# TP microcontrôleur : STM32

J.-M Friedt, 22 juillet 2014

Objectif de ce TP :
— présentation de GNU/Linux et de commandes unix
— compiler un programme en C pour processeur x86 sous GNU/Linux
— compiler un programme pour microcontrôleur (STM32) sous GNU/Linux : faire clignoter une diode, exemples en C
— communication RS232, communication de données en ASCII
— timer, processus périodiques, interruptions
— conversion analogique-numérique et numérique-analogique de données, échantillonnage

## 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**. L'utilisateur sur le CD distribué se nomme `tp` et son répertoire de travail se situe donc dans `/home/tp`,
— `/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` pour les caractéristiques du processeur (type et puissance de calcul), ou `/proc/ioports` pour les plages d'adresses occupées par des périphériques.
— `/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.
— `sudo` : exécuter une commande en tant qu'administrateur, `sudo halt` pour arrêter le système

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 inutilement 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>

#define a 3+2
#define b 5
```

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

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`). L'extension de l'exécutable n'a pas d'importance : le caractère exécutable d'un fichier est donné par ses droits (`ls -l` doit afficher x pour que le fichier soit exécutable) et non par son nom.

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

Pour rappel, les séquences de compilation d'un programme par `gcc` s'obtiennent par :

1. `gcc -E` pour interrompre la compilation au préprocesseur (génération du code C suite à une analyse syntaxique)
2. `gcc -S` pour interrompre la compilation au compilateur (génération du code assembleur)
3. `gcc -c` pour interrompre la compilation à l'assembleur (génération d'un objet)

Le CD qui a été fourni contient les déclinaisons de `gcc` pour *cross*-compiler des binaires à destination d'autres cibles que le processeur x86 qui équipe le plus couramment les PCs. Y sont fournies les versions de `gcc` à destination de ARM, de MSP430, et de M68k.

# 3 Premier pas sur STM32

FIGURE 1 – La carte STM32VL-discovery

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*. La plate-forme utilisée est disponible pour un peu moins de 12 euros chez Farnell (référence 1824325) et ne nécessite aucun matériel additionnel pour fonctionner. Le microcontrôleur équipant cette carte est le bas de gamme de la série STM32 mais le cœur ARM Cortex-M3 est compatible avec toute la gamme de ce fabriquant. En particulier, les caractéristiques de cette carte sont :

- processeur STM32F100RB, cœur Cortex-M3 cadencé à 24 MHz, 128 KB flash, 8 KB RAM,
- 7 canaux DMA, 3 USARTs, 2 SPIs, 2 I2C, ADC, DAC, RTC, horloges, deux chiens de garde,
- interface ST-Link par USB pour déverminer/programmer,
- résonateur externe rapide (HSE) 8 MHz,
- résonateur externe lent (LSE) 32768 Hz.
- deux LEDs (vert, bleu) connectées au **port C** (broches **8 et 9**).

Le STM32 [1] est une classe de microcontrôleurs proposés par ST autour du cœur ARM Cortex-M3. Ce cœur, exploité par divers fondeurs sous d'autres dénominations, 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 (ARM), différente de l'architecture hôte (Intel 32 bits). Il s'agit donc de *cross*compilation. Un ouvrage conséquent d'exemples est proposé en [1]. Cependant, contrairement aux exemples de cet ouvrage, nous utiliserons une implémentation libre des bibliothèques d'accès aux périphériques qu'est `libopencm3`. Le rôle de la bibliothèque est de cacher au néophyte les détails du fonctionnement du matériel pour ne donner accès qu'à des fonctions plus explicites du type `gpio_set()` ou `gpio_clear()`. L'exemple ci-dessous fait clignoter une diode :

```
#include <libopencm3/stm32/f1/rcc.h>
#include <libopencm3/stm32/f1/gpio.h>

static void gpio_setup(void)
{
 rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPCEN);
 gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_2_MHZ, GPIO_CNF_OUTPUT_PUSHPULL, GPIO8|GPIO9);
}

int main(void)
{
 int i;
```

1. `www.st.com/stm32` et en particulier la série STM32F1xx `www.st.com/web/en/resource/technical/document/reference_manual/CD00171190.pdf`

```
 gpio_setup();
 gpio_set(GPIOC,GPIO8);
 gpio_clear(GPIOC,GPIO9);

 while (1) {
   gpio_toggle(GPIOC, GPIO8);
   gpio_toggle(GPIOC, GPIO9);
   for (i = 0; i < 800000; i++) __asm__("nop");
 }
 return 0;
}
```

Ce premier programme se contente de faire clignoter les deux diodes mais donne la séquence de base d'initialisation du microcontrôleur :

1. activer la source d'horloge du périphérique (GPIO)
2. définir la direction d'utilisation du périphérique (entrée ou sortie)
3. définir l'état initial du périphérique
4. une boucle infinie avec changement d'état des diodes et attente.

Le programme se compile [2] en complétant la commande de `gcc` par un certain nombre d'options :

