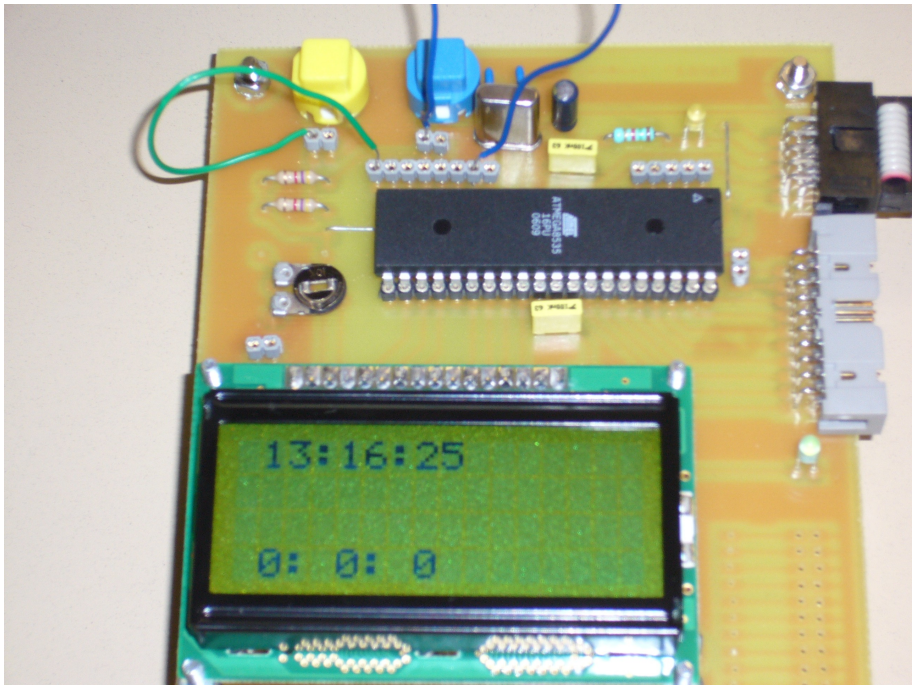
Université François Rabelais de Tours

Institut Universitaire de Technologie de Tours

Département Génie Electrique et Informatique Industrielle



Réveil

Florent Prou
Paul Lerebourg
2ème Année S2
Promotion 2005/2007

Enseignants:
M.Thierry Lequeu
Mme Sophie Laurenceau

Université François Rabelais de Tours

Institut Universitaire de Technologie de Tours

Département Génie Electrique et Informatique Industrielle

Réveil

Florent Prou
Paul Lerebourg
2ème Année S2
Promotion 2005/2007

Enseignants:
M.Thierry Lequeu
Mme Sophie Laurenceau

Sommaire

Introduction

Notre projet a pour but de créer un réveil qui permet de diminuer l'éclairage de l'heure la nuit, grâce a un microcontrôleur. L'affichage de l'heure peut empêcher une personne de dormir, c'est pourquoi nous avons choisi de réaliser ce réveil.

Dans un premier lieu, il faudra réaliser la fonction principal du réveil, l'affichage de l'heure ainsi que la possibilité de régler une alarme. Dans un second temps nous devrons ajouter à celui-ci la possibilité de connaître l'éclairage de la pièce où le réveil se trouve et d'atténuer ou non l'affichage de l'heure.

Nous présenterons dans un premier temps le projet, nous poursuivrons par le détails des configurations nécessaires, pour finir sur la partie programmation.

1. Présentation du projet

Dans le cadre de notre matière appelée Travaux et Réalisation, nous devons réaliser un projet. Notre étude portera sur un réveil dont le rétro éclairage variera selon l'intensité de la lumière qui sera présente dans son environnement.

Tout d'abord nous devons réaliser un réveil « normal » pour ensuite lui ajouter les fonctions que nous désirons. La partie configuration se fera selon ce cahier des charges.

1.1. Cahier des charges

Le réveil que nous avons choisi de réaliser, doit premièrement fonctionner normalement pour que nous ajoutions la possibilité d'atténuer le rétro-éclairage. Le fonctionnement normale d'un réveil consiste à afficher l'heure en temps réel, avec plusieurs boutons permettant le réglage de celle-ci ainsi que de l'alarme. Afin de réaliser ces fonctions nous avons choisi d'utiliser 5 boutons : réglage de l'heure, réglage de l'alarme, incrémentation des heures, incrémentation des minutes, alarme On/Off.

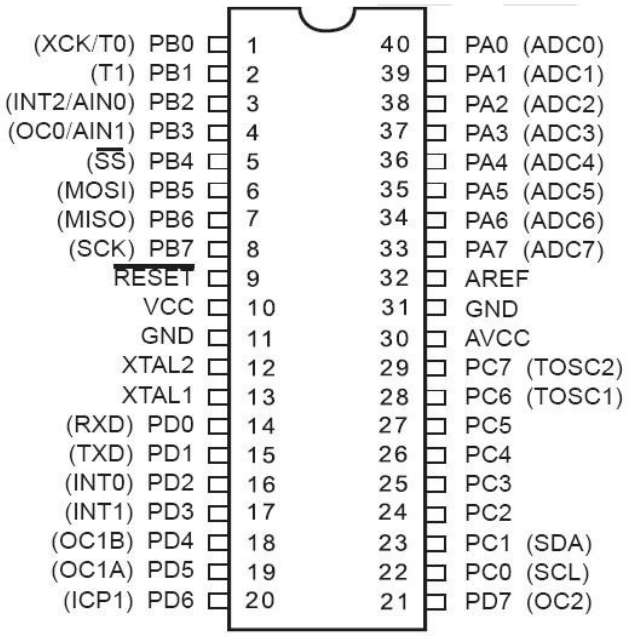
La seconde partie du projet consistera à réaliser à l'aide d'une photo résistance qui nous indiquera grâce à sa valeur ohmique l'intensité lumineuse de l'environnement du réveil le système permettant d'atténuer le rétro éclairage de l'écran LCD utilisé pour afficher l'heure.

1.2. Présentation des outils requis

Pour étudier ce projet et afin de le réaliser, nous avons utiliser le logiciel Code vision AVR (disponible à l'achat sur le site www.codevision.be/) ainsi qu'un microcontrôleur Atmega8535.

a)Le microcontrôleur et ses caractéristiques principales

Nous avons à notre disposition un microcontrôleur avec 4 ports de 8 bits configurables en entrée et sortie. On ne présentera pas les entrées analogiques mais leur configurations se trouvent en page 201 de la documentation technique de l'ATmega8535.



