TP microcontrôleur 16 bits : MSP430

J.-M Friedt, 5 avril 2015

Objectif de ce TP :
- présentation de GNU/Linux et de commandes unix
- faire clignoter des diodes, exemples en C
- communication RS232, communication de données en ASCII
- timer, processus périodiques, interruptions
- conversion numérique-analogique de données, échantillonnage
- communication SPI, accès à la carte SD

1 GNU/Linux

L'arborescence de GNU/Linux et comment s'y retrouver : dans l'ordre de priorité :
- `/` est la racine de l'arborescence. Le séparateur entre répertoire sera toujours le `/`.
- `/home` contient les répertoires des utilisateurs. Unix, et Linux en particulier, impose un confinement strict des fichiers de chaque utilisateur dans son répertoire. Seul l'administrateur (`root`) peut accéder aux fichiers du système en écriture : **ne jamais travailler en tant qu'administrateur**.
- `/usr` contient les programmes locaux à un ordinateur : applications, entêtes de compilation (`/usr/include`), sources du système (`/usr/src`)
- `/lib` contient les bibliothèques du système d'exploitation
- `/proc` contient des pseudo fichiers fournissant des informations sur la configuration et l'état du matériel et du système d'exploitation. Par exemple, `cat /proc/cpuinfo`.
- `/etc` contient les fichiers de configuration du système ou la configuration par défaut des logiciels utilisés par les utilisateurs en l'absence de fichier de configuration locale (par exemple `cat /etc/profile`).
- `/bin` contient les outils du système accessibles aux utilisateurs et `/sbin` contient les outils du systèmes qu'*a priori* seul l'administrateur a à utiliser.

Les commandes de base :
- `ls` : list, afficher le contenu d'un répertoire
- `mv` : move, déplacer ou renommer un fichier. `mv source dest`
- `cd` : change directory, se déplacer dans l'arborescence. Pour remonter dans l'arborescence : `cd ..`
- `rm` : remove, effacer un fichier
- `cp` : copy, copier un fichier. `cp source dest` Afin de copier un répertoire et son contenu, `cp -r repertoire destination`.
- `cat` : afficher le contenu d'un fichier. `cat fichier`
- `man`, manuel, est la commande la plus importante sous Unix, qui donne la liste des options et le mode d'emploi de chaque commande.

Afin de remonter dans l'historique, utiliser SHIFT+PgUp. Toute commande est lancée en tâche de fond lorsqu'elle est terminée par `&`.

L'interface graphique est une sur-couche qui ralentit le système d'exploitation et occupe des ressources, mais peut parfois faciliter la vie en proposant de travailler dans plusieurs fenêtres simultanément. Elle se lance au moyen de `startx`.

Parmi les outils mis à disposition, un éditeur de texte (`scite`), un clone opensource de matlab nommé `octave`, un outil pour tracer des courbes (`gnuplot`), affichage des fichiers au format PDF par `xpdf`, un gestionnaire de fichiers (`pcmanfm`).

La liste des modules noyaux (équivalent des pilotes, chargés de réaliser l'interface logicielle entre le noyau Linux et l'espace utilisateur) s'obtient par `/sbin/lsmod`.

2 Premier pas avec `gcc`

Dans un éditeur de texte (`scite`), taper le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>

