

Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

`jmfriedt@femto-st.fr`

transparents à `jmfriedt.free.fr`

30 novembre 2016

Plan des interventions :

6 cours/TD :

- 1 Électronique numérique et conception du circuit
aspects analogiques, consommation électrique, lecture de datasheet
Survol des divers périphériques qui seront abordés (RS232, SPI, timer, ADC)
représentation des données (tailles/encodage), masques, architecture
 - 2 Rappel des bases du C :
Présentation du compilateur gcc, pointeurs, étapes de compilation
 - 3 Fonctionnement du compilateur (optimisations)
 - 4 Compilation séparée, passages de paramètres, bus de communication série, synchrone/asynchrone
 - 5 bibliothèques
libopencm3, newlib & stubs, ressources requises par les bibliothèques
-
- 6 méthodes de développement séparant algorithmique et bas niveau
timer, gestion des horloges du STM32, ADC
portabilité et test du code sur PC, arithmétique sur systèmes embarqués

4 TP :

- 1 commandes de base sous unix & compilateur C
- 2 prise en main du microcontrôleur STM32 : horloges, UART, SPI
- 3 timer, conversion analogique numérique et exploitation des données
- 4 caractérisation du microsystème

Compiler son compilateur

Une chaîne de compilation nécessite

- ① le compilateur (gcc)
- ② les outils pour convertir les divers formats de fichiers (binutils)
- ③ les **bibliothèques** (newlib)

+ debugger (gdb) + outils de programmation du microcontrôleur
(avrdude, stm32flash, dfu-programmer ...)

```
configure --target=arm-none-eabi --prefix=${HOME}/sat
```

Automatiser ce processus dans un script pour générer un ensemble
"cohérent" d'outils¹ :

<https://github.com/jmfriedt/summon-arm-toolchain.git>

1. explication de son utilisation à http://jmfriedt.free.fr/summon_arm.pdf

Le problème ...

Dans un programme en C, l'appel à `malloc()` se traduit par une erreur à l'édition de liens

```
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-sbrkr.o): In function '_sbrkr':  
newlib/libc/reent/sbrkr.c:58: undefined reference to '_sbrk'
```

Idem pour `printf()`

```
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-sbrkr.o): In function '_sbrkr':  
newlib/libc/reent/sbrkr.c:58: undefined reference to '_sbrk'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-writer.o): In function '_write_r':  
newlib/libc/reent/writer.c:58: undefined reference to '_write'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-closer.o): In function '_close_r':  
newlib/libc/reent/closer.c:53: undefined reference to '_close'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-fstatr.o): In function '_fstat_r':  
newlib/libc/reent/fstatr.c:62: undefined reference to '_fstat'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-isatty.o): In function '_isatty_r':  
newlib/libc/reent/isatty.c:58: undefined reference to '_isatty'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-lseekr.o): In function '_lseek_r':  
newlib/libc/reent/lseekr.c:58: undefined reference to '_lseek'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-readr.o): In function '_read_r':  
newlib/libc/reent/readr.c:58: undefined reference to '_read'
```

Exploitation d'une bibliothèque – newlib

<https://sourceware.org/newlib/libc.html#Stubs> : 17 stubs, colle entre newlib et nos programmes³.

- **_exit** : quitter un programme sans fermer les fichiers ouverts
- environ :
- execve :
- fork :
- fstat :
- getpid :
- isatty :
- kill :
- link :
- lseek :
- open :
- **read** : lire depuis un fichier
- sbrk : *utilisé par malloc*
- stat :
- times :
- unlink :
- wait :
- **write** : écrire dans un fichier, incluant stdout

