# MICROINFORMATIQUE

## NOTE D'APPLICATION 3

## PROGRAMMES EN C POUR LA CARTE IAI MSP430 (PREMIERE PARTIE)

Institut d'
Automatisation
Industrielle

# 1. Documents de référence

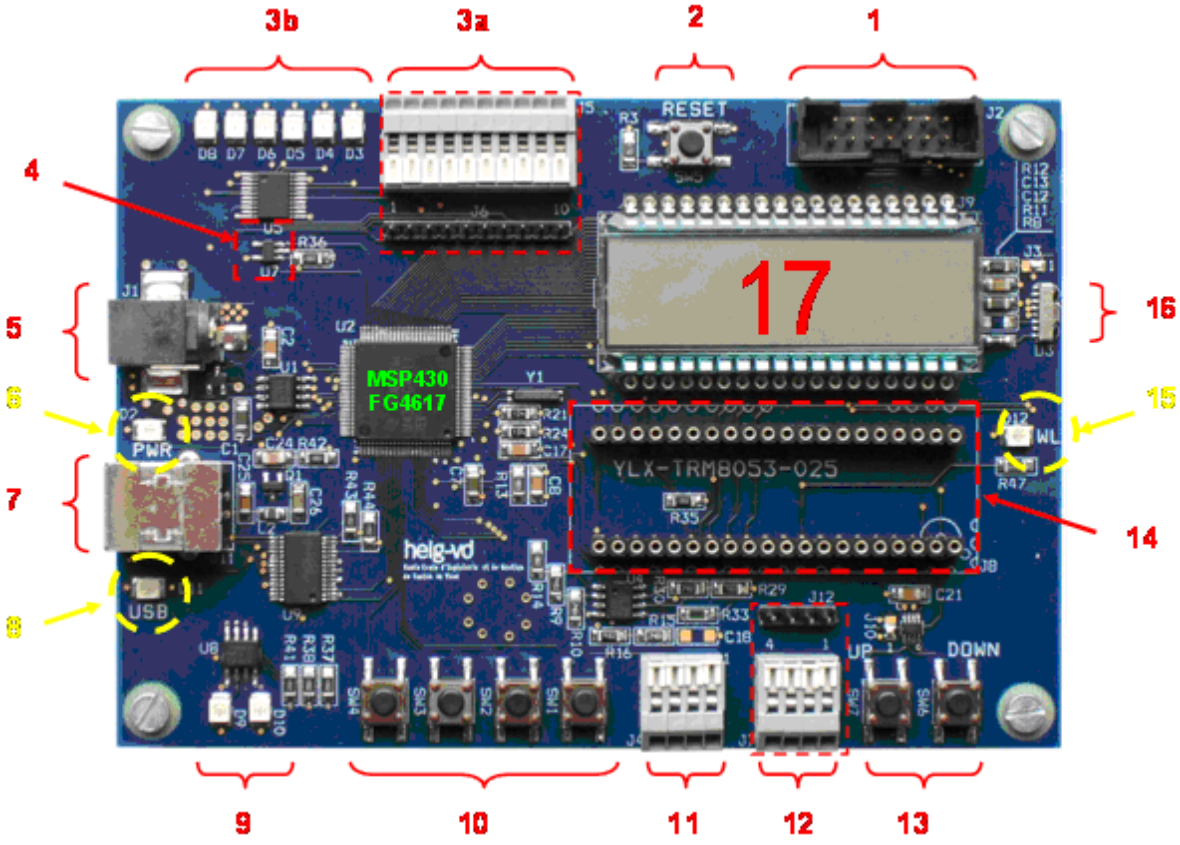
Les documents de références pour cette note d'application sont :

1. ***MSP430x4xx Family.pdf*** : manuel de programmation du MSP430FG4617, accessible sur la page web des supports de cours
2. ***EW430_UserGuide.pdf*** : MSP430 IAR Embedded Workbench® IDE User Guide
3. ***Slides Numération***
4. ***Slides Outils de développement***
5. ***Slides Introduction au TI MSP430, Unité de traitement, Unité Arithmétique et logique, Multiplicateur, Entrées-sorties, Interruptions, Timers***
6. ***Notre technique : Description de la carte IAI à MSP430***
7. ***Note d'application 2 : Premiers programme pour le module eZ430***

# 2. Carte IAI MSP430FG4617

La carte IAI est décrite dans la note séparée (aussi disponible en ligne sur la page des supports de cours http://php.iai.heig-vd.ch/~lzo/pmwiki/pmwiki.php/Microinformatique/SupportsDeCours ), qu'il est demandé de **lire attentivement**..