int main() {printf("hello world %d\n",getpid());}
```

Nous n'utiliserons pas d'environnement intégré de développement (IDE) : un éditeur de texte sert à entrer son programme, et une fenêtre de commande (`xterm` ou Terminal) servira à la compilation.

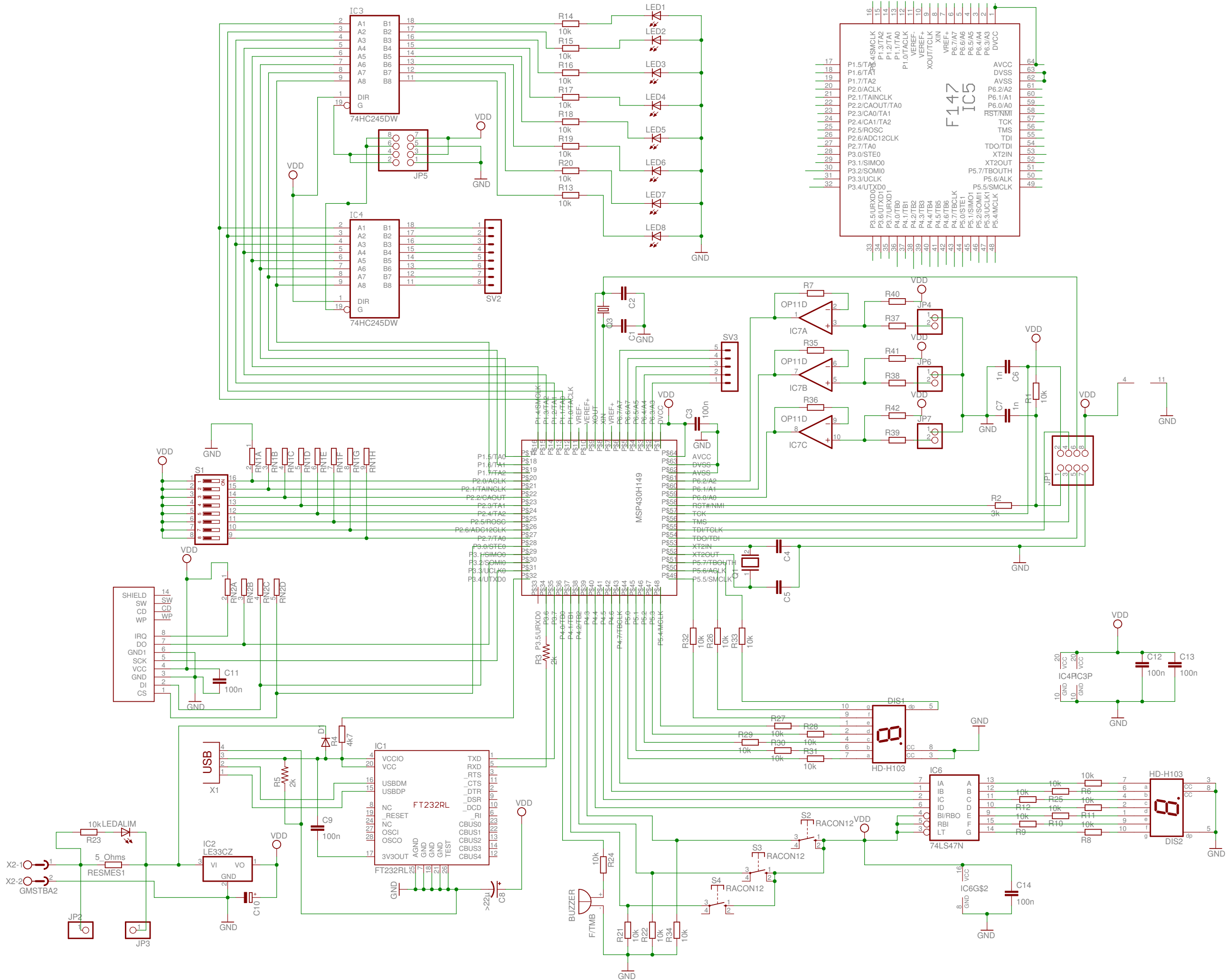
Une fois le programme tapé, sauver dans un fichier `exemple1.c` et compiler dans un terminal en entrant la ligne de commande : `gcc -o exemple1 exemple1.c` (compiler le programme C `exemple1.c` résultant en l'exécutable `exemple1`).

Exercice : exécuter le programme par `./exemple1`.

3 Premier pas sur MSP430

L'exemple précédent était une *compilation* de programme : l'architecture sur laquelle le programme sera exécuté est la même que celle sur laquelle le programme est compilé (ici, Intel x86).

Nous allons désormais compiler des programmes à destination du microcontrôleur : il s'agit de *cross-compilation*.



Le MSP430 [1] se programme en C au moyen du même compilateur que vu précédemment (`gcc`), mais configuré pour générer du code à destination de la nouvelle architecture cible (MSP430), différente de l'architecture hôte (Intel 32 bits). Il s'agit donc de *cross*compilation.

XOR	0	1
0	0	1
1	1	0

```
#include  <msp430x14x.h>
#include  <string.h>
#include  <legacymsp430.h> // #include  <signal.h>
#include  <msp430.h> // #include <io.h>
#include <stdio.h>

void init_PORT(void){
  P1SEL = 0x00;
  P1DIR = 0xFF;
  P1OUT = 0x00;
}

int main (void)
{
 unsigned int led1=0x01,sens=0,k;
 WDTCTL = WDTPW + WDTHOLD;     // Stop WDT
```

1. `focus.ti.com/lit/ds/symlink/msp430f149.pdf` et `http://focus.ti.com/lit/ug/slau049f/slau049f.pdf`

```
 init_PORT();
 while (1) {
    P1OUT=led1;
    if(led1==0x80) {sens=1;}
    if(led1==0x01) {sens=0;}
    if(sens==0) {led1<<=1;}
    if(sens==1) {led1>>=1;}

    for(k=0;k<50000;k++) {}
 }
 return(0);
}
```

Compilation du programme :
`msp430-gcc -Os -mmcu=msp430f149 -o sortie.elf entree.c`
pour générer le listing correspondant :
`msp430-objdump -dSt sortie.elf > sortie.lst`
et la liste d'instructions au format hexadécimal Intel à partir du fichier ELF :
`msp430-objcopy -O ihex sortie.elf sortie.hex`
Rappel : `gcc -S` arrête la compilation à la génération de l'assembleur (fichier .s), `gcc -o` arrête la compilation à la génération des objets (avant le linker).

Une fois le programme compilé, il est transféré au MSP430 par l'interface JTAG, ici émulée par le port parallèle (imprimante) du PC. La commande pour transférer par le convertisseur parallèle-JTAG le programme `sortie.elf` est :
`msp430-jtag -e sortie.elf`
Chaque port d'entrée sortie nécessite de définir :
— la fonction de chaque broche avec le registre PxSEL : 0 pour du GPIO, 1 pour la fonction spécifique
— la direction du port dans PxDIR : 0 pour une entrée, 1 pour une sortie
— le niveau de la broche, 1 pour un niveau haut (3,3 V) et 0 pour le niveau bas (masse)

Exercices

1. analyser le schéma du circuit et identifier le port auquel sont connectées les diodes électroluminescentes (LED).
2. les composants 74245 (IC3 et IC4) se comportent comme des latchs : quelle est leur utilité dans ce circuit ?

Les ports du MSP430 sont capables d'absorber ou fournir la vingtaine de milliampères nécessaires à alimenter une LED.

Cet exemple exploite la fonction de décalage pour périodiquement changer l'état du port et donc allumer ou éteindre les diodes qui y sont connectées.

Exercices :

— au lieu de faire défiler les diodes, faire clignoter l'ensemble des 8 diodes simultanément.
— visualiser le code assembleur généré
— Comment s'écrit 2048 en hexadécimal ?
— comment est initialisé le pointeur de pile (R1) ? commenter ce choix, sachant que la RAM commence à l'adresse 0x200 (page 1-4 de slau049) et que ce processeur contient 2048 octets de RAM.
— modifier les options d'optimisation en modifiant l'option -Os par -O1, -O2 et -O3. Comparer les codes assembleur résultant.

4 Communiquer par RS232

Initialisation d'un port série : pour le port 0, avec un débit de 9600 bauds