Le microcontrôleur fonctionne avec du 5V continue, qui est relié sur les pattes 10 et 30 et la masse sur les pattes 11 et 31 comme on peut le voir sur la figure ci-contre. Ce composant se programme en langage C, il est branché sur une carte permettant d'accueillir un écran lcd directement relié sur le portC, ainsi que de brancher une carte externe sur le portA grâce à une prise 20 broches (voir figure ci contre).

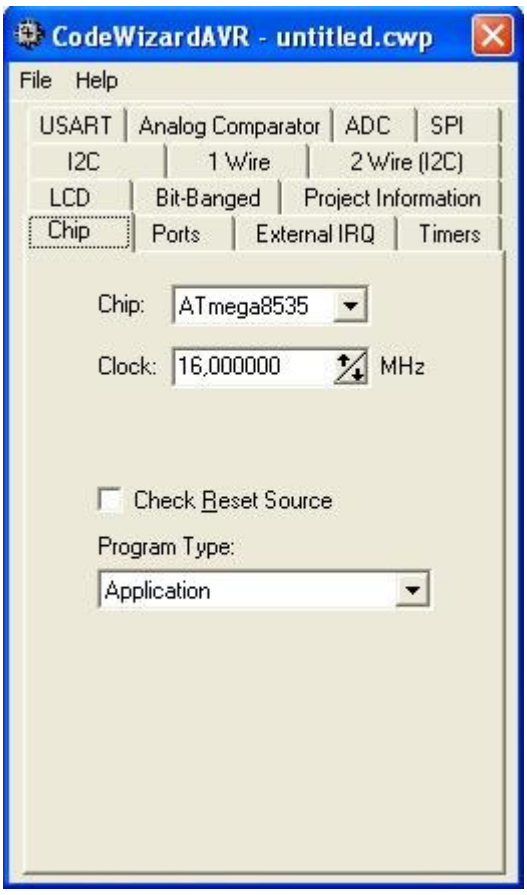
La fréquence de l'horloge de l'ATmega85 peut être de 8 ou 16 Mhz. nous utilisons un quartz de 16 Mhz. On peut utiliser 3 timers pour des interruptions interne dont la configuration sera expliquée au cours de la partie programmation. Il est aussi possible d'utiliser des interruptions externes qui ne seront pas expliqués puisque nous n'en utilisons pas.

Toutes la partie configuration de ces outils est entièrement gérées par le compilateur Code Vision AVR Studio.

b)Le logiciel de programmation

Code vision Avr est un logiciel de compilation en C, qui permet entre autre la possibilité de créer des applications autonomes : création de fichiers exécutable et d'implanter les programmes dans un microprocesseur.

L'avantage principal de ce logiciel est de permettre la configuration complète des outils que l'on veut utiliser tel que : l'utilisation d'un mcrocontrôleur Atmega8535, configuration des ports en entrée ou sortie, utilisation des timers, utilisation du convertisseur analogique numérique, affichage sur un écran lcd et bien d'autre. Toute cette partie de configuration se fait grâce à une seule fenêtre, et en mode graphique comme sur la figure suivante qui est la page principale qui s'ouvre lorsque l'on ouvre un projet. Sur cette page on choisi le microprocesseur dans lequel on va implanter le programme. Les configurations se feront grâce aux multiples onglets, qui seront présentés au cours de la partie programmation.



2. Présentation de la partie configuration

Nous utiliserons des organigrammes afin de faciliter la compréhension des programmes. Pour simplifier le programme nous avons dû le décomposer en plusieurs fonctions: *reglage_H()*, *reglage_A()*, *affichage()*, *void main()* et une fonction interruption. Nous expliquerons chaque fonctions ainsi que les moyens utiliser pour les réaliser mais pour cela nous devons expliquer comment configurer le programme.

2.1. Configuration d'une interruption

Pour que le réveil affiche l'heure, il faut qu'il contienne des variables qui s'incrémentent en temps réels tels que les secondes ensuite suivront les minutes et heures qui évolueront en fonction des secondes. Ces variables seront déclarés en tant qu'entier (int pour integer en anglais donc « entier ») par:

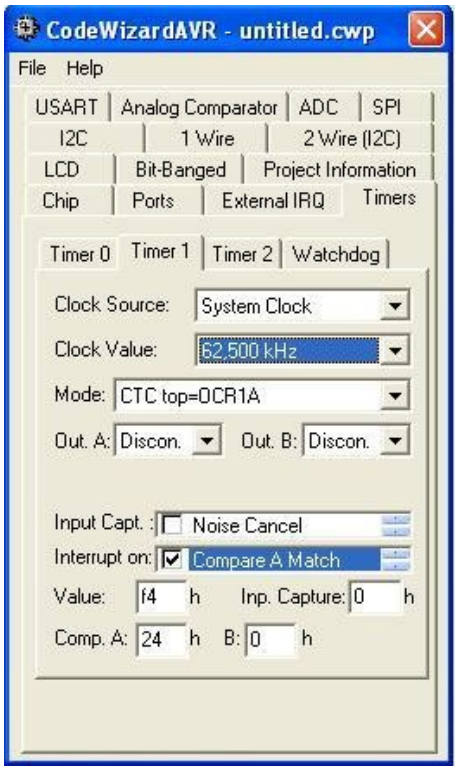
int seconde, minute , heure ;

Ce qui permettra au microcontrôleur d'allouer un espace mémoire pour ces 3 variables.

Chaque seconde, la variable doit s'incrémenter. Pour un fonctionnement optimal on utilisera une interruption interne grâce à un timer.

Le timer permet entre autre de compter de 0 à une valeur définie, avec une fréquence choisie et de se placer dans un endroit précis du programme.

A l'aide du logiciel de compilation on configure le timer.



Lors de la fenêtre de configuration initiale, on choisi parmi les onglets: « Timers » , qui propose la configuration de 3 timers.

Nous choisirons d'utiliser le timer 1. Ensuite il faut choisir la source de l'horloge du timer, interne ou externe, pour nous interne: « system clock ».

