# Informatique embarquée 4/16

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

5 février 2018

# Exploitation d'une bibliothèque

1. dangers d'une bibliothèque
2. utilité d'une bibliothèque

Les ressources additionnelles ne se justifient que sur microcontrôleur haut de gamme ($>$ 16 bits)

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),
- ...

# Exploitation d'une bibliothèque – `newlib`

**Newlib** : implémentation de `libc` répondant à toutes les exigences d'un OS $\Rightarrow$ complet, mais complexe et lourd
L'utilisation de `printf` et du calcul flottant n'est pas à prendre à la légère :

| | | |
|---|---|---|
| Taille du fichier .hex intel [1] : | avec stdio | 80336 |
| | sans stdio | 4048 |

| | |
|---|---|
| sans stdio, avec une division flottante | 12950 |
| sans stdio, avec atan sur flottant | 17090 |
| avec stdio, avec une division flottante | 80397 |
| avec stdio, avec atan sur flottant | 80397 |

1. 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-isattyr.o): In function ‘_isatty_r’
newlib/libc/reent/isattyr.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[2].

- **_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

2. `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)
  {
#ifndef 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.

# Problème de la cohérence de la chaîne de compilation

- Les bibliothèques doivent suivre le même modèle de binaire que ceux générés par le compilateur (EABI)
- ... et être compilées avec les mêmes options (e.g. hard float v.s soft float).
- Recompiler ses outils pour garantir la cohérence : `https://github.com/jmfriedt/summon-arm-toolchain` qui se charge de configurer tous les outils de la même façon

```
TARGET=arm-none-eabi
PREFIX=${HOME}/sat
./configure --target=${TARGET} --prefix=${PREFIX} --enable-multilib ...
```

# 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 Atmega32U**2**

```
unsigned short interroge (unsigned short puissance, unsigned int freq,unsig
{ 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 Atmega32U**2**

```
unsigned short interroge(unsigned short puissance,unsigned int freq, \
  __attribute__((unused))unsigned int offset,__attribute__((unused))unsigne
{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 microcontroleur

```
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();
}
```

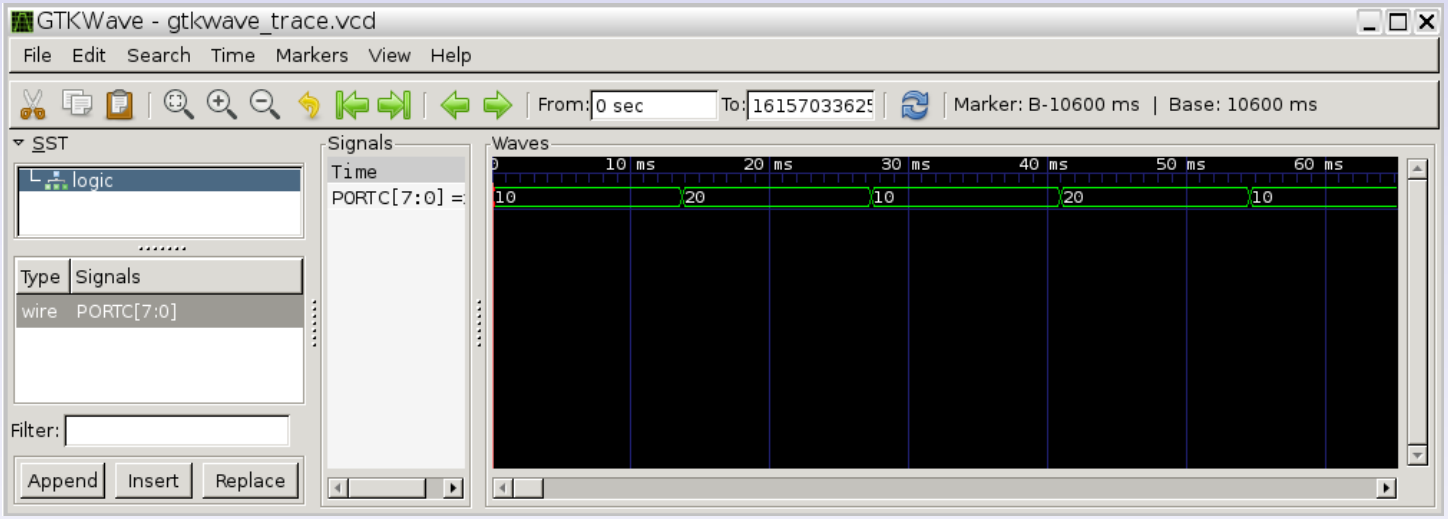
qui se compile avec l'option `-pg` pour générer à l'exécution `gmon.out` ...