```
#include  <msp430x14x.h>
#include  <string.h>
#include  <legacymsp430.h> // #include  <signal.h>
#include <msp430.h> // #include <io.h>
#include <stdio.h>

void init_rs1_9600(){
UCTL1=SWRST;  // SetupUART1: mov.b #SWRST,&UCTL1
UCTL1|=CHAR; //  bis.b #CHAR,&UCTL1
UTCTL1=SSEL0; //  mov.b #SSEL0,&UTCTL1
UBR01=0x03; //  mov.b #0x03,&UBR01
UBR11=0x00; //  mov.b #0x00,&UBR11
UMCTL1=0x4A; //  mov.b #0x4A,&UMCTL1
UCTL1&=~SWRST; //  bic.b #SWRST,&UCTL1
```

```
 ME2|=UTXE1+URXE1;  //  bis.b #UTXE1+URXE1,&ME2
}

void snd_rs1(char cara)
{while ((IFG2 & UTXIFG1)==0); // jz rs_tx1
 TXBUF1=cara;
}

void str_rs1(char* cara)
{int k=0;
 do {snd_rs1(cara[k]);k++;} while (cara[k]!=0);
}

void init_PORT(void){
  P3SEL=0xFE; //  ; P3.6,7 = USART select
  P3DIR=0x5B; //  ; P3.6 = output direction
  P3OUT=0x01; //  ; P3.6 = output val
}
```

```
int main (void)
{ char chaine[30];
 unsigned long k;

 WDTCTL = WDTPW + WDTHOLD;     // Stop WDT
 init_PORT();
 init_rs1_9600();

 while (1) {
    snd_rs1('!');snd_rs1(33);
    sprintf(chaine,"Hello World\r\n");
    str_rs1(chaine);
    for(k=0;k<8000;k++) {}
 }
 return(0);
}
```

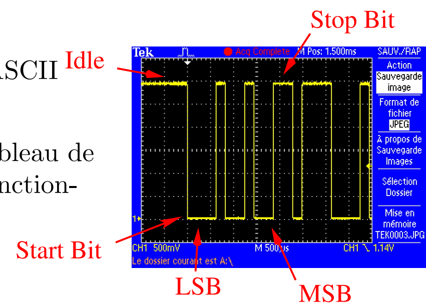
Le MSP430 possède un module de communication asynchrone (RS232) matériel. Les paramètres de configuration incluent le débit de communication (dans cet exemple 9600 bauds), le nombre de bits par trame transmise et l'encapsulation de ces trames. Les ordinateurs étant de moins en moins souvent équipés de port série, nous exploitons un composant convertisseur du format RS232 vers l'USB.

Exercice :
identifier le composant permettant la conversion RS232 vers USB sur le schéma de la carte.

Côté PC, la communication par port série se fait au moyen de `minicom` : `CTRL A-O` pour définir le port (`/dev/ttyUSB0`) et le contrôle de flux (ni matériel, ni logiciel). `CTRL A-P` pour définir la vitesse (ici 9600 bauds, N81).

Exercices :

1. envoyer une valeur hexadécimale (par exemple 0x55 et 0xAA) en ASCII
2. envoyer une valeur en décimal (par exemple 250) en ASCII
3. sachant qu'une chaîne de caractères est représentée en C par un tableau de caractères se terminant par le caractère de valeur 0, expliquer le fonctionnement de la procédure `str_rs1`.
4. dans le programme en C faisant clignoter les diodes, communiquer la valeur courante d'une variable.



5 Exploitation d'une interruption

Nous avons vu que le code assembleur généré depuis un programme C dépend des options d'optimisation. Cela signifie notamment que la durée d'exécution d'une boucle vide dépend de ces options, et évidemment de la nature du compilateur.

Le microcontrôleur possède en interne des compteurs (*timers*) qui permettent de mesurer avec précision un intervalle de temps selon une horloge cadencée sur un oscillateur externe et donc indépendant du code exécuté. Cette stratégie rend la durée d'exécution du programme indépendante de la nature des optimisations.

La fréquence de déclenchement du timer est définie par la variable TBCCR0. Le timer compte de 0 à TBCCR0, et lorsque la valeur stockée dans ce registre est atteinte, l'interruption se déclenche. Noter que de cette façon, le timer compte **TBCCR0+1** fois. La cadence de l'horloge dans notre configuration est 32768 Hz.

Par ailleurs, le microcontrôleur peut être placé en mode de veille au moyen des instructions LPM (Low Power Mode) terminées par le numéro du mode de veille. Plus le numéro est élevé, plus le nombre de périphériques désactivés est important, mais plus le nombre de sources d'interruption devient réduit. Ainsi, une utilisation très utile du timer est la sortie périodique du mode veille dans lequel on place le microcontrôleur afin d'économiser l'énergie.

Timer_B Control Register TBCTL

15	14	13	12	11	10	9	8
Unused	TBCLGRPx		CNTLx		Unused	TBSSELx	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TBCLR	TBIE	TBIFG
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	w–(0)	rw–(0)	rw–(0)

Unused Bit 15 Unused

TBCLGRP Bit 14-13 TBCLx group
- 00 Each TBCLx latch loads independently
- 01 TBCL1+TBCL2 (TBCCR1 CLLDx bits control the update)
  TBCL3+TBCL4 (TBCCR3 CLLDx bits control the update)
  TBCL5+TBCL6 (TBCCR5 CLLDx bits control the update)
  TBCL0 independent
- 10 TBCL1+TBCL2+TBCL3 (TBCCR1 CLLDx bits control the update)
  TBCL4+TBCL5+TBCL6 (TBCCR4 CLLDx bits control the update)
  TBCL0 independent
- 11 TBCL0+TBCL1+TBCL2+TBCL3+TBCL4+TBCL5+TBCL6 (TBCCR1 CLLDx bits control the update)

CNTLx Bits 12-11 Counter Length
- 00 16-bit, $TBR_{(max)}$ = 0FFFFh
- 01 12-bit, $TBR_{(max)}$ = 0FFFh
- 10 10-bit, $TBR_{(max)}$ = 03FFh
- 11 8-bit, $TBR_{(max)}$ = 0FFh

Unused Bit 10 Unused

TBSSELx Bits 9-8 Timer_B clock source select.
- 00 TBCLK
- 01 ACLK
- 10 SMCLK
- 11 Inverted TBCLK

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_B is not in use conserves power.
- 00 Stop mode: the timer is halted
- 01 Up mode: the timer counts up to TBCL0
- 10 Continuous mode: the timer counts up to the value set by TBCNTLx
- 11 Up/down mode: the timer counts up to TBCL0 and down to 0000h

Extrait du manuel de l'utilisateur du MSP430

