TP afficheur 7-segments

É. Carry, J.-M Friedt

23 janvier 2017

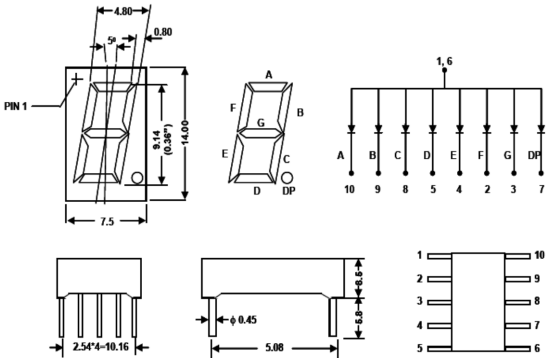
Questions sur ce TP :

1. Quelle structure de donnée permet un échange entre la fonction principale du programme (`main()`) et un gestionnaire d'interruption ?
2. Pourquoi faut-il éviter d'utiliser une telle structure de donnée en dehors de ce cas particulier ?
3. Combien de segment(s) affichant chaque chiffre d'un nombre sont sous tension à un instant donné ?
4. Quel mécanisme évite de consacrer tout le temps d'exécution du programme à l'affichage et permet de faire d'autres tâches en parallèle ?
5. Quelle condition sur un GPIO permet d'afficher un segment de LED ?
6. Quelle condition de polarité sur la base d'un transistor PNP permet le passage du courant entre le collecteur et l'émetteur ?
7. que vaut 0x59+1 en format hexadécimal ? en format BCD ?

L'objectif de ce TP est d'exploiter dans un contexte concret et pratique les connaissances acquises sur la manipulation des ports d'entrée-sortie numériques (GPIO) et sur les horloges (*timer*). En pratique, nous exploiterons l'affichage rapide d'informations sur des segments de diodes électroluminescentes (LED) pour donner l'impression d'afficher des messages. Cette stratégie s'applique dans le contexte plus vaste de *Persistence Of Vision* (POV) [1].

1 Principes de l'affichage séquentiel

Un afficheur 7-segments (Fig. 1) est formé d'un ensemble de 8 diodes (7-segments et un point décimal) que nous manipulons en plaçant le GPIO du microcontrôleur à l'état haut ou bas.



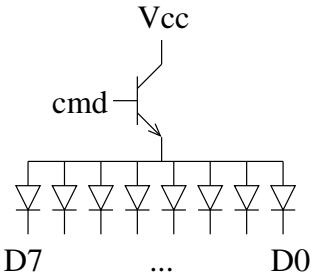


FIGURE 1 – Gauche : extrait de la *datasheet* d'un afficheur 7-segments, repris de `http://datasheet.sparkgo.com.br/LD3361BS.pdf`. Droite : câblage interne à l'afficheur 7-segments.

Le schéma de la carte qui se positionne sous l'Atmega32U4 comporte un transistor pour commuter l'alimentation de l'*anode commune* (activation ou désactivation de l'afficheur) et chaque segment voit sa sortie connectée à un bit du GPIO (Fig. 2)

Quelle condition sur le GPIO permet d'*allumer* un segment ?

Historiquement, un composant est chargé d'afficher une valeur en format BCD sur des afficheurs : le 7447. Cependant, les microcontrôleurs modernes proposent plus que la puissance nécessaire à faire ce travail, nous remplacerons ce périphérique matériel par du logiciel.

Afin d'exploiter le plus facilement possible les afficheurs 7-segments, nous allons petit à petit assembler une bibliothèque de fonctions de complexité croissante. Un point qui apparaîtra particulièrement pénible est que les broches contigües dans