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

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  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.

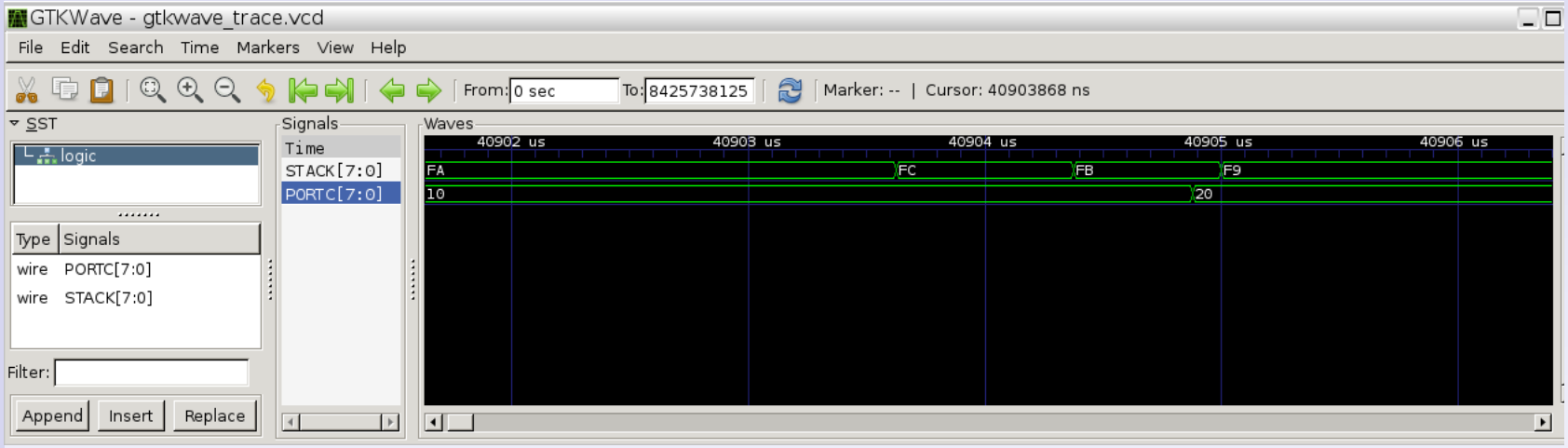
Préfixer son programme des déclarations de simulations :