Il faut choisir la fréquence de cette horloge, parmi les valeurs 16MHz, 2MHz, 250KHz, 62,5KHz, et 15,625 KHz. Cette valeur définit la fréquence à laquelle la valeur du timer est incrémentée jusqu'à obtenir une valeur entre 0 et 65536. Le timer 1 compte sur deux octets donc 16 bits, qui font 2^{16}= 65536. Lorsque le timer arrive à cette valeur, le programme se place dans une fonction spéciale qui sera précisée plus tard.

Pour connaître le temps au bout duquel le microcontrôleur ira dans la fonction interruption, on prends l'inverse de la fréquence: la période (pour obtenir des secondes), que l'on multiplie à la valeur maximale que le timer peut compter: $2^{16.}$

Si on utilise la fréquence de 16 Mhz, la fonction interruption sera utilisée:

$$\frac{1}{16.10^6}\times 2^{16}=0{,}004096\,s$$

Toutes les 0,004096 s le microcontrôleur ira dans la fonction pour incrémenter les secondes. Ce temps étant trop petit on doit refaire ce calcul jusqu'à obtenir la valeur la plus proche de 1s.

$$\frac{1}{2.10^6}\times 2^{16}=0{,}032768\,s$$

$$\frac{1}{250.10^3}\times 2^{16}=0{,}262144\,s$$

$$\frac{1}{62500}\times 2^{16}=1{,}048576\,s$$

Avec la fréquence 62,500 Khz, on obtient une valeur supérieure à 1s.

On en déduit qu'il ne faudra pas compter jusqu'à 65536. Pour savoir jusqu'à combien il faudra compter il faut résoudre cette équation:

$$\frac{1}{62500}\times X=1\text{s}$$

$$X=62500$$

Il faudra donc que le timer compte jusqu'à 62500.

Pour fixer la valeur à laquelle le timer devra interrompre le programme, il faut coder cette valeur en héxadécimale.

$$2^{15}+2^{14}+2^{13}+2^{12}+2^{10}+2^{6}+2^{3}$$

En binaire $62500 = (1111\ \ 0100\ \ 0010\ \ 0100)_2$

En héxadécimal $(F\ \ 4\ \ 2\ \ 4)_{16}$

62500= 0xF424

Il faut donc remplir les cases « Value » et « Comp.A » avec respectivement F4 et 24, car la valeur de comparaison est contenu dans 2 variables de chacune 1 octets, dont une sera pour les bits de poids forts et l'autre pour les bits de poids faible. On retrouvera ces valeurs dans le programme final , qui sont nommées OCR1AH et OCRIAL pour High et low,bit de poids fort et bit de poids faible.

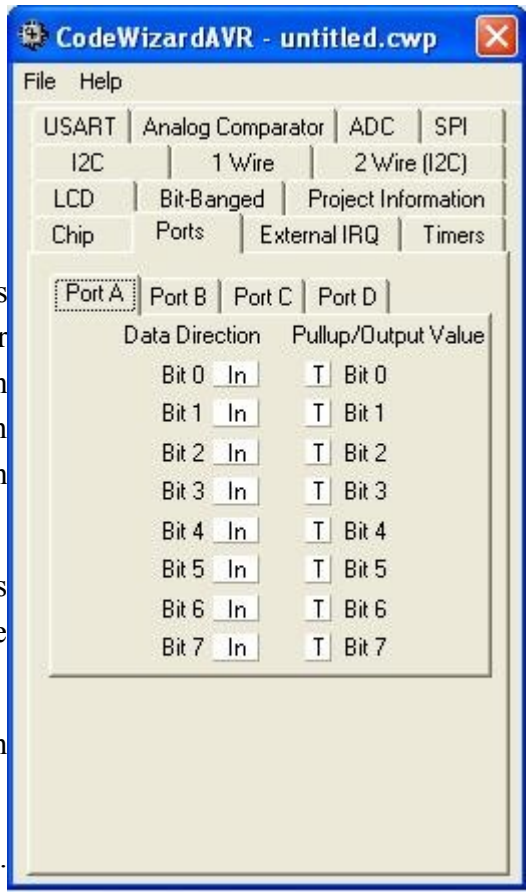
Pour que le timer s'interrompre lorsqu'il atteindra cette valeur il faut choisir dans le menu déroulant le « Mode » « CTC top=OCR1A » et cocher la case « interrupt on » « compa A match » qui permet de choisir une valeur soi-même.

En annexe se trouve les informations issues de la documentation technique du microcontrôleur page 37.

Cette partie de configuration est terminée pour le timer. Par contre il reste à configurer l'utilisation des ports.

2.2. Configuration des ports

Les 4 ports du microcontrôleur sont configurable en entrée ou en sortie, chaque port est configuré en entrée par défaut. Pour notre projet nous utiliserons le port A entrée, où sera branché les boutons, le port C pour l'écran LCD et le premier bit du port D en sortie pour le buzzer.



Dans l'onglet « Ports », nous devons choisir comment utiliser les ports du microcontrôleur en cliquant sur l'onglet correspondant au port et en cliquant sur « In » pour mettre le bit correspondant en « Out » donc en sortie et inversement pour mettre en entrée.

On retrouvera la configuration des ports dans le programme principale sous forme d'attribution tel que:

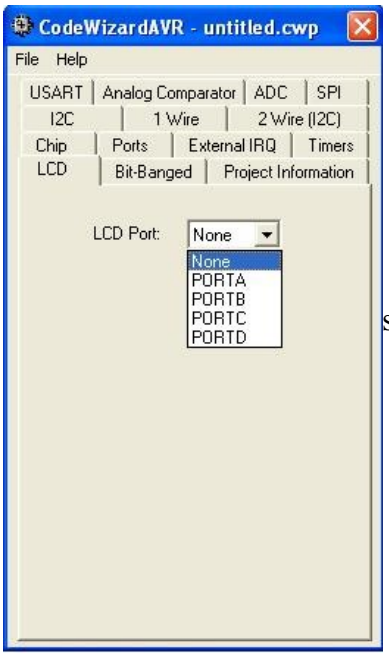
PORTA=0x00; pour déclarer le port A en entrée,

ou PORTA=0xFF; pour qu'il soit en sortie. Chaque port sera ainsi configuré et donc reconfigurable.

L'écran LCD que nous utilisons peut être configuré de la même manière.

