Examen “Systèmes embarqués”

J.-M Friedt, 16 janvier 2014

version numérique à http://jmfriedt.free.fr/exam2011.pdf

1 Remplacement d’une liaison numérique par une liaison analogique

Un dispositif RADAR de mesure des propritétés d’un capteur radiofréquence (RF) fonctionnant entre 33 et 35 MHz exploite une synthèse numérique de fréquence (DDS [1]) pour générer un signal RF de fréquence f_{Hz} programmée par un mot long (32 bits) selon la formule de conversion

$$f_{Hz} = \frac{200.10^6}{2^{32}} \times mot_{bits}$$

(200×10^6 correspond à la fréquence interne du DDS, 200 MHz).

Ce capteur peut être sondé avec un taux de raffraîchissement de l’information de 10 kHz. Normalement, nous communiquons les résultats au moyen d’un port série asynchrone compatible RS232 cadencé à 57600 bauds, protocole 8N1 (8 bits de données, 1 bit d’arrêt, pas de bit de parité, donc 10 bits par donnée transmise) : chaque trame contenant les informations de mesures contient 29 caractères ASCII.

1. quel est le temps de transmission des informations issues de la mesure ? comparer au temps nécessaire à une mesure.

Un convertisseur numérique analogique (DAC) fournit un moyen de communiquer rapidement le résultat d’un calcul à un utilisateur sous forme de tension analogique. De nombreux microcontrôleurs sont actuellement équipés de DAC cadencés à 1 Méchantillons/s. Nous nous proposons, afin d’augmenter la bande passante de la mesure, de remplacer la liaison asynchrone numérique par une sortie en tension, moins précise, mais plus facilement exploitable par l’utilisateur.

Le convertisseur DAC que nous utilisons nécessite une tension de référence de 3,3 V, et propose une résolution de 12 bits.

2. rappeler la relation entre tension de sortie et mot programmé dans le registre du DAC.

Afin de n’expoiter que les bits significatifs du DDS, nous programmons le DAC au moyen des bits 12 à 23 du DDS. Ainsi, nous éliminons les bits qui ne varient pas au cours de la mesure (bits 24 à 32) et ceux qui ne sont pas significatifs car perdus dans le bruit de mesure (bits 0 à 11).

3. Donner la relation entre tension de sortie du DAC et fréquence générée par le DDS.

2 Adressage de la pile

Nous avons vu que la pile est une zone de mémoire volatile (RAM) servant de brouillon au processeur, notamment pour y stocker des informations temporaires lors de l’appel à des fonctions ou lors de la gestion des interruptions.

Par défaut, le script de configuration de l’éditeur de lien fourni par une bibliothèque libre pour Cortex M3 (https://github.com/esden/libopencm3) propose la configuration suivante [2] :

```
MEMORY
{
rom (rx) : ORIGIN = 0x08000000, LENGTH = 128K
ram (rwx) : ORIGIN = 0x20000000, LENGTH = 20K
}
```

puis inclut une fonction qui propose la définition suivante de la pile [3]

```
PROVIDE(_stack = 0x20000800);
```

Toutes ces commandes sont valides et fonctionnelles. Cependant, lors de l’exécution du programme

1. http://www.analog.com/en/rfif-components/direct-digital-synthesis-dds/ad9954/products/product.html
2. https://github.com/esden/libopencm3/blob/master/examples/stm32/f1/stm32-h103/stm32-h103.ld
3. https://github.com/esden/libopencm3/blob/master/lib/stm32/f1/libopencm3_stm32f1.ld

```
#include <libopencm3/stm32/usart.h>

int main()
{short tableau[1024];
 int k;
 char c;

 clock_setup();  // initialisation de l'horloge : fonctionnelle
 usart_setup();  // initialisation des ports de communication : fonctionnelle

 for (k=0;k<1024;k++)
      {tableau[k]=k;
       c=(tableau[k]&0xf000)>>12;
       if (c<10) usart_send_blocking(USART1, c+'0'); else usart_send_blocking(USART1, c+'A'-10);
       c=(tableau[k]&0x0f00)>>8;
       if (c<10) usart_send_blocking(USART1, c+'0'); else usart_send_blocking(USART1, c+'A'-10);
       c=(tableau[k]&0x00f0)>>4;
       if (c<10) usart_send_blocking(USART1, c+'0'); else usart_send_blocking(USART1, c+'A'-10);
       c=(tableau[k]&0x000f);
       if (c<10) usart_send_blocking(USART1, c+'0'); else usart_send_blocking(USART1, c+'A'-10);
       usart_send_blocking(USART1,'\r'); usart_send_blocking(USART1,'\n');
           // fonction fournie par libopencm3 pour transmettre un char sur RS232
      }
}
```

