L'environnement Arduino est-il approprié pour enseigner l'électronique embarquée ?

Émile Carry, Jean-Michel Friedt

***Résumé*—Arduino est un environnement de travail intégré proposant un accès aisé à la programmation de systèmes numériques embarqués, en particulier en fournissant des bibliothèques libres facilitant l'accès aux ressources matérielles. Nous argumenterons que bien que l'utilisation d'un tel environnement pour attirer l'étudiant vers la programmation de systèmes numériques est certes favorable, l'enseignement d'un cursus d'électronique embarquée visant à fournir une aisance en électronique numérique ne saurait s'en contenter. Son utilisation cache les fondements de la communication avec les périphériques et dégrade significativement les performances du matriel et n'encourage pas à respecter un des paradigmes du développement de systèmes embarqués, à savoir l'optimisation des ressources pour utiliser la plateforme la plus petite possible répondant aux besoins de l'application visée.**

***Index Terms*—microcontrôleurs, systèmes embarqués, électronique numérique programmable, Arduino, langage C, gcc**

I. Introduction

L'ENSEIGNEMENT de l'électronique numérique dans le contexte des systèmes embarqués peut s'aborder sur différents niveaux d'abstraction selon l'auditoire visé : programmation bas niveau en assembleur ou C pour les électroniciens plus sensibles aux aspects matériels, environnements exécutifs pour organiser son projet en tâches pour des projets plus ambitieux, systèmes d'exploitation pour les informaticiens désireux d'accéder aux ressources matérielles au travers des abstractions que sont les pilotes, avec un superviseur (l'ordonnanceur) chargé de séquencer les tâches et garantir l'intégrité des ressources en arbitrant leurs accès. Une maîtrise de la chaîne de développement d'un système embarqué nécessite de faire le lien entre logiciel et matériel, et de fait comprendre à la fois des concepts d'électronique et de conception de logiciel. Alors que les formations d'électronique et d'informatique s'efforcent d'aborder l'un *ou* l'autre des aspects, le développement de systèmes embarqués se situe à l'interface et donne l'opportunité de maîtriser les *deux* aspects et les points qui les lient.

Un récent sondage de l'IEEE place le C comme le langage le plus utilisé, tous domaines confondus excluant les interfaces web [1]. Ce choix se justifie probablement par la souplesse du C, qui bien que plus abstrait que l'assembleur (13^e position) laisse l'accès, au travers des pointeurs, aux bus de communication entre le processeur et les périphériques, tout en permettant d'exprimer simplement des algorithmes plus complexes. Ainsi, un fort niveau d'optimisation peut être atteint. Plus surprenant, Arduino apparaît dans cette liste, en 11^e position de ce sondage auprès des membres de l'Institute of *Electrical and Electronics Engineers*. Ce choix, par les membres de cette association tournée vers l'électronique, semble en contradiction avec le choix de conception d'Arduino qui s'annonce [1] comme *an easy tool for fast prototyping, aimed at students without a background in electronics and programming.* Arduino apparaît ainsi comme un excellent outil pédagogique pour des utilisateurs dont la compréhension détaillée du mécanisme de fonctionnement de leur outil n'est pas une priorité : physiciens [2], [3], [4], chimistes [6], ou artistes [7], voir pour introduire un public plus jeune aux concepts de la programmation [8]. Nous ne cautionnons cependant pas l'extension du public visé aux programmeurs [9] et professionnels [7], selon l'argumentaire qui suivra. On notera d'ailleurs que ces deux dernières populations d'utilisateurs n'ont été ajoutées que plus tard dans le document introductif à Arduino, qui ne citait initialement que *intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments* [10].

Dans le contexte de notre enseignement en licence 3 et master d'électronique, automatique et microsystèmes, et dernière année d'école d'ingénieur, nous désirons dans ce document comparer les performances de cette bibliothèque qui vise à cacher au développeur les fonctionnalités du C sous-jacent, et l'impact sur les performances du code. Nous défendrons notre choix d'enseigner, à un auditoire qui a fait le choix de l'électronique comme sujet d'activité professionnelle principale, l'électronique embarquée programmable par le C en s'appuyant sur le compilateur `gcc` par soucis de liberté (accès des étudiants à leur outils de travail pour en comprendre le fonctionnement) et de portabilité. En effet, `gcc` supporte toutes les principales architectures de processeurs [11], si nous négligeons les plus petits PIC de Microchip.