```
arm-none-eabi-gcc -Wall -Wextra -Wimplicit-function-declaration -Wredundant-decls
  -Wmissing-prototypes -Wstrict-prototypes -Wundef -Wshadow -I~/sat/bin/../arm-none-eabi/include
  -fno-common -mthumb -mcpu=cortex-m3 -msoft-float -MD -DSTM32F1 -o usart.o -c usart.c
arm-none-eabi-gcc -o usart.elf usart.o -lopencm3_stm32f1 --static -Wl,--start-group -lc -lgcc
  -lnosys -Wl,--end-group -L~/sat/arm-none-eabi/lib/thumb/cortex-m3 -L~/sat/bin/../arm-none-abi/lib
  -T../stm32vl-discovery.ld -nostartfiles -Wl,--gc-sections -mthumb -mcpu=cortex-m3
  -msoft-float -mfix-cortex-m3-ldrd -Wl,--print-gc-sections
  -L~/sat/arm-none-eabi/lib/thumb/cortex-m3
```

Noter en particulier l'appel aux bibliothèques `c` et `opencm3` par les options `-lc` et `-lopencm3_stm32f1`. Par ailleurs, `-I` renseigne la liste des répertoires contenant les fichiers d'entête (`.h`) et `-L` la liste des répertoires contenant les implémentations des bibliothèques.

Lors du passage d'une famille de STM32 à l'autre, les 3 points auxquels il est bon de veiller sont

1. la fréquence de cadencement du cœur, 24 MHz pour le processeurs bas de gamme (VL), 72 MHz sinon,
2. la taille de la mémoire renseignée dans le *linker script* appelé par l'option `-T` (ici, `-T../stm32vl-discovery.ld`)
3. penser à préciser la famille du modèle de STM32 utilisé (par exemple `-DSTM32F10X_MD` pour la famille *Medium Density* du STM32F103).

Le listing correspondant se génère par :

```
arm-none-eabi-objdump -dSt usart.elf > usart.list
```

et la liste d'instructions aux divers formats attendus par les outils de programmation (format hexadécimal Intel ou Motorola, image binaire) à partir du fichier ELF :

```
arm-none-eabi-objcopy -Obinary usart.elf usart.bin
arm-none-eabi-objcopy -Oihex usart.elf usart.hex
arm-none-eabi-objcopy -Osrec usart.elf usart.srec
```

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 micocontrôleur par l'interface JTAG, ici émulée par le port série virtuel [3]. La commande pour transférer le programme `usart.bin` est :

```
st-flash write v1 usart.bin 0x8000000
```

Les exemples sont inspirés de [1] et des exemples de `libopencm3` distribués à `https://github.com/libopencm3/libopencm3-examples`. On notera par ailleurs que l'outil pour transférer un programme au format binaire depuis le PC vers un processeur STM32F1 en dehors de son installation sur une carte Discovery est disponible sous la forme de `stm32flash`.

**Exercice :** compiler avec deux options d'optimisation, `-O2` et `-O0`. Quel constat sur la fréquence de clignotement des diodes ?

2. la chaîne de compilation croisée proposée sur le CD est issue de `summon-arm-toolchain` disponible à `https://github.com/esden/summon-arm-toolchain`.
3. Le code source du programme `st-flash` est disponible à `https://github.com/texane/stlink`

# 4 Horloge et interruption

Afin de s'affranchir de la dépendance aux options de compilation, il *faut* cadencer les fonctions critiques en termes de latences non par du logiciel mais par le matériel dédié à ces fonctions : les horloges (*timer*). Ainsi, le même résultat que précédemment s'obtient par

```
#include <libopencm3/stm32/f1/rcc.h>
#include <libopencm3/stm32/f1/gpio.h>
#include <libopencm3/stm32/timer.h>
#include <libopencm3/cm3/nvic.h>
#include <libopencm3/stm32/exti.h>

static void clock_setup(void) {rcc_clock_setup_in_hse_8mhz_out_24mhz();} // max 24 MHz

static void gpio_setup(void)
{ rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPCEN); // GPIOC clock
  gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_50_MHZ,
                GPIO_CNF_OUTPUT_PUSHPULL, GPIO8|GPIO9);          // LED output
  gpio_set(GPIOC, GPIO9);                                        // initial state
  gpio_clear(GPIOC, GPIO8);
}

static void tim_setup(void)
{ rcc_peripheral_enable_clock(&RCC_APB1ENR, RCC_APB1ENR_TIM2EN); // TIM2 clock
  nvic_enable_irq(NVIC_TIM2_IRQ);                                // TIM2 interrupt
  timer_reset(TIM2);                                             // Reset TIM2