le microcontrôleur crashe (arrêt de la transmission de données) avant la fin de l'exécution de la boucle `for()` du programme.

1. expliquer la cause du crash
2. proposer un remède.

À titre indicatif (mais le problème ne se trouve pas dans ces lignes de code), l'initialisation du microcontrôleur s'obtient par

```
#include <libopencm3/stm32/f1/rcc.h>
#include <libopencm3/stm32/f1/gpio.h>
#include <libopencm3/stm32/usart.h>

void clock_setup(void) {rcc_clock_setup_in_hse_8mhz_out_72mhz();} // STM32 to 72 MHz.

void usart_setup(void)
{
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_IOPAEN );
  rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_USART1EN);
  gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_10_MHZ,
          GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO_USART1_TX);

  usart_set_baudrate(USART1, 57600);
  usart_set_databits(USART1, 8);
  usart_set_stopbits(USART1, USART_STOPBITS_1);
  usart_set_mode(USART1, USART_MODE_TX_RX);
  usart_set_parity(USART1, USART_PARITY_NONE);
  usart_set_flow_control(USART1, USART_FLOWCONTROL_NONE);
  usart_enable(USART1);
}
```

et la compilation de ce programme est effectuée par le Makefile suivant

```
export BASE_DIR=/home/jmfriedt/sat/arm-none-eabi/
export STM32_INC=${BASE_DIR}/include/

all: st.elf

rs.o: rs.c
  arm-none-eabi-gcc -c rs.c -O0 -g3 -DSTM32F1 -Wall -Wextra -I${STM32_INC} -I${STM32_INC}/libopencm3/stm32 -I${STM32_INC}/libopencm3/stm32/f1 -fno-common -mthumb -msoft-float -mcpu=cortex-m3 -nostartfiles -msoft-float -mthumb -mcpu=cortex-m3

st.o: st.c
  arm-none-eabi-gcc -c st.c -O0 -g3 -DSTM32F1 -Wall -Wextra -I${STM32_INC} -I${STM32_INC}/libopencm3/stm32 -I${STM32_INC}/libopencm3/stm32/f1 -fno-common -mthumb -msoft-float -mcpu=cortex-m3 -nostartfiles -msoft-float -mthumb -mcpu=cortex-m3
```

```
11
st.elf: st.o rs.o
13  arm-none-eabi-gcc -o st.elf st.o rs.o -lopencm3_stm32f1 -O0 -g3 -DSTM32F1 -Wall -->
        ↪Wextra -fno-common -mthumb -msoft-float -mcpu=cortex-m3 -lc -lnosys -L${→
        ↪BASE_DIR}/lib/stm32 -nostartfiles -msoft-float -mthumb -mcpu=cortex-m3 -T./→
        ↪libopencm3_stm32.ld
    arm-none-eabi-objcopy -Obinary st.elf st.bin
15  arm-none-eabi-objcopy -Oihex st.elf st.hex
    arm-none-eabi-objcopy -Osrec st.elf st.srec
17  arm-none-eabi-objdump -dSt st.elf > st.list

19clean:
    rm st.??? *.o st.srec
```