1. `http://www.ladyada.net/make/spokepov/`

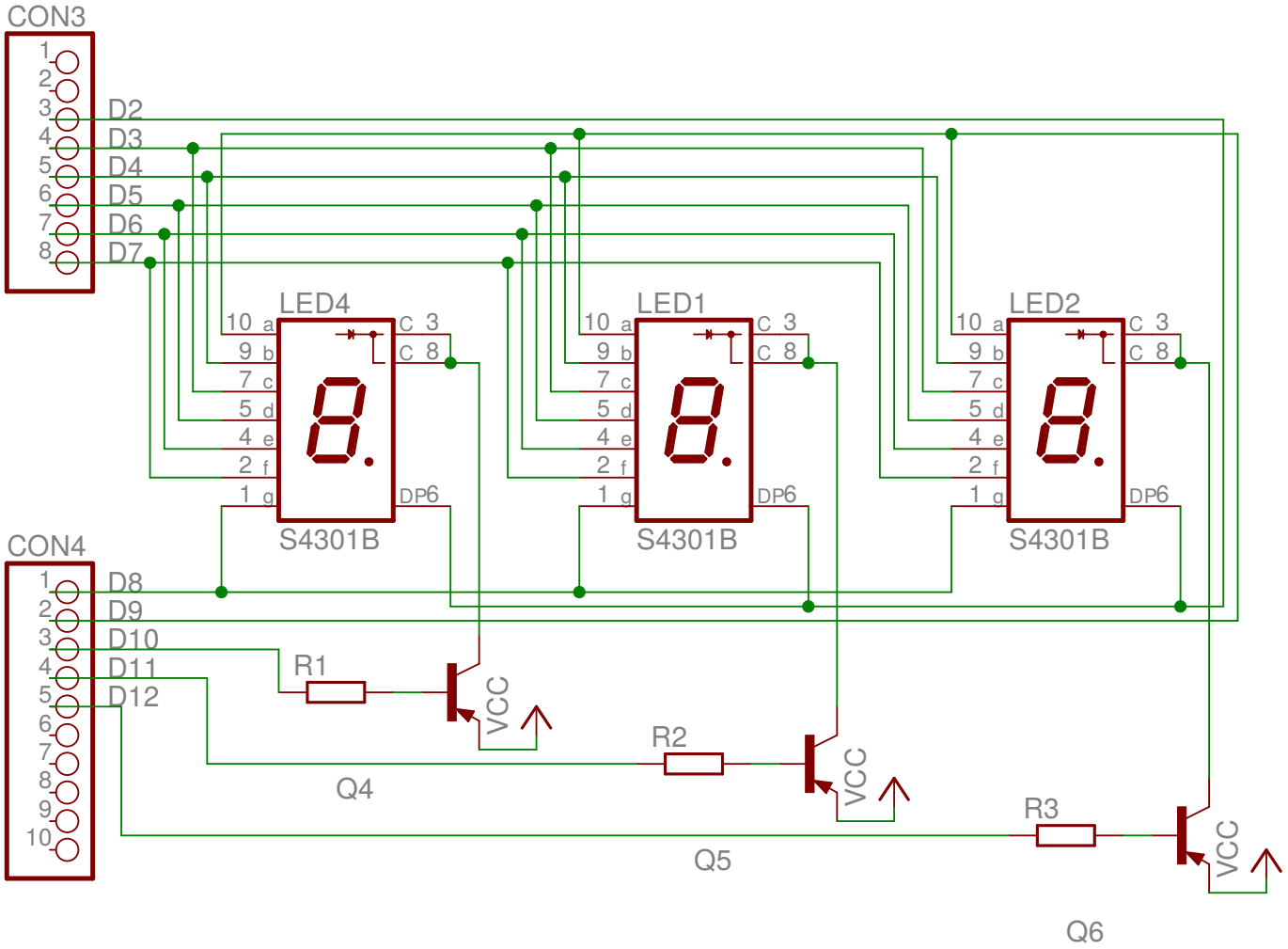


FIGURE 2 – Schéma de principe de la carte 7-segments.

la nomenclature Olimex sont associées à des ports différents du microcontrôleur. Ainsi, D{0-4,6,12} sont associées au port D, D{5,7} au port C, et D{8-11} au port B. Cela signifie que nous utiliserons *plusieurs opérations de masquage* pour définir les états de ces 3 ports en fonction des segments à allumer. Par ailleurs, deux tableaux seront nécessaires, un pour associer un bit d'un port à un segment, et l'autre pour indiquer quel port est utilisé.

Polariser une base de transistor pour alimenter un segment et démontrer l'allumage d'un segment prédéfini

⚠**Attention** : vérifier que la carte du microcontrôleur est alimentée en 5 V (cavalier à côté du connecteur USB)

⚠**Attention** : le transistor est de type PNP, donc devient passant pour une base polarisée à 0 V et bloquant pour une base polarisée à V_{cc}.