Ce document résume trois expériences issues de trois années d'enseignements sur plateforme matérielle visant les utilisateurs Arduino – l'Olimexino 32U4 – et le retour d'expérience d'étudiants d'une école qualifiée d'ingénieurs dont l'électronique n'est pas le cœur de métier mais prétend être partie intégrante de leur option concernant la gestion d'énergie et les transports. Dans un premier temps nous verrons que les performances médiocres de la bibliothèque poussent l'utilisateur vers une plateforme de développement plus rapide et plus puissante au lieu de remettre en cause la qualité de son code et de la bibliothèque associée –

É. Carry et J.-M. Friedt sont maîtres de conférence à l'Université de Franche-Comté à Besançon. Leur activité de recherche au département temps-fréquence de FEMTO-ST inclut une partie significative d'instrumentation et de logiciel embarqué sur cibles à faibles ressources. E-mail : {emile.carry, jmfriedt}@femto-st.fr.

1. www.arduino.cc/en/Guide/Introduction

fuite en avant à laquelle l'informatique grand-public nous a habitué. Dans un deuxième temps nous verrons un exemple d'hypothèse implicite de la bibliothèque – le décalage de l'adresse I2C du périphérique et l'ajout du bit de direction de la transaction – qui nuit à la compréhension du fonctionnement de l'USART et son utilisation plus générale qu'avec Arduino. Enfin de façon plus globale, nous verrons que l'application d'une bibliothèque n'implémentant que des fonctions simples et cantonnant l'utilisateur au niveau de la programmation en C nuit à une ouverture vers des environnements de développement plus abstraits permettant de maîtriser la puissance de calcul de plateformes multi-cœurs (architectures SMP) avec une quantité de mémoire difficilement maîtrisable en se limitant au C, cible pourtant visée par Arduino lorsque son portage sur Intel Edison [2] ou Galileo [3] sont annoncés.

II. Arduino v.s C

Un programme C brut (*baremetal*) se réduit toujours à une phase d'initialisation des périphériques et de déclaration des variables, suivie d'une boucle infinie dans laquelle une série de tâches est exécutée séquentiellement, selon une machine à états dont le comportement est éventuellement modifié par les interruptions [4]. Arduino reprend évidemment ce schéma, en nommant la phase d'initialisation `setup()` et la boucle infinie `loop()`. L'architecture logicielle n'a donc pas été modifiée, comme nous pourrions le trouver avec des concepts plus originaux tels que ceux proposés par TinyOS [12]. La boucle infinie `loop()` peut contenir des fonctions C, mais Arduino s'efforce de proposer des bibliothèques implémentant l'accès à la majorité des périphériques. Non seulement le développeur n'est pas encouragé à comprendre le fonctionnement sous jacent de ces périphériques, mais le soucis d'homogénéiser la nomenclature des diverses plateformes supportant Arduino introduit des complexités telles que celles rencontrées sur la sérigraphie de la carte Olimexino32U4 (Fig. 1), où le port D2 du microcontrôleur Atmel Atmega32U4 (PD2) est sérigraphié D0 pour respecter la convention Arduino, tandis que le port D0 (PD0) est sérigraphié D3, de quoi perturber plus d'un développeur novice qui tente de comprendre la distribution des ports d'entrée-sortie généralistes (GPIO) de son processeur.