3 Afficheurs

Historiquement, une méthode classique pour afficher un nombre avec plusieurs chiffres (par exemple afficheur de calculatrice) consistait à rapidement activer l'afficheur correspondant à chaque chiffre, et ce avec une vitesse de raffraîchissement plus rapide que la persistance rétinienne (POV) [1]. Ainsi, l'œil humain donne l'impression d'un affichage stable d'un nombre défini par plusieurs chiffres, tout en ne nécessitant que peu de ressources matérielles du microcontrôleur puisque seuls un bus de données partagé par tous les afficheurs, et une logique d'adressage pour séquentiellement activer chaque afficheur, sont nécessaires (http://www.opencircuits.com/POV_display).
De la même façon, une méthode élégante pour afficher un motif sur un objet mobile consiste à rapidement modifier la valeur sur un port général (GPIO), et ce à une vitesse ajustée pour toujours positionner le même motif lorsque la barette d'afficheurs se trouve à une position donnée. La persistance rétinienne donne alors l'impression d'un image statique [2, 3]. Un exemple d'un tel montage exploitant un microcontrôleur pour commander 30 diodes est proposé à http://www.ladyada.net/make/spokepov/.

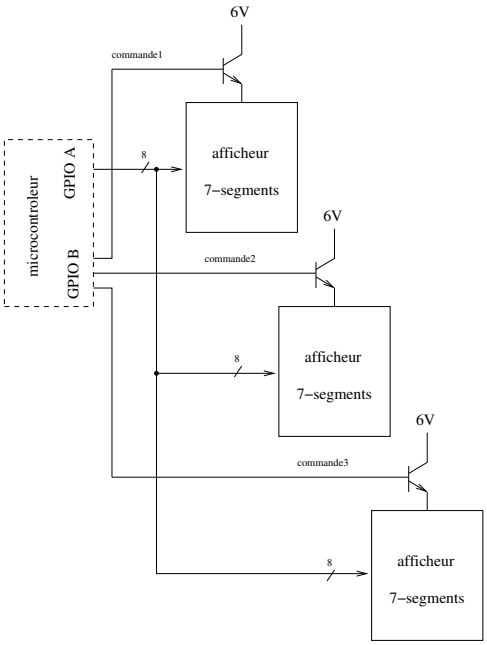


Figure 1: schéma de principe d'un affichage de nombres sur 3 chiffres.

1. Sachant qu'une diode électroluminescente consomme environ 20 mA lorsqu'elle est allumée, commenter (en justifiant) l'affirmation
"Runs on 2-3 AA batteries for 10 hours or more, assuming 3000mAh alkalines and 50% image coverage."

Un afficheur 7-segments est un agencement de diodes électroluminescentes (LED) permettant d'afficher les symboles de 0 à 9 et A à F : ils sont donc appropriés pour afficher des valeurs en hexadécimal.

2. Sachant qu'un afficheur 7-segments nécessite 8 fils pour représenter chaque chiffre (7 segments et le point des décimales), combien de signaux faudrait-il pour afficher un nombre sur 10 chiffres ? Sachant que chacun de ces chiffres représente une valeur hexadécimale, quelle est l'intervalle de nombres (positifs) affichables ?
3. Quelle valeur faut-il mettre sur le GPIO pour afficher le chiffre 5 sur un afficheur à anode commune (attention à la logique ...) ?
4. En supposant que nous possédions une fontion `affiche(char valeur, char position);` capable de prendre en argument une valeur de type `char`, `valeur`, entre 0 et 15 et une adresse `position` afin d'allumer les bonnes LEDs de l'afficheur pour afficher le chiffre désiré, proposer un exemple en C d'exploitation des masques logiques pour afficher le nombre 0x1234 sur les 4 premiers segments.

La méthode POV permet de n'exploiter qu'un bus de données pour commander tous les afficheurs, et l'adressage de chaque afficheur s'obtient en commutant l'alimentation de l'afficheur (Fig. 1).

