

Toutes ces opérations sont réalisées grâce à la module JTAG intégré dans le µC et exploité par le "debugger".

→ *Vérifier le programme par une exécution pas à pas en observant les états des sorties P1.3 à P1.0.*

→ *Câbler le réseau R-2R suivant :*



On prendra R = 39kΩ ou une valeur proche.

Si les résistances sont toutes égales :

$$V_S = \frac{V_{CC}}{2^4} \times \left(P1.3 \times 2^3 + P1.2 \times 2^2 + P1.1 \times 2^1 + P1.0 \times 2^0 \right)$$ avec P1.x = 0 ou 1

→ *Observer l'évolution de Vs en mode "pas à pas" puis "run".*
Mesurer la durée d'un palier. Sont-ils tous de durée égale ?
Mesurer la fréquence de la "dent de scie"

→ *En déduire approximativement l'horloge interne du CPU.*

2.2 Compteur_soft_bis_C.c

On met en œuvre ici une autre structure algorithmique pour réaliser la même fonction :

```
#include  <msp430x11x1.h>

void main(void)
{
  char i;
  WDTCTL = WDTPW + WDTHOLD; // Arrêt du chien de garde
  P1DIR = 0x0F;
  for (;;)
  {
   for (i=0;i!=16;i++)
    {
      P1OUT = i;
    }
  }
}
```

2.2.1 Variable "i"

Le programme utilise une variable identifiée "i" (on peut changer le nom). Elle est déclarée par la ligne:
`char i;` "char" indique au compilateur que la variable a une taille de 8 bits (pour une taille de 16 bits on utilise "int").

Cette variable doit être mémorisée quelque part. Avec le MSP430, on dispose de 2 possibilités :

- en RAM : solution classique. Le compilateur C connaît le plan mémoire du µC cible et place cette variable à une adresse convenable,
- dans un des 12 registres du CPU : on augmente ainsi la vitesse d'exécution des instructions. Mais le nombre de registres est limité !

La position de la déclaration de la variable "i" indique au compilateur le type d'emplacement :

- registre : la déclaration est placée au début d'un bloc de lignes de programme (comme ci-dessus). Si aucun registre n'est disponible, la variable sera placée en RAM.
- RAM : la déclaration est placée à l'extérieur de toute fonction.

→ *Montrer que la variable i est bien placée dans un registre du CPU avec le programme proposé*
→ *Modifier le programme pour que la variable i soit placée en RAM. A quelle adresse est-elle mémorisée ? Est-ce conforme au plan mémoire du MSP430 utilisé ?*
→ *Constater sur la maquette que les vitesses d'exécution des 2 versions sont sensiblement différentes.*

2.2.2 Boucle itérative : for (exp1;exp2;exp3);

Cette structure permet de faire exécuter un certain nombre de fois le bloc du programme délimité par les 2 accolades.

→ *Expliciter le rôle des 3 expressions (exp1, exp2 et exp3) en analysant et simulant la version "assembleur" du programme.*

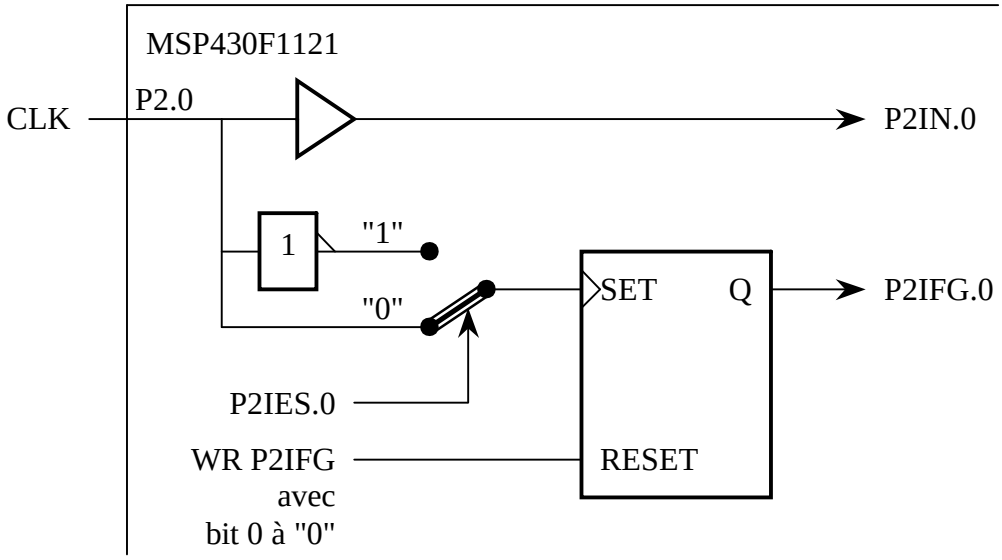
2.2.3 Modification du programme

→ *Modifier le programme pour réaliser la fonction F2 (voir §2)*
→ *Vérifier le fonctionnement sur la maquette et mesurer les caractéristiques du signal S produit.*
→ *Un palier est un peu plus long que les autres. Expliquer pourquoi.*

2.3 Compteur_C.c

Il s'agit de réaliser la fonction F1 avec une horloge CLK externe.
Pour ce faire, on exploite une structure matérielle liée à chaque broche de chaque port d'E/S du µC.

