

# Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

`jmfriedt@femto-st.fr`

transparents à `jmfriedt.free.fr`

22 novembre 2020

## Plan des interventions :

7 cours/TP d'introduction au STM32 en *C baremetal* :

- 1 Électronique numérique et conception du circuit  
L3 : 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  
Rappels sur Atmega32U4 (Makefile, compilation, masques ...)
- 2 Premiers pas sur le STM32, adresses des périphériques, architecture
- 3 Fonctionnement de gcc et optimisations :  
préprocess–compilateur–assembleur–linker, passage C à assembleur, pointeurs
- 4 bibliothèques et séparation algorithme/matériel, simulateurs  
libopencm3, newlib & stubs, ressources requises par les bibliothèques
- 5 Bus de communication série, synchrone/asynchrone
- 6 arithmétique sur systèmes embarqués  
entiers, flottants, convertir un algorithme exprimé en nombres à virgules vers  
des entiers, timers
- 7 interruptions et acquisition de données analogiques, fréquence  
d'échantillonnage  
vecteurs d'interruption, gestion des horloges du STM32, ADC

# 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'outils<sup>1</sup>.

---

1. explication de son utilisation à [http://jmfriedt.free.fr/summon\\_arm.pdf](http://jmfriedt.free.fr/summon_arm.pdf)

Implication de la bibliothèque de **calculs flottants** :

	sans stdio, avec une division flottante	12950
toujours en thumb :	sans stdio, avec atan sur flottant	17090
	avec stdio, avec une division flottante	80397
	avec stdio, avec atan sur flottant	80397

Thumb est un sous-ensemble d'instructions pour processeur ARM codé sur 16 bits au lieu de 32 bits<sup>2</sup>

---

Diverses implémentations libres des bibliothèques standard du C :

- glibc (<http://www.gnu.org/software/libc/>, utilisée par le noyau Linux) ...
- ... et eglibc (*embedded glibc*, <http://www.eglibc.org/home>) ont fusionné,
- uClibc (<http://www.uclibc.org/>, utilisée par uClinux),

---

2. J.-M Friedt, É. Carry, *Développement sur processeur à base de cœur ARM7 sous GNU/Linux*, GNU/Linux Magazine France **117** (Juin 2009), pp.40-59

## 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 '_sbrk_r':  
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 '_sbrk_r':  
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 programmes<sup>3</sup>.

- **\_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](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 *r, 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 *r, 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*1f;}
    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](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

## Stubs

- Un programme principal contient toute l'“intelligence” du code : algorithmique
- Ce programme fait exclusivement appel à des fonctions génériques indépendantes du matériel : *stubs*
- L'accès de ces fonctions au matériel est implémenté dans des fichiers séparés
- Algorithmique est lié aux appels matériel par le compilation (compilation séparée)
- Possibilité de passer des arguments par Makefile : variables et tests conditionnels
- Édition de lien génère un exécutable

algorithmique led_on() led_off() communiquer()	matériel1 GP3DAT  = 0x00070000; GP3DAT  = (~0x00070000); COMTX = a;	matériel2, bibliothèque1 GPIO_SetBits(GPIOC,GPIO_Pin_2); GPIO_ResetBits(GPIOC,GPIO_Pin_1); USART_SendData(USART1, a);	matériel2, bibliothèque2 gpio_set (GPIOA, GPIO4); gpio_clear (GPIOA, GPIO4); usart_send_blocking (USART1,a)
---	--	--	--

```

Makefile
MODBUS = 0
#include makefile.cortex_AD9958
#include makefile.cortex_XE1203
include makefile.aduc
ifeq ($(MODBUS),1)
...
endif

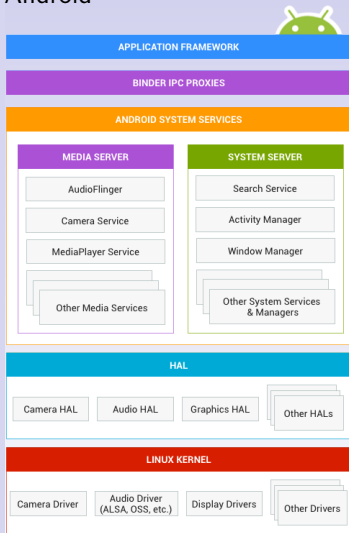
```