5. Combien de fils faut-il dans ce cas pour atteindre le même résultat (afficher un nombre de 10 chiffres) ?

Un inconvénient de cette méthode est de nécessiter trop de signaux pour adresser les divers afficheurs. On utilise par conséquent un démultiplexeur, par exemple le 74LS238, puisque seul un afficheur est actif à un instant donné.

6. Combien de signaux faut-il dans ce cas pour atteindre le même résultat (adressage séquentiel de N afficheurs) ?

7. Proposer un schéma exploitant deux 74LS238 comme décodeurs d'adresse et quelques portes logiques additionnelles pour afficher un nombre à 10 chiffres.

4 ... encore une fois

1. Exprimer 0xfffa en décimal
2. Exprimer 0d321 en hexadécimal
3. Quel est le nom de la première fonction (point d'entrée) dans un programme en C ?
4. Qu'affiche la procédure

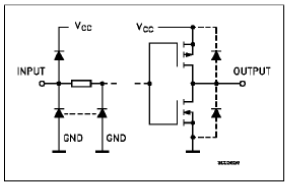
```
volatile unsigned int c=0x1234;
printf("%d ", (c & 0xf0f0) >> 4 );
```

5. Comment afficher le contenu d'un répertoire dans un shell unix ?

Annexe : datasheet de composants utiles

Démultiplexeur 74LS238

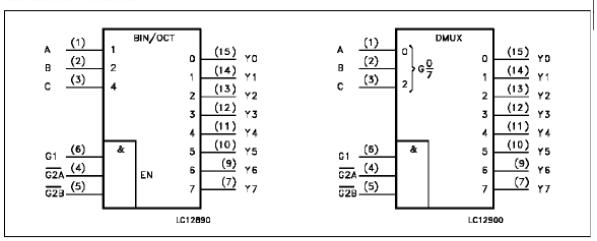
INPUT AND OUTPUT EQUIVALENT CIRCUIT



PIN DESCRIPTION

PIN No	SYMBOL	NAME AND FUNCTION
1, 2, 3	A, B, C	Data Inputs
4, 5	$\overline{G2A}$ $\overline{G2B}$	Enable Input (Active LOW)
6	G1	Data Enable Input (Active HIGH)
15, 14, 13, 12, 11, 10, 9, 7	Y0 to Y7	Outputs
8	GND	Ground (0V)
16	Vcc	Positive Supply Voltage