Les pilotes du FET Debugger et de la puce USB de la carte MSP430 doivent être installés sur le PC.




L'alimentation de la carte peut se faire par le DC plug (No. 5 sur la figure), ou via USB (No. 7).

# 3. Paramétrages

La création d'un projet en mode debugger se fait essentiellement de la même manière qu'en mode simulation.

Il faut veiller à que **dans chaque nouveau projet ouvert** les paramètres suivants soient validés (menu Projet -> Options ) :

- Dans *General Options* :
  - A l'onglet *target* : MSP430FG4617
  - A l'onglet *Library Confiiguration* : CLIB
- Dans *Debugger* :
  - Driver : FET Debugger

Assurez-vous que vous avez bien organisé le travail avec un répertoire **`Carte_IAI`** dans lequel sera savé le fichier définissant le Workspace **Carte_IAI.eww**, ainsi que tous les sous-répertoires des divers projets à réaliser : toujours un séparé pour chaque nouveau projet dans lequel se trouvera le fichier **nom_du_projet.ewp**.

# 4. GPIO – portage d'un programme sur la carte IAI

Pour commencer à se familiariser avec le MSP430FG4617 et la carte IAI on réalisera un programme similaire à celui demandé aux points 10.2 et 9.6 de la note d'application 2 : trois modes d'allumages des LEDs commandés par des boutons.

Le programme désiré aura la **spécification suivante** :

a. Quand on presse le bouton 1, les **6 LED**s s'allument, on le represse et elles s'éteignent.

b. Quand on presse le bouton 2, les **6 LEDs** s'allument et s'éteignent en séquence de chenillard, une après l'autre, après la dernière le cycle recommence avec la première

c. Quand on presse le bouton 3, les **6 LEDs** flashent ensemble régulièrement.

Le passage d'un état à l'autre doit être réalisé par des **interruptions sur le port P1**.

Le canevas peut être facilement réalisé par un copier-coller du code réalisé pour le eZ430. Toutefois plusieurs changements seront nécessaires à cause de la configuration des entrées-sorties sur la carte:

- Les boutons d'entrée sont associés aux ports P1.0 à P1.3
- Les quatre premières LEDs (depuis la droite) sont associées au ports P2.0 à P2.3, mais les deux suivantes au ports P2.6 et P2.7 .

# 5. Gestion de l'affichage

Le MSP430FG4617 a un périphérique interne dédié au contrôle d'affichages LCD (LCD_A controller).

La carte IAI dispose d'un module LCD DE-301 dont le datasheet est disponible sur la page web des supports de cours ( http://php.iai.heig-vd.ch/~lzo/pmwiki/pmwiki.php/Microinformatique/SupportsDeCours ).

En pratique pour la plupart des applications on pourra utiliser une série de fonctions prédéfinies durant le développement de cette carte, qu'on trouve décrites dans un module **LCD.h** . Le correspondant module source **LCD.c** définit en particulier tous les codes pour afficher les caractères ascii. Ces deux modules sont téléchargeables sur la page des supports de cours.