```
#include  <msp430x14x.h>
#include  <string.h>
#include  <legacymsp430.h> // #include  <signal.h>
#include  <msp430.h> // #include <io.h>
#include <stdio.h>

void Sommeil(void);
void Pause_TIMERB(unsigned short Compteur);
void Pause_ms(unsigned short ms);
interrupt(TIMERB0_VECTOR ) wakeup Vector_TIMERB0(void);
void init_horloge(void);
void init_PORT(void);
void init_TIMER(void);

unsigned char TIMERB=0;

void Sommeil(void) {LPM3;}

void Pause_TIMERB(unsigned short Compteur)
{TIMERB=0;
 TBCCR0=Compteur;
 TBCTL=TBSSEL0+TBCLR + MC_1;
 while(TIMERB==0) Sommeil();
}

void Pause_ms(unsigned short ms)
{Pause_TIMERB((unsigned short)(ms*32768/1000));}


interrupt(TIMERB0_VECTOR ) wakeup Vector_TIMERB0(void)
{TIMERB=1;}

void init_horloge(void){
  BCSCTL1 = 0x87;
  BCSCTL2 = SELM1 + SELS;
  WDTCTL = WDTPW + WDTHOLD;
}

void init_PORT(void){
  P1SEL = 0x00;
  P1DIR = 0xFF;
  P1OUT = 0x00;
}

void init_TIMER(void)
{TBCTL=TBSSEL0 + TBCLR;
 TBCCTL0|=CCIE; // CCR0 interrupt enabled
 TBCCR0=16000;
 TBCTL|= MC_0;
}


int main (void)
{unsigned int led1=0x01,sens=0,pause;
 init_horloge();
 init_PORT();
 init_TIMER();
 eint();

 pause=15;

 while (1) {
    P1OUT=led1;
    if(led1==0x80) {sens=1;}
    if(led1==0x01) {sens=0;}
    if(sens==0) {led1<<=1;}
    if(sens==1) {led1>>=1;}
    Pause_ms(pause);
    if(pause>80) pause=1; else pause++;
 }
 return(0);
}
```

Exercices :

1. exploiter une interruption timer pour faire clignoter une diode à fréquence précisément déterminée depuis la fréquence du résonateur à quartz
2. écrire une fonction d'horloge temps-réel capable d'afficher le temps en secondes et minutes
3. modifier la vitesse de défilement du chenillard

Réduire la période d'oscillation des LEDs se traduit à terme par la perte de sensation de clignotement par persistance rétinienne. Cependant, si au lieu de commuter l'état d'une diode, nous commutons un haut parleur, un signal sonore de fréquence égale au signal d'excitation se fait entendre. Ce haut parleur est connecté à la broche P4.0. Un second timer (le MSP430 possède deux timers, TimerA et TimerB) permet de régler la durée des notes ainsi générées.

```
#include  <msp430x14x.h>
#include  <string.h>
#include  <legacymsp430.h> // #include  <signal.h>
#include  <msp430.h> // #include <io.h>
#include <stdio.h>

void Sommeil(void);
void Pause_TIMERB(unsigned short Compteur);
void Pause_ms(unsigned short ms);
interrupt(TIMERB0_VECTOR ) wakeup Vector_TIMERB0(void);
void init_horloge(void);
void init_PORT(void);
void init_TIMER(void);

unsigned int PWM=0;
unsigned int ma_note=0;
unsigned int note[14]={74,66,63,56,50,47,42,37,35, \
                       33,31,27,25,23};

#define labas 0
#define sibas 1
#define do 2
#define re 3
#define mi 4
#define fa 5
#define sol 6
#define la 7
#define sid 8
#define si 9
#define dohaut 10
#define rehaut 11
#define mihaut 12
#define fahaut 13
#define solhaut 14

#define A3 labas
#define B3 sibas
#define C3 do
#define D3 re
#define E3 mi
#define F3 fa
#define G3 sol
#define A4 la
#define B4x sid
#define B4 si
#define C4 dohaut
#define D4 rehaut
#define E4 mihaut
#define F4 fahaut
#define G4 solhaut