3. http://wiki.osdev.org/Porting_Newlib

Exploitation d'une bibliothèque – newlib

Exemple d'implémentation (envoyer stdio sur port série)

```
_ssize_t _read_r(struct _reent *, int file, void *ptr, size_t len)
{char c; int i; unsigned char *p;
 p = (unsigned char*)ptr;
 for (i = 0; i < len; i++)
   {while ( global_index == 0) {} // !uart0_kbhit() );
    c = global_tab[0];global_index=0; // (char) uart0_getc();
    if (c == 0x0D) {*p='\0';break;}
    *p++ = c; jmf_putchar(c,NULL,0,0);
  }
 return len - i;
}

_ssize_t _write_r ( struct _reent *, int file, const void *ptr, size_t len)
{int i; const unsigned char *p;
 p = (const unsigned char*) ptr;
 for (i = 0; i < len; i++)
   {if (*p == '\n') jmf_putchar('\r',NULL,0,0);
    jmf_putchar(*p++,NULL,0,0);
  }
 return len;
}

void jmf_putchar (int a,char *chaine, int* chaine_index,int USARTx)
{ if (chaine != NULL) {chaine[*chaine_index]=a;*chaine_index=*chaine_index+1;}
  else {if (usart_port==1)
    {USART_SendData (USART1, a);
     while(USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET) { ; }
    }
  else
    {USART_SendData (USART2, a);
     while(USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET) { ; }
    }
  }
}
```

Démonstration

```
#ifdef use_stdio
    for (tempe=1;tempe<10;tempe++) {mf=mf*lf;}
    printf( "\n+mf=%d\n" ,(int)mf);

    t=(char*)malloc(200); // malloc requiert _sbrk
    printf( "^%x\n^%x\n" ,(int)t[0] , t[199]);
    memset(t,0x55,200);
    printf( "^%x\n^%x\n" ,(int)t[0] , t[199]);
#endif

    while (1)
    {
#ifdef use_stdio
        put_chars(welcome);
#else
        printf( "%s" ,welcome);
#endif
        if (i<10) put_char(USART1,i+'0');
            else put_char(USART1,i+'A'-10);
        i++;if (i==16) i=0;
    }
    ...
```


Démonstration

Compromis entre temps de développement, fonctionnalités et occupation de ressources

```
3240 main.bin  
31824 main_stdio.bin
```

Ici, affichage par `printf()`, allocation de mémoire par `malloc()`, calculs flottants

Un émulateur pour tester ses programmes en l'absence du matériel :

https://github.com/beckus/qemu_stm32

```
qemu-system-arm -M stm32-p103 -serial stdio -serial stdio -kernel main.bin
```

suppose néanmoins une implémentation “correcte” des périphériques.

`simavr` pour Atmega32U2

Conception de logiciels

- 1 Séparer le code lié au matériel (initialisation et accès aux ressources matérielles) de l'algorithmique
- 2 Compilation séparée : choisir une implémentation du matériel ou une émulation logicielle (gcc supporte une multitude de plateformes)
- 3 Permet de tester son code sur PC avant de le porter au microcontrôleur (simulation de diverses conditions matérielles) – test automatique, en particulier de cas pathologiques
- 4 Exploitation d'un simulateur pour connaître l'état interne de la machine qui séquence le code
- 5 qemu pour un grand nombre de plateformes, simavr pour Atmega32U2

```
unsigned short interroge (unsigned short puissance, unsigned int freq, unsigned int v12);  
{ unsigned int v12;  
  FTW0[1] = (freq & 0xFF000000) >> 24;  
  FTW0[2] = (freq & 0xFF0000) >> 16;  
  FTW0[3] = (freq & 0xFF00) >> 8;  
  FTW0[4] = (freq & 0xFF);  
  [...]  
  v12 = readADC12 ();  
  readerF2_CLR; // coupe reception  
  TIM_ITConfig (TIM3, TIM_IT_Update, ENABLE);  
  return (v&0x03F);
```