### Module LCD.h

```
extern void LCD_clear(void);
extern void LCD_init(void);
extern void LCD_setCursorPos(unsigned char n);
extern unsigned char LCD_getCursorPos(void);
extern void LCD_print(char* str);
```

### Module LCD.c

```
#include "io430.h"

void LCD_init(void);
void LCD_clear(void);
unsigned char cursorPos;

// ATTENTION, il est possible que les variables globales ne soient pas
// initialisées lors l'utilisation d'un package C par un prg assembleur.
// Les constantes, elles, le sont.

// correspondance caractères-segments pour les 128 premiers caractères ASCII
// car 0-31  et 127 pas imprimables... => enlevés
//° = ar ascii 248
const unsigned short int chr2seg[95] = { \
/*  0 0,0,0,0,0,0,0,0,\*/
/*  8 0,0,0,0,0,0,0,0,\*/
/* 16 0,0,0,0,0,0,0,0,\*/
/* 24 0,0,0,0,0,0,0,0,\*/
/* 32*/ 0,0,0,0,0,0,0,0,\
/* 40*/ 0,0,0,0,0x8000,0x0240,0x8000,0,\
/* 48*/ 0x700E,0x6000,0x324C,0x7248,0x6242,0x524A,0x524E,0x7000,\
/* 56*/ 0x724E,0x724A,0,0,0,0x0248,0,0,\
/* 64*/ 0,0x7246,0x7858,0x100E,0x7818,0x120E,0x1206,0x504E,\
/* 72*/ 0x6246,0x1818,0x6008,0x02A6,0x000E,0x6126,0x6186,0x700E,\
/* 80*/ 0x3246,0x708E,0x32C6,0x524A,0x1810,0x600E,0x0426/*0x6180*/,0x6486,\
/* 88*/ 0x05A0,0x0520,0x1428,0,0,0,0,0,\
/* 96*/ 0,0,0,0,0,0,0,0,\
/*104*/ 0,0,0,0,0,0,0,0,\
/*112*/ 0,0,0,0,0,0,0,0,\
/*120*/ 0,0,0,0,0,0,0,0,\
};

void LCD_init(void)
{
  LCDACTL = LCDFREQ_32 + LCD4MUX + LCDON;

  LCDAPCTL0 = 0xFE; // active tous segments sauf S0-S3
  LCDAPCTL1 = 0x01; // active semgments S32-S35
```

```
  LCDAVCTL0 = LCDCPEN; // enable charge pump pour création des niveaux
  LCDAVCTL1 = VLCD_8; //3.02V

  P5SEL |= (BIT2|BIT3|BIT4);  // initialisation des pins COM1-COM3
  P5DIR |= (BIT2|BIT3|BIT4);
  LCD_clear();
}

//*************************************************************************
void LCD_clear(void)   //efface tous les segments
{
  unsigned char n;
  for (n=0 ; n<20 ; n++)
    *((unsigned char *)(&LCDM1+n)) = 0x0;
  cursorPos = 1;
}

//*************************************************************************
void LCD_setCursorPos(unsigned char n)
{
  if (n>8)
    n=8;
  if(n<1)
    n=1;
  cursorPos = n;
}

//*************************************************************************
unsigned char LCD_getCursorPos(void)
{
  return(cursorPos);
}

//*************************************************************************

void LCD_print(char* str)
{
  unsigned char length, i, regNr;

  // détermine la longueur de la chaîne
  length=0;
  while(str[length] != 0)
    length++;

  // allume segments
  for (i=0 ; i<length ; i++)
    if(i<=8)
    {
      regNr = 2*(cursorPos + i);
      *((unsigned char *)(&LCDM1+regNr)) = chr2seg[str[i]-32] & 0x00FF;
      *((unsigned char *)(&LCDM1+regNr+1)) = (chr2seg[str[i]-32]>>8) & 0x00FF;
    }
}
```

## 5.1 EXERCICES

1. Réaliser un programme qui affiche une chaine de 8 caractères sur le LCD (par exemple votre nom, si nécessaire abrévié).
   Ce programme utilisera par exemple les éléments de code suivants, à compléter.

```
...
#include "LCD.h"
#include "intrinsics.h"
#include "stdio.h"

#define LCD_LENGTH 8

int main( void )
{
  char strDisp[LCD_LENGTH+1];
  const char str[] = "HELLO***";

  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  LCD_init();   // initialise et efface LCD
  sprintf(strDisp, "%s", str);

  LCD_clear();
  LCD_print(strDisp);

  while (1)
  . . .
```

   Il sera aussi nécessaire d'inclure dans le projet IAR le fichier LCD.c (menu Project -> Add Files …)

2. Réaliser ensuite un programme qui affiche sur le LCD le contenu de P1IN en hexadécimal pendant que l'utilisateur pousse un ou plusieurs des quatre boutons (10) associés aux broches P1.0 à P1.3.
   Il sera utile à ce propos de se repasser le format de certaines fonction C (par exemple sprintf).

