Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

15 octobre 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. Compilation séparée, Fonctionnement du compilateur (optimisations, passages de paramètres)

---

4. bus de communication & bibliothèques
   protocoles de communication série
   libopencm3, newlib & stubs, ressources requises par les bibliothèques
5. 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
6.

4 TP :

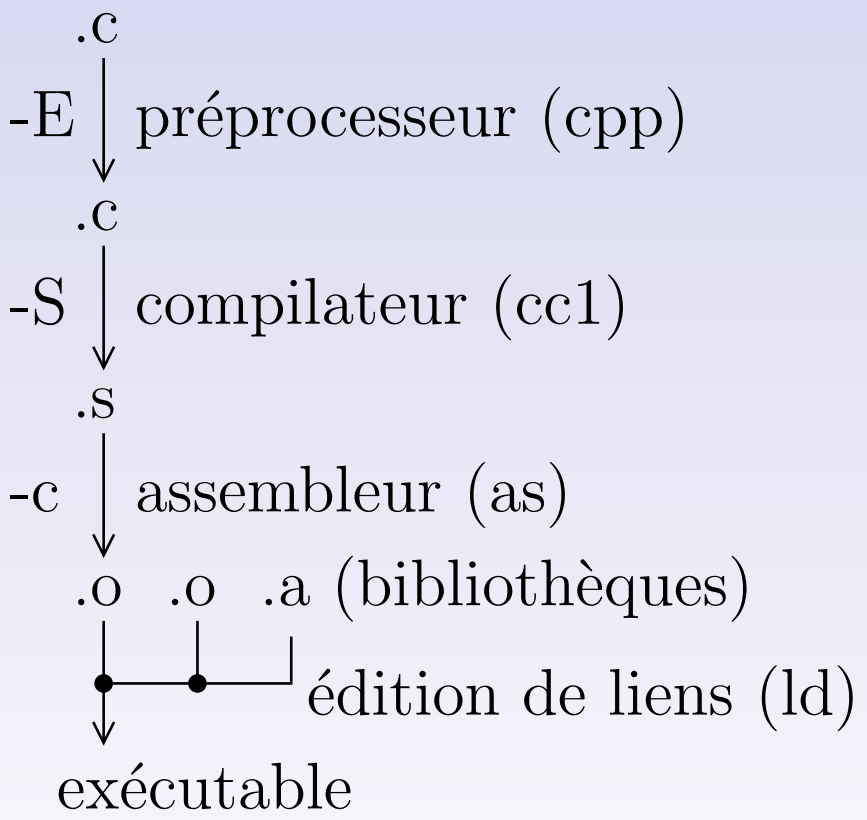
1. commandes de base sous unix & compilateur C
2. prise en main du microcontrôleur STM32 : horloges, UART, SPI
3. timer, conversion analogique numérique et exploitation des données
4. caractérisation du microsystème

Compilateur : pourquoi le maîtriser ?

1. passage du C au langage machine
2. conséquences du compilateur : les options d'optimisation font varier le temps d'exécution
3. spécificités du développement sur système à ressources réduites : pas d'allocation dynamique de mémoire, processus monolithique unique
4. instructions pour contrôler les optimisations
5. variables globales et `goto`, interruptions

Passage du C au langage machine

1. étapes du compilateur : préprocesseur, compilateur, assembleur, éditeur de liens (*linker*)
2. `gcc` est un frontend pour appeler le préprocesseur (`cpp`, `gcc -E`), le compilateur (`cc1`, `gcc -s`), l'assembleur (`as`, `gcc -c`) et le linker (`ld`)
3. séquence de compilation se découvre par `gcc -v`



Séquence de compilation

Comment passer de ce programme en C à une suite d'opcodes compréhensibles par un processeur ?

```c
#include <math.h>

#define i_init 3

int main()
{volatile int i=i_init;
 i=i+1;
}
```

gcc -E

```
...
extern double asin (double __x) __attribute__ ((→
    ↪__nothrow__ , __leaf__)); extern double __asin (→
    ↪double __x) __attribute__ ((__nothrow__ , __leaf__))→
    ↪;

extern double atan (double __x) __attribute__ ((→
    ↪__nothrow__ , __leaf__)); extern double __atan (→
    ↪double __x) __attribute__ ((__nothrow__ , __leaf__))→
    ↪;
...
struct exception

  {
    int type;
    char *name;
    double arg1;
    double arg2;
    double retval;
  };
...
int main()
{volatile int i=3;
 i=i+1;
}
```

gcc -S

```
    .file   "t.c"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushl   %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    subl    $16, %esp
    movl    $3, -4(%ebp)
    movl    -4(%ebp), %eax
    addl    $1, %eax
    movl    %eax, -4(%ebp)
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Debian →
        ↪4.9.1-4) 4.9.1"
    .section        .note.→
        ↪GNU-stack,"",→
        ↪@progbits
```