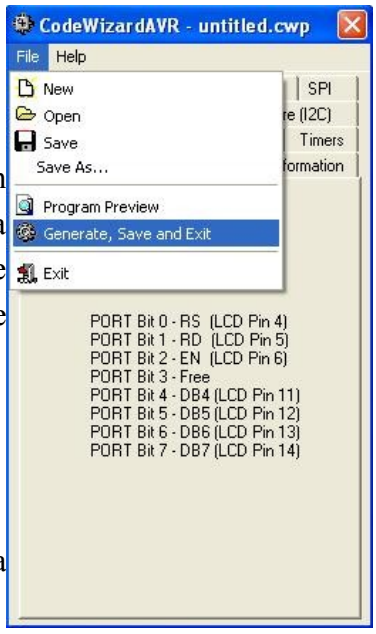
2.3. Configuration de l'écran LCD

Pour configurer l'écran LCD on se place dans l'onglet « LCD ».



On doit simplement sélectionner le port sur lequel l'écran est relié, et il est configuré.

Lorsque l'on a fait ces 3 étapes, la partie configuration est terminée. Pour ensuite commencer l'écriture du programme, on doit choisir dans le menu « File », « Generate,Save and Exit ».



Une fenêtre s'ouvre pour choisir un répertoire de sauvegarde, puis une page s'ouvre avec la partie configuration codée en C, et le programme principale *void main (void)*, ainsi qu'une fonction nommée ***interrupt*** *[TIM1_COMPA] void timer1_compa_isr(void)*.

On peut désormais passer à la programmation.

3. Programmation du réveil

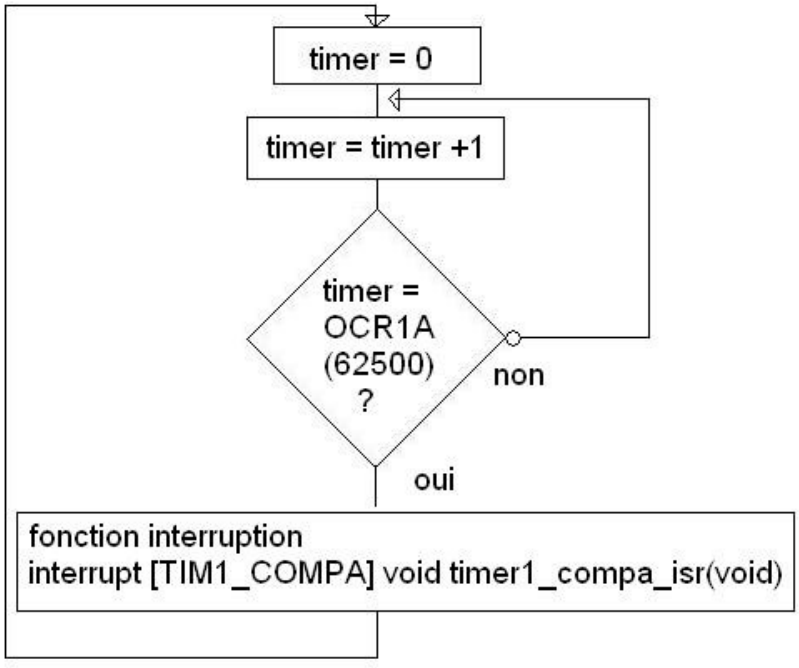
Dans cette partie nous verrons les ordinnogrammes de chaque fonctions, *reglage_H()*, *reglage_A()*, *affichage()*, *void main()* et de la fonction d'interruption., ainsi que leurs codage en C.

3.1. Fonction d'interruption

La configuration réalisée au préalable à laissée rajoutée au programme ces quelques ligne:

```
33  // Timer 1 output compare A interrupt service routine
34  interrupt [TIM1_COMPA] void timer1_compa_isr(void)
35  {
36  // Place your code here
37
38  }
```

C'est ici que le microcontrôleur ira à chaque fin de comparaison avec la valeur 62500 du timer ainsi que l'emplacement où nous programmerons l'incrémentation des variables contenant l'heure. Voyons comment le timer fonctionne.



Le timer est initialisé à 0 on rajoute 1 à la valeur du timer

si la valeur du timer est équivalente à celle mise dans 0CR1A (62500 pour nous)

le microcontrôleur éxécute la fonction interruption

sinon retour à l'étape précedente

Une fois que la fonction interruption est faite le timer peut être remis à 0 pour faire un autre cycle.

Lorsque le microcontrôleur éxecute la fonction interruption, il faut incrémenter la variable seconde et gérer à partir de celle-ci les variables minutes et heures.