   Une documentation en ligne avec toutes les fonctions C standard se trouve à l'adresse

   http://www.cplusplus.com/reference/clibrary/

# 6. Horloges et Timers

Le MSP430FG4617 contient un système d'horloges flexible conçut spécialement pour une utilisation avec une batterie. Le système d'horloges met `a disposition de l'utilisateur trois sources différentes en fonction des besoins en consommation et en fréquence :

– ACLK (Auxillary clock) pour les applications qui doivent consommer très peu et qui n'ont pas besoin de très hautes fréquences (32'768 Hz max)

– MCLK (Master clock) qui est utilisée par le CPU, utilisée pour les applications qui ont besoin de hautes fréquences (jusqu'a 8Mhz)

– SMCLK (Sub main clock) utilisable pour les modules périphériques, dont la fréquence est par défaut 1,048576 MHz (= 32 · 32'768 Hz).

Il contient aussi deux timers internes (Timer_A et Timer_B) configurables indépendamment.

## 6.1 TIMERS INTERNES

Le MSP430FG4617 possède deux Timer internes : Timer A et Timer B qui ont presque les mêmes caractéristiques, le B ayant quelques possibilités en plus.

Nous allons voir le principe très simplifié de fonctionnement du Timer A pour comprendre comment le temps est géré par le nœud. Le Timer A contient deux registres TAR (compteur) et TACCR0 (capture/comparaison) et est relié `a une source d'horloge configurable.

Plusieurs modes de comptage sont au choix:

- mode continu (CONTINUOUS) dans lequel le TAR est incrémenté jusqu'a sa valeur maximale 0xFFFF avant de générer une interruption, l
- mode montant (UP) dans lequel le TAR est incrémenté jusqu'a la valeur contenue dans TACCR0 avant de générer une interruption,
- mode alterné (UP/DOWN) dans lequel TAR est incrémenté jusqu'à la valeur de TACCR0 puis décrémenté jusqu'a TACCR0 avant de générer l'interruption.

L'incrémentation ou la décrémentation est déclenchée par des fronts montants sur l'entrée du registre de comparaison en fonction des cycles d'horloge.

Le timer possède d'autres registres TACCR1, TACCR2 (le timer B en contient 6 au total) permettant de générer des interruptions à des intervalles différents. Plusieurs flags sont modifiables pour contrôler le timer, par exemple pour remettre le TAR à zéro ou désactiver les interruptions.

## 6.2 EXEMPLE

L'exemple suivant illustre un simple timer dans un programme qui fait clignoter les six LEDs de P2 à une fréquence de 1 Hz.

```
#include "io430.h"
#include "intrinsics.h"

int main( void )
{
  float freq;
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD; // Stop WDT

  P2DIR |= 0xCF; // P2.0 output
  P2OUT = 0;
  CCTL0 = CCIE;  // CCR0 interrupt enabled

  // fréquence = 32768/CCR0 [Hz]
  freq = 2; // Hz
  CCR0 = 32768/freq - 1;
  TACTL = TASSEL_1 + MC_1;  // ACLK, upmode
  __enable_interrupt();

  while (1);
}

// Timer_A TACCR0 interrupt vector handler
#pragma vector= TIMERA0_VECTOR
__interrupt void int_timerA(void)
{
  P2OUT ^= 0xCF;      // Toggle P2.x
}
```

## 6.3 EXERCICES

1. Réaliser un chronomètre simple avec une résolution de 1/100 de seconde, qui affiche sa mesure sur le LCD en format MM :SS :CC (minutes : secondes : centièmes), par exemple 03:07:99 .

   La mesure est arrêtée par interruption en pressant le bouton P1.0, et elle repart si on represse le même bouton. Le départ du chrono est mis à zéro et enclenché automatiquement par le bouton de Reset.

