Électronique numérique

J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

22 novembre 2020

Rappels L3

- Analyser la datasheet pour comprendre le lien entre matériel (périphériques tels que GPIO, timer, USART, WDT ...) et logiciel (adresses des registres)
- `gcc` (GNU Compiler Collection) comme outil générique pour aborder tout microcontrôleur/processeur
- automatiser la séquence de compilation par `make` (mais aussi `configure` de `autoconf`, `cmake` et maintenant `ninja`) pour définir les dépendances
- Séquence de développement :
  1. Éditer les codes sources
  2. Compiler
  3. Flasher
  4. Exécuter/déverminer (*debug*)

Un outil pour effectuer au mieux chaque étape.

Comment connaître l'adresse d'un périphérique ...

Liste des registres dans la datasheet du microcontrôleur (Atmega32U4)
$\Rightarrow$ utilisation des masques pour définir un bit individuel

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0
0x14 (0x34)	Reserved	-	-	-	-	-	-	-	-
0x13 (0x33)	Reserved	-	-	-	-	-	-	-	-
0x12 (0x32)	Reserved	-	-	-	-	-	-	-	-
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	-	-	PORTF1	PORTF0
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	-	-	DDF1	DDF0
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	-	-	PINF1	PINF0
0x0E (0x2E)	PORTE	-	PORTE6	-	-	-	PORTE2	-	-
0x0D (0x2D)	DDRE	-	DDE6	-	-	-	DDE2	-	-
0x0C (0x2C)	PINE	-	PINE6	-	-	-	PINE2	-	-
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	PORTC7	PORTC6	-	-	-	-	-	-
0x07 (0x27)	DDRC	DDC7	DDC6	-	-	-	-	-	-
0x06 (0x26)	PINC	PINC7	PINC6	-	-	-	-	-	-
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	Reserved	-	-	-	-	-	-	-	-
0x01 (0x21)	Reserved	-	-	-	-	-	-	-	-
0x00 (0x20)	Reserved	-	-	-	-	-	-	-	-

Extrait de datasheet Atmega 32U4

Lien logiciel-matériel

Syntaxe avr-libc : `PORTB=0x42;`

- `/usr/lib/avr/include/avr/io.h` :

```
#elif defined (__AVR_ATmega32U2__)
#  include <avr/iom32u2.h>
```

- `/usr/lib/avr/include/avr/iom32u2.h`

```
#define PORTB _SFR_IO8(0x05)
```

- `/usr/lib/avr/include/avr/sfr_defs.h`

```
#define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + __SFR_OFFSET)
#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
```

Bus d'adresses (qui), bus de données (quoi), but de contrôle (comment)

`volatile` : interdit de faire des hypothèses sur les variables pour l'optimisation du compilateur

`*(type*)@=val;`

Exemple d'application des pointeurs :

```
short s;char *t; t=&s; printf("%hhx %hhx\n",t[0],t[1]);
```

GCC

- Ne pas s'handicaper avec un compilateur propriétaire compatible avec une unique plateforme matérielle : gcc [1]
- Maîtriser une large gamme de processeurs pour répondre à tous les besoins
  - périphériques dédiés autour d'un cœur de processeur générique (8051, ARM)
  - cœur répondant aux besoins de puissance de calcul
  - vitesse la plus faible possible pour réduire la consommation

aarch64	csky	lm32	mn10300	pru	stormy16
alpha	epiphany	m32c	moxie	**riscv**	tilegx
arc	fr30	m32r	**msp430**	rl78	tilepro
arm	frv	**m68k**	nds32	rs6000	v850
avr	gcn	mcore	nios2	rx	vax
bfin	**h8300**	mep	nvptx	s390	visium
c6x	**i386**	microblaze	pa	sh	xtensa
cr16	ia64	**mips**	pdp11	**sparc**	
cris	iq2000	mmix	powerpcspe	spu	

1. https://gcc.gnu.org/backends.html

GCC

Options de compilation :