lorsque le timer est égal à 62500

La variable *seconde* est incrémentée

on teste si *seconde* est égale à 60

si oui on met *seconde* à 0

et on incrémente *minute*

sinon fin de l'interruption

on teste si *minute* est égale à 60

si oui *minute* est mise à 0

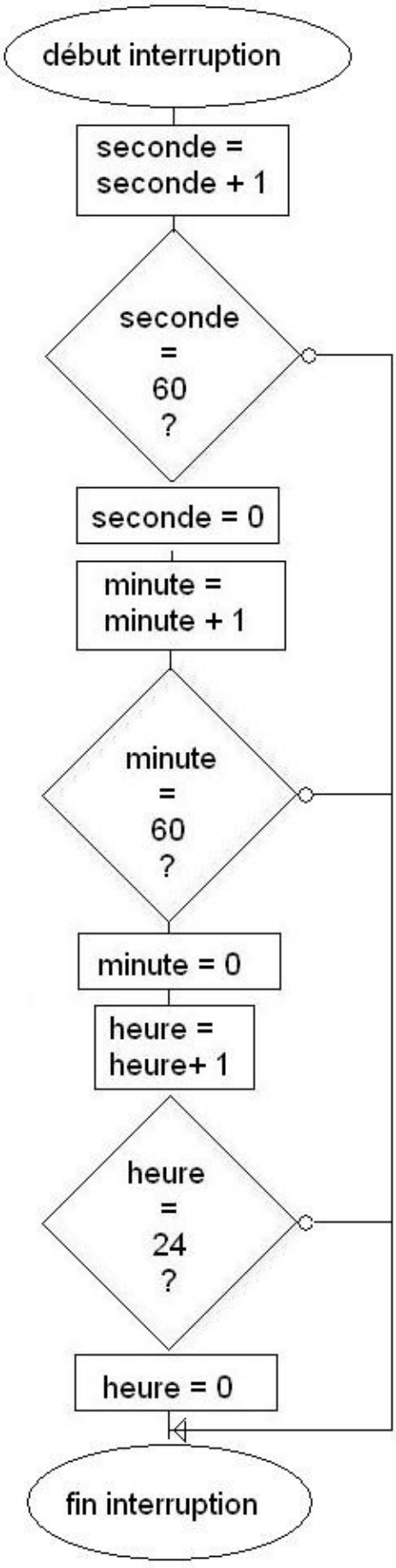
et on incrémente *heure*

sinon fin de l'interruption

on teste si *heure* est égale à 24

si oui *heure* est mise à 0

sinon fin de l'interruption



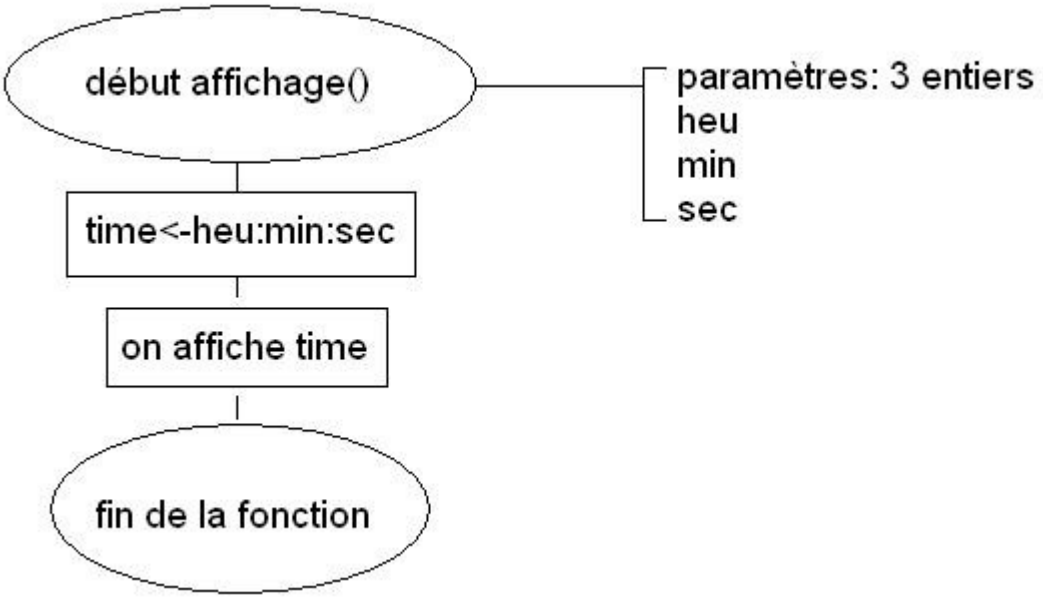
Lorsque l'on traduit l'ordinogramme précédent en langage C, on obtient :

```
46  // Timer 1 output compare A interrupt service routine
47  interrupt [TIM1_COMPA] void timer1_compa_isr(void)
48  {
49  seconde=seconde+1;
50
51          if(seconde==60)
52                  {
53                  seconde=0;
54                  minute=minute+1;
55                          if(minute ==60)
56                          {
57                          minute=0;
58                          heure=heure+1;
59
60                                  if(heure==24)
61                                  {
62                                  heure=0;
63                                  }
64
65                          }
66
67                  }
68
69          affichage(heure,minute,seconde);
70
71
72  }
```

On peut voir à la ligne 69, l'utilisation d'une fonction *affichage(heure,minute,seconde);* qui renvoit à une fonction qui permet d'afficher sur l'écran lcd, l'heure.

3.2. Fonction affichage

Pour simplifier le programme nous avons crée une fonction nommée affichage qui prends en paramètre 3 entiers, qui seront stockés dans un tableau crée en global puis affichés.



La fonction stocke les paramètres dans un tableau de caractères non signés de 20 cases. On a pris 20 cases en prévision d'afficher d'autres informations que l'heure.

Ensuite on affiche l'heure, et on quitte la fonction.

```
73  void affichage (int heu, int min, int sec)
74  {
75
76          sprintf(time,"%2d:%2d:%2d",heu,min,sec);
77          lcd_puts(time);
78
79  }
80
81
```

En C, on utilise la fonction sprintf(), qui rempli un tableau de caractères.

Le premier paramètre de cette fonction est le nom du tableau dans lequel on va stocker le seconde paramètre.

L'affichage est réalisée grâce à la fonction lcd_puts(), déclarée dans la librairie <lcd.h>, et qui affiche sur l'écran lcd, le contenu de la variable *time*.

Maintenant que l'heure est disponible il devient nécessaire de pouvoir la régler.