   Ce programme aura donc deux fonctions d'interruption :

   - Une sur le port P1, qui permet par le registre CCTL0 de activer et désactiver l'interruption du timer.
   - Une sur le timer A, qui incrémente les variables des centièmes, respectivement seconde et minutes.

   La boucle while(1) du main se limite à-afficher les valeurs sur le LCD.

   Ce programme est aussi présenté dans les slides du chapitre « Timers » du cours.

2. Réaliser une horloge digitale, qui affiche l'heure sur le LCD en format HH :MM :SS (heures : minutes : secondes) sur 24 heures. Des interruptions sont aussi programmées sur les boutons P1.0 à P1.3 pour permettre dans l'ordre :

   P1.0 Passage au mode réglage de l'heure, minute, seconde, retour au mode horloge.

   P1.1 Réglage de l'heure par impulsions

   P1.2 Réglage de la minute par impulsions

   P1.3 Réglage de la seconde par impulsions

# 7. Interface avec une console USB

La carte MSP430FG4617 comprend un composant spécifique qui permet de communiquer avec un PC par USB. Le datasheet est disponible sur la page web des supports de cours :
( http://php.iai.heig-vd.ch/~lzo/pmwiki/pmwiki.php/Microinformatique/SupportsDeCours )

La configuration des ports d'entrées-sorties est un bon exemple d'interface parallèle du MSP430 avec 8 lignes parallèles de data + signaux de contrôle.

La création d'une console USB sur une fenêtre terminal tournant sur un PC servira à la fois à mieux comprendre le fonctionnement des GPIO du MSP430 FG4617 et fournira ensuite un outil pratique pour le développement de programmes ultérieurs.

On peut par exemple très facilement utiliser l'hyperterminal de Windows (voir l'annexe à la fin de cette notre d'application) pour afficher des résultats.

- Chip utilisé : FTDI FT245RL (www.ftdichip.com).
- Communication parallèle avec le uC (8 lignes parallèles de data + signaux de contrôle).
- Se référer au datasheet et au schémas inclus pour les détails.

La LED « USB » est allumée lorsque la carte est correctement reliée à un PC, que celui-ci l'a reconnue et a chargé le driver adéquat.

En pratique pour la plupart des applications on pourra utiliser une série de fonctions prédéfinies durant le développement de cette carte, qu'on trouve déclarée dans un module USB.h et explicitées dans le module USB.c . Ces modules sont téléchargeables depuis la page de *supports de cours*.

### Module USB.h

```
// BUT:  fonctions lecture/écriture par USB
//*****************************************************************************

// renvoie vrai si l'USB est connecté
extern unsigned char USB_isConnected(void);

// renvoie vrai si on peut lire un byte sur l'USB
extern unsigned char USB_readyToRead(void);

// lecture d'un byte
extern unsigned char USB_read(void);

// renvoie vrai si on peut écrire un byte à destination du PC
extern unsigned char USB_readyToWrite(void);

// écriture d'un byte à vers le PC
extern void USB_write(unsigned char n);

// initialise signaux de controle pour communication avec FTDI
// (RD# sur P5.0, WR sur P5.1, TXE# sur P5.5, RXF# sur P5.6, PWREN sur P5.7)
extern void USB_init(void);

// lecture d'une chaine de caractères de la console USB
// se référer à la fonction C gets
extern char *usb_gets (char *a);

// affichage d'une chaine de caractères sur la console USB
// se référer à la fonction C puts
extern int usb_puts (char *a);
```