L'argumentaire de portabilité du code Arduino par rapport à C est facilement balayé une fois le fonctionnement de l'éditeur de lien maîtrisé. Un compilateur C a besoin d'une unique information pour compiler un programme pour une nouvelle cible basée sur un cœur de processeur connu : la carte de la mémoire entourant le processeur, information fournie par le *linker script* (option -T de `ld`). Ainsi, porter une application fonctionnant sur une famille de processeurs vers une famille proche est fort simple, puisqu'une modification du script `.ld` [5]

2. www.arduino.cc/en/ArduinoCertified/IntelEdison
3. www.arduino.cc/en/ArduinoCertified/IntelGalileo
4. Exemples de TP introductifs en licence 3 : jmfriedt.free.fr/TP_Atmega32U4_GPIO puis jmfriedt.free.fr/TP_Atmega32U4_interrupt.pdf s'efforçant de pallier aux déficiences discutées dans le texte
5. voir par exemple github.com/jmfriedt/tp_freertos/blob/master/ld/stm32f4-discovery.ld par rapport à github.com/jmfriedt/tp_freertos/blob/master/ld/stm32vl-discovery.ld pour la compilation du même code sur STM32F4 ou F1 respectivement

FIGURE 1. Extrait du schéma de la carte Olimexino32U4 obtenu à www.olimex.com/Products/Duino/AVR/OLIMEXINO-32U4/resources/OLIMEXINO-32U4-SCHEMATIC.pdf .

suffit à informer le compilateur de l'emplacement et la taille de la RAM et de la mémoire non-volatile [6].

III. Performances des GPIO

Un programme trivial Arduino est de la forme :

```
// Blinking LED1 for OLIMEXINO-32u4
// general purpose LED at ARDUINO pins 7
int led1 = 7;

void setup() {pinMode(led1, OUTPUT);}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led1, HIGH);
  digitalWrite(led1, LOW);
}
```

Ce programme a été proposé par un étudiant désireux de générer un signal périodique rapide de façon logicielle (une mauvaise idée dès le départ puisque la temporisation logicielle n'est pas portable, le cadencement par timer aurait au contraire été reproductible puisque s'appuyant sur une couche matérielle de génération des signaux), avec le résultat d'une fréquence de répétition qui se limitait à moins de 100 kHz (Fig. 2, haut). Son paliatif a été de demander si la fréquence de cadencement de 16 MHz du microcontrôleur était insuffisante, et si le passage à un processeur plus rapide, par exemple un STM32 cadencé à plus de 150 MHz, pourrait convenir pour fournir un tel signal avec fréquence supérieure à 100 kHz, l'ordre du MHz étant visé. Évidemment, une architecture Harvard telle que celle qui dirige la conception de l'Atmega32U4 propose une exécution de la majorité des instructions, à l'exception des branchements qui nécessitent de vider le *pipeline*, en un cycle d'horloge, permettant ainsi d'atteindre les performances visées. La période excessivement longue du signal périodique obtenu était à attribuer aux multiples tests inutiles proposés par Arduino lors de la commutation de l'état d'un GPIO. Modifier les appels à la bibliothèque par un appel en C à l'adresse des registres manipulant l'état des ports permet de facilement respecter l'objectif proposé par l'étudiant (Fig. 2, bas),

6. au notera qu'a contrario, oublier de modifier ces paramètres est source de plantage rapide du programme puisque `gcc` place la pile au sommet de la RAM, dont une taille supérieure dans le *linker script* par rapport à la RAM physique induira un positionnement en zone inexistante.

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

int main(void){
 DDRE |=1<<PORTE6;
 PORTE &= ~(1<<PORTE6);

 while (1){
   PORTE=0x00;
   PORTE=(1<<PORTE6);
 }
 return 0;
}
```

en accord avec l'analyse du code assembleur (`avr-gcc -O2 -S`) associé :

```
main:
   sbi 0xd,6
   cbi 0xe,6
   ldi r24,lo8(64)
.L2:
   out 0xe,__zero_reg__
   out 0xe,r24
   rjmp .L2
```

La forme asymétrique du signal résultant est dûe au saut relatif qui augmente la durée d'un des états de la broche. Remplacer l'affectation par une commutation par OU exclusif rend le signal symétrique, au détriment de la fréquence puisque le saut relatif vient maintenant s'intercaler entre chaque transition de la broche, tel que illusté ci-dessous :

