Examen “électronique programmable 2”

13 mars 2017

Tous documents autorisés, pas de téléphone portable ni connexion internet pendant l'examen. Durée : 2 h.

Une horloge temps réel (*Real Time Clock* – RTC) logicielle avec une résolution de 0,1 s est formée d'un compteur qui s'incrémente et induit une interruption périodique. Le gestionnaire de cette interruption doit se charger d'incrémenter les diverses variables qui forment une horloge, à savoir les secondes, les minutes et les heures. Les secondes seront en pratique représentées en unité de 0,1 s, et s'incrémenteront donc de 0 à 600 avant d'induire l'incrément de la minute, qui quant à elle induira l'incrément de l'heure lorsqu'elle atteint 60. Une heure qui atteint 24 repasse à 0.

1. Proposer un programme qui affiche “Hello World” une fois par seconde (attente par un délai vide de 1000 ms) sur le port USB au moyen de `libvirtualserial` disponible dans
   `/home/jmfriedt/LUFA_light_EEA.tar.gz` : le compiler (comment ?) et le flasher (comment ?) pour démontrer son bon fonctionnement en observant (comment ?) la communication avec le PC.
2. Quel type de variable permet de partager des informations entre le gestionnaire d'interruption et les diverses fonctions qui forment un programme, incluant `main` ? Justifier.
3. Pourquoi cette structure de donnée ne devrait pas être utilisée en dehors de ce contexte ?
4. Partant de trois variables nommées `heure`, `minute` et `dseconde` (pour 10ème de seconde), remplir le gestionnaire d'interruption afin que ces variables représentent une horloge. On prendra soin de justifier le type de donnée (taille) qui définit chaque variable.
5. Proposer une fonction `affiche()` qui lit la variable `heure` et affiche son contenu sur le port USB au travers de la bibliothèque `libvirtualserial` en décimal.
6. Proposer une fonction `affiche_dec()` qui lit la variable `dseconde` et affiche son contenu sur le port USB au travers de la bibliothèque `libvirtualserial` en décimal, en insérant une virgule (séparateur décimal) entre la seconde et le dernier chiffre qui représente le dixième de seconde. Ainsi, si `dseconde=123`, alors la fonction afficher `12,3`.
7. Proposer une fonction `main()` qui lit une fois par seconde (attente par un délai vide de 1000 ms) le contenu de `heure`, `minute` et `dseconde` et affiche leur contenu à l'écran en exploitant les fonctions proposées dans les deux questions précédentes, par exemple sous la forme
   12 :34 :56.7
   s'il est 12h34 et 56.7 secondes. Compiler, flasher et démontrer le bon fonctionnement.
8. L'affichage d'une valeur décimale nécessite des divisions par 10, une opération complexe pour un petit microcontrôleur. Une alternative consiste à représenter les nombres en hexadécimal, avec une arithmétique quelque peu modifiée puisque 9+1=10 (et non A comme ça serait le cas en hexadédimal). Comment se nomme cette arithmétique sur les nombres hexadécimaux ?
9. Reprendre le gestionnaire d'interruption proposé en question trois pour modifier les règles d'additions ou de réinitialisation des variables pour tenir compte de ce mode de représentation. On notera en particulier, lors de l'incrément des dizaines, que 0x10-0x0A=0x10-10=6, et lors de l'incrément des centaines, que 0x100-0x0A0=0x100-160=96.
10. Maintenant que les nombres sont représentés par des valeurs hexadécimales et non décimales, leur affichage devient trivial puisque la séparation des symboles s'obtient par masquage. Modifier `affiche()` et `affiche_dec()` pour tenir compte de ce nouveau mode de représentation, et démontrer son bon fonctionnement.
11. Quelle est la fréquence exacte de déclenchement du timer ? Justifier ?
12. En déduire l'exactitude de l'horloge.
13. Proposer une solution pour pallier à un éventuel dysfonctionnement de l'horloge que la question précédente aurait détecté.
14. Sous `minicom`, démontrer un affichage de l'heure, rafraîchie toutes les secondes, qui ne change pas de ligne mais s'affiche toujours en haut à gauche du terminal.
15. la fonction

```
#include <avr/sleep.h>
void enterSleep(void)
{ set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  sleep_mode();
  sleep_disable();
}
```

    place le microcontrôleur en mode veille. Remplacer le délai dans la boucle principale par cette mise en mode veille, et constater que le programme continue à fonctionner, mais cette fois sans expliciter l'attente entre deux affichages mais avec un rythme de communication de 10 Hz.

Correction

1 Décimal

L'ISR gère l'incrément des dixièmes de seconde, minute et heure, considérées comme des valeurs décimales. Les variables manipulées par l'ISR sont des variables globales, sur lesquelles on interdit l'optimisation en les préfixant de `volatile`. L'affichage des dixièmes de secondes porte sur un `short` (valeur comprise entre 0 et 600 donc plus grande que ce que peut contenir un `char`) avec un point décimal après l'unité, tandis que l'affichage des heures et minutes porte sur un `char`.