IEC LOGIC SYMBOLS

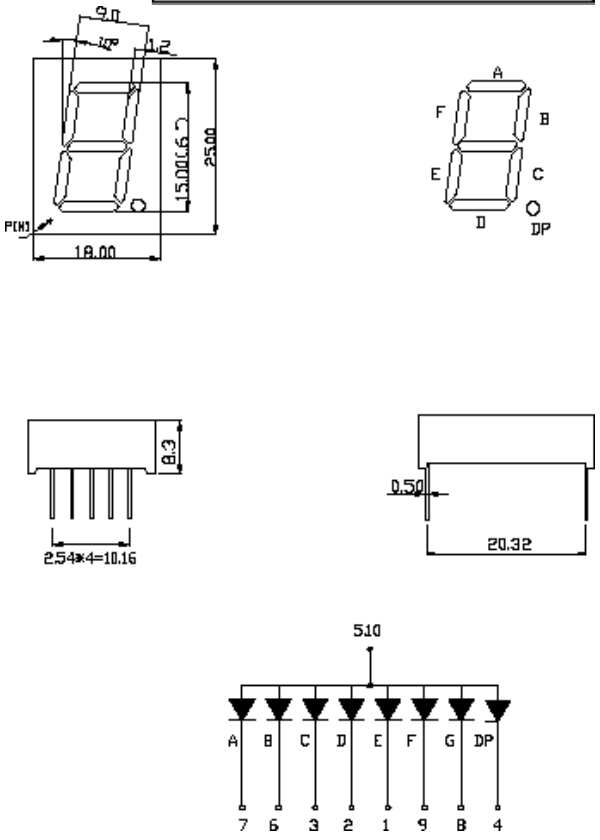


TRUTH TABLE

INPUTS						OUTPUS								SELECTED OUTPUT
ENABLE			SELECT											
$\overline{G2B}$	$\overline{G2A}$	G1	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	
X	X	L	X	X	X	L	L	L	L	L	L	L	L	NONE
X	H	X	X	X	X	L	L	L	L	L	L	L	L	NONE
H	X	X	X	X	X	L	L	L	L	L	L	L	L	NONE
L	L	H	L	L	L	H	L	L	L	L	L	L	L	Y0
L	L	H	L	L	H	L	H	L	L	L	L	L	L	Y1
L	L	H	L	H	L	L	L	H	L	L	L	L	L	Y2
L	L	H	L	H	H	L	L	L	H	L	L	L	L	Y3
L	L	H	H	L	L	L	L	L	L	H	L	L	L	Y4
L	L	H	H	L	H	L	L	L	L	L	H	L	L	Y5
L	L	H	H	H	L	L	L	L	L	L	L	H	L	Y6
L	L	H	H	H	H	L	L	L	L	L	L	L	H	Y7

Afficheur 7-segments (anode commune)

Numerical Designations—Resultant Displays



Références

[1] P. Horowitz & W. Hill, *The Art Of Electronics - 2nd Edition*, Cambridge University Press 1989, pp.546-

[2] https://wiki.blinkenarea.org/index.php/HAR2009

[3] http://www.ouyeah.net/science-and-education/led-cube-this-half-salado-i-did-not-is-a-tuto-1/

Correction

1 Remplacement d'une liaison numérique par une liaison analogique

1. 10 bits par symbole (8 bits de données, start et stop) à 57600 bits/s soit 5760 symboles par secondes et 29 symboles prennent donc 5 ms.
2. $V_{DAC} = \frac{V_{REF}}{2^{res}} \times mot = \frac{3{,}3}{4096} \times mot$
3. $tension = \frac{2^{32}}{200 \times 10^6} \frac{1}{4096} \times \frac{3{,}3}{4096} = 4{,}2 \times 10^{-6}$ V/Hz.

2 Adressage de la pile

En pratique, l'identification d'une erreur de pile est plus complexe, car il est rare d'examiner le fichier de définition de la structure mémoire de la cible ou de configuration du linker. Une façon d'identifier la source du problème vient de l'étude du code désassemblé par

`arm-none-eabi-objdump -dSt st.elf > st.list`

```
SYMBOL TABLE:
[...]
20000000 l    d  .data 00000000 .data
[...]
20000004 g     O .data 00000004 rcc_ppre2_frequency
[...]
20000008 g       .data 00000000 _ebss
[...]
20000008 g       *ABS* 00000000 end
[...]
20000000 g       .data 00000000 _data
[...]
20000008 g       .data 00000000 _edata
[...]
20000800 g       *ABS* 00000000 _stack
[...]
20000000 g     O .data 00000004 rcc_ppre1_frequency
[...]

int main()
{short tableau[1024];
 8000150: b580       push {r7, lr}
 8000152: f5ad 6d00  sub.w sp, sp, #2048 ; 0x800
 8000156: b082       sub sp, #8
 8000158: af00       add r7, sp, #0
 int k;
 char c;
```

Nous constatons dans l'entête que `gcc` a alloué (comme requis) la pile en 0x20000800, mais aussi que les données temporaires se trouvent entre 0x20000000 et 0x20000008.

En plaçant la pile à la position 2048 (0x800) de la RAM, et en exploitant un tableau de 1024 mots de 16 bits (soit 2048 octets) dans le programme principal, on se doute qu'un problème apparaîtra rapidement. Il est néanmoins intéressant de comprendre en détail dans le code la cause de ce dysfonctionneent. Nous savons que la pile sert au stockage des données, et en particulier le programme réserve les 2048 octets dès le démarrage du programme. Nous constatons par la commande `sub.w sp, sp, #2048` que le programme déplace le pointeur de pile de 2048 positions dès la seconde instruction du programme. Ainsi, le pointeur de pile se trouve au premier octet de la RAM, et dès que le processeur tente d'empiler l'adresse de retour lors de l'apppel à la première fonction `usart_send_blocking()` et se trouve en dehors de la RAM, d'où le crash. Le programme n'a ici même pas l'occasion de s'exécuter une seule fois, mais une corruption de pile peut être beaucoup plus difficile à identifier dans un programme plus complexe.