```
; avr-gcc -mmcu=atmega32u4 -nostartfiles blink_xor.s
   sbi 0xd,6
   cbi 0xe,6
   ldi r25,lo8(64)
.L2:
   eor r24,r25
   out 0xe,r24
   rjmp .L2
```

Seule l'analyse du code assembleur permet de prédire la durée d'exécution du code : avec un saut relatif de durée fixe égale à deux cycles d'horloge et toutes les instructions arithmétiques ou logiques exécutées en un cycle d'horloge, le dernier exemple nécessite 4 cycles d'horloge entre chaque commutation de la broche, soit une fréquence du signal de sortie de 16/8=2 MHz, tel qu'observé sur la trace à l'oscilloscope (Fig. 2, bas). La solution d'affectation se traduit par un état pendant un cycle d'horloge – 62 ns à 16 MHz – et 3 cycles d'horloge dans l'autre état à cause du saut relatif, soit une fréquence de 16/4=4 MHz (Fig. 2, milieu).

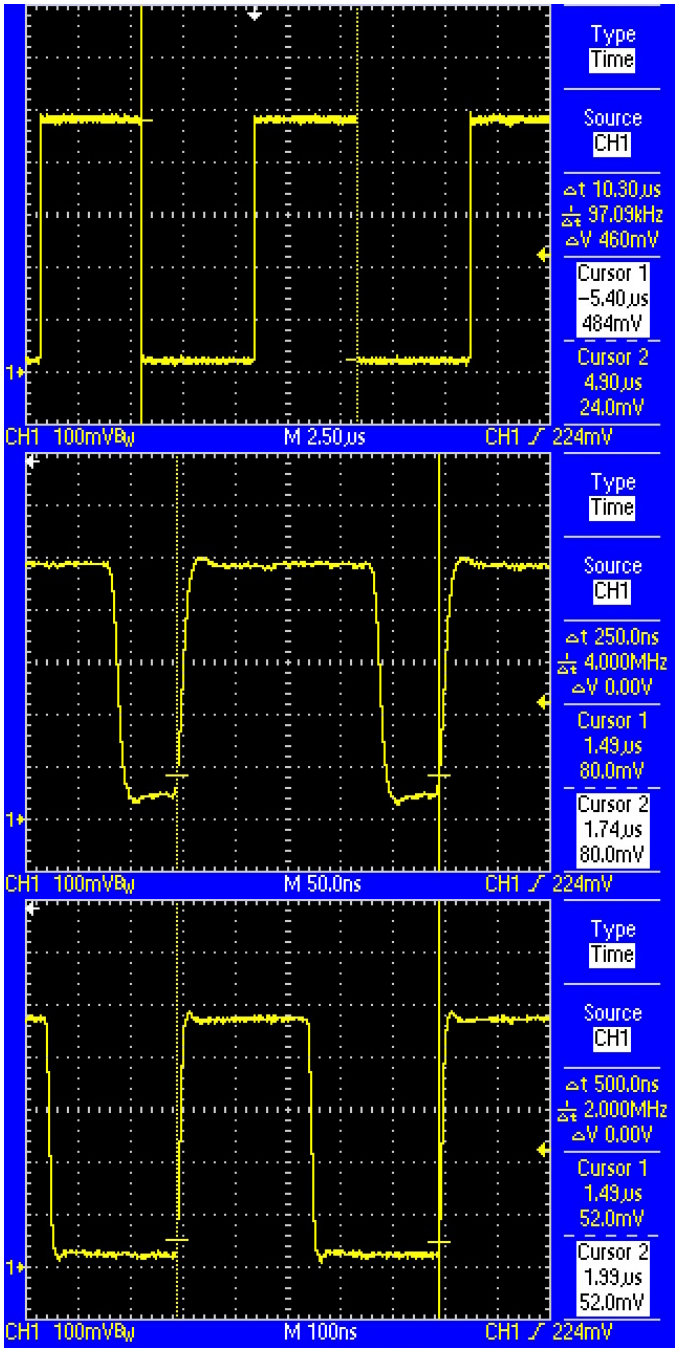


FIGURE 2. Captures d'écran d'oscilloscope représentant le signal créneau généré de façon logicielle : nous reprenons l'intervalle de temps entre les curseurs dans la légende qui suit. De haut en bas : la solution Arduino avec affectation de l'état des broches ($\Delta t = 10,3\ \mu s$) ; la version C avec affectation ($\Delta t = 0,25\ \mu s$) ; la version assembleur avec un XOR ($\Delta t = 0,5\ \mu s$).

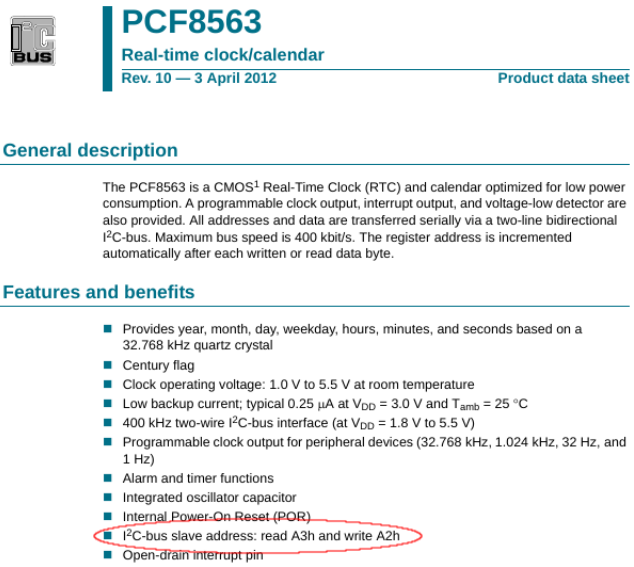
IV. ACCÈS AU BUS I2C

Le deuxième problème a été rencontré lors de l'étude du bus I2C pour accéder aux ressources d'un capteur communiquant par ce protocole – une tâche courante pour tout développeur de système embarqué qui se doit de maîtriser les diverses implémentations de protocoles synchrones et asynchrones et leurs déclinaisons. Arduino propose une implémentation, avec sa bibliothèque `Wire`, du protocole I2C. Nombreux sont les étudiants qui s'inspirent des codes sources issus d'Arduino dans leur découverte du fonctionnement du microcontrôleur. Malheureusement ici encore, cacher le fonctionnement du bus et les paramètres qui y sont transmis, en particulier pour adresser un périphérique, amène à des déboires qu'un novice aura du mal à résoudre.