Pour ce faire, identifier quel segment est connecté à quel port :

Segment	Carte	Port/Broche
a	D9	PB5
b	D4	PD4
c	D3	PD0
d	D5	PC6
e	D6	PD7
f	D7	PE6
g	D8	PB4
.	D2	PD1
S1	D10	PB6
S2	D11	PB7
S3	D12	PD6

Illuminer chaque symbole du premier afficheur. On se référera pour cela au schéma de la Fig. 2 et à la correspondance entre port dans la nomenclature Olimex et port du microcontrôleur.

Afficher séquentiellement tous les segments de tous les afficheurs présents sur la carte. On se référera pour cela au schéma de la Fig. 2 et à la correspondance entre ports dans la nomenclature Olimex et port du microcontrôleur.

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

#define jmfB 1
#define jmfC 2
#define jmfD 3
#define jmfE 4

void aff(int port,int val)
{PORTB|=(1<<PORTB4)+(1<<PORTB5);
 PORTC|=(1<<PORTC6);
 PORTD|=((1<<PORTD0)+(1<<PORTD1)+(1<<PORTD4)+(1<<PORTD7));
 PORTE|=(1<<PORTE6);
 switch(port){
 case jmfB: PORTB&=~(1<<val); break;
 case jmfC: PORTC&=~(1<<val); break;
 case jmfD: PORTD&=~(1<<val); break;
 case jmfE: PORTE&=~(1<<val); break;
 }
}

void afficheur(int valeur)
{PORTB|=((1<<PORTB6)+(1<<PORTB7));
 PORTD|=(1<<PORTD6);
 switch(valeur){
  case 0:PORTB&=~(1<<PORTB6);break;
  case 1:PORTB&=~(1<<PORTB7);break;
  case 2:PORTD&=~(1<<PORTD6);break;
 }
}

int main(void)
{ int k,j;
  // segment A      B        C     D     E      F      G      .
  int p[8]={jmfB, jmfD,  jmfD,  jmfC,  jmfD,  jmfE,  jmfB,  jmfD  };
  int s[8]={PORTB5,PORTD4,PORTD0,PORTC6,PORTD7,PORTE6,PORTB4,PORTD1};
  wdt_disable();

  DDRB=0xF0;  // B{4,5,6,7}
  DDRC=0x40;  // C6
  DDRD=0xD3;  // D{7,6,4,1,0}
  DDRE=0x40;  // E6

  PORTB=0xff;
  PORTC=0;
  PORTD=0xff;
  PORTE=0;
  while (1){
    for (j=0;j<1;j++)
      {afficheur(j);for (k=0;k<8;k++) {aff(p[k],s[k]);_delay_ms(100);}}
  }
  return 0;
}
```

L'exploitation des afficheurs 7-segments est basée sur le principe d'une alimentation commune de toutes les diodes de l'afficheur, et la polarisation à la masse des broches du GPIO connectées aux segments qui doivent s'illuminer. Comme chaque afficheur est composé de 7 segments affichables, nous utilisons 7 bits d'un GPIO pour en commander l'état. Cependant, afficher 3 chiffres nécessiterait dans une approche naïve $3 \times 7 = 21$ broches. Afin de ne pas gaspiller inutilement la ressource précieuse qu'est la broche de GPIO, nous allons multiplexer temporellement (TDMA – *Time Domain Multiple Access*) l'accès aux ressources. La commutation entre les afficheurs se fera tellement rapidement que la persistance rétinienne donnera l'illusion d'un affichage permanent sur les 3 afficheurs.

Constater qu'en abaissant le délai entre deux illuminations de LED, la persistance rétinienne donne l'impression d'un affichage "simultané" de tous les symboles.

2 Affichage de symboles

Calculer le tableau qui définit les valeurs hexadécimales pour allumer tous les segments nécessaires à l'affichage de chaque symbole.

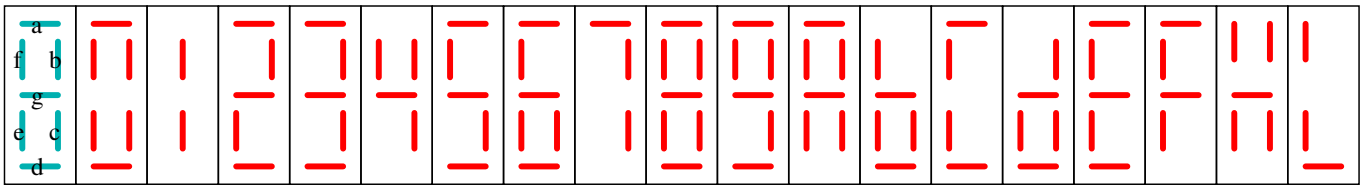


FIGURE 3 – Symboles affichables.

Proposer une solution à l'affichage de symboles, segment par segment. La modification principale tiendra en la remise à l'état *éteint* de tous les segments dans la fonction `dig()`, avant d'allumer les diverses LEDs nécessaires à représenter le symbole.

Réduire la lattence entre deux affichages. Que constatez-vous ?

3 Afficher un nombre au format hexadécimal

Une fois les affichages de chaque segment validé pour afficher un chiffre, nous voulons afficher un nombre en format hexadécimal.

Ajouter les symboles nécessaires, et démontrer.

4 Gestion de l'affichage par interruption

Il est fort peu pratique de devoir gérér manuellement le rafraîchissement de l'affichage, d'autant plus que le taux de rafraîchissement risque, dans un programme plus complexe, de devenir dépendant du temps d'exécution du programme principal. Afin de s'affranchir de ce risque, nous nous proposons de commander les afficheurs depuis une interruption *timer* qui séquencera le rafraîchissement des informations fournies sur chaque segment de LED.

À cet effet, des variables globales pourront être *exceptionnellement* utilisées afin de transférer des arguments du programme principal au gestionnaire d'interruption. En effet, l'interruption peut être appelée à tout moment de l'exécution du programme principal, et il n'existe aucun autre moyen que la zone de mémoire commune à l'interruption et au programme principal (*i.e.* variable globale) d'échanger des informations. Par ailleurs, nous avons besoin de mémoriser le statut d'une machine d'état qui se rappelle de la dernière opération effectuée par l'interruption. En effet, en tentant d'afficher tous les chiffres dans l'interruption, nous ne profiterions plus de l'effet de la persistance rétinienne qui suppose que chaque segment reste allumé pendant une durée égale.