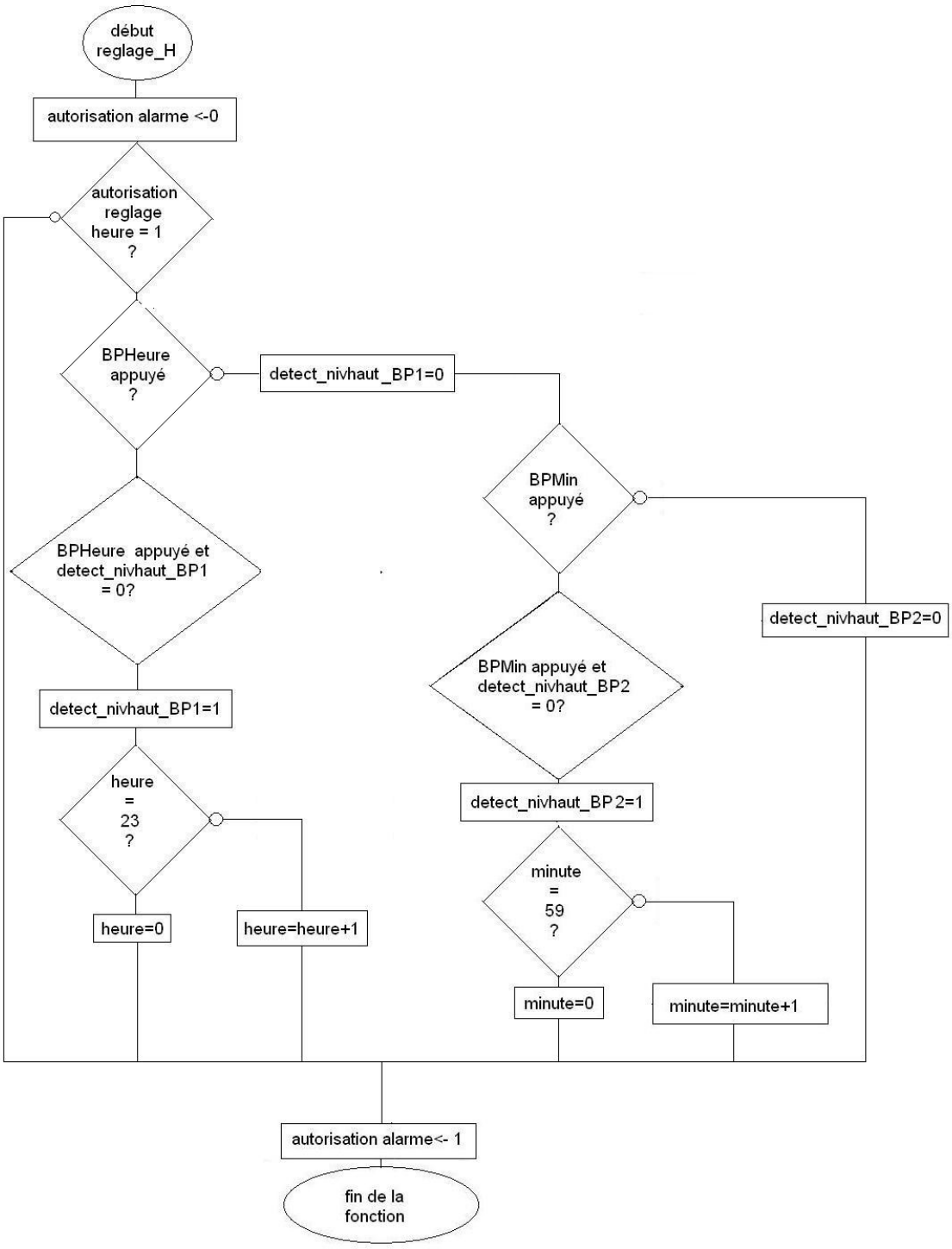
3.3. Réglage de l'heure

La fonction *reglage_H()* ne prends aucun paramètre et ne renvoie rien. Elle est exécutée lorsque l'on appuie sur le bouton *BpreglageHeure* qui est geré dans le programme principal.

Pour régler l'heure il sera nécessaire de maintenir appuyer sur ce bouton tout en appuyant sur BPHeure et BPMin qui incrémente les heures et minutes.

On souhaite ne pas pouvoir appuyer sur les boutons de réglage d'heure et d'alarme en même temps.

On souhaite également devoir appuyer à chaque fois sur les boutons BPHeure et BPMin pour incrémenter les heures, et donc de détecter les fronts montants.



On utilise 2 variables déclarées en entier, qui s'appellent *autorisation_reglage_alarme*, et *autorisation_reglage_heure*. On met à 0 la variable *autorisation_reglage_alarme* et on teste si *autorisation_reglage_heure* est à 1, On effectuera la m^me démarche dans la fonction *reglage_A()* en prenant soin d'inverser les variables.

Pour la détection de front montant on utilise une variable interne *detect_nivhautBP1* pour le bouton des heures et *detect_nivhautBP2* pour le bouton des minutes.

On teste lorsque le bouton est appuyé, s'il ne l'est pas on met à 0, *detect_nivhautBP1*, pour tester ensuite si le bouton est appuyé et en même temps, *detect_nivhautBP1* à 0: ce qui oblige le relâchement du bouton pour rentrer à nouveau dans l'incrémentation.

On traduit l'organigramme en C, ce qui donne le programme suivant:

```
void reglage_H (void)
{
int detect_nivhautBP1, detect_nivhautBP2;

autorisation_reglage_alarme=0;
if(autorisation_reglage_heure==1)
{

                if((BPHeure)==1)
                detect_nivhaut_BP1=0;

                if(((BPHeure)==0)&&(detect_nivhaut_BP1==0))
                                {
                                detect_nivhautBP1=1;
                                if(heure==23)
                                heure=0;
                                else heure =heure+1;
                                }



                if((BPMin)==1)
                detect_nivhautBP2=0;

                if(((BPMin)==0)&&(detect_nivhautBP2==0))
                        {
                        detect_nivhautBP2=1;
                        if(minute==59)
                        minute=0;
                        else minute=minute+1;
                        }


                affichage(heure,minute,seconde);

autorisation_reglage_alarme=1;
}
}
```

Lorsqu'un bouton est appuyé son niveau logique est à 0, ce qui explique les teste effectués.