Edition de liens → exécutable

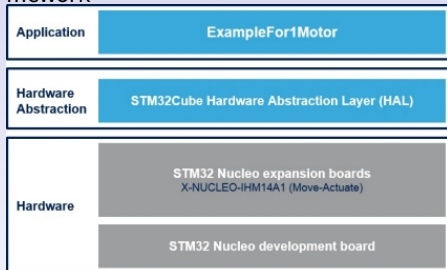
- remplacement “aisé” de la plateforme matérielle sans modifier l’algorithme (HAL)
- “fausses fonctions” sur PC pour émuler les appels : test unitaire

# HAL : Hardware Abstraction Layer Android

# HAL



## ST Microelectronics development framework



<https://source.android.com/devices/architecture>

<https://www.st.com/en/embedded-software/x-cube-spn14.html>

## 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 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

valgrind pour étudier les fuites de mémoire

```
int main()  
{ char *c;  
  c=(char*) malloc(100);  
}
```

est analysé par

valgrind --leak-check=full ./programme  
pour indiquer que

```
==20235== HEAP SUMMARY:  
==20235==      in use at exit: 100 bytes in 1 blocks  
==20235==    total heap usage: 1 allocs, 0 frees, 100 bytes allocated  
==20235==  
==20235== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1  
==20235==    at 0x483577F: malloc (vg_replace_malloc.c:309)  
==20235==    by 0x109146: main (in ...)  
==20235==  
==20235== LEAK SUMMARY:  
==20235==    definitely lost: 100 bytes in 1 blocks
```

valgrind a détecté que le tas n'est pas vide en fin d'exécution

## Profiter des outils d'analyse

valgrind pour étudier les accès illégaux de mémoire

```
int main()  
{ char *c;  
  c=(char*) malloc(100);  
  c[100]=10;  
  free(c);  
}
```

est analysé par

valgrind -q ./programme  
pour indiquer que

```
==21801== Invalid write of size 1  
==21801==    at 0x109163: main (in ...)  
==21801==   Address 0x4a490a4 is 0 bytes after a block of size 100 alloc'd  
==21801==    at 0x483577F: malloc (vg_replace_malloc.c:309)  
==21801==    by 0x109156: main (in ...)
```

Étrangement, alors que

```
char *c; c=(char*)malloc(100); c[100]=10; free(c);
```

et

```
static char c[100]; c[100]=10;
```

assignent tous deux c sur le tas, seule la première erreur d'allocation est détectée par valgrind.



## Profiter des outils d'analyse

valgrind pour étudier accès à une zone mémoire non-allouée  
(moins évident à analyser sur la pile : corruption du compteur de  
programme au retour de la fonction printf())

```
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

```
==22408== Jump to the invalid address stated on the next line  
==22408==    at 0x700000006: ???  
==22408==    by 0x900000007: ???  
==22408==    by 0x48ACBBA: (below main) (libc-start.c:308)  
==22408== Address 0x700000006 is not stack'd, malloc'd or (recently) free'  
==22408==  
==22408==  
==22408== Process terminating with default action of signal 11 (SIGSEGV)  
==22408== Access not within mapped region at address 0x700000006  
==22408==    at 0x700000006: ???  
==22408==    by 0x900000007: ???  
==22408==    by 0x48ACBBA: (below main) (libc-start.c:308)
```

## Profiter des outils d'analyse

gprof pour étudier le temps d'exécution de chaque fonction

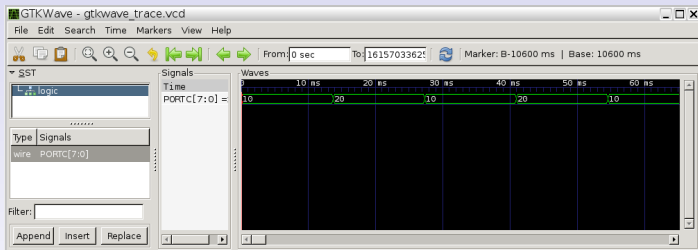
```
void fonction1s()  
{ volatile int k; for (k=0;k<0x1000000;k++) {};}  
  
void fonction2s()  
{ volatile int k; for (k=0;k<0x2000000;k++) {};}  
  
void fonction3s()  
{ volatile int k; for (k=0;k<0x3000000;k++) {};}  
  
int main()  
{ fonction1s();  
  fonction2s();  
  fonction3s();  
}
```

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);}}
```



# Émulation des périphériques

Un émulateur simule le comportement du cœur du processeur, mais comment émuler les périphériques<sup>4</sup> ?