```
1  #include <avr/io.h> //E/S ex PORTB
2  #define F_CPU 16000000UL
3  #include <util/delay.h>  // _delay_ms
4  #include <avr/interrupt.h>
5  #include <avr/wdt.h>
6
7  #define jmfB 1
8  #define jmfC 2
9  #define jmfD 3
10 #define jmfE 4
11
12 void nombre(short);
13
14 volatile int mon_afficheur=0x0555;
15 volatile int status=0;
16
17 ISR(TIMER3_COMPA_vect) {
18   nombre(mon_afficheur);
19   status++;if (status==3) status=0; // quel digit
20 }
21
22   // segment A      B        C     D     E      F      G      .
```

```
23 int p[8]={jmfB, jmfD,  jmfD,  jmfC,  jmfD,  jmfE,  jmfB,  jmfD  };
24 int s[8]={PORTB5,PORTD4,PORTD0,PORTC6,PORTD7,PORTE6,PORTB4,PORTD1};
25 int d[16]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71};
26
27 void aff(int port,int val)
28 {
29  switch(port){
30  case jmfB: PORTB&=~(1<<val); break;
31  case jmfC: PORTC&=~(1<<val); break;
32  case jmfD: PORTD&=~(1<<val); break;
33  case jmfE: PORTE&=~(1<<val); break;
34  }
35 }
36
37 void afficheur(int valeur)
38 {PORTB|=((1<<PORTB6)+(1<<PORTB7));
39  PORTD|=(1<<PORTD6);
40  switch(valeur){
41   case 0:PORTB&=~(1<<PORTB6);break;
42   case 1:PORTB&=~(1<<PORTB7);break;
43   case 2:PORTD&=~(1<<PORTD6);break;
44  }
45 }
46
47 void dig(int val)
48 {int k;
49  PORTB|=(1<<PORTB4)+(1<<PORTB5);  // met afficheur a 0
50  PORTC|=(1<<PORTC6);  // avant d'allumer les bons segments
51  PORTD|=((1<<PORTD0)+(1<<PORTD1)+(1<<PORTD4)+(1<<PORTD7));
52  PORTE|=(1<<PORTE6);
53  val=d[val];
54  for (k=0;k<8;k++) if ((val>>k)&0x01) {aff(p[k],s[k]);}
55 }
56
57 void nombre(short val)
58 {afficheur(2-status);
59  dig(val>>(status*4)&0xf);
60 }
61
62 int main(void)
63 {
64   wdt_disable();
65   USBCON=0;
66   TCCR3A=0;             // init timer 3
67   TCCR3B= (1 << WGM32)+(1 << CS31); // CS=1 Fclk/1, mode OC
68   TCCR3C=0;
69   OCR3A = 60000;      // valeur seuil : remplacer 60000 par 10000
70   TIMSK3= 1<<OCIE3A;
71
72   DDRB=0xF0;  // B{4,5,6,7}
73   DDRC=0x40;  // C6
74   DDRD=0xD3;  // D{7,6,4,1,0}
75   DDRE=0x40;  // E6
76
77   PORTB=0xff;
78   PORTC=0;
79   PORTD=0xff;
80   PORTE=0;
81   sei();
82
83   while (1) {mon_afficheur++;_delay_ms(1000);}
84 }
```

Dans cette implémentation, l'interruption déclenche le passage d'un chiffre au suivant, la fonction `afficheur()` sélectionne quel afficheur est activé, tandis que `dig()` place tous les segments lumineux à l'état éteint avant d'allumer les LEDs

nécessaires.

La machine d'états doit d'une part gérer quel afficheur est actif, et ce afin de balayer séquentiellement chaque chiffre du nombre à afficher. Démontrer une implémentation fonctionnelle.