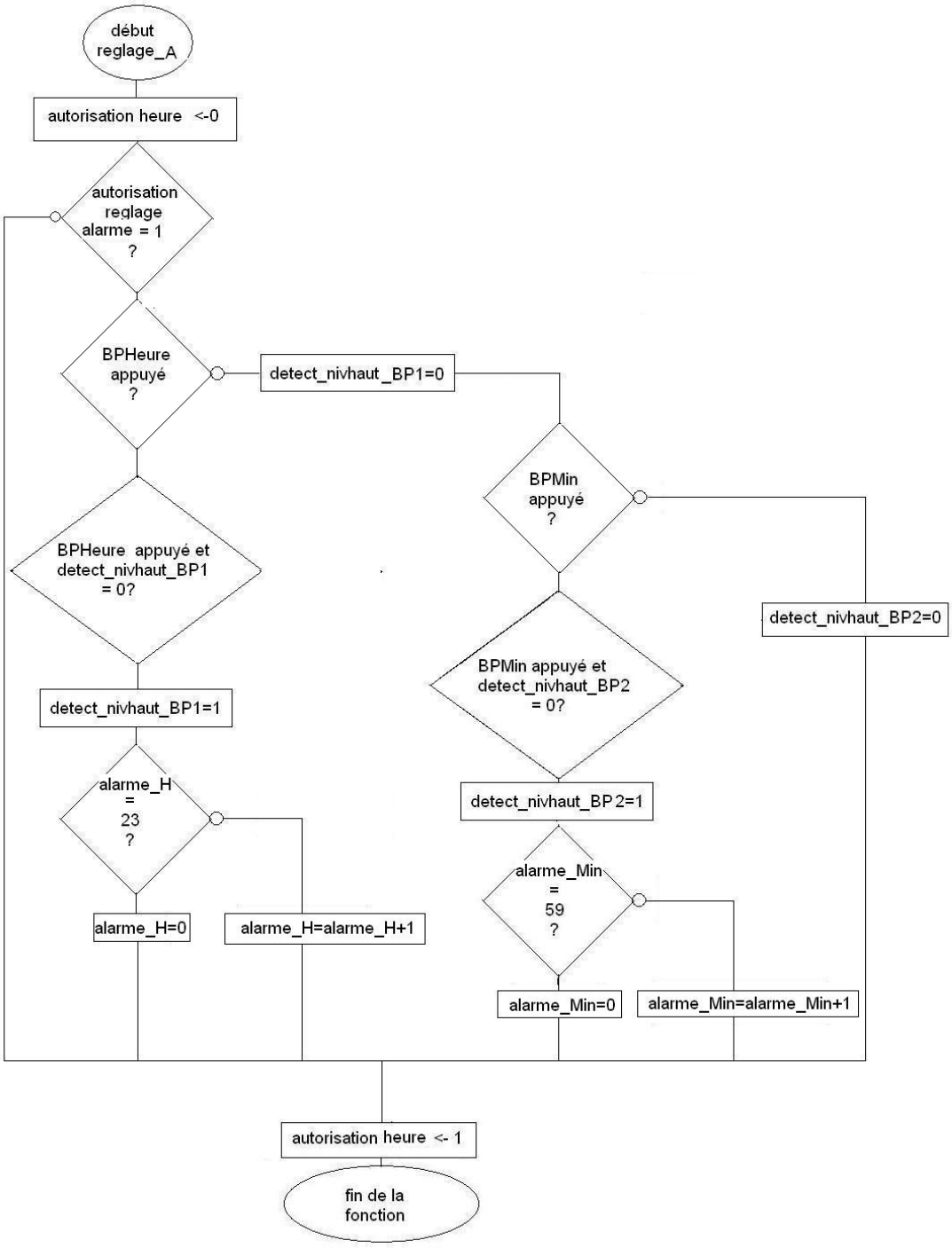
Lorsque toutes les conditions sont validées, on teste la valeur de la variable *heure* si elle vaut 23, si oui, on la met à 0, sinon on l'incrémente. On procède de la même façon pour les minutes.

Lorsque la fonction est terminée on affiche l'heure avec les modifications effectuées grâce à la fonction *affichage()*,et on remet *autorisation_reglage_alarme* à 1.

Le procédé utilisé est identique pour le réglage de l'alarme.

3.4. Réglage de l'alarme

Pour le réglage de l'alarme, nous avons utiliser 2 variables, contenant l'heure et les minutes pour l'alarme. Elles sont appelées *alarme_H* et *alarme_Min*.



Le programme correspondant est le suivant:

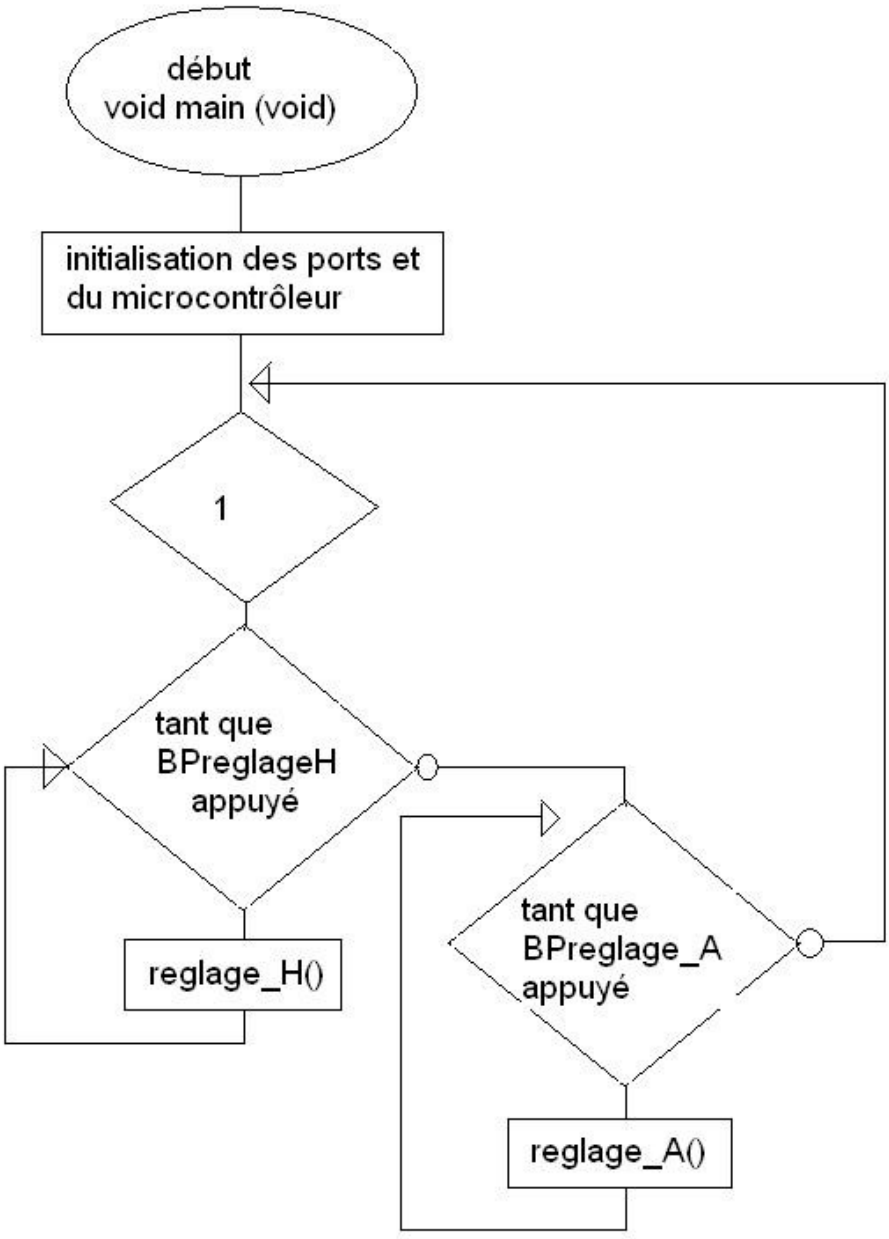
```
void reglage_A(void)
{
int detect_nivhautBP1, detect_nivhautBP2;
autorisation_reglage_heure=0;
if(autorisation_reglage_alarme==1)
{

                        if((BPHeure)==1)
                        detect_nivhaut_BP1=0;

                        if((BPHeure==0)&&(detect_nivhaut_BP1==0))
                                {
                                if(alarme_H==23)
                                alarme_H=0;
                                else alarme_H = alarme_H +1;
                                }

                        if((BPMin)==1)
                        detect_nivhautBP2=0;

                        if((BPMin==0)&&(detect_nivhautBP2==0))
                                {
                                if(alarme_Min==59)
                                alarme_Min=0;
                                else alarme_Min = alarme_Min+1;
                                }

                         affichage(alarme_H, alarme_Min, sec);

autorisation_reglage_heure=1;
}
}
```