  timer_set_mode(TIM2, TIM_CR1_CKD_CK_INT, TIM_CR1_CMS_EDGE, TIM_CR1_DIR_UP);
  timer_set_prescaler(TIM2, 90);          // prescaler value.
  timer_disable_preload(TIM2);            // Enable preload
  timer_continuous_mode(TIM2);            // Continous mode
  timer_set_period(TIM2, 65535);          // Period
  timer_disable_preload(TIM2);            // ARR reload enable
  timer_enable_counter(TIM2);             // Counter enable.
  timer_enable_irq(TIM2, TIM_DIER_CC1IE); // Enable commutation interrupt.
}

void tim2_isr(void)   // isr = Interrupt Service Routine, fonction de gestion de l'interruption
{ if (timer_get_flag(TIM2, TIM_SR_CC1IF)) {
     timer_clear_flag(TIM2, TIM_SR_CC1IF);
     gpio_toggle(GPIOC, GPIO8 | GPIO9);
    }
}

int main(void)
{ clock_setup();
  gpio_setup();
  tim_setup();
  while (1) __asm("nop"); // ce programme ne fait ... rien !
  return 0;
}
```

Le *timer* se configure en divisant l'horloge cadençant le périphérique (*prescaler*) puis en définissant la période sous forme de valeur à atteindre par le compteur pour réinitialiser le décompte et déclencher une interruption.

Noter que le programme principal ne fait rien ! La commutation de l'état des diodes se fait dans le gestionnaire d'interruption qui n'est pas explicitement appelé mais déclenché par un évènement matériel.

# 5 Communiquer par RS232

Le STM32 est muni d'au moins deux interfaces de communication asynchrone (USART). L'interface connectée au convertisseur RS232-USB de la carte de développement est l'USART1. L'exemple qui suit aborde deux aspects :

- communication sur port asynchrone (RS232) avec initialisation des périphériques, initialisation des horloges, et initialisation du port de communication (protocole d'échange des bits). La fonction principale se charge alors de communiquer des caractères sur le port série,
- exploitation des bibliothèques standard du C (`stdio` et `stdlib`) et, en particulier dans cet exemple `printf()` qui nécessite le point d'entrée (*stub*) `_write()`.

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 115200 bauds, N81).

```
#include <libopencm3/stm32/f1/rcc.h>
#include <libopencm3/stm32/f1/gpio.h>
#include <libopencm3/stm32/usart.h>

//#define avec_newlib

#ifdef avec_newlib
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int _write(int file, char *ptr, int len);
#endif

static void clock_setup(void)
{ rcc_clock_setup_in_hse_8mhz_out_24mhz();
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPCEN); // Enable GPIOC clock
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPAEN); // Enable GPIOA clock
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_USART1EN);
}

static void usart_setup(void)
{ // Setup GPIO pin GPIO_USART1_TX/GPIO9 on GPIO port A for transmit. */
  gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_50_MHZ,
      GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO_USART1_TX);

  usart_set_baudrate(USART1, 115200);   // USART_BRR(USART1) = (uint16_t)((24000000<<4)/(38400*16));
  usart_set_databits(USART1, 8);
  usart_set_stopbits(USART1, USART_STOPBITS_1);
  usart_set_mode(USART1, USART_MODE_TX);
  usart_set_parity(USART1, USART_PARITY_NONE);
  usart_set_flow_control(USART1, USART_FLOWCONTROL_NONE);

  usart_enable(USART1);
}

static void gpio_setup(void)
{ gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_2_MHZ, GPIO_CNF_OUTPUT_PUSHPULL, GPIO9);
}

int main(void)
{ int i, c = 0;
  clock_setup();
  gpio_setup();
  usart_setup();
  while (1) {
    gpio_toggle(GPIOC, GPIO9);
    c = (c == 9) ? 0 : c + 1; // cyclic increment c
#ifndef avec_newlib
    usart_send_blocking(USART1, c + '0'); // USART1: send byte
    usart_send_blocking(USART1, '\r');
    usart_send_blocking(USART1, '\n');
#else
    printf("%d\r\n", (int)c);
#endif
    for (i = 0; i < 800000; i++) __asm__("NOP");
  }
  return 0;
}