Afin de ne pas appeler trop fréquemment l'interruption de rafraîchissement des afficheurs, il est souhaitable d'abaisser au maximum la fréquence d'interruption de *timer3*.

Quelle valeur de seuil de remise à 0 de *timer3* permet d'obtenir un affichage stable. À quelle fréquence correspond cette valeur?

5 Horloge

Plutôt qu'afficher un nombre au hasard, nous désirons incrémenter deux compteurs, minutes et secondes, au rythme de 1 Hz pour réaliser une horloge. Ayant choisi d'afficher des nombres au format hexadécimal, nous devons donc implémenter des opérations de comptage au format BCD.

Rappeler le format BCD.

Par ailleurs, le microcontrôleur est susceptible d'effectuer d'autres tâches que simplement rafraîchir toutes les secondes les afficheurs. Nous nous proposons donc d'incrémenter les compteurs de seconde et minute sous commande d'une interruption timer.

```
[...]
volatile int mon_afficheur=0x0000;
volatile int sec=0,min=0;

ISR (TIMER1_COMPA_vect)
{sec++;
 if ((sec&0x0f)==10) sec+=6;    // 10 -> 0x10
 if (sec==0x60) {sec=0;min++;}  // attention a l'ordre
 mon_afficheur=((min&0x0f)<<8)+sec;
}

[...]
int main(void)
{ USBCON=0;
  TCCR3A=0;             // init timer 3
  TCCR3B= (1 << WGM32)+(1 << CS31); // Fclk/8, mode OC
  TCCR3C=0;
  OCR3A = 11000;      // delai : 51000 clignote, remplacer par 11000
  TIMSK3= 1<<OCIE3A;

  TCCR1B |= (1 << WGM12)|(1 << CS12)|(1 << CS10); // 1024
  OCR1A = 15625;      // valeur seuil <- delai
  TIMSK1 |= (1 << OCIE1A);

[...]
  while (1){}
}
```

Commander les afficheurs afin d'obtenir un compteur d'horloge qui s'incrémente chaque seconde sur interruption du *timer1*. La boucle infinie de la fonction `main()` doit se résumer à `while (1) {}`.

6 Message défilant

Afin d'améliorer les fonctionnalités proposées par les 3 afficheurs 7-segments, nous désirons faire *défiler du texte* de droite à gauche.

Démontrer l'affichage d'un nombre de 9 chiffres sur les 3 afficheurs 7-segments.

Deux lettres peuvent s'encoder sur un afficheur 7-segments : `H` et `L`, en plus des symboles A-F du décompte hexadécimal.

Ajouter ces deux symboles supplémentaires, ainsi que l'espace pour lequel tous les segment sont éteints, et faire défiler le texte `HELLO` sur les 3 afficheurs (Fig. 4).

⚠**Attention :** la taille de l'`int` n'est pas normalisée et dépend de l'architecture. **Vérifier que sur cette architecture Atmel, un `int` est codé sur 2 octets.** Dans le contexte d'un encodage du message à afficher sur 3 afficheurs dans un entier, nous prendrons soin d'utiliser un `long` (4 octets, quelque soit l'architecture) et non `int`.