Le cas particulier que nous avons rencontré porte sur l'horloge temps-réel PCF8563 dont nous nous servons pour illustrer le fonctionnement du bus I2C[7]. La documentation technique (Fig. 3) explicite les adresses permettant d'accéder au périphérique, 0xA2 ou 0xA3 puisque le bit de poids faible

7. jmfriedt.free.fr/TP_Atmega32U4_SPI.pdf

indique, sur bus I2C, la direction de la transaction.



PCF8563

Real-time clock/calendar

Rev. 10 — 3 April 2012 Product data sheet

1. General description

The PCF8563 is a CMOS[1] Real-Time Clock (RTC) and calendar optimized for low power consumption. A programmable clock output, interrupt output, and voltage-low detector are also provided. All addresses and data are transferred serially via a two-line bidirectional I²C-bus. Maximum bus speed is 400 kbit/s. The register address is incremented automatically after each written or read data byte.

2. Features and benefits

- Provides year, month, day, weekday, hours, minutes, and seconds based on a 32.768 kHz quartz crystal
- Century flag
- Clock operating voltage: 1.0 V to 5.5 V at room temperature
- Low backup current; typical 0.25 μA at V_{DD} = 3.0 V and T_{amb} = 25 °C
- 400 kHz two-wire I²C-bus interface (at V_{DD} = 1.8 V to 5.5 V)
- Programmable clock output for peripheral devices (32.768 kHz, 1.024 kHz, 32 Hz, and 1 Hz)
- Alarm and timer functions
- Integrated oscillator capacitor
- Internal Power-On Reset (POR)
- I²C-bus slave address: read A3h and write A2h
- Open-drain interrupt pin

FIGURE 3. Extrait de la documentation technique de l'horloge temps-réel PCF8563 explicitant les adresses de communication sur bus I2C.

Cependant, un étudiant préférant s'inspirer de codes disponibles sur internet, au lieu de lire la documentation technique, trouvera l'exemple de code Arduino tel que celui proposé à tronixstuff.com/2013/08/13/tutorial-arduino-and-pcf8563-real-time-clock-ic/ :