- `-o` : nom du fichier de sortie (les extensions n'ont pas d'importance sous Unix)
- `-O` : niveau d'optimisation du code (0, 1, 2, s)
- `-I` : liste des répertoires contenant des fichiers d'entête (`.h`). Répéter autant de fois qu'il y a de répertoire(s). Uniquement utile lors du passage code source `.c` $\rightarrow$ objet `.o`
- `-L` : liste des répertoires contenant des bibliothèques (`.a`). Répéter autant de fois qu'il y a de répertoire(s). Uniquement utile lors du passage des objets `.o` $\rightarrow$ exécutable

Outils de développement

Sans système d'exploitation (*bare-metal*)

- `gcc`+`binutils`+bibliothèques (`newlib`)
  Capacité à recompiler sa chaîne de compilation sur toute plateforme

  Pour STM32 (sans OS) : `arm-none-eabi-gcc`
  (pour AVR : `avr-gcc`)

- un programmeur pour communiquer avec le bootloader (`avrdude`, `st-util`, OpenOCD ...), simple à réimplémenter si nécessaire
- outils périphériques (`gdb`)

Langage C fournit le meilleur compromis entre abstraction (variables, boucles, branchements conditionnels) et accès au matériel (pointeurs).

Quelques subtilités sur l'embarqué : `volatile`, `*(type*)addr=valeur;`

MAIS il faut tout comprendre (pas de pilote, bibliothèques à adapter au matériel)

Makefile

- Séquence de règles pour atteindre l'objectif `all`
- `make` compare les dates entre fichier résultant et dépendances de la règle.
- Si les dépendances sont plus récentes que le résultat, la règle est exécutée, ...
- ... sinon passe à la régle dont le résultat est une dépendance.

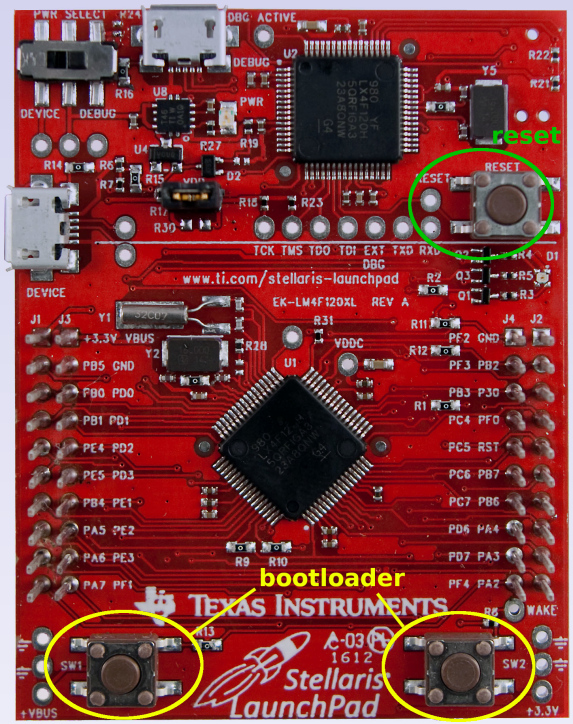
Générateurs de Makefile

Comment compiler une archive téléchargée (e.g. `github`) ?

- si contient `Makefile` ou `makefile` : `make`
- si contient `CMakeLists.txt` :
  `mkdir build && cd build && cmake ../` va créer `Makefile`
- si contient[2] `configure` : `./configure.sh` pour exploiter `Makefile.in` en utilisant `autotools`. Une fois le `Makefile` créé : `make`
- `autotools` exploite souvent `pkg-config` pour trouver les options `-I` et `-L` d'une bibliothèque.

2. `github.com/ngspice` contient `autogen.sh` qui génère `configure` !

flasher/exécuter/debugger



- La majorité des microcontrôleurs actuels ont un *bootloader* activé par une broche (load/execute)
- Si ce bootloader est inexistant ou endommagé : JTAG (*Joint Test Action Group*), interface standardisée pour communiquer avec des circuits intégrés
- Une fois la communication établie, possibilité de *tracer* le programme avec un *debugger* : gdb

- Donne la position du pointeur d'exécution PC, position de la pile SP, état de la pile et des fonctions qui ont été appelées
- Afficher des noms de fonctions au lieu de leur adresse : compiler avec -g (*debug symbols*)
- Émulateur peut aider en affichant messages internes au micro-contrôleur (inaccessibles sinon) ou en augmentant artificiellement les ressources (dépassement mémoire) : `simavr` pour Atmega(32U4).