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

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

# 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 Atmega32U**2**




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}+0x20
  {AVR_MCU_VCD_SYMBOL("STACKH"), .what = (void*)(0x5E), }, // stack=0x3{DE}+0x20
        };
```

# Virgule fixe/virgule flottante

- Représentation d'un nombre en virgule flottante :
  - mantisse/exposant (mantisse $\in [0, 1-1[$), similaire à la notation scientifique $0,1234 \times 10^3$
  - supporte une large gamme de valeurs, au détriment de la précision (codage de la position de la virgule)
  - exposant 8 bits-mantisse 23 bits (float), 11-52 (double) qui code donc $2^{\pm 1024} \simeq 10^{\pm 308}$ sur 15 décimales ($2^{52} \simeq 4 \times 10^{15}$)
  - $\Rightarrow$ arithmétique complexe, addition passe par la dénormalisation pour avoir le même exposant (donc des 0 en début de mantisse du nombre le plus petit), multiplication aisée
  - implémentation matérielle sur processeurs haut de gamme (FPU)
- Représentation en virgule fixe : homothétie pour se ramener à des calculs sur des entiers.
- on note Qm.n pour représenter la partie entière sur m bits et la partie fractionnaire sur n bits [3] (notation en complément à deux, donc inclut un bit de signe)

3. Maxim, Application Note 3722 : Developing FFT Applications with Low-Power Microcontrollers (2006)

# Opérations sur les flottants

- flottant = mantisse $\cdot 2^{exposant}$
- $\Rightarrow$ multiplication somme les exposants et multiplie les mantisses (problème de précision)
- $\Rightarrow$ sommer nécessite d'aligner les mantisses en ajustant les exposants
- en l'absence de FPU, opérations logicielles gourmandes en ressources

Exemple sur STM32F1 (pas de FPU) [4] :

| Opération | durée ($\mu$s) |
|---|---|
| ADC $\rightarrow$ entier | 1,73 |
| ADC $\rightarrow$ flottant | 3,36 |
| entier $\times 10^6/10^6$ | 1,08 |
| entier $(<< 20)(>> 20)$ | 0,83 |
| IIR à 2 coefficients flottants | 30 |
| IIR à 2 coefficients entiers | 3 |

4. données acquises par G. Matten, LMA, FEMTO-ST

# Cas de la fonction affine

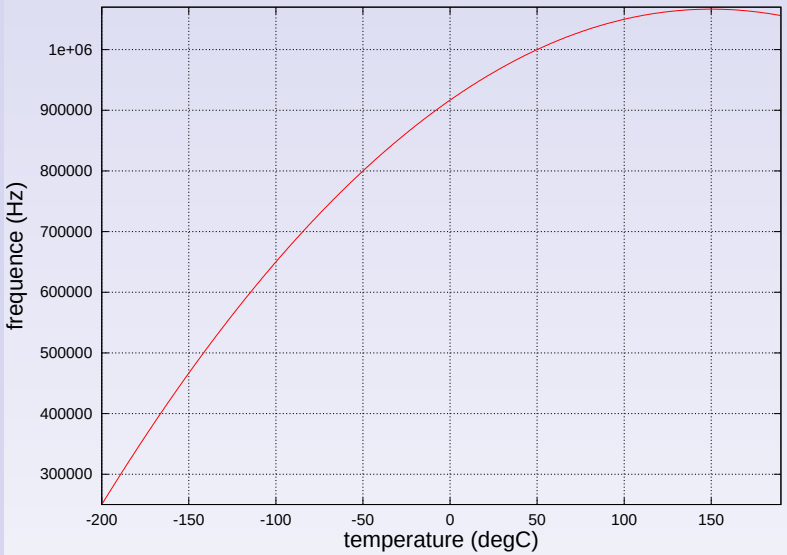
1. Un microcontrôleur manipule une donnée dans sa représentation : une fréquence de DDS est comprise entre 0 et $f_{CK}$, représentée par $[0..2^N - 1]$,
2. un humain veut connaître une fréqeunce en Hz

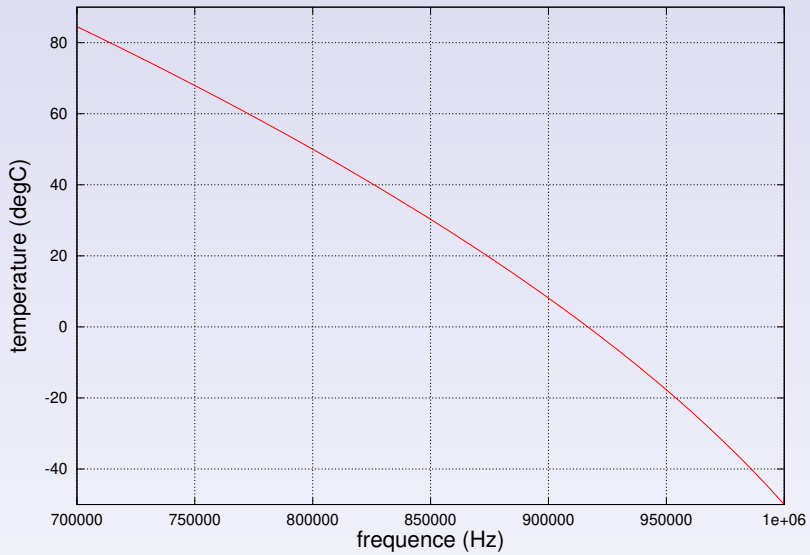
$$f_{Hz} = \frac{mot}{2^N} \cdot f_{CK}$$

3. application numérique, $f_{CK} = 70$ MHz et $N = 28$
$\Rightarrow 70 \cdot 10^6 / 2^{28} = 0.260770320892334$
4. GNU/Octave (Matlab) : `rats(0.260770320892334,5)=6/23` et `rats(0.260770320892334,8)=914/3505`
5. $mot \in [0..2^{28}] \Rightarrow mot \times 914$ sur 38 bits
6. mais si la bande de fréquence est connue (e.g. 8-10 MHz), bits de poids fort peuvent être éliminés
`(mot&0x003fffff)*914/3505` sera exacte, tandis que 0x00400000*914/3505 est précalculé (bits de poids forts connus)