// define newlib stub
#ifdef avec_newlib
int _write(int file, char *ptr, int len)
{ int i;
  if (file == STDOUT_FILENO || file == STDERR_FILENO) {
     for (i = 0; i < len; i++) {
         if (ptr[i] == '\n') {usart_send_blocking(USART1, '\r');}
         usart_send_blocking(USART1, ptr[i]);
       }
     return i;
  }
  errno = EIO;
  return -1;
}
#endif
```

Nous avons vu que l'utilisation de la bibliothèque d'entrée-sortie standard (`stdio`), et en particulier son implémentation pour systèmes embarqués, est gourmande en ressources. Pour s'en convaincre, compiler le programme avec (`#define avec_newlib`) et sans (commenter cette ligne). Nous constatons une augmentation significative du temps de programmation du microcontrôleur associée à une croissance significative de la taille du code – de 1580 octets à 29032 – qui inclut maintenant toutes ces fonctions.

**Exercice :**

1. constater le bon fonctionnement du programme en observant les trames communiquées sous `minicom`,
2. que se passe-t-il si un mauvais débit de communication est sélectionné ?
3. activer ou désactiver les fonctions standard du C implémentées par `newlib`. Quelle conséquence ? analyser en particulier ce que fait la fonction `_write()`, un des *stubs* de newlib.
4. envoyer une valeur hexadécimale (par exemple 0x55 et 0xAA) en ASCII, *i.e.* définir deux variables avec ces valeurs et écrire la fonction qui permette d'en afficher le contenu sous `minicom`,
5. envoyer une valeur en décimal (par exemple 250) en ASCII
6. sachant qu'une chaîne de caractères est représentée en C par un tableau de caractères se terminant par le caractère de valeur 0, ajouter une fonction d'envoi de chaîne de caractères.

L'interface de communication RS232 est le mode de communication privilégié entre un système embarqué et l'utilisateur. À peu près tous les circuits, même aussi complexes qu'un disque NAS Lacie [4], un routeur ethernet [5] ou NeufBox [6], exploitent ce genre d'interface en phase de développement.

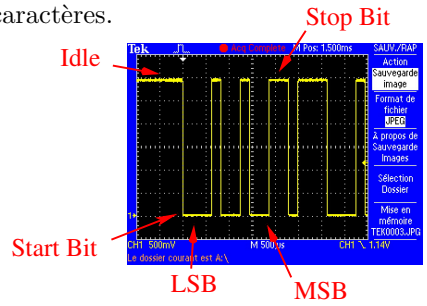
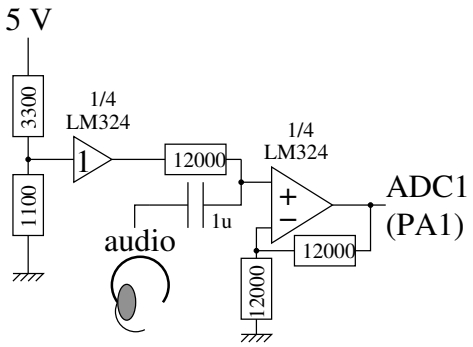



FIGURE 2 – Chronogramme de la liaison asynchrone.

# 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 <errno.h>
#include <stdio.h>
#include <unistd.h>

#include <libopencm3/cm3/nvic.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/usart.h>
#include <libopencm3/stm32/f1/rcc.h>
#include <libopencm3/stm32/f1/adc.h>

int _write(int file, char *ptr, int len);

#define nbr 1024

static void clock_setup(void)
{ rcc_clock_setup_in_hse_8mhz_out_24mhz();
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_USART1EN); // USART1
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPCEN); // port C (diodes)
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPAEN); // port A (USART1)
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_ADC1EN);
}

static void usart_setup(void)
{ gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_50_MHZ,
    GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO_USART1_TX);

  usart_set_baudrate(USART1, 115200);
  usart_set_databits(USART1, 8);
  usart_set_stopbits(USART1, USART_STOPBITS_1);
  usart_set_mode(USART1, USART_MODE_TX);
  usart_set_parity(USART1, USART_PARITY_NONE);
  usart_set_flow_control(USART1, USART_FLOWCONTROL_NONE);
  usart_enable(USART1);
}
```

4. http://lacie.nas-central.org/wiki/Serial_port_%282Big_2%29
5. http://maschke.de/wag54g/
6. http://www.neufbox4.org/wiki/index.php?title=Port_s%C3%A9rie

```
int _write(int file, char *ptr, int len)
{
  [... voir auparavant]
}

static void adc_setup(void)
{ int i;
  gpio_set_mode(GPIOA, GPIO_MODE_INPUT, GPIO_CNF_INPUT_ANALOG, GPIO0|GPIO1|GPIO2);
  adc_off(ADC1);
  adc_disable_scan_mode(ADC1);           // single conversion
  adc_set_single_conversion_mode(ADC1);
  adc_disable_external_trigger_regular(ADC1);
  adc_set_right_aligned(ADC1);
  adc_set_sample_time_on_all_channels(ADC1, ADC_SMPR_SMP_28DOT5CYC);
  adc_power_on(ADC1);
  for (i = 0; i < 800000; i++) __asm__("nop"); // demarrage ADC ...
  adc_reset_calibration(ADC1);
  adc_calibration(ADC1);
}

static uint16_t read_adc_naiive(uint8_t channel)
{ uint8_t channel_array[16];
  channel_array[0] = channel;                 // choix du canal de conversion
  adc_set_regular_sequence(ADC1, 1, channel_array);
  adc_start_conversion_direct(ADC1);
  while (!adc_eoc(ADC1));                     // attend la fin de conversion
  return(adc_read_regular(ADC1));
}

int main(void)
{int i,tab[nbr];
 int j = 0;
 clock_setup();
 usart_setup();
 gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_2_MHZ, GPIO_CNF_OUTPUT_PUSHPULL, GPIO8);
 gpio_set(GPIOC, GPIO8);
 adc_setup();

 while (1) {
      for (i=0;i<nbr;i++) tab[i]= read_adc_naiive(0);
      for (i=0;i<nbr;i++) printf("%d\n", tab[i]);
      gpio_toggle(GPIOC, GPIO8); /* LED on/off */
      for (i = 0; i < 100000; i++) __asm__("NOP");
 }
 return 0;
}
```