Exemple du convertisseur analogique-numérique du STM32 :

```
static void stm32_adc_start_conv(Stm32Adc *s)
{uint64_t curr_time = qemu_clock_get_ns(QEMU_CLOCK_VIRTUAL);
 int channel_number=stm32_ADC_get_channel_number(s,1);
 if (channel_number==16)
 {s->Vdda=rand()%(1200+1) + 2400; // Vdda belongs to the interval [2400 3600] mv
  s->Vref=rand()%(s->Vdda-2400+1) + 2400; // Vref belongs to the interval [2400 Vdda] mv
  s->ADC_DR= s->Vdda - s->Vref;
 }
 else if (channel_number==17)
 {s->ADC_DR= (s->Vref=rand()%(s->Vdda-2400+1) + 2400); //Vref [2400 Vdda] mv
 }
 else
 {s->ADC_DR=((int)(1024.*(sin(2*M.PI*qemu_clock_get_ns(QEMU_CLOCK_VIRTUAL)/1e9)+1.))&0xff->
 ↵);
 }
 s->ADC_SR&=~ADC_SR_EOC; // indicates ongoing conversion
 s->ADC_CR2&=~ADC_CR2_SWSTART;
 // calls conv-complete when expires
 timer_mod(s->conv_timer, curr_time + stm32_ADC_get_nbr_cycle_per_sample(s,channel_number));
}
```

4. [https://github.com/beckus/qemu\\_stm32/blob/stm32/hw/arm/stm32\\_adc.c#L700](https://github.com/beckus/qemu_stm32/blob/stm32/hw/arm/stm32_adc.c#L700)

# Communication RS232<sup>5</sup> (32U4)

```
#include "avr_mcu_section.h"
AVR_MCU(F_CPU, "atmega32");
AVR_MCU_VCD_FILE("trace_file.vcd", 1000);
const struct avr_mmcu_vcd_trace_t _mytrace[] _MMCU = {
    { AVR_MCU_VCD_SYMBOL("UDR1"), .what = (void*)&UDR1, }, };
```

pour déclarer le registre observé (UDR1), puis

```
#define UART_BAUDRATE (57600)
// #define UART_BAUDRATE (9600)

void init_uart(void)
{ unsigned short baud;
  UCSR1A = (1 << UDR1); // importantly U2X1 = 0
  UCSR1B = (1 << RXEN1) | (1 << TXEN1); // enable receiver and transmitter
  UCSR1C = (1 << UCSZ11) | (1 << UCSZ10); // 8N1 // UCSR1D = 0; // no rtc/cts
  baud = ((( F_CPU / ( UART_BAUDRATE * 16UL )) - 1));
  UBRR1H = (unsigned char)(baud >> 8);
  UBRR1L = (unsigned char)baud;
}

void uart_transmit (unsigned char data)
{ while (!(UCSR1A & (1 << UDR1))); // wait while register is free
  UDR1 = data; // load data in the register
}

void send_byte(unsigned char c)
{ unsigned char tmp;
  tmp = c >> 4; if (tmp < 10) uart_transmit(tmp + '0'); else uart_transmit(tmp + 'A' - 10);
  tmp = (c & 0x0f); if (tmp < 10) uart_transmit(tmp + '0'); else uart_transmit(tmp + 'A' - 10);
}
```



5. J.-M Friedt, *Développer sur microcontrôleur sans microcontrôleur : les émulateurs*, GNU/Linux Magazine France HS 103 (Juillet-Aout 2019)

## Nouvelle architecture : RISC V

```
#include <stdio.h>
int main()
{long x=0x12345678;
 char *c=(char*)&x;
 printf("\n%d\n", sizeof(long));
 printf("%hhx %hhx %hhx %hhx\n", c[0], c[1], c[2], c[3]);
}
```