```
#include "Wire.h"
#define PCF8563address 0x51
[...]

void setPCF8563()
// this sets the time and date to the PCF8563
{
  Wire.beginTransmission(PCF8563address);
  Wire.write(0x02);
[...]
```

L'implémentation en C des fonctions issues de cet extrait de code se traduit invariablement par l'échec de la transaction, puisque l'utilisation de l'adresse 0x51 ne permet pas d'adresser le composant que nous avons sélectionné. Il est évident pour un programmeur expérimenté que `0x51=0xA2≫1`, mais comment expliquer cette différence de comportement ?

Un des avantages d'Arduino est d'être opensource, et donc de permettre de remonter le fil des bibliothèques appelées pour en comprendre le fonctionnement : nous allons suivre ce cheminement pour analyser l'origine de la différence d'adresse programmée avec la documentation technique. En effet, la bibliothèque `Wire` de Arduino[8] implémente la fonction `beginTransmission` qui fait appel à `twi_writeTo(txAddress ...)`, fonction elle même implémentée dans une bibliothèque externe dont les sources se trouvent (par exemple) à www.libelium.com/v11-files/api/waspmote/d5/d8c/twi_8c_source.html :

```
  twi_slarw = TW_WRITE;
  twi_slarw |= address << 1;
```

8. github.com/esp8266/Arduino/blob/master/libraries/Wire/Wire.cpp

Nous constatons que cette bibliothèque externe à Arduino se charge du décalage et de définir le bit de poids faible (1 si lecture, 0 si écriture)[9]. Ainsi, avons nous réellement amélioré la capacité de compréhension en devant suivre tout ce cheminement entre la définition par le constructeur des paramètres de communication avec son composant, et l'implémentation dans la bibliothèque Arduino ? Nous ne connaissons pas d'implémentation matérielle d'USART implémentant I2C qui demande à recevoir l'adresse du périphérique en dehors d'un octet comportant dans son bit de poids faible la direction de la transaction. Le code suivant, en C brut, nous semble autrement plus limpide, sous réserve de maîtriser le concept de pointeur :

```
#define PCF8563           0xA2

void I2C_write(uint8_t ad, uint8_t *da, uint8_t n)
{int i;
 I2C_start();
 TWDR=ad; // send the slave address
 for(i=0; i<n; i++) TWDR=(*da++);
 I2C_stop();
}