// #define Nbr_Max 16
// unsigned int musique[Nbr_Max]={sol,sol,sol,la,si,si,la,la,sol,\
//                               si,la,la,sol,sol,sol,sol};

#define Nbr_Max 98
unsigned int musique[Nbr_Max]= \
{A4 ,A4 ,E3 ,F3 ,G3 ,G3 ,F3 ,E3 ,D3 ,D3 ,D3 ,D3 ,F3 ,A4 ,A4 , \
 G3, F3 ,E3 ,E3 ,E3 ,E3 ,F3 ,G3,G3,A4,A4,F3,F3,D3,D3,D3,D3,\
 B4x,B4x,B4x,B4x,C4 ,D4 ,D4 ,C4 ,B4x,A4 ,A4 ,A4 ,A4 ,F3 ,A4 , \
 A4, G3 ,F3 ,E3 ,E3 ,E3 ,E3 ,F3,G3,G3,A4,A4,F3,F3,D3,D3,D3,D3,\
 B4x,B4x,B4x,B4x,C4 ,D4 ,D4 ,C4 ,B4x,A4 ,A4 ,A4 ,A4 ,F3 ,A4 ,A4\
,G3 ,F3 ,E3 ,E3 ,E3 ,E3 ,F3,G3,G3,A4,A4,F3,F3,D3,D3,D3,D3};

interrupt(TIMERA0_VECTOR ) /*wakeup*/ Vector_TIMERA0(void)
{ma_note++;
 if(ma_note==Nbr_Max) ma_note=0;
 P1OUT=(1<<(musique[ma_note]-D3));
 TBCCR0=note[musique[ma_note]];
}

interrupt(TIMERB0_VECTOR ) /*wakeup*/ Vector_TIMERB0(void)
{P4OUT^=0x01;}

void init_horloge(void){
  BCSCTL1 = 0x87;
  BCSCTL2 = SELM1 + SELS;
  WDTCTL = WDTPW + WDTHOLD;
}

void init_PORT(void){
  P1SEL = 0x00;
  P1DIR = 0xFF;
  P1OUT = 0x00;

  P4SEL=0x00;
  P4DIR=0xFF;
  P4OUT=0x00;
}

void init_TIMER(void){
 TACTL=TASSEL0 + TACLR;
 TACCTL0|=CCIE; // CCR0 interrupt enabled
 TACCR0=8192;
 TACTL|= MC_0;

 TBCTL=TBSSEL0 + TBCLR;
 TBCCTL0|=CCIE;
 TBCCR0=PWM;
 TBCTL|= MC_0;
}


int main (void)
{init_horloge();
 init_PORT();
 init_TIMER();
 eint();

 TACTL|= MC_1 + TACLR;
 TBCTL|= MC_1 + TBCLR;
 TBCCR0=note[musique[0]];

 while (1) {LPM3;}
 return(0);
```

6 Lire une valeur analogique

Tous les programmes réalisés jusqu'ici vont nous permettre d'atteindre le but de ce TP : mesurer une grandeur physique à intervalles de temps réguliers et retranscrire ces informations pour les rendre exploitables par un utilisateur.

```
#include  <msp430x14x.h>
#include  <string.h>
#include  <legacymsp430.h> // #include  <signal.h>
#include <msp430.h> // #include <io.h>
#include <stdio.h>

void init_rs1_9600(){
 UCTL1=SWRST;  // SetupUART1: mov.b #SWRST,&UCTL1
 UCTL1|=CHAR; //  bis.b #CHAR,&UCTL1
 UTCTL1=SSEL0; //  mov.b #SSEL0,&UTCTL1
 UBR01=0x03; //  mov.b #0x03,&UBR01
 UBR11=0x00; //  mov.b #0x00,&UBR11
 UMCTL1=0x4A; //  mov.b #0x4A,&UMCTL1
 UCTL1&=~SWRST; //  bic.b #SWRST,&UCTL1
 ME2|=UTXE1+URXE1;  //  bis.b #UTXE1+URXE1,&ME2
}

void snd_rs1(char cara)
{while ((IFG2 & UTXIFG1)==0); // jz rs_tx1
 TXBUF1=cara;
}

void str_rs1(char* cara)
{int k=0;
 do {snd_rs1(cara[k]);k++;} while (cara[k]!=0);
}

void init_PORT(void){
  P1SEL = 0x00;
  P1DIR = 0xFF;
  P1OUT = 0x00;

  P3SEL=0xFE; // ; P3.6,7 = USART select
  P3DIR=0x5B; // ; P3.6 = output direction
  P3OUT=0x01; // ; P3.6 = output val
}

void init_ADC(void){
  ADC12CTL0=SHT0_6+REFON+ADC12ON; // V_REF=1.5 V
  ADC12CTL1=SHP;
  ADC12MCTL0=INCH_10+SREF_1; // p.17-11: single conversion
  ADC12CTL0|=ENC;
}

unsigned short temperature()
{ADC12CTL0|=ADC12SC; //  start conversion
 do {} while ((ADC12CTL1 & 1)!=0);
 return(ADC12MEM0);
}


int main (void)
{ char chaine[30];
 unsigned short t;
 unsigned int temp;
 unsigned long k;