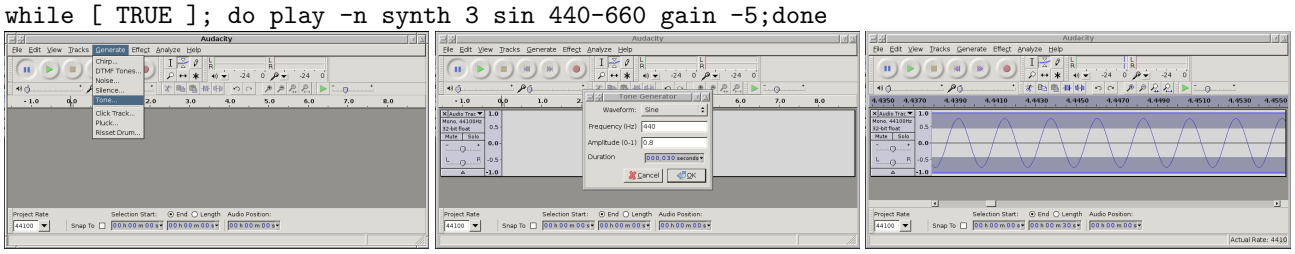
Le canal 16 est un cas particulier : il s'agit d'une sonde interne de température. Prendre soin cependant de l'activer pour l'utiliser.

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`[7] propose de synthétiser un signal et de l'émettre sur la sortie de sa carte son.

Sous GNU/Linux, un outil de configuration en ligne de commande, `play`, est fourni avec la boite à outils pour fichiers son `sox`. Pour générer un signal à fréquence fixe sur la carte son : `play -n synth sin 440 gain -5` (éviter un gain trop élevé pour ne pas saturer l'étage de sortie de la carte son, qui se configure par ailleurs au moyen de `alsamixer`).

Pour balayer la fréquence de sortie (fonction *chirp* de `audacity`) :

```
while [ TRUE ]; do play -n synth 3 sin 440-660 gain -5;done
```



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

**Exercices :**

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

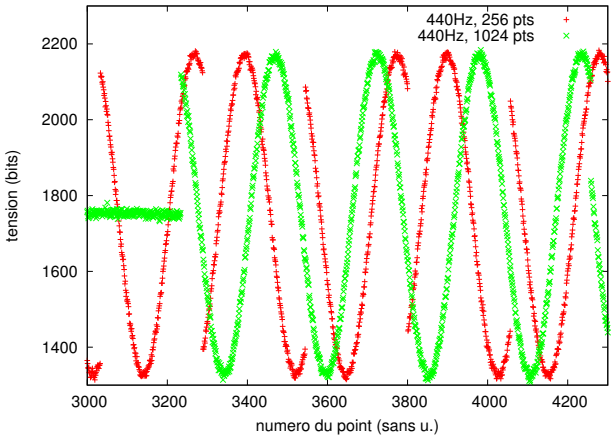
1. enregistrer une séquence issue de la carte son du PC
2. visualiser au moyen de GNU/Octave ou `gnuplot` le résultat de l'acquisition (commandes identiques à celles de Matlab, voir section 10 pour un rappel)
3. établir la fréquence d'échantillonnage.

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

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




## 6.1 GNURadio et `gnuradio-companion`

Un outil de traitement numérique du signal installé sur le CD est `gnuradio` accompagné de son interface graphique `gnuradio-companion`. Cet outil permet d'accéder à divers périphériques – dont la carte son du PC – pour acquérir et générer des signaux (éventuellement simultanément), ainsi que traiter ces signaux pour en extraire les signaux. Les modes de modulation les plus courants y sont fournis ainsi que divers outils classiques tels que l'analyse de Fourier, oscilloscope ... Cet outil *opensource* est particulièrement favorable à l'ajout de ses propres fonctions de traitement selon des modalités d'analyses inspirées des blocs existant (Fig. 3).