```
[...]
```

```
2  // ATTENTION : int sur AVR est 2-byte long. 4-byte = long
3  void nombre(long*);
4
5  volatile long* mon_afficheur;  // ATTENTION : pointeur
6  volatile int status=0;
7
8  ISR(TIMER3_COMPA_vect) {
9    nombre(mon_afficheur);
10   status++;if (status==3) status=0; // quel digit
11 }
12
13   // segment A      B        C     D     E      F      G      .
14 int p[8]={jmfB, jmfD,  jmfD,  jmfC,  jmfD,  jmfE,  jmfB,  jmfD  };
15 int s[8]={PORTB5,PORTD4,PORTD0,PORTC6,PORTD7,PORTE6,PORTB4,PORTD1};
16 int d[19]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39, \
17   0x5E,0x79,0x71,0x76,0x38,0};
18
19 void aff(port,val)
20 {[...]}
21
22 void afficheur(int valeur) // 0 le plus a gauche
23 {[...]}
24
25 void dig(int val)
26 {[...]}
27
28 void nombre(long* val)
29 {afficheur(status);
30  dig((*val)>>(status*8)&0xff); // octet par octet et non par quartet
31 }
32
33 int main(void)
34 {// char msg[9]={1,2,3,4,5,6,7,8,9};
35  char msg[9]={18,18,16,14,17,17,0,18,18};
36  int k=0;
37   USBCON=0;
38   TCCR3A=0;             // init timer 3
39   TCCR3B= (1 << WGM32)+(1 << CS31);
40   TCCR3C=0;
41   OCR3A = 11000;
42   TIMSK3= 1<<OCIE3A;
43 [... initialisation des ports]
44   sei();
45
46   while (1){
47     mon_afficheur=(long*)(msg+k);
48     k++;
49     if (k==7) k=0; // on fait appel a k+2 => k<9
50     _delay_ms(200);
51    }
52 }
```

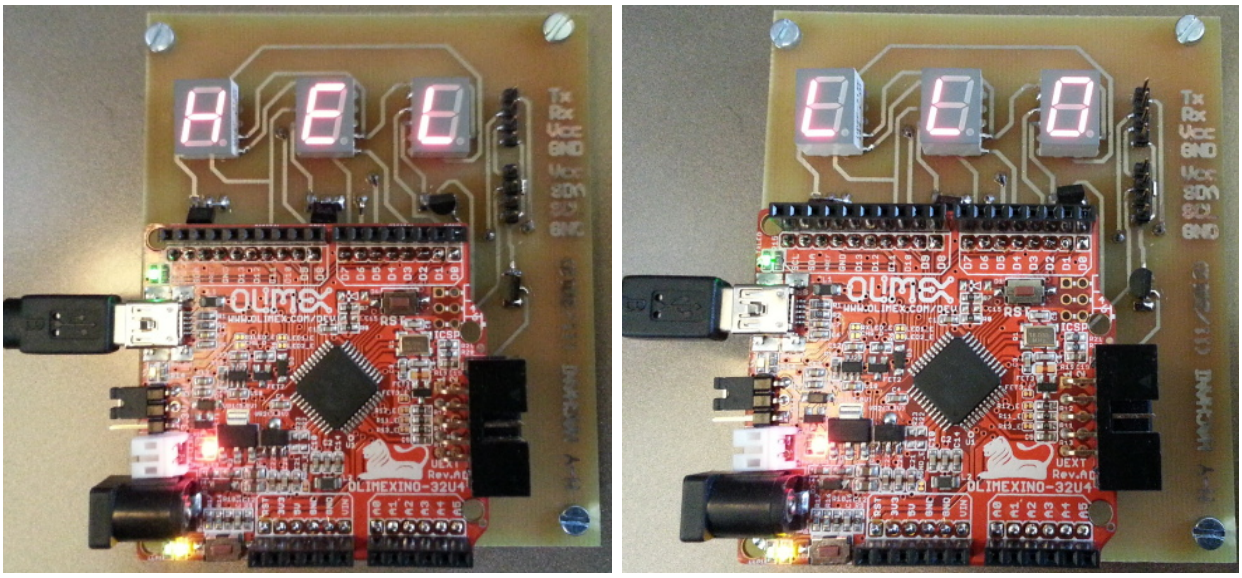


FIGURE 4 – Affichage d'un message défilant sur les afficheurs 7-segments. À gauche les lettres `HEL`, à droite `LLO`.