Conception de logiciels

- 1 Séparer le code lié au matériel (initialisation et accès aux ressources matérielles) de l'algorithmique
- 2 Compilation séparée : choisir une implémentation du matériel ou une émulation logicielle (gcc supporte une multitude de plateformes)
- 3 Permet de tester son code sur PC avant de le porter au microcontrôleur (simulation de diverses conditions matérielles) – test automatique, en particulier de cas pathologiques
- 4 Exploitation d'un simulateur pour connaître l'état interne de la machine qui séquence le code
- 5 qemu pour un grand nombre de plateformes, simavr pour Atmega32U2

```
unsigned short interroge(unsigned short puissance, unsigned int freq, \
    __attribute__((unused)) unsigned int offset, __attribute__((unused)) unsigned
{float reponse;
    reponse = exp(-(((float)freq-f01)/df)*(((float)freq-f01)/df))*3100.;
    reponse += exp(-(((float)freq-f02)/df)*(((float)freq-f02)/df))*3100.;
    reponse += (float)((rand()-RAND_MAX)/2)/bruit; // ajout du bruit;
    if (reponse < 0.) reponse = 0.;
    if (reponse > 4095.) reponse = 4095.;
    usleep(60);
    return((unsigned short)reponse);
}
```

Cas des interruptions

Version PC

```

void handle_alarm(int xxx)
{ // printf("RTC : %d %d %d\n",seconde, →
  ↪ minute, heure);
  tim0+=10;
  seconde+=10;
  if (seconde > 600){minute++;seconde=0;}
  if (minute > 60) {heure++;minute=0;}
  alarm(1);
}

void handle_io(int xxx)
{ // io_happened=1;
  global_tab[global_index] = getchar ();
  if (global_index < (NB_CHARS - 1))
    global_index++;
}

int main() {
  signal(SIGALRM, handle_alarm);

  io_happened=0;
  struct termios buf;
  tcgetattr(0, &buf);
  buf.c_lflag &= ~(ECHO | ICANON);
  buf.c_cc[VMIN]=1;
  buf.c_cc[VTIME]=0;
  tcsetattr(0, TCSAFLUSH, &buf);
  signal(SIGIO, handle_io);
  int savedflags=fcntl(0, F_GETFL, 0);
  fcntl(0, F_SETFL, savedflags | O_ASYNC | →
    ↪ O_NONBLOCK );
  fcntl(0, F_SETOWN, getpid());
  alarm(1); // si on veut simuler le timeout
}

```

Version microcontrôleur

```

void tim3_isr(void) // VERSION LIBOPENCM3
{
  if (TIM_GetITStatus(TIM3, TIM_IT_Update)!== →
    ↪ RESET)
  {TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
  tim0++;
  seconde++; // seconde en 1/10 →
    ↪ seconde
  if (seconde > 600)
    {seconde = 0; minute++;}
  if (minute > 60)
    {minute = 0; heure++;}
  if (heure > 24)
    {heure = 0;}
  }
}

void usart1_isr(void)
{
  if( USART_GetITStatus(USART1, USART_IT_RXNE →
    ↪ ))
  {USART_ClearITPendingBit(USART1, →
    ↪ USART_IT_RXNE);
  USART_ClearFlag(USART1, USART_IT_RXNE);
  {
    global_tab[global_index]= →
    ↪ USART_ReceiveData(USART1);
    if (global_index < (NB_CHARS-1)) →
    ↪ global_index++;
  }
  }
}

```

Profiter des outils d'analyse de code

valgrind pour étudier les fuites de mémoire

```
int main()
{int i[3],k;
  for (k=0;k<10;k++) {i[k]=k; printf("%d ", i[k]);}
}
```

qui s'exécute pour donner Segmentation fault ...

... est analysé par valgrind -q ./mon_programme et donne

```
==7410== Access not within mapped region at address 0x2
==7410== at 0x8048448: main (demo_valgrind.c:5)
```

(noter la différence de comportement en ajoutant fflush(stdout) ou \n!)