Binaire, hexadécimal et masques

Programmer une fréquence dans un synthétiseur numérique de fréquence

WRITING TO A FREQUENCY REGISTER

When writing to a frequency register, Bit DB15 and Bit DB14 give the address of the frequency register.

Table 10. Frequency Register Bits

DB15	DB14	DB13 . . . DB0
0	1	14 FREQ0 REG BITS
1	0	14 FREQ1 REG BITS

If the user wants to alter the entire contents of a frequency register, two consecutive writes to the same address must be performed because the frequency registers are 28 bits wide. The first write contains the 14 LSBs, and the second write contains the 14 MSBs. For this mode of operation, Control Bit B28 (DB13) should be set to 1. An example of a 28-bit write is shown in Table 11.

Note however that continuous writes to the same frequency register are not recommended. This results in intermediate updates during the writes. If a frequency sweep, or something similar, is required, it is recommended that users alternate between the two frequency registers.

extrait de la datasheet AD9834

DDS : composant numérique

$$\boxed{f_o = \frac{W}{2^N} \cdot f_{ck}} \quad (W \in [0..2^N - 1])$$

ici, $f_{ck} = 70$ MHz et $N = 28$

un ordinateur manipule des données sur 32 bits
le port de communication supporte des transactions sur 16 bits

Binaire, hexadécimal et masques

Programmer une fréquence dans un synthétiseur numérique de fréquence

DDS : composant numérique

WRITING TO A FREQUENCY REGISTER

When writing to a frequency register, Bit DB15 and Bit DB14 give the address of the frequency register.

Table 10. Frequency Register Bits

DB15	DB14	DB13 . . . DB0
0	1	14 FREQ0 REG BITS
1	0	14 FREQ1 REG BITS

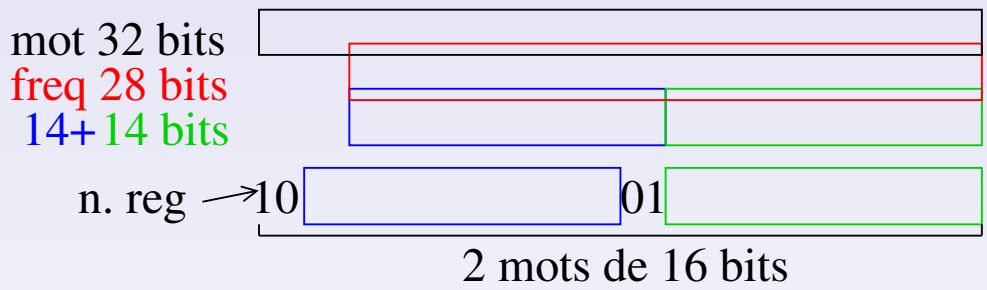
If the user wants to alter the entire contents of a frequency register, two consecutive writes to the same address must be performed because the frequency registers are 28 bits wide. The first write contains the 14 LSBs, and the second write contains the 14 MSBs. For this mode of operation, Control Bit B28 (DB13) should be set to 1. An example of a 28-bit write is shown in Table 11.

Note however that continuous writes to the same frequency register are not recommended. This results in intermediate updates during the writes. If a frequency sweep, or something similar, is required, it is recommended that users alternate between the two frequency registers.

extrait de la datasheet AD9834

$$f_o = \frac{W}{2^N} \cdot f_{ck} \quad (W \in [0..2^N - 1])$$

ici, $f_{ck} = 70$ MHz et $N = 28$



Exercice : proposer les deux lignes de C qui découpent le mot de 32 bits en 2 mots de 16 bits prêts au transfert au DDS.