De l'utilité de connaître l'assembleur

```
int main()
{volatile unsigned short j=3,k;
 for (k=1;k<15;k++) {j+=k;}
}
```

```
.L6:
        add     2(r1), @r1
        add     #1lo(1), 2(r1)
        cmp     #1lo(15), 2(r1)
        jlo     .L6
```

```
int main()
{volatile unsigned short j=3,k;
 for (k=15;k>0;k--) {j+=k;}
}
```

```
.L6:
        add     2(r1), @r1
        add     #1lo(-1), 2(r1)
        jne     .L6
```

Mais un compilateur sait analyser le code : exemple de avr-gcc

```
88 e0           ldi     r24, 0x08       ; 8
for(i=0;i<8;++i) asm("nop"); // Back porch (wait)
00 00           nop
81 50           subi    r24, 0x01       ; 1
e9 f7           brne    .-6             ; 0x198 <__vector_17+0x24>
```

Comparaison de compilateurs

Dans tous les cas :

```
void delay (int length) {while (length >=0) length--;}
  // delay(5000)=1 ms
```

arm-thumb-elf-gcc (GCC) 4.0.0

```
196             delay:
197                     @ lr needed for prologue
198 0000 00E0           b       .L2
199             .L3:
200 0002 0138           sub     r0, r0, #1
201             .L2:
202 0004 0028           cmp     r0, #0
203 0006 FCDA           bge     .L3
204                     @ sp needed for prologue
205 0008 7047           bx      lr
207 000a 0000           .align  2
208                     .global jmf_putchar
209                     .code 16
210                     .thumb_func
```

Keil ARM Compiler V2.42

```
  47: void delay (int length) {       // delay(5000)=1 ms
00000000  B401  PUSH    {R0}
  48:     while (length >=0) length--;
00000002  E003  B       L_1  ; T=0x0000000C
00000004      L_3:
00000004  A800  ADD     R0,R13,#0x0
00000006  6801  LDR     R1,[R0,#0x0] ; length
00000008  3901  SUB     R1,#0x1
0000000A  6001  STR     R1,[R0,#0x0] ; length
0000000C      L_1:
0000000C  A800  ADD     R0,R13,#0x0
0000000E  6800  LDR     R0,[R0,#0x0] ; length
00000010  2800  CMP     R0,#0x0
00000012  DAF7  BGE     L_3  ; T=0x00000004
  49: }
00000014  B001  ADD     R13,#0x4
00000016  4770  BX      R14
00000018      ENDP ; 'delay?T'
```

Fonctionnalités égales mais temps d'éxécution différents

Multiplication

- AVR a une multiplication (signée ou non) en 2 cycles d'horloge
- décaler revient à multiplier/diviser par 2

```
int main()
{volatile char k=
    ↪5;
 k=k*10;
}
```

se compile en

```
ldi r24,lo8(5)
mov r25,r24
lsl r25
lsl r25
add r24,r25
lsl r24
```

lsl puis lsl est $\times 4$, add est $\times 5$, et lsl est $\times 10$.

```
int main()
{volatile char c1=
    ↪5,c2=10;
 c2=c1*10;
}
```

se compile en

```
ldi r24,lo8(5)
lsl r24
mov r25,r24
lsl r25
lsl r25
add r24,r25
```

lsl est $\times 2$, puis lsl lsl est $\times 8$, add est $8+2=10$.

```
int main()
{volatile char c1=
    ↪5,c2=10;
 c2=c1*c2;
}
```

se compile en

```
ldi r24,lo8(5)
ldi r25,lo8(10)
mul r24,r25
```

mul effectue la multiplication en 2 cycles d'horloge !

Exercice : comment multiplier par 15 ?