Une solution évidente est d'optimiser le programme pour utiliser moins de mémoire, mais ceci n'est pas toujours possible. Dans le cas qui nous intéresse, la configuration de la mémoire n'est pas efficace, car notre microcontrôleur possède 20 KB de RAM (`ram (rwx) : ORIGIN = 0x20000000, LENGTH = 20K`) mais positionne la pile à l'adresse 0x800=2048. Ce choix est fait pour maintenir la compatibilité avec tous les microcontrôleurs de la gamme STM32, qui par conséquent handicape les composants haut de gamme comportant plus de 2048 octets de RAM. Il faut donc adapter le script d'édition de lien pour placer la pile en fin de RAM, *i.e.* modifier la définition de pile par `PROVIDE(_stack = 0x20005000);`.

3 Afficheurs

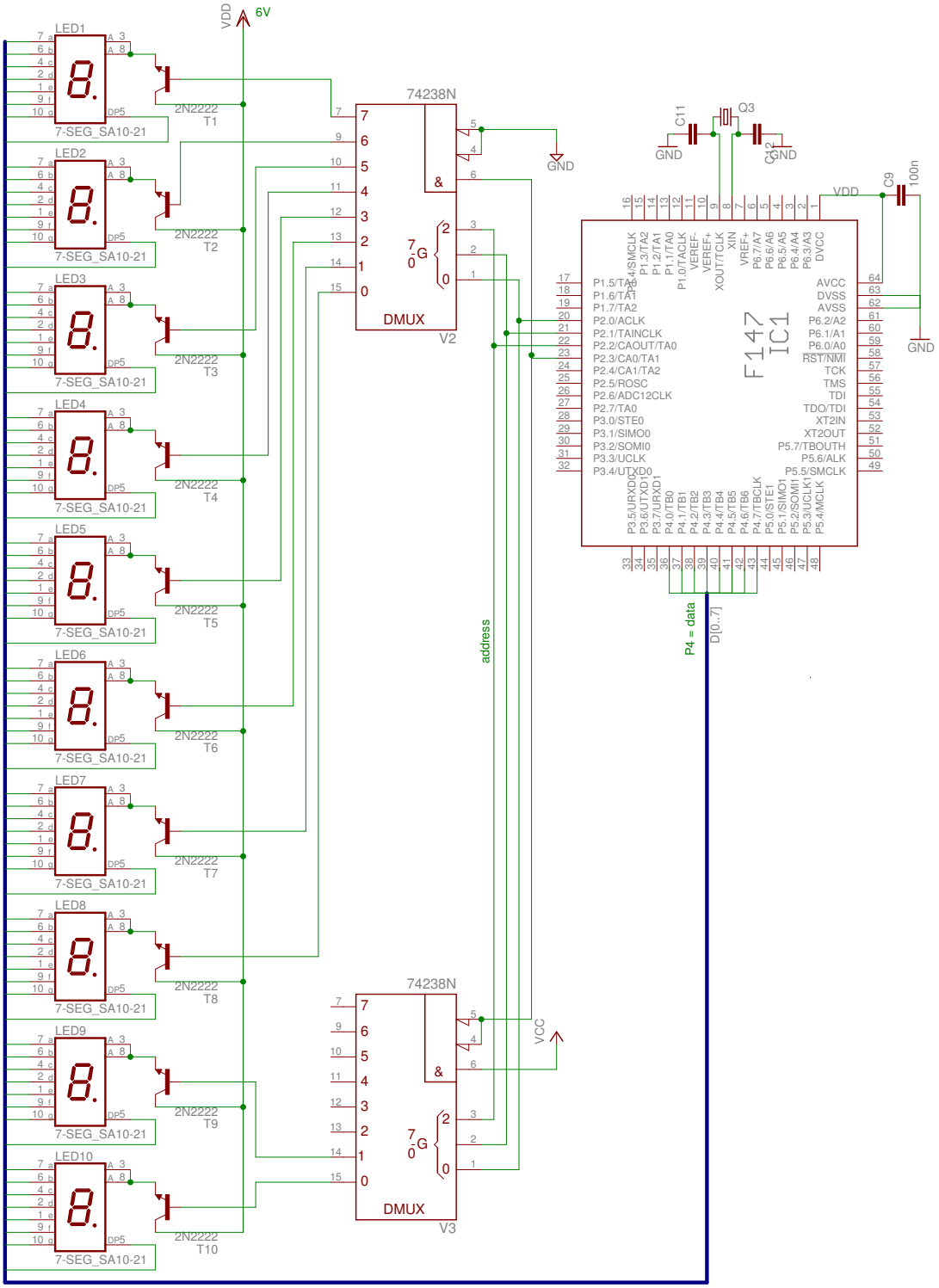
1. 50% coverage = 6000 mA.h pendant 10 heures, soit une consommation moyenne de 600 mA qui est en accord avec la consommation de 30 diodes, chacune consommant 20 mA.
2. 8×10=80 signaux, et $16^{10} = 2^{40} \simeq 10^{12}$ (puisque nous savons que $2^{10} \simeq 10^3$).
3. anode commune donc les bits à 0 du GPIO allument une diode et les bits à 1 les éteignent. Pour afficher le symbole 5, il faut allumer les segments A, G, D, F, C soit placer les broches 7, 8, 2, 9, 3 au niveau bas et 6, 1, 4 au niveau haut. Si on connecte le GPIO dans l'ordre des broches, il faut donc écrire 0b00011001=0x15. Cette valeur reste, dans ce cas particulier, la même si on connecte le GPIO dans l'ordre des segments.
4. l'objectif du masque est de sélectionner chaque quartet de 0x1234 et de l'afficher au bon emplacement :