**Rappel :** newlib nécessite le *stub* write() pour afficher

```
FEMTO=[...]/riscv-probe/
riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -mcmmodel=medany -c h.c
riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -mcmmodel=medany \
  -nostartfiles -nostdlib -nostdinc -static -lgcc \
  -T $(FEMTO)/env/qemu-sifive_e/default.lds \
  $(FEMTO)/build/obj/rv32imac/env/qemu-sifive_e/crt.o \
  $(FEMTO)/build/obj/rv32imac/env/qemu-sifive_e/setup.o h.o \
  $(FEMTO)/build/lib/rv32imac/libfemto.a -o hello32
```

s'exécute dans qemu par

```
$ [...]/riscv-qemu/riscv32-softmmu/qemu-system-riscv32 -M sifive_e \
  -nographic -kernel hello32
```

4

```
00000078 00000056 00000034 00000012
```

→ long est codé sur cette architecture 32 bits sur 4 octets

→ 0x78 apparaît en premier, l'octet de poids faible est à l'adresse la plus faible : *little endian*

## Nouvelle architecture : RISC V

```
#include <stdio.h>
int main()
{long x=0x12345678;
 char *c=(char*)&x;
 printf("\n%d\n",sizeof(long));
 printf("%hhx %hhx %hhx %hhx\n",c[0],c[1],c[2],c[3]);
}
```

**Rappel** : newlib nécessite le *stub* write() pour afficher

En compilant en 64 bits par (options `-march=` et `-mabi`) :

```
riscv64-unknown-elf-gcc -march=rv64imac -mabi=lp64 -mcmmodel=medany -c h.c
riscv64-unknown-elf-gcc -march=rv64imac -mabi=lp64 -mcmmodel=medany \
-nostartfiles -nostdlib -nostdinc -static -lgcc \
-T $(FEMTO)/env/qemu-sifive_e/default.lds \
$(FEMTO)/build/obj/rv64imac/env/qemu-sifive_e/crt.o \
$(FEMTO)/build/obj/rv64imac/env/qemu-sifive_e/setup.o h.o
$(FEMTO)/build/lib/rv64imac/libfemto.a -o hello64
```

nous obtenons sous la version 64 bits de qemu le résultat

```
$ [...] /riscv-qemu/riscv64-softmmu/qemu-system-riscv64 -M sifive_e \
-nographic -kernel hello64
```

8

```
00000078 00000056 00000034 00000012
```

→ long occupe 8 octets (64 bits)

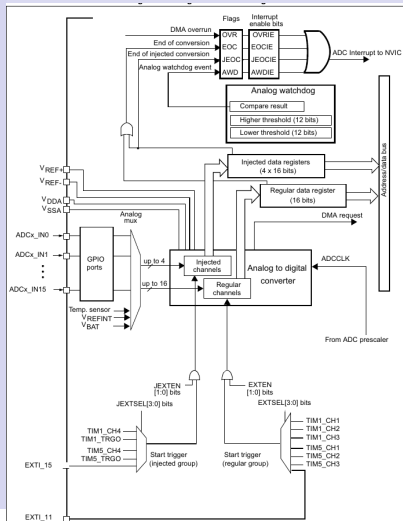
→ toujours *little endian*



# Conclusion

Cette séparation permet d'utiliser

- outils de profilage et d'analyse de code (lint, valgrind ...)
- outils de test unitaire (cunit)
- un nouveau processeur avec le même algorithme de traitement



Unique convertisseur analogique-numérique : 1×12-bit à 2,4 MSPS

```
void init_adc(void)
{
    gpio_mode_setup (GPIOB, GPIO_MODE_ANALOG, →
                    ↪ GPIO_PUPD_NONE, GPIO0|GPIO1);

    adc_power_off (ADC1);
    adc_disable_scan_mode (ADC1);
    // ADCCLK < 30 MHz, from APB2 ⇒ prescale →
    ↪ tq APB2/DIV<30 MHz
    adc_set_clk_prescale (ADC_CCR_ADCPRE_BY4);
    adc_set_sample_time_on_all_channels (ADC1, →
    ↪ ADC_SMPR1_SMP_112CYC);
    // si ADCCLK = 21 MHz, samptime = 144 ⇒ →
    ↪ Tconv = 7.4286 us.
    adc_power_on (ADC1);
}

unsigned short read_adc (unsigned char →
    ↪ channel)
{
    uint8_t channel_array [16];
    channel_array [0] = channel;
    adc_set_regular_sequence (ADC1, 1, →
    ↪ channel_array); // used channel
    adc_start_conversion_regular (ADC1);
    while (!adc_eoc (ADC1));
    return (adc_read_regular (ADC1));
}
```