Binaire, hexadécimal et masques

Programmer une fréquence dans un synthétiseur numérique de fréquence

DDS : composant numérique

WRITING TO A FREQUENCY REGISTER

When writing to a frequency register, Bit DB15 and Bit DB14 give the address of the frequency register.

Table 10. Frequency Register Bits

DB15	DB14	DB13 . . . DB0
0	1	14 FREQ0 REG BITS
1	0	14 FREQ1 REG BITS

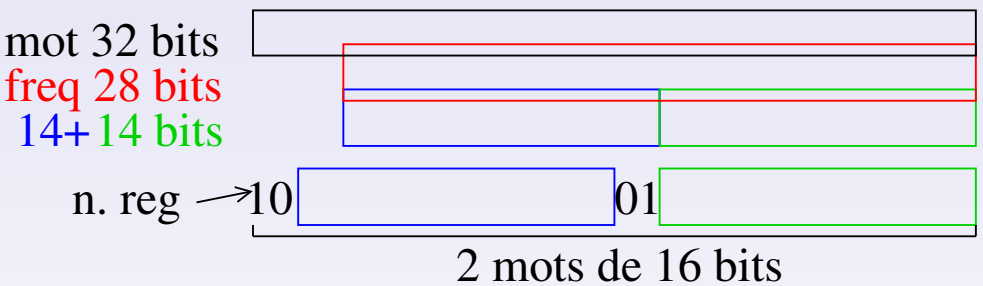
If the user wants to alter the entire contents of a frequency register, two consecutive writes to the same address must be performed because the frequency registers are 28 bits wide. The first write contains the 14 LSBs, and the second write contains the 14 MSBs. For this mode of operation, Control Bit B28 (DB13) should be set to 1. An example of a 28-bit write is shown in Table 11.

Note however that continuous writes to the same frequency register are not recommended. This results in intermediate updates during the writes. If a frequency sweep, or something similar, is required, it is recommended that users alternate between the two frequency registers.

extrait de la datasheet AD9834

$$f_o = \frac{W}{2^N} \cdot f_{ck} \quad (W \in [0..2^N - 1])$$

ici, $f_{ck} = 70$ MHz et $N = 28$



WI=((freq&0x3fff)|0x4000)

Wh=((freq&0x0fffc000)>>14)|0x8000

Binaire, hexadécimal et masques

Programmer une fréquence dans un synthétiseur numérique de fréquence

DDS : composant numérique

WRITING TO A FREQUENCY REGISTER

When writing to a frequency register, Bit DB15 and Bit DB14 give the address of the frequency register.

Table 10. Frequency Register Bits

DB15	DB14	DB13 . . . DB0
0	1	14 FREQ0 REG BITS
1	0	14 FREQ1 REG BITS

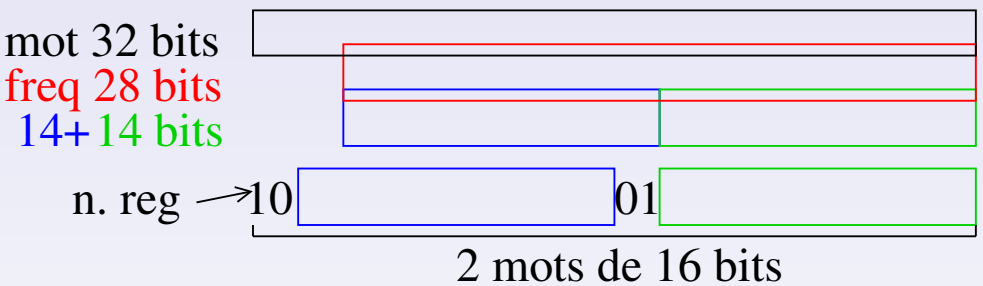
If the user wants to alter the entire contents of a frequency register, two consecutive writes to the same address must be performed because the frequency registers are 28 bits wide. The first write contains the 14 LSBs, and the second write contains the 14 MSBs. For this mode of operation, Control Bit B28 (DB13) should be set to 1. An example of a 28-bit write is shown in Table 11.