 WDTCTL = WDTPW + WDTHOLD;     // Stop WDT
 init_PORT();
 init_ADC();
 init_rs1_9600();

 while (1) {
    t=0;for (k=0;k<16;k++) t+=temperature();
    t/=16;

    temp=(int)(((float)t*1.5/4096.-0.986)/0.00355*10.);
    sprintf(chaine,"Temperature=%x\r\n",temp);
    str_rs1(chaine);
    for(k=0;k<8000;k++) {}
 }
 return(0);
}
```

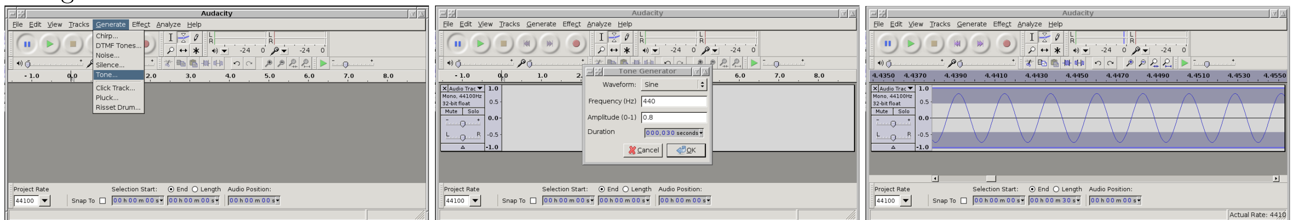
Le canal 10 (`INCH_10`) est un cas particulier : il s'agit d'une sonde interne de température.

Exercice :
lire la température et la transmettre sous forme hexadécimale (caractères ASCII) sur le port série.

La conversion s'obtient (User Manual p.17-16) par $V_{Temp} = 0,00355 \times Temp_{degC} + 0,986$. Nous observons expérimentalement un décalage entre la température et la valeur du microcontrôleur qu'il semble nécessaire de calibrer pour chaque nouveau composant.

La température est une grandeur qui varie lentement. Un microcontrôleur travaille à une fréquence élevée et est donc susceptible de mesurer des phénomènes se déroulant à des fréquences du même ordre de grandeur que les fréquences audibles.

Afin de générer des signaux de l'ordre de quelques dizaines de Hz à quelques kilohertz, nous allons exploiter la carte son qui équipe la majorité des ordinateurs personnels. Un logiciel, `audacity`[2] propose de synthétiser un signal et de l'émettre sur la sortie de sa carte son.



2. disponible pour GNU/Linux, MacOS X et MS-Windows à `http://audacity.sourceforge.net/`

Nous nous contenterons pour le moment de générer une sinusoïde, pour n'exploiter que plus tard les formes d'ondes plus complexes (chirp).

Exercices :

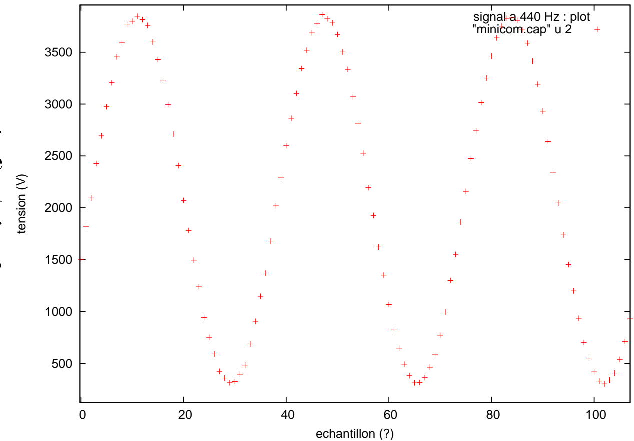
1. enregistrer une séquence issue de la carte son du PC
2. visualiser au moyen de GNU/Octave le résultat de l'acquisition (commandes identiques à celles de Matlab, voir section 8 pour un rappel)
3. établir la fréquence d'échantillonnage.

Nous pouvons suivre deux stratégies pour établir la fréquence d'échantillonnage f_e : temporelle ou fréquentielle. Dans le premier cas nous comptons le nombre M d'échantillons dans N périodes et établissons, connaissant la fréquence f_s du signal mesuré, alors

$$f_e = \frac{M}{N} f_s$$

D'un point de vue fréquentiel, la transformée de Fourier sur M points d'un signal numérisé à f_e s'étend de $-f_e/2$ à $+f_e/2$. Connaissant la fréquence de ce signal numérisé qui se présente dans la transformée de Fourier comme une raie de puissance maximale en position N, nous déduisons que

$$N/M = f_s/f_e \Leftrightarrow f_e = \frac{M}{N} f_s$$



7 Accès carte mémoire

Les microcontrôleurs 8 et 16 bits ne fournissent pas une puissance de calcul importante mais sont capable d'effectuer un certain nombre d'opérations simples telles qu'asservissements et acquisitions de données. Dans tous les cas, par soucis de contrôle *a posteriori* ou de dupliquer en local les données transmises en temps réel, les applications de ces microcontrôleurs sont étendues par la possibilité de stocker ces informations sur un support peu coûteux et consommant peu de courant en mode veille (Fig. 2). Nous nous proposons d'étudier le stockage d'informations sur cartes mémoires de type MutliMediaCard (MMC) ou Secure Digital (SD).

Afin de gagner du temps, nous ne détaillerons pas le protocole d'initialisation et de communication d'une carte SD : nous utiliserons une bibliothèque mise à disposition par son auteur pour conserver et relire des données sur support de stockage de masse non volatile [3]. L'utilisation d'une bibliothèque opensource ne prive cependant pas d'en lire le contenu en vue d'en comprendre le fonctionnement.

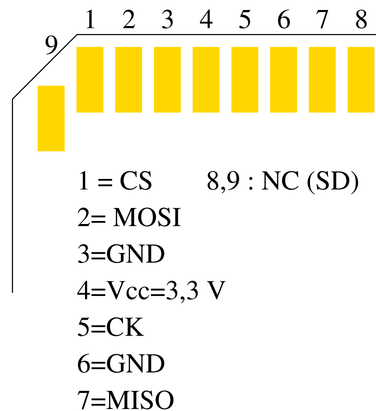


FIGURE 1 – Assignation des broches d'une carte SD. Les broches 8 et 9 n'existent pas dans les cartes MMC, qui fonctionnent sinon de la même façon.

Les cartes SD fonctionnent en deux mode : le mode natif, rapide et supportant l'encryption des données, et le mode SPI, plus lent mais facilement accessible depuis n'importe quel microcontrôleur. Nous utiliserons ce second mode, dans lequel seules 4 broches sont utiles (fig. 1).

3. http://www.true-random.com/homepage/projects/msp430_mmc/

Après initialisation par `if (initMMC() == MMC_SUCCESS)`, deux commandes de base nous servent à écrire (`mmcWriteBlock(adresse)`) et lire (`mmcReadBlock(adresse, taille)`) des données sur la carte. Il s'agit d'écritures non-formatées : la carte n'est pas lisible depuis un PC qui s'attend à trouver un système de fichier. Toute transaction sur une carte SD se fait par blocs de 512 octets (tableau `mmc_buffer`). Les informations ne sont pas stockées en zone non-volatile tant que l'acquittement d'écriture n'est pas émis : lors des acquisitions lentes, une partie de la dernière séquence d'acquisitions pourra être perdue lors de la mise hors tension du circuit.

Afin de compiler notre programme `programme.c` avec les bibliothèques d'accès à la carte SD :