## Module USB.c

```
//*****************************************************************************
// SCD@iai.heig-vd 04.12.07
// USB.c
//
// cible : MSP430FG4617149 sur carte de laboratoire uC MCN
// compilé sous : IAR embedded workbench v3.42A kickstart
//
// BUT: fonctions lecture/écriture par USB
//*****************************************************************************

#include "io430.h"

// ********************************************************************
// renvoie vrai si l'USB est connecté
unsigned char USB_isConnected(void)
{ return (!(P5IN & BIT7));
}

// renvoie vrai si on peut lire un byte sur l'USB
unsigned char USB_readyToRead(void)
{ return (!(P5IN & BIT6));
}

// ********************************************************************
// lecture d'un byte
unsigned char USB_read(void)
{ unsigned char a, b;

  // met IOs en entrées
  P1DIR &= ~(BIT4|BIT5|BIT6|BIT7);
  P3DIR &= ~(BIT4|BIT5|BIT6|BIT7);

  P5OUT &= ~BIT0; //baisse RD#
  a = P1IN;       //lecture...
  b = P3IN;
  P5OUT |= BIT0;  //relève RD#
  return((b&0xF0) | ((a>>4)&0x0F));
}

// ********************************************************************
// renvoie vrai si on peut écrire un byte à destination du PC
unsigned char USB_readyToWrite(void)
{ return (!(P5IN & BIT5));
}

// ********************************************************************
//écriture d'un byte à vers le PC
void USB_write(unsigned char n)
{ unsigned char a;

  P5OUT |= BIT1; //lève WR

  // met IOs en sorties
  P1DIR |= (BIT4|BIT5|BIT6|BIT7);
  P3DIR |= (BIT4|BIT5|BIT6|BIT7);

  // écriture
  a = P1OUT & 0x0F;     // écriture 4 LSBs sur P1.4-7
  P1OUT = a | (n<<4);
  a = P3OUT & 0x0F;     // écriture 4 MSBs sur P3.4-7
  P3OUT = a | (n&0xF0);

  P5OUT &= ~BIT1; // rebaisse WR

  //met IOs en entrées
  P1DIR &= ~(BIT4|BIT5|BIT6|BIT7);
  P3DIR &= ~(BIT4|BIT5|BIT6|BIT7);
}
```

```
// initialise signaux de controle pour communication avec FTDI
// (RD# sur P5.0, WR sur P5.1, TXE# sur P5.5, RXF# sur P5.6, PWREN sur P5.7)
void USB_init(void)
{
  //fixe niveau des signaux
  P5OUT |=  BIT0; //met signal RD# à '1'
  P5OUT &= ~BIT1; //met signal WR à '0'

  //configure IOs
  P5SEL &= ~(BIT0|BIT1|BIT5|BIT6|BIT7);
  P5DIR |= (BIT0|BIT1);
  P5DIR &= ~(BIT5|BIT6|BIT7);
  P1SEL &= ~(BIT4|BIT5|BIT6|BIT7);
  P3SEL &= ~(BIT4|BIT5|BIT6|BIT7);

  //vide buffer de réception
  while (USB_readyToRead())
    USB_read();
}

// ************************************************************
// équivalente à la function C standard puts

int usb_puts (char *a)
{ int j,le;
  le = strlen(a);
  for (j=0 ; j<strlen(a) ; j++)
     USB_write(a[j]);
  USB_write(13);
  return (le);
}

// ************************************************************
// équivalente à la function C standard gets

char *usb_gets (char *a)
{ unsigned char c;
  int i = 0;

  if (USB_readyToRead())
  {
    do
    { if (USB_readyToRead())
      { c = USB_read();
        a[i] = c;
        i++;
      }
    } while (c != 13);
    USB_write(13);
    a[i-1] = 0;
    return a;
  }
  else return 0;
}
```

## 7.1 ETUDE DES FONCTIONS

Analyser la source du module USB.c. Observer que l'interface est réalisée par les ports P1, P3 et P5. Comprendre en particulier les instructions utilisant les divers registres PxDIR et PxSEL.

Comprendre le fonctionnement des diverses fonctions du module USB, en particulier des fonctions qui permettent d'échanger des caractères et des chaines entières. Noter que ces dernières sont formulées sur le modèle des fonctions standard C *gets*() et *puts*().

## 7.2 EXERCICE DE BASE

Charger le driver USB de la carte, disponible sur la page
http://php.iai.heig-vd.ch/~lzo/pmwiki/pmwiki.php/Microinformatique/SupportsDeCours .

Ouvrir un programme de terminal standard (comme *Hyperterminal*) et le connecter au port USB allant à la carte MSP430 (pas à celui vers le FET-Debugger).