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define F_CPU 16000000UL
#include <util/delay.h>
#include "VirtualSerial.h"

volatile char  heure=0,minute=0;
volatile short dseconde=0;

ISR (TIMER1_COMPA_vect)
{PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
 dseconde++;
 if (dseconde==600) {dseconde=0;minute++;}
 if (minute==60) {minute=0;heure++;}
 if (heure==24) heure=0;
}

void timer1_init()
{TCCR1A=0;
 TCCR1B= (1<<WGM12)|(1<<CS12)|(1<<CS10);
 TCNT1=0;
 OCR1A =1562;        // 16e6/1024/10
 TIMSK1=(1<<OCIE1A);
 sei();
}

extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
extern FILE USBSerialStream;

void affiche(char c,char *o)
{o[0]=c/10+'0'; // c\in[0-59] donc c<100;
 o[1]=c-(c/10)*10+'0';
}

void affiche_dec(short s,char *o)
{o[0]=s/100+'0'; // c\in[0-59] donc c<100;
 s=s-(s/100)*100; o[1]=s/10+'0'; o[2]='.';
 s=s-(s/10)*10;   o[3]=s+'0';
}

int main(void)
{ char buffer[40];
  SetupHardware();
  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);

  DDRB |=1<<PORTB5;   DDRE |=1<<PORTE6;
  PORTB |= 1<<PORTB5; PORTE &= ~(1<<PORTE6);
  // USBCON=0;   // pour tester clignotement timer sans VirtualSerial
  timer1_init();
  GlobalInterruptEnable();
  while(1)
  {affiche(heure,buffer);      buffer[2]=':';
   affiche(minute,&buffer[3]); buffer[5]=':';
   affiche_dec(dseconde,&buffer[6]);
   buffer[10]='\r'; buffer[11]='\n'; buffer[12]='\0';
   fputs(buffer,&USBSerialStream);
   _delay_ms(50);
   CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
   CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
   USB_USBTask();
  }
}
```

Ce programme se compile en se liant avec la bibliothèque `libvirtualserial.a` qui nécessite de renseigner, lors de la compilation, le chemin vers les fichiers d'entête associés, par exemple `-I/../VirtualSerial/ -I../lufa-LUFA-140928` si

les bibliothèques sont situées dans le répertoire précédent de l'arborescence. Le microcontrôleur est flashé par `avrdude` et le bon fonctionnement du programme validé sous `minicom -D /dev/ttyACM0`.

2 BCD

Le passage au BCD s'assure que chaque dépassement de la valeur multiple de 10 se traduit par un saut sur la puissance suivante en hexadécimal. Le calcul n'est pas trivial mais quelques essais montrent d'abord que lorsque la valeur multiple de la dizaine est atteinte, il faut incrémenter de 6 (=16-10), et idem pour les centaines (qui en hexadécimal se nomment multiples de 256). Les seules fonctions qui changent sont celles qui manipulent les variables représentant le temps, à savoir l'ISR et les affichages :

```
ISR (TIMER1_COMPA_vect)
{PORTB^=1<<PORTB5;PORTE^=1<<PORTE6;
 dseconde++;
 if ((dseconde & 0x0f)==0x0A) dseconde+=6;  // 16 - 10 = 6
 if ((dseconde & 0xf0)==0xA0) dseconde+=96; // 256-160 = 96
 if (dseconde==0x600) {dseconde=0;minute++; if ((minute & 0x0f )==0x0A) minute+=6;}
 if (minute==0x60)    {minute=0;  heure++;  if ((heure &0x0f)==0x0A) heure+=6;}
 if (heure==0x24) heure=0;
}

void affiche(char c,char *o)
{o[0]=((c>>4)&0x0f)+'0'; // c\in[0-59] donc c<100;
 o[1]=(c&0x0f)+'0';
}

void affiche_dec(short s,char *o)
{o[0]=((s>>8)&0x0f)+'0'; // c\in[0-59] donc c<100;
 o[1]=((s>>4)&0x0f)+'0'; o[2]='.';
 o[3]=(s&0x0f)+'0';
}
```

Le timer se déclenche au rythme de $16 \cdot 10^6/1024/1562 = 10,0032$ Hz, soit une erreur par rapport à la fréquence nominale de 10 Hz de 0.003 Hz ou 0,03%. Une solution pour pallier à ce dysfonctionnement consiste à moins diviser afin d'avoir une solution exacte $16 \cdot 10^6/256/10 = 6250$.

Pour afficher le message toujours au même endroit son `minicom`, on remplace la fin de chaîne `\r\n` par `\r`

Finalement, le rôle d'une horloge est souvent de réveiller du mode veille afin d'économiser de l'énergie. Dans notre cas, cela se traduit par

```
#include <avr/sleep.h>

void enterSleep(void)
{ set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  sleep_mode();
  sleep_disable();
}

int main(void)
{ [...]
  SetupHardware();
  while(1)
  {affiche(heure,buffer);      buffer[2]=':';
   affiche(minute,&buffer[3]); buffer[5]=':';
   affiche_dec(dseconde,&buffer[6]);
   buffer[10]='\r'; buffer[11]='\n'; buffer[12]='\0';
   fputs(buffer,&USBSerialStream);
   enterSleep(); // _delay_ms(100);
   CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
   CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
   USB_USBTask();
  }
}
```