Note however that continuous writes to the same frequency register are not recommended. This results in intermediate updates during the writes. If a frequency sweep, or something similar, is required, it is recommended that users alternate between the two frequency registers.

extrait de la datasheet AD9834

$$f_o = \frac{W}{2^N} \cdot f_{ck} \quad (W \in [0..2^N - 1])$$

ici, $f_{ck} = 70$ MHz et $N = 28$



WI=((freq&0x3fff)|0x4000)

Wh=(((freq)>>14)&0x3fff)|0x8000

Prérequis

1. Programmation en C (structures de données, gestion du flux d'exécution), fichier d'entête (`.h`) et de code (`.c`)
2. Compilation séparée par `gcc`, automatisation par `Makefile`
3. Accès aux ressources matérielles par pointeur (`*(type*)@=val`)
4. Lecture de *datasheet* et identification des adresses de registres pour accéder aux diverses fonctions des périphériques
5. Fonctionnement des périphériques classiques (GPIO, *timer*, ADC) et interruptions matérielles
6. Représentations hexadécimale/décimale/binaire, opérations logiques et masques
7. Organisation de l'arborescence sous Unix, commandes de base (`ls`, `mv`, `rm`, `mkdir`, `cd`) : nous travaillerons exclusivement sous GNU/Linux.

Ressources : ressources L3 à `jmfriedt.free.fr/index.html#l3ep`, codes sources à `github.com/jmfriedt/l3ep`
Un **ordinateur personnel sous GNU/Linux** permettra de reproduire les séances chez soi

Exercice

Rappel de quelques commandes Unix :

- `ls` pour la liste des fichiers (`ls /usr/lib/avr/include`)
- `touch` pour créer un fichier, `rm` pour l'effacer (`touch toto`)
- `mkdir` pour créer un répetoire, `cd` pour y entrer
- `grep` : rechercher un motif dans des fichiers (`grep -r GPIO /usr/lib/avr/`)

Objectif : faire clignoter les LEDs verte et jaune de l'Olimexino32U4

- Squelette de programme :

```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
int main(void) {while (1) {}}
```

- Compilateur : `avr-gcc -mmcu=atmega32u4 source.c -o jmf`
- Programmer le microcontrôleur : `avrdude`

```
avrdude -c avr109 -b57600 -D -p atmega32u4 -P /dev/ttyACM0 -e -U flash:w:jmf
```

- Schéma du circuit : `www.olimex.com/Products/Duino/AVR/OLIMEXINO-32U4/open-source-hardware`

Objectif : émettre un message sur le port USB émulant un port série (bibliothèque LUFA)

Correction

```
#include <avr/io.h>         //E/S ex PORTB
#define F_CPU 16000000UL
#include <util/delay.h>

//  D0  D1  D2  D3  D4  D5  D6  D7  D8  D9
// PD2 PD3 PD1 PD0 PD4 PC6 PD7 PE6 PB4 PB5
int port[10]={3,3,3,3,3,2,3,4,1,1};
int pin [10]={2,3,1,0,4,6,7,6,4,5};

void allume(int pin_index)
{*(unsigned char*)(0x22+3*port[pin_index])|=(1<<pin[pin_index]);}

void eteint(int pin_index)
{*(unsigned char*)(0x22+3*port[pin_index])&=(~(1<<pin[pin_index]));}

int main(void){
  int v=9;
  char direction=1;
  DDRB |=0x7f; DDRC |=0xff; DDRD |=0xff; DDRE |=0xff; // tout en sortie
  PORTB=0x80;  PORTC=0x00;  PORTD=0x00;  PORTE=0x00;  // une LED allumee
  v=0;

  while (1){
    eteint(v);
    // if (v>0) v--;else v=9;     // unidirectionnel
    if (direction==1)             // K2000
       if (v>0) v--;else direction=1-direction;
    else
       if (v<9) v++;else direction=1-direction;
    allume(v);
    _delay_ms(30);
  }
  return 0;
}
```