Réaliser ensuite sur le MSP430FG4617 une console USB avec un *prompt* (en français, interface en ligne de commande), par exemple "> " .qui est le code ASCII 62 .

Wikipedia :

*Une interface en ligne de commande est une interface homme-machine dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode texte :*

- *L'utilisateur tape du texte au clavier pour demander à l'ordinateur d'effectuer diverses opérations.*
- *L'ordinateur affiche du texte correspondant au résultat de l'exécution des commandes demandées ou à des questions qu'un logiciel pose à l'utilisateur.*

*Une interface en ligne de commandes peut servir aussi bien pour lancer l'exécution de divers logiciels au moyen d'un interpréteur de commandes, que pour les dialogues avec l'utilisateur de ces logiciels.*

On peut par exemple utiliser le canevas suivant. Ne pas oublier d'inclure en entête du programme les *includes* USB.h et LCD.h et dans le projet les références aux USB.c et, le cas échéant, LCD.c .

```
int main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  if (USB_isConnected() && USB_readyToWrite())
  { usb_puts ("console OK");
    USB_write(62);  USB_write(32);
  }

  while(1)
  {
    if (USB_isConnected())

    { P2OUT |= 0x01;  // on allume une LED pour indiquer que l'USB est active

      while (USB_readyToRead() && USB_readyToWrite())

      {  usb_gets (com);

         if (!(strcmp("xx", com)))  // commande xx
         { // programmez ici les instructions pour lire un état de la carte ou,
           // par exemple actionner une LED
```

```
            usb_puts ("action xx");
          }
          else if (!(strcmp("yy", com)))  // commande yy
          { // programmez ici une autre instruction

             usb_puts ("action yy");
          }

          else
          { LCD_print(com);
            usb_puts (com);
          }

          USB_write(62);  USB_write(32);  // le prompt > plus un espace
        }
      }

      else

        P2OUT=0;
    }
  }
```

## 7.3 EXERCICES ULTÉRIEURS

1. Réaliser un programme qui reprend toutes les spécifications de l'exercice au chapitre 4 (page 3) et qui ajoute une console USB pour commander et lire depuis le PC l'état des LEDs associées aux broches P2.0 à P2.3. Créer pour cela des commandes ad hoc sur une console qui permettent de modifier et afficher sur le terminal l'état des LEDs. La fonction `strncmp(…)` peut, par exemple, être utilement utilisée pour « parser » la chaine `com` obtenue par la fonction `usb_gets` (référence au listing précédent).

2. Une fois bien familiarisés avec cette technique on peut ajouter une console USB à des programmes plus complexes, par exemple au programme de l'horloge **hh-mm-ss**, avec des commandes en ligne qui permettent de lire l'heure et d'effectuer les divers réglages de l'horloge.