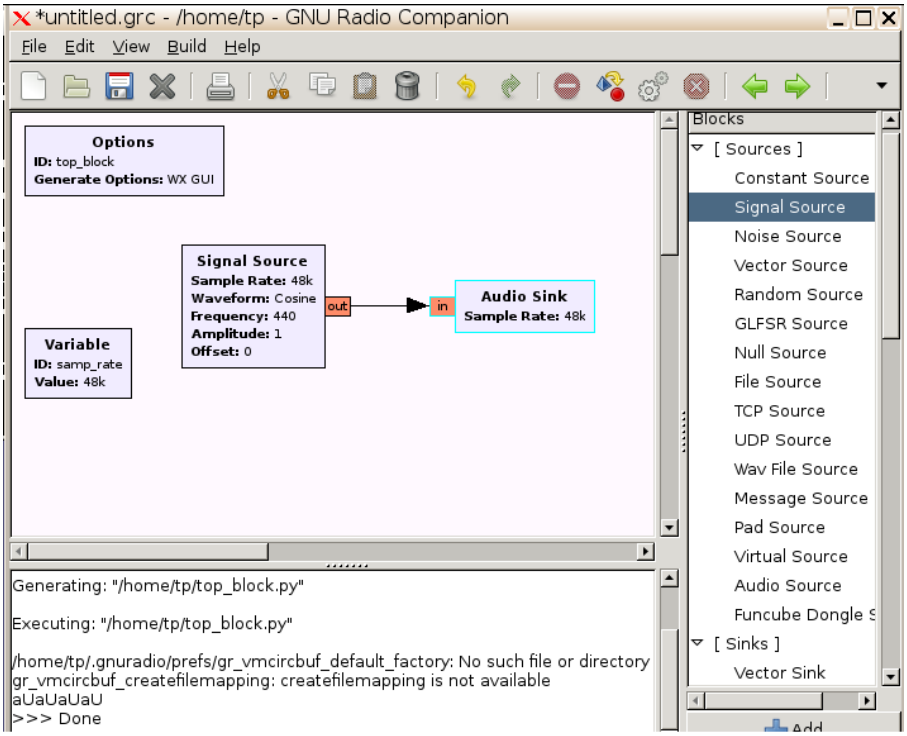



FIGURE 3 – Graphique `gnuradio-companion` pour générer un signal périodique sinusoïdal sur la sortie audio du PC.