```
msp430-gcc -mmcu=msp430f149 -o sd.elf test_jmf.c led.o mmc.o
```

Pour programmer et voir immédiatement le résultat :

```
msp430-jtag -e sd.elf && stty -F /dev/ttyUSB0 9600 && cat < /dev/ttyUSB0
```

Exercices :

— valider quelques fonctions de debuggage dont nous devons être certain du bon fonctionnement : allumage/extinction des diodes, transmission de données en RS232

— écrire dans un bloc de la carte SD une série de valeurs, réinitialiser le tableau de caractères et lire les valeurs stockées. Vérifier qu'il s'agit des valeurs écrites.

— écrire et relire sur plusieurs blocs mémoire. Attention : les adresses doivent *toujours* être multiples de 512.

— écrire des valeurs issues de la conversion analogique-numérique ou de la lecture d'un port série. Que se passe-t-il lors du passage d'un bloc de la carte SD au suivant ?

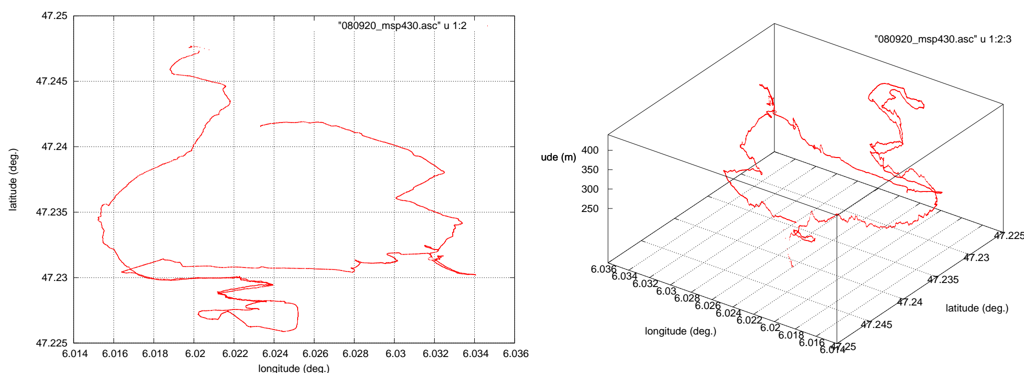


FIGURE 2 – Exemple d'acquisition de trames GPS avec une fréquence de 1 Hz, et restitution du tracé en 2D (gauche) et 3D (droite) sous `gnuplot`.

8 Afficher des données

Deux outils sont mis à disposition pour afficher des courbes : `gnuplot` et `octave` [4].

`gnuplot` affiche des courbes lues dans des fichiers :

```
plot "fichier"
```

on peut préciser les colonnes à afficher, l'axe des abscisses et ordonnées... :