[...]
I2C_write(PCF8563,(uint8_t *)time,8);
```

V. CIBLES ACCESSIBLES ET ALTERNATIVES

Le troisième point que nous désirons aborder est le niveau d'abstraction du développement en relation avec la complexité du matériel disponible. Exploiter pleinement un matériel complexe nécessite non-seulement de passer à un niveau d'abstraction supérieur de la programmation bas niveau compte tenu de la difficulté d'appréhender le fonctionnement de périphériques complexes (bibliothèques), mais aussi d'adapter la méthode de travail à la complexité du logiciel. Un développement complexe à plusieurs programmeurs s'accomode mal d'un unique binaire monolithique fusionnant toutes les fonctionnalités, d'où l'apparition d'environnement exécutifs, voir de systèmes d'exploitations. Dans le premier cas les développeurs ont l'impression d'accéder aux fonctionnalités d'un système d'exploitation même si le binaire résultant est monolithique et statique, tandis que dans le second cas le noyau et son ordonnanceurs sont protégés de l'espace utilisateur par le gestionnaire de mémoire. Le portage d'Arduino à des processeurs multi-cœurs cadencés à plusieurs centaines de megahertz pose la question de la pertinence d'attirer ces utilisateurs vers des plateformes aussi puissantes dont les ressources ne pourront que partiellement être appréhendées avec un niveau d'abstraction aussi faible.

Alors qu'Arduino semble un formalisme de développememt approprié pour de petites cibles aux ressources réduites (l'Atmega32U4 Leonardo propose 2500 octets de RAM), la tendance semble être de promouvoir cette bibliothèque vers des cibles de plus en plus puissantes, telles que l'Intel Edison, un double-cœur cadencé à 500 MHz et proposant 1 GB de RAM. Est-il raisonnable de penser exploiter pleinement de telles ressources avec une programmation avec un niveau d'abstraction aussi bas que Arduino, alors que la même

9. robotika.yweb.sk/skola/!Diplomovka/avr498/avr498/doc/twi__lib_8h.html indique que `TWI_READ` vaut 1 et `TWI_WRITE` vaut 0.

plateforme supporte Linux ? Plus proche de nos considérations pédagogiques, nous avons sélectionné pour notre enseignement de Master des plateformes basées sur le processeur STM32 de ST-Microelectronics, un cœur d'ARM Cortex qui se décline avec une multitude de périphériques et de ressources de mémoire, mais que par soucis de coût nous limitons à 8 KB (STM32F100), voir 32 KB (STM32F410) de RAM. Ici encore Arduino tente de s'imposer [10], au détriment probable, par soucis de simplicité, d'aborder des techniques aux perspectives bien plus alléchantes d'environnements exécutifs [10] tels que FreeRTOS [11] RTEMS [13] [12] ou NuttX qui proposent une réelle capacité d'introduction aux méthodes de travail multitâches et problèmes associés de partage de ressources (sémaphores, mutex et queues de données), ainsi qu'aux notions fondamentales en contrôle et traitement du signal de latences (environnements temps-réel). NuttX est particulièrement attractif, avec sa capacité à s'exécuter sur moins de 20 KB de RAM, certes avec une configuration minimaliste mais une interface utilisateur fonctionnelle, avec une interface de programmation applicative (API) visant la compatibilité POSIX, la liste des appels systèmes sélectionnée par toutes les implémentations récentes d'Unix (et en particulier Linux). Ce résultat est démontré par la configuration `stm32f103-minimum/nsh` de NuttX, que nous flashons dans un STM32F103 muni de 32 KB de RAM

```
http://stm32flash.sourceforge.net/

Using Parser : Raw BINARY
Interface serial_posix: 57600 8E1
Version      : 0x22
Option 1     : 0x00
Option 2     : 0x00
Device ID    : 0x0414 (STM32F10xxx High-density)
- RAM        : 64KiB  (512b reserved by bootloader)
- Flash      : 512KiB (size first sector: 2x2048)
- Option RAM : 16b
- System RAM : 2KiB
Write to memory
Erasing memory
Wrote address 0x08009a54 (100.00%) Done.
```

mais dont seuls 20 KB sont effectivement déclarés au compilateur

```
CONFIG_RAM_START=0x20000000
CONFIG_RAM_SIZE=20480
```

pour donner l'interface utilisateur suivante, exécutée sur le microcontrôleur :

```
nsh> help
help usage:  help [-v] [<cmd>]

  [       dd       false    ls     set      unset
  ?       echo     free     mb     sh       usleep
  cat     printf   help     mh     sleep    xd
  cd      exec     hexdump  mw     test
  cp      exit     kill     pwd    true