Profiter des outils d'analyse de code

gprof pour étudier le temps d'exécution de chaque fonction

```
void fonction1s() {volatile int k;for (k=0;k<0x1000000;k++) {};}
void fonction2s() {volatile int k;for (k=0;k<0x2000000;k++) {};}
void fonction3s() {volatile int k;for (k=0;k<0x3000000;k++) {};}

int main()
{fonction1s();
 fonction2s();
 fonction3s();
}
```

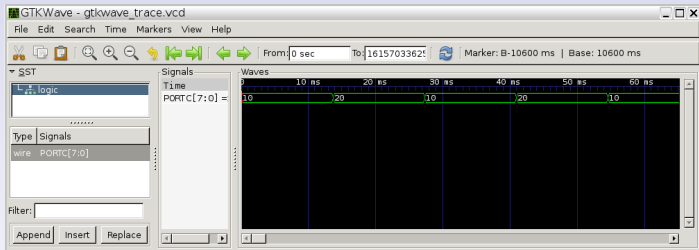
qui se compile avec l'option `-pg` pour générer à l'exécution `gmon.out` ...

... qui s'analyse par `gprof -bp ./mon_programme` et donne

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
55.56	0.10	0.10	1	100.00	100.00	fonction3s
33.33	0.16	0.06	1	60.00	60.00	fonction2s
11.11	0.18	0.02	1	20.00	20.00	fonction1s

Simulateur

- 1 Exploitation d'un simulateur pour connaître l'état interne de la machine qui séquence le code
- 2 qemu pour un grand nombre de plateformes, simavr pour Atmega32U2



Capture d'écran de gtkwave utilisé pour afficher l'évolution du port C.

Préfixer son programme des déclarations de simulations :

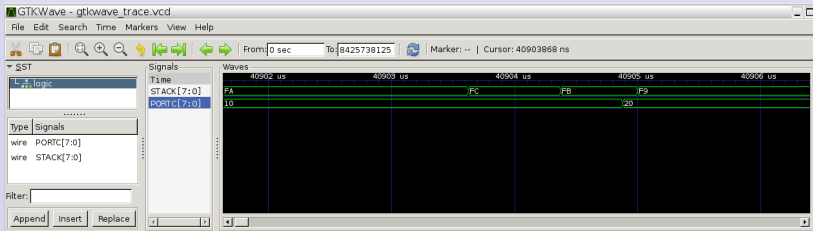
```
#include "avr_mcu_section.h"
AVR_MCU(F_CPU, "atmega32");

const struct avr_mmcu_vcd_trace_t _mytrace[] _MMCUC_ = {
    { AVR_MCU_VCD_SYMBOL("PORTB"), .what = (void*)&PORTB, },
};

int main { DDRB |=1<<PORTB5; while (1){PORTB^=1<<PORTB5; _delay_ms(1000);}}
```

Simulateur

- 1 Exploitation d'un simulateur pour connaître l'état interne de la machine qui séquence le code
- 2 `qemu` pour un grand nombre de plateformes, `simavr` pour `Atmega32U2`



Capture d'écran de `gtkwave` utilisé pour afficher l'évolution de la pile.

Préfixer son programme des déclarations de simulations :

```
#include "avr_mcu_section.h"
AVR_MCU(F_CPU, "atmega32");

const struct avr_mmcu_vcd_trace_t _mytrace[] _MMCU_ = {
    { AVR_MCU_VCD_SYMBOL("PORTC"), .what = (void*)(0x28), }, // &PORTC
    { AVR_MCU_VCD_SYMBOL("STACKL"), .what = (void*)(0x5D), }, // stack=0x3{DE}+0→
    { AVR_MCU_VCD_SYMBOL("STACKH"), .what = (void*)(0x5E), }, // stack=0x3{DE}+0→
};
```


Conclusion

Cette séparation permet

- les outils de profilage et d'analyse de code (`lint`, `valgrind` ...)
- les outils de test unitaire (`cunit`)