7. si la fréquence est imprécise, bits de poids faible peuvent être éliminés : `(mot>>8)*914/3505` perd les 100 derniers Hz

# Exemple de capteur

- soit une mesure de fréquence $f$ issue d'un capteur oscillant, dont la fréquence de vibration change avec la température $T$
- sachant que $f \simeq 1$ MHz avec une précision de 10 Hz, on veut obtenir la température à 0,1 K près :




$$f = -6,6667 \times T^2 + 2000 \times T + 916670$$

$$T = -150 + \sqrt{160000 - 0,15 \times f}$$

**Exprimer cette loi d'étalonnage avec des coefficients entiers**

Rappel : la suite $x_{n+1} = \frac{1}{2}\left(x_n + \frac{y}{x_n}\right)$ converge vers $\sqrt{y}$

(méthode de Newton sur $x^2 - y = 0$ avec $x_{n+1} = x_n - f/f'$)

# Application des coefficients d'étalonnage

$$\boxed{T = A + \sqrt{B + C \cdot f}}$$

Hypothèses :

- $f \simeq 10^6$ (1 MHz)
- $f$ à 10 Hz près
- $A \simeq 100$
- $C \simeq 0,1$
- T à 0,1 K près

# Application des coefficients d'étalonnage

Exemple de calcul en virgule fixe compte tenu de la précision recherchée :

1. la loi liant fréquence et température est du type $f = \alpha T^2 + \beta T + \gamma \Rightarrow T = A \pm \sqrt{B + Cf}$
2. alors $(T-A)^2 = B + C \times f$ donc $2(T-A)dT - 2(T-A)dA = dB + dC \times f \Rightarrow |dT| = \left|\frac{dB}{2(T-A)}\right| + \left|\frac{dC \times f}{2(T-A)}\right| + |dA|$
3. on doit donc travailler avec chacun de ces termes inférieurs à 0,1.
4. Application numérique : par conception, $T - A \geq 100$ et $f \leq 10^6$ donc $dB \leq 10$, $dA \leq 0,1$ et $dC \leq 10^{-5}$
5. on va donc travailler sur $A \times 10$ et $C \times 10^5$ pour respecter la résolution recherchée
6. cependant, $C \simeq 0,1$ donc $C \times 10^5 \times f \simeq 10^{10}$ qui nécessite 34 bits,
7. mais $f$ à 10 Hz donc on peut travailler sur $C \times 10^5 \times (f/10) \simeq 10^9$ qui ne nécessite que 30 bits

**Conclusion : $10 \cdot A$, $10^4 \cdot B$ et $10^5 \cdot C$ pré-calculés**

$$
\begin{aligned}
10 \times T = \quad & 10 \times A + 10\sqrt{B + C \cdot f} \\
& 10 \times A + 10\sqrt{B \cdot 10^4 + C \cdot f \cdot 10^4}/100 \\
& 10 \times A + \sqrt{B \times 10^4 + (10^5 C)\,(f/10)}/10
\end{aligned}
$$

# Mise en pratique

Nous quittons l'AVR qui a choisi de développer sa propre libc, pour passer au STM32

- processeur 32 bits capable d'embarquer des bibliothèques (puis environnements exécutifs)
- `summon-arm-toolchain` fournit un script pour générer un ensemble "cohérent" d'outils – explication de sa compilation à `http://jmfriedt.free.fr/summon_arm.pdf`
- le processeur se décline en une multitude de classes : *linker* script approprié.
- Émulateur pour STM32 : `qemu` pour ARM complété de l'émulation de périphériques – `https://github.com/beckus/qemu_stm32`.

Les divers outils de compilation : `Makefile`, `configure`, `cmake`
Les divers outils d'archivage : `svn`, `git`, `hg` (mercurial)