# 7 Déclencher une lecture de valeur analogique sur *timer*

Nous avons vu auparavant que l'inconvénient de cadencer une opération sur une attente en boucle vide est la dépendance de la vitesse d'exécution avec la nature et version du compilateur ou avec les options d'optimisation. Une utilisation naturelle des horloges internes est de cadencer la conversion analogique-numérique. Une telle fonctionnalité est cablée au niveau du matériel du STM32 et ne nécessite qu'une configuration appropriée pour garantir la fréquence d'échantillonage conditionnée par l'horloge interne au processeur.

```
#include <libopencm3/stm32/f1/rcc.h>
#include <libopencm3/stm32/f1/flash.h>
#include <libopencm3/stm32/f1/gpio.h>
#include <libopencm3/stm32/f1/adc.h>
#include <libopencm3/stm32/usart.h>
#include <libopencm3/stm32/timer.h>
#include <libopencm3/cm3/nvic.h>

volatile uint16_t temperature[256] ;
volatile int jmf_index=0;

static void usart_setup(void)
{
        rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_USART1EN);

        /* Setup GPIO pin GPIO_USART1_TX/GPIO9 on GPIO port A for transmit. */
        gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_50_MHZ,
                 GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO_USART1_TX);


        /* Setup UART parameters. */
        usart_set_baudrate(USART1, 115200);
        usart_set_databits(USART1, 8);
        usart_set_stopbits(USART1, USART_STOPBITS_1);
        usart_set_mode(USART1, USART_MODE_TX_RX);
        usart_set_parity(USART1, USART_PARITY_NONE);
        usart_set_flow_control(USART1, USART_FLOWCONTROL_NONE);

        /* Finally enable the USART. */
        usart_enable(USART1);
}

static void gpio_setup(void)
{ rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPCEN);
  gpio_set_mode(GPIOA, GPIO_MODE_INPUT, GPIO_CNF_INPUT_ANALOG, GPIO0);
  gpio_set_mode(GPIOA, GPIO_MODE_INPUT, GPIO_CNF_INPUT_ANALOG, GPIO1);
  gpio_set_mode(GPIOC, GPIO_MODE_OUTPUT_2_MHZ, GPIO_CNF_OUTPUT_PUSHPULL, GPIO8|GPIO9);
}

static void timer_setup(void)
{ uint32_t timer;
  volatile uint32_t *rcc_apbenr;
  uint32_t rcc_apb;

  timer = TIM2;
  rcc_apbenr = &RCC_APB1ENR;
  rcc_apb = RCC_APB1ENR_TIM2EN;

   rcc_peripheral_enable_clock(rcc_apbenr, rcc_apb);
   timer_reset(timer);
   timer_set_mode(timer, TIM_CR1_CKD_CK_INT, TIM_CR1_CMS_EDGE, TIM_CR1_DIR_UP);
   timer_set_period(timer, 0xF*5);
   timer_set_prescaler(timer, 0x8);
   timer_set_clock_division(timer, 0x0);
   timer_set_master_mode(timer, TIM_CR2_MMS_UPDATE); // Generate TRGO on every update
   timer_enable_counter(timer);
}

static void irq_setup(void)
{ nvic_set_priority(NVIC_ADC1_2_IRQ, 0);
  nvic_enable_irq(NVIC_ADC1_2_IRQ);
}

static void adc_setup(void)
{ int i;
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_ADC1EN);
  adc_off(ADC1);
```

```
  adc_enable_scan_mode(ADC1);
  adc_set_single_conversion_mode(ADC1);
  adc_enable_external_trigger_injected(ADC1,ADC_CR2_JEXTSEL_TIM2_TRGO); // start with TIM2 TRGO
  adc_enable_eoc_interrupt_injected(ADC1);                              // Generate the ADC1_2_IRQ
  adc_set_right_aligned(ADC1);
//adc_enable_temperature_sensor(ADC1);
  adc_set_sample_time_on_all_channels(ADC1, ADC_SMPR_SMP_28DOT5CYC);
  adc_power_on(ADC1);

  for (i = 0; i < 800000; i++) __asm__("nop");
  adc_reset_calibration(ADC1);
  while ((ADC_CR2(ADC1) & ADC_CR2_RSTCAL) != 0);
  adc_calibration(ADC1);
  while ((ADC_CR2(ADC1) & ADC_CR2_CAL) != 0);
}

static void my_usart_print_int(uint32_t usart, int value)
{ int8_t i;
  uint8_t nr_digits = 0;
  char buffer[25];
  if (value < 0) {usart_send_blocking(usart, '-');value=-value;}
  while (value > 0) {buffer[nr_digits++] = "0123456789"[value % 10];value /= 10;}
  for (i = (nr_digits - 1); i >= 0; i--) {usart_send_blocking(usart, buffer[i]);}
  usart_send_blocking(usart, '\r');usart_send_blocking(usart, '\n');
}

int main(void)
{uint8_t channel_array[16];
 int k;

  rcc_clock_setup_in_hse_8mhz_out_24mhz();
  gpio_setup();
  usart_setup();
  timer_setup();
  irq_setup();
  adc_setup();

  gpio_set(GPIOC, GPIO8|GPIO9);
  channel_array[0] = 0; // channel number for conversion
  adc_set_injected_sequence(ADC1, 1, channel_array);
  while (1) {
     if (jmf_index>=255)
        {for (k=0;k<256;k++) my_usart_print_int(USART1, temperature[k]);
         jmf_index=0;
        }
     gpio_toggle(GPIOC, GPIO8);
  }
  return 0;
}

void adc1_2_isr(void)
{ ADC_SR(ADC1) &= ~ADC_SR_JEOC;
  if (jmf_index<256) {temperature[jmf_index] = adc_read_injected(ADC1,1);jmf_index++;}
}
```

# 8 Conversion numérique analogique

Implémenter une loi de commande suppose être capable d'interagir avec le système observé de fqn continue, ou tout au moins avec plus d'états que la simple commande binaire. Un convertisseur numérique-analogique (DAC) fonctionnant sur $N$ bits et configuré par un mot $M$ génère une tension (sortie analogique) $V$ comprise entre 0 V et une tension de référence $V_{REF}$ selon

$$V = \frac{V_{REF} \cdot M}{2^N}$$

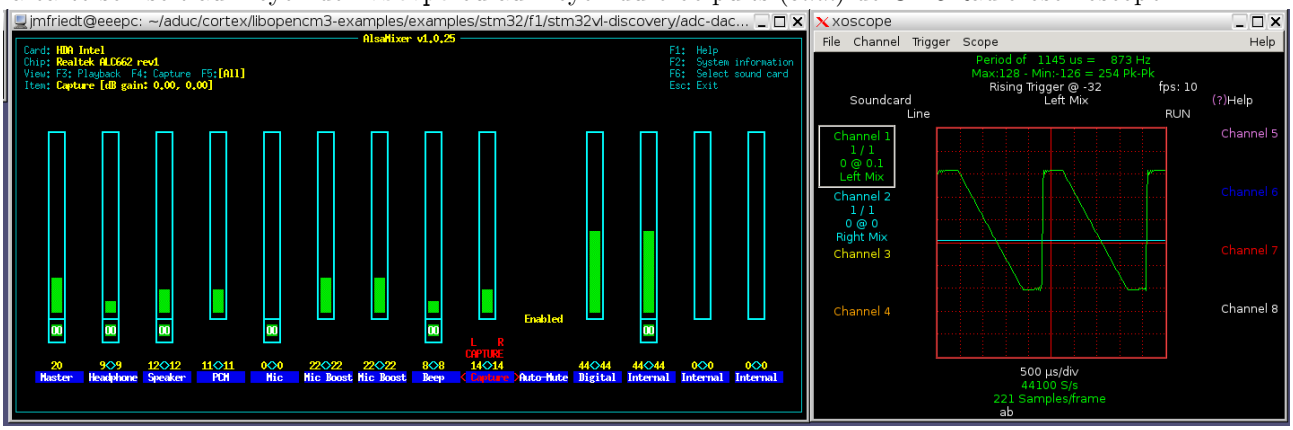
Le DAC s'initialise par

```
rcc_peripheral_enable_clock(&RCC_APB1ENR, RCC_APB1ENR_DACEN);
gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_50_MHZ, GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO4);
dac_disable(CHANNEL_1);
dac_disable_waveform_generation(CHANNEL_1);
dac_enable(CHANNEL_1);
dac_set_trigger_source(DAC_CR_TSEL2_SW);
```