Builtin Apps:
nsh> free
             total       used       free    largest
Mem:         17488      10320       7168       7168
```

10. www.stm32duino.com/
11. Exemple de sujet de TP : jmfriedt.free.fr/tp_freertos.pdf
12. devel.rtems.org/wiki/TBR/BSP/STM32_F4 et exemple de TP jmfriedt.free.fr/tp_rtems.pdf

Nous avons bien un *shell* interactif fonctionnel, qui tient dans 10320 octets de mémoire, avec un système d'exploitation supportant par ailleurs les appels systèmes POSIX.

FreeRTOS, quant à lui, s'exécute désormais, grâce au lien avec `libopencm3`, sur toute cible ARM Cortex supportée par cette bibliothèque, tel que nous le démontrons à github.com/jmfriedt/tp_freertos/, pour des ressource occupées (taille du binaire transféré en mémoire non-volatile) sensiblement identiques à celles requises par Arduino.

Par ailleurs FreeRTOS n'étant composé que de quelques (6) fichiers sources généraux fournissant les fonctionnalités de l'environnement exécutif (tâches, mutex et sémaphores, queues de communication entre autres) et d'un fichier (`port.c`) dépendant de chaque plateforme, il a été possible de le lier à la bibliothèque Arduino [13], proposant ainsi une transition progressive entre les deux environnements de travail.

VI. Conclusion

Il ne fait aucun doute que la bibliothèque Arduino et l'environnement intégré de travail associé amènent des amateurs vers le domaine de l'électronique numérique embarquée, ce dont nous ne pouvons que nous féliciter. Cependant, il nous semble que l'enseignement des méthodes de développement qui sont susceptibles d'être mises en œuvre en dehors d'un cadre ponctuel de développement amateur – qu'il s'agisse de développements robustes ou visant un niveau d'abstraction supérieur par le passage aux environnements exécutifs, voir l'écriture des pilotes nécessaires à un système d'exploitation tel que GNU/Linux pour accéder aux ressources matérielles – ne peut se contenter d'une connaissance superficielle d'une bibliothèque inefficace. Dans le cadre d'un enseignement universitaire orienté vers l'électronique, au-delà la licence 3, une compréhension détaillée de l'architecture du processeur, de ses modes de communications avec ses périphériques, de la chaîne de compilation (`gcc`) et des outils associés (`binutils`, voir `newlib`) sont des conditions incontournables pour fournir des produits fiables et pérennes. La disponibilité des codes sources pour comprendre les implémentations des bibliothèques sont évidemment un point clé lors de l'apprentissage : en ce sens Arduino réspecte nos ambitions d'encourager les étudiants à comprendre leurs outils, mais bien d'autres bibliothèques (`newlib` ou les autres implémentations libres de `libc`) en font autant.

Références

[1] N. Diakopoulos & S. Cass, *The Top Programming Languages 2016* (26 Juillet 2016), à spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

[2] Wen-Hsuan Kuan, Chi-Hung Tseng, Sufen Chen, Ching-Chang Wong, *Development of a Computer-Assisted Instrumentation Curriculum for Physics Students : Using LabVIEW and Arduino Platform*, Journal of Science Education and Technology **25** (3), pp 427–438 (2016) à

[3] C. Galeriu, S. Edwards and G. Esper, *An Arduino Investigation of Simple Harmonic Motion*, The Physics Teacher **52** (157), pp.157–159 (2014)

[4] C. Galeriu, C. Letson and G. Esper, *An Arduino Investigation of the RC Circuit* The Physics Teacher **53** (285) pp.285–288 (2015)

13. *Using FreeRTOS multi-tasking in Arduino* à create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc

[5] F. Bouquet, J. Bobroff, M. Fuchs-Gallezot, & L. Maurines, *Project-based physics labs using low-cost open-source hardware*, arXiv preprint arXiv :1601.06659 (2016), accepté American Journal of Physics.

[6] S. Kubínová and J. Slgr, *ChemDuino : Adapting Arduino for Low-Cost Chemical Measurements in Lecture and Laboratory* J. Chem. Educ. **92** (10), pp.1751–1753 (2015) DOI : 10.1021/ed5008102

[7] www.arduino.cc/en/Guide/Introduction

[8] B.M. Hoffer, *Satisfying STEM Education Using the Arduino Microprocessor in C Programming*, Master Thesis (2012), à dc.etsu.edu/etd/1472

[9] C. Hochgraf, *Using Arduino To Teach Digital Signal Processing*, Proc. ASEE Northeast Section Conference (2013)

[10] P. Jamieson, *Arduino for teaching embedded systems. are computer scientists and engineering educators missing the boat ?*, Proc. FECS, 289-294 (2010)

[11] *Status of Supported Architectures from Maintainers' Point of View*, à gcc.gnu.org/backends.html

[12] P. Levis & D. Gay, *TinyOS programming*, Cambridge University Press (2009)

[13] J.-M Friedt, G. Goavec-Mérou, *Interfaces matérielles et OS libres pour Nintendo DS : DSLinux et RTEMS*, GNU/Linux Magazine France Hors Série **43** (2009), disponible à jmfriedt.free.fr/nds.pdf