```
plot "fichier" using 2:3 with lines
```

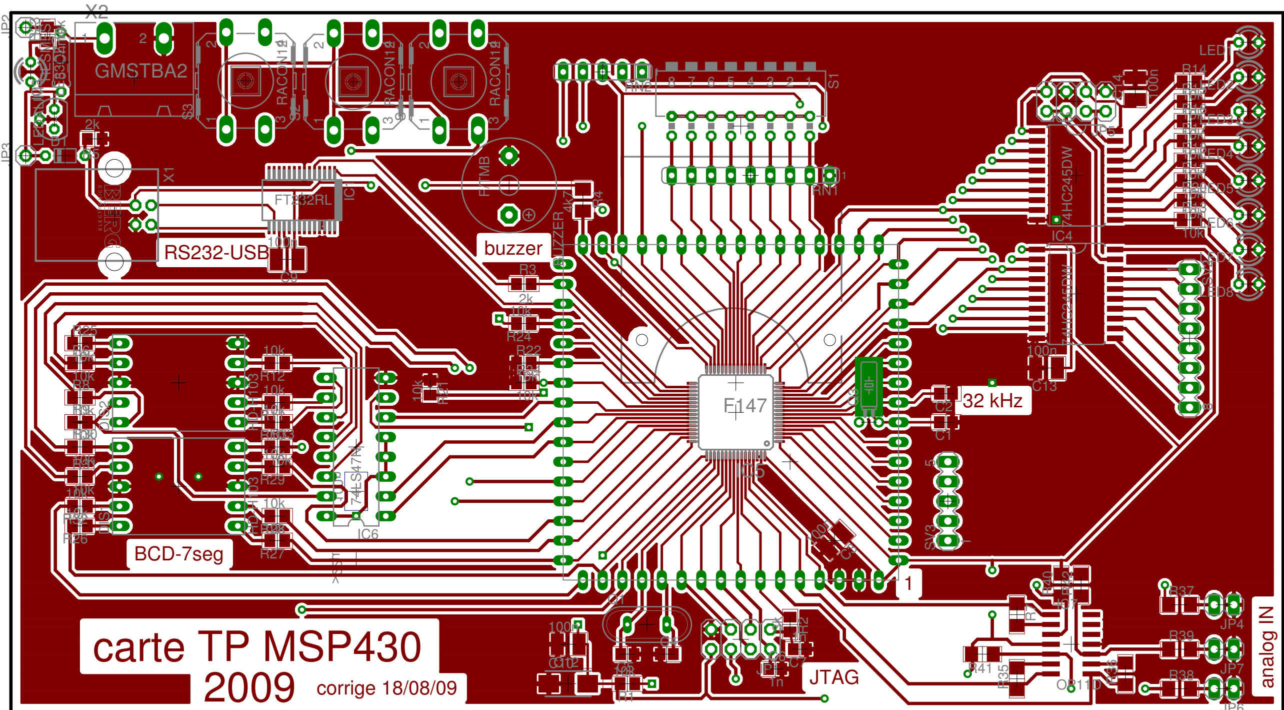
pour utiliser la seconde colonne comme abscisse et la troisième colonne comme ordonnée, les points étant reliés par des lignes. `help plot` pour toutes les options.

`octave` est une version opensource de Matlab et en reprend toutes les syntaxes. Seule fonctionnalité nouvelle qui nous sera utile ici : `octave` peut lire des données au format hexadécimal :

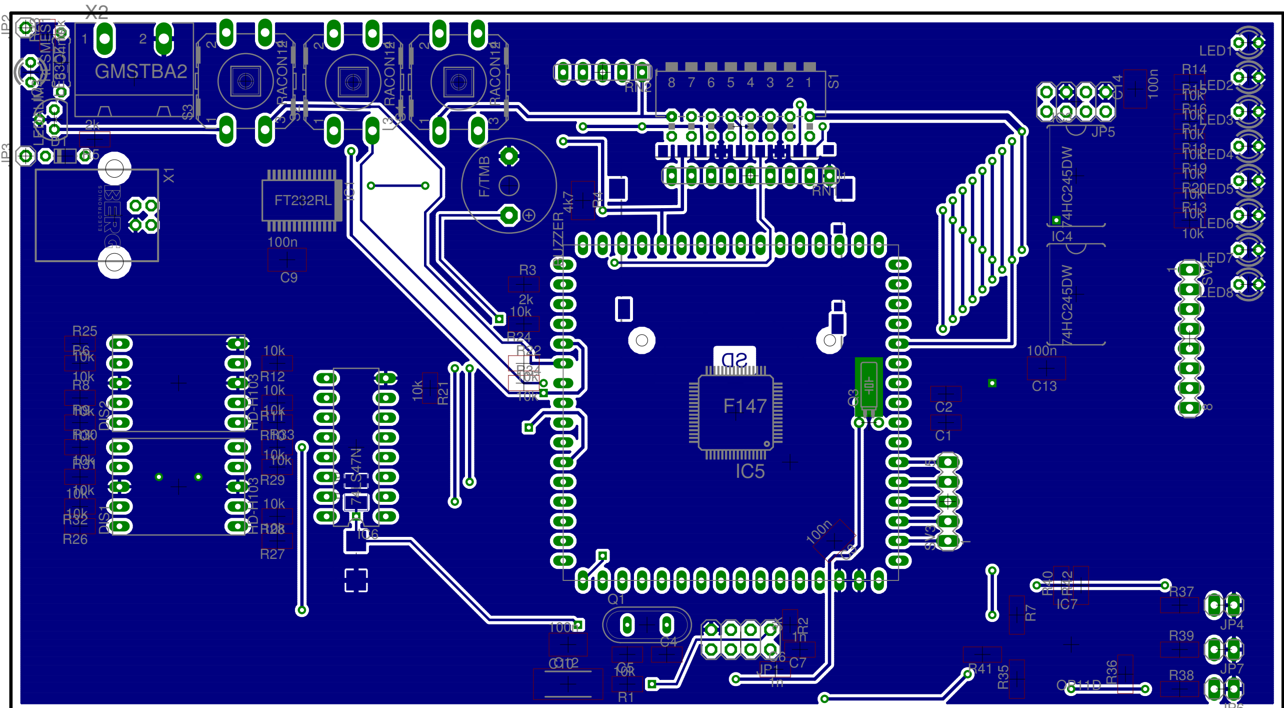
```
f=fopen("fichier","r");d=fscanf("%x");fclose(d);
```

La variable `d` contient alors les informations qui étaient stockées dans le fichier `fichier` au format hexadécimal. Si le fichier contenait `N` lignes, les données sont lues par colonne et les informations d'une colonne donnée d'obtiennent par `d(1:N:length(d))`.

4. Pour une description plus exaustive des fonctionnalités de `gnuplot` et `octave`, on pourra consulter `http://jmfriedt.free.fr/lm_octave.pdf`.



Routage de la face supérieure et implantation des composants



Routage de la face inférieure et implantation des composants