puis s'utilise au moyen de

```
int target=0;
dac_load_data_buffer_single(target, RIGHT12, CHANNEL_1);
dac_software_trigger(CHANNEL_1);
target+=3; if (target>3000) target=0;
```

Le résultat de la génération de rampes sur le convertisseur numérique-analogique s'observe sur l'entrée de la carte son soit au moyen de `xoscope` ou au moyen du bloc puits (*sink*) de GNURadio oscilloscope :




Cette figure illustre l'enregistrement par carte son (logiciel `xoscope` pour exploiter la carte son comme oscilloscope) lorsque le convertisseur numérique-analogique est configuré pour générer des signaux en dent de scie. La carte son inverse le signal en entrée et est configurée (activation de l'entrée, gain) au moyen de `alsamixer`.

# 9 Accès carte mémoire

Les microcontrôleurs ne fournissent pas une puissance de calcul importante (quoi que ...) mais sont capables 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. Nous nous proposons d'étudier le stockage d'informations sur cartes mémoires de type MutliMediaCard (MMC) ou Secure Digital (SD) comme cas ludique d'application de la communication synchrone sur bus SPI.

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. Nous avons abordé en détail le protocole de communication sur bus synchrone SPI et les étapes d'initialisation de la cart mémoire dans [2].

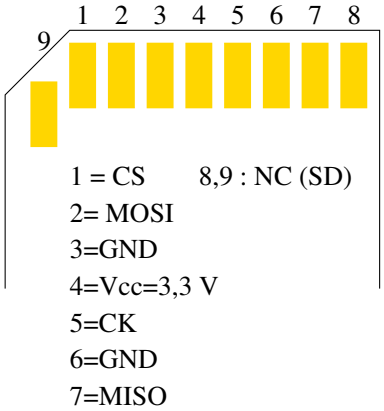



FIGURE 4 – Assignation des broches d'une carte SD. Les broches 8 et 9 n'existent pas dans les cartes MMC, qui fonctionnent sinon de la même façon.

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. 4). Le stockage sur carte SD en mode natif, bloc par

bloc, est efficace du point de vue du microcontrôleur mais ne permet pas de relire les informations depuis un ordinateur personnel qui n'est pas informé du format de stockage. Afin de respecter une norme connue par les systèmes d'exploitation exécutés sur ordinateurs personnels, il est judicieux de recherche un format de stockage encore d'actualité tout en étant compatible avec les ressources de calcul d'un petit microcontrôleur : le format MS-DOS FAT répond à ces exigences. Il reste néanmoins lourd à implémenter et surtout à déverminer : nous profiterons donc des implémentations libres disponibles sur le web [2].

# 10 Afficher des données

Deux outils sont mis à disposition pour afficher des courbes : `gnuplot` et `octave` [8].
`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 (pour ce qui nous concerne ici, `load` et `plot`). Noter que contrairement à `gnuplot` qui peut extraire des colonnes de données d'à peu près m'importe que format de fichier ASCII (*i.e.* quelquesoit le nombre de colonnes par ligne), Matlab et GNU/Octave exigent de charger des fichiers contenant une matrice dont le nombre de colonnes est constant dans toutes les lignes, et sans chaînes de caractères autres que les commentaires commençant par %. Seule fonctionnalité nouvelle de GNU/Octave par rapport à Matlab 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))`.

# Références


[1] G. Brown, *Discovering the STM32 Microcontroller*, version du 8 Janvier 2013, disponible à `www.cs.indiana.edu/~geobrown/book.pdf`

[2] G. Goavec-Mérou, J.-M Friedt, *Le microcontroleur STM32 : un cœur ARM Cortex-M3*, GNU/Linux Magazine France n.148 (Avril 2012), disponible à `jmfriedt.free.fr`

[3] J.- M Friedt, G. Goavec-Mérou, *Traitement du signal sur système embarqué – application au RADAR à onde continue*, GNU/Linux Magazine France n.149 (Mai 2012), disponible à `jmfriedt.free.fr`


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