On choisit de façon arbitraire d'utiliser la broche P2.0 comme entrée CLK. La structure logique interne au MSP430 associée à cette broche peut être représentée par le schéma ci-dessous (le registre de sortie a été volontairement omis car la broche sera utilisée en entrée) :



Plusieurs registres du µC sont utilisés pour piloter cette structure :

P2IN : (7 6 5 4 3 2 1 0)
Utilisé en lecture.
Chaque bit prend l'état logique de la broche associée du port 2.

P2IES : (7 6 5 4 3 2 1 0)
IES = Input Edge Select
Utilisé en écriture et lecture.
Chaque bit contrôle la position de l'inverseur (voir schéma) de la broche associée du port 2.
On choisit en fait le flanc actif (montant ou descendant).

P2IFG : (7 6 5 4 3 2 1 0)
Utilisé en lecture et écriture.
Chaque bit de ce registre donne l'état de la bascule (voir schéma) associée à la broche correspondante du port 2.

L'affectation de cette bascule est particulière :

- elle est mise à "1" par un ↑ sur son entrée SET
- elle est mise à "0" par un état actif sur son entrée RESET, soit quand le µP effectue une écriture dans le registre P2IFG **avec le bit concerné à "0"**.

Note : la mise à "1" d'un bit de ce registre peut provoquer une interruption du programme si elle est autorisée. Ce n'est pas le cas dans cette application.

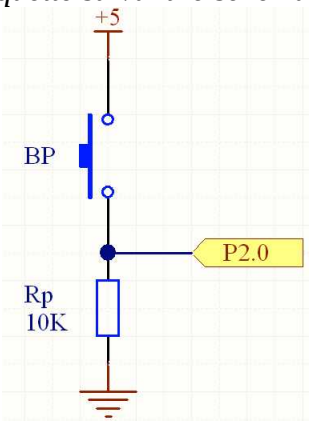
2.3.1 Source du programme :

```
#include  <msp430x11x1.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Arrêt du chien de garde
  P1DIR = 0xFF;
  P2IES = 0x00;
  for (;;)
  {
   if ((P2IFG & 0x01)==0x01)
    {
     P2IFG = 0x00;
     P1OUT = P1OUT + 1;
    }
  }
}
```

→ *Expliciter chaque ligne du programme en rapport avec la fonction à réaliser*

→ *Câbler un bouton poussoir sur la maquette suivant le schéma ci-dessous :*



→ *Tester le programme sur la maquette en mode "pas à pas" :*

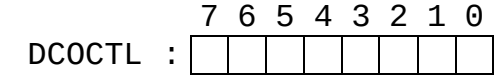
- *constater la mise à "1" de `P2IFG.0` au ↑ sur P2.0,*
- *la prise en compte de ce flanc montant par le programme.*

→ *Tester le programme sur la maquette en mode "run"*

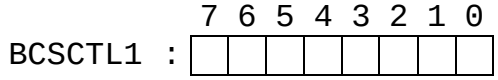
- *constater la réalisation de la fonction en basse fréquence,*
- *mesurer le retard max entre le ↑ sur P2.0 et le changement d'état du port 1. Pourquoi ce retard n'est pas constant ?*
- *évaluer la fréquence max permettant un fonctionnement satisfaisant.*

2.3.2 Amélioration des performances

Les registres DCOCTL et BCSCTL1 du µC contrôlent la fréquence de l'horloge DCO du µP :
Plusieurs registres du µC sont utilisés pour piloter cette structure :



Bits 7 à 5 = `DCO=DCO.2/DCO.1/DCO.0`
DCO = Digital Controled Oscillator



Bits 2 à 0 = `RSEL=RSEL.2/RSEL.1/RSEL.0`
RESEL = Resistor SELect

Dans sa documentation, le constructeur utilise les équivalents décimaux DCO_{10} et $RSEL_{10}$ de valeurs comprises donc entre 0 et 7.
La page 35 de la documentation constructeur du MSP430F1121 donne les fréquences obtenues pour différents couples de valeurs DCO_{10} et $RSEL_{10}$.

→ *Proposer des valeurs pour DCO_{10} et $RSEL_{10}$ permettant de maximaliser les performances du programme.*

→ *Modifier le programme de façon à affecter les bits concernés des registres DCOCTL et BCSCTL1 sans altérer les états des autres bits.*

→ *Mesurer les nouvelles performances.*