   ```
   short t=0x1234;
   char c,k;
   for (k=0;k<4;k++) {c=(t & (short)0x000f<<(k*4)) >> k*4;affiche(c,k);}
   ```

   ou en version éclatée

   ```
   short t=0x1234;
   char c;
   c=(t & 0xf000) >> 12; affiche(c,3);
   c=(t & 0x0f00) >>  8; affiche(c,2);
   c=(t & 0x00f0) >>  4; affiche(c,1);
   c=(t & 0x000f)      ; affiche(c,0);
   ```

5. un bus de données de 8 bits et 10 signaux de commande de l'anode commune de chaque afficheur, soit 18 signaux.
6. au lieu de N signaux de commande, il ne faut plus que $\log_2(N)$ signaux qui sont convertis par le démultiplexeur en N signaux de commande, un seul étant actif à chaque instant.
7. pour 10 chiffres, il nous faut $3,3 \simeq 4$ bits de commande. Un démultiplexeur 74LS238 ne possède que 3 bits en entrée, donc il faut soit ajouter un second démultiplexeur (en prévision d'une multiplication du nombre d'afficheurs dans les versions plus évoluées que ne manquera pas de réclamer le marketing), soit simplement compléter les 3 bits du premier décodeur par un bit additionel de contrôle. On utilise donc 3 bits communs pour adresser l'afficheur voulu, et un bit de poids fort connecté aux broches d'activation G pour sélectionner quel banc d'afficheurs (0 à 7 ou 8 à 9) est activé.


VDD
6V
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
7-SEG_SA10-21
2N2222
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
74238N
DMUX
V2
V3
GND
VCC
address
C11
C12
Q3
C9
100n
IC1
F147
P4 = data
D[0..7]