# *8. Rapport*

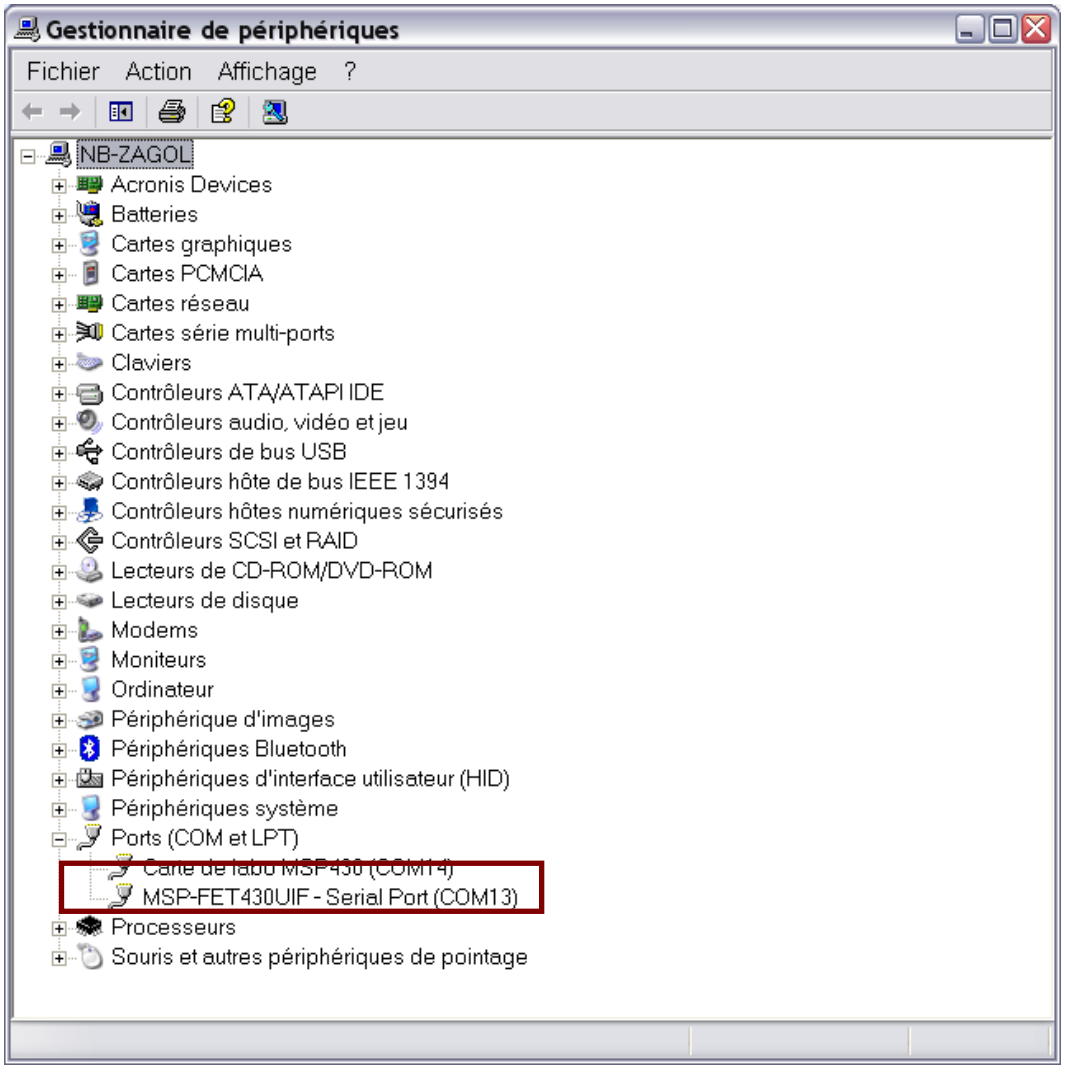
Le rapport sera un simple log-book et enregistrement des divers programmes testés.

Il inclura pour chaque tache effectuée

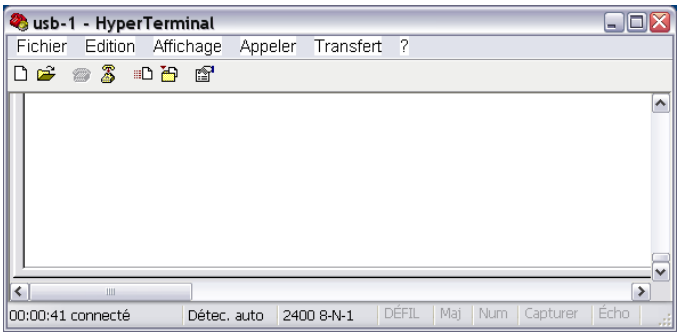
- Titre de la tache
- Copier-coller des *parties significatives* du listing produit
- Remarques, conclusions et suggestions si applicable.

# 9. ANNEXE – Configuration de Hyperterminal sur le PC

- Un câble USB doit connecter la carte au PC (en plus de celui vers le FET-Debugger).
- Déterminer le port COM correspondant pour votre PC à la **carte de labo MSP430**, (dans l'exemple ci-bas le port COM13).




- Lancer Hyperterminal et le relier au port COMx de la carte MSP430.



- Configurer les paramètres ASCII (menu Fichier, onglet Paramètres, bouton Configuration ASCII) comme montré ci-bas.