Nous avons réalisé les fonctions permettant le réglage de l'heure, et de l'alarme ainsi que leur affichage.

Il faut désormais réaliser le programme principal.

3.5. Programme principal

Dans le programme principal nous devons programmer l'utilisation de la fonction *reglage_H()* tant que l'on appuie sur le bouton *BPreglageHeure* et la fonction *reglage_A()*, pour le bouton *BpreglageAlarme.*



Voici le programme correspondant :

```
while (1)
      {

        while(BPreglageHeure==0)
                reglage_H();

        while(BPreglageAlarme==0)
                reglage_A();

      };
}
```

En mettant en place des fils rajoutés à la main, nous avons pu tester le programme sans faire la carte externe, dont le typon est disponible sur la page 36 de l'annexe.

Nous avons vu la totalité du programme, dans ses différentes fonctions, qui se trouve en page 30 de l'annexe.

Le programme fonctionne normalement, mais le manque de temps, et la mauvaise gestion du temps que nous avions, nous ont empêchés de réaliser la totalité du cahier des charges: la gestion du rétro éclairage.

Plusieurs améliorations sont possibles, tels que l'affichage de la date, ou d'avoir plusieurs alarmes différentes.

Conclusion

Au cours de ce projet nous avons vu les difficultés de gérer un projet ainsi que d'établir un cahier des charges. La gestion du temps était aussi une difficulté puisque nous n'avons pas réalisé toutes les fonctions du cahier des charges.

Les principaux problèmes rencontrés, étaient l'utilisation, et la gestion d'interruption interne qui nous étaient complètement inconnu, et le fait de ne pas être autant guidés que lors des TP.

Le fait de ne pas être guidé par le professeur nous a permis une « simulation » de ce qui pourrait arriver durant notre stage: devoir utiliser des composants et logiciels inconnus et de gérer le projet dans sa totalité.

Annexe

Programme complet

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.7f Evaluation
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project : test
Version :
Date    : 16/03/2007
Author  : Freeware, for evaluation and non-commercial use only
Company :
Comments:


Chip type           : ATmega8535
Program type        : Application
Clock frequency     : 16,000000 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 128
*****************************************************/

#include <mega8535.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
#include<stdio.h>
//declaration des noms attirbués aux boutons
#define BPHeure         PINA.0
#define BPMin           PINA.1
#define BPreglageHeure  PINA.2
#define BPreglageAlarme PINA.3
#define AlarmeONOFF     PINA.4


//déclarration des variables
unsigned char time[20];
int heure, minute, seconde;
int alarme_H, alarme_Min;
int autorisation_reglage_alarme;
int autorisation_reglage_heure;
```

```
//déclaration des fonctions
void affichage (int heu, int min, int sec);


// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
seconde=seconde+1;

        if(seconde==60)
                {
                seconde=0;
                minute=minute+1;
                        if(minute ==60)
                        {
                        minute=0;
                        heure=heure+1;

                                if(heure==24)
                                {
                                heure=0;
                                }

                        }

                }

        affichage(heure,minute,seconde);


}
void affichage (int heu, int min, int sec)
{

      sprintf(time,"%2d:%2d:%2d",heu,min,sec);
         lcd_puts(time);

}



void reglage_H (void)
{
int detect_nivhautBP1, detect_nivhautBP2;

autorisation_reglage_alarme=0;
if(autorisation_reglage_heure==1)
{
```

```
            if((BPHeure)==1)
                  detect_nivhaut_BP1=0;

                  if(((BPHeure)==0)&&(detect_nivhaut_BP1==0))
                              {
                              detect_nivhautBP1=1;
                              if(heure==23)
                              heure=0;
                              else heure =heure+1;
                              }



                  if((BPMin)==1)
                  detect_nivhautBP2=0;

                  if(((BPMin)==0)&&(detect_nivhautBP2==0))
                        {
                        detect_nivhautBP2=1;
                        if(minute==59)
                        minute=0;
                        else minute=minute+1;
                        }


                  affichage(heure,minute,seconde);

autorisation_reglage_alarme=1;
}
}


void reglage_A(void)
{
int detect_nivhautBP1, detect_nivhautBP2;
autorisation_reglage_heure=0;
if(autorisation_reglage_alarme==1)
{

                  if((BPHeure)==1)
                        detect_nivhaut_BP1=0;

                        if((BPHeure==0)&&(detect_nivhaut_BP1==0))
                              {
                              if(alarme_H==23)
                              alarme_H=0;
                              else alarme_H = alarme_H +1;
                              }
```

```
                if((BPMin)==1)
                    detect_nivhautBP2=0;

                    if((BPMin==0)&&(detect_nivhautBP2==0))
                            {
                            if(alarme_Min==59)
                            alarme_Min=0;
                            else alarme_Min = alarme_Min+1;
                        }

                 affichage(alarme_H, alarme_Min, sec);

autorisation_reglage_heure=1;
}
}



// Declare your global variables here

void main(void)
{
//configuration des ports et initialisation de ceux-ci.
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 62,500 kHz
// Mode: CTC top=OCR1A
// OC1A output: Discon.
// OC1B output: Discon.
```

```
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x0C;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0xF4;
OCR1AL=0x24;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x10;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// initialisation de l'écran LCD
lcd_init(15);
lcd_clear();
// Global enable interrupts
#asm("sei")
```

```
while (1)
    {

     while(BPreglageHeure==0)
          reglage_H();

     while(BPreglageAlarme==0)
          reglage_A();

    };
}
```

Typon de la carte externe